SOAP API 開発者ガイド

バージョン 62.0, Winter ‘25

最終更新日: 2024/01/02

本書の英語版と翻訳版で相違がある場合は英語版を優先するものとします。

目次

SOAP API の概要

第 1 章　SOAP API について

トピック:

Salesforce では、使いやすく、強力で安全なアプリケーションプログラムインターフェースを使用した組織の情報へのプログラムによるアクセスを提供しています。SOAP API の詳細を理解するには、ソフトウェア開発、Web サービス、そして Salesforce ユーザーインターフェースについての基本的な知識が必要です。

このガイドで説明されている機能は、組織で API 機能が有効化されている場合に使用できます。この機能は、Performance Edition、Unlimited Edition、Enterprise Edition、Developer Edition ではデフォルトで有効になっています。Professional Edition を使用する組織の中には、API が有効化されている組織もあります。このガイドに記載されている機能にアクセスできない場合は、Salesforce にご連絡ください。

Salesforce では、SOAP API に加えていくつかの API を提供しています。SOAP API が使用に適しているかどうかについては、*Salesforce* ヘルプの「どの API を使用するか?」を参照してください。

ヒント: Salesforce SOAP API は、Salesforce オブジェクトを操作するために設計されています。Salesforce オブジェクトの概要と詳細な情報については、『Salesforce プラットフォームのオブジェクトリファレンス』を参照してください。

Salesforce ソリューションのカスタマイズ、インテグレーション、および拡張

Lightning Platform を使用すると、選択した言語およびプラットフォームを使用して、ご使用の Salesforce 組織を次のようにカスタマイズ、インテグレーション、および拡張できます。

- **Salesforce のカスタマイズ:** カスタム項目、カスタムリンク、カスタムオブジェクト、カスタムページレイアウト、カスタムボタン、カスタムレコードタイプ、カスタム S コントロール、カスタムタブを使用して、特定のビジネス要件を満たします。
- **Salesforce のインテグレーション:** 組織の ERP や会計システムを使用して統合します。リアルタイムの販売情報やサポート情報を会社のポータルに配信し、重要なビジネスシステムに顧客情報を入力します。
- **Salesforce の拡張:** 組織のビジネス要件を反映する新機能により、プレゼンテーション、ビジネスロジック、およびデータサービスの面で拡張します。

Lightning Platform ソリューションおよび開発者のリソースについての詳細は、「Salesforce Developers (Salesforce 開発者)」にアクセスしてください。

サポートされているエディションと必要な権限

SOAP API を使用するには、組織で Enterprise Edition、Performance Edition、Unlimited Edition、または Developer Edition を使用する必要があります。既存の Salesforce のお客様が Enterprise Edition、Unlimited Edition、または Performance Edition にアップグレードする場合は、担当者にご連絡ください。

本番組織の設定とライブデータを保護するために、すべてのソリューションおよび変更の作成およびテストには、Developer Edition、Sandbox、またはスクラッチ組織のような分離された環境を使用することをお勧めします。

Developer Edition では、Enterprise Edition で使用できるすべての機能へアクセスできますが、組織ごとのユーザー数とストレージ容量が制限されます。このリンクで、無料の Developer Edition にサインアップできます。Sandbox は、既存の組織のコピーを分離された環境に作成します。詳細は、「Sandbox: カスタマイズとテストのためのステージング環境」を参照してください。スクラッチ組織は、ソース駆動型で破棄可能なリリースであり、あらゆる設定が可能です。機能や設定の異なるさまざまな Salesforce エディションをエミュレートできます。詳細は、「スクラッチ組織」を参照してください。

サポートされている全エディションで、ユーザーが割り当てられているユーザープロファイルの「API の有効化」権限が有効になっている必要があります。この権限は、一部のプロファイルではデフォルトで有効になっています。たとえば、Developer Edition 組織にある多くのプロファイルで有効になっています。また、サポートされているエディションでは、Salesforce インテグレーションユーザーライセンスを使用して、システム間インテグレーションユーザーに組織へのアクセス権を与えながら、操作は API のみに制限することができます。詳細は、Salesforce ヘルプの「インテグレーションユーザーへの API 限定アクセス権の付与」を参照してください。

標準への準拠

SOAP API は、次の標準仕様に準拠するよう実装されています。

標準名	Web サイト
Simple Object Access Protocol (SOAP) 1.1	https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
Web Service Description Language (WSDL) 1.1	http://www.w3.org/TR/2001/NOTE-wsdl-20010315
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

開発プラットフォーム

SOAP API は、Visual Studio .NET 2005 を含む、現在のさまざまな SOAP 開発環境で動作します。本ドキュメントでは、Java および C# (.NET) の例を示しています。Java の例は WSC 20.0 (WSC) および JDK 6 (Java Platform Standard Edition Development Kit 6) に基づいています。他のバージョンの WSC は、https://github.com/forcedotcom/wsc および https://mvnrepository.com/artifact/com.force.api/force-wsc で入手できます。互換性のある開発プラットフォームの詳細やさらなるサンプルコードについては、「developer.salesforce.com」にアクセスしてください。

メモ: 開発プラットフォームは、SOAP の実装によって異なります。特定の開発プラットフォームにおける実装の相違点により、API の一部またはすべての機能にアクセスできないことがあります。.NET 開発に Visual Studio を使用している場合、Visual Studio 2003 以降の使用をお勧めします。

SOAP API サポートポリシー

クライアントアプリケーションでは、より豊富な機能と優れた効率性の利点を十分に生かすよう、最新バージョンの Lightning Platform WSDL ファイルを使用することをお勧めします。組織の最新の WSDL に移動するには、[設定] から、[クイック検索] ボックスに「*API*」と入力し、**[API]** を選択します。新しいバージョンがリリースされた場合は、クイックスタートで次の手順を実行してご使用の WSDL を更新してください。

- WSDL ファイルを再生成する (「ステップ 2: Web サービス WSDL を生成または取得する」)
- WSDL ファイルをご使用の環境にインポートする (「ステップ 3: 開発プラットフォームに WSDL ファイルをインポートする」)

後方互換性

Salesforce は、Lightning プラットフォームを使用する場合の後方互換性を容易に維持できるよう努めています。

新しい Salesforce リリースは、次の 2 つのコンポーネントで構成されています。

- Salesforce システムにある新しいリリースのプラットフォームソフトウェア
- 新しいバージョンの SOAP API

たとえば、Winter '07 リリースには SOAP API バージョン 9.0 が、Summer '07 リリースには SOAP API バージョン 10.0 が含まれていました。

プラットフォームソフトウェアのリリース全体で、各 SOAP API バージョンのサポートを維持しています。指定した SOAP API バージョンで機能するよう作成されたアプリケーションが、その後のプラットフォームソフト

ウェアのリリースでも同じバージョンの SOAP API では継続して機能するという点では、SOAP API には後方互換性があります。

Salesforce では、あるバージョンの SOAP API に対応して作成されたアプリケーションがその後の SOAP API バージョンで機能することは保証されません。SOAP API の拡張は継続して行われているため、メソッド署名およびデータ表示の変更が必要になる場合がよくあります。ただし、変更によりアプリケーションを新しい SOAP API バージョンに移行する必要がある場合、バージョン間の SOAP API の一貫性を最小限に保持しています。

Winter '07 リリースに付属する SOAP API バージョン 9.0 を使用して作成されたアプリケーションは、Summer '07 リリースの SOAP API バージョン 9.0、また今後のリリースにも引き続き対応します。ただし、同じアプリケーションが SOAP API バージョン 10 で機能するようにするために、アプリケーションを変更する必要がある場合があります。

WSDL の選択

API アクセスの WSDL ファイルを取得できる Lightning Platform Web サービスは、次の 2 つです。

- **Lightning Platform Enterprise WSDL** — この API は、組織のクライアントアプリケーションを開発する多くのエンタープライズユーザー向けのものです。Enterprise WSDL ファイルは、組織データを強力に定型化して表示されます。開発環境にスキーマ、データ型、項目に関する情報を提供し、開発環境と Lightning Platform Web サービスとのより緊密なインテグレーションを実現できます。この WSDL は、組織の Salesforce 構成でカスタム項目またはカスタムオブジェクトが追加、名前変更、または削除された場合に変更されます。Enterprise WSDL をダウンロードし、管理パッケージを組織にインストールする場合、生成された WSDL に追加するインストールパッケージのバージョンを選択するという、追加のステップを実行する必要があります。

  Enterprise WSDL を生成する場合は、次の点に注意してください。

  - 組織の情報に対して新しいカスタム項目またはカスタムオブジェクトが追加、名前変更または削除された場合、WSDL ファイルを再生成して項目やオブジェクトにアクセスする必要があります。
  - 生成された WSDL には、選択されたバージョンの各インストールパッケージで使用できるものなど、組織内のオブジェクトや項目が含まれています。項目またはオブジェクトが今後のパッケージバージョンに追加される場合、API インテグレーションのオブジェクトまたは項目と連動するよう、そのパッケージバージョンで Enterprise WSDL を生成する必要があります。

- **Lightning Platform Partner WSDL** — この API は、複数の組織のクライアントアプリケーションを開発する Salesforce パートナー向けのものです。Salesforce オブジェクトモデルはあまり強く型付けされていないため、この Partner WSDL を使用して、組織内のデータにアクセスすることができます。

関連リソース

Salesforce 開発者 Web サイトでは、開発者ツールキット、サンプルコード、サンプル SOAP メッセージ、コミュニティベースのサポート、およびその他のリソースの完全パッケージを提供して、開発プロジェクトを支援します。詳細は、developer.salesforce.com を参照してください。developer.salesforce.com/signup からは、Developer Edition のアカウントを無料で取得できます。

以下の Web サイトにアクセスすると、Salesforce アプリケーションの詳細情報を入手できます。

- Salesforce では、Salesforce アプリケーションの詳細情報が提供されています。

- Salesforce AppExchange では、Salesforce 向けに作成されたアプリケーションにアクセスできます。
- Trailblazer Community では、Salesforce のお客様の成功を実現するサービスが提供されています。

第 2 章　SOAP API リリースノート

SOAP API の最新の更新や変更点については、Salesforce のリリースノートを参照してください。

SOAP API を含む Salesforce Platform に影響する更新や変更については、API のリリースノートを参照してください。

SOAP API に新しい機能や、変更された機能がある場合には、それらの詳細なリリースノートへのリンクをここに示します。特にない場合には、プラットフォームとすべての API に影響を与える最近の変更について、API リリースノートを参照してください。

第 3 章　クイックスタート: SOAP API

トピック:

ここでは、ご使用の開発環境でサンプルアプリケーションを作成する手順について説明します。

メモ: インテグレーションまたはその他のクライアントアプリケーションを作成する前に、次のことを実行してください。

- 製品マニュアルに従って、開発プラットフォームをインストールする。
- このクイックスタートを開始する前に、すべての手順に目を通す。用語およびコンセプトについて理解するために、このマニュアルの残りの部分を確認する必要もあります。

ステップ 1: Salesforce Developer Edition にサインアップする

Salesforce Developer Edition を使用して、サンプルデータに対する API コードの開発、ステージング、およびテストを行います。

アプリケーションを開発する際に独立した組織を使用すると、テスト時にライブデータが保護されます。この推奨事項は、特に、(データをただ参照するだけのアプリケーションに対し)データを挿入、更新または削除するアプリケーションの場合に該当します。コードをテストしたら、API アクセスを使用してエディションに実装します。

Developer Edition 組織を作成するには、`developer.salesforce.com/signup` にアクセスし、Developer Edition 組織のサインアップの説明に従ってください。

Developer Edition 組織をすでに所有している場合は、ユーザープロファイルに「API の有効化」権限があることを確認します。この権限はデフォルトで有効になっていますが、管理者によって変更されている場合があります。詳細は、Salesforce ヘルプを参照してください。

ステップ 2: Web サービス WSDL を生成または取得する

Lightning Platform Web サービスにアクセスするには、Web Service Description Language (WSDL) ファイルが必要です。WSDL ファイルは、使用できる Web サービスを定義します。

開発プラットフォームではこの WSDL を使用して API を生成し、WSDL が定義する Lightning Platform Web サービスにアクセスします。組織の Salesforce 管理者から WSDL ファイルを取得することも、WSDL ダウンロードページへのアクセス権限がある場合は Salesforce ユーザーインターフェースで自分で生成することもできます。組織の最新の WSDL に移動するには、[設定] から、[クイック検索] ボックスに「*API*」と入力し、**[API]** を選択します。

WSDL の詳細は、`http://www.w3.org/TR/wsdl` を参照してください。

組織の WSDL ファイルの生成

「すべてのデータの編集」権限を持つユーザーなら誰でも Web Services Description Language (WSDL) ファイルをダウンロードし、API を使用して Salesforce をインテグレーションおよび拡張できます (システム管理者プロファイルにこの権限が与えられます)。

WSDL ファイルは、ダウンロードする WSDL ファイル (Enterprise または Partner) の種類に基づいて、動的に生成されます。生成された WSDL は組織の API アクセスに使用できるすべての API コール、オブジェクト (標準オブジェクトおよびカスタムオブジェクト)、および項目を定義します。

組織の WSDL ファイルを生成するには、次の手順を行います。

1. Enterprise Edition、Unlimited Edition、Performance Edition、または Developer Edition のいずれかの Salesforce のアカウントにログインします。システム管理者または「すべてのデータの編集」権限を持つユーザーとしてログインします。既知の IP アドレスからログインされていることが確認されます。詳細については、「セキュリティと API」を参照してください。
2. [設定] から、[クイック検索] ボックスに「*API*」と入力し、**[API]** を選択して WSDL ダウンロードページを表示します。

3. 適切な WSDL をダウンロードします: 「API WSDL とクライアント証明書のダウンロード」を参照してください。

ステップ 3: 開発プラットフォームに WSDL ファイルをインポートする

WSDL ファイルを作成または取得したら、開発環境内のクライアント Web サービスアプリケーション構築に必要なオブジェクトを生成できるよう、WSDL ファイルを開発プラットフォームにインポートします。このセクションでは、WSC と Microsoft Visual Studio のサンプルについて説明します。その他の開発環境の指示については、プラットフォームの製品マニュアルを参照してください。

メモ: WSDL ファイルをインポートするプロセスは、Enterprise WSDL ファイルおよび Partner WSDL ファイルで同じです。

Java 環境での使用方法 (WSC)

Java 環境は、サーバー側オブジェクトのプロキシとして機能する Java オブジェクトを使用して、API にアクセスします。API を使用する前に、まず組織の WSDL ファイルからこれらのオブジェクトを生成する必要があります。

SOAP クライアントには、このプロセスで使用する独自のツールがあります。WSC では、`wsdlc` ユーティリティを使用します。

メモ: `wsdlc` を実行する前に、システムに WSC JAR ファイルがインストール済みであり、クラスパスで参照されている必要があります。

`wsdlc` の基本構文は、次のとおりです。

```
java -classpath pathToJAR/force-wsc-XX.X.X-uber.jar com.sforce.ws.tools.wsdlc
pathToWsdl/WsdlFilename pathToOutputJar/OutputJarFilename
```

このコマンドでは、指定された WSDL ファイルに基づいて jar 出力ファイルを生成します。jar 出力ファイルの作成後、Java プログラムで WSC .jar ファイル (例、`force-wsc-57.0.0-uber.jar`) と共に参照し、クライアントアプリケーションを作成します。

Microsoft Visual Studio での使用方法

Visual Studio 言語は、サーバー側オブジェクトのプロキシとして機能するオブジェクトを使用して API にアクセスします。API を使用する前に、まず組織の WSDL ファイルからこれらのオブジェクトを生成する必要があります。

サーバー側オブジェクトのプロキシクラスを取得したら、非文字列項目に値を設定しているかどうかを指定します。詳細は、「.NET、非文字列項目、および Enterprise WSDL」を参照してください。

Visual Studio には、WSDL ファイルをインポートして XML Web サービスクライアントを生成する 2 つの方法があります。IDE ベースの方法と、コマンドラインを使用する方法です。このステップごとの説明では、IDE を使用して WSDL ファイルをインポートする方法を説明します。

メモ: 始める前に、まずアプリケーションを作成するか、Visual Studio で既存のアプリケーションを開きます。また、「組織の WSDL ファイルの生成」で説明されているように WSDL ファイルを生成する必要があります。

XML Web サービスクライアントは、XML Web サービスを参照して使用するコンポーネントまたはアプリケーションです。クライアントベースのアプリケーションである必要はありません。実際、多くの場合、ご使用の XML Web サービスクライアントは、Web フォームなどのその他の Web アプリケーションであったり、その他の XML Web サービスであったりします。マネージコードで XML Web サービスにアクセスする場合、プロキシクラスと .NET Framework がすべてのインフラストラクチャコードを処理します。

マネージコードから XML Web サービスにアクセスするには、次の手順を行います。

1. プロジェクト名 `Walkthrough` を編集します (後述のサンプルの `using` ディレクティブを *`your_project_name.web_reference_name`* に変更します)。Web 参照を、アクセスする XML Web サービスのプロジェクトに追加します。Web 参照は、XML Web サービスの公開されたメソッドのプロキシとして機能するメソッドでプロキシクラスを作成します。
2. Web 参照の名前空間を追加します。
3. プロキシクラスのインスタンスを作成し、その他のクラスのメソッドと同様にそのクラスのメソッドにアクセスします。

Visual Studio のバージョンと推奨される開発者環境に応じて、.NET 2.0 スタイルの Web 参照か、.NET 3.0 スタイルのサービス参照のいずれかを追加できます。.NET 3.0 スタイルの参照では、SforceService ではなく SoapClient のようなサービスを使用します。

Web 参照を追加するには、次の手順を行います。

メモ: これらのステップは使用している Visual Studio のバージョンによって異なります。詳細は、Visual Studio マニュアルの *「Adding and Removing Web References (Web 参照の追加と削除)」* を参照してください。

1. [プロジェクト] メニューで [サービス参照を追加]、[詳細]、[Web 参照を追加] の順に選択します。
2. [Web 参照を追加] ダイアログボックスの URL ボックスに、次のように、アクセスする XML Web サービスについてのサービス説明を取得する URL を入力します。

   ```
   C:\WSDLFiles\enterprise.wsdl
   ```

3. XML Web サービスの情報を取得するには、[Go] をクリックします。
4. Web 参照名のボックスで、Web 参照に使用する名前である、`sforce` への Web 参照の名前を変更します。
5. 対象 XML Web サービスへの Web 参照を追加するには、[参照を追加] をクリックします。
6. Visual Studio は、サービスの説明を取得し、プロキシクラスを生成して、アプリケーションと XML Web サービス間を接続します。

メモ: Visual Basic .Net 1.1 と Enterprise WSDL を使用する場合、生成された Web サービスを変更して、Visual Studio のクライアント生成ユーティリティのバグを解決します。API は、名前が Visual Basic キーワードと競合する 2 つのオブジェクト (Case と Event) を公開します。これらのオブジェクトを表すクラスが作成されると、Visual Studio はクラス名を大かっこで囲みます (`[Case]` と `[Event]`)。これは、キーワードを再利用するメソッドです。

ただし、`SObject` クラスの定義では、`SObject` 定義に含まれる `System.Xml.Serialization.XmlIncludeAttribute` のクラス参照で Case と Event がラップされません。Visual Studio のこの問題を回避するには、次のように Case と Event で `XmlIncludeAttribute` 設定を編集する必

要があります。この編集は、Enterprise WSDL の Enterprise バージョンを使用する場合にのみ必要で、C# では必要ありません。

```
    System.Xml.Serialization.XmlIncludeAttribute(GetType([Event])), _

    System.Xml.Serialization.XmlIncludeAttribute(GetType([Case])), _
```

ステップ 4: サンプルコードを説明する

WSDL ファイルをインポートすると、API を使用するクライアントアプリケーションの構築を開始できます。

次のサンプルを使用して、基本的なクライアントアプリケーションを作成します。サンプルに埋め込まれたコメントは、コードの各セクションを説明します。

Java サンプルコード

このセクションでは、WSC SOAP クライアントを使用する Java クライアントアプリケーションのサンプルについて説明します。このサンプルアプリケーションでは、ログインサーバーにログインするために必要なステップを示し、いくつかの API コールを呼び出し、引き続いて処理する方法を説明します。

このサンプルを実行するには、プログラムの引数として認証エンドポイント URL を渡す必要があります。この URL は WSDL ファイルから取得できます。このサンプルアプリケーションでは、次の主要なタスクを実行します。

1. Salesforce のユーザー名とパスワードの入力画面を表示します。
2. `login()` をコールして、シングルログインサーバーにログインします。ログインが成功すると、ユーザー情報を取得し、セッション情報と共にコンソールに書き込みます。
3. `describeGlobal()` をコールし、組織のデータで使用できるすべてのオブジェクトの一覧を取得します。`describeGlobal` メソッドは、ログインユーザーが使用できるオブジェクトを指定します。API コールから返されるデータは頻繁に変更されることはないため、この API コールは、セッションごとに何度も呼び出すべきではありません。DescribeGlobalResult はコンソールにエコーとして返されます。
4. `describeSObjects()` をコールして、指定されたオブジェクトのメタデータ (項目リストとオブジェクトプロパティ) を取得します。`describeSObject` メソッドは、ユーザーが使用できる各オブジェクトに取得できるメタデータ情報の種類について説明しています。サンプルクライアントアプリケーションはユーザーが指定するオブジェクトに `describeSObjects()` コールを実行し、返されたメタデータ情報をコンソールにエコーとして返します。オブジェクトメタデータ情報には、権限、データ型、選択リスト項目の長さと使用できる値、`referenceTo` 項目のデータ型が記載されています。
5. 単純なクエリ文字列 (`"SELECT FirstName, LastName FROM Contact"`) を渡し、`query()` を呼び出し、返される QueryResult を反復処理します。
6. `logout()` をコールして、ユーザーをログアウトします。

次のサンプルコードでは try/catch ブロックを使用して、API コールで返される可能性のある例外を処理します。

```
package com.example.samples;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
```

```
import java.io.InputStreamReader;
import java.io.IOException;
import com.sforce.soap.enterprise.DeleteResult;
import com.sforce.soap.enterprise.DescribeGlobalResult;
import com.sforce.soap.enterprise.DescribeGlobalSObjectResult;
import com.sforce.soap.enterprise.DescribeSObjectResult;
import com.sforce.soap.enterprise.EnterpriseConnection;
import com.sforce.soap.enterprise.Error;
import com.sforce.soap.enterprise.Field;
import com.sforce.soap.enterprise.FieldType;
import com.sforce.soap.enterprise.GetUserInfoResult;
import com.sforce.soap.enterprise.LoginResult;
import com.sforce.soap.enterprise.PicklistEntry;
import com.sforce.soap.enterprise.QueryResult;
import com.sforce.soap.enterprise.SaveResult;
import com.sforce.soap.enterprise.sobject.Account;
import com.sforce.soap.enterprise.sobject.Contact;
import com.sforce.soap.enterprise.sobject.SObject;
import com.sforce.ws.ConnectorConfig;
import com.sforce.ws.ConnectionException;

public class QuickstartApiSample {

   private static BufferedReader reader = new BufferedReader(
         new InputStreamReader(System.in));

   EnterpriseConnection connection;
   String authEndPoint = "";

   public static void main(String[] args) {
      if (args.length < 1) {
         System.out.println("Usage: com.example.samples."
               + "QuickstartApiSamples <AuthEndPoint>");

         System.exit(-1);
      }

      QuickstartApiSample sample = new QuickstartApiSample(args[0]);
      sample.run();
   }

   public void run() {
      // Make a login call
      if (login()) {
         // Do a describe global
         describeGlobalSample();

         // Describe an object
         describeSObjectsSample();

         // Retrieve some data using a query
         querySample();

         // Log out
```

```
            logout();
        }
    }

    // Constructor
    public QuickstartApiSample(String authEndPoint) {
        this.authEndPoint = authEndPoint;
    }

    private String getUserInput(String prompt) {
        String result = "";
        try {
            System.out.print(prompt);
            result = reader.readLine();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }

        return result;
    }

    private boolean login() {
        boolean success = false;
        String username = getUserInput("Enter username: ");
        String password = getUserInput("Enter password: ");

        try {
            ConnectorConfig config = new ConnectorConfig();
            config.setUsername(username);
            config.setPassword(password);

            System.out.println("AuthEndPoint: " + authEndPoint);
            config.setAuthEndpoint(authEndPoint);

            connection = new EnterpriseConnection(config);
            printUserInfo(config);

            success = true;
        } catch (ConnectionException ce) {
            ce.printStackTrace();
        }

        return success;
    }

    private void printUserInfo(ConnectorConfig config) {
        try {
            GetUserInfoResult userInfo = connection.getUserInfo();

            System.out.println("\nLogging in ...\n");
            System.out.println("UserID: " + userInfo.getUserId());
            System.out.println("User Full Name: " + userInfo.getUserFullName());
            System.out.println("User Email: " + userInfo.getUserEmail());
            System.out.println();
```

```
        System.out.println("SessionID: " + config.getSessionId());
        System.out.println("Auth End Point: " + config.getAuthEndpoint());
        System.out
              .println("Service End Point: " + config.getServiceEndpoint());
        System.out.println();
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

private void logout() {
    try {
        connection.logout();
        System.out.println("Logged out.");
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

/**
 * To determine the objects that are available to the logged-in user, the
 * sample client application executes a describeGlobal call, which returns
 * all of the objects that are visible to the logged-in user. This call
 * should not be made more than once per session, as the data returned from
 * the call likely does not change frequently. The DescribeGlobalResult is
 * simply echoed to the console.
 */
private void describeGlobalSample() {
    try {
        // describeGlobal() returns an array of object results that
        // includes the object names that are available to the logged-in user.
        DescribeGlobalResult dgr = connection.describeGlobal();

        System.out.println("\nDescribe Global Results:\n");
        // Loop through the array echoing the object names to the console
        for (int i = 0; i < dgr.getSobjects().length; i++) {
            System.out.println(dgr.getSobjects()[i].getName());
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}

/**
 * The following method illustrates the type of metadata information that can
 * be obtained for each object available to the user. The sample client
 * application executes a describeSObject call on a given object and then
 * echoes the returned metadata information to the console. Object metadata
 * information includes permissions, field types and length and available
 * values for picklist fields and types for referenceTo fields.
 */
private void describeSObjectsSample() {
    String objectToDescribe = getUserInput("\nType the name of the object to "
          + "describe (try Account): ");
```

```
try {
   // Call describeSObjects() passing in an array with one object type
   // name
   DescribeSObjectResult[] dsrArray = connection
         .describeSObjects(new String[] { objectToDescribe });

   // Since we described only one sObject, we should have only
   // one element in the DescribeSObjectResult array.
   DescribeSObjectResult dsr = dsrArray[0];

   // First, get some object properties
   System.out.println("\n\nObject Name: " + dsr.getName());

   if (dsr.getCustom())
      System.out.println("Custom Object");
   if (dsr.getLabel() != null)
      System.out.println("Label: " + dsr.getLabel());

   // Get the permissions on the object

   if (dsr.getCreateable())
      System.out.println("Createable");
   if (dsr.getDeletable())
      System.out.println("Deleteable");
   if (dsr.getQueryable())
      System.out.println("Queryable");
   if (dsr.getReplicateable())
      System.out.println("Replicateable");
   if (dsr.getRetrieveable())
      System.out.println("Retrieveable");
   if (dsr.getSearchable())
      System.out.println("Searchable");
   if (dsr.getUndeletable())
      System.out.println("Undeleteable");
   if (dsr.getUpdateable())
      System.out.println("Updateable");

   System.out.println("Number of fields: " + dsr.getFields().length);

   // Now, retrieve metadata for each field
   for (int i = 0; i < dsr.getFields().length; i++) {
      // Get the field
      Field field = dsr.getFields()[i];

      // Write some field properties
      System.out.println("Field name: " + field.getName());
      System.out.println("\tField Label: " + field.getLabel());

      // This next property indicates that this
      // field is searched when using
      // the name search group in SOSL
      if (field.getNameField())
         System.out.println("\tThis is a name field.");
```

```
            if (field.getRestrictedPicklist())
               System.out.println("This is a RESTRICTED picklist field.");

            System.out.println("\tType is: " + field.getType());

            if (field.getLength() > 0)
               System.out.println("\tLength: " + field.getLength());

            if (field.getScale() > 0)
               System.out.println("\tScale: " + field.getScale());

            if (field.getPrecision() > 0)
               System.out.println("\tPrecision: " + field.getPrecision());

            if (field.getDigits() > 0)
               System.out.println("\tDigits: " + field.getDigits());

            if (field.getCustom())
               System.out.println("\tThis is a custom field.");

            // Write the permissions of this field
            if (field.getNillable())
               System.out.println("\tCan be nulled.");
            if (field.getCreateable())
               System.out.println("\tCreateable");
            if (field.getFilterable())
               System.out.println("\tFilterable");
            if (field.getUpdateable())
               System.out.println("\tUpdateable");

            // If this is a picklist field, show the picklist values
            if (field.getType().equals(FieldType.picklist)) {
               System.out.println("\t\tPicklist values: ");
               PicklistEntry[] picklistValues = field.getPicklistValues();

               for (int j = 0; j < field.getPicklistValues().length; j++) {
                  System.out.println("\t\tValue: "
                        + picklistValues[j].getValue());
               }
            }

            // If this is a foreign key field (reference),
            // show the values
            if (field.getType().equals(FieldType.reference)) {
               System.out.println("\tCan reference these objects:");
               for (int j = 0; j < field.getReferenceTo().length; j++) {
                  System.out.println("\t\t" + field.getReferenceTo()[j]);
               }
            }
            System.out.println("");
         }
      } catch (ConnectionException ce) {
         ce.printStackTrace();
```

```
        }
    }

    private void querySample() {
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        try {
            QueryResult qr = connection.query(soqlQuery);
            boolean done = false;

            if (qr.getSize() > 0) {
                System.out.println("\nLogged-in user can see "
                        + qr.getRecords().length + " contact records.");

                while (!done) {
                    System.out.println("");
                    SObject[] records = qr.getRecords();
                    for (int i = 0; i < records.length; ++i) {
                        Contact con = (Contact) records[i];
                        String fName = con.getFirstName();
                        String lName = con.getLastName();

                        if (fName == null) {
                            System.out.println("Contact " + (i + 1) + ": " + lName);
                        } else {
                            System.out.println("Contact " + (i + 1) + ": " + fName
                                    + " " + lName);
                        }
                    }

                    if (qr.isDone()) {
                        done = true;
                    } else {
                        qr = connection.queryMore(qr.getQueryLocator());
                    }
                }
            } else {
                System.out.println("No records found.");
            }
        } catch (ConnectionException ce) {
            ce.printStackTrace();
        }
    }
}
```

C# サンプルコード

ここでは、サンプル C# クライアントアプリケーションについて説明します。このサンプルアプリケーションでは、ログインするために必要なステップを示し、いくつかの API コールを呼び出し、引き続いて処理する方法を説明します。

このサンプルアプリケーションでは、次の主要なタスクを実行します。

1. Salesforce のユーザー名とパスワードの入力をユーザーに促します。

2. `login()` をコールして、シングルログインサーバーにログインします。ログインが成功すると、次を実行します。
   - 返された `sessionId` を、後続の API コールのセッション認証に必要なセッションヘッダーに設定します。
   - Lightning プラットフォームエンドポイントを、後続の API コールの対象となる返された `serverUrl` にリセットします。

     API にアクセスするすべてのクライアントアプリケーションは、後続の API コールを実行する前に、このステップでタスクを完了する必要があります。
   - ユーザー情報を取得し、セッション情報と共にコンソールに書き込みます。
3. `describeGlobal()` をコールし、組織のデータで使用できるすべてのオブジェクトの一覧を取得します。`describeGlobal` メソッドは、ログインユーザーが使用できるオブジェクトを指定します。API コールから返されるデータは頻繁に変更されることはないため、この API コールは、セッションごとに何度も呼び出すべきではありません。DescribeGlobalResult はコンソールにエコーとして返されます。
4. `describeSObjects()` をコールして、指定されたオブジェクトのメタデータ (項目リストとオブジェクトプロパティ) を取得します。`describeSObject` メソッドは、ユーザーが使用できる各オブジェクトに取得できるメタデータ情報の種類について説明しています。サンプルクライアントアプリケーションはユーザーが指定するオブジェクトに `describeSObjects()` コールを実行し、返されたメタデータ情報をコンソールにエコーとして返します。オブジェクトメタデータ情報には、権限、データ型、選択リスト項目の長さと使用できる値、`referenceTo` 項目のデータ型が記載されています。
5. 単純なクエリ文字列 (`"SELECT FirstName, LastName FROM Contact"`) を渡し、`query()` を呼び出し、返される QueryResult を反復処理します。
6. `logout()` をコールして、ユーザーをログアウトします。

次のサンプルコードでは try/catch ブロックを使用して、API コールで返される可能性のある例外を処理します。

次のコードによって、サンプル C# クライアントアプリケーションが開始されます。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Services.Protocols;
using Walkthrough.sforce;

namespace Walkthrough
{

   class QuickstartApiSample
   {
      private SforceService binding;

      [STAThread]
      static void Main(string[] args)
      {
         QuickstartApiSample sample = new QuickstartApiSample();
         sample.run();
      }
```

```
public void run()
{
   // Make a login call
   if (login())
   {
      // Do a describe global
      describeGlobalSample();

      // Describe an account object
      describeSObjectsSample();

      // Retrieve some data using a query
      querySample();

      // Log out
      logout();
   }
}

private bool login()
{
   Console.Write("Enter username: ");
   string username = Console.ReadLine();
   Console.Write("Enter password: ");
   string password = Console.ReadLine();

   // Create a service object
   binding = new SforceService();

   // Timeout after a minute
   binding.Timeout = 60000;

   // Try logging in
   LoginResult lr;
   try
   {

      Console.WriteLine("\nLogging in...\n");
      lr = binding.login(username, password);
   }

   // ApiFault is a proxy stub generated from the WSDL contract when
   // the web service was imported
   catch (SoapException e)
   {
      // Write the fault code to the console
      Console.WriteLine(e.Code);

      // Write the fault message to the console
      Console.WriteLine("An unexpected error has occurred: " + e.Message);

      // Write the stack trace to the console
      Console.WriteLine(e.StackTrace);
```

```
            // Return False to indicate that the login was not successful
            return false;
        }



        // Check if the password has expired
        if (lr.passwordExpired)
        {
            Console.WriteLine("An error has occurred. Your password has expired.");
            return false;
        }


        /** Once the client application has logged in successfully, it will use
         * the results of the login call to reset the endpoint of the service
         * to the virtual server instance that is servicing your organization
         */
        // Save old authentication end point URL
        String authEndPoint = binding.Url;
        // Set returned service endpoint URL
        binding.Url = lr.serverUrl;

        /** The sample client application now has an instance of the SforceService
         * that is pointing to the correct endpoint. Next, the sample client
         * application sets a persistent SOAP header (to be included on all
         * subsequent calls that are made with SforceService) that contains the
         * valid sessionId for our login credentials. To do this, the sample
         * client application creates a new SessionHeader object and persist it to
         * the SforceService. Add the session ID returned from the login to the
         * session header
         */
        binding.SessionHeaderValue = new SessionHeader();
        binding.SessionHeaderValue.sessionId = lr.sessionId;

        printUserInfo(lr, authEndPoint);

        // Return true to indicate that we are logged in, pointed
        // at the right URL and have our security token in place.
        return true;
    }

    private void printUserInfo(LoginResult lr, String authEP)
    {
        try
        {
            GetUserInfoResult userInfo = lr.userInfo;

            Console.WriteLine("\nLogging in ...\n");
            Console.WriteLine("UserID: " + userInfo.userId);
            Console.WriteLine("User Full Name: " +
                userInfo.userFullName);
            Console.WriteLine("User Email: " +
```

```
            userInfo.userEmail);
        Console.WriteLine();
        Console.WriteLine("SessionID: " +
            lr.sessionId);
        Console.WriteLine("Auth End Point: " +
            authEP);
        Console.WriteLine("Service End Point: " +
            lr.serverUrl);
        Console.WriteLine();
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
}

private void logout()
{
    try
    {
        binding.logout();
        Console.WriteLine("Logged out.");
    }
    catch (SoapException e)
    {
        // Write the fault code to the console
        Console.WriteLine(e.Code);

        // Write the fault message to the console
        Console.WriteLine("An unexpected error has occurred: " + e.Message);

        // Write the stack trace to the console
        Console.WriteLine(e.StackTrace);
    }
}

/**
* To determine the objects that are available to the logged-in
* user, the sample client application executes a describeGlobal
* call, which returns all of the objects that are visible to
* the logged-in user. This call should not be made more than
* once per session, as the data returned from the call likely
* does not change frequently. The DescribeGlobalResult is
* simply echoed to the console.
*/
private void describeGlobalSample()
{
    try
    {
        // describeGlobal() returns an array of object results that
        // includes the object names that are available to the logged-in user.
        DescribeGlobalResult dgr = binding.describeGlobal();
```

```
        Console.WriteLine("\nDescribe Global Results:\n");
        // Loop through the array echoing the object names to the console

        for (int i = 0; i < dgr.sobjects.Length; i++)
        {
           Console.WriteLine(dgr.sobjects[i].name);
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An exception has occurred: " + e.Message +
            "\nStack trace: " + e.StackTrace);
    }
}

/**
* The following method illustrates the type of metadata
* information that can be obtained for each object available
* to the user. The sample client application executes a
* describeSObject call on a given object and then echoes
* the returned metadata information to the console. Object
* metadata information includes permissions, field types
* and length and available values for picklist fields
* and types for referenceTo fields.
*/
private void describeSObjectsSample()
{
    Console.Write("\nType the name of the object to " +
        "describe (try Account): ");
    string objectType = Console.ReadLine();
    try
    {

        // Call describeSObjects() passing in an array with one object type name
        DescribeSObjectResult[] dsrArray =
              binding.describeSObjects(new string[] { objectType });

        // Since we described only one sObject, we should have only
        // one element in the DescribeSObjectResult array.
        DescribeSObjectResult dsr = dsrArray[0];

        // First, get some object properties
        Console.WriteLine("\n\nObject Name: " + dsr.name);

        if (dsr.custom) Console.WriteLine("Custom Object");
        if (dsr.label != null) Console.WriteLine("Label: " + dsr.label);

        // Get the permissions on the object
        if (dsr.createable) Console.WriteLine("Createable");
        if (dsr.deletable) Console.WriteLine("Deleteable");
        if (dsr.queryable) Console.WriteLine("Queryable");
        if (dsr.replicateable) Console.WriteLine("Replicateable");
        if (dsr.retrieveable) Console.WriteLine("Retrieveable");
        if (dsr.searchable) Console.WriteLine("Searchable");
```

```
        if (dsr.undeletable) Console.WriteLine("Undeleteable");
        if (dsr.updateable) Console.WriteLine("Updateable");

        Console.WriteLine("Number of fields: " + dsr.fields.Length);

        // Now, retrieve metadata for each field
        for (int i = 0; i < dsr.fields.Length; i++)
        {
           // Get the field
           Field field = dsr.fields[i];

           // Write some field properties
           Console.WriteLine("Field name: " + field.name);
           Console.WriteLine("\tField Label: " + field.label);

           // This next property indicates that this
           // field is searched when using
           // the name search group in SOSL
           if (field.nameField)
              Console.WriteLine("\tThis is a name field.");

           if (field.restrictedPicklist)
              Console.WriteLine("This is a RESTRICTED picklist field.");

           Console.WriteLine("\tType is: " + field.type.ToString());

           if (field.length > 0)
              Console.WriteLine("\tLength: " + field.length);

           if (field.scale > 0)
              Console.WriteLine("\tScale: " + field.scale);

           if (field.precision > 0)
              Console.WriteLine("\tPrecision: " + field.precision);

           if (field.digits > 0)
              Console.WriteLine("\tDigits: " + field.digits);

           if (field.custom)
              Console.WriteLine("\tThis is a custom field.");

           // Write the permissions of this field
           if (field.nillable) Console.WriteLine("\tCan be nulled.");
           if (field.createable) Console.WriteLine("\tCreateable");
           if (field.filterable) Console.WriteLine("\tFilterable");
           if (field.updateable) Console.WriteLine("\tUpdateable");

           // If this is a picklist field, show the picklist values
           if (field.type.Equals(fieldType.picklist))
           {
              Console.WriteLine("\tPicklist Values");
              for (int j = 0; j < field.picklistValues.Length; j++)
                 Console.WriteLine("\t\t" + field.picklistValues[j].value);
           }
```

```
            // If this is a foreign key field (reference),
            // show the values
            if (field.type.Equals(fieldType.reference))
            {
               Console.WriteLine("\tCan reference these objects:");
               for (int j = 0; j < field.referenceTo.Length; j++)
                  Console.WriteLine("\t\t" + field.referenceTo[j]);
            }
            Console.WriteLine("");
         }
      }
      catch (SoapException e)
      {
         Console.WriteLine("An exception has occurred: " + e.Message +
             "\nStack trace: " + e.StackTrace);
      }
      Console.WriteLine("Press ENTER to continue...");
      Console.ReadLine();
   }

   private void querySample()
   {
      String soqlQuery = "SELECT FirstName, LastName FROM Contact";
      try
      {
         QueryResult qr = binding.query(soqlQuery);
         bool done = false;

         if (qr.size > 0)
         {
            Console.WriteLine("Logged-in user can see "
                  + qr.records.Length + " contact records.");

            while (!done)
            {
               Console.WriteLine("");
               sObject[] records = qr.records;
               for (int i = 0; i < records.Length; i++)
               {
                  Contact con = (Contact)records[i];
                  string fName = con.FirstName;
                  string lName = con.LastName;
                  if (fName == null)
                     Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                  else
                     Console.WriteLine("Contact " + (i + 1) + ": " + fName
                           + " " + lName);
               }

               if (qr.done)
               {
                  done = true;
               }
```

```
                    else
                    {
                        qr = binding.queryMore(qr.queryLocator);
                    }
                }
            }
            else
            {
                Console.WriteLine("No records found.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("\nFailed to execute query succesfully," +
                "error message was: \n{0}", ex.Message);
        }
        Console.WriteLine("\nPress ENTER to continue...");
        Console.ReadLine();
    }
  }
}
```

次の C# の例は、.NET 2.0 SforceService サービスの代わりに .NET 3.0 SoapClient サービスを使用している点を除き、前の C# の例と同じです。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.ServiceModel;
using Walkthrough.sforce;

namespace Walkthrough
{
    class QuickstartApiSample
    {
        private static SoapClient loginClient; // for login endpoint
        private static SoapClient client; // for API endpoint
        private static SessionHeader header;
        private static EndpointAddress endpoint;

        static void Main(string[] args)
        {
            QuickstartApiSample sample = new QuickstartApiSample();
            sample.run();
        }

        public void run()
        {
            // Make a login call
            if (login())
            {
                // Do a describe global
```

```
            describeGlobalSample();

            // Describe an account object
            describeSObjectsSample();

            // Retrieve some data using a query
            querySample();

            // Log out
            logout();
        }
    }

    private bool login()
    {
        Console.Write("Enter username: ");
        string username = Console.ReadLine();
        Console.Write("Enter password: ");
        string password = Console.ReadLine();

        // Create a SoapClient specifically for logging in
        loginClient = new SoapClient();

        // (combine pw and token if necessary)
        LoginResult lr;
        try
        {
            Console.WriteLine("\nLogging in...\n");
            lr = loginClient.login(null, username, password);
        }
        catch (Exception e)
        {
            // Write the fault message to the console
            Console.WriteLine("An unexpected error has occurred: " + e.Message);

            // Write the stack trace to the console
            Console.WriteLine(e.StackTrace);
            return false;
        }

        // Check if the password has expired
        if (lr.passwordExpired)
        {
            Console.WriteLine("An error has occurred. Your password has expired.");
            return false;
        }

        /** Once the client application has logged in successfully, it will use
         * the results of the login call to reset the endpoint of the service
         * to the virtual server instance that is servicing your organization
         */

        // On successful login, cache session info and API endpoint info
        endpoint = new EndpointAddress(lr.serverUrl);
```

```
        /** The sample client application now has a cached EndpointAddress
        * that is pointing to the correct endpoint. Next, the sample client
        * application sets a persistent SOAP header that contains the
        * valid sessionId for our login credentials. To do this, the sample
        * client application creates a new SessionHeader object. Add the session
        * ID returned from the login to the session header
        */
        header = new SessionHeader();
        header.sessionId = lr.sessionId;

        // Create and cache an API endpoint client
        client = new SoapClient("Soap", endpoint);

        printUserInfo(lr, lr.serverUrl);

        // Return true to indicate that we are logged in, pointed
        // at the right URL and have our security token in place.
        return true;
    }

    private void printUserInfo(LoginResult lr, String authEP)
    {
        try
        {
            GetUserInfoResult userInfo = lr.userInfo;

            Console.WriteLine("\nLogging in ...\n");
            Console.WriteLine("UserID: " + userInfo.userId);
            Console.WriteLine("User Full Name: " +
                userInfo.userFullName);
            Console.WriteLine("User Email: " +
                userInfo.userEmail);
            Console.WriteLine();
            Console.WriteLine("SessionID: " +
                lr.sessionId);
            Console.WriteLine("Auth End Point: " +
                authEP);
            Console.WriteLine("Service End Point: " +
                lr.serverUrl);
            Console.WriteLine();
        }
        catch (Exception e)
        {
            Console.WriteLine("An unexpected error has occurred: " + e.Message +
                " Stack trace: " + e.StackTrace);
        }
    }

    private void logout()
    {
        try
        {
            client.logout(header);
```

```
            Console.WriteLine("Logged out.");
        }
        catch (Exception e)
        {
            // Write the fault message to the console
            Console.WriteLine("An unexpected error has occurred: " + e.Message);

            // Write the stack trace to the console
            Console.WriteLine(e.StackTrace);
        }
    }

    /**
    * To determine the objects that are available to the logged-in
    * user, the sample client application executes a describeGlobal
    * call, which returns all of the objects that are visible to
    * the logged-in user. This call should not be made more than
    * once per session, as the data returned from the call likely
    * does not change frequently. The DescribeGlobalResult is
    * simply echoed to the console.
    */
    private void describeGlobalSample()
    {
        try
        {
            // describeGlobal() returns an array of object results that
            // includes the object names that are available to the logged-in user.
            DescribeGlobalResult dgr = client.describeGlobal(
                      header, // session header
                      null // package version header
                      );

            Console.WriteLine("\nDescribe Global Results:\n");
            // Loop through the array echoing the object names to the console

            for (int i = 0; i < dgr.sobjects.Length; i++)
            {
                Console.WriteLine(dgr.sobjects[i].name);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("An exception has occurred: " + e.Message +
                "\nStack trace: " + e.StackTrace);
        }
    }

    /**
    * The following method illustrates the type of metadata
    * information that can be obtained for each object available
    * to the user. The sample client application executes a
    * describeSObject call on a given object and then echoes
    * the returned metadata information to the console. Object
    * metadata information includes permissions, field types
```

```
* and length and available values for picklist fields
* and types for referenceTo fields.
*/
private void describeSObjectsSample()
{
    Console.Write("\nType the name of the object to " +
         "describe (try Account): ");
    string objectType = Console.ReadLine();
    try
    {

        // Call describeSObjects() passing in an array with one object type name

        DescribeSObjectResult[] dsrArray =
              client.describeSObjects(
                header, // session header
                null, // package version header
                null, // locale options
                new string[] { objectType } // object name array
                );

        // Since we described only one sObject, we should have only
        // one element in the DescribeSObjectResult array.
        DescribeSObjectResult dsr = dsrArray[0];

        // First, get some object properties
        Console.WriteLine("\n\nObject Name: " + dsr.name);

        if (dsr.custom) Console.WriteLine("Custom Object");
        if (dsr.label != null) Console.WriteLine("Label: " + dsr.label);

        // Get the permissions on the object
        if (dsr.createable) Console.WriteLine("Createable");
        if (dsr.deletable) Console.WriteLine("Deleteable");
        if (dsr.queryable) Console.WriteLine("Queryable");
        if (dsr.replicateable) Console.WriteLine("Replicateable");
        if (dsr.retrieveable) Console.WriteLine("Retrieveable");
        if (dsr.searchable) Console.WriteLine("Searchable");
        if (dsr.undeletable) Console.WriteLine("Undeleteable");
        if (dsr.updateable) Console.WriteLine("Updateable");

        Console.WriteLine("Number of fields: " + dsr.fields.Length);

        // Now, retrieve metadata for each field
        for (int i = 0; i < dsr.fields.Length; i++)
        {
            // Get the field
            Field field = dsr.fields[i];

            // Write some field properties
            Console.WriteLine("Field name: " + field.name);
            Console.WriteLine("\tField Label: " + field.label);

            // This next property indicates that this
```

```
            // field is searched when using
            // the name search group in SOSL
            if (field.nameField)
                Console.WriteLine("\tThis is a name field.");

            if (field.restrictedPicklist)
                Console.WriteLine("This is a RESTRICTED picklist field.");

            Console.WriteLine("\tType is: " + field.type.ToString());

            if (field.length > 0)
                Console.WriteLine("\tLength: " + field.length);

            if (field.scale > 0)
                Console.WriteLine("\tScale: " + field.scale);

            if (field.precision > 0)
                Console.WriteLine("\tPrecision: " + field.precision);

            if (field.digits > 0)
                Console.WriteLine("\tDigits: " + field.digits);

            if (field.custom)
                Console.WriteLine("\tThis is a custom field.");

            // Write the permissions of this field
            if (field.nillable) Console.WriteLine("\tCan be nulled.");
            if (field.createable) Console.WriteLine("\tCreateable");
            if (field.filterable) Console.WriteLine("\tFilterable");
            if (field.updateable) Console.WriteLine("\tUpdateable");

            // If this is a picklist field, show the picklist values
            if (field.type.Equals(fieldType.picklist))
            {
                Console.WriteLine("\tPicklist Values");
                for (int j = 0; j < field.picklistValues.Length; j++)
                    Console.WriteLine("\t\t" + field.picklistValues[j].value);
            }

            // If this is a foreign key field (reference),
            // show the values
            if (field.type.Equals(fieldType.reference))
            {
                Console.WriteLine("\tCan reference these objects:");
                for (int j = 0; j < field.referenceTo.Length; j++)
                    Console.WriteLine("\t\t" + field.referenceTo[j]);
            }
            Console.WriteLine("");
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("An exception has occurred: " + e.Message +
            "\nStack trace: " + e.StackTrace);
```

```
        }
        Console.WriteLine("Press ENTER to continue...");
        Console.ReadLine();
    }

    private void querySample()
    {
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        try
        {
            QueryResult qr = client.query(
                header, // session header
                null, // query options
                null, // mru options
                null, // package version header
                soqlQuery // query string
                );

            bool done = false;

            if (qr.size > 0)
            {
                Console.WriteLine("Logged-in user can see "
                    + qr.records.Length + " contact records.");

                while (!done)
                {
                    Console.WriteLine("");
                    sObject[] records = qr.records;
                    for (int i = 0; i < records.Length; i++)
                    {
                        Contact con = (Contact)records[i];
                        string fName = con.FirstName;
                        string lName = con.LastName;
                        if (fName == null)
                            Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                        else
                            Console.WriteLine("Contact " + (i + 1) + ": " + fName
                               + " " + lName);
                    }

                    if (qr.done)
                    {
                        done = true;
                    }
                    else
                    {
                        qr = client.queryMore(
                                header, // session header
                                null, // query options
                                qr.queryLocator // query locator
                                );
                    }
                }
```

```
                }
                else
                {
                    Console.WriteLine("No records found.");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("\nFailed to execute query succesfully," +
                    "error message was: \n{0}", ex.Message);
            }
            Console.WriteLine("\nPress ENTER to continue...");
            Console.ReadLine();
        }
    }
}
```

第 4 章　API コールで使用されるコアデータ型

トピック:

API コールで使用されるコアデータ型とエラー処理オブジェクトの広範なリストです。

API のコールの多くは次のデータ型を使用します。

- sObject
- ID (文字列)。「ID データ型」を参照してください。

API はまたいくつかのエラー処理オブジェクトも使用します。SOAP 要求の最中にエラーが発生すると、API は SOAP 障害メッセージを返します。このメッセージには、エラーの型によってさまざまな内容が含まれます。

- エラーが要求全体に影響を与える場合、API 障害要素が返されます。API 障害要素には、ExceptionCode と関連するエラーメッセージテキストが含まれます。
- エラーがいくつかのレコードには影響を与えるものの、その他のレコードには影響を与えない場合、StatusCode を含む Error が返されます。一般的に、これらのエラーは単一のコールで複数のレコードの作成、更新、削除などを行う一括処理で発生します。

例外コード、状況コード、拡張エラーコードのリストは、組織の WSDL ファイルで確認できます。有効になっている機能によっては、WSDL で一部のコードが表示されません。「Web サービス WSDL を生成または取得する」を参照してください。

sObject

sObject は、Account や Campaign などのオブジェクトを表します。標準オブジェクトの一覧は、「標準オブジェクト」を参照してください。

sObject には次のプロパティがあります。

名前	型	説明
fieldsToNull	string[]	明示的に `null` 値を設定する 1 つ以上の項目名の配列。 `update()` または `upsert()` と共に使用した場合、更新可能で `nillable` プロパティを持つ項目のみを指定できます。`create()` と共に使用した場合、作成可能で `nillable` プロパティまたは `default on create` プロパティを持つ項目のみを指定できます。 たとえば、ID 項目または必須項目を選択してランタイムエラーが発生した場合、その項目名は `fieldsToNull` で指定できます。同

名前	型	説明
		様に、選択リスト項目にデフォルト値があり、代わりに値を null に設定する場合、fieldsToNull で項目を指定します。
ID	ID	個別のオブジェクトの一意な ID。create() コールでは、この値は null 値となります。その他のすべての API コールでは、値を指定する必要があります。

API 障害要素

ApiFault 要素には、サービス要求の処理時に発生した障害に関する情報が含まれます。ApiFault 要素には次のプロパティがあります。

名前	型	説明
exceptionCode	ExceptionCode	例外を特徴付けるコード。
exceptionMessage	string	例外メッセージテキスト。
extendedErrorDetails	ExtendedErrorDetails	拡張エラーコードや追加のエラープロパティ (使用可能な場合) を含む、例外に関する詳細。

次の表に、発生する可能性のあるすべての API 障害を表す API 障害要素を示します。

障害	説明
ApiQueryFault	問題が発生した行番号と列番号。
LoginFault	login() コール中にエラーが発生したことを示します。
InvalidSObjectFault	describeSObject()、describeSObjects()、describeLayout()、describeDataCategoryGroups()、describeDataCategoryGroupStructures()、create()、update()、retrieve()、または query() コールに無効な sObject があります。
InvalidFieldFault	retrieve() または query() コールに無効な項目があります。
InvalidNewPasswordFault	describeAppMenu() コールに無効な appMenuType があります。
MalformedQueryFault	query() コールに渡された queryString に問題があります。
InvalidQueryLocatorFault	queryMore() コールに渡された queryLocator の問題。

障害	説明
MalformedSearchFault	search() コールに渡された search の問題。
InvalidIdFault	setPassword() または resetPassword() コールで指定された ID が無効です。
UnexpectedErrorFault	予期しないエラーが発生しました。エラーは、他の API 障害とは関連付けられていません。

ExceptionCode

エラーが発生した場合に返される可能性がある例外コードを次に示します。

APEX_REST_SERVICES_DISABLED
ユーザーの Apex REST Services 権限が有効になっていません。Apex クラスおよびメソッドに REST Web サービスとしてアクセスするには、このユーザー権限を有効にします。

API_CURRENTLY_DISABLED
システムの問題により、API 機能は一時的に利用できません。

API_DISABLED_FOR_ORG
組織で API アクセスが有効になっていません。API アクセスを有効にするには、Salesforce にご連絡ください。

BIG_OBJECT_UNSUPPORTED_OPERATION
Big Object ではこの操作はサポートされていません。

CANT_ADD_STANDARD_PORTAL_USER_TO_TERRITORY
標準ポータルライセンスを持つユーザーをテリトリーには追加できません。

CATEGORY_NOT_FOUND
商品カテゴリが見つかりません。

CIRCULAR_OBJECT_GRAPH
オブジェクトの循環参照が含まれているため、要求が失敗しました。

CLIENT_NOT_ACCESSIBLE_FOR_USER
現在のユーザーには、指定されたクライアントにアクセスする権限がありません。

CLIENT_REQUIRE_UPDATE_FOR_USER
現在のユーザーは指定されたクライアントの新しいバージョンを使用する必要があり、クライアントが更新されるまでアクセスできません。

CLONE_NOT_SUPPORTED
このエンティティは clone 操作はサポートしません。

CLONE_FIELD_INTEGRITY_EXCEPTION
clone 操作中は項目整合性例外が発生します。

COMMERCE_ADMIN_MISCONFIGURATION
コマース管理者による設定がエラーの原因です。

CONTENT_ALREADY_AN_ASSET_EXCEPTION
ID が {id} であるファイルはすでにアセットです。

CONTENT_HUB_AUTHENTICATION_EXCEPTION
認証トークンは期限切れです。

CONTENT_HUB_FILE_HAS_NO_URL_EXCEPTION
ファイルを開けません。

CONTENT_HUB_FILE_NOT_FOUND_EXCEPTION
オブジェクトが見つかりません。

CONTENT_HUB_INVALID_PAGE_NUMBER_EXCEPTION
このドキュメントにはコンテンツがありません。

CONTENT_CUSTOM_DOWNLOAD_EXCEPTION
Apex の実装で、この ExceptionCode に対応するカスタムエラーメッセージを作成します。

CONTENT_HUB_INVALID_OBJECT_TYPE_EXCEPTION
オブジェクト種別が無効です。

CONTENT_HUB_INVALID_RENDITION_PAGE_NUMBER_EXCEPTION
変換ページ番号が無効です。

CONTENT_HUB_ITEM_TYPE_NOT_FOUND_EXCEPTION
項目種別が見つかりません。

CONTENT_HUB_OBJECT_NOT_FOUND_EXCEPTION
オブジェクトが見つかりません。

CONTENT_HUB_OPERATION_NOT_SUPPORTED_EXCEPTION
操作がサポートされていません。

CONTENT_HUB_SECURITY_EXCEPTION
操作が許可されていません。

CONTENT_HUB_TIMEOUT_EXCEPTION
操作がタイムアウトになりました。

CONTENT_HUB_UNEXPECTED_EXCEPTION
この操作の実行中にエラーが発生しました。

CONTENT_IMAGE_SCALING_INVALID_ARGUMENTS_EXCEPTION
操作の引数種別が無効です。

CONTENT_IMAGE_SCALING_INVALID_IMAGE_EXCEPTION
指定された画像が無効です。

CONTENT_IMAGE_SCALING_MAX_RENDITIONS_EXCEPTION
画像変換の最大回数に達しました。

CONTENT_IMAGE_SCALING_TIMEOUT_EXCEPTION
画像のスケーリング操作がタイムアウトになりました。

CONTENT_IMAGE_SCALING_UNKNOWN_EXCEPTION
画像のスケーリング中にシステムで内部エラーが発生しました。この問題を Salesforce に報告してください。

DELETE_REQUIRED_ON_CASCADE
delete操作によって、レコードに対するカスケード削除がトリガーされましたが、ログインユーザーは、その関連オブジェクトの削除権限を持っていません。

DUPLICATE_COMM_NICKNAME
別のユーザーと同じニックネームのユーザーは作成できません。

DUPLICATE_VALUE
一意でなければならない項目に重複する値は指定できません。たとえば、`invalidateSessions()` コールで同じセッション ID の複製を 2 つ送信できません。

EMAIL_BATCH_SIZE_LIMIT_EXCEEDED
メソッドが、最大バッチサイズよりも多くのメールレコードを処理しようとしました。

EMAIL_TO_CASE_INVALID_ROUTING
処理のためにメール-to-ケースレコードが送信されましたが、機能が有効になっていません。

EMAIL_TO_CASE_LIMIT_EXCEEDED
メール-to-ケース機能のメール変換の一日の上限を超えました。

EMAIL_TO_CASE_NOT_ENABLED
メール-to-ケース機能が有効ではありません。

ENTITY_NOT_QUERYABLE
参照しようとしているオブジェクトはサポートされていません。ユーザーインターフェース API がサポートしているオブジェクトを使用してください。

EXCEEDED_ID_LIMIT
コールで指定された ID の数が多すぎます。たとえば、`retrieve()` コールで 2000 を超える ID が要求されたか、`logout()` コールで 200 を超えるセッション ID が指定されました。

EXCEEDED_LEAD_CONVERT_LIMIT
`convertLead()` コールに渡された ID の数が多すぎます。

EXCEEDED_MAX_SEMIJOIN_SUBSELECTS
リストビューに適用されたトピック検索条件が多すぎます。

EXCEEDED_MAX_SIZE_REQUEST
API に送信されたメッセージサイズが 50 MB を超えています。

EXCEEDED_MAX_TYPES_LIMIT
記述するオブジェクト種別の数が大きすぎます。

EXCEEDED_QUOTA
`create()` コール中に組織のデータストレージのサイズ制限を超えました。

EXTERNAL_SERVICE_CONNECTION_EXCEPTION
アプリケーションが外部サービスに接続できず、HTTPエラーが返されました。たとえば、サーバーが 502 「無効なゲートウェイです」のエラー、403 Forbidden エラー、503 「サービスは使用できません」エラーを返したときに、この例外コードが返される場合があります。

EXTERNAL_SERVICE_EXCEPTION
外部サービスがエラーを返しました。

EXTERNAL_SERVICE_INVALID_STATE_EXCEPTION
サービスが無効な状態です。たとえば、外部サービスを使用しているときに、外部サービスアクションを呼び出してバリデーションエラーが発生した場合には、この例外コードを受け取ります。

FUNCTIONALITY_NOT_ENABLED
機能が一時的に無効になっています。その他のコールの動作は続行します。

GONE
要求されたリソースまたは操作は、廃止または削除されました。このリソースまたは操作へのすべての参照を削除または更新してください。

IAS_TIMEOUT_EXCEPTION
操作がタイムアウトになりました。

IDEMPOTENCY_FEATURE_NOT_ENABLED
組織で、冪等レコード書き込みが有効になっていません。この機能を有効にするには、Salesforce に連絡してください。

IDEMPOTENCY_NOT_SUPPORTED
現在の要求のリソースまたは HTTP メソッドが、冪等性に対応していません。

INACTIVE_OWNER_OR_USER
ユーザーまたはレコード所有者が有効ではありません。

INACTIVE_PORTAL
参照されたポータルが無効です。

INDEX_NOT_FOUND
検索インデックスが見つかりません。

INSUFFICIENT_ACCESS
ユーザーには操作を実行するのに十分な権限がありません。

INSUFFICIENT_BENEFIT_REMAINING
指定された給付割り当て {id} の残りの金額が支払金額未満です。

INTERNAL_ERROR
システムで内部エラーが発生しました。この問題を Salesforce に報告してください。

INVALID_ACCOUNTING_SET
会計セットが有効ではないか、存在しません。

INVALID_ASSIGNMENT_RULE
無効な AssignmentRuleHeader 値が指定されました。

INVALID_BATCH_REQUEST
バッチレコードの操作が無効です。「Create a Batch of Records (バッチレコードの作成)」を参照してください。

INVALID_BATCH_SIZE
クエリオプションのバッチサイズ値が無効です。

INVALID_CLIENT
クライアントが無効です。

INVALID_CROSS_REFERENCE_KEY
項目に無効な外部キーを設定できません。たとえば、AccountShare のようなオブジェクト共有は、行の共有によってルールを共有しているため削除できません。

INVALID_DEFINITION
会計セットに関連する DPE 定義が有効ではありません。

INVALID_FIELD
指定された項目名が無効です。

INVALID_FILTER_LANGUAGE
指定された言語は絞り込みには使用できません。

INVALID_FILTER_VALUE
LIKE を使用した SOQL に無効な文字が指定されました。たとえば、アスタリスク (*) の位置が間違っていることなどが考えられます。クエリを修正し、再送信してください。

INVALID_GOOGLE_DOCS_URL
Google ドキュメントを Salesforce レコードに関連付ける際に使用したレコード URL が無効です。操作をもう一度実行する前に URL 修正してください。

INVALID_ID_FIELD
指定された ID の形式が正しいものの、ID が有効ではありません。たとえば、間違った型の ID や、その ID が指定したオブジェクトが存在しない場合などが考えられます。

INVALID_IDEMPOTENCY_KEY
要求ヘッダーの冪等キーは UUID V4 形式に準拠する必要があります。

INVALID_INPUT
バッチレコードの入力が無効です。「Create a Batch of Records (バッチレコードの作成)」を参照してください。

INVALID_LOCATOR
ロケーターが無効です。

INVALID_LOGIN
`login()` 情報が有効でないか、ログイン数の上限を超えています。詳細は管理者にお問い合わせください。

INVALID_NEW_PASSWORD
新しいパスワードは、組織のパスワードポリシーに適合していません。

INVALID_OPERATION
クライアントアプリケーションが承認プロセスでロックされているレコードを変更しようとしました。

INVALID_OPERATION_WITH_EXPIRED_PASSWORD
パスワード期限が切れたため、コールを実行する前に `setPassword()` を使用して有効なパスワードを設定する必要があります。

INVALID_QUERY_FILTER_OPERATOR
少なくともその項目の、`query()` 検索条件句で無効な演算子が使用されました。

INVALID_QUERY_LOCATOR
コールのクエリロケーターが無効か存在しません。`queryMore()` コール (SOAP API) または `nextRecordUrl` コール (REST API) を使用しても、より多くの結果は見つかりませんでした。別のクエリロケーターでコールをもう一度試してみてください。

INVALID_QUERY_SCOPE
指定された検索範囲が無効です。

INVALID_REPLICATION_DATE
複製の設定された日付が、組織の作成前の日付であるなど、許可された範囲を超えています。

INVALID_RECORD_ATTRIBUTE_VALUE
指定されたレコード属性が無効です。

INVALID_RUNTIME_VALUE
指定されたランタイムが無効です。

INVALID_SETUP_OWNER
設定所有者は、組織、プロファイル、ユーザーでなければなりません。

INVALID_SEARCH
`search()` コールに無効な構文または文法が含まれています。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。

INVALID_SEARCH_SCOPE
指定された検索範囲が無効です。

INVALID_SESSION_ID
指定された `sessionId` の形式が正しくないか (長さまたは形式が不正)、期限が切れています。再度ログインして新しいセッションを起動してください。

INVALID_SOAP_HEADER
SOAP ヘッダーにエラーがあります。以前のバージョンの API から移行する場合、SaveOptions ヘッダーが API 6.0 以降では使用できないことにご注意ください。代わりに、AssignmentRuleHeader を使用してください。

INVALID_SSO_GATEWAY_URL
シングルサインオンゲートウェイの設定のための URL が無効です。

INVALID_TYPE
指定された sObject の型が無効です。

INVALID_TYPE_FOR_OPERATION
指定された操作で、指定された sObject の型が無効です。

INVALID_VERSION
要求されたリソースは存在しません。要求で使用されている API のバージョン番号がサポートされていることを確認し、必要に応じて更新してください。

LIMIT_EXCEEDED
配列が長すぎます。たとえば、BCC アドレス、対象、メールメッセージが多すぎることが考えられます。

LOGIN_CHALLENGE_ISSUED
ユーザーが信頼できない IP アドレスからログインしたため、セキュリティトークンを含むメールがユーザーのメールアドレスに送信されました。ユーザーは、セキュリティトークンをパスワードの末尾に追加しないとログインできません。

LOGIN_CHALLENGE_PENDING
ユーザーが信頼できない IP アドレスからログインしましたが、セキュリティトークンがまだ発行されていません。

LOGIN_DURING_RESTRICTED_DOMAIN
ユーザーはこの IP アドレスからのログインは許可されていません。

LOGIN_DURING_RESTRICTED_TIME
ユーザーはこの時間帯にはログインできません。

LOGIN_MUST_USE_SECURITY_TOKEN
ユーザーは、セキュリティトークンをパスワードの末尾に追加してログインする必要があります。

MALFORMED_ID
無効な ID 文字列が指定されました。ID についての詳細は、「ID データ型」を参照してください。

MALFORMED_QUERY
無効なクエリ文字列が指定されました。たとえば、クエリ文字列が 100,000 文字より多かったことや、連結 ID の数が 1 つのクエリで許可される上限の 500 を超えたことなどが考えられます。

MALFORMED_SEARCH
無効な検索文字列が指定されました。たとえば、検索文字列が 100,000 文字より多い場合などが考えられます。

MISSING_ARGUMENT
必要な引数が指定されていません。

MIXED_DML_OPERATION
同じトランザクションで実行できる DML 操作の種類に制限があります。詳細は、『*Apex 開発者ガイド*』の「Data Manipulation Language」を参照してください。

NO_DEFINITION_ASSOCIATED
会計セットに DPE 定義がありません。

NOT_ACCEPTABLE
XML ファイルがブロックされています。

NOT_MODIFIED
記述用の API コール (describe) のレスポンスが、指定された日付から変更されていません。

NO_SOFTPHONE_LAYOUT
組織で CTI 機能が有効になっているものの、ソフトフォンレイアウトが定義されていない場合、記述用の API コール (describe) が発行されるとこの例外が返されます。この例外の主な原因は、コールセンターが定義されていないことです。デフォルトのソフトフォンレイアウトは、コールセンターの定義中に作成されます。

組織で CTI 機能が有効になっていない場合、代わりに `FUNCTIONALITY_NOT_ENABLED` が返されます。

NO_RECIPIENTS
受信者が指定されていません。

NUMBER_OUTSIDE_VALID_RANGE
指定された数値が、項目の有効な範囲外です。

OPERATION_TOO_LARGE
クエリから返された結果が多すぎます。「すべてのデータの参照」権限のないユーザーが特定のクエリを実行して、多くのレコードが返された場合、クエリには共有ルールチェックが必要です。たとえば、多態的な外部キーを使用する Task などのオブジェクトで実行されるクエリなどが考えられます。これらのクエリは、操作に必要なリソースが多すぎるためこの例外を返します。エラーを修正するには、検索範囲を絞り込むために検索条件を追加するか、日付の範囲などの検索条件を使用しクエリをさらに小さなクエリに分割します。

ORDER_MANAGEMENT_ACTION_NOT_ALLOWED
要求されたアクションは許可されていません。

ORG_LOCKED
組織がロックされています。組織のロックを解除するには、Salesforce にご連絡ください。

ORG_NOT_OWNED_BY_INSTANCE
ユーザーが不正なサーバーインスタンスにログインしようとしました。別のサーバーインスタンスを選択するか、組織の [私のドメイン] のログイン URL を使用するか、`https://login.salesforce.com` でログインしてください。

PASSWORD_LOCKOUT
可能なログイン試行回数を超えています。再度ログインアクセスするには、ユーザーはシステム管理者に連絡する必要があります。

PAYLOAD_ITEM_MAP_ERROR
指定された productId と IndexInfold に関連するカタログペイロードが見つかりません。

PORTAL_NO_ACCESS
指定されたポータルへのアクセスが許可されていません。

PRODUCT_NOT_FOUND
商品が見つかりません。

QUERY_TIMEOUT
クエリがタイムアウトしました。クエリの制限とその回避方法の詳細は、『Developer の制限および割り当てクイックリファレンス』の「SOQL および SOSL の検索クエリの制限」、および『Salesforce SOQL および SOSL リファレンスガイド』の「SOQL SELECT の構文」を参照してください。

QUERY_TOO_COMPLICATED
SOQL が選択している項目数が多すぎるか、検索条件が多すぎます。クエリで参照している数式項目の数を減らしてください。

REQUEST_LIMIT_EXCEEDED
要求が、組織の同時要求制限または要求レート制限を超えています。API 要求の制限についての詳細は、「API 使用状況の測定」を参照してください。

REQUEST_RUNNING_TOO_LONG
要求の処理時間が長すぎます。

SERVER_UNAVAILABLE
このコールに必要なサーバーが使用できません。他のタイプの要求はうまく行く場合もあります。

SSO_SERVICE_DOWN
サービスが利用できないため、組織が指定するシングルサインオンサーバーへの認証コールが失敗しました。

STATE_TRANSITION_NOT_ALLOWED
要求されたトランジションは許可されていません。

TOO_MANY_APEX_REQUESTS
発行された Apex 要求の数が多すぎます。この例外が続く場合は、Salesforce カスタマーサポートまでご連絡ください。

TRIAL_EXPIRED
組織のトライアル期間が終了しています。組織を再度有効にするには、会社の代表者が Salesforce に連絡する必要があります。

UNSUPPORTED_API_VERSION
アクセスされた API のバージョンには存在しないメソッドがコールされました。たとえば、バージョン 5.0 でバージョン 8.0 の新機能である `upsert()` を使おうとした場合などが考えられます。

UNSUPPORTED_CLIENT
このバージョンのクライアントはすでにサポートされていません。

WEBSTORE_NOT_FOUND
Web ストアが見つかりません。

Error

`Error` には、`create()`、`merge()`、`process()`、`update()`、`upsert()`、`delete()`、または `undelete()` コール中に発生したエラーの情報が含まれます。

詳細は、「エラー処理」を参照してください。Error には次のプロパティがあります。

名前	型	説明
statusCode	StatusCode	エラーを説明するコード。
message	string	エラーメッセージテキスト。
fields	string[]	1 つ以上の項目名の配列。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。
extendedErrorDetails	ExtendedErrorDetails	拡張エラーコードや追加エラープロパティ (使用可能な場合) を含む、エラーに関する詳細。

メモ: 組織に有効な重複ルールがあり、重複が検出された場合、SaveResult にはデータ型が DuplicateError の Error が含まれます。

StatusCode

エラーが発生すると、次のいずれかの API 状況コードが返される場合があります。

APEX_DATA_ACCESS_RESTRICTION
並列 Apex テストでは、ユーザーオブジェクトに対する DML 操作が許可されていません。代わりにテストクラスでユーザーを作成することを検討してください。

ALL_OR_NONE_OPERATION_ROLLED_BACK
いずれかのレコードが正常に処理されなかったため、一括操作はロールバックされました。「AllOrNoneHeader」を参照してください。

ALREADY_APPLIED
カートにはすでにプロモーションクーポンが適用されています。

ALREADY_IN_PROCESS
すでに承認プロセスが開始しているレコードは送信できません。前の承認プロセスが完了するまで待機してからこのレコードの要求を再送信します。

ASSIGNEE_TYPE_REQUIRED
承認申請 (ProcessInstanceStep または ProcessInstanceWorkitem) の任命先を指定します。

AURA_COMPILE_ERROR
Lightning コンポーネントにエラーが含まれており、コンパイルされません。

AUTH_PROVIDER_NEEDS_AUTH
認証プロバイダーは、適切な認証なしに接続できません。

AUTH_PROVIDER_NOT_FOUND
認証プロバイダーが見つかりません。

BAD_CUSTOM_ENTITY_PARENT_DOMAIN
関連する主従関係への変更を適用できないため、この変更を完了できません。

BAD_REQUEST
API要求が、無効であるか、構文エラーがあります。リクエストボディとパラメーターを確認します。

BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED
コンプライアンス BCC メールオプションが組織で有効になっているにも関わらず、クライアントアプリケーションがメールアドレスへBCCでメールを送信しようとしました。このオプションは送信されるメールの写しを自動的に受信する特定のメールアドレスを指定します。このオプションが有効な場合、他のメールアドレスにBCC送信できません。オプションを無効にするには、ユーザーインターフェースにログインし、[設定] から、[クイック検索] ボックスに「コンプライアンス *BCC メール*」と入力し、[コンプライアンス **BCC** メール] を選択します。

BCC_SELF_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED
コンプライアンス BCC メールオプションが組織で有効になっているにも関わらず、クライアントアプリケーションがログインユーザーのメールアドレスへ、BCC でメールを送信しようとしました。このオプションは送信されるメールの写しを自動的に受信する特定のメールアドレスを指定します。このオプションが有効な場合、他のメールアドレスにBCC送信できません。オプションを無効にするには、ユーザーインターフェースにログインし、[設定] から、[クイック検索] ボックスに「コンプライアンス *BCC メール*」と入力し、[コンプライアンス **BCC** メール] を選択します。

CANNOT_CASCADE_PRODUCT_ACTIVE
カスケードによる商品への更新は、関連付けられた商品がアクティブであるため実行できません。

CANNOT_CHANGE_FIELD_TYPE_OF_APEX_REFERENCED_FIELD
Apex スクリプトで参照されている項目のデータ型は変更できません。

CANNOT_CREATE_ANOTHER_MANAGED_PACKAGE
組織では管理パッケージは 1 つしか作成できません。

CANNOT_DEACTIVATE_DIVISION
割り当てルールがディビジョンを参照している場合、またはユーザーレコードの `DefaultDivision` 項目が null 値に設定されていない場合は、ディビジョンを無効にできません。

CANNOT_DELETE_LAST_DATED_CONVERSION_RATE
日付の入った換算が有効な場合、少なくとも 1 つの DatedConversionRate レコードが必要です。

CANNOT_DELETE_MANAGED_OBJECT
管理パッケージに含まれているコンポーネントは変更できません。

CANNOT_DISABLE_LAST_ADMIN
少なくとも 1 人の有効な管理者ユーザーが必要です。

CANNOT_ENABLE_IP_RESTRICT_REQUESTS
プロファイルに指定されている 5 つの IP 範囲を超えている場合、IP アドレスによるログイン制限を有効にできません。プロファイルの指定範囲を減らして再度要求を実行してください。

CANNOT_INSERT_UPDATE_ACTIVATE_ENTITY
指定されたレコードの作成、更新、有効化の権限がありません。

CANNOT_MODIFY_MANAGED_OBJECT
管理パッケージに含まれているコンポーネントは変更できません。

CANNOT_RENAME_APEX_REFERENCED_FIELD
Apex スクリプトで参照されている項目の名前は変更できません。

CANNOT_RENAME_APEX_REFERENCED_OBJECT
Apex スクリプトで参照されているオブジェクトの名前は変更できません。

CANNOT_REPARENT_RECORD
指定したレコードに新しい親レコードを定義できません。

CANNOT_RESOLVE_NAME
`sendEmail()` コールがオブジェクト名を解決できませんでした。

CANNOT_UPDATE_CONVERTED_LEAD
取引開始済みのリードを更新できませんでした。

CANNOT_POST_TO_ARCHIVED_GROUP
グループがアーカイブされている場合は、グループに投稿することはできません。

CANT_DISABLE_CORP_CURRENCY
組織のマスター通貨は無効にできません。マスター通貨として設定されている通貨を無効にするには、ユーザーインターフェースを使用してマスター通貨を別の通貨に変更してから、その通貨を無効にします。

CANT_UNSET_CORP_CURRENCY
組織のマスター通貨はAPIから変更できません。マスター通貨はユーザーインターフェースを使用して変更します。

CHECKOUT_UNAUTHORIZED
チェックアウトリソースに対するアクセス権がありません。チェックアウトリソースにアクセスできるのは、ストアとカートへのアクセス権を持つ顧客のみです。

CHILD_SHARE_FAILS_PARENT
親レコードに対する適切な権限が付与されていない場合、子レコードの所有者の変更または子レコードの共有ルールの定義を行うことはできません。たとえば、取引先責任者レコードの所有者を変更するには、親の取引先レコードを編集する権限が必要です。

CIRCULAR_DEPENDENCY
組織内のメタデータオブジェクト間では循環参照は作成できません。たとえば、公開グループ A が公開グループ B に含まれている場合、公開グループ B を公開グループ A に含めることはできません。

COMMUNITY_NOT_ACCESSIBLE
このエンティティが属する Experience Cloud サイトにアクセスする権限がありません。サイトにアクセスする権限を取得してからでないと、このエンティティにはアクセスできません。

CONFLICT
ターゲットリソースの現在の状態に対して要求が競合しています。

CONFLICTING_ENVIRONMENT_HUB_MEMBER
指定された組織は、すでに別の環境ハブのメンバーになっています。

CONFLICTING_SSO_USER_MAPPING
この組織は、すでに別のユーザー経由でこの環境ハブのメンバーになっています。

CONTENT_NOT_FOUND
要求されたコンテンツが見つかりませんでした。

CONTENT_TYPE_NOT_FOUND
コンテンツタイプが見つかりません。

COUPON_REDEMPTION_LIMIT_EXCEEDED
このクーポンを利用できる最大回数に達しました。

CUSTOM_CLOB_FIELD_LIMIT_EXCEEDED
CLOB 項目の最大サイズを超えることはできません。

CUSTOM_ENTITY_OR_FIELD_LIMIT
組織のカスタムオブジェクトまたはカスタム項目の最大数に達しました。

CUSTOM_FIELD_INDEX_LIMIT_EXCEEDED
組織の項目へのインデックスの最大数に達しました。

CUSTOM_INDEX_EXISTS
項目に対して作成できるカスタムインデックスは 1 つだけです。

CUSTOM_LINK_LIMIT_EXCEEDED
組織のカスタムリンクの最大数に達しました。

CUSTOM_METADATA_LIMIT_EXCEEDED
組織のカスタムメタデータの最大制限に達しました。

CUSTOM_SETTINGS_LIMIT_EXCEEDED
組織のカスタム設定の最大制限に達しました。

CUSTOM_TAB_LIMIT_EXCEEDED
組織のカスタムタブの最大数に達しました。

DELETE_FAILED
他のオブジェクトが使用中のため、レコードを削除できません。

DEPENDENCY_EXISTS
指定されたオブジェクトまたは項目に連動関係が存在するため、要求された操作を実行できません。

DUPLICATE_CASE_SOLUTION
指定されたケースとソリューションの関係がすでに存在するため、新たに作成することはできません。

DUPLICATE_CUSTOM_ENTITY_DEFINITION
カスタムオブジェクトまたはカスタム項目の ID は一意でなければなりません。

DUPLICATE_CUSTOM_TAB_MOTIF
カスタムタブのモチーフ名は一意にする必要があります。

DUPLICATE_DEVELOPER_NAME
開発者名が重複するカスタムオブジェクトやカスタム項目は作成できません。

DUPLICATES_DETECTED
重複レコードが検出されました。データ型が DuplicateError の Error オブジェクトに使用されます。

DUPLICATE_EXTERNAL_ID
ユーザーが指定した外部 ID が更新/挿入で複数のレコードに一致しています。

DUPLICATE_MASTER_LABEL
名前が重複するカスタムオブジェクトやカスタム項目は作成できません。

DUPLICATE_SENDER_DISPLAY_NAME
`sendEmail()` コールが OrgWideEmailAddress.`DisplayName` または `senderDisplayName` から選択できませんでした。2 つの項目のいずれかのみ定義できます。

DUPLICATE_USERNAME
ユーザー名が重複しているため、作成、更新、または更新/挿入に失敗しました。

DUPLICATE_VALUE
一意でなければならない項目に重複する値は指定できません。たとえば、`invalidateSessions()` コールで同じセッション ID の複製を 2 つ送信できません。

EMAIL_ADDRESS_BOUNCED
メールが不達になった受信者が 1 人以上います。メールアドレスが有効であることを確認してくだい。

EMAIL_EXTERNAL_TRANSPORT_TOO_MANY_REQUESTS_ERROR
一定時間内に送信される要求が多すぎると Gmail または Outlook365 が判断しました。

EMAIL_NOT_PROCESSED_DUE_TO_PRIOR_ERROR
これより以前にコールで発生したエラーにより、このメールは処理されませんでした。

EMAIL_OPTED_OUT
単一のメールメッセージが `optOutPolicy` 項目の `REJECT` 設定を使用して、受信メールの送信除外を設定している受信者に送信されました。このエラーを回避するには、`optOutPolicy` 項目を別の値に設定します。

EMAIL_TEMPLATE_FORMULA_ERROR
メールテンプレートが無効か、表示不能です。テンプレートに不適切に指定された差し込み項目がないか確認してください。

EMAIL_TEMPLATE_MERGEFIELD_ACCESS_ERROR
このテンプレートの 1 つ以上の差し込み項目に対するアクセス権がありません。アクセス権を要求するには、Salesforce システム管理者に連絡してください。

EMAIL_TEMPLATE_MERGEFIELD_ERROR
1 つ以上の差し込み項目が存在しません。項目名のスペルをチェックしてください。

EMAIL_TEMPLATE_MERGEFIELD_VALUE_ERROR
1 つ以上の差し込み項目に値がありません。値を指定するには、メールを送信する前にレコードを更新します。

EMAIL_TEMPLATE_PROCESSING_ERROR
このメールテンプレートの差し込み項目は処理できません。テンプレートの本文が有効であることを確認してください。

EMPTY_SCONTROL_FILE_NAME
Scontrol のファイル名が空ですが、バイナリが空ではありません。

ENTITY_FAILED_IFLASTMODIFIED_ON_UPDATE
レコードの `LastModifiedDate` 項目の値が現在の日付より後に設定されている場合、レコードを更新できません。

ENTITY_IS_ARCHIVED
アーカイブされたレコードにはアクセスできません。

ENTITY_IS_DELETED
削除されたオブジェクトは参照できません。この状況コードは、API バージョン 10.0 以降でのみ発生します。API の以前のリリースでは、このエラーに `INVALID_ID_FIELD` を使用しています。

ENTITY_IS_LOCKED
承認プロセスでロックされているレコードは編集できません。

ENVIRONMENT_HUB_MEMBERSHIP_CONFLICT
複数の環境ハブに組織を追加することはできません。

ERROR_IN_MAILER
メールアドレスが無効であるか、メール関連のトランザクション中に別のエラーが発生しました。

EXCEEDED_MAX_SEMIJOIN_SUBSELECTS_WRITE
リストビューに適用されたトピック検索条件が多すぎます。

EXCHANGE_WEB_SERVICES_URL_INVALID
Exchange Web サービスの URL が無効です。

EXTERNAL_RESOURCE_FORBIDDEN
OAuth に適切な権限がないなど、必要な外部リソースへのアクセス権がありません。

FAILED_ACTIVATION
Contract の有効化が失敗しました。

FIELD_CUSTOM_VALIDATION_EXCEPTION
項目の整合性規則に違反するカスタム入力規則数式は定義できません。

FIELD_FILTER_VALIDATION_EXCEPTION
項目の整合性規則に違反することはできません。

FIELD_KEYWORD_LIST_MATCH_LIMIT
コメントまたはメッセージに含まれている、許可できないコンテンツに一致するキーワードが多すぎます。

FILE_EXTENSION_NOT_ALLOWED
指定されたファイル拡張子は許可されていません。複数のファイル拡張子を指定する場合は、カンマで区切ります。

FILE_SIZE_LIMIT_EXCEEDED
ファイルサイズが制限を超えました。

FILTERED_LOOKUP_LIMIT_EXCEEDED
オブジェクトごとに使用できるルックアップ検索条件の最大数を超えているため、ルックアップ検索条件の作成に失敗しました。

FIELD_MAPPING_ERROR
項目の対応付けエラーが発生しました。

FIELD_MODERATION_RULE_BLOCK
コメントまたはメッセージがモデレーションルール条件に一致したため公開できません。

FLOW_EXCEPTION
フローの保存中にエラーが発生しました。エラーの重大度には、情報、警告、有効化のブロック、保存のブロックなどがあります。

FUNCTIONALITY_NOT_ENABLED
指定された機能は無効になっています。

HAS_PUBLIC_REFERENCES
カスタムメタデータレコードは、公開レコードで参照されている間は保護として設定できません。

HTML_FILE_UPLOAD_NOT_ALLOWED
HTML ファイルのアップロードが失敗しました。[HTML ドキュメントと添付ファイルの設定] ページで [HTML ドキュメントと添付ファイルを許可しない] チェックボックスがオンである場合は、Solution への HTML 添付ファイルなど、HTML 添付ファイルおよびドキュメントをアップロードすることはできません。

IMAGE_TOO_LARGE
画像が幅、高さ、ファイルサイズの最大値を超えています。

IAS_UNCOMITTED_WORK
同じトランザクションでコミットされていないデータベースの変更が待機中である場合は、Omnichannel Inventory へのコールを実行することはできません。

INACTIVE_OWNER_OR_USER
指定されたデータの所有者が無効なユーザーです。このデータを参照するには、所有者を再度有効化するか、別の有効なユーザーに所有権を再度割り当てます。

INPUTPARAM_INCOMPATIBLE_DATATYPE
入力パラメーターの型が、割り当てられている値の型と一致しません。

INSERT_UPDATE_DELETE_NOT_ALLOWED_DURING_MAINTENANCE
バージョン32.0以降では、組織が含まれるインスタンスを最新リリースにアップグレードしている間は、データを作成、更新、削除できません。リリースが完了してから再度実行してください。リリーススケジュールについての詳細は、trust.salesforce.com を参照してください。バージョン 32.0 より前の場合、コードは INVALID_READ_ONLY_USER_DML になります。

INSUFFICIENT_ACCESS
十分な権限がないためリソースにアクセスできません。

INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY
この操作は指定されたオブジェクトが相互参照しているオブジェクトに影響を与えますが、ログインユーザーは相互参照しているオブジェクトに対する十分な権限がありません。たとえば、ログインユーザーが取引先レコードを変更しようとして、その更新により ProcessInstanceWorkitem が作成されます。ユーザーに ProcessInstanceWorkitem の承認、却下、再割り当て権限がない場合、この例外が発生します。

INSUFFICIENT_ACCESS_OR_READONLY
指定したアクションを実行する十分な権限がないため、実行できません。

INSUFFICIENT_ACCESS_TO_INSIGHTSEXTERNALDATA
InsightsExternalData オブジェクトにアクセスできません。

INSUFFICIENT_BALANCE
ロイヤルティプログラムメンバーのポイント残高が不足しているため、メンバーのポイントをデビット処理できません。

INVALID_ACCESS_LEVEL
指定された組織の共有設定よりアクセス権限の小さい共有ルールを新たに定義することはできません。

INVALID_ACCOUNT
指定したアカウントは、null であるか、有効なアカウントではないか、必須項目がないか、または書き込み不可能な項目を設定しています。

INVALID_ARGUMENT_TYPE
実行しようとする操作に対して不正な型の引数を指定しました。

INVALID_ASSIGNEE_TYPE
1 から 6 の間の有効な整数でない値を任命先種別として指定しました。

INVALID_ASSIGNMENT_RULE
無効な割り当てルールまたは組織で定義されていない割り当てルールを指定しました。

INVALID_AUTH_HEADER
要求の認証ヘッダーが有効ではありません。REST API では、`Authorization: Bearer Access_Token` の形式で HTTP 認証ヘッダーを使用します。SOAP API では、SessionHeader SOAP 認証ヘッダーを使用します。アクセストークンはヘッダーに配置されます。

INVALID_BATCH_OPERATION
指定されたバッチ操作が無効です。

INVALID_CHECKOUT_INPUT
入力がプロモーション固有ではありません。

INVALID_CONTENT_TYPE
送信メールの EmailFileAttachment の `contentType` プロパティが無効です。「RFC2045 - Internet Message Format」を参照してください。

INVALID_COUPON
入力されたクーポンが有効ではないか、または期限切れです。

INVALID_CREDIT_CARD_INFO
指定されたクレジットカード情報が有効ではありません。

INVALID_CROSS_REFERENCE_KEY
リレーション項目に指定した値が有効でないか、データが予期した型ではありません。

INVALID_CROSS_REFERENCE_TYPE_FOR_FIELD
指定された相互参照タイプが、指定された項目で有効な型ではありません。

INVALID_CONTACT
指定された取引先責任者が無効です。

INVALID_CURRENCY_CONV_RATE
通貨換算レートには、0 でない正の値を指定します。

INVALID_CURRENCY_CORP_RATE
マスター通貨の換算レートは変更できません。

INVALID_CURRENCY_ISO
指定された通貨 ISO コードが有効ではありません。

INVALID_EMAIL_ADDRESS
指定されたメールアドレスが無効です。

INVALID_EMPTY_KEY_OWNER
`owner` に null 値は設定できません。

INVALID_ENTITY_FOR_UPSERT
更新/挿入操作で、指定されたオブジェクトが無効です。

INVALID_ENVIRONMENT_HUB_MEMBER
指定されたユーザーで環境ハブメンバーを作成できません。

INVALID_EVENT_DELIVERY
イベント配信 ID が無効です。

INVALID_EVENT_SUBSCRIPTION
イベントへの登録時にパラメーター値が 765 文字の上限を超えています。

INVALID_EXTERNAL_ID_FIELD_NAME
対象エンティティの `externalIdFieldName` パラメーターの項目名が無効です。有効な項目名を指定してください。

INVALID_FIELD
レコードを更新または更新/挿入しようとしたときに無効な項目名を指定しました。

INVALID_FIELD_FOR_INSERT_UPDATE
個人取引先レコードタイプの変更を、他の項目の更新と組み合わせることはできません。

INVALID_FIELD_WHEN_USING_TEMPLATE
無効な項目名でメールテンプレートを使用することはできません。

INVALID_FILTER_ACTION
指定した検索条件アクションは、指定したオブジェクトでは使用できません。たとえば、アラートは Task の有効な条件アクションではありません。

INVALID_ID_FIELD
指定された ID 項目 (`ID`、`ownerId`) または相互参照項目が無効です。

INVALID_INET_ADDRESS
指定された Inet アドレスが無効です。

INVALID_LINEITEM_CLONE_STATE
有効でない Pricebook2 または PricebookEntry レコードをコピーすることはできません。

INVALID_MARKUP
指定されたマークアップにサポートされないタグまたは属性が含まれています。

INVALID_MERCHANT_ACCOUNT_MODE
業者アカウントが本番環境に存在しないため、支払いを受け入れるにはテストモードにする必要があります。

INVALID_MERCHANT_ACCOUNT_MODE_OR_STATUS
業者アカウントが支払いを受け入れるためには、ライブモードにするか、または状況が有効になっている必要があります。支払いを受け取るには、支払プロバイダーで業者アカウントの法人プロファイルを完成させてください。

INVALID_MERGE_RECORD
指定されたレコードをマージできませんでした。

INVALID_MASTER_OR_TRANSLATED_SOLUTION
ソリューションが無効です。たとえば、翻訳ソリューションと、別の翻訳ソリューションが関連付けられているマスターソリューションとを関連付けようとした場合にこの例外が発生します。

INVALID_MESSAGE_ID_REFERENCE
送信メールの References または In-Reply-To 項目が無効です。これらの項目には、有効なメッセージ ID が必要です。「RFC2822 - Internet Message Format」を参照してください。

INVALID_NAMESPACE_PREFIX
指定された名前空間が無効です。

INVALID_OPERATION
指定されたオブジェクトに適用可能な承認プロセスはありません。

INVALID_OPERATOR
ワークフロー条件として使用する場合、このデータ型に指定した演算子は使用できません。

INVALID_OR_NULL_FOR_RESTRICTED_PICKLIST
制限つき選択リストに無効な値または null 値を指定しました。

INVALID_PACKAGE_LICENSE
指定されたパッケージライセンスがこの組織には存在しません。

INVALID_PARTNER_NETWORK_STATUS
指定されたテンプレート項目の、指定されたパートナーネットワーク状況が無効です。

INVALID_PERSON_ACCOUNT_OPERATION
個人取引先は削除できません。

INVALID_PROFILE
指定されたプロファイルは、外部のユーザーライセンスに関連付けられていません。

INVALID_READ_ONLY_USER_DML
バージョン31.0以前:組織が含まれるインスタンスを最新リリースにアップグレードしている間は、データを作成、更新、削除できません。リリースが完了してから再度実行してください。リリーススケジュールについての詳細は、trust.salesforce.com を参照してください。バージョン 31.0 より後の場合、コードは `INSERT_UPDATE_DELETE_NOT_ALLOWED_DURING_MAINTENANCE` になります。

INVALID_RECEIVEDDOCUMENTID_ATTACHMENT
受信したドキュメントレコードには、複数の添付ファイルが含まれています。

INVALID_RECORD_TYPE
指定されたレコードタイプが無効です。

INVALID_SAVE_AS_ACTIVITY_FLAG
`saveAsActivity` フラグには `true` または `false` を指定します。

INVALID_SCS_INBOUND_USER
指定された SCS 受信ユーザーが無効です。

INVALID_SESSION_ID
指定された `sessionId` の形式が正しくないか (長さまたは形式が不正)、期限が切れています。再度ログインして新しいセッションを起動してください。

INVALID_SERVER_ERROR
予期しない状況が発生したため、API 操作ができなくなりました。

INVALID_SIGNUP_OPTION
SignupRequest オブジェクトに提供した一連の値が正しくありません。

INVALID_STATUS
指定された組織の状況変更が有効ではありません。

INVALID_TARGET_OBJECT_NAME
対象エンティティの API 名が無効です。有効な API 名を指定してください。

INVALID_TYPE
指定された型が、指定されたオブジェクトで有効ではありません。

INVALID_TYPE_FOR_OPERATION
指定された型が、指定された操作で有効ではありません。

INVALID_TYPE_ON_FIELD_IN_RECORD
指定された値が、指定された項目の型で有効ではありません。

INVALID_UNMERGE_RECORD
指定されたレコードが、すでにマージされているレコードからマージ解除されませんでした。

INVALID_USERID
指定されたユーザーは組織の有効なメンバーではありません。

INVALID_USER_OBJECT
指定されたユーザーオブジェクトが無効です。

IP_RANGE_LIMIT_EXCEEDED
指定された IP アドレスが、組織に指定された IP 範囲外です。

JIGSAW_IMPORT_LIMIT_EXCEEDED
Data.com から購入を試みたレコードの数が、使用可能なレコード追加制限を上回っています。

LICENSE_LIMIT_EXCEEDED
組織に割り当てられたライセンス数の上限を超えています。

LIGHT_PORTAL_USER_EXCEPTION
許可されていないカスタマーポータルでアクションを実行しようとしました。たとえば、ケースチームへのユーザーの追加などです。

LIMIT_EXCEEDED
項目のサイズまたは値、ライセンス、プラットフォームイベントの公開、またはその他のコンポーネントの制限を超えました。

MALFORMED_ID
ID は 15 文字、または大文字と小文字を識別するための有効な拡張を含めて 18 文字でなければなりません。同じ名前の例外コードが存在します。

MANAGER_NOT_DEFINED
指定された承認プロセスにはマネージャーが定義されていません。

MASSMAIL_RETRY_LIMIT_EXCEEDED
組織の一括メール送信再実行の上限を超えたため、一括メール送信再実行が失敗しました。

MASS_MAIL_LIMIT_EXCEEDED
組織の一括メール送信の 1 日の上限を超えました。翌日になるまで一括メールメッセージは再送信できません。

MATCH_PRECONDITION_FAILED
eTag 検証に失敗しました。

MAXIMUM_CCEMAILS_EXCEEDED
ワークフローメールアラートで指定された CC アドレスが上限数を超えました。

MAXIMUM_DASHBOARD_COMPONENTS_EXCEEDED
ダッシュボードのドキュメントサイズの制限を超えました。

MAXIMUM_HIERARCHY_CHILDREN_REACHED
納入商品階層の子の最大数に達しました。

MAXIMUM_HIERARCHY_LEVELS_REACHED
階層の最大レベル数に達しました。

MAXIMUM_SIZE_OF_ATTACHMENT
添付ファイルのサイズの上限を超えました。

MAXIMUM_SIZE_OF_DOCUMENT
ドキュメントサイズの上限を超えました。

MAX_ACTIONS_PER_RULE_EXCEEDED
ルールあたりのアクション数の上限を超えました。

MAX_ACTIVE_RULES_EXCEEDED
有効なルール数の上限を超えました。

MAX_APPROVAL_STEPS_EXCEEDED
承認プロセスの承認ステップ数の上限を超えました。

MAX_DEPTH_IN_FLOW_EXECUTION
フロートリガーアクションの実行中にエラーが発生しました。

MAX_FORMULAS_PER_RULE_EXCEEDED
ルールあたりの数式の数の上限を超えました。

MAX_RULES_EXCEEDED
オブジェクトのルール数の上限を超えました。

MAX_RULE_ENTRIES_EXCEEDED
ルールのエントリ数の上限を超えました。

MAX_TASK_DESCRIPTION_EXCEEEDED
ToDo の説明が長すぎます。

MAX_TM_RULES_EXCEEDED
Territory あたりのルール数の上限を超えました。

MAX_TM_RULE_ITEMS_EXCEEDED
Territory のルールあたりのルール条件の数の上限を超えました。

MAX_TRIGGERS_EXCEEDED
ワークフロールールごとのトリガー数、または Salesforce 全体のトリガー数が上限を超えています。

MERGE_FAILED
マージ操作が失敗しました。

METHOD_NOT_ALLOWED
ターゲットリソースはこのメソッドをサポートしていません。

METADATA_FIELD_UPDATE_ERROR
指定されたメタデータ項目の更新中にエラーが発生しました。

MISMATCHING_TYPES
指定されたレコードまたはその関連レコードのタイプが同じではありません。

MISSING_ARGUMENT
必要な引数が指定されていません。

MISSING_OMNI_PROCESS_ID
[調査応答サマリーを取得] 呼び出し可能アクションは、有効な OmniScript ID を持たない評価オブジェクトを呼び出します。

NONUNIQUE_SHIPPING_ADDRESS
減額注文品目の納入先住所が元の注文の納入先住所と異なる場合、その減額注文品目を挿入することはできません。

NO_APPLICABLE_PROCESS
送信されたレコードが、ユーザーが権限のあるどの有効承認プロセスのエントリ条件も満たさないため、`process()` 要求が失敗しました。

NO_ATTACHMENT_PERMISSION
組織ではメールへの添付ファイルが許可されていません。

NO_INACTIVE_DIVISION_MEMBERS
無効なディビジョンにメンバーを追加することはできません。

NO_MASS_MAIL_PERMISSION
メールを送信する権限が付与されていません。一括メール送信には「一括メール送信」権限、個別メール送信には「メールの送信」権限が必要です。

NOT_FOUND
このメソッドで参照されているリソースが見つかりません。

NO_PARTNER_PERMISSION
組織に ISV パートナー機能がありません。

NUMBER_OUTSIDE_VALID_RANGE
指定された数値が、値の有効な範囲外です。

NUM_HISTORY_FIELDS_BY_SOBJECT_EXCEEDED
sObject に指定された履歴項目数が制限を超えています。

OP_WITH_INVALID_USER_TYPE_EXCEPTION
1 人以上のユーザーについて、試行した操作を実行できません。たとえば、大規模ポータルユーザーをグループに追加することはできません。

OPERATION_ENQUEUED
非同期プラットフォームイベント公開操作はキューに追加されます。プラットフォームイベントは、システムリソースが使用可能になると公開されます。

OPTED_OUT_OF_MASS_MAIL
指定された User が一括メール送信を除外したため、メールを送信できません。

ORDER_MANAGEMENT_ACTION_NOT_ALLOWED
要求されたアクションは許可されていません。

ORDER_MANAGEMENT_INVALID_RECORD
レコードが無効です。

ORDER_MANAGEMENT_RECORD_EXISTS
現在、レコードは存在しているため作成できません。

ORDER_MANAGEMENT_RECORD_NOT_FOUND
要求されたレコードが見つかりませんでした。

PA_API_EXCEPTION
システムで内部エラーが発生しました。この問題を Salesforce に報告してください。

PA_AXIS_FAULT
システムで内部エラーが発生しました。この問題を Salesforce に報告してください。

PA_INVALID_ID_EXCEPTION
プラットフォームアクションに指定された ID が無効です。

PA_NO_ACCESS_EXCEPTION
アクションプラットフォームにアクセスする権限がありません。

PA_NO_DATA_FOUND_EXCEPTION
要求された Platform Action が見つかりませんでした。

PA_URI_SYNTAX_EXCEPTION
プラットフォームアクションクエリの URI 構文が無効です。

PA_VISIBLE_ACTIONS_FILTER_ORDERING_EXCEPTION
システムで内部エラーが発生しました。この問題を Salesforce に報告してください。

PACKAGE_LICENSE_REQUIRED
パッケージのライセンスがない場合、ログインユーザーはライセンスパッケージに含まれているオブジェクトにアクセスできません。

PAL_INVALID_ASSISTANT_RECOMMENDATION_TYPE_ID
アシスタントのおすすめに指定された種別 ID が無効です。

PAL_INVALID_ENTITY_ID
プラットフォームアクションリストに指定されたオブジェクト ID が無効です。

PAL_INVALID_FLEXIPAGE_ID
プラットフォームアクションリストに指定されたページ ID が無効です。

PAL_INVALID_LAYOUT_ID
プラットフォームアクションリストに指定されたレイアウト ID が無効です。

PAL_INVALID_PARAMETERS
プラットフォームアクションリストに指定されたパラメーターが無効です。

PALI_INVALID_ACTION_ID
プラットフォームアクションリストに指定されたアクション ID が無効です。

PALI_INVALID_ACTION_NAME
プラットフォームアクションリストに指定されたアクション名が無効です。

PALI_INVALID_ACTION_TYPE
プラットフォームアクションリストのコンテキストに指定されたアクション種別が無効です。

PARTICIPANT_RELATIONSHIP_EXISTS
ユーザーが ParticipantGroup を削除しようとしましたが、あるレコードが準拠データ共有を使用してこのグループに共有されていました。グループは、準拠データ共有を介して 1 つ以上の親レコードに関係者として割り当てられています。関係者レコードを削除して、もう一度お試しください。

PLATFORM_EVENT_ENCRYPTION_ERROR
暗号化の問題が原因で、プラットフォームイベントメッセージを公開できませんでした。この問題は、Salesforce 組織の設定の誤り、または暗号化サービスの一般的なエラーが原因で発生する場合があります。

PLATFORM_EVENT_PUBLISHING_UNAVAILABLE
サービスが一時的に使用できなくなっているため、プラットフォームイベントメッセージの公開に失敗しました。しばらくしてからもう一度お試しください。

PLATFORM_EVENT_PUBLISH_FAILED
1 回以上の試行を行いましたが、システムエラーが原因でプラットフォームイベントメッセージを公開できませんでした。しばらくしてからもう一度お試しください。

PORTAL_USER_ALREADY_EXISTS_FOR_CONTACT
Contact の下に 2 つ目のポータルユーザーは作成できないため、User 作成操作が失敗しました。

PORTAL_USER_CREATION_RESTRICTED_WITH_ENCRYPTION
「暗号化されたデータの参照」権限がないユーザーが暗号化した標準項目が取引先責任者レコードに含まれている場合、そのレコードからポータルユーザーを作成することはできません。

PRIVATE_CONTACT_ON_ASSET
納入商品に非公開の取引先責任者は設定できません。

PROCESSING_HALTED
関連する操作が失敗し、トランザクションがロールバックされます。このエラーは、Composite 要求でいずれかのサブ要求が失敗すると発生することがあります。

QA_INVALID_CREATE_FEED_ITEM
フィード項目を作成できません。作成できるのは「作成」および「活動の記録」のフィード項目のみです。

QA_INVALID_SUCCESS_MESSAGE
成功メッセージが無効です。update では成功メッセージが含まれる場合がありますが、フィード項目は作成されません。他のすべてのタイプでは成功メッセージが空で、フィード項目は作成されません。

QUICK_ACTION_LIST_ITEM_NOT_ALLOWED
クイックアクション種別 Lightning コンポーネントでは、クイックアクションリスト項目を保存できません。

QUICK_ACTION_LIST_NOT_ALLOWED
クイックアクションリストは、クイックアクションを許可するオブジェクトでのみ保存できます。

RECORD_IN_USE_BY_WORKFLOW
ワークフロープロセスまたは承認プロセスによって使用されているレコードにはアクセスできません。

RELATED_ENTITY_FILTER_VALIDATION_EXCEPTION
キャンペーンの範囲が設定されていない場合は、キャンペーンメンバーの状況で絞り込まれたリストビューを保存できません。

REL_FIELD_BAD_ACCESSIBILITY
カスタムメタデータ型は、公開型で参照されているときは保護に設定できません。また、保護型を参照しているときは公開に設定できません。

REPUTATION_MINIMUM_NUMBER_NOT_REACHED
Experience Cloud サイトの評価レベルの下限しきい値を下回っています。

REQUEST_RUNNING_TOO_LONG
処理時間が長すぎる要求はキャンセルされます。

REQUIRE_CONNECTED_APP_SCS
ソーシャルカスタマーサービスの接続アプリケーションがインストールされていません。

REQUIRE_CONNECTED_APP_SESSION_SCS
ソーシャル接続サービスのコールには、有効な認証済み接続アプリケーションセッションが必要です。

REQUIRE_RUNAS_USER
[ユーザーとして実行] は、組織の [受信設定] タブの [Social Customer Setup (ソーシャルカスタマー設定)] ページで設定する必要があります。

REQUIRED_FIELD_MISSING
このコールには項目が必要ですが指定されていません。

RETRIEVE_EXCHANGE_ATTACHMENT_FAILED
Exchange サービスアカウントから添付ファイルをアップロードまたはダウンロードできませんでした。

RETRIEVE_EXCHANGE_EMAIL_FAILED
Exchange サービスアカウントからメールを取得できません。

RETRIEVE_EXCHANGE_EVENT_FAILED
Exchange サービスイベントが失敗しました。

ROUTES_EVALUATION_LIMIT_EXCEEDED
ルート評価の上限に達したため、これ以上のルートは評価できませんでした。

SELF_REFERENCE_FROM_TRIGGER
同じオブジェクトを Apex トリガーで繰り返し更新または削除できません。このエラーは、次のような場合に多く発生します。

- オブジェクトを before トリガーで更新または削除しようとした場合。
- オブジェクトを after トリガーで削除しようとした場合。

このエラーは直接操作または間接操作のいずれでも発生します。間接操作の例を次に示します。

- オブジェクト A を更新する要求が送信されました。
- オブジェクト A の `before update` トリガーがオブジェクト B を作成します。
- オブジェクト A が更新されます。
- オブジェクト B の `after insert` トリガーがオブジェクト A にクエリを実行し、更新します。これはオブジェクト A の before トリガーとなり、オブジェクト A の間接更新となるためエラーが生成されます。

SHARE_NEEDED_FOR_CHILD_OWNER
親レコードに共有ルールを必要とする子レコードがある場合、親レコードの共有ルールを削除できません。

SINGLE_EMAIL_LIMIT_EXCEEDED
(API バージョン 18.0 以降) 組織の個別メール送信の 1 日の上限を超えました。翌日になるまで個別メールメッセージは送信できません。

SLACK_API_ERROR
予期しない状況が発生したため、Slack API は要求を完了できませんでした。この値は API バージョン 36.0 以降で使用できます。

SOCIAL_ACCOUNT_NOT_FOUND
ソーシャルアカウントへのアクセス権がないか、または指定された ID を持つ管理対象のソーシャルアカウントが見つかりませんでした。

SOCIAL_POST_INVALID
指定された操作では、指定されたソーシャル投稿を適用できません。たとえば、必須項目が欠落していることや、状況が適切でないことが考えられます。

SOCIAL_POST_NOT_FOUND
投稿へのアクセス権がないか、指定された ID の投稿が見つかりませんでした。

STANDARD_PRICE_NOT_DEFINED
対応する標準価格がなければ、カスタム価格を定義することはできません。

STORAGE_LIMIT_EXCEEDED
組織のストレージ容量の制限を超えています。

STRING_TOO_LONG
指定された文字列は文字列長の上限を超えています。

TABSET_LIMIT_EXCEEDED
タブセットに許可されたタブ数を超えています。

TEMPLATE_NOT_ACTIVE
指定されたテンプレートが利用できません。別のテンプレートを指定するか、指定したテンプレートが利用できるようにします。

TERRITORY_REALIGN_IN_PROGRESS
テリトリーの再配置が進行中のため、この処理を実行できません。

TEXT_DATA_OUTSIDE_SUPPORTED_CHARSET
指定されたテキストがサポートされていない文字コードを使用しています。

TEXT_TO_PICKLIST_VALUES_EXCEEDED
指定されたテキスト内で選択リストの値の数が上限を超えています。

TOO_MANY_APEX_REQUESTS
送信された Apex 要求の数が多すぎます。これは一時的なエラーです。少ししてから要求を再送信してください。

TOO_MANY_ENUM_VALUE
複数選択リストに渡された値が多すぎるため、要求が失敗しました。複数選択リストでは 100 個の値まで選択できます。

TRANSFER_REQUIRES_READ
ユーザーに参照権限がないため、指定された User にレコードを割り当てることができません。

UISF_ENTITY_QUERY_FAILED
試みている変更を完了できません。次のような理由が考えられます。

- レジストリで Slack アプリケーションが見つからない
- ユーザーまたはボットの認証が見つからない
- Slack 設定エンティティを照会または更新/挿入するときに例外が発生した
- ユーザーまたはボットのトークンが null である

UISF_NO_MAPPINGS_FOUND
Salesforce ユーザー ID に対応する Slack ユーザーが見つかりませんでした。

UISF_TOKEN_NOT_FOUND
Slack アプリケーションのトークンが見つかりませんでした。

UISF_UNKNOWN_EXCEPTION
システムで内部エラーが発生しました。この問題を Salesforce に報告してください。

UISF_USER_MAPPING_FAILED
Slack ユーザーを Salesforce ユーザー ID に対応付けることができませんでした。

UNAVAILABLE_REF
カスタムメタデータレコードが使用不可のオブジェクトまたは項目を参照している場合は、インストールできません。

UNABLE_TO_LOCK_ROW
デッドロックまたはタイムアウト条件が検出されました。

- デッドロックには、重なり合うオブジェクトセットを更新しようとする最低 2つのトランザクションが関係しています。トランザクションに集計項目が含まれている場合、親オブジェクトがロックされるため、このようなトランザクションでは特にデッドロックが発生しやすくなります。デバッグするには、コード内のデッドロックを確認し修正してください。通常、デッドロックは Salesforce の操作の問題が原因ではありません。
- タイムアウトは、選択リストの値の置き換えや、カスタム項目定義の変更など、完了までに時間がかかりすぎるトランザクションで発生します。タイムアウト状態は一時的です。修正アクションは不要です。

バッチに含まれるオブジェクトをロックできない場合、バッチ全体でこのエラーが発生し失敗します。この状況コードのエラーでは、エラーメッセージにロックできないレコードの ID が含まれます (使用可能な場合)。

UNAVAILABLE_RECORDTYPE_EXCEPTION
適切なデフォルトのレコードタイプが見つかりませんでした。

UNDELETE_FAILED
オブジェクトが存在しないか、削除されていないため、このオブジェクトを復元できません。

UNKNOWN_EXCEPTION
システムで内部エラーが発生しました。この問題を Salesforce に報告してください。

> メモ: `sendEmail()` コールが原因の場合は、この例外コードを Salesforce には報告しないでください。[メール送信除外]のオプションが選択されている 1 人以上の受信者にメールを送信するのに `sendEmail()` コールを使用した場合、この例外コードが返されます。

UNQUALIFIED_CART
カートの内容が、クーポンに関連付けられているプロモーションの適格条件を満たしていません。

UNSPECIFIED_EMAIL_ADDRESS
指定されたユーザーにはメールアドレスがありません。

UNSUPPORTED_APEX_TRIGGER_OPERATON
Apex トリガーでは定期的な行動を保存できません。

UNSUPPORTED_DML
トランザクション種別が混在しているため、DML 操作はサポートされません。

UNSUPPORTED_MODE
このモードは使用できません。サポートされいているモードを使用してください。

UNSUPPORTED_PAYMENT_REQUEST_TYPE
Salesforce Payments は、この支払い要求種別をサポートしていません。

UNSUPPORTED_QUERY
このクエリでは、Big Objects ではサポートされていない GROUP BY、NULLS FIRST/LAST、または他のクエリ機能が使用されています。

UNVERIFIED_SENDER_ADDRESS
`sendEmail()` コールが、OrgWideEmailAddress オブジェクトで定義された未検証のメールアドレスを使用しようとしました。

USER_WITHOUT_WEM_PERMISSION
ユーザーにワークフォースエンゲージメント権限がありません。

VARIANT_NOT_FOUND
要求されたバリエーションが見つかりませんでした。

WEBLINK_SIZE_LIMIT_EXCEEDED
WebLink URL または JavaScript コードのサイズが制限を超えています。

WEBLINK_URL_INVALID
WebLink URL で URL 文字列の入力規則チェックに失敗しました。

WEM_USER_NOT_ORG_ADMIN
ユーザーは、ワークフォースエンゲージメント権限を所有していますが、管理者ではありません。

WORKSPACE_NOT_FOUND
要求されたワークスペースが見つかりませんでした。

WRONG_CONTROLLER_TYPE
Visualforce メールテンプレートのコントローラータイプが、使用されているオブジェクト種別と一致しません。

ここで取り上げられていない状況コードが返された場合、カスタマーサポートにご連絡ください。

ExtendedErrorDetails

`ExtendedErrorDetails` 要素には、エラーに関する追加の情報が含まれます。`ExtendedErrorDetails` 要素には次のプロパティを含めることができます。

名前	型	説明
extendedErrorCode	ExtendedErrorCode	拡張エラーの詳細を特徴付けるコード。
拡張エラープロパティ	不特定	エラーに関する詳細が含まれる拡張エラーのプロパティ。プロパティの名前と値は、拡張エラーコードによって異なります。たとえば、制限に関するエラーが発生した場合、プロパティの `limit` には制限の値が入っており、`input` には送信された `limit` 以上の値が入っています。

ExtendedErrorCode

各 `ExtendedErrorDetails` には、エラーコードとそれに関連するプロパティが含まれています。ここに示されているコードと説明は、拡張エラーの例です。発生したエラーによっては他の拡張エラーコードが表示される場合があります。

FLOW_CANNOT_BE_REACTIVATED
現在のフローバージョンがキャンセルされているか、完了している場合、ユーザーが同じバージョンを再有効化することはできません。新しいバージョンとしてフローを保存するしかありません。

FLOW_NOT_FOUND
フローを使用できません。この値は API バージョン 39.0 以降で使用できます。

FLOW_TEST_CONDITION_INVALID_DATATYPE_MAPPING
フローの開始または終了のテストポイントに、互換性のないデータ型の条件が含まれています。

FLOW_TEST_INVALID_LHS_REFERENCE
フローテスト条件で無効な参照が見つかりました。

FLOW_TEST_NOT_SUPPORTED
フローテストがプロセス種別でサポートされていません。

FLOW_TEST_PARAMETER_LEFTVALUEREFERENCE_INVALID
フロー開始のフローテストメタデータでは、leftValueReference を ScheduledPathApiName に設定する必要があります。

FLOW_TEST_PARAMETER_TYPE_INVALID
スケジュール済みパスは有効ですが、パスのタイプが ScheduledPath ではありません。

FLOW_TEST_PARAMETER_VALUE_INVALID
フロー開始のフローテストメタデータが、スケジュール済みパスの無効な値を参照しています。

FLOW_TEST_PARAMETER_VALUE_MISSING
フロー開始のフローテストメタデータにスケジュール済みパスの API 参照名がありません。

FLOW_TEST_PARAMS_REQUIRED
フローの開始または終了のテストポイントにパラメーターがありません。

FORM_ALREADY_IN_USE_BY_DRAFT_VERSION
選択したフォームは、すでにドラフトフローバージョンに関連付けられています。1 つのフォームを複数のドラフトフローバージョンに関連付けることはできません。この値は API バージョン 60.0 以降で使用できます。

FORM_ALREADY_IN_USE_BY_FLOW
選択したフォームは、すでにフローに関連付けられています。1 つのフォームを複数のフローに関連付けることはできません。この値は API バージョン 60.0 以降で使用できます。

INVALID_QUERY_LOCATOR_FORMAT
クエリロケーターの形式が無効であるか、正しくありません。

INVALID_SEGMENT_STATUS_FOR_ACTIVATION
使用中のセグメントは、フローの有効化が許可されない無効またはエラー状況にあるため、フローを有効化できません。

LOCATOR_LOCATION_EXCEEDS_SIZE
クエリロケーターで指定された場所が、元のクエリで返されたレコードの数を超えています。

MAX_STATEMENT_SIZE
クエリが文字制限を超えています。「SOQL SELECT Syntax」を参照してください。

MAX_XDS_IMPLICIT_SUBQUERIES
外部オブジェクトで許可されている結合数の上限を超えています。「リレーションクエリ制限について」を参照してください。

QUERY_LOCATOR_EXPIRED
クエリロケーターの有効期限が切れています。

QUERY_LOCATOR_NOT_FOUND
クエリロケーターが存在しないか、またはクエリを実行しているユーザーがカーソルを作成したユーザーと同じではありません。

SCREENFIELD_OBJECTPROVIDED_INVALID_DATATYPE
FlowSceenField の fieldType が objectProvided に設定されている場合、objectFieldReference はサポートされていないデータ型に設定されます。

SURVEY_INVALID_MATRIX_QUESTION_CONFIGURATION
マトリックス形式の質問の設定が無効であるか、またはユーザーにマトリックス形式の質問を作成する権限がありません。

TEMPORARY_QUERY_MORE_FAILURE
現在、追加のクエリ結果を取得することはできません。しばらくしてからもう一度お試しください。

UNAUTHORIZED_USER_FOR_CURSOR
コールを実行するユーザーは、追加のクエリ結果にアクセスできません。元のクエリを実行したユーザーのみが、追加の結果にアクセスできます。

重複管理データ型

DuplicateError

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。

項目

項目	詳細
`duplicateResult`	**型** DuplicateResult **説明** 重複ルールの詳細と、重複ルールで検出された重複レコード。
`fields`	**型** string[] **説明** 1 つ以上の項目名の配列。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。
`message`	**型** string **説明** エラーメッセージテキスト。
`statusCode`	**型** StatusCode **説明** エラーを特徴付けるコード。statusCode の完全な一覧は組織の WSDL ファイルに記述されています (「組織の WSDL ファイルの生成」を参照してください)。

使用方法

重複ルールを使用する組織は、DuplicateError とその構成オブジェクトを使用できます。

DuplicateError は Error のデータ型です。

重複を処理するには、SaveResult の `errors` 項目ですべての Error オブジェクトをループ処理します。データ型が DuplicateError の Error オブジェクトには、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。重複に関する情報にアクセスするには、`duplicateResult` 項目を使用します。

Java のサンプル

次の例に、データ型が DuplicateError の saveResult にエラーがあるかどうかを確認する方法を示します。エラーがある場合は、重複が検出されます。重複するリードをユーザーが入力できないようにし、アラートおよび重複レコードのリストを表示する方法を示す完全なコードサンプルについては、「DuplicateResult」を参照してください。

```
if (!saveResult.isSuccess()) {
    for (Error e : saveResult.getErrors()) {
        if (e instanceof DuplicateError) {
            System.out.println("Duplicate(s) Detected for lead with ID: " +
leads[i].getId());
            System.out.println("ERROR MESSAGE: " + e.getMessage());
            System.out.println("STATUS CODE: " + e.getStatusCode());
            DuplicateResult dupeResult = ((DuplicateError)e).getDuplicateResult();
            System.out.println("Found the following duplicates...");
            for (MatchResult m : dupeResult.getMatchResults()) {
                if (m.isSuccess()) {
                    System.out.println("The match rule that was triggered was " +
m.getRule());
                    for (MatchRecord mr : m.getMatchRecords()) {
                       System.out.println("Your record matched " + mr.getRecord().getId()
 + " of type "
                           + mr.getRecord().getType());
                        System.out.println("The match confidence is " +
mr.getMatchConfidence());
                    }
                }
            }
        }
    }
}
```

DuplicateResult

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。

項目

項目	詳細
allowSave	**型** boolean **説明** 重複を保存できる場合は true。重複を保存できない場合は false。
duplicateRule	**型** string

項目	詳細
	説明 重複レコードを検出した重複ルールの名前。
duplicateRuleEntityType	**型** string **説明** 重複レコードを検出した重複ルールの名前。
errorMessage	**型** string **説明** 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ーに警告するために、システム管理者が設定したエラーメッセージ。このメッセージは重複ルールに関連付けられています。
matchResults	**型** MatchResult **説明** 重複レコードおよび関連する一致情報。

使用方法

重複ルールを使用する組織は、DuplicateResult とその構成要素オブジェクトを使用できます。

Java のサンプル

重複するリードのユーザーによる入力をブロックし、アラートと重複のリストを表示する方法の例を次に示します。

```
package com.doc.example;

import java.io.FileNotFoundException;

import com.sforce.soap.partner.*;
import com.sforce.soap.partner.Error;
import com.sforce.soap.partner.sobject.SObject;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public class SaveResultsWithDupeHeader {

    private PartnerConnection partnerConnection = null;
    static SaveResultsWithDupeHeader tester;
```

```
    public static void main(String[] args) {
        tester = new SaveResultsWithDupeHeader();
        try {
            tester.demoDuplicateRuleHeader();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /*
     * Make sure that you have an active lead duplicate rule linked to an active matching
 rule. This method tries to save
     * an array of leads and inspects whether any duplicates were detected
     */
   public void demoDuplicateRuleHeader() throws FileNotFoundException, ConnectionException
 {
        // Create the configuration for the partner connection
        ConnectorConfig config = new ConnectorConfig();
        config.setUsername("user@domain.com");
        config.setPassword("secret");
        config.setAuthEndpoint("authEndPoint");
        config.setTraceFile("traceLogs.txt");
        config.setTraceMessage(true);
        config.setPrettyPrintXml(true);

        // Initialize the connection
        partnerConnection = new PartnerConnection(config);

        // Get the leads that have to be saved
        SObject[] leads = getLeadsForInsertOrUpdate();

        /* Set a duplicate rule header to return a result if duplicates are detected
         * This sets the allowSave, includeRecordDetails, and runAsCurrentUser flags to
true
         */
        partnerConnection.setDuplicateRuleHeader(true, true, true);

        // Save the leads
        SaveResult[] saveResults = partnerConnection.create(leads);

        // Check the results to see if duplicates were detected
        for (int i = 0; i < leads.length; i++) {
            SaveResult saveResult = saveResults[i];
            if (!saveResult.isSuccess()) {
                for (Error e : saveResult.getErrors()) {
                // See if there are any errors on the saveResult with a data type of
DuplicateError
                    if (e instanceof DuplicateError) {
                        System.out.println("Duplicate(s) Detected for lead with ID: " +
leads[i].getId());
                        System.out.println("ERROR MESSAGE: " + e.getMessage());
                        System.out.println("STATUS CODE: " + e.getStatusCode());
                        DuplicateResult dupeResult =
((DuplicateError)e).getDuplicateResult();
```

```
                        System.out.println("Found the following duplicates...");
                        for (MatchResult m : dupeResult.getMatchResults()) {
                            if (m.isSuccess()) {
                                System.out.println("The match rule that was triggered was
 " + m.getRule());
                                for (MatchRecord mr : m.getMatchRecords()) {
                                    System.out.println("Your record matched " +
mr.getRecord().getId() + " of type "
                                            + mr.getRecord().getType());
                                    System.out.println("The match confidence is " +
mr.getMatchConfidence());
                                    for (FieldDiff f : mr.getFieldDiffs()) {
                                        System.out.println("For field " + f.getName() + "
 field difference is "
                                                + f.getDifference().name());
                                    }
                                }
                            }
                        }
                    }
                }
            }

        // Clear the duplicate rule header
        partnerConnection.clearDuplicateRuleHeader();
    }

    /**
     * The assumption here is that this method is retrieving leads from either UI, a data
 source, etc
     */
    private SObject[] getLeadsForInsertOrUpdate() {
        return new SObject[0];
    }

}
```

MatchResult

一致ルールの重複結果を表します。

MatchResult **オブジェクトには、次の項目があります。**

項目	詳細
errors	**型** Error[] **説明** **一致ルールの照合中に発生したエラー。**

項目	詳細
`entityType`	**型** string **説明** 一致ルールのエンティティ種別。
`matchEngine`	**型** string **説明** 一致ルールのマッチングエンジン。
`matchRecords`	**型** MatchRecord[] **説明** 一致ルールによって検出された重複に関する情報。
`rule`	**型** string **説明** 重複を検出した一致ルールの API 参照名。
`size`	**型** int **説明** 一致ルールによって検出された重複の数。
`success`	**型** boolean **説明** 一致ルールが正常に実行された場合は `true`。一致ルールでエラーが発生した場合は `false`。

使用方法

重複ルールを使用する組織は、MatchResult とその構成オブジェクトを使用できます。

重複を含む保存したレコードごとに 1 つの MatchResult があります。MatchResult には、保存したそのレコードのすべての重複が含まれます。

Java のサンプル

次の例に、リードの保存時に検出されたすべての重複の ID と種別を表示する方法を示します。重複するリードをユーザーが入力できないようにし、アラートおよび重複レコードのリストを表示する方法を示す完全なコードサンプルについては、「DuplicateResult」を参照してください。

```
for (MatchResult m : dupeResult.getMatchResults()) {
    if (m.isSuccess()) {
        System.out.println("The match rule that was triggered was " + m.getRule());
        for (MatchRecord mr : m.getMatchRecords()) {
            System.out.println("Your record matched " + mr.getRecord().getId() + " of type
 "
                + mr.getRecord().getType());
            System.out.println("The match confidence is " + mr.getMatchConfidence());
        }
    }
}
```

MatchRecord

一致ルールで検出された重複レコードを表します。

項目

項目	詳細
additionalInformation	**型** AdditionalInformationMap **説明** 一致レコードに関するその他の情報。
fieldDiffs	**型** FieldDiff[] **説明** 一致ルール項目と、重複とその一致レコードについての各項目値の比較方法。
matchConfidence	**型** double **説明** 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。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。
record	**型** sObject

項目	詳細
	説明 重複レコードの項目と項目値。

使用方法

重複ルールを使用する組織は、MatchRecord とその構成オブジェクトを使用できます。

各 MatchRecord は、レコードの保存時に検出された重複を表します。複数の重複が検出された場合は、保存された 1 つのレコードに対して複数の MatchRecord オブジェクトが存在することがあります。

Java のサンプル

次の例に、リードの保存時に検出されたすべての重複の ID と種別を表示する方法を示します。重複するリードをユーザーが入力できないようにし、アラートおよび重複レコードのリストを表示する方法を示す完全なコードサンプルについては、「DuplicateResult」を参照してください。

```
for (MatchRecord mr : m.getMatchRecords()) {
    System.out.println("Your record matched " + mr.getRecord().getId() + " of type "
        + mr.getRecord().getType());
    System.out.println("The match confidence is " + mr.getMatchConfidence());
}
```

FieldDiff

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。

項目

項目	詳細
`difference`	**型** differenceType **説明** 重複とその一致レコードについての一致ルール項目値の比較結果。 可能な値には、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。
`name`	**型** string

項目	詳細
	説明 重複が検出された一致ルールの項目の名前。

Java のサンプル

次の例に、重複が検出された場合の一致ルール項目の違いを表示する方法を示します。重複するリードをユーザーが入力できないようにし、アラートおよび重複レコードのリストを表示する方法を示す完全なコードサンプルについては、「DuplicateResult」を参照してください。

```
for (FieldDiff f : mr.getFieldDiffs()) {
    System.out.println("For field " + f.getName() + " field difference is "
        + f.getDifference().name());
}
```

AdditionalInformationMap

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)。

項目

項目	詳細
name	**型** string **説明** 要素の名前。
value	**型** string **説明** 要素の値。

第 5 章　Enterprise WSDL での Tooling API オブジェクト

Tooling API で使用される一部のオブジェクトは、Enterprise および Partner WSDL に含まれています。項目レベルのセキュリティが必要な場合は、Tooling WSDL ではなくこれらの WSDL のオブジェクトを使用してください。

次の Tooling API オブジェクトは、Enterprise および Partner WSDL で公開されます。

- BriefcaseDefinition
- DataType
- EntityDefinition
- EntityParticle
- FieldDefinition
- PicklistValueInfo
- Publisher
- SearchLayout
- Service
- ServiceDataType (将来の使用のために予約されています。)
- ServiceFieldDataType (バージョン 35.0 以降では使用できません。使用しないでください。)
- RelationshipDomain
- RelationshipInfo
- UserEntityAccess
- UserFieldAccess
- WsdlDataType (将来の使用のために予約されています。)
- XmlSchema (将来の使用のために予約されています。)

詳細は、『*Tooling API Developer's Guide*』を参照してください。

第 6 章　API コールの基礎

トピック:

API コールとは、クライアントアプリケーションがタスク実行のために実行時に起動できる特定の操作のことです。コールには次のようなものがあります。

- 組織内のデータのクエリ。
- データの追加、更新、削除。
- データのメタデータの取得。
- 管理タスク実行のためのユーティリティの起動。

開発環境を使用し、標準 Web サービスプロトコルを使用してプログラムで次の操作を実行する Web サービスクライアントアプリケーションを作成できます。

- ログインサーバーにログインし (`login()`)、後続のコールで使用する認証情報を受け取る
- 組織の情報に対するクエリを実行する (`query()`、`queryAll()`、`queryMore()`、および `retrieve()` コール)
- 組織の情報に対するテキスト検索を実行する (`search()` コール)
- データを作成、更新、および削除する (`create()`、`merge()`、`update()`、`upsert()`、`delete()`、および `undelete()` コール)
- ユーザー情報の取得 (`getUserInfo()` コール)、パスワードの変更 (`setPassword()` および `resetPassword()` コール)、システム時間の取得 (`getServerTimestamp()` コール) などの管理タスクを実行する
- データをローカルに複製する (`getDeleted()` および `getUpdated()` コール)
- 組織データについてのメタデータを取得およびアクセスする (`describeGlobal()`、`describeSObject()`、`describeSObjects()`、`describeLayout()`、および `describeTabs()` コール)
- 承認プロセスを使用する (`process()`)
- 組織からカテゴリグループとカテゴリを返す (`describeDataCategoryGroups()` および `describeDataCategoryGroupStructures()`)。

各コールについての完全な詳細は、「基本となる API コール」、「記述用の API コール (describe)」、および「ユーティリティ API コール」を参照してください。

API コールの特徴

すべての API コールには次の特徴があります。

- サービス要求とレスポンス — 使用しているクライアントアプリケーションはサービス要求を準備し、Lightning プラットフォーム Web サービスに API 経由で送信します。Lightning プラットフォーム Web サービスは要求を処理してレスポンスを返し、クライアントアプリケーションが応答を処理します。
- 同期 — API コールが起動されると、クライアントアプリケーションはサービスからレスポンスを受け取るまで待機します。非同期コールはサポートしていません。
- 自動コミットとエラー時のロールバック — デフォルトで、Salesforce オブジェクトに書き込みを行うすべての操作は自動的にコミットされます。これは、SQL の AUTOCOMMMIT 設定と類似しています。オブジェクトの複数のレコードへの書き込みを試みる `create()`、`update()`、および `delete()` コールでは、各レコードの書き込み操作はそれぞれ別のトランザクションとして扱われます。たとえば、クライアントアプリケーションが新しい取引先を 2 つ作成する場合、グループとしてではなく、完全に独立した挿入処理で作成され、それぞれ個別に成功または失敗と判断されます。

  API バージョン 20.0 以降では、AllOrNoneHeader ヘッダーが追加され、すべてのレコードが正常に処理されない限り、コールですべての変更をロールバックできるようになりました。このヘッダーは、`create()`、`delete()`、`undelete()`、`update()`、および `upsert()` コールでサポートされます。

メモ: デフォルト動作では、場合によってはクライアントアプリケーション側で失敗の処理が必要になります。たとえば、「発送」というカスタムオブジェクトのレコードを含んだ商談レコードを作成する場合、商談品目は作成されるのに発送レコードの作成には失敗することがあります。ビジネスルールで、すべての商談を発送と共に作成しなければならないと決められている場合、クライアントアプリケーションは商談の作成をロールバックする必要があります。AllOrNoneHeader を使用すると、この処理をきわめて簡単に行うことができます。

データアクセスに影響する要素

API を使用する際、次の要素は組織データへのアクセスに影響を与えます。

アクセス

組織で API アクセスが有効になっている必要があります。

オブジェクトは、Salesforce に連絡してアクセスを要求するまで使用できない場合があります。たとえば、エンタープライズテリトリー管理がアプリケーションで有効になっている場合にのみ Territory2 を表示できます。オブジェクトのアクセスに関する情報は、各オブジェクトの「使用方法」セクションに記述されます。

場合によっては、機能に関連するオブジェクトに API でアクセスできるようにするには、機能を 1 回は使用する必要があります。たとえば、`recordTypeIds` を使用できるのは、ユーザーインターフェースで組織に少なくとも 1 つのレコードタイプが組織に作成された後です。

データアクセスの問題を調査するには、最初に WSDL を調査します。

- **Enterprise WSDL**: 生成された Enterprise WSDL ファイルには、組織で利用可能なすべてのオブジェクトが含まれています。API を使用して、クライアントアプリケーションは Enterprise WSDL ファイルで定義されているオブジェクトにアクセスできます。

- **Partner WSDL**: 生成された Partner WSDL ファイルを使用する際、クライアントアプリケーションは `describeGlobal()` コールで返されたオブジェクトにアクセスできます。

オブジェクトレベルおよび項目レベルのセキュリティ

API はユーザーインターフェースで設定したオブジェクトレベルおよび項目レベルのセキュリティを優先します。ログインユーザーの権限およびアクセス設定で許可されている場合のみ、オブジェクトや項目にアクセスできます。たとえば、ユーザーに表示されない項目は、`query()` または `describeSObjects()` コールでも返されません。同様に、参照のみ項目は更新できません。

ユーザー権限

API へのアクセスを試みるユーザーは、「API 対応」権限が選択されている必要があります。デフォルトでは選択されています。

クライアントアプリケーションは、*ログインユーザー*と呼ばれるユーザーとしてログインします。ログインユーザーの権限によって、組織の特定のオブジェクトおよび項目への次のようなアクセスが許可または拒否されます。

- 参照 — ユーザーは、このタイプのオブジェクトを参照のみできます。
- 作成 — ユーザーは、このタイプのオブジェクトを参照し、作成できます。
- 編集 — ユーザーは、このタイプのオブジェクトを参照し、更新できます。
- 削除 — ユーザーは、このタイプのオブジェクトを参照し、編集、削除できます。

ユーザー権限は項目レベルのセキュリティには影響を与えません。項目レベルのセキュリティが、項目を非表示に指定している場合、そのオブジェクトへの「参照」権限のあるユーザーは、レコードの非表示でない項目のみを参照できます。さらに、オブジェクトへの「参照」権限のあるユーザーは共有設定が許可されているレコードのみを参照できます。唯一の例外は「参照のみ項目の編集」権限です。この権限は項目レベルのセキュリティにより、参照のみに設定されている項目の編集権限をユーザーに付与します。

共有

ほとんどの API コールでは、ログインユーザーの共有モデル外のデータは返されません。アプリケーションの場合と同様に、ユーザーには、組織のデフォルトと手動によるレコード共有の指定のうち、最も権限の大きいアクセス権が付与されます。

共有を無効にするユーザー権限

- すべて表示 — ユーザーは共有設定に関係なく、このオブジェクトに関連付けられたすべてのレコードを表示できます。
- すべて変更 — ユーザーは共有設定に関係なく、このオブジェクトに関連付けられたすべてのレコードの参照、編集、削除、移行、承認を実行できます。
- すべてのデータの編集 — ユーザーは共有設定に関係なく、すべてのレコードの参照、編集、削除、移行、承認を実行できます。この権限は、「すべて表示」および「すべて変更」と異なり、オブジェクトレベルの権限ではありません。

データのセキュリティを保護するには、ログインユーザーにはアプリケーションから行われたすべてのコールを正常に実行するために必要な権限だけを割り当てます。大きなインテグレーションアプリケーションでは、コールの応答時間を高速化するために「すべてのデータの編集」が必要な場合があります。大量のレコードを読み込む場合は、代わりに Bulk API 2.0 を使用します。

関連オブジェクト

オブジェクトによっては、他のオブジェクトに権限を依存する場合があります。たとえば、AccountTeamMember は、共有の権限は Account レコードなど、権限が割り当てられている関連オブジェクトに従います。同様に、Partner は関連する取引先の権限に依存します。

あるレコードの所有者を変更しても、他の関連レコードには自動的にカスケードされません。たとえば、Account の所有者が変更された場合、その Account に関連付けられた Contract の所有者が自動的に変更されることはありません。クライアントアプリケーションでそれぞれの所有者を別個に、明示的に変更する必要があります。

オブジェクトプロパティ

`create()` コールでオブジェクトを作成する場合、オブジェクトの `createable` 属性を `true` に設定する必要があります。オブジェクトで許可されている操作を判断するために、クライアントアプリケーションでオブジェクトに対して `describeSObjects()` コールを実行し、DescribeSObjectResult のプロパティを調べることができます。

メモ: `replicatable` の場合、`getUpdated()` および `getDeleted()` コールが許可されます。

ページレイアウトとレコードタイプ

ページレイアウトとレコードタイプについて Salesforce ユーザーインターフェースで定義した要件は、API では適用されません。

- 特定の項目が必須であるかどうかはページレイアウトで指定できますが、API は、そのようなレイアウト固有の項目制限や入力規則を `create()` および `update()` コールで適用しません。制限を適用するかどうかはクライアントアプリケーションが決定します。
- 指定されたレコードで選択する選択リスト値や、異なるプロファイルを持つユーザーが参照するページレイアウトは、レコード型で制御できます。ただし、ユーザーインターフェースで設定され、適用されるルールは API では適用されません。たとえば、API は、選択リスト項目の値がログインユーザーのプロファイルに関連付けられたレコードタイプ制限で許可されるかどうかを確認しません。同様に、ログインユーザーのプロファイルに関連付けられたレイアウトに項目がないという理由で、クライアントアプリケーションが特定の項目にデータを追加するのを API で拒否することはありません。

参照整合性

参照整合性を確保するために、API は特定の動作を適用または防止します。

- reference 項目の ID 値は `create()` および `update()` コールで検証されます。
- クライアントアプリケーションがレコードを削除した場合、その子の ChildRelationship の `cascadeDelete` プロパティの値が `true` であれば、その子もコールの一部として自動的に削除されます。たとえば、クライアントアプリケーションが Opportunity を削除すると、関連付けられた OpportunityLineItem レコードもまた削除されます。ただし、OpportunityLineItem が削除できない、または使用中の場合、親の Opportunity の削除は失敗します。 たとえば、クライアントアプリケーションが Invoice_Statement を削除すると、関連付けられた Line_Item レコードも削除されます。ただし、Line_Item が削除できない、または使用中の場合、親の Invoice_Statement の削除は失敗します。何が削除されるかを確認するには、DescribeSObjectResult を使用し、ChildRelationship 値を確認します。

  `cascadeDelete` の実行を阻止するいくつかの例外があります。たとえば、取引先にケースが関連付けられている場合、他のユーザーが所有する関連商談がある場合、または関連付けられた取引先責任者がカスタマーポータルで有効になっている場合、その取引先は削除できません。さらに、現在のユーザーによる成立した商談がある場合、または有効な契約のある場合、レコードの削除要求は失敗します。

パッケージバージョン設定

API クライアントが管理パッケージのコンポーネントを参照する場合、インテグレーションで参照するインストールする各パッケージのバージョンを指定できます。これにより、後続バージョンのパッケージをインストールしても、API クライアントは継続して、特定で、既知の動作によって機能することができます。PackageVersionHeader SOAP ヘッダーを使用して、必要に応じてさまざまなコールにそれぞれのパッケージバージョンを設定できます。

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、毎回の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。公開者は、パッケージバージョンを使用して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適切にアップグレードできます。そのパッケージを使用する既存の顧客のインテグレーションに影響を与えることもありません。

パッケージバージョンがAPIコールで指定されていない場合、APIコールのデフォルトのパッケージバージョンによって代替システム設定が指定されます。多くの API クライアントにはパッケージのバージョン情報がないため、デフォルトの設定がこれらのクライアントの既存の動作を保持します。

Enterprise API コールおよび Partner API コールにデフォルトのパッケージバージョンを指定できます。Enterprise WSDL は、Salesforce組織のみとのインテグレーションを構築する顧客を対象とします。非常に強い型付けで、`int` や `string` など特定のデータ型のオブジェクトや項目を指定して呼び出します。Partner WSDL は、カスタムオブジェクトやカスタム項目に関係なく、複数の Salesforce 組織にわたって機能するインテグレーションを構築する顧客、パートナー、および ISV を対象とします。あまり強い型付けでなく、特定のデータ型に依存せずに項目名と値の名前-値ペアで呼び出します。

Enterprise WSDL では、既存のクライアントの処理を維持するために、特定のパッケージバージョンと関連付ける必要があります。Enterprise WSDL または Partner WSDL のいずれかを使用してクライアントアプリケーションから API コールのパッケージバージョンのバインド設定を行うことができます。Enterprise WSDL に基づき、クライアントアプリケーションから発行される API コールのパッケージバージョン情報は、次のうち、最初に一致する設定によって決定されます。

1. PackageVersionHeader SOAP ヘッダー。
2. *serverName*`/services/Soap/c/`*api_version*`/`*ID* の形式で URL が指定されている SOAP エンドポイント。*api_version* は、60.0 など API のバージョンを示し、*ID* は、Enterprise WSDL の生成時にパッケージバージョンの選択内容を符号化したものです。
3. Enterprise パッケージバージョンのデフォルト設定です。

Partner WSDL より柔軟で、複数の組織とのインテグレーションに使用されます。デフォルトのパートナーパッケージバージョンを設定時、パッケージバージョンに[指定なし]オプションを選択すると、最新バージョンのインストールパッケージによって、処理が定義されます。つまり、パッケージがアップグレードし、その変更がすぐにインテグレーションに影響を与える場合、Apex トリガーなどパッケージコンポーネントの処理が異なる場合があります。パッケージバージョンの後続のインストールによって、既存のインテグレーションに影響のないよう、登録ユーザーは、クライアントアプリケーションから、すべての Partner API コールのインストールパッケージの特定バージョンを選択する必要があります。

Partner API コールのパッケージバージョン情報は、次のうち、最初に一致する設定によって決定されます。

1. PackageVersionHeader SOAP ヘッダー。

2. Visualforce ページのパッケージバージョン設定を使用する Visualforce ページからの API コール。
3. パートナーパッケージバージョンのデフォルト設定。

API コールのデフォルトパッケージバージョンを設定する手順は、次のとおりです。

1. [設定] から、［クイック検索］ボックスに「*API*」と入力し、**[API]** を選択します。
2. **[Enterprise パッケージバージョンの設定]** または **[パートナーパッケージバージョンの設定]** をクリックします。これらのリンクは、組織に管理パッケージを少なくとも 1 つインストールしている場合にのみ使用できます。
3. インストールした管理パッケージの［パッケージバージョン］を選択します。選択するパッケージバージョンがわからない場合、デフォルトの選択内容に従います。
4. **[保存]** をクリックします。

メモ: 組織に新しいバージョンのパッケージをインストールしても、現在のデフォルト設定に影響はありません。

第 7 章　エラー処理

トピック:

API コールは、クライアントアプリケーションがランタイムエラーを識別し解決するために使用できるエラーデータを返します。

ほとんどの API コールでは、エラーが発生すると、API によって次のようなエラー処理が行われます。

- 不正に作成されたメッセージ、失敗した認証、または同様の問題によるエラーについて、API は、関連する ExceptionCode を使用して SOAP エラーメッセージを返します。
- 多くのコールについて、クエリ特有の問題によってエラーが発生する場合、API は、Error を返します。たとえば、`create()` に 200 を超えるオブジェクトが含まれる場合などです。

セッション終了のエラー処理

login() コールでサインオンする場合、新しいクライアントセッションが開始し、対応する一意のセッション ID が生成されます。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。API コールを行うと、非活動状態タイマーがゼロにリセットされます。

セッションが終了すると、例外コード INVALID_SESSION_ID が返されます。このコードが返された場合は、login() コールを再度呼び出す必要があります。

エラー処理の詳細

エラーについての詳細は、次のトピックを参照してください。

- API 障害要素
- ExceptionCode
- Error

第 8 章　セキュリティと API

トピック:

Salesforce データにアクセスするクライアントアプリケーションは、Salesforce ユーザーインターフェースで使用されるものと同じセキュリティ保護を受けます。API を使用して Salesforce にアクセスするコンポーネントが含まれている AppExchange 管理パッケージをインストールした組織は、追加の保護を使用できます。

ユーザー認証

クライアントアプリケーションは、有効な資格情報を使用してログインする必要があります。検証後、サーバーによってクライアントアプリケーションに次の情報が提供されます。

- Web サービスに対するすべての後続コールが認証されるよう、セッションヘッダーに設定する `sessionId`
- クライアントアプリケーションの Web サービス要求の URL アドレス (`serverUrl`)

Salesforce は、Transport Layer Security (TLS) プロトコルおよび frontdoor.jsp のみをサポートします。暗号鍵の長さは、128 ビット以上でなければなりません。

ユーザープロファイルおよび権限セット構成

組織の Salesforce システム管理者は、プロファイルおよび権限セットを設定し、ユーザーをそれらに割り当てることによって、ユーザーが使用できる機能やビューを制御します。API にアクセスし、コールを発行してコール結果を受信するには、ユーザーに「API 対応」権限が与えられている必要があります。クライアントアプリケーションは、ログインユーザーの権限を介してアクセス権限を付与されるこれらのオブジェクトおよび項目のみを照会したり更新したりできます。

プロファイルを作成、編集、または削除するには、[設定]から、[クイック検索]ボックスに「*プロファイル*」と入力し、[プロファイル]を選択します。権限セットを作成、編集、または削除するには、[設定]から、[クイック検索] ボックスに「*権限セット*」と入力し、[権限セット] を選択します。

メモ: Web サービス WSDL ファイルは、組織の利用可能なすべてのオブジェクトおよび項目を返します。

セキュリティトークン

ユーザーがユーザーインターフェース、API、または Salesforce for Outlook、Connect Offline、Connect for Office、データローダーなどのデスクトップクライアントを使用して Salesforce にログインする場合、Salesforce は、ログインを次の方法で承認します。

1. Salesforce は、ユーザーのプロファイルにログイン時間の制限が設定されているかどうかを確認します。ユーザーのプロファイルにログイン時間の制限が指定されている場合、それ以外の時間にログインを試みようとすると、拒否されます。
2. ユーザーに「ユーザーインターフェースログインの多要素認証」権限がある場合は、Salesforce ログインプロセスで、ユーザー名とパスワード以外の ID 検証方法も用意するようユーザーに促します。ユーザーのアカウントが Salesforce Authenticator モバイルアプリケーションなどの検証方法にまだ接続されていない場合は、Salesforce では、方法を登録するようユーザーに促します。
3. ユーザーに「API ログインの多要素認証」権限があり、認証アプリケーションがアカウントに接続されている場合、ユーザーは認証アプリケーションで生成される確認コード (TOTP) を入力する必要があります。ユーザーが標準のセキュリティトークンを使用している場合、Salesforce はエラーを返します。
4. Salesforce は次に、ユーザーのプロファイルに IP アドレス範囲の制限が定義されているかどうかを確認します。定義されている場合、その IP アドレス範囲外からのログインは拒否されます。[すべての要求でログイン IP アドレスの制限を適用]セッション設定が有効になっている場合、クライアントアプリケーションからの要求も含め、ページ要求ごとに IP アドレス制限が適用されます。

5. プロファイルベースの IP アドレス制限が設定されていない場合、Salesforce は、ユーザーのログイン元のデバイスが、以前 Salesforce へのアクセスに使用されたかどうかを確認します。
   - Salesforceが認識するデバイスやブラウザーからユーザーがログインしている場合は、ログインが許可されます。
   - 信頼できるIPアドレスのリストに含まれるIPアドレスからのログインであれば、ログインは許可されます。
   - Salesforce で認識された信頼できるIPアドレス、デバイス、またはブラウザーからユーザーがログインしていない場合、ログインはブロックされます。

ログインがブロックされるか、API ログインの失敗エラーが返された場合、Salesforce は、ユーザーの ID を検証します。

- ユーザーインターフェース経由でアクセスする場合、ユーザーは、Salesforce Authenticator (バージョン 2 以降) を使用して検証するか、確認コードを入力するように求められます。

  メモ: ユーザーが Salesforce に初めてログインするときは、確認コードを要求されません。

- API またはクライアントアプリケーション経由でアクセスする場合、ユーザープロファイルに「API ログインの多要素認証」権限が設定されているときは、認証アプリケーションで生成された TOTP 確認コードをユーザーが入力します。

  この権限が設定されていない場合、ユーザーはログインパスワードの末尾にセキュリティトークンを追加する必要があります。セキュリティトークンは Salesforce から生成されるキーです。たとえば、パスワードが *mypassword* で、セキュリティトークンが *XXXXXXXXXX* の場合は、ログイン時に「*mypasswordXXXXXXXXXX*」と入力します。また、クライアントアプリケーションによっては、別個にセキュリティトークン用の項目があります。

  セキュリティトークンを取得するには、Salesforce ユーザーインターフェースを通じてパスワードを変更するか、セキュリティトークンをリセットします。ユーザーがパスワードを変更するか、セキュリティトークンをリセットすると、Salesforceがユーザーの Salesforce レコードのメールアドレス宛に新しいセキュリティトークンを送信します。セキュリティトークンは、ユーザーがセキュリティトークンをリセットするか、パスワードを変更するか、またはパスワードがリセットされるまで有効です。

  メモ: 新しい IP アドレスから Salesforce にアクセスする前に、[私のセキュリティトークンのリセット] を使用して信頼できるネットワークからセキュリティトークンを取得しておくことをお勧めします。

トークンの詳細は、Salesforce 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。

ユーザーのパスワードが変更されると、そのユーザーのセキュリティトークンは自動的にリセットされます。リセット後にログインするには、ユーザーが、生成されたセキュリティトークンをパスワードの末尾に追加する必要があります。または、ユーザーのIPアドレスを組織の信頼できるIPアドレスのリストに追加した後で、新しいパスワードを入力します。

「API 限定ユーザー」権限があるユーザーのパスワードをリセットしても、そのユーザーはセキュリティトークンのリセットメールを受信しません。API限定ユーザーのパスワードをリセットしたときにAPI限定ユーザーにセキュリティトークンのリセットメールを送信するには、次の手順を実行します。

1. 「API限定ユーザー」権限のないプロファイルに一時的にユーザーを割り当てます。ユーザープロファイルと権限についての詳細は、「ユーザー権限およびアクセス」を参照してください。
2. セキュリティトークンを手動でリセットするようにユーザーに要求します。

3. 「API 限定ユーザー」権限のあるプロファイルにユーザーを再割り当てします。

シングルサインオン (SSO) が有効になっており、プロファイルに IP アドレス制限が設定されている場合、API またはデスクトップクライアントにアクセスするユーザーは、IP アドレスが組織の信頼できる IP アドレスのリストまたはプロファイルに記載されていない限りログインできません。代理認証機関は通常、「シングルサインオンを使用する」権限を持つユーザーのログインロックアウトポリシーを処理します。ただし、セキュリティトークンを使用できる場合は、ログインロックアウト設定によって、ロックアウトされるまでの無効なセキュリティトークンを使ってログイン試行できる回数が規定されます。詳細は、Salesforce ヘルプの「ログイン制限の設定」と「パスワードポリシーの設定」を参照してください。

共有

共有とは、あるユーザーやグループに対する参照、更新のアクセス権限がデフォルトのアクセスレベルによって付与されていない場合に、他のユーザーが、所有するレコードを読み書きできる該当のアクセス権限を付与するアクティビティのことを指します。すべての API コールは共有モデルの影響を受けます。

次の表は、アクセスレベルの種類について説明しています。

API 値	Salesforce ユーザーインターフェースの表示ラベル	API 選択リストの表示ラベル	説明
None	非公開	非公開	レコードの所有者と階層内でそのロールの上位にある User のみが、レコードの参照と編集を実行できます。
Read	参照のみ	参照のみ	すべての User および Group はレコードを参照できますが、編集はできません。レコードの所有者と階層内でそのロールの上位にあるユーザーのみが、そのレコードを編集できます。
Edit	参照/更新	参照/更新	すべての User および Group はレコードの参照と編集ができます。
ReadEditTransfer	参照/更新/所有権の移行	参照/更新/所有権の移行	すべての User および Group はレコードの参照、編集、削除、移行ができます。(組織の共有設定では、ケースとリードにのみ使用できます)。
All	フルアクセス	所有者	すべての User および Group はレコードの参照、編集、移行、削除、共有ができます。(組織の共有設定では、キャンペーンにのみ使用できます)。
ControlledByParent	親レコードに連動	親レコードに連動	(取引先責任者のみ)。すべての User および Group は、参照、編集、削除などの操作を取引先責任者に対して実行できます。この

API 値	Salesforce ユーザーインターフェースの表示ラベル	API 選択リストの表示ラベル	説明
			操作ができるかどうかは、ユーザーがその取引先責任者に関連付けられているレコードに対して同様の操作ができるかどうかに基づいています。

すべてのアクセスレベルを、すべてのオブジェクトに設定できるわけではありません。各オブジェクトの項目表を参照して、設定できるアクセスレベル、該当するオブジェクトに固有の共有についての詳細を確認してください。

一般的な共有についての詳細は、Salesforce ヘルプを参照してください。

メモ: API を使用して、レコードの共有エントリを定義する AccountShare や OpportunityShare などのオブジェクトを作成および更新できます。

オブジェクトおよび項目の暗黙的な制限

特定のオブジェクトは、Salesforce ユーザーインターフェースでのみ作成または削除できます。その他のオブジェクトは参照のみです。クライアントアプリケーションは、こうしたオブジェクトに create()、delete()、または update() を実行できません。同様に、一部のオブジェクト内の特定の項目は、create() には指定できますが、update() には指定できません。その他の項目は参照のみです。クライアントアプリケーションは、create() コールまたは update() コールで項目値を指定できません。詳細は、「オブジェクトの基本」の各オブジェクトの説明を参照してください。

Salesforce AppExchange パッケージの API アクセス

API を使用すると、API にログインするユーザーの権限に基づいて、オブジェクトやコールにアクセスできます。インストールしたパッケージに API を使用してデータにアクセスするコンポーネントが含まれている場合、セキュリティ問題が発生しないようにするために、次のような追加のセキュリティを設定します。

- 開発者が API にアクセスするコンポーネントを含む AppExchange パッケージを作成する場合、開発者はこれらのコンポーネントの API アクセスを制限することができます。
- 管理者が AppExchange パッケージをインストールする場合、管理者はアクセスを受け入れることも拒否することもできます。アクセスを拒否すると、インストールがキャンセルされます。
- 管理者がパッケージをインストールすると、管理者は API にアクセスするパッケージのコンポーネントの API アクセスを制限することができます。

パッケージの API アクセスの編集は、Salesforce ユーザーインターフェースで実行します。詳細は、Salesforce ヘルプの「*パッケージの API アクセスおよびダイナミック Apex アクセスの管理*」を参照してください。

パッケージの API アクセスは、パッケージ内のコンポーネントから発生する API 要求に影響を与えます。API 要求がアクセスできるオブジェクトを指定します。パッケージの API アクセスが定義されていない場合、API 要求がアクセス権限を割り当てられているオブジェクトは、ユーザーの権限によって決まります。

パッケージの API アクセスでは、ユーザーに付与された権限を超える操作は実行できません。パッケージの API アクセスでは、ユーザーの権限で実行可能な内容を限定していくことしかできません。

パッケージの［API アクセス］の設定で［制限あり］を選択した場合に影響を受ける項目は、次のとおりです。

- パッケージの API アクセスは、次のユーザー権限を上書きします。
  - Apex 開発
  - アプリケーションのカスタマイズ
  - HTML テンプレートの編集
  - 参照のみ項目の編集
  - 請求情報の管理
  - コールセンターの管理
  - カテゴリの管理
  - カスタムレポートタイプの管理
  - ダッシュボードの管理
  - レターヘッドの管理
  - パッケージライセンスの管理
  - 公開ドキュメントの管理
  - 公開リストビューの管理
  - 公開レポートの管理
  - 公開テンプレートの管理
  - ユーザーの管理
  - 所有権の移行
  - チーム再割り当てウィザードの使用
  - 設定・定義を参照する
  - ウィークリーデータのエクスポート
- ［参照］、［作成］、［編集］、および［削除］アクセスがオブジェクトの API アクセス設定で選択されていない場合、たとえ「すべてのデータの編集」権限および「すべてのデータの参照」権限を持つユーザーであっても、パッケージコンポーネントのオブジェクトへのアクセス権はありません。
- ［制限あり］API アクセスが有効になっているパッケージでは、ユーザーを作成できません。
- 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。

次の考慮事項は、パッケージの API アクセスにも適用されます。

パッケージへの API アクセスを管理するには、Salesforce ヘルプの「パッケージの API アクセスおよびダイナミック Apex アクセスの管理」を参照してください。

メモ: 制限されたパッケージからの XML-RPC 要求はアクセスが拒否されます。

送信ポートの制限

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。

ポートの制限は、送信メッセージ、AJAX プロキシ、またはシングルサインオンなど、ポートが指定されている機能に適用されます。

第 9 章　API の有効期限のポリシー

どの SOAP API バージョンがサポートされているか、サポートされていないか、または利用できないかを確認します。

Salesforce は、各 API バージョンを最初のリリース日から最低 3 年間サポートします。API の品質およびパフォーマンスを改善するために、3 年を超えるバージョンのサポートは停止される場合があります。

Salesforce は、廃止予定の API バージョンを使用するお客様に、そのバージョンのサポートが終了する最低 1 年前までに通知します。

Salesforce API バージョン	バージョンサポート状況	バージョン廃止情報
バージョン 31.0 ～ 60.0	サポート済み。	
バージョン 21.0 ～ 30.0	Summer '22 以降、次のバージョンが非推奨となり、Salesforce でサポートされなくなりました。 Summer '25 以降は、これらのバージョンは廃止されて使用できなくなります。	Salesforce Platform API バージョン 21.0 ～ 30.0 の廃止
バージョン 7.0 ～ 20.0	Summer '22 以降、次のバージョンが廃止されて使用できなくなりました。	Salesforce Platform API バージョン 7.0 ～ 20.0 の廃止

廃止された API バージョンからリソースを要求したり、操作を使用したりすると、SOAP API から `500:UNSUPPORTED_API_VERSION` エラーコードが返されます。

古い API バージョンやサポートされていない API バージョンで実行された要求を識別するには、API 合計使用量イベント種別を使用します。

第 10 章　Partner WSDL の使用

トピック:

API では、次の 2 つの WSDL を使用できます。

- **Enterprise Web サービス WSDL** — 単独の Salesforce 組織のクライアントアプリケーションを構築する Enterprise 開発者によって使用されます。Enterprise WSDL は非常に強い型付けであるため、コールは `int` や `string` など特定のデータ型のオブジェクトや項目を含みます。Enterprise WSDL ドキュメントを使用する顧客は、組織でカスタムオブジェクトまたはカスタム項目を変更したとき、あるいは異なるバージョンの API を使用するときに、Enterprise WSDL ドキュメントをダウンロードし、改めて使用する必要があります。組織の現在の WSDL にアクセスするには、Salesforce 組織にログインし、[設定] から、［クイック検索］ボックスに *「API」* と入力します。次に、API ページで **[Enterprise WSDL の生成]** を選択します。
- **Partner Web サービス WSDL** — メタデータ駆動で本来動的なクライアントアプリケーションに使用されます。特に、複数の組織が利用するクライアントアプリケーションを構築する Salesforce パートナーにとって役立ちます (もちろん、他のケースでも役立ちます)。特定のデータ型ではなく、項目名と値の名前-値のペアを使用する Salesforce データモデルはあまり強く型付けされていないため、いかなる組織内のデータにアクセスするためにも使用できます。この WSDL は、オブジェクトに対して処理を行う前にオブジェクトの情報を取得するクエリコールを発行できるクライアントアプリケーションの開発に最も適しています。Partner WSDL ドキュメントは、API のバージョンごとに 1 回のみダウンロードして使用する必要があります。組織の現在の WSDL にアクセスするには、Salesforce 組織にログインし、[設定] から、［クイック検索］ボックスに *「API」* と入力します。次に、API ページで **[パートナー WSDL の生成]** を選択します。

通常、Enterprise WSDL がより使用しやすいですが、Partner WSDL は柔軟性があり、さまざまな組織に動的に適応可能です。Partner WSDL を使用すると、複数のユーザーおよび複数の組織に使用できる単一のアプリケーションを更新できます。

高精度バージョン

正規の WSDL よりも高い精度が必要な場合、取引先チームまで、「高精度 API」機能についてお問い合わせください。この機能が有効になっている場合、ダウンロードした WSDL (Enterprise と Partner の両方) でより精度の高いデータ型が使用されます。たとえば、丸めエラーが発生しやすい複雑な数値数式を組織で使用する場合には、この機能が役立ちます。

メモ: この機能は限定されたパイロットであり、現在のところ正式にリリースされた機能ではありません。

WSDLの正規バージョンを使用していて、高精度バージョンに変更する場合、次の手順を実行します。

1. 新しい WSDL をダウンロードします。
2. スタブコードを再生成します。(「Java 開発者環境の設定」を参照)。
3. コードの数値を保存するために使用される変数の型が新しい型に対応できることを確認します。

Partner WSDL ファイルの取得

Partner WSDL を使用するには、次のいずれかの方法でファイルのコピーをダウンロードします。

- 組織の Salesforce システム管理者から取得する。
- 「ステップ 2: Web サービス WSDL を生成または取得する」の指示に従い、Salesforce の [設定] ([クイック検索] ボックスに *「API」* と入力し、**[API]** を選択) から生成する。

Enterprise WSDL は、カスタム項目またはカスタムオブジェクトが組織の Salesforce 情報に追加されたときは再生成が必要ですが、Partner WSDL は元となる組織の Salesforce データが変更されても変更の必要はありません。

コールと Partner WSDL

Partner WSDL ファイルは、Enterprise WSDL ファイルとまったく同一の API コールを定義します。Partner WSDL を使用しているクライアントアプリケーションでは、組織のメタデータの定義に次の API コールを使用します。

タスク/コール	説明
`describeGlobal()`	組織のデータで利用可能なオブジェクトの一覧を取得します。
`describeLayout()`	指定されたオブジェクト種別のページレイアウトに関するメタデータを取得します。
`describeSObject()`	`describeSObject()` は `describeSObjects()` で置き換えられました。
`describeSObjects()`	指定されたオブジェクトのメタデータの取得に使用します。まず、組織のすべてのオブジェクトのリスト取得のためにコールし、その後リスト内を反復参照し個別のオブジェクトのメタデータを取得します。
`describeTabs()`	Salesforce ユーザーインターフェースでは、ページ上部の Lightning プラットフォームアプリケーションメニューに示されている、標準的なアプリケーションへのアクセス権限を持ちます (カスタムアプリケーションへのアクセス権限が付与されていることもあります)。ユーザーインターフェースで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を選択し、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。

組織のメタデータを参照するために、クライアントアプリケーションは次の処理を実行します。

1. `describeGlobal()` をコールし、利用可能なオブジェクトの一覧を取得。
2. 返された DescribeGlobalResult オブジェクトで、`sobjects` をコールし、DescribeGlobalSObjectResult オブジェクトの配列を取得。
3. DescribeGlobalSObjectResult オブジェクトに対して `name` をコールし、返された sObject ごとの sObject の種別名を取得。
4. DescribeGlobalSObjectResult オブジェクトは sObject が作成可能かまたは更新可能かなど sObject に関するいくつかのメタデータを提供する。項目および子リレーションなど、特定の sObject の詳細を取得する必要がある

場合は、詳細を取得する必要のある sObject の種別名の配列を `describeSObjects()` に渡すことによって、describeSObjects() をコールする。

sObject 参照の再利用

sObject 参照は、1 回の操作内では再使用できません。

別の参照を使用してください。たとえば、次のコードスニペットは、2 つの異なる参照を使用し、カスタム項目とイベントを持つ取引先と取引先責任者を作成します。

```
SObject account = new com.sforce.soap.partner.sobject.wsc.SObject();
    account.setType("Account");
    account.setField("Name","myAccount");
    account.setField("XID1__c", "1");
    SObject refAcc1 = new com.sforce.soap.partner.sobject.wsc.SObject();
    refAcc1.setType("Account");
    refAcc1.setField("XID1__c", "1");
    SObject refAcc2 = new com.sforce.soap.partner.sobject.wsc.SObject();
    refAcc2.setType("Account");
    refAcc2.setField("XID1__c", "1");

    SObject contact = new com.sforce.soap.partner.sobject.wsc.SObject();
    contact.setType("Contact");
    contact.setField("LastName", "LName");
    contact.setField( "XID2__c", "2");
    contact.setField( "Account", refAcc1);
    SObject refCon = new com.sforce.soap.partner.sobject.wsc.SObject();
    contact.setType("Contact");
    contact.setField( "XID2__c", "2");


    SObject event = new com.sforce.soap.partner.sobject.wsc.SObject();
    contact.setType("Event");
    contact.setField("Subject", "myEvent");
    contact.setField( "ActivityDateTime", Calendar.getInstance());
    contact.setField("DurationInMinutes", 60);
    contact.setField("Who", refCon);
    contact.setField("What", refAcc2);

    client.create(new SObject[] { account, contact, event}); // exception thrown here
```

`sObject[] sObjects` 形式のパラメーターを取り込むコールは、すべてこの制限の対象となります。

オブジェクト、項目、項目データおよび Partner WSDL

Enterprise WSDL ファイルでは、Salesforce 組織の特定のオブジェクト (取引先や取引先責任者など) をすべて定義します。一方、Partner WSDL ファイルでは、すべてのオブジェクトを表す単一の汎用オブジェクト (sObject) を定義します。特定のオブジェクトの場合、その種類は返される DescribeSObjectResult の `name` 項目で定義されます。

Partner WSDL を使用すると、クライアントアプリケーションコードは、項目データを表す名前-値のペアの配列として、項目を処理します。各項目の名前を参照する場合、DescribeSObjectResult の Field 型の `name` 項目の値を使用します。

名前-値のペアを処理する方法、型指定された値を SOAP メッセージで定義されたプリミティブ XML データ型に対応付ける方法は、言語によって変わります。Enterprise WSDL では、対応付けは暗黙的に処理されます。一方、Partner WSDL では、クライアントアプリケーション構築時に値とデータ型を手動で管理します。項目値を割り当てる前にオブジェクト種別を指定します。特定の項目の値を指定する場合、項目に有効な値を使用します (範囲、形式、データ型)。プログラミング言語のデータ型と XML プリミティブデータ型間の対応付けを理解していることを確認してください。詳細は、「SOAPType」を参照してください。

クエリと Partner WSDL

Partner WSDL で `query()` コールを使用する場合、次のガイドラインに従ってください。

- queryString パラメーターは、大文字と小文字を区別しません。API は、大文字と小文字の組み合わせを使用した *fieldList* の項目名を受け入れます。ただし、QueryResult では、項目名 (定義済みの項目およびカスタム項目) の大文字と小文字は、DescribeSObjectResult の Field 型の `name` 項目の値と完全に一致する必要があります。*fieldList* で項目を指定する場合、大文字と小文字を適切に使用することをお勧めします。
- Partner WSDL の場合、QueryResult の項目の順序は、WSDL の項目順ではなく、*fieldList* の項目順で指定されます。
- *fieldList* には、重複する項目名を指定できません。次に例を示します。
  - 無効 (エラー発生): `"SELECT Firstname, Lastname, Firstname FROM User"`
  - 有効: `"SELECT Firstname, Lastname FROM User"`
- QueryResult では、項目の一部にデータがない (`null`) 場合でも、*fieldList* で指定されたすべての項目が指定されます。SOAP を使用すると結果セットに値が指定されていない項目を省略できますが、API は、すべての項目を含む配列を返します。
- Partner WSDL を使用する場合、ID を含むクエリでは SOAP XML 応答データに ID 項目が 2 回返されます。同様に、ID を含まないクエリでは、SOAP XML 応答データに単一の null ID 項目が返されます。たとえば、`SELECT ID, FirstName, LastName FROM Contact` に対するクエリによって、次のようなレコードとともに SOAP XML 応答が返される場合があります。

```
<records xsi:type="sf:sObject" xmlns="urn:partner.soap.sforce.com">
    <sf:type>Contact</sf:type>
    <sf:Id>0038000000FrjoBQRW</sf:Id>
    <sf:Id>0038000000FrjoBQRW</sf:Id>
    <sf:FirstName>John</sf:FirstName>
    <sf:LastName>Smith</sf:LastName>
</records>
```

これは予期される動作であり、すべての SOAP XML 応答データにアクセスし、WSC を使用して Web サービス応答にアクセスしていない場合に把握しておく必要があります。

Partner WSDL の名前空間

XML では、すべてのタグには定義済みの名前空間が割り当てられます。`enterprise.wsdl` では、名前空間は暗黙的に処理されます。API コールを Partner WSDL で使用する場合は、API コール、オブジェクト、項目、障害に適切な名前空間を明示的に指定する必要があります。このルールは、定義済みおよびカスタムのオブジェクトに適用されます。

対象	名前空間
API コール	`urn:partner.soap.sforce.com`
sObjects	`urn:sobject.partner.soap.sforce.com`
項目	`urn:sobject.partner.soap.sforce.com`
障害	`urn:fault.partner.soap.sforce.com`

パッケージバージョンと Partner WSDL

Partner WSDL は、強い型付けではありません。これにより、複数の組織と連携を行うパートナーに、より柔軟な方法を提供します。パッケージバージョンが API コールで指定されていない場合、API コールのデフォルトのパッケージバージョンによって代替システム設定が指定されます。

Partner API コールのパッケージ処理は、デフォルト値（[未指定]）がインストール済みパッケージに選択されている場合、最新バージョンのインストールパッケージによって定義されます。つまり、パッケージがアップグレードし、その変更がすぐにインテグレーションに影響を与える場合、Apex トリガーなどパッケージコンポーネントの処理が異なる場合があります。パッケージバージョンの後続のインストールによって、既存のインテグレーションに影響のないよう、登録ユーザーは、クライアントアプリケーションから、すべての Partner API コールのインストールパッケージの特定バージョンを選択する必要があります。

API クライアント開発者は、デフォルトのパートナーパッケージバージョン設定が組織内の 2 つの異なるロールに該当し、開発者として設定の変更を推奨する場合には、管理者にその旨を伝える必要があります。また、API クライアント開発者は、クライアントの PackageVersionHeader SOAP ヘッダーでパッケージバージョンを設定できます。

別のパッケージを参照するパッケージの開発を行うパートナーは、必ず Partner API コールの基本パッケージのバージョン情報を提供する必要があります。これにより、拡張パッケージは、基本パッケージで非推奨となっているコンポーネントの影響を受けません。

Partner API コールのパッケージバージョン情報は、次のうち、最初に一致する設定によって決定されます。

1. PackageVersionHeader SOAP ヘッダー。
2. Visualforce ページのパッケージバージョン設定を使用する Visualforce ページからの API コール。
3. パートナーパッケージバージョンのデフォルト設定。

Partner WSDL で API コールのデフォルトパッケージバージョンを設定するには、「パッケージバージョン設定」を参照してください。

ユーザーインターフェースのテーマ

Winter '06 リリースから、Salesforce は複数のユーザーインターフェースのテーマのサポートを開始し、ユーザーインターフェースでさまざまなアイコンと色のセットを使用できるようにしました。ただし、これらのユーザーインターフェースのテーマは、組織が Lightning Experience を使用している場合は適用されません。

エディション

使用可能なインターフェース: Salesforce Classic 以前

以前の Salesforce バージョンには 2 つのユーザーインターフェースのテーマが適合します。

- Theme3 — 「Salesforce Classic 2010 ユーザーインターフェースのテーマ」。このインターフェースは以前「Salesforce」または「新しいユーザーインターフェースのテーマ」と呼ばれていました。Salesforce *Aloha* インターフェースとしても馴染みがあるかもしれません。
- Theme2 — 「Salesforce Classic 2005 ユーザーインターフェースのテーマ」。このインターフェースは以前「Salesforce Classic」または「従来のユーザーインターフェースのテーマ」と呼ばれていました。

`getUserInfo()` コールは `getUserInfoResult` オブジェクトを返します。このオブジェクトには、`userUiSkin` プロパティがあります。このプロパティは、ユーザーの現在のユーザーインターフェースのテーマについて情報を提供します。

`describeQuickActions()`、`describeTabs()`、および `describeTheme()` コールとそれらの戻り値の型を使用して、テーマのアイコンおよび色に関する情報を取得します。

以前のユーザーインターフェースの外観を模倣するように、スタイルシートを使用できます。詳細は、*『Visualforce 開発者ガイド』* の「Visualforce ページのスタイル設定」を参照してください。ただし、Lightning Experience への切り替えを計画する場合は、新しい UI フレームワークである Lightning コンポーネントフレームワークを検討してください。詳細は、「Lightning Experience 向けの開発」 Trailhead トレイルの「Lightning コンポーネント」モジュールを参照してください。

Partner WSDL の使用例

このセクションでは、Partner WSDL を使用して API コールを行う Java および C# の例を示します。これらのサンプルを実行する前に、クイックスタートチュートリアルの次のステップを実行して、Partner WSDL ファイルを取得し、開発環境用のプロキシクライアントコードを生成してください。

- ステップ 2: Web サービス WSDL を生成または取得する
- ステップ 3: 開発プラットフォームに WSDL ファイルをインポートする

プロキシクライアントコードを生成し、開発環境を設定した後に、クライアントアプリケーションの作成を開始できます。まず、パートナー認証エンドポイントを使用して、アプリケーションで Salesforce サービスにログインする必要があります。ログインが成功したら、サンプルメソッドを実行できます。

利便性を実現するため、login コールを実行するテンプレートクラスが 1 つは Java で、または 1 つは C# で提供されています。これらのテンプレートクラスを使用して、このセクションの後で説明するサンプルメソッドを実行できます。

Java 用のサンプルテンプレートクラス: このサンプルでは、ユーザー名、パスワード、認証エンドポイントの入力画面を表示します。次に、ユーザーをログインします。認証エンドポイント URL では、Partner WSDL ファイルで見つかったエンドポイントを渡します。

```
import com.sforce.soap.partner.PartnerConnection;
import com.sforce.soap.partner.sobject.*;
import com.sforce.soap.partner.*;
import com.sforce.ws.ConnectorConfig;
import com.sforce.ws.ConnectionException;
import com.sforce.soap.partner.Error;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.BufferedReader;
import java.util.*;

public class PartnerSamples {
    PartnerConnection partnerConnection = null;
    private static BufferedReader reader =
        new BufferedReader(new InputStreamReader(System.in));

    public static void main(String[] args) {
        PartnerSamples samples = new PartnerSamples();
        if (samples.login()) {
            // Add calls to the methods in this class.
            // For example:
            // samples.querySample();
        }
    }

    private String getUserInput(String prompt) {
        String result = "";
        try {
          System.out.print(prompt);
          result = reader.readLine();
        } catch (IOException ioe) {
          ioe.printStackTrace();
        }
        return result;
    }

    private boolean login() {
        boolean success = false;
        String username = getUserInput("Enter username: ");
        String password = getUserInput("Enter password: ");
        String authEndPoint = getUserInput("Enter auth end point: ");

        try {
          ConnectorConfig config = new ConnectorConfig();
          config.setUsername(username);
          config.setPassword(password);

          config.setAuthEndpoint(authEndPoint);
          config.setTraceFile("traceLogs.txt");
          config.setTraceMessage(true);
          config.setPrettyPrintXml(true);

          partnerConnection = new PartnerConnection(config);

          success = true;
        } catch (ConnectionException ce) {
          ce.printStackTrace();
        } catch (FileNotFoundException fnfe) {
          fnfe.printStackTrace();
        }

        return success;
```

```
      }

    //
    // Add your methods here.
    //
}
```

C# 用のサンプルテンプレートクラス: このサンプルでは、ユーザー名とパスワードの入力画面を表示します。次に、ユーザーをログインします。このサンプルのプロジェクト名は *TemplatePartner* で、Web 参照名は *sforce* であることを想定しています。これらの値がプロジェクトで異なると、using ディレクティブを「`using your_project_name.web_reference_name;`」のように、実際のプロジェクト名と Web 参照名を含む値に変更してください。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Services.Protocols;
using System.Collections;
using TemplatePartner.sforce;

namespace TemplatePartner
{
    class PartnerSamples
    {
        private SforceService binding;

        static void Main(string[] args)
        {
            PartnerSamples samples = new PartnerSamples();
            if (samples.login())
            {
            // Add calls to the methods in this class.
            // For example:
            // samples.querySample();
            }
        }

        private bool login()
        {
            Console.Write("Enter username: ");
            string username = Console.ReadLine();
            Console.Write("Enter password: ");
            string password = Console.ReadLine();

            // Create a service object
            binding = new SforceService();

            // Timeout after a minute
            binding.Timeout = 60000;

            // Try logging in
            LoginResult lr;
            try
```

```
            {

                Console.WriteLine("\nLogging in...\n");
                lr = binding.login(username, password);
            }

            // ApiFault is a proxy stub generated from the WSDL contract when
            // the web service was imported
            catch (SoapException e)
            {
                // Write the fault code to the console
                Console.WriteLine(e.Code);

                // Write the fault message to the console
                Console.WriteLine("An unexpected error has occurred: " + e.Message);

                // Write the stack trace to the console
                Console.WriteLine(e.StackTrace);

                // Return False to indicate that the login was not successful
                return false;
            }

            // Check if the password has expired
            if (lr.passwordExpired)
            {
                Console.WriteLine("An error has occurred. Your password has expired.");
                return false;
            }

            // Set the returned service endpoint URL
            binding.Url = lr.serverUrl;

            // Set the SOAP header with the session ID returned by
            // the login result. This will be included in all
            // API calls.
            binding.SessionHeaderValue = new SessionHeader();
            binding.SessionHeaderValue.sessionId = lr.sessionId;

            // Return true to indicate that we are logged in, pointed
            // at the right URL and have our security token in place.
            return true;
        }

    //
    // Add your methods here.
    //
}
```

この Partner WSDL のサンプルは次のとおりです。

- query コールおよび queryMore コールのサンプル
- search コール例
- create コール例

- update コール例

query コールおよび queryMore コールのサンプル

次の Java および C# の例では、Partner WSDL の `query()` コールおよび `queryMore()` コールの使用方法を示します。各例では、クエリのバッチサイズを設定して、250項目が返されるようにします。次に、query コールを実行して、すべての取引先責任者の名と姓を取得し、返された取引先責任者レコードを反復処理します。取引先責任者ごとに、取引先責任者の名と姓、または名が null の場合は姓のみを出力に書き込みます。最後に、クエリによって返される項目がさらにある場合 (`QueryResult.done` プロパティの値が `false` をとる場合) は、`queryMore()` をコールして、項目の次のバッチを取得し、返されるレコードがなくなるまで処理を繰り返します。

サンプルメソッドを実行するには、Partner WSDL の使用例で提供されている対応する Java または C# テンプレートクラスを使用できます。

Java の例

```
public void querySample() {
    try {
        // Set query batch size
        partnerConnection.setQueryOptions(250);

        // SOQL query to use
        String soqlQuery = "SELECT FirstName, LastName FROM Contact";
        // Make the query call and get the query results
        QueryResult qr = partnerConnection.query(soqlQuery);

        boolean done = false;
        int loopCount = 0;
        // Loop through the batches of returned results
        while (!done) {
            System.out.println("Records in results set " + loopCount++
                    + " - ");
            SObject[] records = qr.getRecords();
            // Process the query results
            for (int i = 0; i < records.length; i++) {
                SObject contact = records[i];
                Object firstName = contact.getField("FirstName");
                Object lastName = contact.getField("LastName");
                if (firstName == null) {
                    System.out.println("Contact " + (i + 1) +
                            ": " + lastName
                    );
                } else {
                    System.out.println("Contact " + (i + 1) + ": " +
                            firstName + " " + lastName);
                }
            }
            if (qr.isDone()) {
                done = true;
            } else {
```

```
                qr = partnerConnection.queryMore(qr.getQueryLocator());
            }
        }
    } catch(ConnectionException ce) {
        ce.printStackTrace();
    }
    System.out.println("\nQuery execution completed.");
}
```

C# の例

```
public void querySample()
{
    try
    {
        QueryResult qr = null;
        binding.QueryOptionsValue = new sforce.QueryOptions();
        binding.QueryOptionsValue.batchSize = 250;
        binding.QueryOptionsValue.batchSizeSpecified = true;

        qr = binding.query("SELECT FirstName, LastName FROM Contact");

        bool done = false;
        int loopCount = 0;
        while (!done)
        {
            Console.WriteLine("\nRecords in results set " +
                Convert.ToString(loopCount++)
                    + " - ");
            // Process the query results
            for (int i = 0; i < qr.records.Length; i++)
            {
                sforce.sObject con = qr.records[i];
                string fName = con.Any[0].InnerText;
                string lName = con.Any[1].InnerText;
                if (fName == null)
                    Console.WriteLine("Contact " + (i + 1) + ": " + lName);
                else
                    Console.WriteLine("Contact " + (i + 1) + ": " + fName
                        + " " + lName);
            }

            if (qr.done)
                done = true;
            else
                qr = binding.queryMore(qr.queryLocator);
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
```

```
    Console.WriteLine("\nQuery execution completed.");
}
```

search コール例

次の Java および C# の例では、Partner WSDL の `search()` コールの使用方法を示します。各例では、SOQL クエリで使用されている電話番号の文字列を受け入れます。search コールでは、すべての取引先責任者、リード、および取引先の電話番号項目の中から、渡された値に一致する項目を検索します。次に、この例では、一致するレコードを含む返された検索結果を反復処理し、配列にそれらを追加し、コンソールにその項目の値を書き込みます。返されたレコード項目は、各レコードタイプの SOQL クエリで指定された項目に対応します。

サンプルメソッドを実行するには、Partner WSDL の使用例で提供されている対応する Java または C# テンプレートクラスを使用できます。

Java の例

```
public void searchSample(String phoneNumber) {
    try {
        // Example of phoneNumber format: 4155551212
        String soslQuery =
            "FIND {" + phoneNumber + "} IN Phone FIELDS " +
            "RETURNING " +
            "Contact(Id, Phone, FirstName, LastName), " +
            "Lead(Id, Phone, FirstName, LastName)," +
            "Account(Id, Phone, Name)";
        // Perform SOSL query
        SearchResult sResult = partnerConnection.search(soslQuery);
        // Get the records returned by the search result
        SearchRecord[] records = sResult.getSearchRecords();
        // Create lists of objects to hold search result records
        List<SObject> contacts = new ArrayList<SObject>();
        List<SObject> leads = new ArrayList<SObject>();
        List<SObject> accounts = new ArrayList<SObject>();

        // Iterate through the search result records
        // and store the records in their corresponding lists
        // based on record type.
        if (records != null && records.length > 0) {
          for (int i = 0; i < records.length; i++){
            SObject record = records[i].getRecord();
            if (record.getType().toLowerCase().equals("contact")) {
              contacts.add(record);
            } else if (record.getType().toLowerCase().equals("lead")){
              leads.add(record);
            } else if (record.getType().toLowerCase().equals("account")) {
              accounts.add(record);
            }
          }
          // Display the contacts that the search returned
          if (contacts.size() > 0) {
            System.out.println("Found " + contacts.size() +
```

```
            " contact(s):");
          for (SObject contact : contacts) {
            System.out.println(contact.getId() + " - " +
                contact.getField("FirstName") + " " +
                contact.getField("LastName") + " - " +
                contact.getField("Phone")
            );
          }
        }
        // Display the leads that the search returned
        if (leads.size() > 0) {
          System.out.println("Found " + leads.size() +
              " lead(s):");
          for (SObject lead : leads) {
            System.out.println(lead.getId() + " - " +
                lead.getField("FirstName") + " " +
                lead.getField("LastName") + " - " +
                lead.getField("Phone")
            );
          }
        }
        // Display the accounts that the search returned
        if (accounts.size() > 0) {
          System.out.println("Found " +
              accounts.size() + " account(s):");
          for (SObject account : accounts) {
            System.out.println(account.getId() + " - " +
                account.getField("Name") + " - " +
                account.getField("Phone")
            );
          }
        }
      } else {
        // The search returned no records
        System.out.println("No records were found for the search.");
      }
    } catch (ConnectionException ce) {
      ce.printStackTrace();
    }
}
```

C# の例

```
public void searchSample(String phoneNumber)
{
    try
    {
        // Example of phoneNumber format: 4155551212
        String soslQuery =
            "FIND {" + phoneNumber + "} IN Phone FIELDS " +
            "RETURNING " +
            "Contact(Id, Phone, FirstName, LastName), " +
            "Lead(Id, Phone, FirstName, LastName)," +
```

```
    "Account(Id, Phone, Name)";
// Perform SOSL query
SearchResult sResult = binding.search(soslQuery);
// Get the records returned by the search result
SearchRecord[] records = sResult.searchRecords;
// Create lists of objects to hold search result records
ArrayList contacts = new System.Collections.ArrayList();
ArrayList leads = new System.Collections.ArrayList();
ArrayList accounts = new System.Collections.ArrayList();

// Iterate through the search result records
// and store the records in their corresponding lists
// based on record type.
if ((records != null) && (records.Length > 0))
{
    for (int i = 0; i < records.Length; i++)
    {
        sObject record = records[i].record;

        if (record.type.ToLower().Equals("contact"))
        {
            contacts.Add(record);
        }
        else if (record.type.ToLower().Equals("lead"))
        {
            leads.Add(record);
        }
        else if (record.type.ToLower().Equals("account"))
        {
            accounts.Add(record);
        }
    }
    // Display the contacts that the search returned
    if (contacts.Count > 0)
    {
        Console.WriteLine("Found " + contacts.Count + " contact(s):");
        for (int i = 0; i < contacts.Count; i++)
        {
            sObject c = (sObject)contacts[i];
            Console.WriteLine(c.Any[0].InnerText + " - " +
                        c.Any[2].InnerText + " " +
                        c.Any[3].InnerText + " - " + c.Any[1].InnerText);
        }
    }
    // Display the leads that the search returned
    if (leads.Count > 0)
    {
        Console.WriteLine("Found " + leads.Count + " lead(s):");
        for (int i = 0; i < leads.Count; i++)
        {
            sObject l = (sObject)leads[i];
            Console.WriteLine(l.Any[0].InnerText + " - " +
                        l.Any[2].InnerText + " " +
                        l.Any[3].InnerText + " - " + l.Any[1].InnerText);
```

```
                }
            }
            // Display the accounts that the search returned
            if (accounts.Count > 0)
            {
                Console.WriteLine("Found " + accounts.Count + " account(s):");
                for (int i = 0; i < accounts.Count; i++)
                {
                    sObject a = (sObject)accounts[i];
                    Console.WriteLine(a.Any[0].InnerText + " - " +
                                      a.Any[2].InnerText + " - " +
                                      a.Any[1].InnerText);
                }
            }
        }
        else
        {
            // The search returned no records
            Console.WriteLine("No records were found for the search.");
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
}
```

create コール例

次の Java および C# の例では、Partner WSDL の `create()` コールの使用方法を示します。各例では、複数の項目を持つ取引先責任者レコードを作成します。create コールの結果を反復処理し、操作が成功したかどうかを確認します。作成操作が成功した場合は、作成された取引先責任者の ID をコンソールに書き込みます。成功しなかった場合は、エラーを反復処理し、各エラーの詳細をコンソールに書き込みます。この例では、出力は新しい取引先責任者の ID です。

サンプルメソッドを実行するには、Partner WSDL の使用例で提供されている対応する Java または C# テンプレートクラスを使用できます。

Java の例

```
public String createSample() {
    String result = null;
    try {
        // Create a new sObject of type Contact
           // and fill out its fields.
        SObject contact = new SObject();
        contact.setType("Contact");
        contact.setField("FirstName", "Otto");
        contact.setField("LastName", "Jespersen");
        contact.setField("Salutation", "Professor");
```

```
        contact.setField("Phone", "(999) 555-1234");
        contact.setField("Title", "Philologist");

        // Add this sObject to an array
        SObject[] contacts = new SObject[1];
        contacts[0] = contact;
        // Make a create call and pass it the array of sObjects
        SaveResult[] results = partnerConnection.create(contacts);

        // Iterate through the results list
        // and write the ID of the new sObject
        // or the errors if the object creation failed.
        // In this case, we only have one result
        // since we created one contact.
        for (int j = 0; j < results.length; j++) {
            if (results[j].isSuccess()) {
                result = results[j].getId();
                System.out.println(
                    "\nA contact was created with an ID of: " + result
                );
             } else {
                // There were errors during the create call,
                // go through the errors array and write
                // them to the console
                for (int i = 0; i < results[j].getErrors().length; i++) {
                    Error err = results[j].getErrors()[i];
                    System.out.println("Errors were found on item " + j);
                    System.out.println("Error code: " +
                        err.getStatusCode().toString());
                    System.out.println("Error message: " + err.getMessage());
                }
             }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
    return result;
}
```

C# の例

```
public void createSample()
{
    try
    {
        // Create a new sObject of type Contact
        // and fill out its fields.
        sObject contact = new sforce.sObject();
        System.Xml.XmlElement[] contactFields = new System.Xml.XmlElement[6];

        // Create the contact's fields
        System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
        contactFields[0] = doc.CreateElement("FirstName");
```

```
        contactFields[0].InnerText = "Otto";
        contactFields[1] = doc.CreateElement("LastName");
        contactFields[1].InnerText = "Jespersen";
        contactFields[2] = doc.CreateElement("Salutation");
        contactFields[2].InnerText = "Professor";
        contactFields[3] = doc.CreateElement("Phone");
        contactFields[3].InnerText = "(999) 555-1234";
        contactFields[4] = doc.CreateElement("Title");
        contactFields[4].InnerText = "Philologist";

        contact.type = "Contact";
        contact.Any = contactFields;

        // Add this sObject to an array
        sObject[] contactList = new sObject[1];
        contactList[0] = contact;

        // Make a create call and pass it the array of sObjects
        SaveResult[] results = binding.create(contactList);
        // Iterate through the results list
        // and write the ID of the new sObject
        // or the errors if the object creation failed.
        // In this case, we only have one result
        // since we created one contact.
        for (int j = 0; j < results.Length; j++)
        {
            if (results[j].success)
            {
                Console.Write("\nA contact was created with an ID of: "
                              + results[j].id);
            }
            else
            {
                // There were errors during the create call,
                // go through the errors array and write
                // them to the console
                for (int i = 0; i < results[j].errors.Length; i++)
                {
                    Error err = results[j].errors[i];
                    Console.WriteLine("Errors were found on item " + j.ToString());
                    Console.WriteLine("Error code is: " + err.statusCode.ToString());
                    Console.WriteLine("Error message: " + err.message);
                }
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
}
```

update() コール例

次の Java および C# の例では、Partner WSDL の `update()` コールの使用方法を示します。

各例では、取引先責任者の ID を `update()` への引数として取ります。種別が取引先責任者の 2 つの sObject レコード (1 つは ID で渡された有効な ID を持ち、もう一方は無効な ID を持つ) を作成します。次に、有効な取引先責任者に新しい電話番号を設定し、無効な取引先責任者の姓に `null` を設定します。その後、update コールを実行し、結果を反復処理します。更新操作が成功した場合は、更新された取引先責任者の ID を書き込みます。更新操作が失敗した場合は、返されたすべてのエラーの詳細をコンソールに書き込みます。この場合、出力は正常に更新された取引先責任者の ID と無効な取引先責任者の更新のエラーです。

サンプルメソッドを実行するには、Partner WSDL の使用例で提供されている対応する Java または C# テンプレートクラスを使用できます。

Java の例

```
public void updateSample(String id) {
  try {
      // Create an sObject of type contact
      SObject updateContact = new SObject();
      updateContact.setType("Contact");

      // Set the ID of the contact to update
      updateContact.setId(id);
      // Set the Phone field with a new value
      updateContact.setField("Phone", "(415) 555-1212");

      // Create another contact that will cause an error
      // because it has an invalid ID.
      SObject errorContact = new SObject();
      errorContact.setType("Contact");
      // Set an invalid ID on purpose
      errorContact.setId("SLFKJLFKJ");
      // Set the value of LastName to null
      errorContact.setFieldsToNull(new String[] {"LastName"});

      // Make the update call by passing an array containing
      // the two objects.
      SaveResult[] saveResults = partnerConnection.update(
         new SObject[] {updateContact, errorContact}
      );
      // Iterate through the results and write the ID of
      // the updated contacts to the console, in this case one contact.
      // If the result is not successful, write the errors
      // to the console. In this case, one item failed to update.
      for (int j = 0; j < saveResults.length; j++) {
          System.out.println("\nItem: " + j);
          if (saveResults[j].isSuccess()) {
              System.out.println("Contact with an ID of " +
                      saveResults[j].getId() + " was updated.");
          }
          else {
            // There were errors during the update call,
```

```
            // go through the errors array and write
            // them to the console.
            for (int i = 0; i < saveResults[j].getErrors().length; i++) {
              Error err = saveResults[j].getErrors()[i];
              System.out.println("Errors were found on item " + j);
              System.out.println("Error code: " +
                  err.getStatusCode().toString());
              System.out.println("Error message: " + err.getMessage());
            }
          }
      }
  } catch (ConnectionException ce) {
      ce.printStackTrace();
  }
}
```

`setFieldsToNull` (または WSC 以外のクライアントツールでの同様の機能) の詳細は、「`fieldsToNull`」を参照してください。

C# の例

```
public void updateSample(String id) {
    try
    {
        // Create an sObject of type contact
        sObject updateContact = new sObject();
        updateContact.type = "Contact";

        // Set the ID of the contact to update
        updateContact.Id = id;
        // Set the Phone field to a new value.
        // The Phone field needs to be created as an XML element.
        System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
        System.Xml.XmlElement phoneField = doc.CreateElement("Phone");
        phoneField.InnerText = "(415) 555-1212";

        // Add the Phone field to the contact
        updateContact.Any = new System.Xml.XmlElement[] {phoneField};

        // Create another contact that will cause an error
        // because it has an invalid ID.
        sObject errorContact = new sObject();
        errorContact.type = "Contact";
        // Set an invalid ID on purpose
        errorContact.Id = "SLFKJLFKJ";
        // Set the value of LastName to null
        errorContact.fieldsToNull = new String[] { "LastName" };

        // Make the update call by passing an array containing
        // the two objects.
        SaveResult[] saveResults = binding.update(
            new sObject[] {updateContact, errorContact});
        // Iterate through the results and write the ID of
```

```
        // the updated contacts to the console, in this case one contact.
        // If the result is not successful, write the errors
        // to the console. In this case, one item failed to update.
        for (int j = 0; j < saveResults.Length; j++) {
            Console.WriteLine("\nItem: " + j);
            if (saveResults[j].success)
            {
                Console.WriteLine("Contact with an ID of " +
                saveResults[j].id + " was updated.");
            }
            else
            {
                // There were errors during the update call,
                // go through the errors array and write
                // them to the console.
                for (int i = 0; i < saveResults[j].errors.Length; i++) {
                    Error err = saveResults[j].errors[i];
                    Console.WriteLine("Errors were found on item " + j.ToString());
                    Console.WriteLine("Error code: " +
                        err.statusCode.ToString());
                    Console.WriteLine("Error message: " + err.message);
                }
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message +
            " Stack trace: " + e.StackTrace);
    }
}
```

参照

第 11 章　Apex 関連のコール

トピック:

次の表は、APIがサポートしているコールをアルファベット順に表示し、それぞれの簡単な説明を示しています。コール名をクリックすると、構文、使用方法、各コールの詳細情報を確認できます。

メモ: 基本となる API コールのリストは「基本となる API コール」、記述用の API コールのリストは「記述用の API コール」、ユーティリティ API コールのリストは「ユーティリティ API コール」を参照してください。

コール	説明
`compileAndTest()`	単一のコールで Apex をコンパイルおよびテストします。
`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。
`compileTriggers()`	Developer Edition または Sandbox を使用している組織の Apex トリガーをコンパイルします。
`executeanonymous()`	Apex のブロックを実行します。
`runTests()`	Apex 単体テストを実行します。

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 (ページ 398) と SessionHeader (ページ 413) をサポートしています。

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ー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	削除されるトリガーの名前。

名前	型	説明
`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ーの内容。

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、クラスもトリガーは組織に保存されません。これは、Salesforce AppExchange パッケージテストと同じ要件です。
- すべてのトリガーには、コードカバー率が設定されている必要があります。トリガーにコードカバー率がない場合、クラスもトリガー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	トリガー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、トリガー、単体テストが正常に実行されています。クラス、トリガー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

名前	型	説明
`triggers`	CompileTriggerResult	トリガー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。

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ーファイルの 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。

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	トリガーファイルの 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ーの 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	トリガーの名前。

名前	型	説明
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、指定されたトリガーは正常にコンパイルされ、実行されています。トリガーのコンパイルまたは実行が失敗した場合、値は false です。

DeleteApexResult

このオブジェクトは、compileAndTest() コールがクラスまたはトリガー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	削除されたトリガーまたはクラスの 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ーはすべて正常に削除されています。クラスまたはトリガーが削除されていない場合、値は false です。

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 (ページ 398) と SessionHeader (ページ 413) をサポートしています。

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。

応答

CompileClassResult

compileTriggers()

Developer Edition または Sandbox を使用している組織の Apex トリガー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 トリガー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 (ページ 398) と SessionHeader (ページ 413) をサポートしています。

引数

名前	型	説明
`scripts`	string	Apex トリガー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

「CompileTriggerResult」を参照してください。

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 (ページ 398) と SessionHeader (ページ 413) 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 クラスとトリガー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。

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。
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。

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、データベースにデータをコミットせず、メールを送信しません。このメソッドは、メ

ソッド定義内で `@isTest` アノテーションでフラグ付けされます。単体テストメソッドは、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 (ページ 398) と SessionHeader (ページ 413) をサポートしています。

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。この項目は、Tooling SOAP APIコールrunTests()にも渡されます。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。トリガー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、トリガー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。

名前	型	説明
`maxFailedTests`	int	Tooling SOAP API コール `runTests()` の必須パラメーター。すべてのテストの実行を許可するには、`maxFailedTests` を `-1` に設定します。指定した数のテストに失敗した後に新しいテストの実行を停止するには、`maxFailedTests` を `0` ～ `1,000,000` の整数値に設定します。この整数値で、許容されるテスト失敗の最大数を設定します。値を `0` に設定すると、1 回の失敗でテスト実行が停止されます。値を `1` に設定すると、2 回目の失敗でテスト実行が停止されます。以降も同様に処理されます。
`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。
`skipCodeCoverage`	boolean	Apex テスト実行時にコードカバー率情報の収集を除外するかどうかを指定します。API バージョン 43.0 以降で利用できます。
`tests`	TestsNode[]	Tooling SOAP API コール `runTests()` の必須パラメーター。Apex テストクラスの個々のテストメソッドを指定します。 TestsNode[] の代わりにクラスまたはスイートを指定するには、`tests` を `null` に設定します。 このプロパティは配列を受け入れますが、配列に含めることができるエントリは 1 つのみです。

TestsNode

Apex テストクラスの個々のテストメソッドを指定します。

名前	型	説明
`classId`	string	**説明** 実行するテストメソッドが含まれる Apex クラスの ID。 `classId` または `className` が必要です。 **サポートされているメソッド** • `getClassId()` • `setClassId(new String "<your class ID>")`
`className`	string	**説明** 実行するテストメソッドが含まれる Apex クラスの名前。 管理パッケージからテストを選択するには、ドット表記を使用してパッケージの名前空間を含めます。 `classId` または `className` が必要です。

名前	型	説明
		サポートされているメソッド • `getClassName()` • `setClassName(new String "YourClassName")`
`testMethods`	string[]	説明 実行するテストメソッド。 必須。 サポートされているメソッド • `getTestMethods()` • `setTestMethods(new String[] {"testMethod1", "testMethod2"}))`

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
`apexLogId`	string	テスト実行の終了時に作成される ApexLog オブジェクトの ID。ApexLog オブジェクトは、Apex テストを実行しているユーザーや、実行されているクラスまたはトリガーに有効な追跡フラグがある場合に作成されます。 この項目は、API バージョン 35.0 以降で使用できます。
`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。

名前	型	説明
`flowCoverage`	FlowCoverageResult[]	フローを実行したテスト実行の結果の配列。この項目は、API バージョン 44.0 以降で使用できます。
`flowCoverageWarnings`	FlowCoverageWarning[]	フローを実行したテスト実行によって生成された警告の配列。この項目は、API バージョン 44.0 以降で使用できます。
`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(ミリ秒単位)。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ーについて、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ーについて、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ーについて、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ーの名前。

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ーについて、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の指定に使用されていました。

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。
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 (ミリ秒単位)。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。

FlowCoverageResult

このオブジェクトには、テスト実行で実行されたフローバージョンと要素数に関する情報が含まれます。このオブジェクトは API バージョン 44.0 以降で使用できます。

名前	型	説明
elementsNotCovered	string	テスト実行で実行されなかったフローバージョンの要素のリスト。
flowId	string	フローバージョンの ID。ID は組織内で一意です。
flowName	string	テスト実行で実行されたフローの名前。
flowNamespace	string	指定されている場合、フローを含む名前空間。
numElements	int	フローバージョンの要素の合計数。
numElementsNotCovered	int	テスト実行で実行されなかったフローバージョンの要素数。
processType	FlowProcessType (string 型の列挙)	フローバージョンのプロセス種別。

FlowCoverageWarning

このオブジェクトには、警告を生成したフローバージョンに関する情報が含まれます。このオブジェクトは API バージョン 44.0 以降で使用できます。

名前	型	説明
`flowId`	string	警告を生成したフローバージョンの ID。
`flowName`	string	警告を生成したフローの名前。警告が組織内のフローのテストカバー率全体に適用された場合、この値は null になります。
`flowNamespace`	string	指定されている場合、フローを含む名前空間。
`message`	string	生成された警告のメッセージ。

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。

第 12 章　基本となる API コール

次のリストは、API がサポートしているコールをアルファベット順に表示し、それぞれの簡単な説明を示しています。コール名をクリックすると、構文、使用方法、各コールの詳細情報を確認できます。

メモ: Apex 関連コールのリストは「Apex 関連コール」、記述用の API コールのリストは「記述用の API コール」、ユーティリティ API コールのリストは「ユーティリティ API コール」を参照してください。

コール	説明
`convertLead()`	Lead を Account、Contact または必要に応じて Opportunity に変換します。
`create()`	組織のデータに 1 つ以上の個別オブジェクトを追加します。
`delete()`	組織のデータから 1 つ以上の個別オブジェクトを削除します。
`deleteByExample()`	削除対象を指定するテンプレートとして sObject を使用して組織のデータからオブジェクトを削除します。sObject テンプレートと一致する Big Object 内のすべてのデータが削除されます。
`emptyRecycleBin()`	ごみ箱からレコードを直ちに削除します。
`executeListView()`	リストビューの SOQL クエリを実行して、リストビューからデータ、表示ラベル、およびアクションを取得します。
`findDuplicates()`	ルールに基づいて、重複レコードの検索を実行します。入力は sObject の配列で、それぞれが検索する値と重複ルールを示すオブジェクトの種別を指定します。重複ルールを示すオブジェクトごとに、検出された重複が出力で識別されます。`findDuplicates()` はルールを値に適用し、検索を実行します。sObject ごとに、検出された重複が出力で識別されます。
`findDuplicatesByIds()`	ルールに基づいて、重複レコードの検索を実行します。入力は ID の配列で、各要素は重複を検索するレコードを指定します。重複ルールを示すオブジェクトごとに、検出された重複が出力で識別されます。`findDuplicatesByIds()` はルールをレコード ID に適用し、検索を実行します。ID ごとに、検出された重複が出力で識別されます。
`getDeleted()`	指定された時間以降削除された、指定されたオブジェクトの個別オブジェクトの ID を取得します。ID についての詳細は、「ID データ型」を参照してください。

コール	説明
getUpdated()	指定された時間以降更新された、指定されたオブジェクトの個別オブジェクトの ID を取得します。ID についての詳細は、「ID データ型」を参照してください。
invalidateSessions()	sessionId で指定された 1 つ以上のセッションを終了します。
login()	ログインサーバーにログインし、クライアントセッションを開始します。
logout()	ログインユーザーのセッションを終了します。
merge()	同じオブジェクト種別のレコードをマージします。
performQuickActions()	種別が create または update のクイックアクションを実行します。
process()	承認のために承認プロセスインスタンスの配列を送信します。または、承認、却下、削除される承認プロセスインスタンスの配列を処理します。
query()	指定のオブジェクトに対してクエリを実行して、指定の条件に一致したデータを返します。
queryAll()	query() と同様ですが、削除されたアイテムとアーカイブされたアイテムを含みます。
queryMore()	クエリから次のオブジェクトのバッチを取得します。
retrieve()	指定したオブジェクト ID に基づいて 1 つ以上のオブジェクトを取得します。
search()	組織のデータをテキスト検索します。
undelete()	queryAll() で指定されたレコードを復元します。
update()	組織のデータ内にある 1 つ以上の既存オブジェクトを更新します。
upsert()	オブジェクトを作成するか、既存のオブジェクトを更新します。既存オブジェクトの存在を確認するために、カスタムオブジェクトの一致を確認します。

サンプル

このセクションのサンプルは、Enterprise WSDL ファイルに基づいてます。これらのサンプルでは、WSDL ファイルがインポート済みであり、接続が作成済みであることを前提としています。これを実行する方法の詳細については、「クイックスタート」のチュートリアルを参照してください。

convertLead()

リードから他のオブジェクトへ変換します。

Lead を Account、Contact または必要に応じて Opportunity に変換します。

構文

```
LeadConvertResult[] = connection.convertLead(leadConverts LeadConvert[]);
```

使用方法

convertLead() は Lead を Account、Contact、および必要に応じて Opportunity へと変換するために使用します。ビジネスによっては、convertLead() を使用してリードを取引先責任者ではなく、取引先と個人取引先に変換することもできます。Lead を変換するには、リードでは「リードの取引の開始」権限と「編集」権限、Account、Contact および Opportunity では「作成」権限と「編集」権限が割り当てられた状態でクライアントアプリケーションにログインする必要があります。

このコールにより、評価済みリードの情報を新規作成または更新された取引先、取引先責任者、商談に簡単に変換できます。組織は、リードを評価する時期を決定するガイドラインを独自に設定することができます。予測したい実際の商談になったときにリードを変換するのが一般的です。

データが既存の取引先、取引先責任者、商談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、既存のデータ (ID を含む) は上書きされません。唯一の例外は、クライアントアプリケーションが `overwriteLeadSource` を `true` に設定している場合です。この場合、変換先の Contact オブジェクトの `LeadSource` 項目は変換元の Lead オブジェクトの `LeadSource` 項目の内容で上書きされます。

リードを変換する際は、次のルールやガイドラインを考慮する必要があります。

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。

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。レコードタイプについての詳細は、Salesforce 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。

文字列値

API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了していました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

エラー

一括操作でいずれかのリードが変換に失敗した場合、リードごとに個別にリード変換が再試行されます。

Chatter フィードの自動登録

リードを新規の取引先、取引先責任者、商談に変換すると、リード所有者はそのリードレコードのChatterフィード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ーは、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。

ユーザーはレコードまたは他のユーザーを登録できます。レコードへの変更とユーザーからの更新は、ユーザーのホームページのChatterフィードに表示されます。これは、Salesforce で他のユーザーやレコードに加えられた変更の最新状況を把握するのに役立ちます。フィードは、API バージョン 18.0 以降で使用できます。

リードの変換の基本ステップ

リードの変換は、次の基本ステップに従います。

1. クライアント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、クライアント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。クライアントアプリケーションは SOSL または 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、クライアント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。クライアントアプリケーションは SOSL または 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、クライアントアプリケーションは商談をリードから作成するか、またはリードをマージする商談の ID から作成するかを決定します。クライアントアプリケーションは SOSL または 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 opportunity where name='OpportunityNameOfOpportunityBeingMerged'
```

5. クライアントアプリケーションはLeadStatusテーブルへのクエリを実行して、考えられる取引開始後の状況オプションを取得します。

```
SELECT Id, MasterLabel FROM LeadStatus WHERE IsConverted=true
```

   その後、取引開始後の状況の値を選択します。
6. クライアントアプリケーションは `convertLead()` をコールします。
7. クライアントアプリケーションは返された結果の各LeadConvertResultオブジェクトを反復処理して調べ、各リードの変換が成功したかどうかを確認します。
8. ベストプラクティスは、クライアントアプリケーションで `WhoId` が `ContactId` であるタスクを実行することです。商談が作成された場合、`WhatId` が `OpportunityId` となります。
9. 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。キューは、取引先と取引先責任者を所有することができません。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。

サンプルコード — Java

このサンプルでは、リードの取引の開始方法を示します。2つのリードを作成し、リードを開始します。次に、リードの取引開始結果を反復処理し、リードごとに作成された取引先、取引先責任者、商談のIDを書き込みます。

```
public String[] convertLeadRecords() {
   String[] result = new String[4];
   try {

      // Create two leads to convert
      Lead[] leads = new Lead[2];
      Lead lead = new Lead();
      lead.setLastName("Mallard");
      lead.setFirstName("Jay");
      lead.setCompany("Wingo Ducks");
      lead.setPhone("(707) 555-0328");
      leads[0] = lead;
      lead = new Lead();
      lead.setLastName("Platypus");
      lead.setFirstName("Ogden");
      lead.setCompany("Denio Water Co.");
      lead.setPhone("(775) 555-1245");
      leads[1] = lead;
      SaveResult[] saveResults = connection.create(leads);

      // Create a LeadConvert array to be used
      // in the convertLead() call
      LeadConvert[] leadsToConvert = new LeadConvert[saveResults.length];

      for (int i = 0; i < saveResults.length; ++i) {
         if (saveResults[i].isSuccess()) {
            System.out
                  .println("Created new Lead: " + saveResults[i].getId());
            leadsToConvert[i] = new LeadConvert();
```

```
            leadsToConvert[i].setConvertedStatus("Closed - Converted");
            leadsToConvert[i].setLeadId(saveResults[i].getId());
            result[0] = saveResults[i].getId();
         } else {
            System.out.println("\nError creating new Lead: "
                  + saveResults[i].getErrors()[0].getMessage());
         }
      }
      // Convert the leads and iterate through the results
      LeadConvertResult[] lcResults = connection.convertLead(leadsToConvert);
      for (int j = 0; j < lcResults.length; ++j) {
         if (lcResults[j].isSuccess()) {
            System.out.println("Lead converted successfully!");
            System.out.println("Account ID: " + lcResults[j].getAccountId());
            System.out.println("Contact ID: " + lcResults[j].getContactId());
            System.out.println("Opportunity ID: "
                  + lcResults[j].getOpportunityId());
         } else {
            System.out.println("\nError converting new Lead: "
                  + lcResults[j].getErrors()[0].getMessage());
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
   return result;
}
```

サンプルコード — C#

このサンプルでは、リードの取引の開始方法を示します。2 つのリードを作成し、リードを開始します。次に、リードの取引開始結果を反復処理し、リードごとに作成された取引先、取引先責任者、商談の ID を書き込みます。

```
public String[] convertLeadRecords()
{
   String[] result = new String[4];
   try
   {
      // Create two leads to convert
      Lead[] leads = new Lead[2];
      Lead lead = new Lead();
      lead.LastName = "Mallard";
      lead.FirstName = "Jay";
      lead.Company = "Wingo Ducks";
      lead.Phone = "(707) 555-0328";
      leads[0] = lead;
      lead = new Lead();
      lead.LastName = "Platypus";
      lead.FirstName = "Ogden";
      lead.Company = "Denio Water Co.";
      lead.Phone = "(775) 555-1245";
      leads[1] = lead;
```

```
      SaveResult[] saveResults = binding.create(leads);

      // Create a LeadConvert array to be used
      //   in the convertLead() call
      LeadConvert[] leadsToConvert =
            new LeadConvert[saveResults.Length]; ;
      for (int i = 0; i < saveResults.Length; ++i)
      {
         if (saveResults[i].success)
         {
            Console.WriteLine("Created new Lead: " +
                  saveResults[i].id);
            leadsToConvert[i] = new LeadConvert();
            leadsToConvert[i].convertedStatus = "Closed - Converted";
            leadsToConvert[i].leadId = saveResults[i].id;
            result[0] = saveResults[i].id;
         }
         else
         {
            Console.WriteLine("\nError creating new Lead: " +
                  saveResults[i].errors[0].message);
         }
      }
      // Convert the leads and iterate through the results
      LeadConvertResult[] lcResults =
            binding.convertLead(leadsToConvert);
      for (int j = 0; j < lcResults.Length; ++j)
      {
         if (lcResults[j].success)
         {
            Console.WriteLine("Lead converted successfully!");
            Console.WriteLine("Account ID: " +
                     lcResults[j].accountId);
            Console.WriteLine("Contact ID: " +
                     lcResults[j].contactId);
            Console.WriteLine("Opportunity ID: " +
                     lcResults[j].opportunityId);
         }
         else
         {
            Console.WriteLine("\nError converting new Lead: " +
                  lcResults[j].errors[0].message);
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                        e.Message + "\n" + e.StackTrace);
   }
   return result;
}
```

LeadConvert の引数

このコールは LeadConvert オブジェクトの配列を受け取ります (100 個まで)。LeadConvert 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
accountId	ID	リードをマージする取引先の ID。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。accountID が指定されない場合、API は取引先を作成します。取引先を作成するには、クライアントアプリケーションは条件を満たすアクセス権限でログインする必要があります。リードを既存の取引先にマージするには、クライアントアプリケーションは指定された取引先への「参照・更新」アクセス権を持ってログインする必要があります。取引先名と他の既存のデータは上書きされません。ID についての詳細は、「ID データ型」を参照してください。
contactId	ID	リードがマージされる Contact の ID (この取引先責任者は、指定された accountId と関連付けられていなければならず、accountId の指定が必要です)。既存の取引先を更新する場合のみ必要です。 リードを個人取引先に変換する場合は、contactId を指定しないでください。指定するとエラーになります。個人取引先の accountId のみを指定してください。 contactID が指定されない場合、API は Account と暗黙的に関連付けられる取引先責任者を作成します。取引先責任者を作成するには、クライアントアプリケーションは条件を満たすアクセス権限でログインする必要があります。リードを既存の取引先責任者にマージするには、クライアントアプリケーションは指定された取引先責任者への「参照・更新」アクセス権を持ってログインする必要があります。取引先責任者名とその他の既存データは上書きされません (ただし overwriteLeadSource が true に設定されている場合には、LeadSource 項目のみが上書きされます)。

convertedStatus	string	変換されたリードの有効な LeadStatus 値。必須。必須。この項目に設定できる値のリストを取得するには、クライアントアプリケーションが LeadStatus オブジェクトへのクエリを実行します。次に例を示します。
		
		`SELECT Id, MasterLabel`
		`FROM LeadStatus WHERE IsConverted=true`

doNotCreateOpportunity	boolean	リード変換時に Opportunity を作成するかどうかを指定 (デフォルトは false で商談を作成し、true では作成しない)。リー

名前	型	説明
		ドから商談を作成しない場合にのみ、このフラグを `true` に設定します。デフォルトでは、商談は作成されます。
`leadId`	ID	変換する Lead の ID。必須。ID についての詳細は、「ID データ型」を参照してください。
`opportunityId`	ID	リードに関連付ける既存の商談の ID。`opportunityId` 引数と `opportunityName` 引数は相互に排他的です。両方の値を指定すると、エラーが発生します。`doNotCreateOpportunity` 引数が `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。
`opportunityName`	string	作成する商談の名前。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。この項目の文字数は 80 文字までです。`opportunityId` 引数と `opportunityName` 引数は相互に排他的です。両方の値を指定すると、エラーが発生します。`doNotCreateOpportunity` 引数が `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。
`overwriteLeadSource`	boolean	変換先の Contact オブジェクトの `LeadSource` 項目に、変換元の Lead 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)。この項目を `true` に設定するには、クライアントアプリケーションは変換先の取引先責任者の `contactId` を指定する必要があります。
`ownerId`	ID	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ーの ID を指定します。クライアントアプリケーションでこの値を指定しない場合、新しいオブジェクトの所有者はリードの所有者になります。既存のオブジェクトにマージする場合には適用されません。`ownerId` が指定されている場合、API は既存の取引先または取引先責任者の `ownerId` 項目を上書きしません。ID についての詳細は、「ID データ型」を参照してください。
`relatedPersonAccountId`	ID	リードを取引先責任者ではなく、法人取引先と個人取引先に変換する場合、リードを変換する*既存*の個人取引先の ID を指定します。
`relatedPersonAccountRecord`	Entity	リードを取引先責任者ではなく、法人取引先と個人取引先に変換する場合、リードを変換する*新しい*個人取引先のエンティティレコードを指定します。

名前	型	説明
sendNotificationEmail	boolean	ownerId 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(送信する場合は true、送信しない場合は false。デフォルトは false)。

応答

LeadConvertResult[]

障害

UnexpectedErrorFault

関連トピック:

API コールの基礎

LeadConvertResult

convertLead() コールは、LeadConvertResult オブジェクトの配列を返します。LeadConvertResult 配列の各要素は、convertLead() コールの leadConverts パラメーターとして渡された LeadConvert[] 配列に対応します。たとえば、LeadConvertResult 配列の最初のインデックスで返されたオブジェクトは、LeadConvert[] 配列の最初のインデックスで指定されたオブジェクトに一致します。

LeadConvertResult 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
accountId	ID	新しい Account 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。
contactId	ID	新しい Contact 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。ID についての詳細は、「ID データ型」を参照してください。
leadId	ID	変換された Lead の ID。ID についての詳細は、「ID データ型」を参照してください。
opportunityId	ID	新規または既存の Opportunity の ID (convertLead() が呼び出されたときに作成または関連付けられた場合)。ID についての詳細は、「ID データ型」を参照してください。
relatedPersonAccountId	ID	新規または既存の関連する Person Account の ID (convertLead() が呼び出されたときに作成または関連付けられた場合)。ID についての詳細は、「ID データ型」を参照してください。

名前	型	説明
success	boolean	オブジェクトの convertLead() コールが成功したかどうかを示します (成功した場合は true、失敗した場合は false)。
errors	Error[]	create() コール中に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 Error オブジェクトからなる配列。

create()

組織のデータに 1 つ以上の新規レコードを追加します。

[other]: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

構文

```
SaveResult[] = connection.create(sObject[] sObjects);
```

使用方法

create() は、Account や Contact レコードなど、1 つ以上のレコードを組織の情報に追加するのに使用します。create() コールは SQL の INSERT ステートメントに類似しています。

オブジェクトを作成する際は、次のルールやガイドラインを考慮する必要があります。

権限

クライアントアプリケーションは、指定したオブジェクト内にレコードを作成するのに十分なアクセス権限でログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

特別な処理

特定のオブジェクト、またこれらのオブジェクト内にある特定の項目には、特別な処理または権限が必要です。たとえば、オブジェクトの親オブジェクトへのアクセス権限も必要となります。create() で特定のオブジェクトのレコードを作成する前に、必ず「標準オブジェクト」の説明を読んでください。

作成可能な項目

create() コールで作成できるのは、createable が true に設定されているオブジェクトのみです。指定されたオブジェクトが作成可能かどうかを確認するには、クライアントアプリケーションでオブジェクトに対する describeSObjects() describeSObjects() コールを実行し、createable プロパティを確認します。

自動的に保持される項目

API は ID 項目に一意な値を自動的に生成します。`create()` では、`sObject` に ID 値を明示的に指定することはできません。`saveResult[]` オブジェクトには、正常に作成された各レコードの ID が含まれます。ID についての詳細は、「ID データ型」を参照してください。

API は、`CreatedDate`、`CreatedById`、`LastModifiedDate`、`LastModifiedById` および `SystemModstamp` などの特定の項目を自動的に入力します。これらの値は明示的に指定できません。

必須項目

デフォルト値が事前に設定されていない必須項目には、値を指定する必要があります。詳細は、「必須項目」を参照してください。

デフォルト値

オブジェクトによっては、`OwnerID` などデフォルト値のある項目があります。そのような項目に値を指定しないと、API は項目にデフォルト値を入力します。たとえば、`OwnerID` を上書きしない場合、API はクライアントアプリケーションがログインユーザーに関連したユーザー ID を項目に入力します。

- デフォルト値が事前に設定されていない必須項目には、値を指定する必要があります。
- オブジェクトの他の項目では、値を明示的に指定しない場合は `null (VT_EMPTY)` となります。

参照整合性

クライアントアプリケーションは参照整合性に従う必要があります。たとえば、親オブジェクトの子であるオブジェクトのレコードを作成する場合、子を親にリンクする外部キー情報を指定する必要があります。たとえば、CaseComment を作成する場合、親の Case の有効なケース ID を指定する必要があり、また、親のケースはデータベース内に存在しなければなりません。

データ値の検証

整数項目については整数(英字は不可)、項目のデータ型に対して有効な値を入力する必要があります。クライアントアプリケーションでは、使用しているプログラム言語および開発ツールに指定されたデータ形式ルールに従ってください (開発ツールは、SOAP メッセージのデータ型の適切な対応付けを処理します)。

文字列値

文字列項目に値を格納する場合、前後にある空白は API が切り捨てます。たとえば、名前項目の値に `" ABC Company "` と入力されると、その値はデータベースに `"ABC Company"` と保存されます。

API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了していました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

割り当てルール

Account (取引先はテリトリー管理割り当てルールを実行)、Case、または Lead レコードを作成する場合、クライアントアプリケーションでは、Salesforce ユーザーインターフェースで設定された割り当てルールに基づいて 1 人以上のユーザーにケースまたはリードを自動的に割り当てるために、AssignmentRuleHeader にオプションを設定することができます。

最大作成レコード数

クライアントアプリケーションは、1 回の `create()` コールで最大 200 個のレコードを追加できます。create 要求が 200 レコードを超えると、操作全体が失敗します。

エラー時のロールバック

AllOrNoneHeader ヘッダーを使用すると、すべてのレコードが正常に処理されない限り、すべての変更をロールバックできます。このヘッダーは、API バージョン 20.0 以降で使用できます。正常に処理されないレコードがあった場合に、コールですべての変更をロールバックできます。

Chatter フィードの自動登録

作成するレコードを登録するには、ユーザーは個人設定の [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。自動登録が有効化されている場合、ユーザーは自分が作成したレコードを自動的にフォローし、それらのレコードへの変更が [ホーム] タブの Chatter フィードに表示されます。

ユーザーはレコードまたは他のユーザーを登録できます。レコードへの変更とユーザーからの更新は、ユーザーのホームページの Chatter フィードに表示されます。これは、Salesforce で他のユーザーやレコードに加えられた変更の最新状況を把握するのに役立ちます。フィードは、API バージョン 18.0 以降で使用できます。EntitySubscription オブジェクトは、レコードまたは他のユーザーをフォローしているユーザーの登録を表します。

フィード通知の無効化

大量のレコードを処理していて、レコードに関連するさまざまな項目を追跡しない場合、DisableFeedTrackingHeader を使用します。このヘッダーは一括変更で特に役立ちます。

オブジェクト種別が異なるレコードの作成

API バージョン 20.0 以降では、1 回のコールで、カスタムオブジェクトも含め、複数のオブジェクト種別のレコードを作成できます。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 種類のオブジェクトのレコードを作成できます。

レコードは、`sObjects` 入力配列に入力された順序で保存されます。親子リレーションを持つ新規レコードを入力する場合、`sObjects` 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、`sObjects` 配列で取引先の

インデックスは取引先責任者のインデックスよりも小さくする必要があります。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、別の取引先責任者を参照する［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 sObjects 配列の別の取引先責任者を参照する場合、その両方の取引先責任者を 1 回のコールで追加することはできません。すでに作成された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できます。

Salesforce では、オブジェクト種別が異なるレコードは複数のチャンクに分けられます。1 つのチャンクは、sObjects 入力配列のサブセットで、各チャンクにはオブジェクト種別が同じレコードが含まれます。データは、チャンク単位にコミットされます。チャンクに含まれるレコードに関連する Apex トリガーは、チャンクごとに 1 回起動されます。次のレコードのセットを含む sObjects 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。

- account1, account2
- contact1, contact2, contact3
- case1
- account3, account4
- contact4

コールごとに最大 10 個のチャンクを処理できます。sObjects 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。

警告: 複数のオブジェクト種別のレコードを作成する場合、いずれかのオブジェクト種別が Salesforce の［設定］領域の機能に関連していると、1 回のコールで作成することはできません。唯一の例外は次のオブジェクトです。

- カスタム設定のオブジェクト。カスタムオブジェクトに類似しています。詳細は、Salesforce ヘルプの「カスタム設定の作成」を参照してください。
- GroupMember
- Group
- User。UserRoleId 項目が設定されていない場合のみ。

create() と外部キー

外部 ID 項目を外部キーとして使用することによって、最初に親レコードの ID を照会するのではなく、レコードを作成して他の既存のレコードに関連付ける操作を 1 つの手順で実行できます。指定された外部 ID 項目のみを持つ親 sObject のインスタンスに外部キー項目を設定します。この外部 ID は親レコードの外部 ID の値と一致する必要があります。

次の Java および C# の例では、MyExtId__c というカスタム外部 ID 項目を使用して商談を作成して、既存の取引先に関連付ける方法を示します。各例では商談を作成し、必須項目を設定してから、指定された外部 ID 項目のみを持つ取引先オブジェクトに商談の外部 ID 項目を設定します。次に、コードは商談を作成します。商談が作成されると、取引先はその親になります。

Java の例

```
public void createForeignKeySample() {
   try {
      Opportunity newOpportunity = new Opportunity();
      newOpportunity.setName("OpportunityWithFK");
      newOpportunity.setStageName("Prospecting");
      Calendar dt = connection.getServerTimestamp().getTimestamp();
      dt.add(Calendar.DAY_OF_MONTH, 7);
      newOpportunity.setCloseDate(dt);

      Account parentAccountRef = new Account();
      parentAccountRef.setMyExtId__c("SAP1111111");
      newOpportunity.setAccount(parentAccountRef);

      SaveResult[] results = connection
            .create(new SObject[] { newOpportunity });
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

C# の例

```
public void createForeignKeySample()
{
   try
   {
      Opportunity newOpportunity = new Opportunity();
      newOpportunity.Name = "OpportunityWithFK";
      newOpportunity.StageName = "Prospecting";
      DateTime dt = (DateTime)binding.getServerTimestamp().timestamp;
      newOpportunity.CloseDate = dt.AddDays(7);
      newOpportunity.CloseDateSpecified = true;

      // Create the parent reference.
      // Used only for foreign key reference
      //  and doesn't contain any other fields
      Account accountReference = new Account();
      accountReference.MyExtId__c = "SAP1111111";
      newOpportunity.Account = accountReference;

      // Create the account and the opportunity
      SaveResult[] results = binding.create(new sObject[] {
                              newOpportunity });
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                              e.Message + "\n" + e.StackTrace);
   }
}
```

外部キーを使用して1つのコール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 を照会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 つのコールで作成することができます。外部 ID 項目を外部キーとして使用し、1 回の呼び出しで異なる sObject 種別の親レコードと子レコードを作成する手順は、次のとおりです。

1. 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。
2. 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。
3. 作成した親参照 sObject に子 sObject の外部キー項目を設定します。
4. `create()` コールに渡すその他の親 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。
5. 作成する sObject の配列を渡して、`create()` 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。

親レコードと子レコードは、主従関係または参照関係など、事前定義された関係によって関連付けられているレコードです。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、「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 Java および C# の例では、1 回の `create()` コールで親取引先を持つ商談を作成する方法を示します。各例では、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、`create()` 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。`create()` コールでは、1 回のコール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、コールの結果を確認し、作成されたレコードの ID をコンソールに書き込むか、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。

Java の例

```
public void createForeignKeySample() {
   try {
      Opportunity newOpportunity = new Opportunity();
      newOpportunity.setName("OpportunityWithAccountInsert");
      newOpportunity.setStageName("Prospecting");
      Calendar dt = connection.getServerTimestamp().getTimestamp();
      dt.add(Calendar.DAY_OF_MONTH, 7);
      newOpportunity.setCloseDate(dt);

      // Create the parent reference.
      // Used only for foreign key reference
      // and doesn't contain any other fields.
      Account accountReference = new Account();
      accountReference.setMyExtID__c("SAP111111");
      newOpportunity.setAccount(accountReference);
```

```
      // Create the Account object to insert.
      // Same as above but has Name field.
      // Used for the create call.
      Account parentAccount = new Account();
      parentAccount.setName("Hallie");
      parentAccount.setMyExtID__c("SAP111111");

      // Create the account and the opportunity.
      SaveResult[] results = connection.create(new SObject[] {
            parentAccount, newOpportunity });

      // Check results.
      for (int i = 0; i < results.length; i++) {
         if (results[i].isSuccess()) {
            System.out.println("Successfully created ID: "
                  + results[i].getId());
         } else {
            System.out.println("Error: could not create sobject "
                  + "for array element " + i + ".");
            System.out.println("   The error reported was: "
                  + results[i].getErrors()[0].getMessage() + "\n");
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

C# の例

```
public void createForeignKeySample()
{
   try
   {
      Opportunity newOpportunity = new Opportunity();
      newOpportunity.Name = "OpportunityWithAccountInsert";
      newOpportunity.StageName = "Prospecting";
      DateTime dt = (DateTime)binding.getServerTimestamp().timestamp;
      newOpportunity.CloseDate = dt.AddDays(7);
      newOpportunity.CloseDateSpecified = true;

      // Create the parent reference.
      // Used only for foreign key reference
      //  and doesn't contain any other fields.
      Account accountReference = new Account();
      accountReference.MyExtID__c = "SAP111111";
      newOpportunity.Account = accountReference;

      // Create the Account object to insert.
      // Same as above but has Name field.
      // Used for the create call.
      Account parentAccount = new Account();
      parentAccount.Name = "Hallie";
      parentAccount.MyExtID__c = "SAP111111";
```

```
      // Create the account and the opportunity.
      SaveResult[] results = binding.create(new sObject[] {
      parentAccount, newOpportunity });

      // Check results.
      for (int i = 0; i < results.Length; i++)
      {
         if (results[i].success)
         {
            Console.WriteLine("Successfully created ID: "
                  + results[i].id);
         }
         else
         {
            Console.WriteLine("Error: could not create sobject "
                  + "for array element " + i + ".");
            Console.WriteLine("   The error reported was: "
                  + results[i].errors[0].message + "\n");
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                           e.Message + "\n" + e.StackTrace);
   }
}
```

レコード作成の基本手順

レコードは、次の基本手順に従って作成します。

1. 1 つ以上のオブジェクトの `sObject` を作成します。各レコードに対し、追加するデータを項目に入力します。
2. `sObject[]` 配列を作成し、作成するオブジェクトを配列に入力します。
3. `create()` をコールし、`sObject[]` 配列を渡します。
4. `saveResult[]` オブジェクトの結果を処理してレコードの作成が成功したかどうかを確認します。

サンプルコード — Java

このサンプルでは、レコードの作成方法を示します。ここでは、2 つの取引先オブジェクトを作成し、それらの項目を設定します。2 番目の取引先の Name は設定されていないため、作成時にエラーが発生します。これは、Name が必須項目であるためです。このサンプルでは、2 つの取引先を含む配列を渡して、`create()` コールを行った後、結果を反復処理して、新しい取引先の ID を書き込むか、または、取引先の作成が失敗した場合はエラーメッセージを書き込みます。最後に、新しい取引先 ID の配列を返します。この例では、1 つの ID のみが含まれています。

```
public String[] createRecords() {
   // Create two accounts
```

```
    String[] result = new String[2];
    Account account1 = new Account();
    Account account2 = new Account();

    // Set some fields on the account object
    account1.setName("The Brick Hut");
    account1.setBillingStreet("403 McAdoo St");
    account1.setBillingCity("Truth or Consequences");
    account1.setBillingState("NM");
    account1.setBillingPostalCode("87901");
    account1.setBillingCountry("US");
    // Required Name field is not being set on account2,
    // so this record should fail during create.
    // account2.setName("Camp One Creations");
    account2.setBillingStreet("25800 Arnold Dr");
    account2.setBillingCity("Sonoma");
    account2.setBillingState("CA");
    account2.setBillingPostalCode("95476");
    account2.setBillingCountry("US");
    Account[] accounts = { account1, account2 };

    try {
       // Call create() to add the accounts
       SaveResult[] saveResults = connection.create(accounts);
       // Iterate through the results.
       // There should be one successful creation
       // and one failed creation.
       for (int i = 0; i < saveResults.length; i++) {
          if (saveResults[i].isSuccess()) {
             System.out.println("Successfully created Account ID: "
                   + saveResults[i].getId());
             result[i] = saveResults[i].getId();
          } else {
             System.out.println("Error: could not create Account "
                   + "for array element " + i + ".");
             System.out.println("   The error reported was: "
                   + saveResults[i].getErrors()[0].getMessage() + "\n");
             result[i] = saveResults[i].getId();
          }
       }
    } catch (ConnectionException ce) {
       ce.printStackTrace();
    }
    return result;
}
```

サンプルコード — C#

このサンプルでは、レコードの作成方法を示します。ここでは、2つの取引先オブジェクトを作成し、それらの項目を設定します。2番目の取引先の Name は設定されていないため、作成時にエラーが発生します。これは、Nameが必須項目であるためです。このサンプルでは、2つの取引先を含む配列を渡して、`create()` コールを行った後、結果を反復処理して、新しい取引先の ID を書き込むか、または、取引先の作成が失敗した場

合はエラーメッセージを書き込みます。最後に、新しい取引先IDの配列を返します。この例では、1つのIDのみが含まれています。

```
public String[] createRecords()
{
   // Create two accounts
   String[] result = new String[2];
   Account account1 = new Account();
   Account account2 = new Account();

   // Set some fields on the account object
   account1.Name = "The Brick Hut";
   account1.BillingStreet = "403 McAdoo St";
   account1.BillingCity = "Truth or Consequences";
   account1.BillingState = "NM";
   account1.BillingPostalCode = "87901";
   account1.BillingCountry = "US";
   // Required Name field is not being set on account2,
   // so this record should fail during create.
   // account2.Name = "Camp One Creations";
   account2.BillingStreet = "25800 Arnold Dr";
   account2.BillingCity = "Sonoma";
   account2.BillingState = "CA";
   account2.BillingPostalCode = "95476";
   account2.BillingCountry = "US";
   Account[] accounts = { account1, account2 };

   try
   {
      // Call create() to add the accounts
      SaveResult[] saveResults = binding.create(accounts);
      // Iterate through the results.
      // There should be one successful creation
      //   and one failed creation.
      for (int i = 0; i < saveResults.Length; i++)
      {
         if (saveResults[i].success)
         {
            Console.WriteLine("Successfully created Account ID: " +
                  saveResults[i].id);
            result[i] = saveResults[i].id;
         }
         else
         {
            Console.WriteLine("Error: could not create Account " +
                  "for array element " + i + "."
            );
            Console.WriteLine("   The error reported was: " +
            saveResults[i].errors[0].message + "\n");
            result[i] = saveResults[i].id;
         }
      }
   }
   catch (SoapException e)
```

```
    {
        Console.WriteLine("An unexpected error has occurred: " +
                          e.Message + "\n" + e.StackTrace);
    }

    return result;
}
```

引数

名前	型	説明
sObjects	sObject[]	create() で作成する 1 つ以上の sObject オブジェクトの配列。制限: 200 個の sObject 値。

応答

saveResult[]

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

- upsert()
- API コールの基礎

SaveResult

create() コールは、SaveResult オブジェクトの配列を返します。

SaveResult 配列の各要素は、create() コールの sObjects パラメーターとして渡された sObject[] 配列に対応します。たとえば、SaveResult 配列の最初のインデックスで返されるオブジェクトは、sObject[] 配列の最初のインデックスで指定されるオブジェクトに一致します。

SaveResult 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	create() で作成しようとした sObject の ID。この項目に値が含まれている場合、オブジェクトの作成は成功です。この項目が空の場合、オブジェクトは作成されず、API はエラー情報を返します。
success	boolean	オブジェクトの create() コールが成功したかどうかを示します (成功した場合は true、失敗した場合は false)。

名前	型	説明
errors	Error[]	create() コール中に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 Error オブジェクトからなる配列。 組織に有効な重複ルールがあり、重複が検出された場合、SaveResult にはデータ型が DuplicateError の Error が含まれます。

delete()

組織のデータから 1 つ以上のレコードを削除します。

構文

```
DeleteResult[] = connection.delete(ID[] ids);
```

使用方法

delete() を使用して、個別の取引先や取引先責任者など、組織のデータ内にある 1 つ以上の既存レコードを削除します。delete() コールは SQL の DELETE ステートメントに類似しています。

ルールとガイドライン

オブジェクトを削除する際は、次のルールやガイドラインを考慮する必要があります。

- 指定したオブジェクト内の個別のオブジェクトを削除するには、クライアントアプリケーションは、実行に十分なアクセス権限でログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。
- また、オブジェクトの親オブジェクトへのアクセス権限も必要となる場合があります。特別なアクセス要件については、「標準オブジェクト」のオブジェクトの説明を参照してください。
- 参照整合性を確実にするために、delete() コールは、カスケード削除をサポートします。親オブジェクトを削除すると、各子オブジェクトが削除可能な場合は自動的に削除されます。たとえば、Case を削除すると、API はケースに関連付けられたすべての CaseComment、CaseHistory および CaseSolution オブジェクトを自動的に削除します。ただし、CaseComment が削除可能でない場合、または使用中の場合、親 Case オブジェクトの delete() コールは失敗します。
- 特定のオブジェクトは、API からは削除できません。delete() コールを使用してオブジェクトを削除するには、オブジェクトを削除可能に設定する (deletable を true にする) 必要があります。指定されたオブジェクトが削除可能かどうかを確認するには、クライアントアプリケーションでオブジェクトに対する describeSObjects() コールを実行し、deletable プロパティを確認します。
- 複数のオブジェクト種別のレコードは、いずれかのオブジェクト種別が Salesforce の [設定] 領域の機能に関連していると、1 回のコールで削除することはできません。唯一の例外は次のオブジェクトです。
  - カスタム設定のオブジェクト。カスタムオブジェクトに類似しています。詳細は、Salesforce ヘルプの「カスタム設定の作成」を参照してください。

- GroupMember
- Group
- User

エラー時のロールバック

AllOrNoneHeader ヘッダーを使用すると、すべてのレコードが正常に処理されない限り、すべての変更をロールバックできます。このヘッダーは、API バージョン 20.0 以降で使用できます。正常に処理されないレコードがあった場合に、コールですべての変更をロールバックできます。

レコード削除の基本手順

レコードは、次の基本手順に従って削除します。

1. 削除する各レコードの ID を確認します。たとえば、`query()` を特定の検索条件に基づいてコールし、削除するレコードのセットを取得します。
2. ID[] 配列を作成し、削除する各オブジェクトの ID を入力します。同じコールで、複数のオブジェクト種別の ID を指定できます。たとえば、個別の Account の ID と個別の Contact の ID を同じ配列で指定できます。ID についての詳細は、「ID データ型」を参照してください。
3. `delete()` をコールし ID[] 配列を渡します。
4. `DeleteResult[]` の結果を処理してレコードの削除が成功したかどうかを確認します。

サンプルコード — Java

このサンプルでは、レコード ID に基づいてレコードを削除する方法を示します。このサンプルのメソッドは、ID の配列を受け取ります。この ID の配列を `delete()` コールに渡し、コールを実行します。次に、その結果を解析し、削除されたレコードの ID をコンソールに書き込むか、削除が失敗した場合は最初に返されるエラーを書き込みます。

```
public void deleteRecords(String[] ids) {
   try {
      DeleteResult[] deleteResults = connection.delete(ids);
      for (int i = 0; i < deleteResults.length; i++) {
         DeleteResult deleteResult = deleteResults[i];
         if (deleteResult.isSuccess()) {
            System.out
                  .println("Deleted Record ID: " + deleteResult.getId());
         } else {
            // Handle the errors.
            // We just print the first error out for sample purposes.
            Error[] errors = deleteResult.getErrors();
            if (errors.length > 0) {
               System.out.println("Error: could not delete " + "Record ID "
                     + deleteResult.getId() + ".");
               System.out.println("   The error reported was: ("
                     + errors[0].getStatusCode() + ") "
                     + errors[0].getMessage() + "\n");
```

```
            }
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、レコードIDに基づいてレコードを削除する方法を示します。このサンプルのメソッドは、IDの配列を受け取ります。このIDの配列を `delete()` コールに渡し、コールを実行します。次に、その結果を解析し、削除されたレコードのIDをコンソールに書き込むか、削除が失敗した場合は最初に返されるエラーを書き込みます。

```
public void deleteRecords(String[] ids)
{
   try
   {
      DeleteResult[] deleteResults = binding.delete(ids);
      for (int i = 0; i < deleteResults.Length; i++)
      {
         DeleteResult deleteResult = deleteResults[i];
         if (deleteResult.success)
         {
            Console.WriteLine("Deleted Record ID: " + deleteResult.id);
         }
         else
         {
            // Handle the errors.
            // We just print the first error out for sample purposes.
            Error[] errors = deleteResult.errors;
            if (errors.Length > 0)
            {
               Console.WriteLine("Error: could not delete " + "Record ID "
                     + deleteResult.id + ".");
               Console.WriteLine("   The error reported was: ("
                     + errors[0].statusCode + ") "
                     + errors[0].message + "\n");
            }
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                           e.Message + "\n" + e.StackTrace);
   }
}
```

引数

名前	型	説明
ids	ID[]	削除するオブジェクトに関連付けられた 1 つ以上の ID の配列。バージョン 7.0 以降では、最大 200 個のオブジェクト ID を delete() コールに渡すことができます。バージョン 6.0 以前では、最大値は 2,000 個です。

応答

DeleteResult[]

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

API コールの基礎

DeleteResult

delete() コールは、DeleteResult オブジェクトの配列を返します。

DeleteResult 配列の各要素は、delete() コールの ids パラメーターとして渡された ID[] 配列に対応します。たとえば、DeleteResult 配列の最初のインデックスで返されたオブジェクトは、ID[] 配列の最初のインデックスで指定されたオブジェクトに一致します。

DeleteResult 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	削除しようとした sObject の ID。ID についての詳細は、「ID データ型」を参照してください。
success	boolean	オブジェクトの delete() コールが成功したかどうかを示します (成功した場合は true、失敗した場合は false)。
errors	Error[]	delete() コールで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 Error オブジェクトの配列。

deleteByExample()

削除対象を指定するテンプレートとして sObject を使用して組織から Big Object データを削除する場合は、`deleteByExample()` を使用します。sObject テンプレートと一致する Big Object 内のすべてのデータが削除されます。

構文

```
DeleteByExampleResult[] = connection.deleteByExample(sObject[] sObjects);
```

ルールとガイドライン

データを削除する際は、次のルールやガイドラインを考慮する必要があります。

- 指定したオブジェクト内の個別のオブジェクトを削除するには、クライアントアプリケーションは、実行に十分なアクセス権限でログインする必要があります。詳細は、「データアクセスに影響する要素」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は、いずれかのオブジェクト種別が Salesforce の [設定] 領域の機能に関連していると、1 回のコールで削除することはできません。唯一の例外は次のオブジェクトです。
  - カスタム設定のオブジェクト。カスタムオブジェクトに類似しています。詳細は、Salesforce ヘルプの「カスタム設定の作成」を参照してください。
  - GroupMember
  - Group
  - User

データ削除の基本手順

データは、次の基本手順に従って削除します。

1. Big Object のインデックスを構成するすべての項目を使用して sObject を定義します。
2. 各項目の値を指定します。
3. `deleteByExample()` をコールし、作成した sObject を渡します。
4. `DeleteByExampleResult[]` の結果を処理してレコードの削除が成功したかどうかを確認します。

メモ: 成功した `deleteByExample()` 処理を繰り返すと、データがすでに削除済みであっても、結果は成功になります。

サンプルコード — カスタム Big Object

このサンプルでは、カスタム Big Object 内のレコードを削除する方法を示します。この例では、`Account__c`、`Game_Platform__c`、`Play_Date__c` がカスタム Big Object のインデックスの一部です。`Account__c` が

"001d000000Ky3xIAB"、Game_Platform__c が "iOS"、Play_Date__c が "2017-11-28T19:13:36.000z" のすべての行が削除されます。

```
public static void main(String[] args) {
    try{
        //Declare an sObject that has the values to delete
        sObject[] sObjectsToDelete = new sObject[1];
        sObject[] customerBO = new sObject();
        customerBO.setType("Customer_Interaction__b");
        customerBO.setField("Account__c","001d000000Ky3xIAB");
        customerBO.setField("Game_Platform__c","iOS");
        customerBO.setField("Play_Date__c","2017-11-28T19:13:36.000z");
        sObjectsToDelete[0] = customerBO;

        DeleteByExampleResult[] result = connection.deleteByExample(sObjectsToDelete);
    }
}
```

サンプルコード — 項目監査履歴

このサンプルでは、FieldHistoryArchive のレコードを削除する方法を示します。指定された条件を持つすべての行が削除されます。

```
public static void main(String[] args) {
    try{
        //Declare an sObject that has the values to delete
        sObject[] sObjectsToDelete = new sObject[2];
        sObject[] fieldHistoryArchive_1 = new sObject();
        fieldHistoryArchive_1.setType("FieldHistoryArchive");
        fieldHistoryArchive_1.setField("FieldHistoryType","Account");
        fieldHistoryArchive_1.setField("ParentId","001d000000Ky3xIAB");
        fieldHistoryArchive_1.setField("CreatedDate","2017-11-28T19:13:36.000z");
        fieldHistoryArchive_1.setField("HistoryId","017D000000ESURXIA5");
        sObjectsToDelete[0] = fieldHistoryArchive_1;

        sObject[] fieldHistoryArchive_2 = new sObject();
        fieldHistoryArchive_2.setType("FieldHistoryArchive");
        fieldHistoryArchive_2.setField("FieldHistoryType","Account");
        fieldHistoryArchive_2.setField("ParentId","001d000000Ky3xIAB");
        fieldHistoryArchive_2.setField("CreatedDate","2017-11-29T19:13:36.000z");
        fieldHistoryArchive_2.setField("HistoryId","017D000000ESURMIA5");
        sObjectsToDelete[1] = fieldHistoryArchive_2;

        DeleteByExampleResult[] result = connection.deleteByExample(sObjectsToDelete);
    }
}
```

引数

名前	型	説明
sObject	sObject[]	削除のテンプレートとして使用する 1 つ以上の sObject の配列。

応答

DeleteByExampleResult[]

障害

InvalidSObjectFault

UnexpectedErrorFault

DeleteByExampleResult

deleteByExample() コールは、DeleteByExampleResult オブジェクトの配列を返します。

DeleteByExampleResult 配列の各要素は、deleteByExample() コールに渡された sObject[] 配列に対応します。たとえば、DeleteByExampleResult 配列の最初のインデックスで返されるオブジェクトは、sObject[] 配列の最初のインデックスで指定される sObject に一致します。

DeleteByExampleResult 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
entity	sObject	削除しようとした sObject の詳細。
rowCount	Long	削除された行数を示します。
success	boolean	このオブジェクトの deleteByExample() コールが成功したかどうかを示します (成功した場合は true、失敗した場合は false)。
errors	Error[]	deleteByExample() コールで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 Error オブジェクトの配列。

emptyRecycleBin()

ごみ箱からレコードを直ちに削除します。

構文

```
EmptyRecycleBinResult[] = connection.emptyRecycleBin(ID[] ids);
```

使用方法

ごみ箱では、過去 15 日間に削除したレコードを参照および復元できます。15 日保存された後、レコードは完全に削除されます。組織では、一度に 1 ライセンスあたり最大 5,000 件のレコードをごみ箱に入れることができます。たとえば、組織に 5 つのユーザーライセンスがある場合、25,000 件のレコードをごみ箱に格納できます。組織のごみ箱が上限に達すると、Salesforce によって、ごみ箱に入れられてから 2 時間以上経過しているレコードが古い方から順に自動で削除されます。

大量のレコードをごみ箱に入れ、そのレコードを `undelete()` で復元する必要がないとわかっている場合に、Salesforce プロセスがレコードを削除する前にごみ箱から削除できます。たとえば、テストで大量のレコードを読み込む場合、または大量の `create()` コールと `delete()` コールを続けて行う場合などにこのコールを使用できます。

ルールとガイドライン

ごみ箱を空にする際は、次のルールやガイドラインを考慮する必要があります。

- ログインユーザーは、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。ログインユーザーが「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を照会して削除できます。
- バージョン 10.0 以降で利用できます。
- レコードの最大数は 200 件です。
- カスケード削除できるレコードの ID は含めないようにしてください。エラーが発生します。
- このコールを使用してレコードを削除すると、`undelete()` で復元することはできません。
- このコールを使用してごみ箱からレコードを削除した後もしばらくは `queryAll()` を使用してクエリを実行できます。一般的には 24 時間ですが、前後することがあります。

サンプルコード — Java

このサンプルでは、ごみ箱を空にする方法を示します。ごみ箱から削除するレコードの ID を含む配列を受け取ります。`emptyRecycleBin()` をコールし、IDの配列を渡します。次に、結果を反復処理し、削除されたレコードの ID または失敗したレコードの最初のエラーをコンソールに書き込みます。

```
public void emptyRecycleBin(String[] ids) {
   try {
      EmptyRecycleBinResult[] emptyRecycleBinResults = connection
            .emptyRecycleBin(ids);
      for (int i = 0; i < emptyRecycleBinResults.length; i++) {
         EmptyRecycleBinResult emptyRecycleBinResult = emptyRecycleBinResults[i];
         if (emptyRecycleBinResult.isSuccess()) {
            System.out.println("Recycled ID: "
                  + emptyRecycleBinResult.getId());
         } else {
            Error[] errors = emptyRecycleBinResult.getErrors();
            if (errors.length > 0) {
               System.out
                     .println("Error code: " + errors[0].getStatusCode());
               System.out
```

```
                    .println("Error message: " + errors[0].getMessage());
            }
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、ごみ箱を空にする方法を示します。ごみ箱から削除するレコードの ID を含む配列を受け取ります。`emptyRecycleBin()` をコールし、IDの配列を渡します。次に、結果を反復処理し、削除されたレコードの ID または失敗したレコードの最初のエラーをコンソールに書き込みます。

```
public void emptyRecycleBin(String[] ids)
{
   try
   {
      EmptyRecycleBinResult[] emptyRecycleBinResults =
         binding.emptyRecycleBin(ids);
      for (int i = 0; i < emptyRecycleBinResults.Length; i++)
      {
         EmptyRecycleBinResult emptyRecycleBinResult = emptyRecycleBinResults[i];
         if (emptyRecycleBinResult.success)
         {
            Console.WriteLine("Recycled ID: "
                  + emptyRecycleBinResult.id);
         }
         else
         {
            Error[] errors = emptyRecycleBinResult.errors;
            if (errors.Length > 0)
            {
               Console.WriteLine("Error code: " + errors[0].statusCode);
               Console.WriteLine("Error message: " + errors[0].message);
            }
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

引数

名前	型	説明
ids	ID[]	ごみ箱から削除するレコードに関連付けられた 1 つ以上の ID の配列。レコードの最大数は 200 件です。

応答

EmptyRecycleBinResult

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

- delete()
- undelete()

EmptyRecycleBinResult

emptyRecycleBin() コールは、EmptyRecycleBinResult オブジェクトの配列を返します。配列の各要素は、emptyRecycleBin() コールのパラメーターとして渡された ID[] 配列の要素に対応しています。たとえば、EmptyRecycleBinResult 配列の最初のインデックスで返されたオブジェクトは、ID[] 配列の最初のインデックスで指定されたオブジェクトに一致します。

EmptyRecycleBinResult 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	ごみ箱から削除しようとしている sObject の ID。ID についての詳細は、「ID データ型」を参照してください。
isSuccess	boolean	コールが成功したかどうかを示します(成功した場合は true、失敗した場合は false)。
errors	Error[]	コールで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 Error オブジェクトの配列。

executeListView()

リストビューの SOQL クエリを実行して、リストビューからデータ、表示ラベル、およびアクションを取得します。

構文

```
ExecuteListViewResult result = connection.executeListView(ExecuteListViewResult request);
```

使用方法

executeListView() コールは、ExecuteListViewRequest オブジェクトを取り込み、リストビューの SOQL クエリを実行し、ExecuteListViewResult オブジェクトでその結果のデータと表示情報を返します。このコールは、API バージョン 32.0 以降で使用できます。

サンプルコード — Java

```
private void example(ApiProtocol protocol, AppVersion version) throws Exception {

        // Get the list results via the list view API
        EnterpriseConnection connection =
makeClient(getUserUtil().getUserWithModifyAllData(), AppVersion.VERSION_190,
                getName());
        ExecuteListViewRequest request = new ExecuteListViewRequest();
        request.setSobjectType("Account");
        request.setDeveloperNameOrId(listViews[0].getId());
        request.setLimit(50000);

        com.sforce.soap.enterprise.ExecuteListViewResult result =
connection.executeListView(request);
    }
```

引数

名前	型	説明
request	ExecuteListViewRequest	リストビューとその結果の制限、オフセット、および順序を指定するオブジェクト。

応答

ExecuteListViewResult オブジェクト。

ExecuteListViewRequest

リストビューからデータ、表示ラベル、およびアクションを取得するには、executeListView() で ExecuteListViewRequest オブジェクトを使用します。

ExecuteListViewRequest 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
developerNameOrId	string	リストビューの ID または完全修飾開発者名。
limit	int	返すレコードの最大数。デフォルト: 25
offset	int	スキップするレコード数。デフォルト: 0
orderBy	ListViewOrderBy[]	レコードを返す順序。
sobjectType	string	リストビューの sObject の API 名。

ExecuteListViewResult

プログラムで取得したリストビューデータが含まれます。

executeListViewResult オブジェクトを取得するには、executeListView() コールを使用します。executeListViewResult 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
columns	ListViewColumn[]	リストビュー内の列の配列。
developerName	string	リストビューの完全修飾開発者名。
done	boolean	true の場合、すべてのレコードが返されたことを示します。
id	ID	リストビューの ID。
label	string	リストビューの表示ラベル。
records	ListViewRecord[]	リストビュークエリと一致するレコードの配列。
size	int	リストビュークエリから返されるレコードの数。

ListViewColumn

1 つのリストビュー列に関するメタデータが含まれます。

ListViewColumn オブジェクトは describeSoqlListViews() および executeListView() コールによって返されます。ListViewColumn には次のプロパティがあります。

名前	型	説明
ascendingLabel	string	列を昇順で並び替えるための、ローカライズされた種別固有の表示ラベル。たとえば、テキスト項目の場合は「A-Z」、数値項目の場合は「Low to High」などになります。列が並び替えできない場合は、null に設定します。
descendingLabel	string	列を昇順で並び替えるための、ローカライズされた種別固有の表示ラベル。たとえば、テキスト項目の場合は「Z-A」、数値項目の

名前	型	説明
		場合は「HightoLow」などになります。列が並び替えできない場合は、null に設定します。
`fieldNameOrPath`	string	列の項目名または SOQL 項目パス。
`hidden`	boolean	trueの場合、列は表示されず、他の列の表示や他のクライアント側ロジックをサポートするためにのみ存在していることを示します。
`label`	string	列のローカライズされた表示ラベル。
`searchable`	boolean	列が検索可能かどうか。
`selectListItem`	string	列の SOQL SELECT 項目。この項目は、表示形式の違いにより、項目名またはパスとは異なる場合があります (選択リストの toLabel など)。
`sortDirection`	orderByDirection	列挙値。列が並び替え可能な場合は次のいずれかになります。 • 昇順 • 降順 列が並び替えできない場合は、null に設定します。
`sortIndex`	int	複数レベルの並び替え内での列の位置を示すゼロベースのインデックス。または、レコードが列で並び替えられない場合は null。
`sortable`	boolean	列が並び替え可能かどうか。並び替え可能な場合は、ExecuteListView orderBy パラメーターで参照されることがあります。
`type`	FieldType	列のデータ型。

ListViewRecord

リストビューの 1 行を表します。

`ListViewRecord` オブジェクトは、`ExecuteListViewResult` オブジェクトのメンバーであり、次のプロパティがあります。

名前	型	説明
`columns`	ListViewRecordColumn[]	レコードの列とその値。レコードデータ列は、メタデータおよび describe 列と同じ順序で返されます。`ExecuteListViewResult.getRecords()[0].getColumns[index]` を使用して取得されるデータ列の場合、対応する describe 列は `ExecuteListViewResult.getColumns[index]` で取得できます。

ListViewRecordColumn

リストビュー内のある行の 1 つのセルを表します。

`ListViewRecordColumn` オブジェクトは `ListViewRecord` 行の 1 つのセル (列) であり、次のプロパティがあります。

名前	型	説明
`fieldNameOrPath`	string	列の項目名または SOQL 項目パス。
`value`	string	特定の列のレコードのコンテンツ。必要に応じてローカライズされ、値がない場合は null になります。

`findDuplicates()`

ルールに基づいて、重複レコードの検索を実行します。

入力は sObject の配列で、それぞれが検索する値と重複ルールを示すオブジェクトの種別を指定します。重複ルールを示すオブジェクトごとに、検出された重複が出力で識別されます。`findDuplicates()` はルールを値に適用し、検索を実行します。sObject ごとに、検出された重複が出力で識別されます。

構文

```
FindDuplicatesResult[] duplicateResults =
    connection.findDuplicates(SObject[] inputSObjectArray);
```

使用方法

オブジェクトに関連付けられた重複ルールを各 sObject で指定された値に適用するには、`findDuplicates()` を使用します。各 sObject には、オブジェクトに対応する種別もあります。

`findDuplicates()` では、sObject と同じ種別のオブジェクトの重複ルールを使用します。たとえば、sObject の種別が取引先の場合、`findDuplicates()` は Account オブジェクトに関連付けられた重複ルールを使用します。

入力 `sObject` ごとに、`findDuplicates()` は FindDuplicatesResult オブジェクトを出力配列に追加します。

メモ:

- 入力配列内のすべての sObject 要素は同じ種別である必要があり、その種別は重複ルールをサポートするオブジェクト種別に対応する必要があります。
- 入力配列は 50 要素までに制限されます。この制限を超えた場合、SOAP コールは次の項目を含む API 障害要素を返します。
  - ExceptionCode: `LIMIT_EXCEEDED`
  - exceptionMessage: `Configuration error: The number of records to check is greater than the permitted batch size.`

sObject コントロール一致で指定された値。値にはレコード ID、項目の対応付け、またはその両方が含まれる可能性があります。指定された値によって、`findDuplicates()` の動作が決まります。

レコード ID のみ
`findDuplicates()` は、重複ルールで定義されたオブジェクトで同じ ID を持つ既存のレコードを検索します。次に、そのレコードから値を読み込み、それらの値に基づいて重複があるか検索します。

項目の対応付けのみ
`findDuplicates()` は、対応付けから値を読み込み、それらの値に基づいて重複があるか検索します。

レコード ID と項目の対応付け
`findDuplicates()` は、重複ルールで定義されたオブジェクトで同じ ID を持つ既存のレコードを検索します。対応付けで指定されていないレコードから値を読み込み、次に対応付けから値を読み込みます。値の結合結果に基づいて、`findDuplicates()` は重複があるか検索します。

`findDuplicates()` の出力は、入力配列と要素数と順序が同じ FindDuplicatesResult オブジェクトの配列です。重複レコードがある場合、出力オブジェクトには、そのレコード ID がカプセル化されます。必要に応じて、出力オブジェクトには重複レコードからの値も含まれます。

各 FindDuplicatesResult 要素には、DuplicateResult オブジェクトが含まれます。`findDuplicates()` で sObject の重複が検出されなかった場合、DuplicateResult の `duplicateRule` 項目には、`findDuplicates()` が適用した重複ルールの名前が含まれますが、`matchResults` 配列は空です。

DuplicateRuleHeader の `includeRecordDetails` フラグが `false` に設定されている場合、`findDuplicates()` は一致するレコードのレコード ID のみを返します。それ以外の場合、`findDuplicates()` は対象オブジェクトに関連付けられているプライマリ CompactLayout で指定されたすべての項目を返します。

基本的な使用手順

1. 使用する重複ルールを持つオブジェクトに対応する種別の sObject オブジェクトを 1 つ以上作成します。
2. 各 sObject で、レコード ID または項目の対応付け (または両方) を指定して、オブジェクトのレコードと比較します。
3. 必要な出力を制御するために、DuplicateRuleHeader を設定します。

サンプル

次の Java サンプルは、標準のリード重複ルールを使用してリードの重複を検索する方法を示します。

```
package wsc;

import com.sforce.soap.partner.*;
import com.sforce.soap.partner.Error;
import com.sforce.soap.partner.sobject.SObject;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;

public class Main {

  private static final String USERNAME = "YOUR-USERNAME";
  private static final String PASSWORD = "YOUR-PASSWORD&SECURITY-TOKEN";
```

```
  private static PartnerConnection connection = null;

  public static void main(String[] args) throws ConnectionException {

    // Create the configuration for the partner connection
    ConnectorConfig config = new ConnectorConfig();
    config.setUsername(USERNAME);
    config.setPassword(PASSWORD);

    // Initialize the connection
    connection = new PartnerConnection(config);

    SObject[] inputSObjectArray = new SObject[1];
    // Instantiate an empty Java SObject
    SObject searchCriteria = new SObject();
    // Set its type to Lead. This tells findDuplicates() to use the duplicate rules
    // for Lead
    searchCriteria.setType("Lead");
    /*
     * Set the necessary fields for matching, based on the standard matching rules for
Lead (Search
     * help.salesforce.com for "Standard Contact and Lead Matching Rule" to see the rules).

     */
    searchCriteria.setField("FirstName", "Marc");
    searchCriteria.setField("LastName", "Benioff");
    searchCriteria.setField("Company", "Salesforce.com Inc");
    searchCriteria.setField("Title", "CEO");
    searchCriteria.setField("Email", "ceo@salesforce.com");
    // Add the sObject to the input array
    inputSObjectArray[0] = searchCriteria;
    /*
     * By default, findDuplicates() returns only record IDs. To return additional values,
 set the second parameter
     * to true.
     */
    connection.setDuplicateRuleHeader(
        /*
         * @param allowSave - Not Applicable for this API call
         */
        false,
        /* @param includeRecordDetails */
        false,
        /*
         * @param runAsCurrentUser - Not Applicable for this API call
         */
        false);

    // Invoke findDuplicates() to find duplicates based on the information in the
    // SObject array
    FindDuplicatesResult[] callResults = connection.findDuplicates(inputSObjectArray);

    // Iterate through the results
    // For each SObject in the input array, get the duplicate results
```

```
    for (FindDuplicatesResult findDupeResult : callResults) {
      // If errors were found for this SObject, print them out
      if (!findDupeResult.isSuccess()) {
        for (Error findDupError : findDupeResult.getErrors()) {
          System.out.println("FindDuplicatesRule errors detected: " +
findDupError.getMessage());
        }
      } else {
        /*
         * Get the DuplicateResult object array for the result. Each element in the array
 represents the result
         * of testing one duplicate rule for the SObject. Process each DuplicateResult.
         */
        for (DuplicateResult dupeResult : findDupeResult.getDuplicateResults()) {
          System.out.println("Duplicate rule: " + dupeResult.getDuplicateRule());
          // Print out the name of the object associated with the duplicate
          // rule
          System.out.println("Source of this duplicate rule is: " +
dupeResult.getDuplicateRuleEntityType());
          for (MatchResult matchResult : dupeResult.getMatchResults()) {
            if (!matchResult.isSuccess()) {
              for (Error e : matchResult.getErrors()) {
                System.out.println("Errors detected: " + e.getMessage());
              }
            } else {
              System.out.println("Matching rule is: " + matchResult.getRule());
              System.out.println("Object type for this matching rule is: " +
matchResult.getEntityType());
              for (MatchRecord matchRecord : matchResult.getMatchRecords()) {
                System.out.println("Duplicate record ID: " +
matchRecord.getRecord().getId());
              }
            }
          }
        }
      }
    }
  }
}
```

引数

名前	型	説明
sObjects	sObject の配列	必須。検索する値が含まれる sObject オブジェクトのリスト。

応答

FindDuplicatesResult オブジェクトの配列。

FindDuplicatesResult

入力配列内の 1 つの sObject の重複検索の結果を表します。sObject に関連付けられたオブジェクトには複数の重複ルールを設定できるため、`FindDuplicatesResult` には DuplicateResult オブジェクトの配列が含まれます。

項目

項目名	項目の型	説明
`duplicateResults`	DuplicateResult オブジェクトの配列	`findDuplicates()` によって 1 つの sObject に適用された各重複ルールの結果。
`errors`	Error オブジェクトの配列	`findDuplicates()` で発生したエラーの配列が含まれます。
`success`	boolean	`findDuplicates()` でエラーが発生しなかった場合、この項目は `true` に設定されます。

障害

`InvalidFieldFault`

`InvalidSObjectFault`

`UnexpectedErrorFault`

findDuplicatesByIds()

`findDuplicatesByIds()` は、ルールに基づいて、重複レコードの検索を実行します。入力はレコード ID の配列で、各要素は重複を検索するレコードを指定します。レコード ID ごとに、検出された重複が出力で識別されます。

構文

```
FindDuplicatesResult[] duplicateResults =
    connection.findDuplicatesByIds(Id[] inputIdArray);
```

使用方法

オブジェクトに関連付けられた重複ルールを、レコード ID で表されたレコードに適用するには、`findDuplicatesByIds()` を使用します。

`findDuplicatesByIds()` は、入力レコード ID と同じ種別のオブジェクトの重複ルールを使用します。たとえば、レコード ID が取引先を表す場合、`findDuplicatesByIds()` は Account オブジェクトに関連付けられた重複ルールを使用します。

入力 ID ごとに、`findDuplicatesByIds()` はオブジェクトを出力配列に追加します。

メモ:

- 入力配列内のすべてのレコード ID は同じオブジェクト種別である必要があり、その種別は重複ルールをサポートするオブジェクト種別に対応する必要があります。
- 入力配列は 50 要素までに制限されます。この制限を超えた場合、SOAP コールは次の項目を含む API 障害要素を返します。
  - ExceptionCode: `LIMIT_EXCEEDED`
  - exceptionMessage: `Configuration error: The number of records to check is greater than the permitted batch size.`

照合は、入力レコード ID で指定された値によって制御されます。値にはレコード ID のみを含めることができます。

`findDuplicatesByIds()` は、重複ルールで定義されたオブジェクトで同じ ID を持つ既存のレコードを検索します。次に、そのレコードから値を読み込み、それらの値に基づいて重複があるか検索します。

`findDuplicatesByIds()` の出力は、入力配列と同じ要素数と順序のオブジェクトの配列です。出力オブジェクトでは、重複レコードのレコード ID がカプセル化されます。必要に応じて、出力オブジェクトには重複レコードからの値も含まれます。

各要素に DuplicateResult オブジェクトが含まれます。`findDuplicatesByIds()` で sObject の重複が検出されなかった場合、DuplicateResult の `duplicateRule` 項目には、`findDuplicatesByIds()` が適用した重複ルールの名前が含まれますが、`matchResults` 配列は空です。

DuplicateRuleHeader の `includeRecordDetails` フラグが `false` に設定されている場合、`findDuplicatesByIds()` は一致するレコードのレコード ID のみを返します。それ以外の場合、`findDuplicatesByIds()` は対象オブジェクトに関連付けられているプライマリ CompactLayout で指定されたすべての項目を返します。

基本的な使用手順

1. 使用する重複ルールを持つオブジェクトに対応する ID オブジェクトを 1 つ以上作成します。
2. レコード ID を指定して、オブジェクトのレコードと比較します。
3. 必要な出力を制御するために、DuplicateRuleHeader を設定します。

サンプル

次の Java サンプルは、標準のリード重複ルールを使用してリードの重複を検索する方法を示します。

```
package wsc;

import com.sforce.soap.partner.*;
import com.sforce.soap.partner.Error;
import com.sforce.soap.partner.sobject.SObject;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;
```

```
public class Main {

  private static final String USERNAME = "YOUR-USERNAME";
  private static final String PASSWORD = "YOUR-PASSWORD&SECURITY-TOKEN";
  private static PartnerConnection connection = null;

  public static void main(String[] args) throws ConnectionException {

    // Create the configuration for the partner connection
    ConnectorConfig config = new ConnectorConfig();
    config.setUsername(USERNAME);
    config.setPassword(PASSWORD);

    // Initialize the connection
    connection = new PartnerConnection(config);

    SObject[] objectsToSearch = new SObject[2];
    String[] inputIds = new String[2];
    // Instantiate an empty Java SObject
    SObject searchCriteria = new SObject();
    // Set its type to Lead. This tells findDuplicatesByIds() to use the duplicate rules
    // for Lead
    searchCriteria.setType("Lead");
    /*
     * Set the necessary fields for matching, based on the standard matching rules for
Lead
     * (Search help.salesforce.com for "Standard Contact and Lead Matching Rule" to see
the
     * rules).
     */
    searchCriteria.setField("FirstName", "Marc");
    searchCriteria.setField("LastName", "Benioff");
    searchCriteria.setField("Company", "Salesforce.com Inc");
    searchCriteria.setField("Title", "CEO");
    searchCriteria.setField("Email", "ceo@salesforce.com");
    // Add the sObjects to the input array
    objectsToSearch[0] = searchCriteria;
    objectsToSearch[1] = searchCriteria;

    SaveResult[] saveResults = connection.create(objectsToSearch);

    for (int i = 0; i < saveResults.length; ++i) {
      if (saveResults[i].isSuccess()) {
        System.out.println("Successfully created ID: " + saveResults[i].getId());
        inputIds[i] = saveResults[i].getId();
      } else {
        System.out.println("Error: could not create SObject.");
        System.out.println("The error reported was: " +
          saveResults[i].getErrors()[0].getMessage() + "\n");
      }
    }
    /*
     * By default, findDuplicatesByIds() returns only record IDs. To return additional
values,
```

```
     * set the second parameter to true.
     */
    connection.setDuplicateRuleHeader(
        /*
         * @param allowSave - Not Applicable for this API call
         */
        false,
        /* @param includeRecordDetails */
        false,
        /*
         * @param runAsCurrentUser - Not Applicable for this API call
         */
        false);

    // Invoke findDuplicatesByIds() to find duplicates based on the information in the
    // SObject array
    FindDuplicatesResult[] callResults = connection.findDuplicatesByIds(inputIds);

    // Iterate through the results
    /* For each Id in the input array, get the duplicate results. There could be more
matches
     * depending on the data in the organization.
     */
    for (FindDuplicatesResult findDupeResult : callResults) {
      // If errors were found for this Id, print them out
      if (!findDupeResult.isSuccess()) {
        for (Error findDupError : findDupeResult.getErrors()) {
          System.out.println("FindDuplicatesRule errors detected: " +
findDupError.getMessage());
        }
      } else {
        /*
         * Get the DuplicateResult object array for the result. Each element in the array
 represents
        * the result of testing one duplicate rule for the Id. Process each DuplicateResult.

         */
        for (DuplicateResult dupeResult : findDupeResult.getDuplicateResults()) {
          System.out.println("Duplicate rule: " + dupeResult.getDuplicateRule());
          // Print out the name of the object associated with the duplicate
          // rule
          System.out.println("Source of this duplicate rule is: " +
            dupeResult.getDuplicateRuleEntityType());
          for (MatchResult matchResult : dupeResult.getMatchResults()) {
            if (!matchResult.isSuccess()) {
              for (Error e : matchResult.getErrors()) {
                System.out.println("Errors detected: " + e.getMessage());
              }
            } else {
              System.out.println("Matching rule is: " + matchResult.getRule());
              System.out.println("Object type for this matching rule is: " +
matchResult.getEntityType());
              for (MatchRecord matchRecord : matchResult.getMatchRecords()) {
                System.out.println("Duplicate record ID: " +
```

```
matchRecord.getRecord().getId());
                }
              }
            }
          }
        }
      }
    }
}
```

引数

名前	種別	説明
IDs	ID の配列	必須。検索する値が含まれる ID のリスト。

返答

FindDuplicatesResult オブジェクトの配列。

FindDuplicatesResult

入力配列内の 1 つの ID の重複検索の結果を表します。sObject に関連付けられたオブジェクトには複数の重複ルールを設定できるため、FindDuplicatesResult には DuplicateResult オブジェクトの配列が含まれます。

項目

項目名	項目の型	説明
duplicateResults	DuplicateResult オブジェクトの配列	findDuplicatesByIds() によって 1 つの sObject に適用された各重複ルールの結果。
errors	Error オブジェクトの配列	findDuplicatesByIds() で発生したエラーの配列が含まれます。
success	boolean	findDuplicatesByIds() でエラーが発生しなかった場合、この項目は true に設定されます。

障害

InvalidSObjectFault

UnexpectedErrorFault

InvalidFieldFault

getDeleted()

指定されたオブジェクトについて、特定の期間内に削除された個々のレコードのリストを取得します。

構文

```
GetDeletedResult = connection.getDeleted(string sObjectType, dateTime startDate, dateTime
 EndDate);
```

使用方法

データ複製アプリケーションに getDeleted() を使用すると、指定した期間内に組織のデータから削除されたレコードのリストを取得できます。getDeleted() コールは GetDeletedResult オブジェクトを取得します。このオブジェクトに含まれる DeletedRecord オブジェクトの配列には、削除された各レコードの ID と、削除された日時(協定世界時(UTC)タイムゾーン)が入っています。クライアントアプリケーションで getDeleted() を使用する前に、「データ複製」に目を通してください。(ID についての詳細は、「ID データ型」を参照してください)。

リリース 8.0 現在、getDeleted() コールは、ユーザーの共有モデルに従います。

ルールとガイドライン

削除されたレコードを複製する場合、次のルールとガイドラインを考慮してください。

- startDate は、endDate の値より 1 分以上過去の日時でなければなりません。startDate は、endDate 以降の値にすることはできません。そのような場合、API は INVALID_REPLICATION_DATE エラーを返します。
- レコードは、ユーザーがアクセスした場合のみ返されます。
- 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 (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)。getDeleted() コールを実行する前に消去処理を実行すると、INVALID_REPLICATION_DATE エラーが返されます。
- latestDateCovered が endDate より前の値の場合、コールは失敗し、latestDateCovered の値と共に INVALID_REPLICATION_DATE エラーが返されます。
- 削除されたレコードは、getDeleted() からアクセス可能な削除ログに出力されます。2 時間ごとに実行されるバックグラウンドプロセスは、削除ログのレコード数が制限を超えた場合、削除ログに書き込まれてから 2 時間以上経過したレコードを消去します。最も古いレコードから順に、削除ログが制限を下回るまで消去を行います。この処理は、大量の削除ログによる Salesforce のパフォーマンス上の問題を防ぎます。制限値は次の数式を使用して算出します。

  ```
  5000 * number of licenses in the organization
  ```

  たとえば、1,000 ライセンスを所有する組織では、削除ログのレコード数が 5,000,000 (5 百万) レコード以上になると消去処理を開始します。getDeleted() コールを実行する前に消去処理を実行すると、INVALID_REPLICATION_DATE エラーが返されます。この例外が発生した場合は、テーブル全体に対するプル処理を実行してください。

- 大量のレコードを削除する場合は、`getDeleted()` コールがすべてのレコードを返すようにするため、データの複製が 2 時間おきよりも短い間隔で実行されます。
- クライアントアプリケーションは、定期的にデータ変更のポーリングを行います。ポーリング時の重要な検討事項については、「変更のポーリング」を参照してください。
- 特定のオブジェクトのレコードは、APIからは複製できません。`getDeleted()` コールからレコードを複製するには、そのオブジェクトが複製可能に設定されている (`replicateable` が `true`) 必要があります。指定されたオブジェクトが複製可能かどうかを確認するには、クライアントアプリケーションでオブジェクトに対する `describeSObjects()` コールを実行し、`replicateable` プロパティを確認します。
- 時間データの処理方法は、開発ツールごとに異なります。開発ツールによってはローカル時間を表示するものも、協定世界時(UTC)を表示するものもあります。開発ツールごとの時間の処理方法はツールのドキュメントを参照してください。
- 履歴オブジェクトの `getDeleted()` をコールする場合、このコールでは、ユーザーが指定した履歴オブジェクトだけではなく、すべての履歴オブジェクトの特定の日付範囲の間に削除されたレコードを返します。たとえば、AccountHistory の `getDeleted()` をコールすると、AccountHistory、ContactHistory などの特定の日付範囲で削除されたレコードを取得します。ただし、OpportunityHistory に対する `getDeleted()` コールは削除された OpportunityHistory レコードのみを返し、その他の関連する削除された履歴オブジェクトは返しません。

削除されたオブジェクトを複製するための基本手順

削除されたレコードは、オブジェクトごとに次の基本手順を使用して複製できます。

1. (必要に応じて実行) 直近の複製要求以降、オブジェクトの構造が変更されたかどうかを確認します (「オブジェクトの構造変更のチェック」を参照)。
2. `getDeleted()` をコールし、削除されたレコードのオブジェクトと関連する期間を渡します。
3. DeleteResult オブジェクトで、返された `DeletedRecord` オブジェクトの配列を反復処理します。この配列には、削除された各レコードの ID と削除日 (協定世界時 (UTC) タイムゾーン) が含まれます。
4. ローカルデータに対して適切なアクションを実行し、削除されたレコードを消去するか、削除済みのフラグを設定します。
5. 必要に応じて、latestDateCovered の値を使用して、今後の参照のために要求の期間を保存します。

クライアントアプリケーションは、データ複製処理に関連付けられた他のタスクを実行することもあります。たとえば、商談がクローズしている場合、クライアントアプリケーションは新たな収益レポートを実行することが考えられます。同様にタスクが完了している場合、プロセスは他のシステムのログに記述します。

サンプルコード — Java

このサンプルでは、`getDeleted()` をコールして、直近の60分間に削除されたすべての取引先を取得します。次に、ID および返された各取引先が削除された日付をコンソールに書き込みます。

```
public void getDeletedRecords() {
   try {
      GregorianCalendar endTime = (GregorianCalendar)
         connection.getServerTimestamp().getTimestamp();
      GregorianCalendar startTime = (GregorianCalendar) endTime.clone();
```

```
      // Subtract 60 minutes from the server time so that we have
      // a valid time frame.
      startTime.add(GregorianCalendar.MINUTE, -60);
      System.out.println("Checking deletes at or after: "
            + startTime.getTime().toString());

      // Get records deleted during the specified time frame.
      GetDeletedResult gdResult = connection.getDeleted("Account",
            startTime, endTime);

      // Check the number of records contained in the results,
      // to check if something was deleted in the 60 minute span.
      DeletedRecord[] deletedRecords = gdResult.getDeletedRecords();
      if (deletedRecords != null && deletedRecords.length > 0) {
         for (int i = 0; i < deletedRecords.length; i++) {
            DeletedRecord dr = deletedRecords[i];
            System.out.println(dr.getId() + " was deleted on "
                  + dr.getDeletedDate().getTime().toString());
         }
      } else {
         System.out.println("No deletions of Account records in "
               + "the last 60 minutes.");
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、getDeleted() をコールして、直近の60分間に削除されたすべての取引先を取得します。次に、ID および返された各取引先が削除された日付をコンソールに書き込みます。

```
public void getDeletedRecords()
{
   try
   {
      DateTime endTime = binding.getServerTimestamp().timestamp;
      // Subtract 60 minutes from the server time so that we have
      // a valid time frame.
      DateTime startTime = endTime.AddMinutes(-60);
      Console.WriteLine("Checking deletes at or after: "
            + startTime.ToLocalTime().ToString());

      // Get records deleted during the specified time frame.
      GetDeletedResult gdResult = binding.getDeleted("Account",
            startTime, endTime);

      // Check the number of records contained in the results,
      // to check if something was deleted in the 60 minute span.
      DeletedRecord[] deletedRecords = gdResult.deletedRecords;
      if (deletedRecords != null && deletedRecords.Length > 0)
      {
```

```
            for (int i = 0; i < deletedRecords.Length; i++)
            {
               DeletedRecord dr = deletedRecords[i];
               Console.WriteLine(dr.id + " was deleted on "
                     + dr.deletedDate.ToLocalTime().ToString());
            }
         }
         else
         {
            Console.WriteLine("No deletions of Account records in "
                  + "the last 60 minutes.");
         }
      }
      catch (SoapException e)
      {
         Console.WriteLine("An unexpected error has occurred: " +
                                    e.Message + "\n" + e.StackTrace);
      }
   }
```

引数

名前	型	説明
sObjectTypeEntityType	string	**オブジェクト種別。指定された値は、組織で有効なオブジェクトである必要があります。**「sObject」**を参照してください。**
startDate	dateTime	**データを取得する期間の開始日時 (ローカル時間ではなく協定世界時** (UTC))。API **は、指定された** dateTime **値に含まれる秒の部分を切り捨てます (たとえば、**12:30:15 **は** 12:30:00 UTC **となります)。**
endDate	dateTime	**データを取得する期間の終了日時 (ローカル時間ではなく協定世界時** (UTC))。API **は、指定された** dateTime **値に含まれる秒の部分を切り捨てます (たとえば、**12:35:15 **は** 12:35:00 UTC **となります)。**

制限

getDeleted() **コールで返される結果が多すぎる場合は、レスポンスで例外** EXCEEDED_ID_LIMIT **が返されます。返すことができるレコード数については、**「SOAP API **コールの制限**」**を参照してください。**

応答

GetDeletedResult

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

- データ複製
- API コールの基礎

GetDeletedResult

getDeleted() コールは、deletedRecords レコードを含む GetDeletedResult オブジェクトを返します。

GetDeletedResult オブジェクトには、次の項目があります。

名前	型	説明
earliestDateAvailable	dateTime	getDeleted() コールのオブジェクト種別について、最後に物理的に削除されたオブジェクトのタイムスタンプ(ローカル時間ではなく協定世界時 (UTC) タイムゾーン)。この値が endDate より前の値の場合、コールは失敗するため、他のデータ複製を実行する前にデータの再同期が必要となります。
deletedRecords[]	deletedRecords=	getDeleted() コールで指定された開始日と終了日を満たす削除されたレコードの配列。
latestDateCovered	dateTime	getDeleted() コールの対象となる最終日のタイムスタンプ(ローカル時間ではなく協定世界時 (UTC) タイムゾーン)。値がある場合は、endDate 以前のタイムスタンプになります。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。

deletedRecords

GetDeletedResult オブジェクトには、deletedRecords レコードの配列が含まれます。

deletedRecords レコードには次のプロパティが含まれます。

名前	型	説明
deletedDate	dateTime	このレコードが削除された日時(ローカル時間ではなく協定世界時 (UTC))。この情報は、使用可能な場合は SystemModstamp システムを使用して取得されます。
id	ID	削除された sObject の ID。

getUpdated()

指定されたオブジェクトに対して指定された期間内に更新された(追加または変更された)個別のレコードのリストを取得します。

構文

```
GetUpdatedResult[] = connection.getUpdated(string sObjectType, dateTime startDate, dateTime
 EndDate);
```

使用方法

データ複製アプリケーションに `getUpdated()` を使用すると、指定されたオブジェクトのうち、指定された期間内に作成または更新されたオブジェクトの ID のセットを取得できます。`getUpdated()` コールは、`GetUpdatedResult` オブジェクトの配列を取得します。この配列には、作成または更新された各オブジェクトの ID と、作成または更新された日時 (協定世界時 (UTC) タイムゾーン) がそれぞれ含まれます。クライアントアプリケーションで `getUpdated()` を使用する前に、「データ複製」に目を通してください。

メモ: `getUpdated()` コールは、ログインユーザーがアクセス権を持つオブジェクトの ID のみを取得します。

ルールとガイドライン

作成または更新されたオブジェクトを複製する際は、次のルールやガイドラインを考慮する必要があります。

- `startDate` は、`endDate` の値より過去の日時でなければなりません。`startDate` は、`endDate` と同じか未来の日時であってはなりません。そのような場合、API は `INVALID_REPLICATION_DATE` エラーを返します。
- コールが実行された日から 30 日以内の結果が返されます。
- クライアントアプリケーションは、定期的にデータ変更のポーリングを行います。ポーリング時の重要な検討事項については、「変更のポーリング」を参照してください。
- クライアントアプリケーションは、適切な権限が付与されている場合、任意のオブジェクトを複製できます。たとえば、組織のすべてのデータを複製するには、クライアントアプリケーションは指定されたオブジェクトの「すべてのデータの参照」アクセス権限を持ってログインしなければなりません。同様に、オブジェクトはそのユーザーの共有ルールに含まれていなければなりません。詳細は、「データアクセスに影響する要素」を参照してください。
- 特定のオブジェクトは、APIからは複製できません。`getUpdated()` コールからオブジェクトを複製するには、オブジェクトが複製可能に設定されている (`replicateable` が `true`) 必要があります。指定されたオブジェクトが複製可能かどうかを確認するには、クライアントアプリケーションでオブジェクトに対する `describeSObjects()` コールを実行し、`replicateable` プロパティを確認します。
- Group、User、Contract、Product2 オブジェクトなど、オブジェクトによっては削除できないものがあります。ただし、これらのオブジェクトのインスタンスが Salesforce ユーザーインターフェースにはすでに表示されていない場合、管理者アクセス権限を持つユーザーのみが参照できるよう無効にすることができます。表

示されていないオブジェクトインスタンスが無効かどうかを判別するには、クライアントアプリケーションから `getUpdated()` をコールし、オブジェクトの有効フラグを確認します。

- 時間データの処理方法は、開発ツールごとに異なります。開発ツールによってはローカル時間を表示するものも、協定世界時(UTC)を表示するものもあります。開発ツールごとの時間の処理方法はツールのドキュメントを参照してください。

更新されたオブジェクトの複製の基本手順

更新されたオブジェクトの複製では、複製する各オブジェクトで次の基本手順に従います。

1. 必要に応じて、直近の複製要求以降、オブジェクトの構造が変更されたかどうかを確認します(「オブジェクトの構造変更のチェック」を参照)。
2. `getUpdated()` をコールし、データを取得するオブジェクトと期間を渡します。
3. ID の配列すべてに対して反復処理を行います。配列の各 ID 要素に対し、`retrieve()` をコールして関連するオブジェクトから必要な最新情報を取得します。その後クライアントアプリケーションは、ローカルデータに新しい行の挿入や、既存の行を最新情報で更新するなどの適切な処理を行います。
4. 必要に応じて、今後の参照のために要求のタイムスタンプを保存します。

クライアントアプリケーションは、データ複製処理に関連付けられた他のタスクを実行することもあります。たとえば、商談がクローズしている場合、クライアントアプリケーションは新たな収益レポートを実行することが考えられます。同様にタスクが完了している場合、プロセスは他のシステムのログに別の方法で記述することが考えられます。

サンプルコード — Java

このサンプルでは、直近の 60 分で更新された取引先を取得し、コンソールにその ID を書き込みます。

```
public void getUpdatedRecords() {
   try {
      GregorianCalendar endTime = (GregorianCalendar) connection
            .getServerTimestamp().getTimestamp();
      GregorianCalendar startTime = (GregorianCalendar) endTime.clone();
      // Subtract 60 minutes from the server time so that we have
      // a valid time frame.
      startTime.add(GregorianCalendar.MINUTE, -60);
      System.out.println("Checking updates as of: "
            + startTime.getTime().toString());

      // Get the updated accounts within the specified time frame
      GetUpdatedResult ur = connection.getUpdated("Account", startTime,
            endTime);
      System.out.println("GetUpdateResult: " + ur.getIds().length);

      // Write the results
      if (ur.getIds() != null && ur.getIds().length > 0) {
         for (int i = 0; i < ur.getIds().length; i++) {
            System.out.println(ur.getIds()[i] + " was updated between "
                  + startTime.getTime().toString() + " and "
                  + endTime.getTime().toString());
```

```
        }
      } else {
        System.out.println("No updates to accounts in "
              + "the last 60 minutes.");
      }
    } catch (ConnectionException ce) {
      ce.printStackTrace();
    }
}
```

サンプルコード — C#

このサンプルでは、直近の 60 分で更新された取引先を取得し、コンソールにその ID を書き込みます。

```
public void getUpdatedRecords()
{
   try
   {
      DateTime endTime = binding.getServerTimestamp().timestamp;
      // Subtract 60 minutes from the server time so that we have
      // a valid time frame.
      DateTime startTime = endTime.AddMinutes(-60);
      Console.WriteLine("Checking updates as of: "
            + startTime.ToLocalTime().ToString());

      // Get the updated accounts within the specified time frame
      GetUpdatedResult ur = binding.getUpdated("Account", startTime,
            endTime);
      Console.WriteLine("GetUpdateResult: " + ur.ids.Length);

      // Write the results
      if (ur.ids != null && ur.ids.Length > 0)
      {
         for (int i = 0; i < ur.ids.Length; i++)
         {
            Console.WriteLine(ur.ids[i] + " was updated between "
                  + startTime.ToLocalTime().ToString() + " and "
                  + endTime.ToLocalTime().ToString());
         }
      }
      else
      {
         Console.WriteLine("No updates to accounts in "
               + "the last 60 minutes.");
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

引数

名前	型	説明
sObjectTypeEntityType	string	オブジェクト種別。指定された値は、組織で有効なオブジェクトである必要があります。標準オブジェクトの一覧は、「標準オブジェクト」を参照してください。
startDate	dateTime	データを取得する期間の開始日時 (ローカル時間ではなく協定世界時 (UTC))。API は、指定された dateTime 値に含まれる秒の部分を切り捨てます (たとえば、12:30:15 は 12:30:00 UTC となります)。
endDate	dateTime	データを取得する期間の終了日時 (ローカル時間ではなく協定世界時 (UTC))。API は、指定された dateTime 値に含まれる秒の部分を切り捨てます (たとえば、12:35:15 は 12:35:00 UTC となります)。

重要: 結果の GetUpdatedResult[] の結果には、ID は 600,000 件までという制限があります。getUpdated() コールが 600,000 件以上の ID を返した場合、EXCEEDED_ID_LIMIT 例外が返されます。開始日と終了日の期間を短くすることでこのエラーを回避できます。

応答

GetUpdatedResult[]

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

- データ複製
- API コールの基礎

GetUpdatedResult

getUpdated() コールは、特定の期間内に挿入または更新された各レコードに関する情報が含まれる GetUpdatedResult オブジェクトを返します。GetUpdatedResult 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	更新された各オブジェクトの ID の配列。
latestDateCovered	dateTime	getUpdated() コールの対象となる最終日のタイムスタンプ (ローカル時間ではなく協定世界時 (UTC) タイムゾーン)。値がある場合は、endDate

名前	型	説明
		以前のタイムスタンプになります。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。 メモ: Salesforce が、インスタンスで長時間のトランザクションを実行する場合、その長時間トランザクションが完了するまで、この項目の値が開始時刻になります。これは、長時間トランザクションがユーザーデータに影響を与える可能性があるためです(一括処理など)。

invalidateSessions()

sessionId で指定された 1 つ以上のセッションを終了します。

「sessionId」を参照してください。

構文

```
InvalidateSessionsResult = connection.invalidateSessions(string[] sessionIds);
```

使用方法

1 つ以上のセッションの終了にこのコールを使用します。

ログインユーザーの 1 つのセッションのみを終了するには、logout() を使用することもできます。

サンプルコード — Java

このサンプルでは、一連のセッションを無効化します。このサンプルのメソッドでは、渡されたセッション ID の配列を string 値として取ります。次に、このメソッドではこの配列で invalidateSessions() をコールし、エラーがないかその結果を確認します。

```
public void invalidateSessionsSample(String[] sessionIds) {
   try {
      InvalidateSessionsResult[] results;
      results = connection.invalidateSessions(sessionIds);
      for (InvalidateSessionsResult result : results) {
         // Check results for errors
         if (!result.isSuccess()) {
            if (result.getErrors().length > 0) {
               System.out.println("Status code: "
                     + result.getErrors()[0].getStatusCode());
               System.out.println("Error message: "
                     + result.getErrors()[0].getMessage());
```

```
            }
         } else {
            System.out.println("Success.");
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、一連のセッションを無効化します。このサンプルのメソッドでは、渡されたセッションIDの配列をstring値として取ります。次に、このメソッドではこの配列で `invalidateSessions()` をコールし、エラーがないかその結果を確認します。

```
public void invalidateSessionsSample(string[] sessionIds)
{
   try
   {
      InvalidateSessionsResult[] results;
      results = binding.invalidateSessions(sessionIds);
      foreach (InvalidateSessionsResult result in results)
      {
         // Check results for errors
         if (!result.success)
         {
            if (result.errors.Length > 0)
            {
               Console.WriteLine("Status code: " +
                  result.errors[0].statusCode);
               Console.WriteLine("Error message: " +
                  result.errors[0].message);
            }
         }
         else
         {
            Console.WriteLine("Success.");
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

引数

名前	型	説明
sessionIds	string[]	1 つ以上の sessionId 文字列。上限 200。sessionId は SessionHeader から取得できます。

応答

InvalidateSessionsResult[]

障害

UnexpectedErrorFault

InvalidateSessionsResult

invalidateSessions() コールは、LogoutResult オブジェクトの配列を返します。

「invalidateSessions()」を参照してください。

LogoutResult 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
success	boolean	セッションが正常に終了したか (true)、否か (false) を示します。
errors	Error[]	コールでエラーが発生した場合、1 つ以上の Error オブジェクトの配列。各オブジェクトにはエラーコードと説明が含まれます。

login()

ログインサーバーにログインし、クライアントセッションを開始します。

メモ: login() コールは、ログイン数の制限に含まれます。

構文

```
LoginResult = connection.login(string username, string password);
```

使用方法

login() コールを使用して、ログインサーバーにログインし、クライアントセッションを開始します。クライアントアプリケーションは、他の API コールを行う前に、ログインし、sessionId とサーバー URL を取得します。

クライアントアプリケーションは、`login()` コールを呼び出すとき、ログイン情報としてユーザー名とパスワードを渡します。呼び出し時に、APIがログイン情報を認証します。後続のすべてのAPIコールで使用する、`sessionId`、ログインユーザー名に関連付けられているユーザー ID、および Lightning プラットフォーム API を指す URL を返します。

Salesforceは、クライアントアプリケーションがログインしているIPアドレスを確認し、不明なIPアドレスからのログインをブロックします。API でブロックされたログインに関しては、Salesforce がログイン失敗エラーを返します。ログインするには、ユーザーがセキュリティトークンをユーザーパスワードの末尾に追加する必要があります。たとえば、パスワードが *mypassword* で、セキュリティトークンが *XXXXXXXXXX* の場合は、「*mypasswordXXXXXXXXXX*」と入力する必要があります。セキュリティトークンを取得するには、Salesforce ユーザーインターフェースからパスワードを変更するか、セキュリティトークンをリセットします。ユーザーがパスワードを変更するか、セキュリティトークンをリセットすると、ユーザーのSalesforceレコードに指定されたメールアドレス宛に新しいセキュリティトークンが送信されます。セキュリティトークンは、ユーザーがセキュリティトークンをリセットするか、パスワードを変更するか、またはユーザーのパスワードがリセットされるまで有効です。トークンが無効な場合、ユーザーはログインプロセスを再度行う必要があります。再度ログインを行わないようにするには、クライアントの IP アドレスを組織の信頼できる IP アドレスのリストに追加します。詳細は、「セキュリティトークン」を参照してください。

ログイン後、クライアントアプリケーションで次のタスクが実行されていることを確認します。

- APIがこのセッションに対する後続要求を検証できるように、SOAPヘッダー内にセッションIDを設定する。
- 後続サービス要求の要求先としてサーバー URL を指定する。ログインサーバーでは、ログインコールしかサポートされません。

開発ツールごとに、セッションヘッダーとサーバーURLの指定方法は異なります。詳細は、使用している開発ツールのマニュアルを参照してください。

`login()` は、1 時間につき 1 ユーザーあたり最大 3,600 コールに制限されています。この制限を超えると、「Login Rate Exceeded」（ログイン数の制限を超えました。）エラーが表示されます。1 時間の上限を超えると、Salesforce によりユーザーのログインがブロックされます。ユーザーはブロックされてから 1 時間後、もう一度ログインを試行できます。

Enterprise と Partner エンドポイント

バージョン 11.1 以前の API では、Partner WSDL で構築されたクライアントアプリケーションは Enterprise エンドポイントに要求を送信可能で、Enterprise WSDL で構築されたアプリケーションは Partner エンドポイントに要求を送信可能です。バージョン 12.0 以降では、この機能はサポートされていません。

エンドポイントのベース URL

Salesforce 組織のエンドポイントを指定するときは、ベース URL に関して 3 つのオプションがあります。

1. 推奨: [私のドメイン] のログイン URL: 本番組織の場合は `https://MyDomainName.my.salesforce.com` 形式、Sandbox の場合は `https://MyDomainName--SandboxName.sandbox.my.salesforce.com` 形式。
2. デフォルトの Salesforce ログイン URL: 本番組織および Developer Edition 組織の場合は `https://login.salesforce.com`、Sandbox の場合は `https://test.salesforce.com`。

メモ: システム管理者は、[私のドメイン] の設定を介してデフォルトの Salesforce ログイン URL から SOAP API ログインが行われることを防止できます。

すべての例では、本番組織で推奨される [私のドメイン] のログイン URL 形式が使用されています。Sandbox のエンドポイントを指定したり、デフォルトの Salesforce ログイン URL を使用したりする場合は、必要に応じて例を変更してください。

[私のドメイン] のログイン URL を使用する利点とデフォルトの Salesforce ログイン URL を使用する利点を理解するには、Salesforce ヘルプの「コードを使用した Salesforce へのログイン」を参照してください。

プロキシを使用したログイン

プロキシ経由で Salesforce にログインする場合は、ログインに使用する `ConnectorConfig` クラスのインスタンスでプロキシホストおよびポート番号を設定します。プロキシを認証する必要がある場合は、ユーザー名およびパスワードを設定します。

```
ConnectorConfig config = new ConnectorConfig();
config.setUsername(userId);
config.setPassword(passwd);
config.setAuthEndpoint(authEndPoint);
config.setProxy(proxyHost, proxyPort);
// Set the username and password if your proxy must be authenticated
config.setProxyUsername(proxyUsername);
config.setProxyPassword(proxyPassword);
try {
   EnterpriseConnection connection = new EnterpriseConnection(config);
   // etc.
} catch (ConnectionException ce) {
  ce.printStackTrace();
}
```

セッション終了

クライアントアプリケーションは、セッションを終えるために明示的にログアウトする必要はありません。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。デフォルト値は 2 時間です。API コールを行うと、非活動状態タイマーがゼロにリセットされます。セッションの有効期限 (タイムアウト) の値を変更するには、[設定] から、[クイック検索] ボックスに「セッションの設定」と入力し、[セッションの設定] を選択します。

有効なセルフサービスユーザーの認証

メモ: Spring '12 以降、新しい Salesforce 組織ではセルフサービスポータルを利用できません。既存の組織は、引き続きセルフサービスポータルを使用できます。

アクティブセルフサービスユーザーを認証するには、セルフサービスがユーザーの認証に対して LoginScopeHeader を使用して、OrganizationID を指定します。認証の前提として、セルフサービスユーザーが存在しており、かつ有効である必要があります (「SelfServiceUser」参照)。

顧客の Experience Cloud サイトのユーザー認証

[API の有効化] 権限を持つ有効な Experience Cloud サイトユーザーを認証するには、LoginScopeHeader を使用して、Experience Cloud サイトを含む組織の Organization ID を指定します。サイトユーザーは、存在する有効なユーザーであり、認証される前に Experience Cloud サイトに属している必要があります。

Experience Cloud サイトのユーザーを認証するエンドポイントを指定する場合、ベース URL の形式は、本番組織では `https://MyDomainName.my.site.com`、Sandbox 組織では `https://MyDomainName--SandboxName.sandbox.my.site.com` となります。

Experience Cloud サイトのユーザー認証のすべての例では、拡張ドメインが有効な本番組織用のベース URL の形式が使用されています。Sandbox 組織のエンドポイントを指定する場合、または拡張ドメインをまだ有効にしていない場合は、ベース URL を更新してください。

エンドポイントの例

クライアントアプリケーションでは、ログイン要求を次のエンドポイントに送信できます (この際に、認証エンドポイントの有効な値を使用します)。

Enterprise WSDL:

- `String authEndPoint = "https://MyDomainName.my.salesforce.com/services/Soap/c/version/"`
- `String authEndPoint = "https://MyDomainName.my.site.com/path-prefix/services/Soap/c/version/"`

Partner WSDL:

- `String authEndPoint = "https://MyDomainName.my.salesforce.com/services/Soap/u/version/"`
- `String authEndPoint = "https://MyDomainName.my.site.com/path-prefix/services/Soap/u/version/"`

ログアウト

Salesforce では、不要になったセッションを終了するために、必ず `logout()` をコールすることをお勧めします。このコールは、子セッションも終了させます。ほとんどの保護が行われるようにするには、セッションが期限切れになるのを待つのではなく、ユーザーをログアウトします。

サンプルコード — Java

このサンプルでは、指定されたユーザー名、パスワード、および認証エンドポイント URL を使用してユーザーをログインします。このサンプルでは、ログインの成功後、ユーザー情報とセッション情報をコンソールに書き込みます。このサンプルを実行する前に、ユーザー名、パスワード、認証エンドポイントの値を有効な値に置き換えてください。

API コールを作成するために必要な Web サービス WSDL を生成してインポートする方法については、「クイックスタート」の「ステップ 2: Web サービス WSDL を生成または取得する」を参照してください。

```
public boolean loginSample() {
   boolean success = false;
   String username = "username";
   String password = "password";
   String authEndPoint = "https://MyDomainName.my.salesforce.com/services/Soap/c/24.0/";

   try {
      ConnectorConfig config = new ConnectorConfig();
      config.setUsername(username);
      config.setPassword(password);

      System.out.println("AuthEndPoint: " + authEndPoint);
      config.setAuthEndpoint(authEndPoint);

      connection = new EnterpriseConnection(config);

      // Print user and session info
      GetUserInfoResult userInfo = connection.getUserInfo();
      System.out.println("UserID: " + userInfo.getUserId());
      System.out.println("User Full Name: " + userInfo.getUserFullName());
      System.out.println("User Email: " + userInfo.getUserEmail());
      System.out.println();
      System.out.println("SessionID: " + config.getSessionId());
      System.out.println("Auth End Point: " + config.getAuthEndpoint());
      System.out
            .println("Service End Point: " + config.getServiceEndpoint());
      System.out.println();

      success = true;
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }

   return success;
}
```

サンプルコード — C#

このサンプルでは、指定されたユーザー名、パスワードを使用してユーザーをログインします。login コールの結果には、サービスエンドポイント URL が含まれます。この URL は、組織にサービスしている仮想サーバーインスタンスであり、一意のセッション ID です。このサンプルでは、バインドにこれらの返された値を設定します。バインド URL に返されたサービスエンドポイントを設定します。また、すべての API コールで使用されているセッションヘッダーのセッション ID を設定します。次に、ログインの成功後、ユーザー情報とセッション情報をコンソールに書き込みます。このサンプルを実行する前に、ユーザー名とパスワードの値を有効な値に置き換えてください。

API コールを作成するために必要な Web サービス WSDL を生成してインポートする方法については、「クイックスタート」の「ステップ 2: Web サービス WSDL を生成または取得する」を参照してください。

```
public bool loginSample()
{
   Boolean success = false;
   string username = "username";
   string password = "password";

   // Create a service object
   binding = new SforceService();

   LoginResult lr;
   try
   {
      Console.WriteLine("\nLogging in...\n");
      lr = binding.login(username, password);

      /**
         * The login results contain the endpoint of the virtual server instance
         * that is servicing your org. Set the URL of the binding
         * to this endpoint.
         */
      // Save old authentication end point URL
      String authEndPoint = binding.Url;
      // Set returned service endpoint URL
      binding.Url = lr.serverUrl;

      /** Get the session ID from the login result and set it for the
         * session header that will be used for all subsequent calls.
         */
      binding.SessionHeaderValue = new SessionHeader();
      binding.SessionHeaderValue.sessionId = lr.sessionId;

      // Print user and session info
      GetUserInfoResult userInfo = lr.userInfo;
      Console.WriteLine("UserID: " + userInfo.userId);
      Console.WriteLine("User Full Name: " +
            userInfo.userFullName);
      Console.WriteLine("User Email: " +
            userInfo.userEmail);
      Console.WriteLine();
      Console.WriteLine("SessionID: " +
            lr.sessionId);
      Console.WriteLine("Auth End Point: " +
            authEndPoint);
      Console.WriteLine("Service End Point: " +
            lr.serverUrl);
      Console.WriteLine();

      // Return true to indicate that we are logged in, pointed
      // at the right URL and have our security token in place.
      success = true;
   }
```

```
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
                                    e.Message + "\n" + e.StackTrace);
    }
    return success;
}
```

引数

名前	型	説明
username	string	ログインで使用する username。
password	string	指定された username に関連付けられているログインパスワード。

ログイン要求のサイズは、10 KB 未満に制限されています。

応答

LoginResult

障害

LoginFault

UnexpectedErrorFault

関連トピック:

- API コールの基礎
- API コールの基礎
- クイックスタート: SOAP API

LoginResult

login() コールが返す LoginResult 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
metadataServerUrl	string	後続のメタデータ API コールを処理するエンドポイントの URL。クライアントアプリケーションのエンドポイントを設定する必要があります。
passwordExpired	boolean	ログイン試行時に使用されたパスワードが失効したか (true)、否か (false) を示します。パスワードが失効した場合、API

名前	型	説明
		は、有効な `sessionId` を返しますが、許可される操作は、`setPassword()` コールのみです。
`serverUrl`	string	後続の API コールを処理するエンドポイントの URL。クライアントアプリケーションのエンドポイントを設定する必要があります。
`sessionId`	string	このセッションに関連付けられた一意の ID。クライアントアプリケーションは、この値をセッションヘッダーに設定する必要があります。
`userId`	ID	特定のユーザー名とパスワードに関連付けられたユーザーの ID。
`userInfo`	getUserInfoResult	ユーザー情報項目。この項目の一覧は、「getUserInfoResult」を参照してください。

`logout()`

ログインユーザーのセッションを終了します。

構文

```
connection.logout();
```

使用方法

このコールは、コールを発行したログインユーザーのセッションを終了させます。引数は不要です。

ログインユーザー以外のユーザーによって開始された 1 つ以上のセッションを終了させる方法については、「`invalidateSessions()`」を参照してください。

サンプルコード — Java

このサンプルでは `logout()` をコールし、現在のユーザーをログアウトして、メッセージをコンソールに書き込みます。

```
public void logoutSample() {
   try {
      connection.logout();
      System.out.println("Logged out.");
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは `logout()` をコールし、現在のユーザーをログアウトして、メッセージをコンソールに書き込みます。

```
public void logoutSample()
{
   try
   {
      binding.logout();
      Console.WriteLine("Logged out.");
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

引数

このコールは、引数を使用しません。コールを発行したログインユーザーのセッションを終了させるため、引数は不要です。ログインユーザーは、このコール用に SessionHeader 内で指定された `sessionId` によって判別されます。

応答

voidが返されます。コールの失敗エラーが、セッションがすでにログアウトされていることを意味しているため、結果は必要ありません。システム利用不能などの予期せぬエラーは、クライアントアプリケーションによって処理される必要があるエラーを発生させます。

障害

`UnexpectedErrorFault`

merge()

同じ種別の最大 3 つのレコードを 1 つのレコードに統合します。入力は `MergeRequest` 要素の配列で、各要素は統合するレコードを指定します。出力は、マージ結果に関する情報を含むMergeResultオブジェクトです。

重要: 可能な場合は、Equalityの会社の値に一致するように、含めない用語を変更しました。コード内の用語を変更すると、現在の実装が壊れる可能性があるため、このオブジェクトの名前は維持しました。

構文

```
MergeResult[]= connection.merge(MergeRequest[] mergeRequests);
```

使用方法

`merge()` を使用して、同じオブジェクト種別のレコードを、*主レコード*と呼ばれる 1 つのレコードに統合します。`merge()` は、*ビクティムレコード*と呼ばれる他のレコードを削除します。ビクティムレコードに関連レコードがある場合、`merge()` は主レコードを関連レコードの新しい親にします。

ルールとガイドライン

非主レコードの値
: `merge()` をコールする前に、非主レコードの項目値を主レコード値よりも優先するかどうかを決定します。優先する場合は、`MergeRequest` の `masterRecord` によって識別されるレコードの項目名と値を設定します。

取引先責任者、リード、データプライバシーレコード
: 個人オブジェクトに基づいた対応するデータプライバシーレコードがある取引先責任者またはリードをマージする場合、`merge()` は主レコードに関連付ける適切なデータプライバシーレコードを決定します。

- リードと取引先責任者のマージに対して最後に更新されたデータプライバシーレコードを保持するオプションを選択した場合、`merge()` は最後に更新されたデータプライバシーレコードを選択します。
- それ以外の場合、`merge()` はすでに主レコードに関連付けられているデータプライバシーレコードを選択します。

連続するマージ
: `merge()` は入力引数の各 `MergeResult` 要素を別個のトランザクションとして処理するため、複数のレコードを連続して同じ主レコードにマージできます。

  連続するマージを実行するには、`MergeResult` 要素の配列を指定して `merge()` をコールします。`MergeResult` 要素ごとに、以下を設定します。

- `masterRecord` を主レコード ID に設定します。
- `recordToMergeIds` 配列の各要素を、主レコードに統合するレコードの ID に設定します。

削除されたレコード
: `queryAll()` を使用して、マージ中に削除されたレコードを表示します。

マージされたレコードの表示
: 指定の時間内にマージされたすべてのレコードを見つけるには、SELECT ステートメントと一緒に `queryAll()` をコールします。次に例を示します。

```
SELECT Id FROM Contact WHERE isDeleted=true and masterRecordId != null
   AND SystemModstamp > 2006-01-01T23:01:01+01:00
```

サポートされているオブジェクト種別
: サポートされているオブジェクト種別は、Lead、Contact、Account、Person Account、および Individual です。同じ種別のオブジェクトのみをマージできます。たとえば、リードはリードとのみマージできます。

関連取引先
: 関連取引先の一部である取引先をマージする場合、`merge()` はビクティムの子レコードを主レコードの子として設定しようとします。このアクションによって循環的なリレーションが生まれる場合、`merge()` はエラーを返します。

取引先責任者の上司リレーション
`ReportsToId` 項目に値がある取引先責任者をマージする場合、`merge()` はビクティムの `ReportsToId` 値を主レコードに追加しようとします。このアクションによって循環的なリレーションが生まれる場合、`merge()` はエラーを報告します。

取引先責任者とポータルユーザー
ポータルユーザーが関連付けられている取引先責任者ビクティムレコードをマージする場合、ビクティムレコードの `MergeRequest` 要素の `AdditionalInformationMap` を設定します。1 つのビクティムとポータルユーザーのみ主レコードにマージできます。Salesforce Classic では、カスタマーポータルの使用が有効になっている個人取引先をマージできません。

考慮事項
次の制限が、マージ要求に適用されます。

- 1 つの SOAP コール内に、最大 200 件のマージ要求を作成可能です。
- 主レコードを含めて、最大 3 つのレコードが 1 つの要求にマージ可能です。この制限は、Salesforce ユーザーインターフェースによって適用される制限と同じです。3 件を超えるレコードをマージするには、連続するマージを実行します。
- 外部 ID 項目では、`merge()` を使用することはできません。
- リードと取引先責任者のマージに対して最後に更新されたデータプライバシーレコードを保持するオプションを選択し、選択されたデータプライバシーレコードの CRUD 権限がコール元にない場合、メインレコードにすでに関連付けられているデータプライバシーレコードがマージプロセスにより選択されます。

重複するリレーション
同じ取引先責任者に関連付けられている取引先または個人取引先はマージできません。取引先をマージする前に、取引先と取引先責任者の重複リレーションを削除します。

メモ: API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了していました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

サンプルコード — Java

このサンプルでは、1 つのビクティム取引先を主取引先とマージします。2 つの取引先が作成され、ビクティムにメモが添付されます。マージ後、コードはビクティム取引先の ID と更新された子レコード数を出力します。この例では、マージされた取引先のメモは主レコードに移動するため、更新されたレコード数は 1 です。

```
public Boolean mergeRecords() {
   Boolean success = false;
   // Array to hold the results
   String[] accountIds = new String[2];
   try {
      // Create two accounts to merge
      Account[] accounts = new Account[2];
      Account masterAccount = new Account();
      masterAccount.setName("MasterAccount");
      masterAccount.setDescription("The Account record to merge with.");
```

```
accounts[0] = masterAccount;
Account accountToMerge = new Account();
accountToMerge.setName("AccountToMerge");
accountToMerge
      .setDescription("The Account record that will be merged.");
accounts[1] = accountToMerge;
SaveResult[] saveResults = connection.create(accounts);

if (saveResults.length > 0) {
   for (int i = 0; i < saveResults.length; i++) {
      if (saveResults[i].isSuccess()) {
         accountIds[i] = saveResults[i].getId();
         System.out.println("Created Account ID: "
               + accountIds[i]);
      } else {
         // If any account is not created,
         // print the error returned and exit
         System.out
               .println("An error occurred while creating account."
                     + " Error message: "
                     + saveResults[i].getErrors()[0].getMessage());
         return success;
      }
   }
}

// Set the Ids of the accounts
masterAccount.setId(accountIds[0]);
accountToMerge.setId(accountIds[1]);

// Attach a note to the account to be merged with the master,
// which will get re-parented after the merge
Note note = new Note();
System.out.println("Attaching note to record " +
      accountIds[1]);
note.setParentId(accountIds[1]);
note.setTitle("Merged Notes");
note.setBody("This note will be moved to the "
      + "MasterAccount during merge");
SaveResult[] sRes = connection.create(new SObject[] { note });
if (sRes[0].isSuccess()) {
   System.out.println("Created Note record.");
} else {
   Error[] errors = sRes[0].getErrors();
   System.out.println("Could not create Note record: "
         + errors[0].getMessage());
}

// Perform the merge
MergeRequest mReq = new MergeRequest();
masterAccount.setDescription("Was merged");
mReq.setMasterRecord(masterAccount);
mReq.setRecordToMergeIds(new String[] { saveResults[1].getId() });
MergeResult mRes = connection.merge(new MergeRequest[] { mReq })[0];
```

```
      if (mRes.isSuccess())
      {
         System.out.println("Merge successful.");
         // Write the IDs of merged records
         for(String mergedId : mRes.getMergedRecordIds()) {
            System.out.println("Merged Record ID: " + mergedId);

         }
         // Write the updated child records. (In this case the note.)
         System.out.println(
               "Child records updated: " + mRes.getUpdatedRelatedIds().length);
         success = true;
      } else {
         System.out.println("Failed to merge records. Error message: " +
               mRes.getErrors()[0].getMessage());
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
   return success;
}
```

サンプルコード — C#

このサンプルでは、1 つのビクティム取引先を主取引先とマージします。2 つの取引先が作成され、ビクティムにメモが添付されます。マージ後、コードはビクティム取引先の ID と更新された子レコード数を出力します。この例では、マージされた取引先のメモは主レコードに移動するため、更新されたレコード数は 1 です。

```
public Boolean mergeRecords()
{
   Boolean success = false;
   // Array to hold the results
   String[] accountIds = new String[2];
   try
   {
      // Create two accounts to merge
      Account[] accounts = new Account[2];
      Account masterAccount = new Account();
      masterAccount.Name = "MasterAccount";
      masterAccount.Description = "The Account record to merge with.";
      accounts[0] = masterAccount;
      Account accountToMerge = new Account();
      accountToMerge.Name = "AccountToMerge";
      accountToMerge
            .Description = "The Account record that will be merged.";
      accounts[1] = accountToMerge;
      SaveResult[] saveResults = binding.create(accounts);

      if (saveResults.Length > 0)
      {
         for (int i = 0; i < saveResults.Length; i++)
         {
```

```
            if (saveResults[i].success)
            {
                accountIds[i] = saveResults[i].id;
                Console.WriteLine("Created Account ID: "
                      + accountIds[i]);
            }
            else
            {
                // If any account is not created,
                // print the error returned and exit
                Console.WriteLine("An error occurred while creating account."
                            + " Error message: "
                            + saveResults[i].errors[0].message);
                return success;
            }
        }
    }

    // Set the Ids of the accounts
    masterAccount.Id = accountIds[0];
    accountToMerge.Id = accountIds[1];

    // Attach a note to the account to be merged with the master,
    // which will get re-parented after the merge
    Note note = new Note();
    Console.WriteLine("Attaching note to record " +
          accountIds[1]);
    note.ParentId = accountIds[1];
    note.Title = "Merged Notes";
    note.Body = "This note will be moved to the "
          + "MasterAccount during merge";
    SaveResult[] sRes = binding.create(new sObject[] { note });
    if (sRes[0].success)
    {
        Console.WriteLine("Created Note record.");
    }
    else
    {
        Error[] errors = sRes[0].errors;
        Console.WriteLine("Could not create Note record: "
              + errors[0].message);
    }

    // Perform the merge
    MergeRequest mReq = new MergeRequest();
    masterAccount.Description = "Was merged";
    mReq.masterRecord = masterAccount;
    mReq.recordToMergeIds = new String[] { saveResults[1].id };

    MergeResult mRes = binding.merge(new MergeRequest[] { mReq })[0];

    if (mRes.success)
    {
        Console.WriteLine("Merge successful.");
```

```
            // Write the IDs of merged records
            foreach (String mergedId in mRes.mergedRecordIds)
            {
               Console.WriteLine("Merged Record ID: " + mergedId);
            }
            // Write the updated child records. (In this case the note.)
            Console.WriteLine(
                  "Child records updated: " + mRes.updatedRelatedIds.Length);
            success = true;
         }
         else
         {
            Console.WriteLine("Failed to merge records. Error message: " +
                  mRes.errors[0].message);
         }
      }
      catch (SoapException e)
      {
         Console.WriteLine("An unexpected error has occurred: " +
                                    e.Message + "\n" + e.StackTrace);
      }
      return success;
}
```

引数

このコールは MergeRequest オブジェクトの配列を受け取ります。MergeRequest オブジェクトには、次のプロパティが含まれます。

名前	型	説明
masterRecord	sObject	必須。その他のレコードのマージ先であるオブジェクトの ID を提供する必要があります。状況に応じて、更新される項目とその値を提供してください。
recordToMergeIds	ID[]	必須。最小で 1 つ、最大で 2 つ。主レコードにマージされるその他のレコード。
AdditionalInformationMap	map	項目と値の対応付け。 • ポータルユーザー ID のマージ: – 名前: PortalUserId – 値: ポータルユーザーの ID • 他のすべてのマージケースでは、null に設定します。

応答

MergeResult[]

障害

InvalidIdFault

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

API コールの基礎

MergeResult

merge() コールは、MergeResult オブジェクトの配列を返します。

MergeResult 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
errors	Error[]	merge() コール中に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 Error オブジェクトからなる配列。
id	ID	プライマリレコード (他のレコードがマージされたレコード) の ID。
mergedRecordIds	ID[]	プライマリレコードにマージされたレコードの ID。成功した場合、その値は mergeRequest.recordToMergeIds に一致します。
success	boolean	マージが正常に行われたか (true)、否か (false) を示します。
updatedRelatedIds	ID[]	マージの結果として移動し (親が再設定され)、merge コールを送信するユーザーから表示可能なすべての関連レコードの ID。

performQuickActions()

種別が create または update のクイックアクションを実行します。

構文

```
PerformQuickActionResult[] = connection.performQuickActions(PerformQuickActionRequest
PerformQuickActionRequest[]);
```

使用方法

performQuickActions() コールは、特定のクイックアクションを実行するために使用します。
PerformQuickActionResult オブジェクトの配列を返します。

メモ: API バージョン 46.0 以降では、`performQuickActions()` リクエストボディで、グローバルクイックアクションの `apiName` にプレフィックス `Global.` を追加できます。また、このリクエストボディでは、プレフィックスのないグローバルクイックアクションの API 名も使用できます。

メモ: カスタムコミュニティ URL を使用して API にアクセスし、`performQuickActions()` コールを使用してグループを作成する場合、グループはそのコミュニティ内でのみ使用できます。

メモ: OutgoingEmail エンティティは、performQuickAction API からのコールでのみ作成できます。

サンプル — Java

この例では、クイックアクションを使用して新規の取引先責任者を作成します。

```
public void example() throws Exception {

    PerformQuickActionRequest req = new PerformQuickActionRequest();

    Contact con = new Contact();
    con.setLastName("Smith");

    req.setQuickActionName("Account.QuickCreateContact");
    req.setParentId("001D000000JSaHa");
/* For version 29.0 and greater, use setContextId */
    req.setRecords(new SObject[] { con });  //you can only save one record here
    PerformQuickActionResult[] pResult =
        conn.performQuickActions(new PerformQuickActionRequest[] { req } );
    for(PerformQuickActionResult pr : pResult) {
        assert pr.getSuccess();
        assert pr.getCreated();
        assert pr.getErrors().length == 0;
        System.out.println("Id of the record created: " + pr.getIds()[0]);
        System.out.println("Id of the feeditem for action performed: " +
            pr.getFeedItemIds()[0]);
    }
}
```

引数

名前	型	説明
`quickActions`	PerformQuickActionRequest	実行するアクション要求。

PerformQuickActionRequest

名前	型	説明
`parentOrContextId`	ID	• API バージョン 28.0 では、`parentId` は要求のレコードが作成される sObject の ID です。

名前	型	説明
		• API バージョン 29.0 以降では、`contextId` は要求のレコードが作成されるコンテキストの ID です。
`quickActionName`	string	親 sObject またはコンテキスト sObject およびアクション名 (Opportunity.QuickCreateOpp など)。
`records`	SObject[]	作成するレコード。一度に保存できるレコードは 1 つのみです。

応答

PerformQuickActionResult

PerformQuickActionResult

`performQuickActions()` コールは、PerformQuickActionResult オブジェクトの配列を返します。

名前	型	説明
`created`	boolean	レコードが正常に作成された場合は `true`、レコードが作成されなかった場合は `false`。
`errors`	Error[]	コールで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 Error オブジェクトの配列。
`feedItemIds`	ID[]	ID を含む文字列の形式で、フィード項目の一意の識別子の配列を返します。パートナーポータルでは ID を含む種別を返します。
`ids`	ID[]	ID の配列。
`success`	boolean	`true` であれば、アクションは正常に実行されました。`False` であれば、アクションは失敗しました。
`successMessage`	string	アクションが正常に完了するとユーザーに表示されるメッセージを返します。

`process()`

承認のために承認プロセスインスタンスの配列を送信します。または、承認、却下、削除される承認プロセスインスタンスの配列を処理します。詳細は、Salesforceヘルプの「承認プロセスの設定」を参照してください。

構文

```
ProcessResult = connection.process( processType processRequest[])
```

processType は、ProcessSubmitRequest または ProcessWorkitemRequest のいずれかになる可能性があります。

使用方法

process() を使用して、次の 2 つのタスクのいずれかを実行します。

- オブジェクトの配列を承認プロセスに送信します。送信時、オブジェクトがすでに承認プロセスにあってはいけません。ProcessSubmitRequest 署名を使用します。
- 承認アクション (承認または却下) を実行することによって、承認プロセスに送信されているオブジェクトを処理します。ProcessWorkitemRequest 署名を使用します。

要求が処理されると、送信した要求と同じプロセスインスタンスを含む ProcessResult が返されます。

個別のレコードの失敗は、要求全体の失敗にはなりません。

メモ: このコールで Apex トリガーを起動できるため、文字列を含む項目が更新される場合があります。

API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了していました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

サンプルコード — Java

このサンプルでは、承認を処理する sObject の ID と、次の承認者の ID が格納された配列を受け取ります。プロセス承認申請を作成し、承認を得るために送信します。最後に、process() コールの結果を解析します。

```
public void processRecords(String id, String[] approverIds) {
   ProcessSubmitRequest request = new ProcessSubmitRequest();
   request.setComments("A comment about this approval.");
   request.setObjectId(id);
   request.setNextApproverIds(approverIds);
   try {
      ProcessResult[] processResults = connection
            .process(new ProcessSubmitRequest[] { request });
      for (ProcessResult processResult : processResults) {
         if (processResult.isSuccess()) {
            System.out.println("Approval submitted for: " + id + ":");
            for (int i = 0; i < approverIds.length; i++) {
               System.out
                     .println("\tBy: " + approverIds[i] + " successful.");
            }
            System.out.println("Process Instance Status: "
                  + processResult.getInstanceStatus());
         } else {
            System.out.println("Approval submitted for: " + id
                  + ", approverIds: " + approverIds.toString() + " FAILED.");
            System.out.println("Error: "
```

```
                    + processResult.getErrors()[0].toString());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

サンプルコード — C#

このサンプルでは、承認を処理する sObject の ID と、次の承認者の ID が格納された配列を受け取ります。プロセス承認申請を作成し、承認を得るために送信します。最後に、`process()` コールの結果を解析します。

```
public void processRecords(String id, String[] approverIds)
{
    ProcessSubmitRequest request = new ProcessSubmitRequest();
    request.comments = "A comment about this approval.";
    request.objectId = id;
    request.nextApproverIds = approverIds;
    try
    {
        ProcessResult[] processResults = binding.process(
                        new ProcessSubmitRequest[] { request });
        foreach (ProcessResult processResult in processResults)
        {
            if (processResult.success)
            {
                Console.WriteLine("Approval submitted for: " + id + ":");
                for (int i = 0; i < approverIds.Length; i++)
                {
                    Console.WriteLine("\tBy: " + approverIds[i] + " successful.");
                }
                Console.WriteLine("Process Instance Status: "
                        + processResult.instanceStatus);
            }
            else
            {
                Console.WriteLine("Approval submitted for: " + id
                        + ", approverIds: " + approverIds.ToString() + " FAILED.");
                Console.WriteLine("Error: "
                        + processResult.errors.ToString());
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
                                    e.Message + "\n" + e.StackTrace);
    }
}
```

ProcessSubmitRequest 引数

名前	型	説明
comments	string	申請に付記するテキスト。差し込み項目または数式を参照しないでください。 申請コメントは、指定したレコードの承認履歴に表示されます。テンプレートで {!ApprovalRequest.Comments} 差し込み項目を使用する場合は、最初の承認申請メールにもこのテキストが表示されます。
nextApproverIds	ID	プロセスが引き続き承認の詳細を要求する場合、次の要求に割り当てられるユーザー ID。
objectId	ID	承認申請するレコード。
processDefinitionNameOrId	string	レコードを申請する特定の承認プロセスの一意の名前または ID。このプロセスは、オブジェクト種別が objectId で指定したレコードと同じでなければなりません。 skipEntryCriteria が *true* である場合は必須です。
skipEntryCriteria	boolean	*true* の場合、processDefinitionNameOrId で定義されたプロセスに設定された開始条件に照らして評価されません。
submitterId	ID	レコードの承認申請を行ったユーザーの ID。このユーザーは、承認申請に対する返答に関する通知を受信します。 ユーザーは、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。

ProcessWorkitemRequest 引数

名前	型	説明
action	string	承認用に送信された項目の後処理のアクションの種類 (承認、却下、または削除) を示す文字列。Removed を指定できるのは、Salesforce システム管理者だけです。承認プロセスで [申請者に承認申請の取り消しを許可] オプションが選択されている場合、承認の申請者も削除を指定可能です。
comments	string	申請に付記するテキスト。差し込み項目または数式を参照しないでください。 申請コメントは、指定したレコードの承認履歴に表示されます。テンプレートで {!ApprovalRequest.Comments} 差し込み項目を使用する場合は、最初の承認申請メールにもこのテキストが表示されます。

名前	型	説明
nextApproverIds	ID	プロセスが引き続き承認の詳細を要求する場合、次の要求に割り当てられるユーザー ID。
workitemId	ID	処理対象 (承認、却下、または削除) の ProcessInstanceWorkitem の ID。

応答

ProcessResult[]

障害

ALREADY_IN_PROCESS

NO_APPLICABLE_PROCESS

関連トピック:

API コールの基礎

ProcessResult

process() コールは ProcessResult オブジェクトを返します。コールの種別 (承認申請、または承認用にすでに送信済みのプロセスオブジェクト) に基づき、次のプロパティが割り当てられます。

名前	型	説明
actorIds	ID	この承認ステップに現在割り当てられているユーザーの ID。
entityId	ID	処理されているオブジェクト。
errors	Error[]	要求が失敗した場合に返されるエラーのセット。
instanceId	ID	処理用に提出されるオブジェクトに関連付けられている ProcessInstance の ID。
instanceStatus	string	現在のプロセスインスタンスの状態 (個別のオブジェクトではなく、全体のプロセスインスタンス)。有効値は、「Approved」、「Rejected」、「Removed」、または「Pending」です。
newWorkItemIds	ID[]	ProcessInstanceWorkitem 項目を示す、大文字と小文字が区別されない ID (保留中の承認要求セット)。
success	boolean	処理または承認が正常に完了した場合、true。

query()

指定のオブジェクトに対してクエリを実行して、指定の条件に一致したデータを返します。

構文

```
QueryResult = connection.query(string queryString);
```

使用方法

query() コールは、オブジェクトからデータを取得する場合に使用します。クライアントアプリケーションが query() コールを呼び出すとき、照会するオブジェクトを指定するクエリ式、取得する項目、および特定のオブジェクトが該当するかどうかを判断する条件を渡します。クエリで使用される構文や規則についての詳細は、『Salesforce SOQL および SOSL リファレンスガイド』の「SOQL SELECT の構文」を参照してください。

呼び出し時に、API は特定のオブジェクトに対するクエリを実行し、API 上のクエリの結果をキャッシュし、クエリレスポンスオブジェクト QueryResult を返します。それから、クライアントアプリケーションは、クエリレスポンスの行を通して反復し、情報を取得するために、QueryResult 上のメソッドを使用できます。

指定のオブジェクトを照会したり、指定の項目リスト内で項目を照会したりするには、適切なアクセス権でクライアントアプリケーションにログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

オブジェクトは、APIを使用して照会するように設定する必要があります。ただし、オブジェクトによっては、API 経由で照会できないものもあります。オブジェクトがクエリ可能かどうかを判断するには、オブジェクト上の describeSObjects() コールを使用して、その queryable プロパティを確認します。

ヒント: Enterprise WSDL を使用している場合、選択リストへのデータ入力に describe を使用しないでください。たとえば実行後に、Salesforce システム管理者が、そのsObject に項目を追加する場合、describe コールは項目をプルダウンしますが、ツールキットではシリアライズを処理できないため、このインテグレーションは失敗します。

query() コールは、削除されたレコードとアーカイブされたレコードを自動的に除外します。削除されたレコードまたはアーカイブされたレコードを結果に含めるには、queryAll() を使用します。

クエリ結果のオブジェクトでは、最大 2,000 行のデータを返すことができます。この最大値は、デフォルト設定にもなっています。ただし、パフォーマンスを最適化するために、返されるバッチサイズは、照会されたレコードのサイズと複雑さに基づいて、最大値またはリクエストで設定された値よりも少なくすることができます。クエリ結果がデフォルト値を超える場合は、queryMore() コールを使用して、バッチの追加の行を取得します。デフォルトのバッチの結果の数は QueryOptions (ページ 413) ヘッダーで調整します。詳細およびバッチサイズの更新の例については、『SOQL および SOSL リファレンス』の「クエリのバッチサイズの変更」を参照してください。

処理が 2 分以上経過したクエリは、タイムアウトになります。タイムアウトしたクエリに関しては、API は、InvalidQueryLocatorFault の API 失敗エラー要素を返します。タイムアウトが発生した場合は、より少量のデータをスキャンするか、クエリを再構成します。

型が base64 の項目を照会する場合には、クエリレスポンスオブジェクトは、一度に 1 つのレコードしか返しません。query() コールのバッチサイズを変更しても、このプロセスを変更することはできません。

メモ: マルチ通貨組織に関しては、異なる通貨の値を含んでいる通貨項目を照会するには、特別な処理が必要です。たとえば、クライアントアプリケーションで、UnitPrice 項目の値に基づいて PricebookEntry オブジェクトを照会する場合や、UnitPrice の金額が異なる通貨で表示される場合には、クエリのロジックで、この状況を正しく処理する必要があります。次に、10USD以上の単価の全商品の商品コードを取得する場合のクエリ式の例を示します。

```
SELECT Product2Id,ProductCode,UnitPrice FROM PricebookEntry
   WHERE (UnitPrice >= 10 and CurrencyIsoCode='USD')
   OR (UnitPrice >= 5.47 and CurrencyIsoCode='GBP')
   OR (UnitPrice >= 8.19 and CurrencyIsoCode='EUR')
```

引数

名前	型	説明
queryString	string	クエリの対象となるオブジェクト、受け取る項目、およびクエリに特定のオブジェクトを含む任意の条件を指定する SOQL クエリが含まれます。『Salesforce SOQL および SOSL リファレンス』を参照してください。

サンプルコード — Java

このサンプルでは、すべての取引先責任者の名および姓を取得するクエリを実行します。クエリ文字列を渡して、query() をコールし、レコードの最初のバッチを取得します。また、この例では、ループで queryMore() (ページ 218) をコールして、レコードが返されなくなるまで、後続のバッチを取得します。照会した取引先責任者の名と姓をコンソールに書き込みます。

```
public void queryRecords() {
   QueryResult qResult = null;
   try {
      String soqlQuery = "SELECT FirstName, LastName FROM Contact";
      qResult = connection.query(soqlQuery);
      boolean done = false;
      if (qResult.getSize() > 0) {
         System.out.println("Logged-in user can see a total of "
            + qResult.getSize() + " contact records.");
         while (!done) {
            SObject[] records = qResult.getRecords();
            for (int i = 0; i < records.length; ++i) {
               Contact con = (Contact) records[i];
               String fName = con.getFirstName();
               String lName = con.getLastName();
               if (fName == null) {
                  System.out.println("Contact " + (i + 1) + ": " + lName);
               } else {
                  System.out.println("Contact " + (i + 1) + ": " + fName
                        + " " + lName);
               }
            }
            if (qResult.isDone()) {
```

```
                done = true;
            } else {
                qResult = connection.queryMore(qResult.getQueryLocator());
            }
         }
      } else {
         System.out.println("No records found.");
      }
      System.out.println("\nQuery succesfully executed.");
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、すべての取引先責任者の名および姓を取得するクエリを実行します。クエリ文字列を渡して、`query()` をコールし、レコードの最初のバッチを取得します。また、この例では、ループで queryMore() をコールして、レコードが返されなくなるまで、後続のバッチを取得します。照会した取引先責任者の名と姓をコンソールに書き込みます。

```
public void queryRecords()
{
   QueryResult qResult = null;
   try
   {
      String soqlQuery = "SELECT FirstName, LastName FROM Contact";
      qResult = binding.query(soqlQuery);
      Boolean done = false;
      if (qResult.size > 0)
      {
         Console.WriteLine("Logged-in user can see a total of "
            + qResult.size + " contact records.");
         while (!done)
         {
            sObject[] records = qResult.records;
            for (int i = 0; i < records.Length; ++i)
            {
               Contact con = (Contact)records[i];
               String fName = con.FirstName;
               String lName = con.LastName;
               if (fName == null)
               {
                  Console.WriteLine("Contact " + (i + 1) + ": " + lName);
               }
               else
               {
                  Console.WriteLine("Contact " + (i + 1) + ": " + fName
                        + " " + lName);
               }
            }
            if (qResult.done)
            {
```

```
                done = true;
            }
            else
            {
                qResult = binding.queryMore(qResult.queryLocator);
            }
        }
      }
      else
      {
         Console.WriteLine("No records found.");
      }
      Console.WriteLine("\nQuery succesfully executed.");
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

応答

QueryResult

障害

MalformedQueryFault

InvalidSObjectFault

InvalidFieldFault

UnexpectedErrorFault

関連トピック:

- queryAll()
- queryMore()
- API コールの基礎
- クエリのバッチサイズの変更

AggregateResult

クエリコールに `MAX()` などの集計関数が含まれる場合にのみクエリ結果を返す参照のみの SObject。

`query()` **または** `queryMore()` **コールに集計関数が含まれる場合にのみ結果が** AggregateResult **で返されます。このコールに集計関数が含まれない場合、結果は** QueryResult SObject **で返されます。**

たとえば、次のクエリは `records` 項目で Contact レコードの配列を返します。

```
SELECT Id, LastName
FROM Contact
WHERE FirstName = 'Bob'
```

SOQL クエリに集計関数が含まれる場合、レコード項目は AggregateResult レコードの配列を返します。

項目

各 AggregateResult オブジェクトには、`SELECT` リストのアイテムごとに別個の項目が含まれます。Enterprise WSDL の場合、各アイテムの結果を取得するには、WSC クライアントフレームワークを使用するときに AggregateResult オブジェクトに対して `getField()` をコールします。Partner WSDL の場合、各アイテムの結果を取得するには、オブジェクトに対して `getField()` をコールします。

サンプルコード — Java

```
public void queryAggregateResult() {
  try {
    String groupByQuery = "SELECT Account.Name n, " +
        "MAX(Amount) max, MIN(Amount) min " +
        "FROM Opportunity GROUP BY Account.Name";
    QueryResult qr = connection.query(groupByQuery);
    if (qr.getSize() > 0) {
      System.out.println("Query returned " +
          qr.getRecords().length +  " results."
      );
      for (SObject sObj : qr.getRecords()) {
        AggregateResult result = (AggregateResult) sObj;
        System.out.println("aggResult.Account.Name: " +
            result.getField("n")
        );
        System.out.println("aggResult.max: " +
            result.getField("max")
        );
        System.out.println("aggResult.min: " +
            result.getField("min")
        );
        System.out.println();
      }
    } else {
      System.out.println("No results found.");
    }
    System.out.println("\nQuery successfully executed.");
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

サンプルコード — C#

```
private void testAggregateResult()
{
   try
   {
      QueryResult qr = null;

      binding.QueryOptionsValue = new QueryOptions();

      String soqlStr = "SELECT Name, " +
                       "MAX(Amount), " +
                       "MIN(Amount) " +
                       "FROM Opportunity " +
                       "GROUP BY Name";

      qr = binding.query(soqlStr);

      if (qr.size > 0)
      {

         for (int i = 0; i < qr.records.Length; i++)
         {

                sforce.AggregateResult ar = (AggregateResult)qr.records[i];

                foreach (XmlElement e in ar.Any)
                    Console.WriteLine(
                        "{0} - {1}",
                        e.LocalName,
                        e.InnerText
                    );

         }
      }
      else
      {
         Console.WriteLine("No records found");
      }
      Console.WriteLine("Query successfully executed.");
   }
   catch (Exception ex)
   {
      Console.WriteLine(
          "\nFailed to execute query successfully." +
          "error message was: \n" +
          ex.Message
      );

   }
}
```

QueryLocator

QueryResult で返され、queryMore() で使用される文字列。クエリ結果をさらに確認および取得するために使用されます。サーバー側のカーソルを表します。

大きすぎるまたは複雑なクエリの結果を 1 つのバッチで返せない場合、1 つ以上のサーバー側カーソルと対応するクエリロケーターが自動的に作成されます。カーソルは、データベース内の追加のクエリ結果のロケーションを示し、クエリロケーターはカーソルを検出します。

レコードの数が、1 回の query() または queryMore() コールで返すことができる数より多い場合は、新しい queryLocator が QueryResult オブジェクトで返されます。次の結果セットを取得するには、最後に返されたクエリロケーター値を使用して、後続の queryMore() コールをフォローアップします。queryLocator 値はそれぞれ、一度だけ使用します。

queryLocator は、カスタムメタデータ型にアクセスするクエリでは使用できません。

クエリロケーターが無効な場合、API は障害要素の InvalidQueryLocatorFault を返し、さらに起こりうる拡張エラーコードと追加情報を返します。

関連トピック:

- QueryResult
- API コールで使用されるコアデータ型
- SOQL および SOSL の検索クエリの制限

QueryResult

クエリ結果とクエリに関する追加データを含む参照のみの SObject。query() または queryMore() コールに対する応答として返されます。

QueryResult は、MAX() などの集計関数を使用しないクエリにのみ応答を返します。集計関数を使用するクエリの場合、結果は AggregateResult で返されます。

QueryResult 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
queryLocator	QueryLocator	データベースのサーバー側カーソルを指します。クエリ結果の現在の処理場所を追跡し、追加の結果の場所を示します。クエリロケーターは、ID に似た特殊な文字列です。長さは 18 文字で、プレフィックス 0r8 で始まります。必要に応じて、後続のオブジェクトセットをクエリ結果から取得するために、queryMore() で使用されます。
done	boolean	追加行を、queryMore() を使用して、クエリ結果から取得する必要があるか(false)、ないか(true)を示します。クエリ結果を通して反復処理を実行する際、ループ条件としてこの値を使用します。
records	sObject[]	指定された個別オブジェクトを表していて、queryString で指定された項目リストで定義されたデータを含んでいる sObject の配列。

名前	型	説明
size	int	クエリ内で取得された行の総数。この値を使用すると、クエリが行を取得したか (siza > 0)、否か (size = 0) を判断できます。外部オブジェクトの照会がサポートされていますが、外部データソースから取得した行数は数えられません。外部データソースから取得した行数が不明の場合、size の値は自動的に -1 に設定されます。

queryAll()

削除されたレコードやアーカイブされたレコードなどの指定されたオブジェクトからデータを取得します。

[other]: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

構文

```
QueryResult = connection.queryAll(string queryString);
```

使用方法

queryAll() を使用して、マージまたは削除によって削除されたレコードを識別します。queryAll() には、項目 isDeleted への参照のみアクセス権があります。それ以外は query() と同じです。queryAll() は、undelete() コールでレコードを復元する準備として使用します。

削除されたレコードを探すには、クエリ文字列に isDeleted = true を指定します。マージされたレコードの場合は、masterRecord を要求します。次に例を示します。

```
SELECT id, isDeleted, masterRecordId FROM Account WHERE masterRecordId='100000000000Abc'
```

アーカイブ済みのオブジェクトのみを検索するには、isArchived 項目を絞り込みに利用します。アーカイブ済みのレコードの更新や削除は可能ですが、isArchived 項目を更新することはできません。

メモ: Salesforce では外部データへの変更を追跡しないため、queryAll() コールの動作は、外部オブジェクトに対する query() と同じです。

引数

名前	型	説明
queryString	string	クエリの対象となるオブジェクト、受け取る項目、およびクエリに特定のオブジェクトを含む任意の条件を指定する SOQL クエリが含まれます。『Salesforce SOQL および SOSL リファレンス』を参照してください。

サンプルコード — Java

このサンプルでは、クエリを実行して、取引先が削除されているかどうかに関わらず、すべての取引先を取得します。カスタムバッチサイズを 250 レコードに設定します。最初に `queryAll()`、次に `queryMore()` をコールしてレコードのすべてのバッチを取得します。すべての返された取引先の名前および isDeleted 項目の値をコンソールに書き込みます。

```
public void queryAllRecords() {
   // Setting custom batch size
   connection.setQueryOptions(250);

   try {
      String soqlQuery = "SELECT Name, IsDeleted FROM Account";
      QueryResult qr = connection.queryAll(soqlQuery);
      boolean done = false;
      if (qr.getSize() > 0) {
         System.out.println("Logged-in user can see a total of "
               + qr.getSize()
               + " contact records (including deleted records).");
         while (!done) {
            SObject[] records = qr.getRecords();
            for (int i = 0; i < records.length; i++) {
               Account account = (Account) records[i];
               boolean isDel = account.getIsDeleted();
               System.out.println("Account " + (i + 1) + ": "
                     + account.getName() + " isDeleted = "
                     + account.getIsDeleted());
            }
            if (qr.isDone()) {
               done = true;
            } else {
               qr = connection.queryMore(qr.getQueryLocator());
            }
         }
      } else {
         System.out.println("No records found.");
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、クエリを実行して、取引先が削除されているかどうかに関わらず、すべての取引先を取得します。カスタムバッチサイズを 250 レコードに設定します。最初に `queryAll()`、次に `queryMore()` をコールしてレコードのすべてのバッチを取得します。すべての返された取引先の名前および isDeleted 項目の値をコンソールに書き込みます。

```
public void queryAllRecords()
{
   // Setting custom batch size
```

```
   QueryOptions qo = new QueryOptions();
   qo.batchSize = 250;
   qo.batchSizeSpecified = true;
   binding.QueryOptionsValue = qo;

   try
   {
      String soqlQuery = "SELECT Name, IsDeleted FROM Account";
      QueryResult qr = binding.queryAll(soqlQuery);
      Boolean done = false;
      if (qr.size > 0)
      {
         Console.WriteLine("Logged-in user can see a total of "
               + qr.size
               + " contact records (including deleted records).");
         while (!done)
         {
            sObject[] records = qr.records;
            for (int i = 0; i < records.Length; i++)
            {
               Account account = (Account)records[i];
               Boolean isDel = (Boolean)account.IsDeleted;
               Console.WriteLine("Account " + (i + 1) + ": "
                     + account.Name + " isDeleted = "
                     + account.IsDeleted);
            }
            if (qr.done)
            {
               done = true;
            }
            else
            {
               qr = binding.queryMore(qr.queryLocator);
            }
         }
      }
      else
      {
         Console.WriteLine("No records found.");
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

応答

QueryResult

障害

MalformedQueryFault

InvalidSObjectFault

InvalidFieldFault

UnexpectedErrorFault

関連トピック:

API コールの基礎

queryMore()

queryMore()

クエリ結果の次のバッチを取得します。

構文

```
QueryResult queryResult = connection.queryMore(queryLocator)
```

queryMore() には queryLocator パラメーターが必要です。このパラメーターの値は、先行する query() で返されるクエリロケーター ID です。

使用方法

このコールは、query() または先行する queryMore() コールで多数の結果が返される場合に、その他の結果を取得するために使用します。query() および queryMore() コールは、最大 2,000 レコードをバッチで取得できます。結果の数が、1 つのバッチで返すことができる数より多い場合は、サーバー側のカーソルと QueryLocator (ページ 214) が作成されます。クエリロケーターは、QueryResult で返され、結果の次のバッチを探すために queryMore() で使用されます。

queryMore() コールは、追加のバッチの後続のレコードを処理し、サーバー側カーソルをリセットして、新規に生成されたクエリロケーターを返します。結果セット内のレコードを反復処理するには、結果セット内のすべてのレコードが処理されて Done フラグが true になるまで、queryMore() を繰り返しコールします。

クエリに GROUP BY 句がある場合、queryMore() を使用できません。詳細は、『Salesforce SOQL および SOSL リファレンスガイド』の「GROUP BY」を参照してください。

外部オブジェクトを照会するとき、Salesforce Connect は Web サービスコールアウト経由で外部データにリアルタイムにアクセスし、queryMore() コールを実行するたびにコールアウトが行われます。バッチの区切りとページサイズは、アダプターと外部データ取得元の設定方法に応じて異なります。

次を推奨します。

- 外部オブジェクトのクエリを、デフォルトのバッチサイズである 2,000 行よりも少ない行を返すように絞り込んでページングを避けます。バッチを取得するたびに queryMore() コールが必要になり、その結果、Web サービスコールアウトが行われます。

- 外部データが頻繁に変更される場合は、queryMore() コールの使用を避けてください。次の queryMore() コールまでの間に外部データが変更された場合、予期しない QueryResult になることがあります。

SELECT ステートメントの主オブジェクトが外部オブジェクトの場合、queryMore() は主オブジェクトのみをサポートし、サブクエリをサポートしません。

「Salesforce Connect — OData 2.0 および 4.0 アダプターのクライアント駆動ページングとサーバー駆動ページング」を参照してください。

サンプルコード — Java

このサンプルでは、すべての取引先責任者の名および姓を取得するクエリを実行します。クエリ文字列を渡して、query() をコールし、レコードの最初のバッチを取得します。次に、ループで queryMore() をコールして、レコードが返されなくなるまで、後続のバッチを取得します。照会した取引先責任者の名と姓をコンソールに書き込みます。

```
public void queryRecords() {
   QueryResult qResult = null;
   try {
      String soqlQuery = "SELECT FirstName, LastName FROM Contact";
      qResult = connection.query(soqlQuery);
      boolean done = false;
      if (qResult.getSize() > 0) {
         System.out.println("Logged-in user can see a total of "
            + qResult.getSize() + " contact records.");
         while (!done) {
            SObject[] records = qResult.getRecords();
            for (int i = 0; i < records.length; ++i) {
               Contact con = (Contact) records[i];
               String fName = con.getFirstName();
               String lName = con.getLastName();
               if (fName == null) {
                  System.out.println("Contact " + (i + 1) + ": " + lName);
               } else {
                  System.out.println("Contact " + (i + 1) + ": " + fName
                        + " " + lName);
               }
            }
            if (qResult.isDone()) {
               done = true;
            } else {
               qResult = connection.queryMore(qResult.getQueryLocator());
            }
         }
      } else {
         System.out.println("No records found.");
      }
      System.out.println("\nQuery succesfully executed.");
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、すべての取引先責任者の名および姓を取得するクエリを実行します。クエリ文字列を渡して、`query()` をコールし、レコードの最初のバッチを取得します。次に、ループで `queryMore()` をコールして、レコードが返されなくなるまで、後続のバッチを取得します。照会した取引先責任者の名と姓をコンソールに書き込みます。

```
public void queryRecords()
{
   QueryResult qResult = null;
   try
   {
      String soqlQuery = "SELECT FirstName, LastName FROM Contact";
      qResult = binding.query(soqlQuery);
      Boolean done = false;
      if (qResult.size > 0)
      {
         Console.WriteLine("Logged-in user can see a total of "
            + qResult.size + " contact records.");
         while (!done)
         {
            sObject[] records = qResult.records;
            for (int i = 0; i < records.Length; ++i)
            {
               Contact con = (Contact)records[i];
               String fName = con.FirstName;
               String lName = con.LastName;
               if (fName == null)
               {
                  Console.WriteLine("Contact " + (i + 1) + ": " + lName);
               }
               else
               {
                  Console.WriteLine("Contact " + (i + 1) + ": " + fName
                        + " " + lName);
               }
            }
            if (qResult.done)
            {
               done = true;
            }
            else
            {
               qResult = binding.queryMore(qResult.queryLocator);
            }
         }
      }
      else
      {
         Console.WriteLine("No records found.");
      }
      Console.WriteLine("\nQuery succesfully executed.");
   }
   catch (SoapException e)
```

```
    {
        Console.WriteLine("An unexpected error has occurred: " +
                                    e.Message + "\n" + e.StackTrace);
    }
}
```

引数

名前	型	説明
queryLocator	QueryLocator	データベースのサーバー側カーソルを指します。クエリ結果セットの現在の処理場所を追跡し、追加の結果の場所を示します。 クエリロケーターは、ID に似た特殊な文字列です。長さは 18 文字で、プレフィックス 0r8 で始まります。クエリ結果から後続のオブジェクトセットを取得する場合に必要です。この引数は、必要に応じて使用します。

応答

QueryResult

障害

InvalidQueryLocatorFault

UnexpectedErrorFault

関連トピック:

- query()
- API コールの基礎
- クエリのバッチサイズの変更

retrieve()

指定した ID に基づいて 1 つ以上のレコードを取得します。

構文

```
sObject[] result = connection.retrieve(string fieldList, string sObjectType, ID ids[]);
```

使用方法

retrieve() コールを使用すると、オブジェクトから個別のレコードを取得できます。クライアントアプリケーションは、取得する項目、オブジェクト、および取得するレコード ID の配列を渡します。retrieve() コールは、削除されたレコードを返しません。

一般に、取得するレコードの ID が前もってわかっている場合に、retrieve() を使用します。ID がわかっていないか、他の選択条件を指定する場合は、代わりに query() を使用します。

クライアントアプリケーションは、retrieve() を使用してクライアント側で結合を実行できます。たとえば、クライアントアプリケーションは、query() を実行して、一連の Opportunity レコードを取得し、返された商談レコードを反復処理しながら、商談ごとに accountId を取得し、retrieve() をコールして各 accountId の Account 情報を取得できます。

特定のオブジェクトのレコードは、API からは取得できません。retrieve() コールからレコードを取得するには、そのオブジェクトが取得可能に設定されている (retrieveable が true) 必要があります。取得可能かどうかを決定するために、クライアントアプリケーションは、オブジェクトで describeSObjects() コールを呼び出すことができ、retrievable プロパティを確認することができます。

クライアントアプリケーションは、指定したオブジェクト内のレコードを取得し、指定した項目リストの項目を取得するのに十分なアクセス権限でログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

サンプルコード — Java

このサンプルでは、指定の取引先レコードの「ID」、「Name」、「Website」を取得します。取得したレコードの項目をコンソールに書き込みます。

```
public void retrieveRecords(String[] ids) {
   try {
      SObject[] sObjects = connection.retrieve("ID, Name, Website",
            "Account", ids);
      // Verify that some objects were returned.
      // Even though we began with valid object IDs,
      // someone else might have deleted them in the meantime.
      if (sObjects != null) {
         for (int i = 0; i < sObjects.length; i++) {
            // Cast the SObject into an Account object
            Account retrievedAccount = (Account) sObjects[i];
            if (retrievedAccount != null) {
               System.out.println("Account ID: " + retrievedAccount.getId());
               System.out.println("Account Name: " + retrievedAccount.getName());
               System.out.println("Account Website: "
                     + retrievedAccount.getWebsite());
            }
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、指定の取引先レコードの「ID」、「Name」、「Website」を取得します。取得したレコードの項目をコンソールに書き込みます。

```
public void retrieveRecords(String[] ids)
{
   try
   {
      sObject[] sObjects = binding.retrieve("ID, Name, Website",
            "Account", ids);
      // Verify that some objects were returned.
      // Even though we began with valid object IDs,
      // someone else might have deleted them in the meantime.
      if (sObjects != null)
      {
         for (int i = 0; i < sObjects.Length; i++)
         {
            // Cast the SObject into an Account object
            Account retrievedAccount = (Account)sObjects[i];
            if (retrievedAccount != null)
            {
               Console.WriteLine("Account ID: " + retrievedAccount.Id);
               Console.WriteLine("Account Name: " + retrievedAccount.Name);
               Console.WriteLine("Account Website: "
                     + retrievedAccount.Website);
            }
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

引数

名前	型	説明
fieldList	string	カンマで区切られた指定オブジェクトの 1 つ以上のリスト。有効な項目名を指定する必要があり、各指定項目の参照レベルの権限を持っている必要があります。fieldList は、result 内の項目の順序を定義します。
sObjectType	string	データ取得元のオブジェクト。指定された値は、組織で有効なオブジェクトである必要があります。完全なオブジェクトのセットについては、「標準オブジェクト」を参照してください。

名前	型	説明
ids	ID[]	取得するオブジェクトの 1 つ以上の ID の配列。retrieve() コールには、最大 2000 個のオブジェクト ID を渡すことができます。ID についての詳細は、「ID データ型」を参照してください。

応答

名前	型	説明
result	sObject[]	指定されたオブジェクトの個別レコードを表す、1 つ以上の sObject からなる配列。配列で返される sObject の数は、retrieve() コールに渡す ID の数と同じになります。オブジェクトへのアクセス権を持っていない場合や、渡された ID が無効の場合、配列は、そのオブジェクトに null を返します。ID についての詳細は、「ID データ型」を参照してください。

障害

InvalidSObjectFault

InvalidFieldFault

UnexpectedErrorFault

関連トピック:

API コールの基礎

search()

組織のデータをテキスト検索します。

構文

```
SearchResult = connection.search(String searchString);
```

使用方法

search() は、検索文字列に基づいてレコードを検索する場合に使用します。search コールは、カスタムオブジェクトの検索をサポートしています。テキスト検索に使用される構文と規則に関するさまざまなトピックについては、『Salesforce SOQL および SOSL リファレンスガイド』を参照してください。

Attachment オブジェクトのような、特定のオブジェクトは、API 経由で検索できません。search() コールを使用してオブジェクトを検索するには、オブジェクトが検索可能に設定されている (isSearchable が true) 必要が

あります。検索可能かどうかを判断するために、クライアントアプリケーションは、オブジェクト上の `describeSObjects()` コールを呼び出して、`searchable` プロパティを確認できます。

サンプルコード — Java

このサンプルでは、電話番号項目に指定の値が含まれる取引先責任者、リード、および取引先を返す SOSL クエリを `search()` コールに渡すことにより、このコールを実行します。次に、結果から sObject レコードを取得し、レコードタイプに応じて配列にレコードを保存します。最後に、返された取引先責任者、リード、および取引先の項目をコンソールに書き込みます。

```
public void searchSample() {
            try {
            // Perform the search using the SOSL query.
            SearchResult sr = connection.search(
            "FIND {4159017000} IN Phone FIELDS RETURNING "
            + "Contact(Id, Phone, FirstName, LastName), "
            + "Lead(Id, Phone, FirstName, LastName), "
            + "Account(Id, Phone, Name)");

            // Get the records from the search results.
            SearchRecord[] records = sr.getSearchRecords();

            ArrayList<Contact> contacts = new ArrayList<Contact>();
            ArrayList<Lead> leads = new ArrayList<Lead>();
            ArrayList<Account> accounts = new ArrayList<Account>();

            // For each record returned, find out if it's a
            // contact, lead, or account and add it to the
            // appropriate array, then write the records
            // to the console.
            if (records.length > 0) {
            for (int i = 0; i < records.length; i++) {
            SObject record = records[i].getRecord();
            if (record instanceof Contact) {
            contacts.add((Contact) record);
            } else if (record instanceof Lead) {
            leads.add((Lead) record);
            } else if (record instanceof Account) {
            accounts.add((Account) record);
            }
            }

            System.out.println("Found " + contacts.size() + " contacts.");
            for (Contact c : contacts) {
            System.out.println(c.getId() + ", " + c.getFirstName() + ", "
            + c.getLastName() + ", " + c.getPhone());
            }
            System.out.println("Found " + leads.size() + " leads.");
            for (Lead d : leads) {
            System.out.println(d.getId() + ", " + d.getFirstName() + ", "
            + d.getLastName() + ", " + d.getPhone());
            }
            System.out.println("Found " + accounts.size() + " accounts.");
```

```
            for (Account a : accounts) {
            System.out.println(a.getId() + ", " + a.getName() + ", "
            + a.getPhone());
            }
            } else {
            System.out.println("No records were found for the search.");
            }
            } catch (Exception ce) {
            ce.printStackTrace();
            }
            }
```

サンプルコード — C#

このサンプルでは、電話番号項目に指定の値が含まれる取引先責任者、リード、および取引先を返す SOSL クエリを `search()` コールに渡すことにより、このコールを実行します。次に、結果から sObject レコードを取得し、レコードタイプに応じて配列にレコードを保存します。最後に、返された取引先責任者、リード、および取引先の項目をコンソールに書き込みます。

```
public void searchSample()
            {
            try
            {
            // Perform the search using the SOSL query.
            SearchResult sr = binding.search(
            "FIND {4159017000} IN Phone FIELDS RETURNING "
            + "Contact(Id, Phone, FirstName, LastName), "
            + "Lead(Id, Phone, FirstName, LastName), "
            + "Account(Id, Phone, Name)");

            // Get the records from the search results.
            SearchRecord[] records = sr.searchRecords;

            List<Contact> contacts = new List<Contact>();
            List<Lead> leads = new List<Lead>();
            List<Account> accounts = new List<Account>();

            // For each record returned, find out if it's a
            // contact, lead, or account and add it to the
            // appropriate array, then write the records
            // to the console.
            if (records.Length > 0)
            {
            for (int i = 0; i < records.Length; i++)
            {
            sObject record = records[i].record;
            if (record is Contact)
            {
            contacts.Add((Contact)record);
            }
            else if (record is Lead)
            {
            leads.Add((Lead)record);
```

```
}
else if (record is Account)
{
accounts.Add((Account)record);
}
}

Console.WriteLine("Found " + contacts.Count + " contacts.");
foreach (Contact c in contacts)
{
Console.WriteLine(c.Id + ", " +
c.FirstName + ", " +
c.LastName + ", " +
c.Phone);
}
Console.WriteLine("Found " + leads.Count + " leads.");
foreach (Lead d in leads)
{
Console.WriteLine(d.Id + ", " +
d.FirstName + ", " +
d.LastName + ", " +
d.Phone);
}
Console.WriteLine("Found " + accounts.Count + " accounts.");
foreach (Account a in accounts)
{
Console.WriteLine(a.Id + ", " +
a.Name + ", " +
a.Phone);
}
}
else
{
Console.WriteLine("No records were found for the search.");
}
}
catch (SoapException e)
{
Console.WriteLine("An unexpected error has occurred: " +
e.Message + "\n" + e.StackTrace);
}
}
```

引数

名前	型	説明
search	string	検索されるテキスト式、検索される項目範囲、取得するオブジェクトと項目のリスト、および返すレコードの最大数を指定する検索文字列。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。

応答

SearchResult

障害

InvalidFieldFault

InvalidSObjectFault

MalformedSearchFault

UnexpectedErrorFault

関連トピック:

API コールの基礎

SearchResult

search() コールは SearchResult オブジェクトを返します。

SearchResult オブジェクトには、次の項目があります。

名前	型	説明
queryId	string	SOSL 検索の一意の識別子。
searchRecords	SearchRecord[]	SearchRecord オブジェクトの配列。sObject をそれぞれ含んでいます。
searchResultsMetadata	SearchResultsMetadata	SearchRecords のメタデータ。

SearchRecord

検索から返される個々のレコードを表します。

名前	型	説明
record	sObject	検索で返される個々のレコード。
searchRecordMetadata	SearchRecordMetadata	searchRecords のメタデータ。
snippet	SearchSnippet	検索結果ページで、検索文字列に一致する語が周囲のテキスト内で強調表示されます。

SearchRecordMetadata

レコードレベルでの検索結果のメタデータ。

名前	型	説明
`searchPromoted`	boolean	記事が昇格されたことを検索結果に示します。システム管理者は、昇格済み用語をナレッジ記事に追加して昇格済み検索語を定義します。このようなキーワードを検索するユーザーには、関連付けられた記事が検索結果で最初に表示されます。API バージョン 42.0 以降で使用できます。
`spellCorrected`	boolean	レコードが、スペル修正された検索語と一致することを示します。true の場合にのみ応答に表示されます。

SearchSnippet

記事、ケース、フィード、アイデア検索の検索結果ページに表示される抜粋。

名前	型	説明
`text`	string	検索語の一致を含む抜粋。
`wholeFields`	WholeFields	強調表示された項目のリスト。

WholeFields

検索クエリに一致する語の強調表示を含む、各項目の完全なテキストが含まれます。強調表示された語は `<mark>` タグで囲まれます。

名前	型	説明
`name`	string	強調表示された項目の名前。
`value`	string	強調表示されたテキスト。

SearchResultsMetadata

検索結果のグローバルメタデータ。

名前	型	説明
`entityMetadata`	EntitySearchMetadata	オブジェクトレベルでの検索結果のメタデータ。

EntitySearchMetadata

オブジェクトレベルでの検索結果のメタデータ。

名前	型	説明
`fieldMetadata`	FieldLevelSearchMetadata	項目レベルでの検索結果のメタデータ。

名前	型	説明
searchPromotedMetadata	EntitySearchPromotionMetadata	オブジェクトレベルでの検索語の昇格のメタデータ。API バージョン 42.0 以降で使用できます。
spellCorrectionMetadata	EntitySpellCorrectionMetadata	オブジェクトレベルでのスペル修正のメタデータ。
entityName	string	オブジェクトを特定します。

FieldLevelSearchMetadata

項目レベルでの検索結果のメタデータ。

名前	型	説明
name	string	項目名。
label	string	項目表示ラベル。
type	string	項目のデータ型。

EntitySearchPromotionMetadata

オブジェクトレベルでの検索語の昇格のメタデータ。オブジェクトの少なくとも 1 つの記事が昇格済み結果の場合にのみ応答に表示されます。API バージョン 42.0 以降で使用できます。

名前	型	説明
promotedResultCount	int	オブジェクトレベルでの昇格済み記事の結果の数。

EntitySpellCorrectionMetadata

オブジェクトレベルでのスペル修正のメタデータ。オブジェクトの少なくとも 1 つのレコードがスペル修正された検索語と一致する場合にのみ応答に表示されます。

名前	型	説明
correctedQuery	string	スペル修正された検索語。
hasNonCorrectedResults	boolean	true の場合、ユーザーがスペル修正されていない検索語と一致する少なくとも 1 つのレコードにアクセス権があることを示します。各オブジェクトが異なる値を返すこともあります。

関連トピック:

- WITH SNIPPET
- WITH SPELL_CORRECTION

undelete()

ごみ箱からレコードを復元します。

構文

```
UndeleteResult[] = connection.undelete(ID[] ids );
```

使用方法

このコールを使用して、ごみ箱内のレコードを含め、削除済みのレコードの中で復元可能なレコードを復元します。レコードは、`merge()` または `delete()` コールの結果として、ごみ箱に入れられる場合があります。`queryAll()` コールを使用して、結合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ことができます。

削除する前に、レコードが復元できることを確認する必要があります。復元できないレコードもあります。たとえば、Account レコードは復元できますが、AccountTeamMember レコードは復元できません。オブジェクトが復元できることを確認するには、そのオブジェクトの DescribeSObjectResult の `undeletable` フラグの値が `true` に設定されていることを確認します。

delete コールは、子レコードをカスケード削除しますが、undelete コールは、カスケード削除されたレコードを復元します。たとえば、取引先を削除すると、その取引先に関連するすべての取引先責任者を削除します。

結合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。

メモ: API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了していました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

このコールは、AllOrNoneHeader、AllowFieldTruncationHeader、および CallOptions ヘッダーをサポートしています。

エラー時のロールバック

AllOrNoneHeader ヘッダーを使用すると、すべてのレコードが正常に処理されない限り、すべての変更をロールバックできます。このヘッダーは、API バージョン 20.0 以降で使用できます。正常に処理されないレコードがあった場合に、コールですべての変更をロールバックできます。

サンプルコード — Java

このサンプルでは、`queryAll()` をコールして、直近に削除された取引先を 5 つ取得します。次に、これらの取引先の ID を `undelete()` に渡し、これらの取引先を復元します。最後に、コールの結果を確認して、復元された取引先の ID またはエラーをコンソールに書き込みます。

```
public void undeleteRecords() {
   try {
```

```
      // Get the accounts that were last deleted
      // (up to 5 accounts)
      QueryResult qResult = connection
            .queryAll("SELECT Id, SystemModstamp FROM "
                  + "Account WHERE IsDeleted=true "
                  + "ORDER BY SystemModstamp DESC LIMIT 5");

      String[] Ids = new String[qResult.getSize()];
      // Get the IDs of the deleted records
      for (int i = 0; i < qResult.getSize(); i++) {
         Ids[i] = qResult.getRecords()[i].getId();
      }

      // Restore the records
      UndeleteResult[] undelResults = connection.undelete(Ids);

      // Check the results
      for (UndeleteResult result : undelResults) {
         if (result.isSuccess()) {
            System.out.println("Undeleted Account ID: " + result.getId());
         } else {
            if (result.getErrors().length > 0) {
               System.out.println("Error message: "
                     + result.getErrors()[0].getMessage());
            }
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、`queryAll()` をコールして、直近に削除された取引先を5つ取得します。次に、これらの取引先の ID を `undelete()` に渡し、これらの取引先を復元します。最後に、コールの結果を確認して、復元された取引先の ID またはエラーをコンソールに書き込みます。

```
public void undeleteRecords()
{
   try
   {
      // Get the accounts that were last deleted
      //   (up to 5 accounts)
      QueryResult qResult = binding.queryAll(
            "SELECT Id, SystemModstamp FROM " +
            "Account WHERE IsDeleted=true " +
            "ORDER BY SystemModstamp DESC LIMIT 5");

      String[] Ids = new String[qResult.size];
      // Get the IDs of the deleted records
      for (int i = 0; i < qResult.size; i++)
      {
```

```
            Ids[i] = qResult.records[i].Id;
        }

        // Restore the records
        UndeleteResult[] undelResults = binding.undelete(Ids);

        // Check the results
        foreach (UndeleteResult result in undelResults)
        {
            if (result.success)
            {
                Console.WriteLine("Undeleted Account ID: " +
                      result.id);
            }
            else
            {
                if (result.errors.Length > 0)
                {
                    Console.WriteLine("Error message: " +
                          result.errors[0].message);
                }
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
                                  e.Message + "\n" + e.StackTrace);
    }
}
```

引数

名前	型	説明
ids	ID[]	復元するレコードの ID。

応答

UndeleteResult

障害

UnexpectedErrorFault

関連トピック:

delete()

UndeleteResult

undelete() コールは、次のプロパティを持つ undeleteResult オブジェクトを返します。

名前	型	説明
Id	ID	復元されるレコードの ID。
success	boolean	復元が正常に行われたか (true)、失敗したか (false) を示します。
errors	Error[]	update() コール中に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 Error オブジェクトからなる配列。

update()

組織のデータ内にある 1 つ以上の既存レコードを更新します。

構文

```
SaveResult[] = connection.update(sObject[] sObjects);
```

使用方法

取引先や取引先責任者など、組織のデータ内にある1つ以上の既存レコードを更新するには、このコールを使用します。update() コールは、SQL の UPDATE ステートメントに類似しています。

権限

クライアントアプリケーションには、指定されたオブジェクトと、そのオブジェクト内の個々の項目に対して update() を実行するのに十分な権限でログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

特別な処理

特定のオブジェクト、またこれらのオブジェクト内にある特定の項目には、特別な処理または権限が必要です。たとえば、オブジェクトの親オブジェクトへのアクセス権限も必要になる場合があります。特定のオブジェクトのレコードを更新する前に、「標準オブジェクト」およびSalesforceヘルプの説明を必ず読んでください。

更新可能なオブジェクト

一部のレコードは、API を使用して更新できません。update() コールからレコードを更新するには、そのオブジェクトが更新可能に設定されている (updateable が true) 必要があります。 オブジェクトが更新できる

かどうかを確認するために、オブジェクトに対して `describeSObjects()` コールを実行して `updateable` プロパティを調べることができます。

必須項目

必須項目を更新する場合、値を入力する必要があります。値を `null` に設定することはできません。詳細は、「必須項目」を参照してください。

ID 項目

名前に「`Id`」が含まれている項目は、オブジェクトの主キー(「IDデータ型」を参照)または外部キー(「reference データ型」を参照)のいずれかです。クライアントアプリケーションは、主キーを更新することはできませんが、外部キーを更新することができます。たとえば、`OwnerID` は取引先レコードを所有するユーザーを参照する外部キーであるため、クライアントアプリケーションはAccountの `OwnerId` を更新することができます。`describeSObjects()` を使用して、項目が更新可能かどうかを確認できます。

このコールでは、バッチに重複する `Id` 値があるかどうかチェックし、重複がある場合は、最初の12個を処理します。それ以降の重複 `Id` 値については、それらのエントリの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 (12 allowed).
```

自動的に更新される項目

API は、`LastModifiedDate`、`LastModifiedById`、および `SystemModstamp` などの特定の項目を自動的に更新します。これらの値を `update()` コールで明示的に指定することはできません。

値の null へのリセット

項目値を `null` にリセットするには、sObject の `fieldsToNull` の配列に項目名を追加します。必須項目(`nillable` が `false`)を `null` に設定することはできません。

有効な項目値

整数項目については整数(英字は不可)、項目のデータ型に対して有効な値を入力する必要があります。クライアントアプリケーションでは、使用しているプログラム言語および開発ツールに指定されたデータ形式ルールに従ってください(開発ツールは、SOAP メッセージのデータ型の適切な対応付けを処理します)。

文字列値

String 項目に値を保存する場合、API は先頭および末尾の空白文字を削除します。たとえば、名前項目の値に `" ABC Company "` と入力されると、その値はデータベースに `"ABC Company"` と保存されます。

API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了してい

ました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

割り当てルール

Case または Lead オブジェクトを更新するとき、クライアントアプリケーションで AssignmentRuleHeader オプションを設定して、Salesforce ユーザーインターフェースで設定した割り当てルールに基づいて、ケースまたはリードが1つ以上のユーザーに自動的に割り当てられるようにすることができます。詳細は、「Case」または「Lead」を参照してください。

最大オブジェクト更新数

クライアントアプリケーションは、1 回の `update()` コールで最大 200 レコードを変更できます。200 レコードを超える更新要求がある場合、操作全体が失敗します。

エラー時のロールバック

AllOrNoneHeader ヘッダーを使用すると、すべてのレコードが正常に処理されない限り、すべての変更をロールバックできます。このヘッダーは、API バージョン 20.0 以降で使用できます。正常に処理されないレコードがあった場合に、コールですべての変更をロールバックできます。

Chatter フィードの自動登録

作成するレコードを登録するには、ユーザーは個人設定の [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。自動登録が有効化されている場合、ユーザーは自分が作成したレコードを自動的にフォローし、それらのレコードへの変更が [ホーム] タブの Chatter フィードに表示されます。

レコードの所有者を更新する場合、新しい所有者が Chatter フィード設定でレコードの自動登録が有効化されていない限り、新しい所有者は自動的にレコードに登録されません。前の所有者の登録は自動的には解除されません。新しい所有者のレコードの自動登録が有効化されている場合は、新旧両方の所有者のニュースフィードでレコードへの変更が表示されます。

ユーザーはレコードまたは他のユーザーを登録できます。レコードへの変更とユーザーからの更新は、ユーザーのホームページの Chatter フィードに表示されます。これは、Salesforce で他のユーザーやレコードに加えられた変更の最新状況を把握するのに役立ちます。フィードは、API バージョン 18.0 以降で使用できます。

オブジェクト種別が異なるレコードの更新

API バージョン 20.0 以降では、1 回のコールで、カスタムオブジェクトも含め、複数のオブジェクト種別のレコードを更新できます。たとえば、取引先責任者と取引先を 1 回のコールで更新できます。1 回のコールで、最大 10 種類のオブジェクトのレコードを更新できます。

レコードは、`sObjects` 入力配列に入力された順序で保存されます。

Salesforce では、オブジェクト種別が異なるレコードは複数のチャンクに分けられます。1 つのチャンクは、`sObjects` 入力配列のサブセットで、各チャンクにはオブジェクト種別が同じレコードが含まれます。データ

は、チャンク単位にコミットされます。チャンクに含まれるレコードに関連する Apex トリガーは、チャンクごとに 1 回起動されます。次のレコードのセットを含む `sObjects` 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。

- `account1, account2`
- `contact1, contact2, contact3`
- `case1`
- `account3, account4`
- `contact4`

コールごとに最大 10 個のチャンクを処理できます。`sObjects` 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。

警告: 複数のオブジェクト種別のレコードを更新する場合、いずれかのオブジェクト種別が Salesforce の [設定] 領域の機能に関連していると、1 回のコールで更新することはできません。唯一の例外は次のオブジェクトです。

- カスタム設定のオブジェクト。カスタムオブジェクトに類似しています。詳細は、Salesforce ヘルプの「カスタム設定の作成」を参照してください。
- GroupMember
- Group
- User。次の項目が更新中でない場合のみ。
  - `UserRoleId`
  - `IsActive`
  - `ForecastEnabled`
  - `IsPortalEnabled`
  - `Username`
  - `ProfileId`

`update()` と外部キー

外部 ID 項目を外部キーとして使用することによって、最初に親レコードの ID を照会するのではなく、レコードを更新して他の既存のレコードに関連付ける操作を 1 つの手順で実行できます。外部 ID を外部キーとして使用するには、指定された外部 ID 項目のみを持つ親 sObject のインスタンスに、外部キーを設定します。この外部 ID は親レコードの外部 ID の値と一致する必要があります。

次の Java および C# の例では、`MyExtId__c` というカスタム外部 ID 項目を使用して商談を更新して、既存の取引先に関連付ける方法を示します。各例には更新する商談の ID を受け取るメソッドがあります。このサンプルでは、商談 sObject を作成し、そのオブジェクトが更新対象の既存の商談を指定するよう ID 項目を設定した後、フェーズ項目の新しい値を設定し、外部 ID 項目を取引先オブジェクトに設定します。次に商談を更新します。商談が更新されると、取引先がその商談の親になり、フェーズ名が更新されます。

Java の例

```
public void updateForeignKeySample(String oppId) {
   try {
      Opportunity updateOpportunity = new Opportunity();
      // Point to an existing opportunity to update
      updateOpportunity.setId(oppId);
      updateOpportunity.setStageName("Qualification");

      Account parentAccountRef = new Account();
      parentAccountRef.setMyExtId__c("SAP1111111");
      updateOpportunity.setAccount(parentAccountRef);

      SaveResult[] results = connection
            .update(new SObject[] { updateOpportunity });
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

C# の例

```
public void updateForeignKeySample(String oppId)
{
   try
   {
      Opportunity updateOpportunity = new Opportunity();
      // Point to an existing opportunity to update
      updateOpportunity.Id = oppId;
      updateOpportunity.StageName = "Prospecting";

      Account parentAccountRef = new Account();
      parentAccountRef.MyExtId__c = "SAP1111111";
      updateOpportunity.Account = parentAccountRef;

      SaveResult[] results = binding.update(
         new sObject[] { updateOpportunity });
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
   }
}
```

レコード更新の基本手順

レコードを更新するには、次のプロセスを使用します。

1. update() の対象となる各レコードの ID を確認します。たとえば、query() を特定の検索条件に基づいてコールし、更新するレコード (とその ID) のセットを取得します。更新するレコードの ID がわかっている場合は、retrieve() をコールすることもできます。ID についての詳細は、「ID データ型」を参照してください。
2. レコードごとに sObject を作成し、項目に更新するデータを入力します。

3. sObject[] 配列を作成し、配列に更新するレコードを入力します。
4. update() をコールし、sObject[] 配列を渡します。
5. SaveResult[] オブジェクトの結果を処理してレコードの更新が成功したかどうかを確認します。

サンプルコード — Java

このサンプルでは、更新する取引先の ID を受け取ります。2 つの取引先 sObject を作成し、それぞれの sObjects が既存の取引先を指すよう、受け取った ID を 1 つずつ設定した後、その他の項目を設定します。次に、update() コールを実行し、結果を検証します。

```
public void updateRecords(String[] ids) {
   Account[] updates = new Account[2];

   Account account1 = new Account();
   account1.setId(ids[0]);
   account1.setShippingPostalCode("89044");
   updates[0] = account1;

   Account account2 = new Account();
   account2.setId(ids[1]);
   account2.setNumberOfEmployees(1000);
   updates[1] = account2;

   // Invoke the update call and save the results
   try {
      SaveResult[] saveResults = connection.update(updates);
      for (SaveResult saveResult : saveResults) {
         if (saveResult.isSuccess()) {
            System.out.println("Successfully updated Account ID: "
                  + saveResult.getId());
         } else {
            // Handle the errors.
            // We just print the first error out for sample purposes.
            Error[] errors = saveResult.getErrors();
            if (errors.length > 0) {
               System.out.println("Error: could not update " + "Account ID "
                     + saveResult.getId() + ".");
               System.out.println("\tThe error reported was: ("
                     + errors[0].getStatusCode() + ") "
                     + errors[0].getMessage() + ".");
            }
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、更新する取引先の ID を受け取ります。2 つの取引先 sObject を作成し、それぞれの sObjects が既存の取引先を指すよう、受け取った ID を 1 つずつ設定した後、その他の項目を設定します。次に、update() コールを実行し、結果を検証します。

```
public void updateRecords(String[] ids)
{
   Account[] updates = new Account[2];

   Account account1 = new Account();
   account1.Id = ids[0];
   account1.ShippingPostalCode = "89044";
   updates[0] = account1;

   Account account2 = new Account();
   account2.Id = ids[1];
   account2.NumberOfEmployees = 1000;
   updates[1] = account2;

   // Invoke the update call and save the results
   try
   {
      SaveResult[] saveResults = binding.update(updates);
      foreach (SaveResult saveResult in saveResults)
      {
         if (saveResult.success)
         {
            Console.WriteLine("Successfully updated Account ID: " +
                  saveResult.id);
         }
         else
         {
            // Handle the errors.
            // We just print the first error out for sample purposes.
            Error[] errors = saveResult.errors;
            if (errors.Length > 0)
            {
               Console.WriteLine("Error: could not update " +
                     "Account ID " + saveResult.id + "."
               );
               Console.WriteLine("\tThe error reported was: (" +
                     errors[0].statusCode + ") " +
                     errors[0].message + "."
               );
            }
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                                 e.Message + "\n" + e.StackTrace);
```

```
    }
}
```

引数

名前	型	説明
sObjects	sObject[]	更新する 1 つ以上 (最大 200) のレコードの配列。

応答

SaveResult[]

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

API コールの基礎

SaveResult

update() コールは、SaveResult オブジェクトの配列を返します。

SaveResult 配列の各要素は、update() コールの sObjects パラメーターとして渡された sObject[] 配列に対応します。たとえば、SaveResult 配列の最初のインデックスで返されるオブジェクトは、sObject[] 配列の最初のインデックスで指定されるオブジェクトに一致します。

SaveResult 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	正常に更新された sObject の ID。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されています。この項目が空の場合、オブジェクトは更新されず、API はエラー情報を返しています。
success	boolean	オブジェクトの update() コールが成功したかどうかを示します (成功した場合は true、失敗した場合は false)。
errors	Error[]	update() コール中に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 Error オブジェクトからなる配列。 組織に有効な重複ルールがあり、重複が検出された場合、SaveResult にはデータ型が DuplicateError の Error が含まれます。

upsert()

オブジェクトを作成するか、既存のオブジェクトを更新します。カスタム項目を使用して、既存オブジェクトの有無を判別します。セールスフォース・ドットコムでは、多くの場合、不要な重複レコードが作成されないようにするために、`create()` の代わりに `upsert()` を使用することをお勧めしています。このコールは、API バージョン 7.0 以降で使用できます。

メモ: API バージョン 15.0 以降、文字列を含む項目に値を指定し、値が項目に対して大きすぎる場合、コールは失敗してエラーが返されます。これまでのバージョンの API では、値は切り捨てられ、コールは正常に終了していました。バージョン 15.0 以降でもこの動作を保持する場合、AllowFieldTruncationHeader SOAP ヘッダーを使用してください。

構文

```
UpsertResult[] = connection.upsert(String externalIdFieldName, sObject[] sObjects);
```

使用方法

Upsert は、insert と update の結合したものです。このコールは、オブジェクトに外部 ID 項目または項目プロパティが idLookup の項目がある場合、オブジェクトに使用できます。

カスタムオブジェクトの場合、このコールは、外部 ID というインデックス付けされたカスタム項目を使用して、レコードを作成するか、既存オブジェクトを更新するかを判別します。標準オブジェクトの場合、このコールでは外部 ID の代わりに idLookup を持つ項目の名前を使用できます。

メモ: 外部 ID 項目では、`merge()` を使用することはできません。

外部ID項目など、カスタム項目のオブジェクトへの追加に関する詳細は、Salesforceヘルプの項目の追加に関するトピックを参照してください。

このコールを使用すると、特に次のような場合に、必要なコール数を大幅に減らすことができます。

- 組織の Salesforce データと、経理や製造などの ERP (統合業務ソフト) システムと統合する場合。
- データをインポートして、重複オブジェクトの作成を回避する場合。

[外部 ID] 属性および Unique 属性の両方が選択された (インデックスが一意) カスタム項目を持つオブジェクトのレコードを更新/挿入する場合、Unique 属性によって重複作成が防止されるため、特別な権限は必要ありません。[外部 ID] 属性が選択されている (インデックスが一意ではない) オブジェクトのレコードを更新/挿入する場合、クライアントアプリケーションがこのコールを実行するには「すべてのデータの参照」権限が必要です。

メモ: 外部 ID による一致は、外部 ID 項目で Unique 属性と [「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)] オプションを選択している場合にのみ大文字と小文字の区別をしません。これらのオプションは、項目作成時に Salesforce ユーザーインターフェースで選択されています。これらのオプションが選択されている場合、「ABC123」は「abc123」と一致します。操作を実行する前に、外部 ID 項目で大文字と小文字を区別しないオプションを選択していない場合は、大文字と小文字を考慮しない場合に一致する値の外部 ID を確認します。そのような値が存在する場合、それらの値が一意のものとなるよう変更することを検討したり、外部 ID 項目の大文字と小文字を区別するオプションを選択したり

してください。項目属性についての詳細は、Salesforceヘルプの「カスタム項目の属性」を参照してください。

upsert が update() と create() を判別する方法

upsert では、外部 ID を使用して、レコードを作成するか、既存レコードを更新するかを判別します。

- 外部 ID が一致しない場合、新規レコードが作成されます。
- 外部 ID が一度だけ一致したら、既存レコードが更新されます。
- 外部 ID が複数回一致すると、エラーが報告されます。
- バッチコールで外部 ID が同じ複数のレコードを一括更新すると、それらのレコードは、UpsertResult ファイルでエラーとしてマークされます。レコードは作成されません。

エラー時のロールバック

AllOrNoneHeader ヘッダーを使用すると、すべてのレコードが正常に処理されない限り、すべての変更をロールバックできます。このヘッダーは、API バージョン 20.0 以降で使用できます。

Chatter フィードの自動登録

作成するレコードを登録するには、ユーザーは個人設定の［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。自動登録が有効化されている場合、ユーザーは自分が作成したレコードを自動的にフォローし、それらのレコードへの変更が [ホーム] タブの Chatter フィードに表示されます。

レコードの所有者を更新する場合、新しい所有者がChatterフィード設定でレコードの自動登録が有効化されていない限り、新しい所有者は自動的にレコードに登録されません。前の所有者の登録は自動的には解除されません。新しい所有者のレコードの自動登録が有効化されている場合は、新旧両方の所有者のニュースフィードでレコードへの変更が表示されます。

ユーザーはレコードまたは他のユーザーを登録できます。レコードへの変更とユーザーからの更新は、ユーザーのホームページのChatterフィードに表示されます。これは、Salesforceで他のユーザーやレコードに加えられた変更の最新状況を把握するのに役立ちます。フィードは、API バージョン 18.0 以降で使用できます。

upsert() と外部キー

外部 ID 項目を外部キーとして使用することによって、最初に親レコードの ID を照会するのではなく、レコードを作成または更新して他の既存のレコードに関連付ける操作を 1 つの手順で実行できます。外部ID項目を外部キーとして使用するには、指定された外部 ID 項目のみを持つ親 sObject のインスタンスに、外部キーを設定します。この外部IDは親レコードの外部IDの値と一致する必要があります。`create()` とは異なり、`upsert()` を使用して外部キーで関連する子レコードを作成または更新するには、親レコードが現時点で存在する必要があります。

次の Java および C# の例では、商談を更新/挿入します。この場合、商談はデータベースに存在しないので、`upsert()` コールによって作成されます。この商談は既存の取引先を参照します。個別のクエリを取得する必要がある取引先 ID を指定せず、取引先の外部 ID、この例では `MyExtId__c` カスタム項目を指定します。

Java の例

```
public void upsertForeignKeySample() {
   try {
      Opportunity newOpportunity = new Opportunity();
      newOpportunity.setName("UpsertOpportunity");
      newOpportunity.setStageName("Prospecting");
      Calendar dt = connection.getServerTimestamp().getTimestamp();
      dt.add(Calendar.DAY_OF_MONTH, 7);
      newOpportunity.setCloseDate(dt);
      newOpportunity.setMyExtId__c("UPSERTID001");

      // Parent Account record must already exist
      Account parentAccountRef = new Account();
      parentAccountRef.setMyExtId__c("SAP111111");
      newOpportunity.setAccount(parentAccountRef);

      SaveResult[] results = connection
            .upsert("MyExtId__c", new SObject[] { newOpportunity });
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

C# の例

```
public void upsertForeignKeySample()
{
   try
   {
      Opportunity newOpportunity = new Opportunity();
      newOpportunity.Name = "UpsertOpportunity";
      newOpportunity.StageName = "Prospecting";
      DateTime dt = (DateTime)binding.getServerTimestamp().timestamp;
      newOpportunity.CloseDate = dt.AddDays(7);
      newOpportunity.CloseDateSpecified = true;
      newOpportunity.MyExtId__c = "UPSERTID001";

      // Parent Account record must already exist
      Account parentAccountRef = new Account();
      parentAccountRef.MyExtId__c = "SAP111111";
      newOpportunity.Account = parentAccountRef;

      SaveResult[] results = binding
            .upsert("MyExtId", new sObject[] { newOpportunity });
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                              e.Message + "\n" + e.StackTrace);
   }
}
```

サンプルコード — Java

このサンプルでは、`MyExtId__c` というカスタム外部 ID 項目を使用して 2 つの取引先を更新/挿入します。`upsert()` コールでは、取引先を作成するか更新するかを決定するため、`MyExtId__c` 項目に基づいて取引先を照合します。このサンプルを実行する前に、`MyExtId__c` 項目名を組織の既存のカスタム ID 項目名に変更してください。

```
public void upsertRecords() {
   SObject[] upserts = new Account[2];

   Account upsertAccount1 = new Account();
   upsertAccount1.setName("Begonia");
   upsertAccount1.setIndustry("Education");
   upsertAccount1.setMyExtId__c("1111111111");
   upserts[0] = upsertAccount1;

   Account upsertAccount2 = new Account();
   upsertAccount2 = new Account();
   upsertAccount2.setName("Bluebell");
   upsertAccount2.setIndustry("Technology");
   upsertAccount2.setMyExtId__c("2222222222");
   upserts[1] = upsertAccount2;

   try {
      // Invoke the upsert call and save the results.
      // Use External_Id custom field for matching records.
      UpsertResult[] upsertResults = connection.upsert(
                                 "MyExtId__c", upserts);
      for (UpsertResult result : upsertResults) {
         if (result.isSuccess()) {
            System.out.println("\nUpsert succeeded.");
            System.out.println((result.isCreated() ? "Insert" : "Update")
                  + " was performed.");
            System.out.println("Account ID: " + result.getId());
         } else {
            System.out.println("The Upsert failed because: "
                  + result.getErrors()[0].getMessage());
         }
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、`MyExtId__c` というカスタム外部 ID 項目を使用して 2 つの取引先を更新/挿入します。`upsert()` コールでは、取引先を作成するか更新するかを決定するため、`MyExtId__c` 項目に基づいて取引先

を照合します。このサンプルを実行する前に、`MyExtId__c` 項目名を組織の既存のカスタム ID 項目名に変更してください。

```
public void upsertRecords()
{
   sObject[] upserts = new Account[2];

   Account upsertAccount1 = new Account();
   upsertAccount1.Name = "Begonia";
   upsertAccount1.Industry = "Education";
   upsertAccount1.MyExtId__c = "1111111111";
   upserts[0] = upsertAccount1;

   Account upsertAccount2 = new Account();
   upsertAccount2 = new Account();
   upsertAccount2.Name = "Bluebell";
   upsertAccount2.Industry = "Technology";
   upsertAccount2.MyExtId__c = "2222222222";
   upserts[1] = upsertAccount2;

   try
   {
      // Invoke the upsert call and save the results.
      // Use External_Id custom field for matching records.
      UpsertResult[] upsertResults =
            binding.upsert("MyExtId__c", upserts);
      foreach (UpsertResult result in upsertResults)
      {
         if (result.success)
         {
            Console.WriteLine("\nUpsert succeeded.");
            Console.WriteLine(
                  (result.created ? "Insert" : "Update") +
                  " was performed."
            );
            Console.WriteLine("Account ID: " + result.id);
         }
         else
         {
            Console.WriteLine("The Upsert failed because: " +
                  result.errors[0].message);
         }
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                              e.Message + "\n" + e.StackTrace);
   }
}
```

引数

名前	型	説明
`ExternalIDFieldName`	string	カスタムオブジェクトの外部 ID 項目属性または標準オブジェクトの idLookup 項目プロパティを持つこのオブジェクトの項目名を含みます。idLookup 項目プロパティは通常、オブジェクトの ID 項目または名前項目である項目にありますが、例外があるため、`upsert()` を実行するオブジェクトのプロパティの有無を確認してください。
`sObjects`	sObject[]	作成または更新する 1 つ以上 (最大 200) のレコードの配列。すべてのレコードは同じオブジェクト種別である必要があります。

応答

UpsertResult[]

障害

`InvalidSObjectFault`

`UnexpectedErrorFault`

関連トピック:

- create()
- update()
- API コールの基礎

UpsertResult

`upsert()` コールは、UpsertResult オブジェクトの配列を返します。配列の各要素は、`upsert()` コールの `sObjects` パラメーターとして渡された sObject[] 配列に対応します。たとえば、UpsertResult 配列の最初のインデックスで返されるオブジェクトは、sObject[] 配列の最初のインデックスで指定されるオブジェクトに一致します。

UpsertResult 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
`created`	boolean	レコードが作成されたか (`true`)、更新されたか (`false`) を示します。
`errors`	Error[]	コール時にエラーが発生した場合、エラーコードおよび説明を提供する配列 Error オブジェクトが返されます。
`id`	ID	コールが正常に行われると、項目には、更新または作成されたレコードの ID が入力されます。エラーが発生した場合、項目は Null です。詳細は、「ID データ型」を参照してください。

名前	型	説明
`success`	boolean	このオブジェクトのコールが正常に行われたか(`true`)、失敗したか(`false`)を示します。 組織に有効な重複ルールがあり、重複が検出された場合、UpsertResult にはデータ型が DuplicateError の Error が含まれます。

第 13 章　記述用の API コール (describe)

API でサポートされている記述用の API コールのリストを表示します。

次の表は、API がサポートしている記述用の API コールをアルファベット順に表示し、それぞれの簡単な説明を示しています。コール名をクリックすると、構文、使用方法、各コールの詳細情報を確認できます。

メモ: Apex 関連コールのリストは「Apex 関連コール」、基本となる API コールのリストは「基本となる API コール」、ユーティリティ API コールのリストは「ユーティリティ API コール」を参照してください。

コール	説明
describeAllTabs()	ユーザーが [すべてのタブ] ([+]) タブカスタマイズ機能を使用して自分のユーザーインターフェースでタブを非表示にしているかどうかに関係なく、ログインユーザーが使用できるすべてのタブ (Lightning ページタブを含む) に関する情報を返します。
describeAppMenu()	Salesforce モバイルアプリケーションナビゲーションメニューまたは Salesforce ドロップダウンアプリケーションメニューの項目に関するメタデータを取得します。
describeApprovalLayout()	指定されたオブジェクト種別の承認レイアウトに関するメタデータを取得します。
describeAvailableQuickActions()	API バージョン 28.0 では、指定された親で使用できるアクションの詳細を記述します。API バージョン 29.0 以降では、指定されたコンテキストで使用可能なアクションの詳細を記述します。
describeCompactLayouts()	指定されたオブジェクト種別のコンパクトレイアウトに関するメタデータを取得します。
describeDataCategoryGroups()	要求で指定されたエンティティで使用できるカテゴリグループを取得します。
describeDataCategoryGroupStructures()	要求で指定されたエンティティで使用できるカテゴリグループとそのデータカテゴリ構造を返します。
describeGlobal()	組織のデータで利用可能なオブジェクトの一覧を取得します。
describeGlobalTheme()	現在のログインユーザーが使用できるオブジェクトとテーマの両方の情報を返します。
describeKnowledge()	組織のナレッジ言語設定を取得します。

コール	説明
describeLayout()	指定されたオブジェクト種別のページレイアウトに関するメタデータを取得します。
describePrimaryCompactLayouts()	指定されたオブジェクト種別ごとに主コンパクトレイアウトに関するメタデータを取得します。
describeQuickActions()	指定されたアクションの詳細を取得します。
describeSearchScopeOrder()	ユーザーの検索結果ページの固定表示オブジェクトを含め、ログインユーザーのデフォルトのグローバル検索範囲内にあるオブジェクトの順序付きリストを取得します。
describeSObject()	指定されたオブジェクト種別のメタデータ(項目リストとオブジェクトプロパティ)を表します。describeSObjects() で置き換えられました。
describeSObjects()	describeSObject の配列ベースのバージョン。
describeSoftphoneLayout()	組織のために作成された 1 つ以上のソフトフォンレイアウトの説明です。
describeSoqlListViews()	リストビューに関する SOQL クエリおよびその他の情報を取得します。
describeTabs()	ページ上部の Lightning プラットフォームアプリケーションメニューに表示される、ログインユーザーが使用できる標準アプリケーションおよびカスタムアプリケーションに関する情報を返します。
describeTheme()	現在のログインユーザーが使用できるテーマの情報を返します。

サンプル

このセクションのサンプルは、Enterprise WSDL ファイルに基づいてます。これらのサンプルでは、WSDL ファイルがインポート済みであり、接続が作成済みであることを前提としています。これを実行する方法の詳細については、「クイックスタート」のチュートリアルを参照してください。

describeAllTabs()

ログインユーザーが使用できるすべてのタブ (Lightning ページタブを含む) に関する情報を返します。これは、ユーザーが [すべてのタブ] ([+]) タブカスタマイズ機能を使用して自分のユーザーインターフェースで、タブを非表示にしている場合にも可能です。

構文

```
DescribeTab [] = connection.describeAllTabs();
```

使用方法

ログインユーザーが使用できるすべてのタブに関する情報を取得するのに、`describeAllTabs()` コールを使用します。

または、ログインユーザーのSalesforceユーザーインターフェースに表示されるタブに関する情報のみが必要な場合は、`describeTabs()` を使用します。

サンプルコード — Java

このサンプルでは、`describeAllTabs()` をコールします。これは、DescribeTab の結果の配列を返します。

```
public void describeAllTabsSample() {
  try {
    // Describe tabs
    DescribeTab[] tabs = connection.describeAllTabs();
    System.out.println("There are " + tabs.length +
        " tabs available to you.");

      // Iterate through the returned tabs
      for (int j = 0; j < tabs.length; j++) {
        DescribeTab tab = tabs[j];
        System.out.println("\tTab " + (j + 1) + ":");
        System.out.println("\t\tName: " + tab.getName());
        System.out.println("\t\t\Associated SObject" + tab.getSobjectName());
        System.out.println("\t\tLabel: " + tab.getLabel());
        System.out.println("\t\tURL: " + tab.getUrl());
        DescribeColor[] tabColors = tab.getColors();
        // Iterate through tab colors as needed
        DescribeIcon[] tabIcons = tab.getIcons();
        // Iterate through tab icons as needed
      }
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

引数

なし。

応答

DescribeTab

describeAppMenu()

Salesforceモバイルアプリケーションナビゲーションメニューまたは Salesforce ドロップダウンアプリケーションメニューの項目に関するメタデータを取得します。

このコールは、API バージョン 29.0 以降で使用できます。

カスタムコミュニティ URL を使用して API にアクセスする場合、`describeAppMenu()` コールによって、指定したコミュニティ ID に関連付けられたタブセットが取得されます。

構文

```
DescribeAppMenuResult describeResult = connection.describeAppMenu(String appMenuType,
String networkId);
```

コードサンプル — Java

次のコードサンプルは、Salesforce モバイルアプリケーションナビゲーションメニューからメニュー項目を取得する方法を示します。

```
public void describeAppMenu() {
    try {
        //The following two lines are equivalent
        DescribeAppMenuResult describe = connection.describeAppMenu("Salesforce1", "");
       DescribeAppMenuResult appMenu = getClient().describeAppMenu(AppMenuType.Salesforce1);


        for (DescribeAppMenuItem menuItem : appMenu.getAppMenuItems()) {

            if (menuItem.getType() == "Tab.apexPage") {

                String visualforceUrl = menuItem.getContent();

                System.out.println("URL to Visualforce page: " + visualforceUrl);

            }

        }

    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

引数

名前	型	説明
appMenuType	string	返されるメニューデータを、指定されたメニュー種別に制限します。 有効な値は、次のとおりです。 • AppSwitcher — Salesforce ドロップダウンアプリケーションメニューからデータを取得する

名前	型	説明
		• `Salesforce1` — Salesforce モバイルアプリケーションナビゲーションメニューからデータを取得する • `NetworkTabs` — コミュニティタブセットからデータを取得する
`networkId`	ID	`appMenuType` が `NetworkTabs` に設定されている場合、タブセットの取得元になるコミュニティの ID を入力します。`appMenuType` が `NetworkTabs` でない場合、この項目は null または空である必要があります。

応答

DescribeAppMenuResult

障害

`InvalidOrNullForRestrictedPicklist`

DescribeAppMenuResult

`describeAppMenu()` コールは、指定されたメニュー種別に含まれるメニュー項目のリストを返します。

API バージョン 29.0 以降では次の種別を使用できます。

名前	型	説明
`appMenuItems`	DescribeAppMenuItem[]	選択されたメニュー種別の 1 つ以上のメニュー項目の配列。

DescribeAppMenuItem

各 DescribeAppMenuItem オブジェクトには、次の項目があります。

名前	型	説明
`colors`	DescribeColor[]	メニュー項目に関連付けられたタブで使用される色情報の配列。
`content`	string	メニュー項目の作成に役立つ情報。各メニュー項目は、この項目に異なるタイプのコンテンツを含みます。たとえば、Salesforce アプリケーションメニュー種別には次が含まれる可能性があります。 • FlexiPage — Lightning ページの ID

名前	型	説明
		• Visualforce タブ — ページの URL (`/apex/myApexPage` など)。 その他の種別のメニュー項目では、この項目は使用されません。
`icons`	DescribeColor[]	メニュー項目に関連付けられたタブで使用されるアイコン情報の配列。
`label`	string	メニュー項目の表示ラベル。
`name`	string	メニュー項目の API 名。
`type`	string	メニュー項目の種別およびそのサブ種別(存在する場合)。Salesforce アプリケーションメニュー種別では、次の値を使用できます。 • Standard.Dashboards — [ダッシュボード] メニュー項目 • Standard.Feed — Chatter の [フィード] メニュー項目 • Standard.Today — [Today] メニュー項目 • Standard.Tasks — [ToDo] メニュー項目 • Tab.apexPage — Visualforce タブメニュー項目 • Tab.flexipage — [Lightning ページ] タブメニュー項目
`url`	string	メニュー項目で参照する Salesforce URL。 Salesforce アプリケーションメニュー種別の場合、ダッシュボード、フィード、Today、ToDo、Lightning ページメニュー項目ではこの項目は `null` です。

describeApprovalLayout()

指定されたオブジェクト種別の承認レイアウトに関するメタデータを取得します。

構文

```
DescribeApprovalLayoutResult approvalLayoutResult = connection.describeApprovalLayout(string
 sObjectType, string[] approvalProcessNames);
```

使用方法

このコールを使用して、指定されたオブジェクト種別の承認レイアウトに関する情報を取得します。承認プロセスごとに 1 つの承認レイアウトがあります。

`approvalProcessNames` に null 値を指定した場合、指定された各承認プロセスの承認レイアウトではなく、そのオブジェクトのすべての承認レイアウトが返されます。

サンプルコード — Java

このサンプルでは、取引先 sObject の承認レイアウトを取得する方法を示します。記述する sObject 種別の名前を使用して、`describeApprovalLayout()` をコールします。承認レイアウトの取得後、検出された各承認レイアウトの名前と項目が出力されます。

```
public void describeApprovalLayoutSample() {
  try {
    String objectToDescribe = "Account";
    DescribeApprovalLayoutResult approvalLayoutResult =
      connection.describeApprovalLayout(objectToDescribe, null);
    System.out.print("There are " + approvalLayoutResult.getApprovalLayouts().length);
    System.out.println(" approval layouts for the " + objectToDescribe + " object.");

    // Get all the approval layouts for the sObject
    for (int i = 0; i < approvalLayoutResult.getApprovalLayouts().length; i++) {
      DescribeApprovalLayout aLayout = approvalLayoutResult.getApprovalLayouts()[i];
      System.out.println(" There is an approval layout with name: " + aLayout.getName());

      DescribeLayoutItem[] layoutItems = aLayout.getLayoutItems();
      System.out.print(" There are " + layoutItems.length);
      System.out.println(" fields in this approval layout.");
      for (int j = 0; j < layoutItems.length; j++) {
        System.out.print("This approval layout has a field with name: ");
        System.out.println(layoutItems[j].getLabel());
      }
    }
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

引数

名前	型	説明
`sObjectType`	string	指定された値は、組織で有効なオブジェクトである必要があります。オブジェクトが個人取引先である場合は Account を指定し、個人取引先責任者である場合は Contact を指定します。
`approvalProcessNames`	string[]	承認レイアウトメタデータを返す対象の承認プロセス API 名の配列 (任意指定)。

応答

DescribeApprovalLayoutResult

障害

InvalidSObjectFault

UnexpectedErrorFault

DescribeApprovalLayoutResult

describeApprovalLayout() コールは、渡された sObjectType に関する最上位のレコードタイプ情報を含む DescribeApprovalLayoutResult オブジェクトを返します。

クライアントアプリケーションは、このオブジェクトをトラバースして、承認レイアウトに関する詳細なメタデータを取得できます。

名前	型	説明
approvalLayouts	DescribeApprovalLayout[]	オブジェクトにより使用中のすべての承認レイアウトのリスト。

DescribeApprovalLayout

DescribeApprovalLayout リストの個別の項目を表します。

名前	型	説明
id	ID	この ApprovalLayout の一意の ID。ID についての詳細は、「ID データ型」を参照してください。
label	string	承認レイアウトの表示ラベル。
layoutItems	DescribeLayoutItem[]	承認レイアウトに関連付けられた 1 つ以上の項目の配列。
name	string	承認レイアウトの API 名。

describeAvailableQuickActions()

API バージョン 28.0 では、指定された親で使用できるアクションの詳細を記述します。API バージョン 29.0 以降では、指定されたコンテキストで使用可能なアクションの詳細を記述します。

構文

```
DescribeAvailableQuickActionResult [] = connection.describeAvailableQuickActions(string
parentOrContextType );
```

使用方法

describeAvailableQuickActions() を使用して、親 (API バージョン 28.0) またはコンテキスト (API バージョン 29.0 以降) エンティティ名と、標準およびグローバルアクションが指定されているアクションのリストを取得します。describeAvailableQuickActions() コールは、グローバルアクションに「Account」や「null」などの親エンティティ名を使用し、API バージョン 29.0 以降の場合はコンテキストエンティティ名を使用して、DescribeAvailableQuickActionResult の配列を返します。

サンプル — Java

このサンプルでは、Account オブジェクトについて使用可能なアクション情報を取得して表示します。

```
public void example() throws Exception {
    DescribeAvailableQuickActionResult[] aResult =
        conn.describeAvailableQuickActions("Account");
    for(DescribeAvailableQuickActionResult ar : aResult) {
        System.out.println("Action label: " + ar.getLabel());
        System.out.println("Action name: " + ar.getName());
        System.out.println("Action type: " + ar.getType());
    }
}
```

引数

名前	型	説明
parentOrContextType	string	標準オブジェクトまたはカスタムオブジェクト。 • parentType は API バージョン 28.0 にのみ適用されます。 • contextType は API バージョン 29.0 以降に適用されます。

応答

DescribeAvailableQuickActionResult オブジェクトの配列。

障害

connection.exception エラー

DescribeAvailableQuickActionResult

DescribeAvailableQuickActionResult オブジェクトの配列を返します。

`describeAvailableQuickActions()` コールは、DescribeAvailableQuickActionResult オブジェクトの配列を返します。API バージョン 28.0 では、各 DescribeAvailableQuickActionResult オブジェクトは、指定された親で使用できるアクションの詳細を表します。API バージョン 29.0 以降の各 DescribeAvailableQuickActionResult オブジェクトは、指定されたコンテキストで使用できるアクションの詳細を表します。

名前	型	説明
`actionEnumOrId`	string	アクションの一意の ID。アクションに ID がない場合、API 名が使用されます。 この項目は、API バージョン 35.0 以降で使用できます。
`label`	string	アクションの表示ラベル。
`name`	string	アクション名。
`type`	string	• `LogACall` • `SocialPost` • `Canvas` • `Create` • `VisualforcePage` • `Update`

describeCompactLayouts()

このコールを使用し、指定されたオブジェクト種別のコンパクトレイアウトに関する情報を取得します。

指定されたオブジェクト種別のコンパクトレイアウトに関するメタデータを取得します。

構文

```
DescribeCompactLayoutsResult compactLayoutResult = connection.describeCompactLayouts(string
 sObjectType, ID[] recordTypeId);
```

使用方法

このコールを使用し、指定されたオブジェクト種別のコンパクトレイアウトに関する情報を取得します。このコールは、レコードタイプの対応付けを含む、指定されたコンパクトレイアウトに関するメタデータを返します。コンパクトレイアウトについての詳細は、Salesforce ヘルプを参照してください。

サンプルコード — Java

このサンプルでは、取引先 sObject のコンパクトレイアウトを取得する方法を示します。記述する sObject 種別の名前を使用して、`describeCompactLayouts()` をコールします。コンパクトレイアウトの取得後、各コンパクトレイアウトで検出された画像、項目、およびアクションボタンが出力されます。次に、オブジェクトのシステムデフォルトコンパクトレイアウト、コンパクトレイアウトへのレコードタイプの対応付け情報の順に出力されます。

```
public void testDescribeCompactLayoutsSample() {
    try {
        String objectToDescribe = "Account";
        DescribeCompactLayoutsResult compactLayoutResult = connection
                .describeCompactLayouts(objectToDescribe, null);
        System.out.println("There are " + compactLayoutResult.getCompactLayouts().length
                + " compact layouts for the " + objectToDescribe + " object.");

        // Get all the compact layouts for the sObject
        for (int i = 0; i < compactLayoutResult.getCompactLayouts().length; i++) {
            DescribeCompactLayout cLayout = compactLayoutResult.getCompactLayouts()[i];
           System.out.println(" There is a compact layout with name: " + cLayout.getName());

            DescribeLayoutItem[] fieldItems = cLayout.getFieldItems();
           System.out.println(" There are " + fieldItems.length + " fields in this compact
 layout.");

            // Write field items
            for (int j = 0; j < fieldItems.length; j++) {
                System.out.println(j + " This compact layout has a field with name: " +
fieldItems[j].getLabel());
            }

            DescribeLayoutItem[] imageItems = cLayout.getImageItems();
            System.out.println(" There are " + imageItems.length + " image fields in this
 compact layout.");

            // Write the image items
            for (int j = 0; j < imageItems.length; j++) {
               System.out.println(j + " This compact layout has an image field with name:
 " + imageItems[j].getLabel());
            }

            DescribeLayoutButton[] actions = cLayout.getActions();
            System.out.println(" There are " + actions.length + " buttons in this compact
 layout.");

            // Write the action buttons
            for (int j = 0; j < actions.length; j++) {
                System.out.println(j + " This compact layout has a button with name: " +
actions[j].getLabel());
            }

            System.out.println("This object's default compact layout is: "
                    + compactLayoutResult.getDefaultCompactLayoutId());
```

```
            RecordTypeCompactLayoutMapping[] mappings =
compactLayoutResult.getRecordTypeCompactLayoutMappings();
            System.out.println("There are " + mappings.length + " record type to compact
layout mapping for the "
                    + objectToDescribe + " object.");
            for (int j = 0; j < mappings.length; j++) {
                System.out.println(j + " Record type " + mappings[j].getRecordTypeId()
                        + " is mapped to compact layout " +
mappings[j].getCompactLayoutId());
            }
        }

    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }

}
```

引数

名前	型	説明
sObjectType	string	指定された値は、組織で有効なオブジェクトである必要があります。オブジェクトが個人取引先である場合は Account を指定し、個人取引先責任者である場合は Contact を指定します。
recordTypeId	ID[]	任意で指定できるパラメーター。返されるコンパクトレイアウトデータを、指定されたレコードタイプに制限します。

応答

DescribeCompactLayoutsResult

障害

InvalidSObjectFault

UnexpectedErrorFault

DescribeCompactLayoutsResult

最上位のレコードタイプの情報を含む DescribeCompactLayoutsResult オブジェクトを返します。

describeCompactLayouts() コールは、渡された sObjectType に関する最上位のレコードタイプ情報を含む DescribeCompactLayoutsResult オブジェクトを返します。また、レコードタイプからコンパクトレイアウトへの対応付けも返します。使用しているクライアントアプリケーションは、このオブジェクトを詳細に辿り、コンパクトレイアウトの詳細なメタデータを取得します。

名前	型	説明
compactLayouts	DescribeCompactLayout[]	オブジェクトにより使用中のすべてのコンパクトレイアウトのリスト。
defaultCompactLayoutId	ID	オブジェクトに割り当てられた主コンパクトレイアウトの ID。システムデフォルトコンパクトレイアウト ID の値は null です。
recordTypeCompactLayoutMappings	RecordTypeCompactLayoutMapping[]	オブジェクトのレコードタイプの対応付け。オブジェクトに関連付けられたコンパクトレイアウトは、複数のレコードタイプに対応付けられている場合があります。

DescribeCompactLayout

DescribeCompactLayout リストの個別の項目を表します。

名前	型	説明
actions	DescribeLayoutButtonSection[]	コンパクトレイアウトに関連付けられた 1 つ以上の DescribeLayoutButtonSection 項目の配列。このリストは Salesforce によって設定され、参照のみ可能です。
fieldItems	DescribeLayoutItem[]	コンパクトレイアウトに関連付けられた 1 つ以上の項目の配列。
id	ID	このコンパクトレイアウトの一意の ID。ID についての詳細は、「ID データ型」を参照してください。
imageItems	DescribeLayoutItem[]	コンパクトレイアウトに関連付けられた 1 つ以上の画像の配列。このリストは Salesforce によって設定され、参照のみ可能です。
label	string	コンパクトレイアウトの表示ラベル。
name	string	コンパクトレイアウトの API 名。
objectType	string	コンパクトレイアウトが割り当てられているオブジェクトの名前。

RecordTypeCompactLayoutMapping

DescribeCompactLayoutsResult オブジェクトの recordTypeCompactLayoutMappings 項目の単一のレコードタイプの対応付けを表します。オブジェクトは、有効な recordTypeId の compactLayoutId に対する対応付けです。

名前	型	説明
available	boolean	このレコードタイプが利用可能かどうかを示します (利用可能な場合は true、そうでない場合は false)。利用可能かどうかという情報は、新しいレコードの作成時に利用可能なレコードタイプの一覧をユーザーに表示するのに使用します。
compactLayoutId	ID	このレコードタイプに関連付けられたコンパクトレイアウトの ID。レコードタイプがシステムデフォルトコンパクトレイアウトに関連付けられている場合、この項目の値は null です。
compactLayoutName	string	コンパクトレイアウトの API 名。
recordTypeName	string	レコードタイプの API 名。
recordTypeId	ID	レコードタイプの ID。

describeDataCategoryGroups()

要求で指定されたオブジェクトで使用できるカテゴリグループを取得します。

構文

```
DescribeDataCategoryGroupResult[] = connection.describeDataCategoryGroups()(string[]
sObjectTypes);
```

使用方法

要求で指定されたオブジェクトで使用できるカテゴリグループを取得するには、このコールを使用します。このコールを describeDataCategoryGroupStructures() コールと一緒に使用すると、特定のオブジェクトで使用できるすべてのカテゴリを取得できます。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「データカテゴリの操作」を参照してください。

サンプルコード — Java

このサンプルでは、次に関連付けられているデータカテゴリグループを取得する方法を示します。

- Salesforce ナレッジ記事
- アンサー機能の質問

このサンプルは、カテゴリグループの名前、表示ラベル、および説明と、関連付けられた sobject (記事または質問) の名前を返します。また、データカテゴリグループ内のデータカテゴリ数も返します。

```
public void describeDataCategoryGroupsSample() {
                try {
                // Make the describe call for data category groups
                DescribeDataCategoryGroupResult[] results =
                connection.describeDataCategoryGroups(new String[] {
                "KnowledgeArticleVersion", "Question"});

                // Get the properties of each data category group
                for (int i = 0; i < results.length; i++) {
                System.out.println("sObject: " +
                results[i].getSobject());
                System.out.println("Group name: " +
                results[i].getName());
                System.out.println("Group label: " +
                results[i].getLabel());
                System.out.println("Group description: " +
                (results[i].getDescription()==null? "" :
                results[i].getDescription()));
                System.out.println("Number of categories: " +
                results[i].getCategoryCount());
                }
                } catch (ConnectionException ce) {
                ce.printStackTrace();
                }
                }
```

サンプルコード — C#

このサンプルでは、次に関連付けられているデータカテゴリグループを取得する方法を示します。

- Salesforce ナレッジ記事
- アンサー機能の質問

このサンプルは、カテゴリグループの名前、表示ラベル、および説明と、関連付けられた sobject (記事または質問) の名前を返します。また、データカテゴリグループ内のデータカテゴリ数も返します。

```
public void describeDataCategoryGroups() {
                try {
                // Make the describe call for data category groups
                DescribeDataCategoryGroupResult[] results =
                binding.describeDataCategoryGroups(new String[] {
                "KnowledgeArticleVersion", "Question"});

                // Get the properties of each data category group
                for (int i = 0; i < results.Length; i++) {
                Console.WriteLine("sObject: " +
                results[i].sobject);
                Console.WriteLine("Group name: " +
                results[i].name);
                Console.WriteLine("Group label: " +
```

```
results[i].label);
Console.WriteLine("Group description: " +
(results[i].description==null? "" :
results[i].description));
Console.WriteLine("Number of categories: " +
results[i].categoryCount);
}
} catch (SoapException e) {
Console.WriteLine("An unexpected error has occurred: " +
e.Message + "\n" + e.StackTrace);
}
}
```

引数

名前	型	説明
sObjectTypes	string[]	次のいずれかの値を指定できます。 • 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 — 質問タイプに関連するカテゴリグループを取得します。 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「記事および翻訳の操作」を参照してください。

応答

DescribeDataCategoryGroupResult

障害

InvalidSObjectFault

UnexpectedErrorFault

DescribeDataCategoryGroupResult

describeDataCategoryGroups() コールは、指定の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 DescribeDataCategoryGroupResult オブジェクトを返します。

名前	型	説明
categoryCount	int	データカテゴリグループ内の表示可能なデータカテゴリ数。
description	string	データカテゴリグループの説明。

名前	型	説明
label	string	Salesforceユーザーインターフェースでのデータカテゴリグループの表示ラベル。
name	string	データカテゴリグループへの API アクセスに使用される一意の名前。
sobject	string	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。

describeDataCategoryGroupStructures()

要求で指定されたオブジェクトで使用できるカテゴリグループとそのデータカテゴリ構造を返します。

構文

```
describeDataCategoryGroupStructures()[] = connection.
              describeDataCategoryGroupStructures()(DataCategoryGroupSObjectTypePair[]
 pairs, boolean topCategoriesOnly)
```

使用方法

特定のオブジェクトとカテゴリグループのペアについて表示可能なデータカテゴリ構造を取得するには、このコールを使用します。最初に `describeDataCategoryGroups()` コールを使用して、指定したオブジェクトで使用できるカテゴリグループを検索します。返されたリストから、オブジェクトとカテゴリグループのペアを選択し、`describeDataCategoryGroupStructures()` の入力として渡します。このコールでは、出力としてすべての表示可能なカテゴリとデータカテゴリ構造が返されます。データカテゴリとデータカテゴリの表示設定についての詳細は、Salesforceヘルプの「データカテゴリの操作」および「データカテゴリの表示設定」を参照してください。

サンプルコード — Java

このサンプルでは、sObject とデータカテゴリグループのペアを使用して、各ペアのデータカテゴリを取得する方法を示します。KnowledgeArticleVersion/Regions および Question/Regions という 2 つのペアを使用して `describeDataCategoryGroupStructures()` をコールし、このコールの結果を反復処理します。各結果の最上位のカテゴリ、つまり「すべて」を取得してから、第1レベルの子カテゴリを取得します。このサンプルでは、複数の子カテゴリを持つ *Regions* というデータカテゴリグループを設定して、ナレッジの記事および質問に関連付ける必要があります。または、他の名前を持つ組織の既存のデータカテゴリグループを使用する場合は、サンプルのデータカテゴリグループ名を置き換えることができます。

```
public void describeDataCateogryGroupStructuresSample() {
                try {
                // Create the data category pairs
                DataCategoryGroupSobjectTypePair pair1 =
                new DataCategoryGroupSobjectTypePair();
```

```
DataCategoryGroupSobjectTypePair pair2 =
new DataCategoryGroupSobjectTypePair();
pair1.setSobject("KnowledgeArticleVersion");
pair1.setDataCategoryGroupName("Regions");
pair2.setSobject("Question");
pair2.setDataCategoryGroupName("Regions");

DataCategoryGroupSobjectTypePair[] pairs =
new DataCategoryGroupSobjectTypePair[] {
pair1,
pair2
};

// Get the list of top level categories using the describe call
DescribeDataCategoryGroupStructureResult[] results =
connection.describeDataCategoryGroupStructures(
pairs,
false
);

// Iterate through each result and get some properties
// including top categories and child categories
for (int i = 0; i < results.length; i++) {
DescribeDataCategoryGroupStructureResult result =
results[i];
String sObject = result.getSobject();
System.out.println("sObject: " + sObject);
System.out.println("Group name: " + result.getName());
System.out.println("Group label: " + result.getLabel());
System.out.println("Group description: " +
result.getDescription());

// Get the top-level categories
DataCategory[] topCategories = result.getTopCategories();

// Iterate through the top level categories and retrieve
// some information
for (int j = 0; j < topCategories.length; j++) {
DataCategory topCategory = topCategories[j];
System.out.println("Category name: " +
topCategory.getName());
System.out.println("Category label: " +
topCategory.getLabel());
DataCategory [] childCategories =
topCategory.getChildCategories();
System.out.println("Child categories: ");
for (int k = 0; k < childCategories.length; k++) {
System.out.println("\t" + k + ". Category name: " +
childCategories[k].getName());
System.out.println("\t" + k + ". Category label: " +
childCategories[k].getLabel());
}
}
}
```

```
                } catch (ConnectionException ce) {
                ce.printStackTrace();
                }
                }
```

サンプルコード — C#

このサンプルでは、sObject とデータカテゴリグループのペアを使用して、各ペアのデータカテゴリを取得する方法を示します。KnowledgeArticleVersion/Regions および Question/Regions という 2 つのペアを使用して `describeDataCategoryGroupStructures()` をコールし、このコールの結果を反復処理します。各結果の最上位のカテゴリ、つまり「すべて」を取得してから、第 1 レベルの子カテゴリを取得します。このサンプルでは、複数の子カテゴリを持つ *Regions* というデータカテゴリグループを設定して、ナレッジの記事および質問に関連付ける必要があります。または、他の名前を持つ組織の既存のデータカテゴリグループを使用する場合は、サンプルのデータカテゴリグループ名を置き換えることができます。

```
public void describeDataCateogryGroupStructuresSample() {
                try {
                // Create the data category pairs
                DataCategoryGroupSobjectTypePair pair1 =
                new DataCategoryGroupSobjectTypePair();
                DataCategoryGroupSobjectTypePair pair2 =
                new DataCategoryGroupSobjectTypePair();
                pair1.sobject = "KnowledgeArticleVersion";
                //pair1.setDataCategoryGroupName("Regions");
                pair1.dataCategoryGroupName = "KBArticleCategories";
                pair2.sobject = "Question";
                //pair2.setDataCategoryGroupName("Regions");
                pair2.dataCategoryGroupName = "KBArticleCategories";

                DataCategoryGroupSobjectTypePair[] pairs =
                new DataCategoryGroupSobjectTypePair[] {
                pair1,
                pair2
                };

                // Get the list of top level categories using the describe call
                DescribeDataCategoryGroupStructureResult[] results =
                binding.describeDataCategoryGroupStructures(
                pairs,
                false
                );

                // Iterate through each result and get some properties
                // including top categories and child categories
                for (int i = 0; i < results.Length; i++) {
                DescribeDataCategoryGroupStructureResult result =
                results[i];
                String sObject = result.sobject;
                Console.WriteLine("sObject: " + sObject);
                Console.WriteLine("Group name: " + result.name);
                Console.WriteLine("Group label: " + result.label);
                Console.WriteLine("Group description: " +
```

```
result.description);

// Get the top-level categories
DataCategory[] topCategories = result.topCategories;

// Iterate through the top level categories and retrieve
// some information
for (int j = 0; j < topCategories.Length; j++) {
DataCategory topCategory = topCategories[j];
Console.WriteLine("Category name: " +
topCategory.name);
Console.WriteLine("Category label: " +
topCategory.label);
DataCategory [] childCategories =
topCategory.childCategories;
Console.WriteLine("Child categories: ");
for (int k = 0; k < childCategories.Length; k++) {
Console.WriteLine("\t" + k + ". Category name: " +
childCategories[k].name);
Console.WriteLine("\t" + k + ". Category label: " +
childCategories[k].label);
}
}
}
}
catch (SoapException e)
{
Console.WriteLine("An unexpected error has occurred: " +
e.Message + "\n" + e.StackTrace);
}
}
```

引数

名前	型	説明
`pairs`	DataCategoryGroupSObjectTypePair[]	クエリ対象のカテゴリグループとオブジェクトを指定します。その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。
`topCategoriesOnly`	boolean	ユーザーのデータカテゴリグループ表示設定に応じて、コールが表示可能な最上位カテゴリのみ (`true`) を返すか、またはすべてのカテゴリ (`false`) を返すかを指定します。データカテゴリグループ表示設定についての詳細は、Salesforce ヘルプの「データカテゴリの表示設定」を参照してください。

DataCategoryGroupSObjectTypePair には次の項目が含まれます。

名前	型	説明
dataCategoryGroupName	string	データカテゴリグループへのAPIアクセスに使用される一意の名前。
sobject	string	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

応答

describeDataCategoryGroupStructures()

障害

InvalidSObjectFault

UnexpectedErrorFault

describeDataCategoryGroupStructures()

describeDataCategoryGroupStructures() コールは、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 DescribeDataCategoryGroupStructureResult オブジェクトの配列を返します。

名前	型	説明
description	string	データカテゴリグループの説明。
label	string	Salesforceユーザーインターフェースでのデータカテゴリグループの表示ラベル。
name	string	データカテゴリグループへの API アクセスに使用される一意の名前。
sobject	string	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。
topCategories	DataCategory[]	表示可能な最上位カテゴリのリストは、ユーザーのデータカテゴリグループ表示設定に応じて異なります。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「データカテゴリの表示設定」を参照してください。

DataCategory

名前	型	説明
childDataCategories	DataCategory[]	データカテゴリ内の表示可能なサブカテゴリの再帰的リスト。

名前	型	説明
label	string	Salesforce ユーザーインターフェースでのデータカテゴリの表示ラベル。
name	string	データカテゴリへの API アクセスに使用される一意の名前。

describeGlobal()

組織のデータで利用可能なオブジェクトの一覧を取得します。

構文

```
DescribeGlobalResult = connection.describeGlobal();
```

使用方法

describeGlobal() を使用し、組織で利用可能なオブジェクトの一覧を取得します。リスト内で反復処理し、describeSObjects() を使用して個別のオブジェクトのメタデータを取得できます。

組織のデータのメタデータを取得するには、クライアントアプリケーションは条件を満たすアクセス権限でログインする必要があります。

サンプルコード — Java

このサンプルでは、describeGlobal() の実行方法を示します。次に、返された結果から sObject を取得し、その名前をコンソールに書き込みます。

```
public void describeGlobalSample() {
    try {
        // Make the describeGlobal() call
        DescribeGlobalResult describeGlobalResult =
            connection.describeGlobal();

        // Get the sObjects from the describe global result
        DescribeGlobalSObjectResult[] sobjectResults =
            describeGlobalResult.getSobjects();

        // Write the name of each sObject to the console
        for (int i = 0; i < sobjectResults.length; i++) {
          System.out.println(sobjectResults[i].getName());
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

サンプルコード — C#

このサンプルでは、describeGlobal() の実行方法を示します。次に、返された結果から sObject を取得し、その名前をコンソールに書き込みます。

```
public void describeGlobalSample()
{
    try
    {
        // Make the describeGlobal() call
        DescribeGlobalResult dgr = binding.describeGlobal();

        // Get the sObjects from the describe global result
        DescribeGlobalSObjectResult[] sObjResults = dgr.sobjects;

        // Write the name of each sObject to the console
        for (int i = 0; i < sObjResults.Length; i++)
        {
            Console.WriteLine(sObjResults[i].name);
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

引数

なし。

応答

DescribeGlobalResult

障害

UnexpectedErrorFault

関連トピック:

- describeSObjects()
- API コールの基礎
- Partner WSDL の使用
- https://developer.salesforce.com/page/Sample_SOAP_Messages

DescribeGlobalResult

describeGlobal() コールは DescribeGlobalResult オブジェクトを返します。

DescribeGlobalResult 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
encoding	string	UTF-8 や ISO-8859-1 など、組織データのエンコーディング方法を指定します。
maxBatchSize	int	create()、update() または delete() コールで許容される最大レコード数。
sobjects	DescribeGlobalSObjectResult[]	組織に使用できるオブジェクトについての情報を返す結果オブジェクトの一覧。API バージョン 17.0 以降で利用できます。以前のバージョンの types プロパティで提供されていた情報を強化するものです。
types	string[]	組織で利用可能なオブジェクトの一覧。この一覧の内容を反復処理し、describeSObjects() に渡すオブジェクト文字列を取得します。 API バージョン 17.0 以降では、このプロパティはサポートされていません。代わりに、DescribeGlobalSObjectResult で name プロパティを使用します。

DescribeGlobalSObjectResult

組織で使用できるオブジェクトのいずれかのプロパティを示します。各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
activateable	boolean	将来の使用のために予約されています。
createable	boolean	オブジェクトが create() コールで作成可能か (true)、否か (false) を示します。
custom	boolean	カスタムオブジェクトであるかどうかを示します (カスタムオブジェクトの場合は true、そうでない場合は false)。
customSetting	boolean	カスタム設定項目であるか (true)、否か (false) を示します。
dataTranslationEnabled	boolean	オブジェクトでデータ翻訳が有効になっているか (true)、否か (false) を示します。API バージョン 49.0 以降で利用できます。
deepCloneable	boolean	将来の使用のために予約されています。
deletable	boolean	オブジェクトが delete() コールで削除できるか (true)、否か (false) を示します。

名前	型	説明
deprecatedAndHidden	boolean	将来の使用のために予約されています。
feedEnabled	boolean	オブジェクトで Chatter 項目が有効か (true)、否か (false) を示します。このプロパティは、API バージョン 19.0 以降で使用できます。
isInterface	boolean	将来の使用のために予約されています。
keyPrefix	string	オブジェクト ID の 3 文字のプレフィックスコード。オブジェクト ID はオブジェクト型を示す 3 文字のコードが先頭に付けられます。たとえば、Account オブジェクトのプレフィックスは 001、Opportunity オブジェクトのプレフィックスは 006 です。主要なプレフィックスを複数のオブジェクトで共有する場合があるため、必ずしもオブジェクトを一意に識別するわけではありません。 子に 1 つ以上の親オブジェクトが存在する場合 (多態的な場合)、親のオブジェクト種別を確認するためにこの項目の値を使用します。たとえば、Task または Event の親の keyPrefix 値を取得しなければならない場合にこの値を使用します。
label	string	適用される場合は、ユーザーインターフェースで名前が変更されたタブまたは項目のラベルテキスト。変更されていない場合はオブジェクト名。たとえば、医療分野の組織では「取引先」を Patient に変更する場合があります。タブや項目は Salesforce ユーザーインターフェースで名前を変更できます。詳細は、Salesforce オンラインヘルプを参照してください。
labelPlural	string	オブジェクト名の複数形を表すラベルテキストです。
layoutable	boolean	オブジェクトが describeLayout() コールをサポートしているか (true)、否か (false) を示します。
mergeable	boolean	オブジェクトが同じ型の他のオブジェクトとマージ可能か (true)、否か (false) を示します。リード、取引先責任者、および取引先の場合は true です。
mruEnabled	boolean	最後に使用 (MRU) リスト機能がオブジェクトで有効か (true)、否か (false) を示します。
name	string	オブジェクトの名前。この name は、API バージョン 17.0 以降でサポートされなくなった types リストのエントリに相当します。
queryable	boolean	オブジェクトが query() コールでクエリ可能か (true)、否か (false) を示します。
replicateable	boolean	オブジェクトを getUpdated() および getDeleted() コールで複製可能か (true)、否か (false) を示します。

名前	型	説明
retrieveable	boolean	オブジェクトが retrieve() コールで取得可能か (true)、否か (false) を示します。
searchable	boolean	オブジェクトが search() コールで検索可能か (true)、否か (false) を示します。
triggerable	boolean	オブジェクトが Apex トリガーをサポートしているかどうかを示します。
undeletable	boolean	オブジェクトが undelete() コールで復元可能か (true)、否か (false) を示します。
updateable	boolean	オブジェクトが update() コールで更新可能か (true)、否か (false) を示します。

describeGlobalTheme()

現在のログインユーザーが使用できるオブジェクトとテーマの両方の情報を返します。

構文

```
DescribeGlobalTheme = connection.describeGlobalTheme();
```

使用方法

describeGlobalTheme() を使用して、組織で使用可能なオブジェクトのリストと、それらのオブジェクトに関するテーマ情報の両方を取得します。describeGlobalTheme() は、describeGlobal() と describeTheme() を単一のコールに組み合わせたものです。

組織のデータのテーマとオブジェクト情報を取得するには、条件を満たすアクセス権限でクライアントアプリケーションにログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

describeGlobalTheme() は、API バージョン 29.0 以降で使用できます。

サンプル

この Java のサンプルでは、describeGlobalTheme() をコールした後、取得したオブジェクトとテーマ情報を反復します。

```
public static void describeGlobalThemeExample() {
    try {
        // Get current theme and object information
        DescribeGlobalTheme globalThemeResult = connection.describeGlobalTheme();
        DescribeGlobalResult globalResult = globalThemeResult.getGlobal();
```

```
        DescribeThemeResult globalTheme = globalThemeResult.getTheme();
        // For the themes, get the array of theme items, one per object
        DescribeThemeItem[] themeItems = globalTheme.getThemeItems();
        for (int i = 0; i < themeItems.length; i++) {
            DescribeThemeItem themeItem = themeItems[i];
            System.out.println("Theme information for object " + themeItem.getName());
            // Get color and icon info for each themeItem
            DescribeColor colors[] = themeItem.getColors();
            System.out.println("    Number of colors: " + colors.length);
            int k;
            for (k = 0; k < colors.length; k++) {
                DescribeColor color = colors[k];
                System.out.println("    For Color #" + k + ":");
                System.out.println("      Web RGB Color: " + color.getColor());
                System.out.println("      Context: " + color.getContext());
                System.out.println("      Theme: " + color.getTheme());
            }
            DescribeIcon icons[] = themeItem.getIcons();
            System.out.println("    Number of icons: " + icons.length);
            for (k = 0; k < icons.length; k++) {
                DescribeIcon icon = icons[k];
                System.out.println("    For Icon #" + k + ":");
                System.out.println("      ContentType: " + icon.getContentType());
                System.out.println("      Height: " + icon.getHeight());
                System.out.println("      Theme: " + icon.getTheme());
                System.out.println("      URL: " + icon.getUrl());
                System.out.println("      Width: " + icon.getWidth());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

応答

DescribeGlobalTheme

障害

UnexpectedErrorFault

関連トピック:

- DescribeGlobalTheme
- DescribeThemeResult
- DescribeThemeItem
- DescribeColor
- DescribeIcon

DescribeGlobalTheme

DescribeGlobalTheme オブジェクトを返します。

`describeGlobalTheme()` コールは、DescribeThemeResult と DescribeGlobalResult を含む DescribeGlobalTheme を返します。

名前	型	説明
global	DescribeGlobalResult	オブジェクト情報。
theme	DescribeThemeResult	テーマ情報。

describeKnowledge()

組織のナレッジ言語設定を取得します。

構文

```
KnowledgeSettings result = _connection.describeKnowledgeSettings();
```

使用方法

デフォルトのナレッジ言語、サポート対象言語、およびナレッジ言語情報のリストなどの既存のナレッジ言語設定を取得するには、このコールを使用します。メタデータ API の KnowledgeSettings を使用して、同様の情報を取得することもできます。

サンプルコード — Java

このサンプルでは、ナレッジ言語設定の取得方法を示します。デフォルトのナレッジ言語、ナレッジでサポートされる言語のリスト (言語コードを含む) および有効なナレッジ言語かどうかを返します。

```
public void describeKnowledgeSettingsSample() {
try {

// Make the describe call for KnowledgeSettings
KnowledgeSettings result = connection.describeKnowledgeSettings();

// Get the properties of KnowledgeSettings
System.out.println("Knowledge default language: " + result.getDefaultLanguage());
for (KnowledgeLanguageItem lang : result.getLanguages()) {
System.out.println("Language: " + lang.getName());
System.out.println("Active: " + lang.isActive());
}
} catch (ConnectionException ex) {
ex.printStackTrace();
}
}
```

サンプルコード — C#

このサンプルでは、ナレッジ言語設定の取得方法を示します。デフォルトのナレッジ言語、ナレッジでサポートされる言語のリスト (言語コードを含む) および有効なナレッジ言語かどうかを返します。

```
public void describeKnowledgeSettingsSample() {
try {

// Make the describe call for KnowledgeSettings
KnowledgeSettings result = connection.describeKnowledgeSettings();

// Get the properties of KnowledgeSettings
Console.WriteLine("Knowledge default language: " + result.getDefaultLanguage());
for (KnowledgeLanguageItem lang : result.getLanguages()) {
Console.WriteLine("Language: " + lang.getName());
Console.WriteLine("Active: " + lang.isActive());
}
} catch (SoapException ex) {
ex.printStackTrace();
}
}
```

応答

KnowledgeSettings

describeLayout()

指定されたオブジェクト種別のページレイアウトに関するメタデータを取得します。

構文

```
DescribeLayoutResult = connection.describeLayout(string sObjectType, string layoutName,
ID recordTypeID[]);
```

使用方法

このコールを使用し、指定されたオブジェクト種別のレイアウト情報(ユーザーへのデータの表示方法)を取得します。このコールは、詳細ページレイアウト、編集ページレイアウトおよびレコードタイプの関連付けなど、指定されたページレイアウトのメタデータを返します。詳細は、Salesforceヘルプの「ページレイアウト」を参照してください。

一般的に、ユーザープロファイルには各オブジェクトに 1 つのレイアウトが関連付けられています。Enterprise Edition、Unlimited Edition、および Performance Edition では、ユーザープロファイルはオブジェクトごとに複数のレイアウトを使用でき、各レイアウトは指定されたレコードタイプに固有です。このコールは、適用される場合複数のレイアウトのメタデータを返します。

定義された名前付きレイアウトがある標準オブジェクトではレイアウトをさらにカスタマイズできます。名前付きレイアウトは、プロファイルとレコードタイプの両方で主レイアウトとは分離されています。名前付きレイアウトの例として、User オブジェクトで定義される UserAlt レイアウトが挙げられます。これは、主 User レイアウトではなく、Salesforce モバイルアプリケーションで使用されます。新規レイアウト名は、Salesforce でのみ定義できますが、名前付きレイアウトのカスタマイズは主レイアウトと同様にシステム管理者によって制御されます。

`recordTypeIds` に null 値を指定した場合、指定されたレコードタイプに対するレイアウトだけではなく、そのユーザーのすべてのレイアウトが返されます。同じレイアウトが、ユーザープロファイルの複数のレコードタイプに割り当てられていることもあるため、その場合は 1 つのレイアウトのみが返されます。

メモ: このコールは複雑な API コールであり、通常は、特殊な端末 (PDA など) のために、カスタムページ表示コードを記述したパートナーが使用します。また、ページの出力結果を表示する前に、レイアウトの詳細を確認する必要があります。

レイアウトを記述する手順は次のとおりです。

1. 既存のレコードの詳細ページまたは編集ページを表示するには、クライアントアプリケーションはまずレコードの `recordTypeIds` を取得し、recordTypeMapping を使用してその `recordTypeIds` に関連付けられた layoutId を見つけます。そのレイアウト情報を使用し、最終的にページを表示します。
2. 編集ページの作成バージョンを表示するには、クライアントアプリケーションは最初に 1 つ以上のレコードタイプが利用可能かどうかを確認します。利用可能な場合、ユーザーに選択肢を表示します。レコードタイプを選択すると、クライアントアプリケーションはレイアウト情報を使用しページを表示します。また、RecordTypeMapping の選択リストの値を使用し、選択リスト項目で使用する有効な選択リスト値を表示します。
3. クライアントアプリケーションは DescribeLayoutResult を使用してレイアウトのラベルにアクセスできるようになります。

個人取引先レコードタイプでは次の制限が適用されます。

- バージョン 7.0 以前の `describeLayout()` は、タブのデフォルトが個人取引先レコードタイプである場合も、デフォルトのレコードタイプとして法人取引先レコードタイプを返します。バージョン 8.0 以降では、デフォルトのレコードタイプは常にタブのデフォルトとなります。
- バージョン 7.0 以前では、`describeLayout()` は個人取引先レコードタイプは一切返しません。

個人取引先レコードタイプについての詳細は、「個人取引先のレコードタイプ」を参照してください。

サンプルコード — Java

このサンプルでは、取引先 sObject のレイアウトを取得する方法を示します。記述する sObject 種別の名前を使用して、`describeLayout()` をコールします。このサンプルでは、3 番目の引数としてレコードタイプ ID を指定しません。つまり、指定された sObject に対して組織でレコードタイプが定義されている場合は、すべてのレコードタイプのレイアウトが返されます。レイアウトの取得後に、このサンプルでは、見つかった詳細セクションと編集セクションの数とそのヘッダーを書き込みます。次に、編集レイアウトセクションごとに反復処理を行い、そのコンポーネントを取得します。

```
public void describeLayoutSample(){
  try {
      String objectToDescribe = "Account";
```

```
DescribeLayoutResult dlr =
    connection.describeLayout(objectToDescribe, null, null);
System.out.println("There are " + dlr.getLayouts().length +
    " layouts for the " + objectToDescribe + " object."
);

// Get all the layouts for the sObject
for(int i = 0; i < dlr.getLayouts().length; i++) {
  DescribeLayout layout = dlr.getLayouts()[i];
  DescribeLayoutSection[] detailLayoutSectionList =
    layout.getDetailLayoutSections();
  System.out.println(" There are " +
    detailLayoutSectionList.length +
      " detail layout sections");
  DescribeLayoutSection[] editLayoutSectionList =
    layout.getEditLayoutSections();
  System.out.println(" There are " +
    editLayoutSectionList.length +
    " edit layout sections");

  // Write the headings of the detail layout sections
  for(int j = 0; j < detailLayoutSectionList.length; j++) {
    System.out.println(j +
        " This detail layout section has a heading of " +
        detailLayoutSectionList[j].getHeading());
  }

  // Write the headings of the edit layout sections
  for(int x = 0; x < editLayoutSectionList.length; x++) {
    System.out.println(x +
        " This edit layout section has a heading of " +
        editLayoutSectionList[x].getHeading());
  }

  // For each edit layout section, get its details.
  for(int k = 0; k < editLayoutSectionList.length; k++) {
    DescribeLayoutSection els =
      editLayoutSectionList[k];
    System.out.println("Edit layout section heading: " +
        els.getHeading());
    DescribeLayoutRow[] dlrList = els.getLayoutRows();
      System.out.println("This edit layout section has " +
          dlrList.length + " layout rows.");
      for(int m = 0; m < dlrList.length; m++) {
        DescribeLayoutRow lr = dlrList[m];
        System.out.println(" This row has " +
            lr.getNumItems() + " layout items.");
        DescribeLayoutItem[] dliList = lr.getLayoutItems();
        for(int n = 0; n < dliList.length; n++) {
          DescribeLayoutItem li = dliList[n];
          if ((li.getLayoutComponents() != null) &&
            (li.getLayoutComponents().length > 0)) {
            System.out.println("\tLayout item " + n +
                ", layout component: " +
```

```
                    li.getLayoutComponents()[0].getValue());
                }
                else {
                System.out.println("\tLayout item " + n +
                    ", no layout component");
                }
              }
            }
          }
      }

      // Get record type mappings
      if (dlr.getRecordTypeMappings() != null) {
        System.out.println("There are " +
            dlr.getRecordTypeMappings().length +
            " record type mappings for the " +
            objectToDescribe + " object"
        );
      } else {
        System.out.println(
            "There are no record type mappings for the " +
            objectToDescribe + " object."
        );
      }
    } catch (ConnectionException ce) {
      ce.printStackTrace();
    }
}
```

サンプルコード — C#

このサンプルでは、取引先 sObject のレイアウトを取得する方法を示します。記述する sObject 種別の名前を使用して、`describeLayout()` をコールします。このサンプルでは、3 番目の引数としてレコードタイプ ID を指定しません。つまり、指定された sObject に対して組織でレコードタイプが定義されている場合は、すべてのレコードタイプのレイアウトが返されます。レイアウトの取得後に、このサンプルでは、見つかった詳細セクションと編集セクションの数とそのヘッダーを書き込みます。次に、編集レイアウトセクションごとに反復処理を行い、そのコンポーネントを取得します。

```
public void describeLayoutSample()
{
    try
    {
        String objectToDescribe = "Account";
        DescribeLayoutResult dlr =
            binding.describeLayout(objectToDescribe, null, null);
        Console.WriteLine("There are " + dlr.layouts.Length +
            " layouts for the " + objectToDescribe + " object."
        );

        // Get all the layouts for the sObject
        for (int i = 0; i < dlr.layouts.Length; i++)
        {
```

```
DescribeLayout layout = dlr.layouts[i];
DescribeLayoutSection[] detailLayoutSectionList =
    layout.detailLayoutSections;
Console.WriteLine(" There are " +
    detailLayoutSectionList.Length +
    " detail layout sections");
DescribeLayoutSection[] editLayoutSectionList =
    layout.editLayoutSections;
Console.WriteLine(" There are " +
    editLayoutSectionList.Length +
    " edit layout sections");

// Write the headings of the detail layout sections
for (int j = 0; j < detailLayoutSectionList.Length; j++)
{
    Console.WriteLine(j +
        " This detail layout section has a heading of " +
        detailLayoutSectionList[j].heading);
}

// Write the headings of the edit layout sections
for (int x = 0; x < editLayoutSectionList.Length; x++)
{
    Console.WriteLine(x +
        " This edit layout section has a heading of " +
        editLayoutSectionList[x].heading);
}

// For each edit layout, get its details.
for (int k = 0; k < editLayoutSectionList.Length; k++)
{
    DescribeLayoutSection els =
        editLayoutSectionList[k];
    Console.WriteLine("Edit layout section heading: " +
            els.heading);
    DescribeLayoutRow[] dlrList = els.layoutRows;
    Console.WriteLine("This edit layout section has " +
            dlrList.Length + " layout rows.");
    for (int m = 0; m < dlrList.Length; m++)
    {
        DescribeLayoutRow lr = dlrList[m];
        Console.WriteLine(" This row has " +
            lr.numItems + " layout items.");
        DescribeLayoutItem[] dliList = lr.layoutItems;
        for (int n = 0; n < dliList.Length; n++)
        {
            DescribeLayoutItem li = dliList[n];
            if ((li.layoutComponents != null) &&
                    (li.layoutComponents.Length > 0))
            {
                Console.WriteLine("\tLayout item " + n +
                        ", layout component: " +
                    li.layoutComponents[0].value);
            }
```

```
                        else
                        {
                            Console.WriteLine("\tLayout item " + n +
                                    ", no layout component");
                        }
                    }
                }
            }

            // Get record type mappings
            if (dlr.recordTypeMappings != null)
            {
                Console.WriteLine("There are " +
                    dlr.recordTypeMappings.Length +
                    " record type mappings for the " +
                    objectToDescribe + " object");
            }
            else
            {
                Console.WriteLine(
                    "There are no record type mappings for the " +
                    objectToDescribe + " object.");
            }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

引数

名前	型	説明
sObjectType	string	**指定された値は、組織で有効なオブジェクトである必要があります。オブジェクトが個人取引先である場合は** Account **を指定し、個人取引先責任者である場合は** Contact **を指定します。**
layoutName	string	**指定された値は、このオブジェクトの有効な名前付きレイアウトである必要があります。レイアウト名は、**DescribeSObjectResult **の** namedLayoutInfos **から取得されます。主レイアウトは「名前付き」とは見なされないため、エンティティ名は有効ではありません。**
recordTypeIds	ID[]	**任意で指定できるパラメーターにより指定されたレコードタイプに返されたレイアウトデータを制限します。** **プライマリレコードタイプのレイアウトを取得するには、オブジェクトに関わらず** recordTypeIds **に値** 012000000000000AAA **を指定します。この値は、**DescribeSObjectResult **でプライマリレコードタイプの** recordTypeInfos

名前	型	説明
		として返されます。SOQL クエリでは、`012000000000000AAA` ではなく null 値を返します。 ID についての詳細は、「ID データ型」を参照してください。

応答

DescribeLayoutResult

障害

`InvalidSObjectFault`

`UnexpectedErrorFault`

関連トピック:

API コールの基礎

https://developer.salesforce.com/page/Sample_SOAP_Messages

DescribeLayoutResult

`describeLayout()` コールは、渡された sObjectType に関する最上位のレコードタイプ情報を含む DescribeLayoutResult オブジェクトを返します。また、レコードタイプからレイアウトへの対応付けも返します。使用しているクライアントアプリケーションは、このオブジェクトを詳細に辿り、レイアウトの詳細なメタデータを取得します。

ヒント: 組織でパブリッシャーのアクションが有効になっている場合、sObjectType として `Global`、`recordTypeId` として `null` を使用することで、グローバルパブリッシャーレイアウトのレイアウト定義を取得できます。

DescribeLayoutResult オブジェクトには、次の項目があります。

名前	型	説明
`feedView`	DescribeLayoutFeedView[]	フィードベースのレイアウトのフィードビュー関連レイアウトデータ。この項目は、フィードベースではないページレイアウトでは null になります。
`layouts`	DescribeLayout[]	指定された sObjectType と関連付けられたレイアウト。通常、レイアウトとオブジェクトは 1 対 1 対応です。ただし、場合によっては 1 つのオブジェクトにユーザープロファイルによって異なる複数のレイアウトが割り当てられている場合があります。

名前	型	説明
recordTypeMappings	RecordTypeMapping[]	このユーザーにはレコードタイプの対応付けを利用できます。ユーザープロファイルのオブジェクトには複数のレコードタイプが存在する場合があります。コールしたユーザーが利用可能なレコードタイプだけではなく、すべてのレコードタイプが返されます。すべてのレコードタイプを返すことで、クライアントアプリケーションは指定されたユーザープロファイルに適切なレイアウトで表示できます。たとえば、ユーザー A がレコードの所有者であり、このレコードにはレコードタイプ X が設定されているとします。ユーザー B がレコードを表示しようとした場合、クライアントアプリケーションはユーザー B のプロファイルのレコードタイプに関連付けられたレイアウトを使用しレコードを表示します (そのユーザーが利用できないレコードタイプの場合も同様)。
recordTypeSelectorRequired	boolean	true の場合、レコードタイプ選択ページが必要です。false の場合、デフォルトのレコードタイプを使用します。

DescribeLayout

指定された sObjectType 固有のレイアウトを表します。各 DescribeLayout は一意なレイアウト ID を使用して参照し、2 つのタイプのビューが存在します (DescribeLayoutSection の配列としてこのオブジェクトで表されます)。

- 詳細ビュー — オブジェクトを参照のみの形式で表示します。詳細レイアウトでは、特定の情報 (詳細な住所など) を 1 つの DescribeLayoutItem にまとめることができます。
- 編集ビュー — オブジェクトを編集可能な形式で表示します。編集レイアウトでは、個々の情報 (住所など) はそれぞれの項目に分けられます。

個々の DescribeLayout には、次の項目があります。

名前	型	説明
buttonLayoutSection	DescribeLayoutButtonSection[]	指定されたレイアウトに関連付けられた標準ボタンとカスタムボタンのセクション。
detailLayoutSections	DescribeLayoutSection[]	詳細ビューのレイアウトセクション。
editLayoutSections	DescribeLayoutSection[]	編集ビューのレイアウトセクション。
highlightsPanelLayoutSection	DescribeLayoutSection[]	強調表示パネルビューのレイアウトセクション。

名前	型	説明
multirowEditLayoutSections	DescribeLayoutSection[]	複数行レイアウトビューのレイアウトセクション。この項目は、API バージョン 35.0 以降で使用できます。
id	ID	このレイアウトの一意の ID。ID についての詳細は、「ID データ型」を参照してください。
quickActionList	DescribeQuickActionListResult	指定されたレイアウトに関連付けられたアクションのリスト。この項目は、API バージョン 28.0 以降で使用できます。
relatedContent	RelatedContent	指定されたレイアウトに関連付けられたモバイルカードセクション。この項目は、API バージョン 29.0 以降で使用できます。
relatedLists	RelatedList[]	指定されたレイアウトと関連付けられた関連リスト。
saveOptions	DescribeLayoutSaveOption[]	レイアウトの保存オプションのリスト。

DescribeLayoutButtonSection

標準ボタンまたはカスタムボタンのいずれかを含むレイアウトの 2 つのセクションのいずれかを表します。

名前	型	説明
detailButtons	DescribeLayoutButton[]	指定されたボタンセクションに関連付けられた標準ボタンまたはカスタムボタン。

DescribeLayoutButton

DescribeLayout の単一の標準ボタン、カスタムボタン、またはカスタムリンクを表します。

名前	型	説明
behavior	WebLinkWindowType	ボタンまたはリンクをクリックしたときの動作 (JavaScript を実行したり、そのコンテンツソースを新しいウィンドウで開いたりする動作)。 この項目は、API バージョン 31.0 以降で使用できます。
colors	DescribeColor[]	このボタンまたはリンクに関連付けられたアイコンの色情報の配列。各色はテーマに関連付けられています。 この項目は、API バージョン 32.0 以降で使用できます。

名前	型	説明
`content`	string	配信される Visualforce ページまたは S コントロールの API 名。 この項目は、API バージョン 31.0 以降で使用できます。
`contentSource`	WebLinkType	カスタムボタンまたはリンクのコンテンツソース。上書きされていない標準ボタンの `contentSource` は、`null` です。 この項目は、API バージョン 31.0 以降で使用できます。
`custom`	boolean	必須。カスタムボタンまたはリンクであるか (`true`)、否か (`false`) を示します。
`encoding`	string	ボタンまたはリンクでコールされる URL に割り当てられているエンコードタイプ。有効な値は、次のとおりです。 • `UTF-8` — Unicode (UTF-8) • `ISO-8859-1` — 米国一般および西ヨーロッパ (ISO-8859-1、ISO-LATIN-1) • `Shift_JIS` — 日本語 (Shift-JIS) • `ISO-2022-JP` — 日本語 (JIS) • `EUC-JP` — 日本語 (EUC-JP) • `x-SJIS_0213` — 日本語 (Shift-JIS_2004) • `ks_c_5601-1987` — 韓国語 (ks_c_5601-1987) • `Big5` — 繁体字中国語 (Big5) • `GB2312` — 簡体字中国語 (GB2312) • `Big5-HKSCS` — 繁体字中国語香港 (Big5–HKSCS) この項目は、API バージョン 31.0 以降で使用できます。
`height`	int	ボタンまたはリンクの `behavior` 項目値が `newWindow`、`sidebar`、または `noSidebar` に設定されている場合の高さ (ピクセル)。 この項目は、API バージョン 31.0 以降で使用できます。
`icons`	DescribeIcon[]	このボタンまたはリンクのアイコンの配列。各アイコンはテーマに関連付けられています。この項目は、API バージョン 29.0 以降で使用できます。
`label`	string	Salesforce ユーザーインターフェースに表示されたボタンまたはリンクの表示ラベル。

名前	型	説明
menubar	boolean	ボタンまたはリンクの behavior 項目値が newWindow に設定されている場合にメニューバーが表示されるか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
name	string	ボタンまたはリンクの API 名。
overridden	boolean	必須。標準ボタンが上書きされているか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
resizeable	boolean	ボタンまたはリンクの behavior 項目値が newWindow に設定されている場合に新しいウィンドウのサイズを変更できるか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
scrollbars	boolean	ボタンまたはリンクの behavior 項目値が newWindow に設定されている場合にスクロールバーが表示されるか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
showsLocation	boolean	ボタンまたはリンクの behavior 項目値が newWindow に設定されている場合にアドレスバーが表示されるか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
showsStatus	boolean	ボタンまたはリンクの behavior 項目値が newWindow に設定されている場合にステータスバーが表示されるか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
toolbar	boolean	ボタンまたはリンクの behavior 項目値が newWindow に設定されている場合にツールバーが表示されるか (true)、否か (false) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
url	string	ボタンまたはリンクでコールされる URL。関連リストの標準ボタンの場合、この項目は null です。 この項目は、API バージョン 31.0 以降で使用できます。

名前	型	説明
`width`	int	ボタンまたはリンクの `behavior` 項目値が `newWindow` に設定されている場合の幅 (ピクセル)。 この項目は、API バージョン 31.0 以降で使用できます。
`windowPosition`	WebLinkPosition	ボタンまたはリンクの `behavior` 項目値が `newWindow` に設定されている場合のウィンドウの位置を示します。 この項目は、API バージョン 31.0 以降で使用できます。

DescribeLayoutComponent

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。表示するための項目の参照において、クライアントアプリケーションは次の表記法を使用し describeSObjects() コールの項目を参照します。`LayoutComponent.fieldName`。

API バージョン 31.0 以降では、LayoutComponentType 値が `Field` で、記述される項目が `Address` または `[PersonName]` のいずれかの複合項目である場合、DescribeLayoutComponent は、FieldLayoutComponent によって拡張されます。

名前	型	説明
`displayLines`	int	編集ビューの項目に表示される垂直な線の数。`textarea` および複数選択リストに適用されます。
`tabOrder`	int	行のアイテムのタブの順序を示します。
`type`	LayoutComponentType	この LayoutComponent の LayoutComponentType。
`value`	string	この LayoutComponent の値。LayoutComponentType の値が `Field` の場合、項目の名前。LayoutComponentType 値が `Canvas` の場合のキャンバスアプリケーションの API 名。

DescribeLayoutFeedFilter

フィードの絞り込みに使用可能な個々のフィード条件オプションを表します。

名前	型	説明
`label`	string	検索条件の表示ラベル。
`name`	string	検索条件の API 名。
`type`	FeedLayoutFilterType Enum	以下の標準のフィード条件種別。 • AllUpdates

名前	型	説明
		• FeedItemType

DescribeLayoutFeedView

フィードベースのページレイアウトのフィードビューのレイアウトを表します。

名前	型	説明
`feedFilters`	DescribeLayoutFeedFilter[]	フィードと一緒に表示されるフィード条件オプションをリストします。

DescribeLayoutItem

DescribeLayoutRow の個別のアイテムを表します。DescribeLayoutItem はコンポーネントセット (DescribeLayoutComponent) で構成され、それぞれは項目または境界です。レイアウトのほとんどの項目で、1 つのレイアウトアイテム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、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 DescribeLayoutItem に分けられます。

名前	型	説明
`editable`	boolean	この DescribeLayoutItem が編集可能であるか (`true`)、否か (`false`) を示します。この項目は、API バージョン 30.0 以前で使用できます。API バージョン 31.0 では、`editableForNew` および `editableForUpdate` 項目に置き換わりました。
`editableForNew`	boolean	レコードの作成時に、新規 DescribeLayoutItem を編集できるか (`true`)、否か (`false`) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
`editableForUpdate`	boolean	レコードの編集時に、既存の DescribeLayoutItem を編集できるか (`true`)、否か (`false`) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
`label`	string	この DescribeLayoutItem の表示ラベルテキスト。
`layoutComponents`	DescribeLayoutComponent[]	この DescribeLayoutItem の DescribeLayoutComponent。
`placeholder`	boolean	この DescribeLayoutItem がプレースホルダーか (`true`)、否か (`false`) を示します。`true` の場合、この DescribeLayoutItem は空白となります。

名前	型	説明
`required`	boolean	この DescribeLayoutItem が必須項目か (`true`)、否か (`false`) を示します。この機能は、目立つ色 (赤など) で必須項目を表示する場合などに便利です。

DescribeLayoutRow

DescribeLayoutSection の行を表します。DescribeLayoutRow は DescribeLayoutItem で構成されています。それぞれの DescribeLayoutRow について、DescribeLayoutItem は特定の項目または「空白の」 DescribeLayoutItem (DescribeLayoutComponent オブジェクトを含まない DescribeLayoutItem) を参照します。空の DescribeLayoutItem は、指定された DescribeLayoutRow がまばらである場合返されます (たとえば、左の列より右の列の方が項目が多い場合)。レイアウトに空白がある場合、空の DescribeLayoutItem がプレースホルダーとして返されます。

名前	型	説明
`layoutItems`	DescribeLayoutItem[]	特定の項目または「空」の LayoutItem (DescribeLayoutComponent オブジェクトを含まない LayoutItem) を参照します。
`numItems`	int	`layoutItems` の数。この情報は冗長ですが、一般的な SOAP ツールキットのバグによる逐次化の問題を避けるために必要です。

DescribeLayoutSection

DescribeLayout 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 (DescribeLayoutRow の配列)。

名前	型	説明
`columns`	int	この DescribeLayoutSection の列数。
`heading`	string	この DescribeLayoutSection のヘッダーのテキスト (表示ラベル)。
`layoutRows`	DescribeLayoutRow[]	1 つ以上の DescribeLayoutRow オブジェクトの配列。
`parentLayoutId`	ID	この DescribeLayoutSection が存在するレイアウトの ID。 この項目は、API バージョン 35.0 以降で使用できます。
`rows`	int	この DescribeLayoutSection の行数。
`tabOrder`	string	編集ビューのセクションの項目のタブ順序を示します。有効な値は、次のとおりです。 • LeftToRight • TopToBottom

名前	型	説明
		この項目は、API バージョン 31.0 以降で使用できます。
`useCollapsibleSection`	boolean	この DescribeLayoutSection が折りたたみ可能な (「twistie」な) セクションかどうかを示します (可能な場合は `true`、不可能な場合は `false`)。
`useHeading`	boolean	`heading` を表示するか (`true`)、否か (`false`) を示します。

DescribeQuickActionListResult

ページレイアウトに割り当てられたアクションのリストを表します。API バージョン 28.0 以降で利用できます。

名前	型	説明
`quickActionListItems`	DescribeQuickActionListResult[]	0 個以上の QuickActionListItemResult オブジェクトの配列。

DescribeQuickActionListItemResult

ページレイアウトのアクションリストに割り当てられた QuickAction を表します。API バージョン 28.0 以降で利用できます。

名前	型	説明
`colors`	DescribeColor[]	色情報の配列。各色はテーマに関連付けられています。この項目は、API バージョン 29.0 以降で使用できます。
`iconUrl`	string	アクションに関連付けられたアイコンの URL。このアイコン URL は、Spring '10 で導入された現在の Salesforce テーマに使用される 32x32 アイコンに対応します。
`icons`	DescribeIcon[]	このアクションのアイコンの配列。各アイコンはテーマに関連付けられています。この項目は、API バージョン 29.0 以降で使用できます。
`label`	string	アクションの表示ラベル。
`miniIconUrl`	string	アクションに関連付けられたミニアイコンの URL。このアイコン URL は、Spring '10 で導入された現在の Salesforce テーマに使用される 16x16 アイコンに対応します。
`quickActionName`	string	アクションの API 名。
`targetSobjectType`	string	アクションの対象オブジェクトの API 名。

名前	型	説明
`type`	string	アクションの QuickActionType。有効な値は、次のとおりです。 • Create • VisualforcePage

CustomLinkComponent

LayoutComponentType の値が `CustomLink` の場合、この型にはページレイアウト上の 1 つのカスタムリンクに関する情報が含まれます。

名前	型	説明
`customLink`	DescribeLayoutButton	種別が `CustomLink` の 1 つの LayoutComponent オブジェクト。

FieldLayoutComponent

describeLayoutComponent で返される情報を拡張します。LayoutComponentType の値が `Field` で、記述される項目が `Address` または `Person Name` である場合、FieldLayoutComponent には項目のコンポーネントに関する情報が含まれます。LayoutComponentType の値が `Field` で、記述される項目が `Address` または `Person Name` などの複合項目である場合、FieldLayoutComponent には項目のコンポーネントに関する情報が含まれます。

API バージョン 31.0 以降で利用できます。

名前	型	説明
`components`	describeLayoutComponent[]	種別が `Field` の 0 個以上の LayoutComponent オブジェクトの配列。
`fieldType`	FieldType	項目のデータ型。

FieldLayoutComponent の使用方法を示すサンプルコード

```
        DescribeLayoutComponent layoutComponent = layoutComponents[n];
        // Look for a component representing the BillingAddress field
        if (layoutComponent.getType() == LayoutComponentType.Field.toString() &&
        layoutComponent.getValue().equals("BillingAddress") {
        // Cast this component as a FieldLayoutComponent
        DescribeLayoutComponent.FieldLayoutComponent addressFieldComponent =
(FieldLayoutComponent)layoutComponent;
        // At this point you can access addressFieldComponent
FieldLayoutComponent-specific methods such as getComponents() or
getFieldType()
        }
```

LayoutComponentType

DescribeLayoutComponent のタイプを表します。次の値の 1 つを含みます。

- `AnalyticsCloud` — ページレイアウトの CRM Analytics ダッシュボード。API バージョン 34.0 以降で利用できます。
- `Canvas` — ページレイアウトのキャンバスコンポーネント。このレイアウトコンポーネントの種類は、API バージョン 31.0 以降で使用できます。
- `CustomLink` — ページレイアウトのカスタムリンク。
- `EmptySpace` — ページレイアウトの空白スペース。
- `ExpandedLookup` — ページレイアウトのモバイルカードセクションにある拡張ルックアップコンポーネント。
- `Field` — 項目名。describeSObjectResult の RecordTypeInfo 項目への対応付け。
- `ReportChart` — ページレイアウトのレポートグラフ。
- `SControl` — 今後の使用のための予約。
- `Separator` — セミコロン (:) やスラッシュ (/) などの区切り文字。
- `VisualforcePage` — ページレイアウトの Visualforce コンポーネント。

PicklistForRecordType

RecordTypeMapping の単一のレコードタイプ選択リストを表します。`picklistName` は、describeSObjectResult の `fields` 配列の各項目の `name` 属性に一致します。`picklistValues` は、`recordType` の利用可能な値セットです。

名前	型	説明
`picklistName`	string	選択リストの名前。
`picklistValues`	PicklistEntry[]	RecordTypeMapping の `recordTypeIds` に関連付けられた選択リストの値セット。 **注意:** `picklistValues` を取得した場合、PicklistEntry の値は null 値となります。PicklistEntry 値が必要な場合、DescribeSObjectResult に関連付けられた `Field` オブジェクトから取得した PicklistEntry から取得します。

RecordTypeMapping

DescribeLayoutResult オブジェクトの `recordTypeMappings` 項目の単一のレコードタイプの対応付けを表します。このオブジェクトは、有効な `recordTypeIds` の `layoutId` に対する対応付けです。詳細ビューを表示するために、クライアントアプリケーションはこの対応付けを使用して、レコードのレコードタイプに関連付けられたレイアウトを確認します。編集ビューを表示するために、クライアントアプリケーションはこの対応付けを使用して使用するレイアウトを確認します (また、ユーザーによる複数のレコードタイプの選択を許可します)。また、利用可能な選択リストの値セットも確認します。

名前	型	説明
available	boolean	このレコードタイプが利用可能かどうかを示します (利用可能な場合は true、そうでない場合は false)。利用可能かどうかという情報は、レコードの作成時に利用可能なレコードタイプの一覧をユーザーに表示するのに使用します。
defaultRecordTypeMapping	boolean	デフォルトのレコードタイプの対応付けかどうかを示します (デフォルトの場合は true、そうでない場合は false)。
layoutId	ID	このレコードタイプに関連付けられたレイアウトの ID。
name	string	レコードタイプの名前。
picklistsForRecordType	PicklistForRecordType[]	recordTypeIds に対応付けられたレコードタイプ選択リスト。
recordTypeId	ID	レコードタイプの ID。

メモ: 以前のバージョンでこの結果に含まれていたいくつかの項目は RecordTypeInfo (ページ 333) に移動されました。

RelatedContent

DescribeLayout のモバイルカードセクションを表します。API バージョン 29.0 以降で使用できます。

名前	型	説明
relatedContentItems	DescribeRelatedContentItem[]	ページレイアウトのモバイルカードセクションにある項目の配列。

DescribeRelatedContentItem

DescribeRelatedContentItem リスト内の個々の項目を表します。API バージョン 29.0 以降で使用できます。

名前	型	説明
describeLayoutItem	DescribeLayoutItem	モバイルカードセクションにある個々のレイアウト項目。モバイルカードセクションに追加するには、DescribeRelatedContentItem でラップする必要があります。

RelatedList

DescribeLayoutResult の単一の関連リストを表します。

名前	型	説明
`buttons`	DescribeLayoutButton[]	この関連リストに関連付けられたボタン。この項目は、API バージョン 32.0 以降で使用できます。
`columns`	RelatedListColumn[]	関連リストに関連付けられた列。 この値を Field と組み合わせ、次のような項目の判別など、数々の便利なタスクを実行できます。 • 名前項目かどうか (詳細へのリンクを表すため) • 並び替え可能かどうか (指定された列で行を並び替える目的でユーザーによる `ORDER BY` 句での使用を許可するため) • 通貨項目かどうか (通貨記号またはコードを含めるため)
`custom`	boolean	`true` の場合、関連リストはカスタムとなります。
`field`	string	関連するオブジェクトとのリレーションを確立する関連オブジェクトの項目名。たとえば、Account の Contact 関連リストにおいて、値は `AccountId` となります。
`label`	string	Salesforce ユーザーインターフェースに表示された関連リストの表示ラベル。
`limitRows`	int	表示する行数。
`name`	string	DescribeLayout の引数として提供されている `sObjectType` の DescribeSObjectResult の中の ChildRelationship の名前。
`sobject`	string	Name of the `sObjectType` that is the row type for rows within this related list.
`sort`	RelatedListSort[]	null 値でない場合、関連オブジェクトの並び替えに使用される列。

RelatedListColumn

DescribeLayoutResult が返した関連リストの単一項目を表します。

名前	型	説明
field	string	項目の API 名。この値は常に ***object_type.field_name*** の形式で表されます。たとえば、name が Contact.Account.Owner.Alias の場合、値は User.Alias となります。
fieldApiName	string	関連リストのメインの sObject に関連付けられた項目の SOQL 項目構文。この値は常に ***object_type.field_name*** の形式で表されます。Name とは異なり、返された SOQL 結果の翻訳の形式で値を返しません。
format	string	date または dateTime 形式で表します。
label	string	項目の表示ラベル。
lookupId	string	メインの関連リスト sObject のルックアップ ID 値を取得するための省略可能な SOQL 項目構文。この値は、ドット表記を使用した SOQL リレーションのクエリを使用する形式を使用します。 たとえば、関連リスト sObjectType が Case であり、列の表示値が Owner.Alias の場合、ルックアップ ID 値は Owner.Id となります。
name	string	関連リストのメインの sObject に関連付けられた項目の SOQL 項目構文。この値は、ドット表記を使用した SOQL リレーションのクエリの形式で表すか、返された SOQL 結果の翻訳または convertCurrency() の形式を使用します。 たとえば、関連リスト sObjectType が Case の場合、値は Owner.Alias または toLabel(Case.Status) となります。

RelatedListSort

関連リストのオブジェクトの並び替え設定を表します。

名前	型	説明
column	string	関連オブジェクトの並び替えに使用する項目名。
ascending	boolean	true の場合、並び替え順序は昇順です。false の場合、並び替え順序は降順です。

ほとんどの場合、配列には RelatedListSort は 1 つしか存在しませんが、標準の関連リストの中には RelatedListSort が複数存在する特殊なものもあります。配列が複数の RelatedListSort を持つ場合は、対応する SOQL クエリに含める方法によって並び替えられます。以下に例を示します。

```
ORDER BY relatedListSort[0].getColumn() DIRECTION, relatedListSort[1].getColumn() DIRECTION
```

DescribeLayoutSaveOption

レイアウトの保存オプションを表します。保存オプションでは、特定のレイアウトを使用してオブジェクトが作成または変更されたときに実行する動作を定義します。たとえば、ケースとリードの場合、「UseDefaultAssignmentRule」保存オプションが表示されます。これにより、ケースまたはリードが作成または編集されたときに割り当てルールを適用するかどうかを制御します。

名前	型	説明
`defaultValue`	boolean	保存オプションのデフォルト値。保存オプションのデフォルトを Salesforce ユーザーインターフェースで有効にするかどうかを制御します。 たとえば、「UseDefaultAssignmentRule」保存オプションの場合、`defaultValue` を `true` にすると、デフォルトでは、システムは取引先、ケース、またはリードが作成または編集されたときにデフォルトの割り当てルールをトリガーします。`false` の場合は、ユーザーが Salesforce ユーザーインターフェースで保存オプションを有効にしない限り、取引先、ケース、またはリードが作成または編集されたときにデフォルトの割り当てルールは適用されません。
`isDisplayed`	boolean	`true` の場合、保存オプションがレイアウトに表示されます。`false` の場合、保存オプションはレイアウトに表示されません。
`label`	string	Salesforce ユーザーインターフェースに表示される保存オプションの表示ラベル。
`name`	string	保存オプションの API 参照名。
`restHeaderName`	string	保存オプションに対応する REST API ヘッダー。
`soapHeaderName`	string	保存オプションに対応する SOAP API ヘッダー。

WebLinkPosition

DescribeLayoutButton をクリックすると開く新規ウィンドウのウィンドウ位置を表します。カスタムボタンにのみ適用されます。API バージョン 31.0 以降で利用できます。次のいずれかの値が含まれます。

- `fullScreen` — 全画面で新しいウィンドウが開きます。このオプションを選択すると、新しいウィンドウに設定された幅または高さのパラメーターは無視されます。
- `none` — ウィンドウの位置が設定されていません。
- `topLeft` — 画面の左上で新しいウィンドウが開きます。

WebLinkType

カスタムボタンが配信するコンテンツを表します。次のいずれかの値が含まれます。

- `javascript`
- `page` — Visualforce ページ
- `sControl`
- `url`

WebLinkWindowType

DescribeLayoutButton の動作を表します。カスタムボタンにのみ適用されます。API バージョン 31.0 以降で利用できます。次のいずれかの値が含まれます。

- `newWindow` — カスタムボタンのコンテンツが新しいブラウザーウィンドウに開きます。
- `noSidebar` — カスタムボタンのコンテンツがサイドバーのない既存のブラウザーウィンドウに表示されます。
- `onClickJavaScript` — DescribeLayoutButton の `contentSource` 項目値が `javascript` の場合にのみ有効です。ボタンまたはリンクをクリックすると、JavaScript が実行されます。
- `replace` — カスタムボタンのコンテンツがサイドバーやヘッダーのない既存のブラウザーウィンドウに表示されます。
- `sidebar` — カスタムボタンのコンテンツがサイドバーのある既存のブラウザーウィンドウに表示されます。

describePrimaryCompactLayouts()

指定されたオブジェクト種別ごとに主コンパクトレイアウトに関するメタデータを取得します。

最大で 100 個の情報が返されます。

構文

```
DescribeCompactLayout[] primaryCompactLayouts =
connection.describePrimaryCompactLayouts(string[] sObjectType)
```

使用方法

このコールを使用し、指定されたオブジェクト種別の主コンパクトレイアウトに関する情報を取得します。このコールは、指定された主コンパクトレイアウトに関するメタデータを返します。コンパクトレイアウトについての詳細は、Salesforce ヘルプを参照してください。

サンプルコード — Java

```
public void testDescribePrimaryCompactLayoutsSample() {
  try {
    String[] objectsToDescribe = new String[] {"Account","Lead"};
      DescribeCompactLayout[] primaryCompactLayouts =
connection.describePrimaryCompactLayouts(objectsToDescribe);

      for (int i = 0; i < primaryCompactLayouts.length; i++) {
        DescribeCompactLayout cLayout = primaryCompactLayouts[i];
        System.out.println(" There is a compact layout with name: " + cLayout.getName());


        // Write the objectType
        System.out.println(" This compact layout is the primary compact layout for: " +
cLayout.getObjectType());

        DescribeLayoutItem[] fieldItems = cLayout.getFieldItems();
        System.out.println(" There are " + fieldItems.length + " fields in this compact
layout.");

        // Write field items
        for (int j = 0; j < fieldItems.length; j++) {
          System.out.println(j + " This compact layout has a field with name: " +
fieldItems[j].getLabel());
        }

        DescribeLayoutItem[] imageItems = cLayout.getImageItems();
        System.out.println(" There are " + imageItems.length + " image fields in this
compact layout.");

        // Write the image items
        for (int j = 0; j < imageItems.length; j++) {
          System.out.println(j + " This compact layout has an image field with name: " +
imageItems[j].getLabel());
        }

        DescribeLayoutButton[] actions = cLayout.getActions();
        System.out.println(" There are " + actions.length + " buttons in this compact
layout.");

        // Write the action buttons
        for (int j = 0; j < actions.length; j++) {
          System.out.println(j + " This compact layout has a button with name: " +
actions[j].getLabel());
        }
```

```
      }

    } catch (ConnectionException ce) {
      ce.printStackTrace();
  }
}
```

引数

名前	型	説明
sObjectTypes	string[]	1 つ以上のオブジェクトの配列。指定する値は、組織で有効なオブジェクトである必要があります。

応答

DescribeCompactLayout

障害

InvalidSObjectFault

UnexpectedErrorFault

describeQuickActions()

指定されたアクションの詳細を取得します。

構文

```
DescribeQuickActionResult[] = connection.describeQuickActions(string[] quickActionNames);
```

使用方法

describeQuickActions() コールは、指定されたアクションの詳細を取得するために使用します。API バージョン 28.0 では、describeQuickActions() コールは *ParentEntity.ActionName* の形式でアクション名を取得します。API バージョン 29.0 以降では、*ContextEntity.ActionName* 形式でアクション名を取得します。DescribeQuickActionResult の配列を返します。たとえば、最初に describeAvailableQuickActions() をコールして指定されたコンテキストで使用できるアクションのリストを取得し、次に describeQuickActions() を使用して特定のアクションに関する詳細を取得できます。

メモ: API バージョン 46.0 以降では、describeQuickActions() リクエストボディで、グローバルクイックアクションの apiName にプレフィックス Global. を追加できます。また、このリクエストボディでは、プレフィックスのないグローバルクイックアクションの API 名も使用できます。

メモ: Apex クラスが管理パッケージの一部としてインストールされている場合、`describeQuickActions()` コールは、組織で作成されたアクションの結果を返しません。

サンプル — Java

このサンプルでは、Account オブジェクトに対する作成アクションについてパブリッシャーアクションの詳細を取得して表示します。

```
public void example() throws Exception {
    DescribeQuickActionResult[] result =
        conn.describeQuickActions(new String[]
            { "Account.QuickCreateContact", "Account.QuickCreateTask" });
    for(DescribeQuickActionResult r : result) {
        assert r != null;
    DescribeQuickActionDefaultValue [] describeQuickActionDefaultValues =
r.getDefaultValues();
    for(DescribeQuickActionDefaultValue defaultValue : describeQuickActionDefaultValues)
{
        System.out.println("Target Object Field:  " + defaultValue.getField() );
        System.out.println("Target Object Field's default Value:  " +
defaultValue.getDefaultValue );
}
        System.out.println("Action name: " + r.getName());
        System.out.println("Action label: " + r.getLabel());
        System.out.println("ParentOrContext object: " + r.getSourceSobjectType());
        System.out.println("Target object: " + r.getTargetSobjectType());
        System.out.println("Target object record type: " + r.getTargetRecordTypeId());
        System.out.println("Relationship field: " + r.getTargetParentField());
        System.out.println("Quick action type: " + r.getType());
        System.out.println("VF page name for custom actions: " +
        r.getVisualforcePageName());
        System.out.println("Icon name: " + r.getIconName());
        System.out.println("Icon URL: " + r.getIconUrl());
        System.out.println("Mini icon URL: " + r.getMiniIconUrl());
        assert r.getLayout() != null;
        System.out.println("Height of VF page for custom actions: " + r.getHeight());
        System.out.println("Width of VF page for custom actions: " + r.getWidth());
    }
}
```

引数

名前	型	説明
`quickActions`	string[]	取得するクイックアクションの配列。

応答

DescribeQuickActionResult

DescribeQuickActionResult

DescribeQuickActionResult オブジェクトの配列を返します。

`describeQuickActions()` コールは、DescribeQuickActionResult オブジェクトの配列を返します。各 DescribeQuickActionResult オブジェクトは、指定されたオブジェクトへのクイックアクションを表します。

名前	型	説明
`actionEnumOrId`	string	アクションの一意の ID。アクションに ID がない場合、API 名が使用されます。 この項目は、API バージョン 35.0 以降で使用できます。
`canvasApplicationName`	string	キャンバスアプリケーションの名前(使用されている場合)。
`colors`	DescribeColor[]	色情報の配列。各色はテーマに関連付けられています。 この項目は、API バージョン 29.0 以降で使用できます。
`defaultValues`	DescribeQuickActionDefaultValue[]	アクションのデフォルト値。
`height`	int	アクションペインの高さ (ピクセル単位)。
`iconName`	string	アクションに使用するアイコンの名前。カスタムアイコンが使用されていない場合、この値は設定されません。
`iconUrl`	string	アクションに使用するアイコンの URL。このアイコン URL は、Spring '10 で導入された現在の Salesforce テーマに使用される 32x32 アイコン、またはカスタムアイコン (存在する場合) に対応します。
`icons`	DescribeIcon[]	アイコンの配列。各アイコンはテーマに関連付けられています。 クイックアクションに関連付けられたカスタムアイコンがなく、クイックアクションによって特定のオブジェクトが作成される場合、アイコンは作成されるオブジェクトに使用されるアイコンに対応します。たとえば、クイックアクションによって取引先が作成される場合、アイコン配列には取引先に使用されるアイコンが含まれます。

名前	型	説明
		カスタムアイコンがクイックアクションに関連付けられていた場合、配列にはそのカスタムアイコンが含まれます。 この項目は、API バージョン 29.0 以降で使用できます。API バージョン 32.0 以降では、それより前の API バージョンとは異なるアイコンを返します。
`label`	string	アクションの表示ラベル。
`layout`	DescribeLayoutSection	アクションが存在するレイアウトのセクション。
`lightningComponentBundleId`	ID	`type` が `LightningComponent` の場合、アクションでコールされる Lightning コンポーネントバンドルの ID。 この項目は、API バージョン 38.0 以降で使用できます。
`lightningComponentBundleName`	string	`type` が `LightningComponent` の場合、アクションでコールされる Lightning コンポーネントバンドルの名前。 この項目は、API バージョン 38.0 以降で使用できます。
`miniIconUrl`	string	アイコンの URL。このアイコン URL は、Spring '10 で導入された現在の Salesforce テーマに使用される 16×16 アイコン、またはカスタムアイコン (存在する場合) に対応します。
`name`	string	アクションの名前。
`contextSobjectType`	string	アクションに使用するオブジェクト。バージョン 29.0 以前の名前付き `sourceSobjectType`。
`showQuickActionVfHeader`	boolean	Visualforce クイックアクションヘッダーとフッターを表示するかどうか。`false` に設定すると、クイックアクションのタイトルが含まれるヘッダーと、[保存] および [キャンセル] ボタンが含まれるフッターの両方が非表示になります。
`targetParentField`	string	アクションの親オブジェクト種別。対象オブジェクトを親オブジェクトにリンクします。たとえば、対象オブジェクトが取引先責任者であり、親オブジェクトが取引先である場合、取引先を使用します。

名前	型	説明
`targetRecordTypeId`	ID	対象レコードのレコードタイプ。
`targetSobjectType`	string	アクションの対象オブジェクト種別。
`type`	string	アクションの種別。有効な値は、次のとおりです。 • `Canvas` • `Create` • `Flow` (この値は API バージョン 41.0 以降でベータとして使用できます) • `LightningComponent` (この値は API バージョン 38.0 以降で使用できます)。 • `LogACall` • `Post` • `SendEmail` (この値は API バージョン 31.0 以降で使用できます)。 • `SocialPost` • `Update` • `VisualforcePage`
`visualforcePageName`	string	`type` が Visualforce の場合、アクションに関連付けられたページのページ名。
`visualforcePageUrl`	string	`type` が Visualforce の場合、アクションに関連付けられたページの URL。
`width`	int	カスタムアクションを作成する場合、この値がアクションペインの幅 (ピクセル単位) になります。

DescribeQuickActionDefaultValue

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。

名前	型	説明
`defaultValue`	string	自動的に取り込まれたデフォルトアクションの値。
`field`	string	アクションの項目名。

DescribeLayoutSection

DescribeLayout 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 (DescribeLayoutRow の配列)。

名前	型	説明
columns	int	この DescribeLayoutSection の列数。
heading	string	この DescribeLayoutSection のヘッダーのテキスト (表示ラベル)。
layoutRows	DescribeLayoutRow[]	1 つ以上の DescribeLayoutRow オブジェクトの配列。
parentLayoutId	ID	この DescribeLayoutSection が存在するレイアウトの ID。 この項目は、API バージョン 35.0 以降で使用できます。
rows	int	この DescribeLayoutSection の行数。
tabOrder	string	編集ビューのセクションの項目のタブ順序を示します。有効な値は、次のとおりです。 • LeftToRight • TopToBottom この項目は、API バージョン 31.0 以降で使用できます。
useCollapsibleSection	boolean	この DescribeLayoutSection が折りたたみ可能な (「twistie」な) セクションかどうかを示します (可能な場合は true、不可能な場合は false)。
useHeading	boolean	heading を表示するか (true)、否か (false) を示します。

DescribeLayoutRow

DescribeLayoutSection の行を表します。DescribeLayoutRow は DescribeLayoutItem で構成されています。それぞれの DescribeLayoutRow について、DescribeLayoutItem は特定の項目または「空白の」 DescribeLayoutItem (DescribeLayoutComponent オブジェクトを含まない DescribeLayoutItem) を参照します。空の DescribeLayoutItem は、指定された DescribeLayoutRow がまばらである場合返されます (たとえば、左の列より右の列の方が項目が多い場合)。レイアウトに空白がある場合、空の DescribeLayoutItem がプレースホルダーとして返されます。

名前	型	説明
layoutItems	DescribeLayoutItem[]	特定の項目または「空」の LayoutItem (DescribeLayoutComponent オブジェクトを含まない LayoutItem) を参照します。
numItems	int	layoutItems の数。この情報は冗長ですが、一般的な SOAP ツールキットのバグによる逐次化の問題を避けるために必要です。

DescribeLayoutItem

DescribeLayoutRow の個別のアイテムを表します。DescribeLayoutItem はコンポーネントセット (DescribeLayoutComponent) で構成され、それぞれは項目または境界です。レイアウトのほとんどの項目で、1 つのレイアウトアイテム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、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 DescribeLayoutItem に分けられます。

名前	型	説明
`editable`	boolean	この DescribeLayoutItem が編集可能であるか (`true`)、否か (`false`) を示します。この項目は、API バージョン 30.0 以前で使用できます。API バージョン 31.0 では、`editableForNew` および `editableForUpdate` 項目に置き換わりました。
`editableForNew`	boolean	レコードの新規作成時に、新規 DescribeLayoutItem を編集できるか (`true`)、否か (`false`) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
`editableForUpdate`	boolean	レコードの編集時に、既存の DescribeLayoutItem を編集できるか (`true`)、否か (`false`) を示します。 この項目は、API バージョン 31.0 以降で使用できます。
`label`	string	この DescribeLayoutItem の表示ラベルテキスト。
`layoutComponents`	DescribeLayoutComponent[]	この DescribeLayoutItem の DescribeLayoutComponent。
`placeholder`	boolean	この DescribeLayoutItem がプレースホルダーか (`true`)、否か (`false`) を示します。`true` の場合、この DescribeLayoutItem は空白となります。
`required`	boolean	この DescribeLayoutItem が必須項目か (`true`)、否か (`false`) を示します。この機能は、目立つ色 (赤など) で必須項目を表示する場合などに便利です。

DescribeLayoutComponent

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。表示するための項目の参照において、クライアントアプリケーションは次の表記法を使用し describeSObjects() コールの項目を参照します。

`LayoutComponent.`*`fieldName`*。

API バージョン 31.0 以降では、DescribeLayoutComponent 値が `Field` で、記述される項目が `Address` または `[Person Name]` のいずれかの複合項目である場合、DescribeLayoutComponent は、FieldLayoutComponent によって拡張されます。

名前	型	説明
`displayLines`	int	編集ビューの項目に表示される垂直な線の数。`textarea` および複数選択リストに適用されます。
`tabOrder`	int	行のアイテムのタブの順序を示します。
`type`	LayoutComponentType	この LayoutComponent の LayoutComponentType。
`value`	string	この LayoutComponent の値。LayoutComponentType の値が `Field` の場合、項目の名前。LayoutComponentType 値が `Canvas` の場合のキャンバスアプリケーションの API 名。

LayoutComponentType

DescribeLayoutComponent のタイプを表します。次のいずれかの値が含まれます。

- `AnalyticsCloud` — ページレイアウトの CRM Analytics ダッシュボード。API バージョン 34.0 以降で利用できます。
- `Canvas` — ページレイアウトのキャンバスコンポーネント。このレイアウトコンポーネントの種類は、API バージョン 31.0 以降で使用できます。
- `CustomLink` — ページレイアウトのカスタムリンク。
- `EmptySpace` — ページレイアウトの空白スペース。
- `ExpandedLookup` — ページレイアウトのモバイルカードセクションにある拡張ルックアップコンポーネント。
- `Field` — 項目名。describeSObjectResult の `name` 項目への対応付け。
- `ReportChart` — ページレイアウトのレポートグラフ。
- `SControl` — 今後の使用のための予約。
- `Separator` — セミコロン (:) やスラッシュ (/) などの区切り文字。
- `VisualforcePage` — ページレイアウトの Visualforce コンポーネント。

describeSearchScopeOrder()

ユーザーのデフォルトのグローバル検索範囲内にあるオブジェクトの順序付きリストを取得します。

構文

```
DescribeSearchScopeOrderResult[] describeSearchScopeOrderResults =
connection.describeSearchScopeOrder();
```

使用方法

ログインユーザーのデフォルトのグローバル検索範囲内にあるオブジェクトの順序付きリストを取得するには、`describeSearchScopeOrder()` を使用します。グローバル検索は、操作するオブジェクトとそれらを操作する頻度を追跡し、それに基づいて検索結果を編成します。最もよく使用されるオブジェクトは、リストの最上部に表示されます。返されるリストには、ユーザーの検索結果ページの固定表示オブジェクトを含め、ユーザーのデフォルトの検索範囲でのオブジェクト順が反映されます。このコールは、最適化されたグローバル検索範囲を使用してカスタム検索結果ページを実装する場合に役立ちます。

メモ: グローバル検索を有効にするには、Chatter を有効にする必要があります。

サンプルコード — Java

このサンプルでは、ユーザーのグローバル検索範囲を取得して、その範囲内の各オブジェクトの名前を反復的に表示する方法を示します。

```
public void describeSearchScopeOrderSample() {
  try {
    //Get the order of objects in search smart scope for the logged-in user
    DescribeSearchScopeOrderResult[] describeSearchScopeOrderResults =
        connection.describeSearchScopeOrder();
    //Iterate through the results and display the name of each object
    for (int i = 0; i < describeSearchScopeOrderResults.length; i++) {
      System.out.println(describeSearchScopeOrderResults[i].getName());
    }
  }
  catch(ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

引数

なし。

応答

DescribeSearchScopeOrderResult オブジェクトの配列

障害

`UnexpectedErrorFault`

関連トピック:

API コールの基礎

DescribeSearchScopeOrderResult

DescribeSearchScopeOrderResult オブジェクトの配列を返します。

describeSearchScopeOrder() コールは、DescribeSearchScopeOrderResult オブジェクトの配列を返します。各 DescribeSearchScopeOrderResult オブジェクトはユーザーのグローバル検索範囲内の 1 つのオブジェクトを表します。リストには、固定表示オブジェクトを含む、ユーザーの範囲内のオブジェクトの順序が反映されます。DescribeSearchScopeOrderResult 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
keyPrefix	string	オブジェクト ID の 3 文字のプレフィックスコード。オブジェクト ID はオブジェクト型を示す 3 文字のコードが先頭に付けられます。たとえば、Account オブジェクトのプレフィックスは 001、Opportunity オブジェクトのプレフィックスは 006 です。主要なプレフィックスを複数のオブジェクトで共有する場合があるため、必ずしもオブジェクトを一意に識別するわけではありません。
name	string	オブジェクトの名前。英語のみを使用します。

describeSearchLayouts()

1 つ以上のオブジェクトの検索結果レイアウト設定を取得します。

構文

```
DescribeSearchLayoutResult[] = binding.describeSearchLayouts(string sObjectType[]);
```

使用方法

describeSearchLayouts() は、1つ以上のオブジェクトの検索結果レイアウト情報を取得するために使用します。このコールは、Salesforceの場合と同じレイアウト設定でカスタム検索結果ページを作成する場合に便利です。

サンプル

このサンプルは、オブジェクトのリストについて検索結果レイアウト情報を取得する方法を示します。

```
public void describeSearchLayoutSample(String[] sObjectTypes) {
        try {
            // Get the search layout of Account and Group
            DescribeSearchLayoutResult[] searchLayoutResults =
connection.describeSearchLayouts(sObjectTypes);
            // Iterate through the results and display the label of each column
            for (int i = 0; i < sObjectTypes.length; i += 1) {
```

```
                String sObjectType = sObjectTypes[i];
                DescribeSearchLayoutResult result = searchLayoutResults[i];
                System.out.println("Top label for search results for " + sObjectType + "
is " + result.getLabel() + " and should display " + result.getLimitRows() + " rows");
                System.out.println("Column labels for search results for " + sObjectType
+ " are: ");
                for (DescribeColumn column : result.getSearchColumns()) {
                    System.out.println(column.getLabel());
                }
            }
        } catch (ConnectionException ce) {
            ce.printStackTrace();
        }
    }
```

引数

名前	型	説明
sObjectType	string[]	検索結果レイアウト設定を取得するオブジェクトのリスト。たとえば、オブジェクトが個人取引先である場合は Account と指定し、個人取引先責任者である場合、Contact と指定します。指定する値は、組織内で有効なオブジェクトである必要があります。すべての標準オブジェクトの一覧は、「標準オブジェクト」を参照してください。

応答

DescribeSearchLayoutResult

障害

InvalidSObjectFault

UnexpectedErrorFault

DescribeSearchLayoutResult

describeSearchLayouts() コールは、DescribeSearchLayoutResult オブジェクトの配列を返します。

各 DescribeSearchLayoutResult オブジェクトは、クエリ対象の各オブジェクトの検索レイアウト設定を表します。DescribeSearchLayoutResult 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
label	string	検索結果ページに使用するブラウザータイトル。

名前	型	説明
limitRows	int	検索結果の最初のページに表示する最大行数。この数値は、システム管理者が変更できます。
searchColumns	DescribeColumn (ページ 310)[]	このオブジェクトの検索結果に関連付けられた列。

DescribeColumn

describeSearchLayouts() (ページ 309) コールで返された各 DescribeSearchLayoutResult オブジェクトの検索レイアウト設定内の列を表します。

名前	型	説明
field	string	属するオブジェクトに対する項目参照。たとえば、「Lead.Phone」などです。
format	string	項目のデータ形式。たとえば、「date」などです。この値は null にできます。
label	string	ユーザーインターフェースでのこの項目の表示テキスト。たとえば、「会社電話」または「電話」などです。
name	string	項目名。SOQL クエリまたはコードで使用します。たとえば、「Name」などです。

describeSObject()

指定されたオブジェクトのメタデータ (項目リストとオブジェクトプロパティ) を表します。

メモ: describeSObjects() は、describeSObject() よりも優先されます。describeSObject() ではなく describeSObjects() を使用してください。

構文

```
DescribeSObjectResult = connection.describeSObject(string sObjectType);
```

使用方法

指定されたオブジェクトのメタデータを取得するには、describeSObject() を使用します。最初に describeGlobal() 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、describeSObject() 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。

組織のデータのメタデータを取得するには、クライアントアプリケーションは条件を満たすアクセス権限でログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

サンプルコード — Java

このサンプルでは、`describeSObject()` をコールし、取引先 sObject に describe を実行します。sObject 名、表示ラベル、項目などの、sObject の describe result の一部のプロパティを取得します。次に、項目を反復処理し、項目のプロパティを取得します。選択リスト項目では、選択リスト値を取得して、参照項目では参照されているオブジェクト名を取得します。このサンプルでは、取得した sObject および項目のプロパティをコンソールに書き込みます。

```
public void describeSObjectSample() {
    try {
        // Make the describe call
        DescribeSObjectResult describeSObjectResult =
            connection.describeSObject("Account");

        // Get sObject metadata
        if (describeSObjectResult != null) {
          System.out.println("sObject name: " +
                  describeSObjectResult.getName());
        if (describeSObjectResult.isCreateable())
            System.out.println("Createable");

        // Get the fields
        Field[] fields = describeSObjectResult.getFields();
        System.out.println("Has " + fields.length + " fields");

        // Iterate through each field and gets its properties
        for (int i = 0; i < fields.length; i++) {
          Field field = fields[i];
          System.out.println("Field name: " + field.getName());
          System.out.println("Field label: " + field.getLabel());

          // If this is a picklist field, show the picklist values
          if (field.getType().equals(FieldType.picklist)) {
              PicklistEntry[] picklistValues =
                  field.getPicklistValues();
              if (picklistValues != null) {
                System.out.println("Picklist values: ");
                for (int j = 0; j < picklistValues.length; j++) {
                  if (picklistValues[j].getLabel() != null) {
                    System.out.println("\tItem: " +
                        picklistValues[j].getLabel()
                    );
                  }
                }
              }
          }

          // If a reference field, show what it references
          if (field.getType().equals(FieldType.reference)) {
              System.out.println("Field references the " +
                      "following objects:");
              String[] referenceTos = field.getReferenceTo();
              for (int j = 0; j < referenceTos.length; j++) {
```

```
                    System.out.println("\t" + referenceTos[j]);
                }
            }
        }
    }
 } catch (ConnectionException ce) {
     ce.printStackTrace();
    }
}
```

サンプルコード — C#

このサンプルでは、describeSObject()** をコールし、取引先** sObject **に** describe **を実行します。**sObject **名、表示ラベル、項目などの、**sObject **の** describe result **の一部のプロパティを取得します。次に、項目を反復処理し、項目のプロパティを取得します。選択リスト項目では、選択リスト値を取得して、参照項目では参照されているオブジェクト名を取得します。このサンプルでは、取得した** sObject **および項目のプロパティをコンソールに書き込みます。**

```
public void describeSObjectSample() {
    try {
        // Make the describe call
        DescribeSObjectResult describeSObjectResult =
            binding.describeSObject("Account");

        // Get sObject metadata
        if (describeSObjectResult != null) {
            Console.WriteLine("sObject name: " +
                    describeSObjectResult.name);
        if (describeSObjectResult.createable)
            Console.WriteLine("Createable");

        // Get the fields
        Field[] fields = describeSObjectResult.fields;
        Console.WriteLine("Has " + fields.Length + " fields");

        // Iterate through each field and gets its properties
        for (int i = 0; i < fields.Length; i++) {
            Field field = fields[i];
            Console.WriteLine("Field name: " + field.name);
            Console.WriteLine("Field label: " + field.label);

            // If this is a picklist field, show the picklist values
            if (field.type.Equals(fieldType.picklist)) {
                PicklistEntry[] picklistValues =
                    field.picklistValues;
                if (picklistValues != null) {
                    Console.WriteLine("Picklist values: ");
                    for (int j = 0; j < picklistValues.Length; j++) {
                        if (picklistValues[j].label != null) {
                        Console.WriteLine("\tItem: " +
                            picklistValues[j].label);
                        }
```

```
                    }
                }
            }

            // If a reference field, show what it references
            if (field.type.Equals(fieldType.reference)) {
                Console.WriteLine("Field references the " +
                      "following objects:");
                String[] referenceTos = field.referenceTo;
                for (int j = 0; j < referenceTos.Length; j++) {
                    Console.WriteLine("\t" + referenceTos[j]);
                }
            }
        }
        }
    } catch (SoapException e) {
        Console.WriteLine("An unexpected error has occurred: " +
            e.Message + "\n" + e.StackTrace);
    }
}
```

引数

名前	型	説明
sObjectType	string	**オブジェクト。指定された値は、組織で有効なオブジェクトである必要があります。完全なオブジェクトのセットについては、「標準オブジェクト」を参照してください。**

応答

DescribeSObjectResult

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

- describeSObjects()
- describeGlobal()
- API **コールの基礎**
- Partner WSDL **の使用**
- https://developer.salesforce.com/page/Sample_SOAP_Messages

describeSObjectResult

describeSObject() コールは DescribeSObjectResult オブジェクトを返します。

メモ: `describeSObjects()` は、`describeSObject()` よりも優先されます。`describeSObject()` ではなく `describeSObjects()` を使用してください。

describeSObjects()

`describeSObject()` の配列をベースとするバージョン。指定されたオブジェクトまたはオブジェクト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) を取得します。`describeSObject()` の代わりにこのコールを使用してください。

構文

```
DescribeSObjectResult [] = connection.describeSObjects(string sObjectType[] );
```

使用方法

指定されたオブジェクトまたはオブジェクト配列のメタデータを取得するには `describeSObjects()` を使用します。最初に `describeGlobal()` 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。`describeSObjects()` コールが返すことができるオブジェクトの最大数は 100 個です。

組織のデータのメタデータを取得するには、クライアントアプリケーションは条件を満たすアクセス権限でログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

個人取引先が有効な組織では、法人取引先を示すレコードタイプへのアクセス権がプロファイルに割り当てられない限り、このコールでは Account が作成不可能として表されます。

サンプルコード — Java

このサンプルでは、`describeSObjects()` をコールし、取引先、取引先責任者、およびリードの describe を実行します。返された sObject のリストを反復処理し、各 sObject のプロパティおよび項目を取得して、コンソールにそれらを書き込みます。選択リスト項目では、選択リスト値を書き込みます。参照項目では、参照されているオブジェクト名を書き込みます。

```
public void describeSObjectsSample()
{
  try {
    // Call describeSObjectResults and pass it an array with
    // the names of the objects to describe.
    DescribeSObjectResult[] describeSObjectResults =
                        connection.describeSObjects(
                         new String[] { "account", "contact", "lead" });

    // Iterate through the list of describe sObject results
    for (int i=0;i < describeSObjectResults.length; i++)
```

```
{
    DescribeSObjectResult desObj = describeSObjectResults[i];
    // Get the name of the sObject
    String objectName = desObj.getName();
    System.out.println("sObject name: " + objectName);

    // For each described sObject, get the fields
    Field[] fields = desObj.getFields();

    // Get some other properties
    if (desObj.getActivateable()) System.out.println("\tActivateable");

    // Iterate through the fields to get properties for each field
    for(int j=0;j < fields.length; j++)
    {
        Field field = fields[j];
        System.out.println("\tField: " + field.getName());
        System.out.println("\t\tLabel: " + field.getLabel());
        if (field.isCustom())
            System.out.println("\t\tThis is a custom field.");
        System.out.println("\t\tType: " + field.getType());
        if (field.getLength() > 0)
            System.out.println("\t\tLength: " + field.getLength());
        if (field.getPrecision() > 0)
            System.out.println("\t\tPrecision: " + field.getPrecision());

        // Determine whether this is a picklist field
        if (field.getType() == FieldType.picklist)
        {
            // Determine whether there are picklist values
            PicklistEntry[] picklistValues = field.getPicklistValues();
            if (picklistValues != null && picklistValues[0] != null)
            {
                System.out.println("\t\tPicklist values = ");
                for (int k = 0; k < picklistValues.length; k++)
                {
                   System.out.println("\t\t\tItem: " + picklistValues[k].getLabel());

                }
            }
        }

        // Determine whether this is a reference field
        if (field.getType() == FieldType.reference)
        {
            // Determine whether this field refers to another object
            String[] referenceTos = field.getReferenceTo();
            if (referenceTos != null && referenceTos[0] != null)
            {
                System.out.println("\t\tField references the following objects:");
                for (int k = 0; k < referenceTos.length; k++)
                {
                    System.out.println("\t\t\t" + referenceTos[k]);
                }
```

```
                }
            }
        }
    }
  } catch(ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

サンプルコード — C#

このサンプルでは、`describeSObjects()` をコールし、取引先、取引先責任者、およびリードの describe を実行します。返された sObject のリストを反復処理し、各 sObject のプロパティおよび項目を取得して、コンソールにそれらを書き込みます。選択リスト項目では、選択リスト値を書き込みます。参照項目では、参照されているオブジェクト名を書き込みます。

```
public void describeSObjectsSample()
{
    try
    {
    // Call describeSObjectResults and pass it an array with
    // the names of the objects to describe.
    DescribeSObjectResult[] describeSObjectResults =
                        binding.describeSObjects(
                        new string[] { "account", "contact", "lead" });

    // Iterate through the list of describe sObject results
    foreach (DescribeSObjectResult describeSObjectResult in describeSObjectResults)
    {
        // Get the name of the sObject
        String objectName = describeSObjectResult.name;
        Console.WriteLine("sObject name: " + objectName);

        // For each described sObject, get the fields
        Field[] fields = describeSObjectResult.fields;

        // Get some other properties
        if (describeSObjectResult.activateable) Console.WriteLine("\tActivateable");

        // Iterate through the fields to get properties for each field
        foreach (Field field in fields)
        {
            Console.WriteLine("\tField: " + field.name);
            Console.WriteLine("\t\tLabel: " + field.label);
            if (field.custom)
                Console.WriteLine("\t\tThis is a custom field.");
            Console.WriteLine("\t\tType: " + field.type);
            if (field.length > 0)
                Console.WriteLine("\t\tLength: " + field.length);
            if (field.precision > 0)
                Console.WriteLine("\t\tPrecision: " + field.precision);

            // Determine whether this is a picklist field
```

```
            if (field.type == fieldType.picklist)
            {
                // Determine whether there are picklist values
                PicklistEntry[] picklistValues = field.picklistValues;
                if (picklistValues != null && picklistValues[0] != null)
                {
                    Console.WriteLine("\t\tPicklist values = ");
                    for (int j = 0; j < picklistValues.Length; j++)
                    {
                        Console.WriteLine("\t\t\tItem: " + picklistValues[j].label);
                    }
                }
            }

            // Determine whether this is a reference field
            if (field.type == fieldType.reference)
            {
                // Determine whether this field refers to another object
                string[] referenceTos = field.referenceTo;
                if (referenceTos != null && referenceTos[0] != null)
                {
                    Console.WriteLine("\t\tField references the following objects:");
                    for (int j = 0; j < referenceTos.Length; j++)
                    {
                        Console.WriteLine("\t\t\t" + referenceTos[j]);
                    }
                }
            }
        }
    }
    }
    catch (SoapException e)
    {
        Console.WriteLine("An unexpected error has occurred: " + e.Message
            + "\n" + e.StackTrace);
    }
}
```

引数

`describeSObjects()` コールは sObject の任意の配列を取ることができます。

名前	型	説明
`sObjectType`	string	オブジェクト。指定された値は、組織で有効なオブジェクトである必要があります。完全なオブジェクトのセットについては、「標準オブジェクト」を参照してください。

応答

DescribeSObjectResult

障害

InvalidSObjectFault

UnexpectedErrorFault

関連トピック:

- describeSObject()
- describeGlobal()
- API コールの基礎
- Partner WSDL の使用

DescribeSObjectResult

重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。コード内の用語を変更すると、現在の実装が壊れる可能性があるため、このオブジェクトの名前は維持しました。

`describeSObjects()` コールは、`DescribeSObjectResult` オブジェクトの配列を返します。各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
`actionOverrides`	ActionOverride[]	override アクションの配列。override アクションによって、`urlDetail`、`urlEdit`、および `urlNew` 項目に指定された URL が置き換えられます。この項目は、API バージョン 32.0 以降で使用できます。
`activateable`	boolean	将来の使用のために予約されています。
`actionOverrides`	ActionOverride[]	override アクションの配列。override アクションによって、`urlDetail`、`urlEdit`、および `urlNew` 項目に指定された URL が置き換えられます。この項目は、API バージョン 32.0 以降で使用できます。
`associateEntityType`	string	オブジェクトが親オブジェクトに関連付けられている場合、その親への関連付けの種別(`History` など)。それ以外の場合、値は `null` です。API バージョン 50.0 以降で利用できます。
`associateParentEntity`	string	オブジェクトが親オブジェクトに関連付けられている場合、関連付けられている親オブジェクト。それ以外の場合、値は `null` です。API バージョン 50.0 以降で利用できます。
`childRelationships`	ChildRelationship[]	子リレーションの配列で、記述されている sObject への外部キーを持つ sObject の名前です。
`compactLayoutable`	boolean	オブジェクトが `describeCompactLayouts()` で使用できることを示します。

名前	型	説明
createable	boolean	オブジェクトが create() コールで作成可能か (true)、否か (false) を示します。
custom	boolean	カスタムオブジェクトであるかどうかを示します (カスタムオブジェクトの場合は true、そうでない場合は false)。
customSetting	boolean	カスタム設定項目であるか (true)、否か (false) を示します。
dataTranslationEnabled	boolean	オブジェクトでデータ翻訳が有効になっているか (true)、否か (false) を示します。API バージョン 49.0 以降で利用できます。
deepCloneable	boolean	将来の使用のために予約されています。
defaultImplementation	string	将来の使用のために予約されています。
deletable	boolean	オブジェクトが delete() コールで削除できるか (true)、否か (false) を示します。
deprecatedAndHidden	boolean	将来の使用のために予約されています。
extendedBy	string	将来の使用のために予約されています。
extendsInterfaces	string	将来の使用のために予約されています。
feedEnabled	boolean	オブジェクトで Chatter 項目が有効か (true)、否か (false) を示します。このプロパティは、API バージョン 19.0 以降で使用できます。
fields	Field[]	オブジェクトに関連付けられた項目の配列。このリストから情報を取得するメカニズムは、開発ツールにより異なります。
implementedBy	string	将来の使用のために予約されています。
implementsInterfaces	string	将来の使用のために予約されています。
isInterface	boolean	将来の使用のために予約されています。
keyPrefix	string	オブジェクト ID の 3 文字のプレフィックスコード。オブジェクト ID はオブジェクト型を示す 3 文字のコードが先頭に付けられます。たとえば、Account オブジェクトのプレフィックスは 001、Opportunity オブジェクトのプレフィックスは 006 です。主要なプレフィックスを複数のオブジェクトで共有する場合があるため、必ずしもオブジェクトを一意に識別するわけではありません。 子に 1 つ以上の親オブジェクトが存在する場合 (多態的な場合)、親のオブジェクト種別を確認するためにこの項目の値を使用します。たとえば、Task または Event の親の keyPrefix 値を取得しなければならない場合にこの値を使用します。

名前	型	説明
label	string	適用される場合は、ユーザーインターフェースで名前が変更されたタブまたは項目のラベルテキスト。変更されていない場合はオブジェクト名。たとえば、医療分野の組織では「取引先」を Patient に変更する場合があります。タブや項目は Salesforce ユーザーインターフェースで名前を変更できます。詳細は、Salesforce オンラインヘルプを参照してください。
labelPlural	string	オブジェクト名の複数形を表すラベルテキストです。
layoutable	boolean	オブジェクトが describeLayout() コールをサポートしているか (true)、否か (false) を示します。
mergeable	boolean	オブジェクトが同じ型の他のオブジェクトとマージ可能か (true)、否か (false) を示します。リード、取引先責任者、および取引先の場合は true です。
mruEnabled	boolean	最後に使用 (MRU) リスト機能がオブジェクトで有効か (true)、否か (false) を示します。
name	string	オブジェクトの名前。sObjectType パラメーターとして渡された文字列と同じです。
namedLayoutInfos	NamedLayoutInfo[]	デフォルトレイアウト以外の、オブジェクトで使用可能な特定の名前付きレイアウト。
networkScopeFieldName	string	エンティティを Experience Cloud サイトに絞り込む networkScopeField の API 参照名。ほとんどのエンティティでは、このプロパティの値は null です。
queryable	boolean	オブジェクトが query() コールでクエリ可能か (true)、否か (false) を示します。
recordTypeInfos	[]	このオブジェクトがサポートするレコードタイプの配列。ユーザーはここに表示されるものすべてを参照するのに、返されたすべてのレコードタイプへのアクセス権を所有している必要はありません。
replicateable	boolean	オブジェクトを getUpdated() および getDeleted() コールで複製可能か (true)、否か (false) を示します。
retrieveable	boolean	オブジェクトが retrieve() コールで取得可能か (true)、否か (false) を示します。
searchable	boolean	オブジェクトが search() コールで検索可能か (true)、否か (false) を示します。
searchLayoutable	boolean	検索レイアウト情報を describeSearchLayouts() コールで取得できるか (true)、否か (false) を示します。

名前	型	説明
supportedScopes	ScopeInfo	オブジェクトのサポートされる範囲のリスト。たとえば、取引先では [すべての取引先]、[私の取引先] および [私のチームの取引先] の範囲がサポートされる可能性があります。
triggerable	boolean	オブジェクトが Apex トリガーをサポートしているかどうかを示します。
undeletable	boolean	オブジェクトが undelete() コールで復元可能か (true)、否か (false) を示します。
updateable	boolean	オブジェクトが update() コールで更新可能か (true)、否か (false) を示します。
urlDetail	string	オブジェクトの参照のみの詳細ページへの URL。参照・更新可能である urlEdit と比較してください。クライアントアプリケーションはこの URL を使用し標準オブジェクトまたはカスタムオブジェクトの Salesforce ユーザーインターフェースへリダイレクトまたはアクセスできます。柔軟性を持たせ、将来の拡張に備えるために、返される urlDetail 値は動的です。クライアントアプリケーションの上位互換性を保証するため、できる限りこの機能を使用することをお勧めします。固定の URL が利用できないオブジェクトについては、この項目は空の値を返します。
urlEdit	string	オブジェクトの編集ページへの URL。たとえば、取引先オブジェクトの urlEdit 項目は https://***yourInstance***.salesforce.com/{ID}/e を返します。{ID} 項目を現在のオブジェクト ID で置き換えると、Salesforce ユーザーインターフェースの指定された取引先の編集ページを返します。参照のみである urlDetail と比較してください。クライアントアプリケーションはこの URL を使用し標準オブジェクトまたはカスタムオブジェクトの Salesforce ユーザーインターフェースへリダイレクトまたはアクセスできます。柔軟性を持たせ、将来の拡張に備えるために、返される urlDetail 値は動的です。クライアントアプリケーションの上位互換性を保証するため、できる限りこの機能を使用することをお勧めします。固定の URL が利用できないオブジェクトについては、この項目は空の値を返します。
urlNew	string	オブジェクトの新規/作成ページへの URL。クライアントアプリケーションはこの URL を使用し標準オブジェクトまたはカスタムオブジェクトの Salesforce ユーザーインターフェースへリダイレクトまたはアクセスできます。柔軟性を持たせ、将来の拡張に備えるために、返される urlNew 値は動的です。クライアントアプリケーションの上位互換性を保証するため、できる限りこの機能を使用することをお勧めします。固定の

名前	型	説明
		URL が利用できないオブジェクトについては、この項目は空の値を返します。

メモ: Boolean 値を持つプロパティは、特定の API コールがオブジェクトで使用できるかどうかを示します。ただし、権限などのその他の要素も、オブジェクトでその操作を実行できるかどうかに影響します。

ActionOverride

ActionOverride は、オブジェクトのデフォルトアクションページを置き換えるアクションに関する詳細を提供します。たとえば、オブジェクトの新規/作成ページをカスタムページで置き換えるように設定されている場合があります。この種別は、API バージョン 32.0 以降で使用できます。

名前	型	説明
`formFactor`	string	アクションの上書きが適用される環境を表します。たとえば、この項目の `Large` 値は Lightning Experience デスクトップ環境を表し、Lightning ページおよび Lightning コンポーネントで有効です。`Small` 値は、携帯電話またはタブレットの Salesforce モバイルアプリケーションを表します。 この項目は、API バージョン 37.0 以降で使用できます。
`isAvailableInTouch`	boolean	Salesforce モバイルアプリケーションで override アクションが有効になっているか (`true`)、否か (`false`) を示します。
`name`	string	デフォルトのアクションを上書きするアクションの名前。たとえば、新規/作成ページがカスタムアクションで上書きされている場合、アクション名は「New」などである場合があります。
`pageId`	reference	override アクションのページの ID。
`url`	string	Visualforce ページなど、アクション上書きに使用する項目の URL。Lightning ページの上書きでは `null` として返されます。

ChildRelationship

説明されている sObject への外部キーを持つ sObject の名前です。

名前	型	説明
`cascadeDelete`	boolean	親オブジェクトが削除されるときに子オブジェクトも削除されるか (`true`)、否か (`false`) を示します。

名前	型	説明
childSObject	string	親の sObject への外部キーを持つオブジェクトの名前。
deprecatedAndHidden	boolean	将来の使用のために予約されています。
field	string	親の sObject への外部キーを持つ項目の名前。
junctionIdListNames	String[]	オブジェクトに関連付けられている連結 ID のリストの名前。各 ID は、関連付けられたオブジェクトとのリレーションがあるオブジェクトを表します。 JunctionIdList 項目についての詳細は、「データ型」を参照してください。
junctionReferenceTo	String[]	junctionIdListNames プロパティの多態的なキーで参照できるオブジェクト名のコレクション。 これらのオブジェクト名を照会できます。
relationshipName	string	リレーションの名前。通常は childSObject の値の複数形。
restrictedDelete	boolean	親オブジェクトが子オブジェクトによって参照されているため、親オブジェクトを削除できるか (true)、否か (false) を示します。

Field

DescribeSObjectResult において、Field プロパティには fields オブジェクトの配列が含まれます。各項目は API オブジェクトの項目を表します。配列は、ユーザーの項目レベルのセキュリティ設定の定義に基づき、ユーザーが参照できる項目のみを含みます。

名前	型	説明
autonumber	boolean	この項目が自動採番項目であるか (true)、否か (false) を示します。SQL の IDENTITY 型に似て、自動採番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。自動採番項目は参照のみの項目であり、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)。自動採番項目は、全体的に Salesforce ユーザーインターフェースで構成されています。API はこの属性へのアクセスを提供しているため、クライアントアプリケーションは指定された項目が自動採番項目かどうかを確認できます。
byteLength	int	可変長項目 (バイナリ項目も含む) の最大サイズをバイトで指定。
calculated	boolean	項目がカスタム数式項目であるか (true)、否か (false) を示します。カスタム数式項目は常に参照のみです。

名前	型	説明
caseSensitive	boolean	この項目が大文字と小文字を区別するかどうかを示します(区別する場合は true、しない場合は false)。
controllerName	string	この選択リストの値を制御する項目の名前。type が picklist または multipicklist であり、dependentPicklist が true の場合のみ適用されます。制御項目から連動項目への対応付けは、この選択リストの各 PicklistEntry の validFor 属性に格納されます。
createable	boolean	項目を作成できるか(true)、否か(false)を示します。true の場合、この項目値を create() コールで設定できます。
custom	boolean	項目がカスタム項目であるか(true)、否か(false)を示します。
dataTranslationEnabled	boolean	項目でデータ翻訳が有効になっているか(true)、否か(false)を示します。API バージョン 49.0 以降で利用できます。
defaultedOnCreate	boolean	作成時にこの項目がデフォルト設定されているか(true)、否か(false)を示します。true の場合、この項目の値が create() 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、値が Stage 項目から取得されている、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、値は現在のユーザーから取得されます。
defaultValueFormula	string	数式が使用されていない場合に、この値に指定されるデフォルト値。値が指定されていない場合、この項目は返されません。
dependentPicklist	boolean	選択リストが連動選択リストであるかどうかを示します(利用可能な値が制御項目で選択された値に従う場合は true、そうでない場合は false)。「連動選択リストについて」を参照してください。
deprecatedAndHidden	boolean	将来の使用のために予約されています。
digits	int	整数型の項目で使用。最大桁数。整数値がこの桁数を超えた場合、API はエラーを返します。
encrypted	boolean	メモ: このページは、従来の暗号化ではなく Shield Platform Encryption について書かれています。この違いについては、こちらをクリックしてください。 この項目が暗号化されているかどうかを示します。この値は、describeSObjects() コールの結果が true の場合にのみ結果に表示され、それ以外の場合は結果から除外されます。この項目は、API バージョン 31.0 以降で使用できます。

名前	型	説明
extraTypeInfo	string	項目が textarea データ型の場合、テキストエリアがプレーンテキストか (plaintextarea)、リッチテキストか (richtextarea) を示します。 項目が url データ型の場合、この値が imageurl ならば、URL は画像ファイルを参照します。標準オブジェクトの標準項目でのみ使用できます (Account.photoUrl、Contact.photoUrl など)。 項目が reference データ型の場合、外部オブジェクトリレーションの種別を示します。外部オブジェクトでのみ使用できます。 • null — 参照関係 • externallookup — 外部参照関係 • indirectlookup — 間接参照関係
filterable	boolean	項目を除外できるか (true)、否か (false) を示します。true の場合、この項目を query() コールのクエリ文字列の WHERE 句で指定できます。
filteredLookupInfo	FilteredLookupInfo	項目がルックアップ検索条件を持つ reference データ型の場合、filteredLookupInfo にはその項目のルックアップ検索条件情報が含まれます。ルックアップ検索条件がない場合、または検索条件が無効な場合、この項目は null になります。 この項目は、API バージョン 31.0 以降で使用できます。
formula	string	この項目に指定された数式。数式が指定されていない場合、この項目は返されません。
groupable	boolean	項目を SOQL クエリの GROUP BY 句に含めることができるかどうかを示します (できる場合は true、できない場合は false)。『Salesforce SOQL および SOSL リファレンス』の「GROUP BY」を参照してください。API バージョン 18.0 以降で利用できます。
highScaleNumber	boolean	項目詳細の指定内容に関係なく、項目に小数点以下 8 桁までの数値が保存されるか (true)、否か (false) を示します。価格が数分の 1 セントの大量の製品の通貨を処理するために使用されます。組織で、高スケールの単価設定が有効ではない場合、この項目は返されません。API バージョン 33.0 以降で使用できます。
htmlFormatted	boolean	ハイパーリンクのカスタム数式項目などの項目が HTML のために形式化されており、HTML として表示するためのエンコーディングが必要かどうかを示します (必要な場合は true、必要でない場合は false)。カスタム数式項目の項目に IMAGE テキスト関数があるかどうかも示します。
idLookup	boolean	upsert() コールのレコードの指定に項目を使用できるかどうかを指定します (指定できる場合は true、できない場合は false)。

名前	型	説明
`inlineHelpText`	string	この項目の項目レベルのヘルプでフロート表示テキストとして表示されるテキストです。 メモ: このプロパティは、オブジェクトの 1 つ以上の項目に値が含まれていないと返されません。少なくとも1つの項目に項目レベルのヘルプが存在する場合、オブジェクトのすべての項目がプロパティを項目レベルのヘルプの値で表示します。項目レベルのヘルプが空の項目では null 値となります。
`label`	string	Salesforce ユーザー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。このラベルはローカライズが可能です。
`length`	int	Unicode 文字での最大サイズ (バイトではないことに注意) と 255 のうちいずれか少ない方を返します。getLength() プロパティによって返される最大値は 255 です。API バージョン 49.0 以降で利用できます。
`mask`	string	将来の使用のために予約されています。
`maskType`	string	将来の使用のために予約されています。
`name`	string	`create()`、`delete()`、`query()` など、API コールで使用される項目名。
`nameField`	boolean	項目が名前項目であるか (`true`)、否か (`false`) を示します。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 オブジェクトなど、`FirstName` および `LastName` 項目が使用されている場合を除き、1オブジェクトにつき1つに制限されます。 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 `nameField` は `true` に設定されます。複合名が存在しない場合、`FirstName` および `LastName` ではこの項目が `true` に設定されます。
`namePointing`	boolean	項目の値が、このオブジェクトの親の Name であるか (`true`)、否か (`false`) を示します。親のオブジェクト種別が2つ以上ある可能性のあるオブジェクトで使用します。たとえば、ToDo は取引先と取引先責任者が親であることが考えられます。
`nillable`	boolean	項目を空白にできるか (`true`)、否か (`false`) を示します。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。null 値が許可されないオブジェクトでは、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。

名前	型	説明
permissionable	boolean	項目に FieldPermissions を指定可能か (true)、否か (false) を示します。
picklistValues	PicklistEntry[]	選択リストの有効な値のリストを提供します。restrictedPicklist が true の場合のみ指定します。
polymorphicForeignKey	boolean	外部キーに複数のエンティティ種別が含まれるか (true)、否か (false) を示します。
precision	int	倍精度浮動小数点数型の項目で使用します。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示します。
relationshipName	string	主従関係の項目の場合、リレーションの名前。
relationshipOrder	int	主従関係の項目の場合、リレーションの種類。有効な値は、次のとおりです。 • 0: 項目が主リレーションの場合 • 1: 項目がセカンダリリレーションの場合
referenceTargetField	string	外部オブジェクトの間接参照関係にのみ適用されます。値が子外部オブジェクトの間接参照関係項目の値と照合される、親標準またはカスタムオブジェクトのカスタム項目の名前。この照合は、相互に関連するレコードを判別するために行われます。この項目は、API バージョン 32.0 以降で使用できます。
referenceTo	string[]	他のオブジェクトを参照する項目において、この配列は参照されるオブジェクトのオブジェクト種別を示します。
restrictedPicklist	boolean	項目が制限付き選択リストであるか (true)、否か (false) を示します。
scale	int	倍精度浮動小数点数型の項目で使用します。小数点の右側の最大桁数です。API は小数点の右側の余分な桁は通知することなく切り捨てます。ただし、小数点の左側の桁数が多すぎる場合は、エラーのレスポンスを返します。
searchPrefilterable	boolean	外部キーが SOSL WHERE 句で使用されたときに外部キーを事前絞り込みに組み込むか (true) 組み込まないか (false) を示します。*事前絞り込み*では、完全な検索クエリを実行する前に特定の項目値で絞り込みを行います。API バージョン 40.0 以降で使用できます。
soapType	SOAPType	設定可能な値のリストは、「SOAPType」を参照してください。
sortable	boolean	クエリでこの項目に基づいて並び替えができるか (true)、できないか (false)) を示します。
type	FieldType	設定可能な値のリストは、「FieldType」を参照してください。

名前	型	説明
unique	boolean	この項目が一意である必要があるかどうかを示します(一意である必要がある場合は true、そうでない場合は false)。
updateable	boolean	次のいずれかを示します。 • 項目を更新できるか (true)、否か (false)。 true の場合、この項目値を update() コールで設定できます。 • 項目がカスタムオブジェクトの主従関係項目である場合は、子レコードの親を別の親レコードに変更できるか (true)、否か (false) を示します。
writeRequiresMasterRead	boolean	この項目は主従関係にのみ適用されます。ユーザーが、子レコードの挿入、更新および削除を行う場合に親レコードに対して必要なのは、参照の共有アクセス権限であるのか (true)、編集の共有アクセス権限であるのか (false) を示します。いずれの場合も、ユーザーは子オブジェクトに対しても、作成、編集および削除のオブジェクト権限が必要です。

FieldType

DescribeSObjectResult で関連付けられた Field オブジェクトにおいて、type 項目には次の文字列の 1 つを格納できます。項目のデータ型の詳細は、「データ型」を参照してください。

type 項目値	Field オブジェクトに含まれる内容
string	文字列の値。
boolean	boolean の (true/false) の値。
int	整数値。
double	double 値。
date	日付の値。
datetime	日付/時間の値。
base64	Base64 でエンコーディングされた任意のバイナリデータ (型は base64Binary)。Attachment、Document および Scontrol オブジェクトで使用します。
ID	オブジェクトの主キー項目。ID については、「データ型」を参照してください。
reference	別のオブジェクトへの相互参照。SQL の外部キー項目に似ています。
currency	通貨の値。

type 項目値	Field オブジェクトに含まれる内容
textarea	複数行のテキスト項目として表示される文字列。
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。
url	URL の値。通常クライアントアプリケーションではハイパーリンクとして表示されます。 Field.extraTypeInfo が imageurl の場合、URL は画像を参照し、代わりに画像として表示できます。
email	メールアドレス。
combobox	列挙値のセットを提供するコンボボックスで、ユーザーはリストにない値も指定できます。
picklist	単一の値を選択可能な列挙値のセットを含む、1 つの値のみを選択できる選択リスト。
multipicklist	複数の値を選択可能な列挙値のセットを提供する複数選択リスト。
anyType	次のいずれかの型の値を指定できます。string、picklist、boolean、int、double、percent、ID、date、dateTime、url または email。

FilteredLookupInfo

DescribeSObjectResult に関連付けられた Field オブジェクトの filteredLookupInfo 項目には、この項目に関連付けられたルックアップ検索条件に関する情報が含まれます。

このサブタイプは、API バージョン 31.0 以降で使用できます。

名前	型	説明
controllingFields	string[]	ルックアップ検索条件が参照元オブジェクトに依存している場合の項目の制御項目の配列。
dependent	boolean	ルックアップ検索条件が参照元オブジェクトに依存しているか (true)、否か (false) を示します。
optionalFilter	boolean	ルックアップ検索条件が省略可能か (true)、否か (false) を示します。

SOAPType

DescribeSObjectResult は、記述されるオブジェクトについての情報を提供する値の項目の配列を含む fields プロパティを返します。その項目の 1 つである soapType は、次の文字列値の 1 つを含みます。xsd: から始まるすべての値は、XML スキーマのプリミティブデータ型です。XML スキーマのプリミティブデータ型の詳細

は、http://www.w3.org/TR/xmlschema-2/ の World Wide Web Consortium の公開資料『*XML Schema Part 2: Data Types*』を参照してください。

値	説明
`tns:ID`	sObject に関連付けられた一意な ID。ID については、「データ型」を参照してください。
`xsd:anyType`	ID、boolean、double、integer、string、date または dateTime のいずれかを指定できます。
`xsd:base64Binary`	Base 64 で符号化されたバイナリデータ。
`xsd:boolean`	boolean の (`true`/`false`) の値。
`xsd:date`	日付の値。
`xsd:dateTime`	日付/時間の値。
`xsd:double`	double 値。
`xsd:int`	整数値。
`xsd:string`	文字列。

PicklistEntry

DescribeSObjectResult で関連付けられた `Field` オブジェクトにおいて、`picklistValues` 項目は `PicklistEntry` プロパティの配列を含みます。各 `PicklistEntry` は次の文字列値の 1 つを含むことができます。詳細は、「データ型」を参照してください。

名前	型	説明
`active`	boolean	ユーザー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か(`true`)、否か(`false`)を示します。
`validFor`	byte[]	ビットセットで、各ビットはこの `PicklistEntry` が有効かどうかを表す制御値を示します。「連動選択リストについて」を参照してください。
`defaultValue`	boolean	この項目が選択リストのデフォルト項目であるか(`true`)、否か(`false`)を示します。選択リスト内の 1 つのアイテムのみをデフォルトに設定できます。
`label`	string	選択リスト内のこのアイテムの名前を表示します。
`value`	string	選択リスト内のこのアイテムの値。

連動選択リストについて

連動選択リストは、制御項目と共に動作し、その値に検索条件を適用します。制御項目で選択した値は、連動選択リストの値にも適用されます。

連動選択リストには、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を使用できます。制御項目には、対応する1つ以上の連動項目で使用可能な値を制御する、標準またはカスタムの選択リスト(1つ以上、200以下の値を含む)やチェックボックスの項目を使用できます。

次の例では、制御選択リスト `Beverage` には2つの値があり、その値は連動選択リスト［飲み物の種類］に関連付けられています。

飲み物	飲み物の種類
コーヒー	カフェインなし
	レギュラー
紅茶	カモミール
	アールグレイ
	イングリッシュブレックファスト

連動選択リストの値を表す各 `PicklistEntry` では、`validFor` にビットセットが含まれています。各ビットは、`PicklistEntry` が有効かどうかを表す制御値を示します。ビットは左から右へと読みます。

連動選択リストについての詳細は、Salesforce オンラインヘルプの「連動選択リスト」を参照してください。

連動選択リストのサンプル Java コード

```
public void dependentPicklistSample() {
  // inner class to decode a "validFor" bitset
  class Bitset {
    byte[] data;

    public Bitset(byte[] data) {
      this.data = data == null ? new byte[0] : data;
    }

    public boolean testBit(int n) {
      return (data[n >> 3] & (0x80 >> n % 8)) != 0;
    }

    public int size() {
      return data.length * 8;
    }
  }

  try {
    DescribeSObjectResult describeSObjectResult = connection.describeSObject("Case");
```

```
    Field[] fields = describeSObjectResult.getFields();
    // create a map of all fields for later lookup
    Map fieldMap = new HashMap();
    for (int i = 0; i < fields.length; i++) {
      fieldMap.put(fields[i].getName(), fields[i]);
    }
    for (int i = 0; i < fields.length; i++) {
      // check whether this is a dependent picklist
      if (fields[i].getDependentPicklist()) {
        // get the controller by name
        Field controller = (Field)fieldMap.get(fields[i].getControllerName());
        System.out.println("Field '" + fields[i].getLabel() + "' depends on '" +
        controller.getLabel() + "'");
        PicklistEntry[] picklistValues = fields[i].getPicklistValues();
        for (int j = 0; j < picklistValues.length; j++) {
          // for each PicklistEntry: list all controlling values for which it is valid
          System.out.println("Item: '" + picklistValues[j].getLabel() +
          "' is valid for: ");
          Bitset validFor = new Bitset(picklistValues[j].getValidFor());
          if (FieldType.picklist == controller.getType()) {
            // if the controller is a picklist, list all
            // controlling values for which this entry is valid
            for (int k = 0; k < validFor.size(); k++) {
              if (validFor.testBit(k)) {
                // if bit k is set, this entry is valid for the
                // for the controlling entry at index k
                System.out.println(controller.getPicklistValues()[k].getLabel());
              }
            }
          } else if (FieldType._boolean == controller.getType()) {
            // the controller is a checkbox
            // if bit 1 is set this entry is valid if the controller is checked
            if (validFor.testBit(1)) {
              System.out.println(" checked");
            }
            // if bit 0 is set this entry is valid if the controller is not checked
            if (validFor.testBit(0)) {
              System.out.println(" unchecked");
            }
          }
        }
      }
    } catch (ConnectionException ce) {
      ce.printStackTrace();
    }
}
```

RecordTypeInfo

古いオブジェクト RecordTypeMapping **のベースクラスです。このオブジェクトには、** layoutId **と** picklistForRecordType **以外の** RecordtypeMapping **のすべての既存項目が含まれています。**

名前	型	説明
available	boolean	このレコードタイプが利用可能かどうかを示します(利用可能な場合は true、そうでない場合は false)。利用可能かどうかという情報は、新しいレコードの作成時に利用可能なレコードタイプの一覧をユーザーに表示するのに使用します。
defaultRecordTypeMapping	boolean	デフォルトのレコードタイプの対応付けかどうかを示します(デフォルトの場合は true、そうでない場合は false)。
developerName	string	このレコードタイプの API 参照名。API バージョン 43.0 以降で使用できます。
master	boolean	これが主レコードタイプであるか(true)、否か(false)を示します。主レコードタイプは、レコードに関連付けられているカスタムレコードタイプがない場合に使用される、デフォルトのレコードタイプです。
name	string	レコードタイプの名前。
recordTypeId	ID	レコードタイプの ID。

NamedLayoutInfo

オブジェクトの名前付きレイアウトの名前。標準オブジェクトには定義された名前付きレイアウトを使用できます。名前付きレイアウトは、プロファイルとレコードタイプの両方で主レイアウトとは分離されています。レイアウト名についての詳細は、「describeLayout()」を参照してください。

名前	型	説明
name	string	このレイアウトの名前。

ScopeInfo

オブジェクトレコードの絞り込みに使用できるオブジェクトの範囲。たとえば、取引先では現在のユーザーの取引先レコードのみを絞り込む、ScopeInfo の「mine」 (UI 表示ラベルの [私の取引先]) がサポートされる可能性があります。

名前	型	説明
label	string	この範囲の UI 表示ラベル。
name	string	この範囲の名前。

describeSoftphoneLayout()

Salesforce CRM Call Center のソフトフォンのレイアウト情報を取得します。

構文

```
DescribeSoftphoneLayoutResult[] = connection.describeSoftphoneLayout();
```

使用方法

ソフトフォンレイアウト情報の取得にこのコールを使用します。Salesforce CRM Call Center のコンテキストでのみ使用し、クライアントプログラムから直接コールしないでください。

引数

このコールはオブジェクトを取りません。

応答

レスポンスは DescribeSoftphoneLayoutResult オブジェクトです。

名前	型	説明
callTypes	DescribeSoftphoneLayoutCallType[]	許可された各通話種別と関連付けられた属性のセット。通話種別は、着信、発信、および内線のいずれかとなります。
id	ID	レイアウトの ID。レイアウトオブジェクトは API からは公開されません。
name	string	通話種別名。着信、発信、および内線のいずれかとなります。

DescribeSoftphoneLayoutCallType

各 DescribeSoftphoneLayoutResult オブジェクトには 1 つ以上の通話種別が含まれます。

名前	型	説明
infoFields	DescribeSoftphoneLayoutInfoField[]	ソフトフォンレイアウト内の情報項目のセット。
name	string	レイアウトの名前。

名前	型	説明
`screenPopOptions`	DescribeSoftphoneScreenPopOption[]	通話の詳細が既存のレコードに一致する場合、またはしない場合のスクリーンポップ表示方法を指定するソフトフォンレイアウトの設定。 この項目は、API バージョン 18.0 以降で使用できます。
`screenPopsOpenWithin`	string	通話の詳細が既存のレコードに一致する場合、または一致しない場合に、新しいブラウザーウィンドウに画面ポップを表示するかどうかを指定するソフトフォンレイアウトの設定。 この項目は、API バージョン 18.0 以降で使用できます。
`sections`	DescribeSoftphoneLayoutSection[]	ソフトフォンレイアウト内のオブジェクト名と対応する項目名のセット。通話種別には、オブジェクトごとに 1 つのセクションがあります。

DescribeSoftphoneLayoutInfoField

ソフトフォンレイアウト内の情報項目。

名前	型	説明
`name`	string	ソフトフォンレイアウト内の Salesforce オブジェクトに対応しない情報を含む項目の名前。たとえば、通話者 ID はこの項目で指定することができます。この項目には通話種別の静的情報が格納されます。

DescribeSoftphoneLayoutSection

DescribeSoftphoneLayoutResult オブジェクトで返される各通話種別には、通話種別ごとに 1 つのセクションが含まれます。各セクションには、オブジェクトと項目のペアが含まれます。

名前	型	説明
`entityApiName`	string	取引先とケースのセットなど、ソフトフォンレイアウトに表示される項目に対応する Salesforce アプリケーションのオブジェクト名。
`items`	DescribeSoftphoneLayoutItem[]	ソフトフォンレイアウト項目のセット。

DescribeSoftphoneLayoutItem

各レイアウト項目は、Salesforce のレコードに対応します。

名前	型	説明
itemApiName	string	取引先「Acme」など、ソフトフォンレイアウトに表示される項目に対応する Salesforce アプリケーションのレコード名。

DescribeSoftphoneScreenPopOption

DescribeSoftphoneLayoutResult オブジェクトで返される各通話種別には、通話種別ごとに 1 つの screenPopOptions 項目が含まれます。各 screenPopOptions 項目には、スクリーンポップ設定の詳細が含まれます。

名前	型	説明
matchType	string	単一レコードまたは複数レコードに一致する通話詳細、あるいは一致するレコードがない通話詳細に対して、スクリーンポップを実行するためのソフトフォンレイアウトの設定。
screenPopData	string	通話の matchType に応じてポップする特定のオブジェクトまたはページについてのソフトフォンレイアウトの設定。たとえば、通話の詳細があるレコードに一致した場合、指定した Visualforce ページをポップします。
screenPopType	picklist	通話の matchType に応じてスクリーンポップを実行する方法を指定する設定。たとえば、通話の詳細があるレコードに一致した場合に詳細ページを表示するか、どのページも表示しないか、といった指定を行います。

サンプルコード — Java

このサンプルでは、ソフトフォンのレイアウトを describe して、コンソールにそのプロパティを書き込みます。次に、許可された通話種別を取得します。各通話種別で、情報項目、レイアウトセクション、およびレイアウトセクションのレイアウト項目を取得します。これらの値をコンソールに書き込みます。

```
public void describeSoftphoneLayout() {
  try {
    DescribeSoftphoneLayoutResult result =
        connection.describeSoftphoneLayout();
    System.out.println("ID of retrieved Softphone layout: " +
        result.getId());
    System.out.println("Name of retrieved Softphone layout: " +
        result.getName());
    System.out.println("\nContains following " +
        "Call Type Layouts\n");
    for (DescribeSoftphoneLayoutCallType type :
          result.getCallTypes()) {
      System.out.println("Layout for " + type.getName() +
          " calls");
      System.out.println("\tCall-related fields:");
      for (DescribeSoftphoneLayoutInfoField field :
            type.getInfoFields()) {
        System.out.println("\t\t{" + field.getName());
      }
```

```
      System.out.println("\tDisplayed Objects:");
      for (DescribeSoftphoneLayoutSection section :
            type.getSections()) {
        System.out.println("\t\tFor entity " +
            section.getEntityApiName() +
            " following records are displayed:"
        );
        for (DescribeSoftphoneLayoutItem item :
              section.getItems()) {
          System.out.println("\t\t\t" + item.getItemApiName());
        }
      }
    }
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

サンプルコード — C#

このサンプルでは、ソフトフォンのレイアウトをdescribeして、コンソールにそのプロパティを書き込みます。次に、許可された通話種別を取得します。各通話種別で、情報項目、レイアウトセクション、およびレイアウトセクションのレイアウト項目を取得します。これらの値をコンソールに書き込みます。

```
/// Demonstrates how to retrieve the layout information
/// for a Salesforce CRM Call Center Softphone
public void DescribeSoftphoneLayoutSample()
{
   try
   {
       DescribeSoftphoneLayoutResult dsplResult = binding.describeSoftphoneLayout();

       // Display the ID and Name of the layout
       Console.WriteLine("ID of retrieved Softphone layout: {0}", dsplResult.id);
       Console.WriteLine("Name of retrieved Softphone layout: {0}", dsplResult.name);

       // Display the contents of each Call Type
       Console.WriteLine("\nContains following Call Type Layouts\n");
       foreach (DescribeSoftphoneLayoutCallType dsplCallType in dsplResult.callTypes)
       {
           Console.WriteLine("Layout for {0} calls", dsplCallType.name);

           // Display the call-related fields contained in the call type
           Console.WriteLine("\tCall-related fields:");
           foreach (DescribeSoftphoneLayoutInfoField dsplInfoField
               in dsplCallType.infoFields)
           {
            Console.WriteLine("\t\t{0}", dsplInfoField.name);
           }

           // Display the objects that are included in the layout
           Console.WriteLine("\tDisplayed Objects:");
           foreach (DescribeSoftphoneLayoutSection dsplSection
```

```
                in dsplCallType.sections)
            {
                Console.WriteLine("\t\tFor entity {0} following records are displayed:",
                    dsplSection.entityApiName);
                foreach (DescribeSoftphoneLayoutItem dsplItem in dsplSection.items)
                {
                    Console.WriteLine("\t\t\t{0}", dsplItem.itemApiName);
                }
             }
        }
    }
    catch (SoapException e)
    {
        Console.WriteLine(e.Message);
        Console.WriteLine(e.StackTrace);
        Console.WriteLine(e.InnerException);
    }
}
```

describeSoqlListViews()

リストビューに関する SOQL クエリおよびその他の情報を取得します。

構文

```
connection.describeSoqlListViews(DescribeSoqlListViewsRequest request);
```

使用方法

`describeSoqlListViews()` コールを使用して、ID、列、SOQL クエリなど、リストビューに関する情報を取得します。このコールは、カスタムアプリケーション内で既存のリストビューを操作する SOQL を使用する場合に便利です。このコールは、API バージョン 32.0 以降で使用できます。

サンプルコード — Java

```
public void example() throws Exception {
         DescribeSoqlListViewsRequest request =
createDescribeSoqlListViewsRequest(listViewId, null);
         this.getClient().describeSoqlListViews(request);
      }
```

引数

名前	型	説明
request	DescribeSoqlListViewsRequest	リストビューの完全修飾名または ID と、リストビューが関連付けられているオブジェクト。

応答

1 つ以上の DescribeSoqlListView オブジェクトが含まれる DescribeSoqlListViewResult オブジェクト。

障害

InvalidSObjectFault

UnexpectedErrorFault

DescribeSoqlListView

列、sObject 種別、SOQL クエリなど、指定されたリストビューに関する情報が含まれます。

DescribeSoqlListView 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
columns	ListViewColumn[]	リストビュークエリから返される列。
id	ID	リストビューの完全修飾 ID。
orderBy	ListViewOrderBy[]	結果の並び替えのキーになっている項目、並び替え順、および並び替え後の null 値の位置のリスト。
query	string	リストビューの完全に構成された SOQL クエリ。
relatedEntityId	ID	キャンペーンの ID (キャンペーンの範囲が使用された場合)。
scope	string	結果の制限に使用する filterScope。
scopeEntityId	ID	キューまたは価格表の ID (キューまたは価格表の範囲が使用された場合)。
sobjectType	string	リストビューに関連付けられたオブジェクト。
whereCondition	SoqlWhereCondition	リストビューに対する検索条件。検索条件によって、リストビューに表示されるレコードを詳細に制御できます。

関連トピック:

https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select_using_scope.htm

DescribeSoqlListViewParams

リストビューから SOQL を取得するには、`describeSoqlListViews()` で `DescribeSoqlListViewParams` オブジェクトを使用します。

`DescribeSoqlListViewParams` 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
`developerNameOrId`	string	リストビューの ID または完全修飾開発者名。
`sobjectType`	string	リストビューの sObject の API 名。

DescribeSoqlListViewResult

1 つ以上の `DescribeSoqlListView` オブジェクトが含まれ、各オブジェクトには、1 つ以上のリストビューに関する情報 (それぞれの ID、sObject 種別、列、SOQL クエリなど) が含まれます。

`DescribeSoqlListViewResult` 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
`describeSoqlListViews`	DescribeSoqlListView[]	1 つ以上のリストビューに関する情報 (それぞれの ID、sObject 種別、列、SOQL クエリなど)。

DescribeSoqlListViewsRequest

リストビューに関する情報を取得するには、`describeSoqlListViews()` で `DescribeSoqlListViewsRequest` オブジェクトを使用します。

`DescribeSoqlListViewsRequest` 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
`listViewParams`	DescribeSoqlListViewParams[]	記述するリストビューを指定するパラメーターのリスト。

エディション

使用可能なインターフェース: Salesforce Classic

使用可能なエディション:

ListViewColumn

1 つのリストビュー列に関するメタデータが含まれます。

ListViewColumn オブジェクトは `describeSoqlListViews()` および `executeListView()` コールによって返されます。ListViewColumn には次のプロパティがあります。

名前	型	説明
`ascendingLabel`	string	列を昇順で並び替えるための、ローカライズされた種別固有の表示ラベル。たとえば、テキスト項目の場合は「A-Z」、数値項目の場合は「Low to High」などになります。列が並び替えできない場合は、null に設定します。
`descendingLabel`	string	列を昇順で並び替えるための、ローカライズされた種別固有の表示ラベル。たとえば、テキスト項目の場合は「Z-A」、数値項目の場合は「High to Low」などになります。列が並び替えできない場合は、null に設定します。
`fieldNameOrPath`	string	列の項目名または SOQL 項目パス。
`hidden`	boolean	`true` の場合、列は表示されず、他の列の表示や他のクライアント側ロジックをサポートするためにのみ存在していることを示します。
`label`	string	列のローカライズされた表示ラベル。
`searchable`	boolean	列が検索可能かどうか。
`selectListItem`	string	列の SOQL SELECT 項目。この項目は、表示形式の違いにより、項目名またはパスとは異なる場合があります (選択リストの toLabel など)。
`sortDirection`	orderByDirection	列挙値。列が並び替え可能な場合は次のいずれかになります。 • 昇順 • 降順 列が並び替えできない場合は、null に設定します。
`sortIndex`	int	複数レベルの並び替え内での列の位置を示すゼロベースのインデックス。または、レコードが列で並び替えられない場合は null。
`sortable`	boolean	列が並び替え可能かどうか。並び替え可能な場合は、ExecuteListViewOrderBy パラメーターで参照されることがあります。
`type`	FieldType	列のデータ型。

ListViewOrderBy

リストビューからレコードが返される順序を決定するには、ListViewOrderBy オブジェクトで `executeListView()` を使用します。

ListViewOrderBy オブジェクトは、`describeSoqlListViews()` コールから返され、`executeListView()` コールへの省略可能な入力となります。次のプロパティがあります。

名前	型	説明
`fieldNameOrPath`	string	レコードの並び替えキーとなる項目の項目名または SOQL パス。
`nullsPosition`	orderByNullsPosition	結果内での並び替え後の null の位置を決定する列挙値。 • first • last
`sortDirection`	orderByDirection	結果の並び替え順を決定する列挙値。 • ascending • descending

SoqlWhereCondition

リストビューの SOQL 検索条件に関する情報が含まれます。

SoqlWhereCondition にリストされる各条件は、条件の演算子を使用して項目値を比較値と比較する SOQL WHERE 句の条件式を表します。各条件には、次のプロパティが含まれます。

名前	型	説明
`field`	string	検索条件で使用するオブジェクト項目。
`operator`	soqlOperator	検索条件の演算子。次の演算子があります。 • equals — 項目値が指定された値と一致する場合、条件は true。等号演算子を使用する文字列の比較の場合、大文字と小文字が区別される一意の項目は大文字と小文字が区別され、他のすべての項目は大文字と小文字が区別されません。 • excludes — 複数選択リスト項目で選択された項目値が条件値のリストにない場合、条件は true。 • greaterThan — 項目値が指定された値より大きい場合、条件は true。 • greaterThenOrEqualTo — 項目値が指定された値以上の場合、条件は true。 • in — 項目値が値リスト内の指定された値と一致する場合、条件は true。 • includes — 複数選択リスト項目で選択された項目値が条件値のリストにある場合、条件は true。 • lessThan — 項目値が指定された値より小さい場合、条件は true。 • lessThanOrEqualTo — 項目値が指定された値以下の場合、条件は true。

名前	型	説明
		• like — 『SOQL および SOSL リファレンス』の「比較演算子」に記載されている文字の一致ロジックを使用して、項目値が指定された値と一致する場合、条件は true。 • notEquals — 項目値が指定された値と一致しない場合、条件は true。 • notIn — 項目値が値リスト内の指定された値と一致しない場合、条件は true。 • notLike — 『SOQL および SOSL リファレンス』の「比較演算子」に記載されている文字の一致ロジックを使用して、項目値が指定された値と一致しない場合、条件は true。API バージョン 41.0 以降で使用できます。 • within — 位置情報に基づく比較を使用して、項目値の位置が距離の値内にある場合、条件は true。詳細は、『SOQL および SOSL リファレンス』の「位置情報に基づく SOQL クエリ」を参照してください。
`values`	string[]	`operator` 比較ロジックを使用して項目値と比較する 1 つ以上の値のリスト。

SoqlWhereConditions の評価

SOAP API では、Salesforce は SoqlWhereCondition のサブクラスを使用して条件のさまざまなカテゴリを表します。開発言語の型の比較機能 (Java の `instanceof` 演算子など) を使用して、SoqlWhereCondition の特定のインスタンスで使用されているサブクラスを判断します。

SoqlConditionGroup サブクラスは、SOQL WHERE 句の条件のグループを表し、次のプロパティを使用します。

名前	型	説明
`conditions`	condition[]	検索条件のリスト。リストビューで検索条件ロジックを使用する場合、論理的な各検索条件グループが条件リストで表されます。
`conjunction`	soqlConjunction	論理的な各検索条件グループで、複数の条件に使用する検索条件ロジックを表す結合演算子。次のような値があります。 • and — SoqlWhereCondition 全体ですべての条件が true である必要があります。 • or — SoqlWhereCondition 全体でいずれかの条件が true である必要があります。

SoqlNotCondition サブクラスは、`like` 演算子の特別な使用を表します。API バージョン 40.0 以前では、`not like` 演算子 (UI では [次の文字列を含まない] と表示) を使用して作成された SoqlWhereCondition を評価する場合、条件

の演算子の値は `like` になります。Salesforce は SoqlWhereCondition の SoqlNotCondition サブクラスも使用して、完全な条件を表します。次の例では、Java の `instanceof` 演算子を使用して、`not like` 演算子が指定されているかどうかを判断します。

```
if (resultSoqlWhereCondition instanceof SoqlNotCondition) {
  // condition is really NOT condition
  // if operator is "like", this condition really means "not like"
  ...
}
```

API バージョン 41.0 以降では、SoqlNotCondition と `like` 演算子の代わりに、`notLike` 演算子が使用されます。`notLike` 演算子は、リストビューでのみ使用できます。他の Salesforce 機能で使用される SOQL クエリでは使用できません。

describeTabs()

ログインユーザーが利用可能な Salesforce Classic 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。

ページ上部の Lightning プラットフォームアプリケーションメニューに表示される、ログインユーザーが使用できる Salesforce Classic 標準アプリケーションおよびカスタムアプリケーションに関する情報を返します。アプリケーションは、アプリケーション機能を提供するために 1 つの単位として機能するタブのセットです。たとえば、標準 Salesforce アプリケーションとして「セールス」と「サービス」があります。

構文

```
describeTabSetResult []  = connection.describeTabs();
```

使用方法

ログインユーザーがアクセスできる Salesforce Classic 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取得するのに、`describeTabs()` コールを使用します。`describeTabs()` コールは、アプリケーションを別のユーザー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ーインターフェース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。

アプリケーションに対するコールでは、アプリケーション名、ロゴの URL のほか、それがユーザーに対して現在指定されているアプリケーションかどうか、アプリケーションにはどのようなタブが含まれているか、といった情報が返されます。

重要: `describeTabs()` コールは、ログインユーザーの Salesforce ユーザーインターフェースに表示されるタブに関する情報のみを返します。[すべてのタブ] (+) タブをクリックして、Salesforce ユーザーインターフェースの一部のタブを非表示にする場合、ユーザーが非表示にしたこれらのタブは、`describeTabs()` で返されるタブのセットに含まれません。

ログインユーザーが使用できるすべてのタブに関する情報を取得するのに、`describeAllTabs()` コールを使用します。

各タブに対するコールでは、タブ名、タブに表示されるメインの sObject、タブを参照する URL のほか、そのタブがカスタムタブかどうかといった情報が返されます。[すべてのタブ] タブおよび Lightning ページのタブは、タブリストには含まれません。

サンプルコード — Java

このサンプルでは、`describeTabs()` をコールします。これは、タブセットの結果の配列を返します。次に、Salesforce Classic アプリケーションを表すタブセットの結果ごとに、プロパティの一部を取得し、このアプリケーションのすべてのタブを取得します。このサンプルでは、取得したすべてのプロパティをコンソールに書き込みます。

```
public void describeTabsSample() {
  try {
    // Describe tabs
    DescribeTabSetResult[] dtsrs = connection.describeTabs();
    System.out.println("There are " + dtsrs.length +
        " tab sets defined.");

    // For each tab set describe result, get some properties
    for (int i = 0; i < dtsrs.length; i++) {
      System.out.println("Tab Set " + (i + 1) + ":");
      DescribeTabSetResult dtsr = dtsrs[i];
      System.out.println("Label: " + dtsr.getLabel());
      System.out.println("\tLogo URL: " + dtsr.getLogoUrl());
      System.out.println("\tTab selected: " +
          dtsr.isSelected());

      // Describe the tabs for the tab set
      DescribeTab[] tabs = dtsr.getTabs();
      System.out.println("\tTabs defined: " + tabs.length);

      // Iterate through the returned tabs
      for (int j = 0; j < tabs.length; j++) {
        DescribeTab tab = tabs[j];
        System.out.println("\tTab " + (j + 1) + ":");
        System.out.println("\t\tName: " +
            tab.getSobjectName());
        System.out.println("\t\tLabel: " + tab.getLabel());
        System.out.println("\t\tURL: " + tab.getUrl());
        DescribeColor[] tabColors = tab.getColors();
        // Iterate through tab colors as needed
        DescribeIcon[] tabIcons = tab.getIcons();
        // Iterate through tab icons as needed
      }
    }
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

サンプルコード — C#

このサンプルでは、`describeTabs()` をコールします。これは、タブセットの結果の配列を返します。次に、Salesforce Classic アプリケーションを表すタブセットの結果ごとに、プロパティの一部を取得し、このアプリケーションのすべてのタブを取得します。このサンプルでは、取得したすべてのプロパティをコンソールに書き込みます。

```
public void describeTabsSample() {
    try {
      // Describe tabs
      DescribeTabSetResult[] dtsrs = binding.describeTabs();
      Console.WriteLine("There are " + dtsrs.Length +
          " tab sets defined.");

      // For each tab set describe result, get some properties
      for (int i = 0; i < dtsrs.Length; i++) {
          Console.WriteLine("Tab Set " + (i + 1) + ":");
          DescribeTabSetResult dtsr = dtsrs[i];
          Console.WriteLine("Label: " + dtsr.label);
          Console.WriteLine("\tLogo URL: " + dtsr.logoUrl);
          Console.WriteLine("\tTab selected: " +
              dtsr.selected);

          // Describe the tabs for the tab set
          DescribeTab[] tabs = dtsr.tabs;
          Console.WriteLine("\tTabs defined: " + tabs.Length);

          // Iterate through the returned tabs
          for (int j = 0; j < tabs.Length; j++) {
              DescribeTab tab = tabs[j];
              Console.WriteLine("\tTab " + (j + 1) + ":");
              Console.WriteLine("\t\tName: " +
                  tab.sobjectName);
              Console.WriteLine("\t\tLabel: " + tab.label);
              Console.WriteLine("\t\tURL: " + tab.url);
              DescribeColor[] tabColors = tab.colors;
              // Iterate through tab colors as needed
              DescribeIcon[] tabIcons = tab.icons;
              // Iterate through tab icons as needed
          }
      }
    } catch (SoapException e) {
        Console.WriteLine("An unexpected error has occurred: " +
                    e.Message + "\n" + e.StackTrace);
    }
}
```

引数

なし。

応答

describeTabSetResult、 DescribeTab

関連トピック:

- API コールの基礎
- Partner WSDL の使用
- DescribeTab
- DescribeTabSetResult

DescribeTabSetResult

describeTabs() コールは、DescribeTabSetResult オブジェクトの配列を返します。

DescribeTabSetResult 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
description	string	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の説明。
label	string	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の表示ラベル。この値は、Salesforce ユーザー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 ヘルプを参照してください。
logoUrl	string	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。
namespace	string	カスタムアプリケーションを使用していて、カスタムアプリケーション内のタブのセットが管理パッケージの一部としてインストールされている場合、Developer Edition 組織が管理パッケージの公開を許可したときにパッケージ作成者が設定した値が、この属性の値となります。この属性は、AppExchange パッケージの要素を識別します。
selected	boolean	true の場合、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ユーザーの現在の選択されたアプリケーションとなります。
tabs	DescribeTab	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で表示されたタブの配列。

DescribeColor

DescribeColor には、タブの色メタデータ情報が含まれます。

DescribeColor には、タブの色メタデータ情報が含まれます。describeTabs() コールは、DescribeTabSetResult の値の配列を返します。各 DescribeTabSetResult には DescribeTab の値の配列が含まれ、各 DescribeTab には DescribeColor の値の配列が含まれます。

各 DescribeColor は Salesforce ユーザーインターフェースのテーマに関連付けられています。テーマについての詳細は、『Visualforce 開発者ガイド』の「ユーザーに表示する Salesforce スタイルの識別」を参照してください。

色情報は、`describeTheme()` コールおよび `describeGlobalTheme()` コールからも取得できます。これらのコールは、テーマのアイコンと色を使用できる組織内の各オブジェクトで使用される色の情報を返します。

名前	型	説明
`color`	string	Web 色の RGB 形式で示される色 (00FF00 など)。
`context`	string	タブでその色がメインの色 (primary) であるかどうかを決定する色のコンテキスト。
`theme`	string	関連付けられたテーマ。可能な値には、次のものがあります。 • *`theme2`* — Spring '10 より前に使用されていた「Salesforce Classic 2005 ユーザーインターフェースのテーマ」という名前のテーマ • *`theme3`* — Spring '10 で導入された「Salesforce Classic 2010 ユーザーインターフェースのテーマ」という名前のテーマ • *`theme4`* — Winter '14 (Lightning Experience では Winter '16) に導入されたモバイルタッチスクリーンバージョンの Salesforce テーマ • *`custom`* — カスタムアイコンに関連付けられたテーマ

DescribeIcon

DescribeIcon にはタブアイコンのメタデータ情報が含まれます。

`describeTabs()` コールは、DescribeTabSetResult の値の配列を返します。各 DescribeTabSetResult には、DescribeTab 値の配列が含まれ、各 DescribeTab には DescribeIcon 値の配列が含まれます。

アイコン情報は、`describeTheme()` コールと `describeGlobalTheme()` コールを介して取得することもできます。これらのコールは、テーマのアイコンと色を使用できる組織内の各オブジェクトに使用されるアイコンに関する情報を返します。

名前	型	説明
`contentType`	string	タブアイコンのコンテンツタイプは、「image/png」などです。
`height`	int	タブアイコンの高さ (ピクセル単位)。アイコンのコンテンツタイプが SVG タイプである場合、高さと幅の値は使用されません。

名前	型	説明
theme	string	関連付けられたテーマ。可能な値には、次のものがあります。 • *theme2* — Spring '10 より前に使用されていた「Salesforce Classic 2005 ユーザーインターフェースのテーマ」という名前のテーマ • *theme3* — Spring '10 で導入された「Salesforce Classic 2010 ユーザーインターフェースのテーマ」という名前のテーマ • *theme4* — Winter '14 (Lightning Experience では Winter '16) に導入されたモバイルタッチスクリーンバージョンの Salesforce テーマ • *custom* — カスタムアイコンに関連付けられたテーマ
url	string	このアイコンの完全修飾 URL。
width	string	タブアイコンの幅(ピクセル単位)。アイコンのコンテンツタイプが SVG タイプである場合、高さと幅の値は使用されません。

DescribeTab

describeTabs() コールは、DescribeTab をプロパティとして持つ DescribeTabSetResult オブジェクトの配列を返します。

DescribeTab 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
colors	DescribeColor[]	タブに使用される色情報の配列。この項目は、API バージョン 29.0 以降で使用できます。
custom	boolean	この項目がカスタムタブの場合は true、標準タブの場合は false。
iconUrl	string	タブのメインの 32x32 ピクセルのアイコンの URL。アイコンは、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。このアイコン URL は、Salesforce Classic 2010 ユーザーインターフェースのテーマに使用される 32×32 アイコンに対応します。
icons	DescribeIcon[]	タブに使用されるアイコン情報の配列。この項目は、API バージョン 29.0 以降で使用できます。
label	string	このタブに表示されるラベル。

名前	型	説明
`miniIconUrl`	string	タブを表す 16x16 ピクセルのアイコンの URL。このアイコンは、関連リストなどに表示されます。このアイコン URL は、Spring '10 で導入された現在の Salesforce テーマに使用される 16×16 アイコンに対応します。
`name`	string	タブの API 名。
`sobjectName`	string	このタブにメインで表示される sObject の名前 (特定の sObject を表示するタブ)。オブジェクトのリストは、「標準オブジェクト」を参照してください。
`url`	string	このタブを表示するための完全修飾 URL。

関連トピック:

- DescribeColor
- DescribeIcon

describeTheme()

現在のログインユーザーが使用できるテーマの情報を返します。

構文

```
DescribeThemeResult = connection.describeTheme(string sObjectType[]);
```

使用方法

`describeTheme()` を使用して、指定されたオブジェクト配列の現在のテーマ情報を取得します。テーマ情報は、特定のテーマに使用される、Salesforce 内のオブジェクトの色およびアイコンで構成されています。たとえば、Merchandise__c オブジェクトは通常の Salesforce アプリケーションテーマには「computer32」アイコンと、赤色の主タブ色を使用し、モバイルタッチスクリーンバージョンの Salesforce には別のセットの色とアイコンを使用できます。

オブジェクト配列ではなく、`null` を渡すと、`describeTheme()` は、テーマの色とアイコンを使用する組織内のすべてのオブジェクトのテーマ情報を返します。

組織のデータのテーマ情報を取得するには、条件を満たすアクセス権限でクライアントアプリケーションにログインする必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

`describeTheme()` は、API バージョン 29.0 以降で使用できます。

サンプル

この Java のサンプルでは、`describeTheme()` をコールして取引先と取引先責任者のテーマ情報を取得した後、取得したテーマ情報を反復処理します。

```
public static void describeThemeExample() {
    try {
        // Get current themes
        DescribeTheme themeResult = connection.describeTheme(
                                        new String[] { "Account", "Contact" });
        DescribeThemeItem[] themeItems = themeResult.getThemeItems();
        for (int i = 0; i < themeItems.length; i++) {
            DescribeThemeItem themeItem = themeItems[i];
            System.out.println("Theme information for object " + themeItem.getName());
            // Get color and icon info for each themeItem
            DescribeColor colors[] = themeItem.getColors();
            System.out.println("    Number of colors: " + colors.length);
            int k;
            for (k = 0; k < colors.length; k++) {
                DescribeColor color = colors[k];
                System.out.println("    For Color #" + k + ":");
                System.out.println("      Web RGB Color: " + color.getColor());
                System.out.println("      Context: " + color.getContext());
                System.out.println("      Theme: " + color.getTheme());
            }
            DescribeIcon icons[] = themeItem.getIcons();
            System.out.println("    Number of icons: " + icons.length);
            for (k = 0; k < icons.length; k++) {
                DescribeIcon icon = icons[k];
                System.out.println("    For Icon #" + k + ":");
                System.out.println("      ContentType: " + icon.getContentType());
                System.out.println("      Height: " + icon.getHeight());
                System.out.println("      Theme: " + icon.getTheme());
                System.out.println("      URL: " + icon.getUrl());
                System.out.println("      Width: " + icon.getWidth());
            }
        }
    } catch (ConnectionException ce) {
        ce.printStackTrace();
    }
}
```

応答

DescribeThemeResult

障害

`UnexpectedErrorFault`

関連トピック:

- DescribeThemeResult
- DescribeThemeItem
- DescribeColor
- DescribeIcon

DescribeThemeResult

DescribeThemeResult オブジェクトを返します。

`describeTheme()` コールと `describeGlobalTheme()` コールは、DescribeThemeItem 値の配列を含む DescribeThemeResult を返します。

名前	型	説明
`themes`	DescribeThemeItem[]	テーマの配列。テーマ情報は、テーマのアイコンと色を使用できる組織内のオブジェクトごとに提供されます。

DescribeThemeItem

DescribeThemeItem 値の配列を含む DescribeThemeResult オブジェクトを返します。

`describeTheme()` コールと `describeGlobalTheme()` コールは、DescribeThemeItem 値の配列を含む DescribeThemeResult を返します。各 DescribeThemeItem には、テーマに使用される色とアイコンの配列と、テーマ情報が適用されるオブジェクトの名前が含まれています。

名前	型	説明
`colors`	DescribeColor[]	色の配列。
`icons`	DescribeIcon[]	アイコンの配列。
`name`	string	テーマの色とアイコンが関連付けられるオブジェクトの名前。

第 14 章　ユーティリティ API コール

ここでは、クライアントアプリケーションがシステムのタイムスタンプ、ユーザー情報を取得し、ユーザーのパスワードを変更するために呼び出すことができる API コールについて説明しています。

メモ: Apex 関連コールのリストは「Apex 関連コール」、基本となる API コールのリストは「基本となる API コール」、記述用の API コールのリストは「記述用の API コール」を参照してください。

次は、このトピックで説明されているユーティリティ API コールの一覧表です。

タスク/コール	説明
`getServerTimestamp()`	API の現在のシステムタイムスタンプを取得します。
`changeOwnPassword()`	ユーザーが、自分自身のパスワードを変更できるようになります。
`getUserInfo()`	現在のセッションに関連するユーザーの個人情報を取得します。
`match()`	入力の MatchOptions で指定した一致ルールを使用して、リード間の照合の入力として提供される sObject を評価します。このコールは、取引先でリードに適用する標準的な一致ルールとのみ使用できます。
`renderEmailTemplate()`	メールテンプレートのテキスト本文の差し込み項目を、多態的な項目の場合も含め、Salesforce レコードの値で置き換えます。メールテンプレートの本文と対応する `whoId` 値および `whatId` 値は引数で指定します。
`resetPassword()`	ユーザーのパスワードをシステムで生成された値に変更します。
`sendEmail()`	メールメッセージをただちに送信します。
`sendEmailMessage()`	最大 10 件のドラフトメールメッセージをただちに送信します。
`setPassword()`	指定されたユーザーのパスワードを指定された値に設定します。

サンプル

このセクションのサンプルは、Enterprise WSDL ファイルに基づいてます。これらのサンプルでは、WSDL ファイルがインポート済みであり、接続が作成済みであることを前提としています。これを実行する方法の詳細については、「クイックスタート」のチュートリアルを参照してください。

changeOwnPassword()

ユーザーが自分のパスワードを古い値からユーザーが指定する新しい値に変更できるようにします。

構文

```
ChangeOwnPasswordResult changeOwnPasswordResult = connection.changeOwnPassword(string
oldPassword, string newPassword);
```

使用方法

ユーザーが自分のパスワードをユーザーが指定する値に変更できるようにする場合は、changeOwnPassword() を使用します。たとえば、クライアントアプリケーションがユーザーに異なるパスワードを指定するように促した後、ユーザーのパスワード変更のために changeOwnPassword() を呼び出します。別のユーザーのパスワードをあなたが指定する値に変更する場合は、setPassword() を使用します。API によって生成されたランダム値に対象ユーザーのパスワードをリセットする場合は、resetPassword() を使用します。

サンプルコード — Java

このサンプルでは、古いパスワードパラメーターと新しいパスワードパラメーターを受け取り、これらを使用してユーザーの新しいパスワードを設定する changeOwnPassword() コールを実行します。

```
public void doChangeOwnPassword(String oldPasswd, String newPasswd) {
   try {
     ChangeOwnPasswordResult result = connection.changeOwnPassword(oldPasswd, newPasswd);

      System.out.println("Your password was changed to "
            + newPasswd);
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、古いパスワードパラメーターと新しいパスワードパラメーターを受け取り、これらを使用してユーザーの新しいパスワードを設定する changeOwnPassword() コールを実行します。

```
public void doChangeOwnPassword(String oldPasswd, String newPasswd)
{
   try
   {
      ChangeOwnPasswordResult result = binding.changeOwnPassword(oldPasswd, newPasswd);
      Console.WriteLine("Your password was changed to "
            + newPasswd);
   }
   catch (SoapException e)
   {
```

```
        Console.WriteLine("An unexpected error has occurred: " +
                          e.Message + "\n" + e.StackTrace);
    }
}
```

引数

名前	型	説明
oldPassword	string	置き換えられる、ユーザーの過去のパスワード。
newPassword	string	ユーザーの新しいパスワード。

応答

ChangeOwnPasswordResult

障害

InvalidOldPasswordFault

InvalidNewPasswordFault

UnexpectedErrorFault

関連トピック:

- resetPassword()
- ユーティリティ API コール
- setPassword()

getServerTimestamp()

API からシステムの現在のタイムスタンプ (協定世界時 (UTC)) を取得します。

構文

```
GetServerTimestampResult timestamp = connection.getServerTimestamp();
```

使用方法

API からシステムの現在のタイムスタンプを取得するには、getServerTimestamp() を使用します。たとえば、タイミングまたはデータ同期のために正確なタイムスタンプが必要な場合に実行します。オブジェクトに対して create() または update() を実行する場合、API はシステムのタイムスタンプを使用してオブジェクトの CreatedDate および LastModifiedDate 項目をそれぞれ更新します。

getServerTimestamp() コールはタイムスタンプを常に協定世界時 (UTC) で返します。ただし、ローカルシステムが、結果をシステムが属するタイムゾーンの設定に基づいて自動的に表示してしまいます。

メモ: 時間データの処理方法は、開発ツールごとに異なります。開発ツールによってはローカル時間を表示するものも、協定世界時 (UTC) を表示するものもあります。開発ツールごとの時間の処理方法はツールのドキュメントを参照してください。

サンプルコード — Java

このサンプルでは、サーバー時間を取得し、ユーザーのローカルタイムゾーンでコンソールにそのサーバー時間を書き込みます。

```
public void doGetServerTimestamp() {
   try {
      GetServerTimestampResult result = connection.getServerTimestamp();
      Calendar serverTime = result.getTimestamp();
      System.out.println("Server time is: "
            + serverTime.getTime().toString());
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、サーバー時間を取得し、ユーザーのローカルタイムゾーンでコンソールにそのサーバー時間を書き込みます。

```
public void doGetServerTimestamp()
{
   try
   {
      GetServerTimestampResult result =
            binding.getServerTimestamp();
      DateTime serverTime = result.timestamp;
      Console.WriteLine("Server time is: " +
            serverTime.ToLocalTime().ToString());
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                        e.Message + "\n" + e.StackTrace);
   }
}
```

引数

なし。

応答

getServerTimestampResult

障害

UnexpectedErrorFault

関連トピック:

ユーティリティ API コール

getServerTimestampResult

GetServerTimestampResult オブジェクトを返します。

getServerTimestamp() コールが返す GetServerTimestampResult 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
timestamp	dateTime	getServerTimestamp() コールが実行されたときの API のシステムタイムスタンプ。

getUserInfo()

現在のセッションに関連するユーザーの個人情報を取得します。

構文

```
getUserInfoResult result = connection.getUserInfo();
```

使用方法

現在ログインユーザーの個人情報を取得するには、getUserInfo() を使用します。この便利な API コールは、クライアントアプリケーションで表示や通貨計算などを行うための一般的なプロファイル情報を取得し、まとめます。

getUserInfo() は、クライアントアプリケーションがログインしたユーザー名にのみ適用されます。getUserInfoResult オブジェクトでは見つからないその他の個人情報を取得するには、User オブジェクトに対して retrieve() を実行し、このコールで返された userID を渡すことができます。他のユーザーの個人情報を取得するには、retrieve() をコールするか (ユーザー ID がわかっている場合)、User オブジェクトに対して query() コールを実行します。

サンプルコード — Java

このサンプルでは、`getUserInfo()` をコールし、現在のユーザーに関する情報をコンソールに書き込みます。

```
public void doGetUserInfo() {
   try {
      GetUserInfoResult result = connection.getUserInfo();
      System.out.println("\nUser Information");
      System.out.println("\tFull name: " + result.getUserFullName());
      System.out.println("\tEmail: " + result.getUserEmail());
      System.out.println("\tLocale: " + result.getUserLocale());
      System.out.println("\tTimezone: " + result.getUserTimeZone());
      System.out.println("\tCurrency symbol: " + result.getCurrencySymbol());
      System.out.println("\tOrganization is multi-currency: " +
            result.isOrganizationMultiCurrency());
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、`getUserInfo()` をコールし、現在のユーザーに関する情報をコンソールに書き込みます。

```
public void doGetUserInfo()
{
   try
   {
      GetUserInfoResult result = binding.getUserInfo();
      Console.WriteLine("\nUser Information");
      Console.WriteLine("\tFull name: " + result.userFullName);
      Console.WriteLine("\tEmail: " + result.userEmail);
      Console.WriteLine("\tLocale: " + result.userLocale);
      Console.WriteLine("\tTimezone: " + result.userTimeZone);
      Console.WriteLine("\tCurrency symbol: " + result.currencySymbol);
      Console.WriteLine("\tOrganization is multi-currency: " +
            result.organizationMultiCurrency);
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                        e.Message + "\n" + e.StackTrace);
   }
}
```

引数

なし。

応答

getUserInfoResult

障害

UnexpectedErrorFault

関連トピック:

ユーティリティ API コール

getUserInfoResult

getUserInfo() コールは GetUserInfoResult オブジェクトを返します。

「getUserInfo()」を参照してください。

名前	型	説明
accessibilityMode	boolean	API バージョン 7.0 以降で使用できます。視覚障害者のためのユーザーインターフェースの変更が可能かどうかを示します (可能な場合は true、不可能な場合は false)。変更により、JAWS などの画面リーダーを簡単に利用できるようになります。
chatterExternal	boolean	ユーザーに関連付けられた Profile に割り当てられたユーザーライセンスの種別。ユーザーが組織内のユーザーか、外部のユーザーかを示します。API 40.0 以降で使用できます。
currencySymbol	string	通貨の値を表示するために使用する通貨記号。organizationMultiCurrency が false の場合のみ有効です。
organizationId	ID	組織の ID。サードパーティ製のツールが Salesforce の各組織を一意に識別できます。請求情報や組織全体の設定情報を取得する際に役立ちます。
organizationMultiCurrency	boolean	ユーザーの組織がマルチ通貨を使用するか (true)、否か (false) を示します。
organizationName	string	ユーザーの組織または企業の名前。
orgDefaultCurrencyIsoCode	string	デフォルトの通貨 ISO コード。organizationMultiCurrency が false の場合のみ有効です。ログインユーザーが通貨 ISO コードのあるオブジェクトを作成する場合、create() コールで明示的に指定しなければ API はこの通貨 ISO コードを使用します。
profileID	ID	現在ユーザーに割り当てられているロールに関連付けられたプロファイルの ID。
roleID	ID	現在ユーザーに割り当てられているロールの ID。

名前	型	説明
sessionSecondsValid	int	最後の更新時刻から開始し、セッションが期限切れになるまでの秒数。 API バージョン 21.0 以降で利用できます。
userDefaultCurrencyIsoCode	string	デフォルトの通貨 ISO コード。organizationMultiCurrency が true の場合のみ有効です。ログインユーザーが通貨 ISO コードのあるオブジェクトを作成する場合、create() コールで明示的に指定しなければ API はこの通貨 ISO コードを使用します。
userEmail	string	ユーザーのメールアドレス。
userFullName	string	ユーザーの氏名。
userID	ID	ユーザー ID。
userLanguage	string	ユーザーの言語で、アプリケーションに表示されるラベルの言語を制御します。文字列の長さは、2-5 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。
userLocale	string	ユーザーのロケールで、日付の形式や通貨記号の選択を制御します。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。
userName	string	ユーザーのログイン名。
userTimeZone	string	ユーザーのタイムゾーン。
userType	string	ユーザーに関連付けられた Profile に割り当てられたユーザーライセンスの種別。
userUiSkin	string	API バージョン 7.0 以降で利用できます。可能な値は次のとおりです。 • theme3 — ユーザーが Salesforce Classic 2010 ユーザーインターフェースのテーマ (*Aloha* インターフェース) を使用している場合

名前	型	説明
		• `theme2` — ユーザーが Salesforce Classic 2005 ユーザーインターフェースのテーマを使用している場合 • `theme1` — ユーザーが最も古いユーザーインターフェースのテーマ (廃止) を使用している場合 オンラインアプリケーションでは、このデザイン設定は、[設定] から、[クイック検索] ボックスに「ユーザーインターフェース」と入力し、[ユーザーインターフェース] を選択して設定できます。「ユーザーインターフェースのテーマ」を参照してください。

match()

入力の MatchOptions で指定した一致ルールを使用して、リード間の照合の入力として提供される sObject を評価します。このコールは、取引先でリードに適用する標準的な一致ルールとのみ使用できます。

これは、**Professional** Edition、**Enterprise** Edition、**Performance** Edition、および **Pardot Pro** Edition または **Pardot Ultimate** Edition を使用する **Unlimited** Edition で、API バージョン 42.0 以降で使用できます。

構文

```
MatchResult[] callResults = connection.match(SObject[] inputSObjectArray, MatchOptions
   matchOptions);
```

引数

名前	型	説明
inputSObjectArray	sObject の配列	一致を評価する sObject のリスト。
matchOptions	MatchOptions	マッチ処理中に使用されるマッチングルールなどのオプション。

応答

MatchResult

MatchOptions

マッチ処理で使用される種別を表します。マッチ処理中に使用されるオプションを説明します。この種別は、取引先でリードに適用する標準的な一致ルールとのみ使用できます。

この種別は、Professional Edition、**Enterprise** Edition、**Performance** Edition、および **Pardot Pro** Edition または **Pardot Ultimate** Edition を使用する **Unlimited** Edition で、API バージョン 42.0 以降で使用できます。

項目

項目	詳細	説明
`Fields`	string	指定されたルールによるマッチ操作で使用されるカンマで区切られた 1 つ以上の項目の

項目	詳細	説明
		リスト
`MatchEngine`	string	マッチ処理で使用されるマッチエンジン完全一致またはあいまい
`Rule`	string	マッチ処理で使用される

項目	詳細	説明
		の R 参照名
SobjectType	string	一致を評価するオブジェクトの種別

renderEmailTemplate()

メールテンプレートのテキスト本文の差し込み項目を、多態的な項目の場合も含め、Salesforce レコードの値で置き換えます。メールテンプレートの本文と対応する `whoId` 値および `whatId` 値は引数で指定します。

構文

```
RenderEmailTemplateResult = connection.renderEmailTemplate(RenderEmailTemplateRequest[]
renderRequests);
```

使用方法

`renderEmailTemplate()` コールは、カスタムテンプレートを使用したメールを sendEmail() コールで送信する場合の差し込み項目の表示に相当します。

`renderEmailTemplate()` コールの配列引数には最大 10 個の RenderEmailTemplateRequest 要素を指定でき、各 RenderEmailTemplateRequest には最大 10 個のテンプレート本文を含めることができます。各要求は配列内の他の要求から独立しており、1 つの要求でエラーが発生しても他の要求には影響しません。同様に、1 つのテンプレート本文でエラーが発生しても、同じ要求内の他のテキスト本文ではエラーになりません。

`renderEmailTemplate()` コールでは、差し込み項目が RenderEmailTemplateRequest の `whatId` または `whoId` の値に置き換えられます。

RenderEmailTemplateRequest の `whatId` および `whoId` 項目の値は、要求ごとに検証されます。`whatId` が有効な What ID (人以外のオブジェクト) を参照していないか、`whoId` が有効な Who ID (人のオブジェクト) を参照していないと、要求に対してエラーが設定されます。

- 差し込み項目が人以外のオブジェクトを参照している場合は、`whatId` の対応する値に置き換えられます。たとえば、差し込み項目が取引先または商談を参照している場合、`whatId` の値に置き換えられます。
- 差し込み項目が人のオブジェクトを参照している場合は、`whoId` の対応する値に置き換えられます。たとえば、差し込み項目が取引先責任者、リード、またはユーザーを参照している場合、`whoId` の値に置き換えられます。

サンプルコード — Java

次のサンプルでは、`renderEmailTemplate()` コールで、すべての取引先責任者差し込み項目を、指定された *`whoId`* 引数の値で置き換えます。同様に、商談差し込み項目 (`{!Opportunity.Name}`) を、指定された *`whatId`* の値で置き換えます。このサンプルの 2 番目のテンプレート本文は不正な差し込み項目 (`{!Contact.SNARF}`) であるため、2 番目のテンプレートはエラーになります。ただし、要求を表示するテンプレート全体は成功します。

```
public void renderTemplates(String whoId, String whatId)
        throws ConnectionException, RemoteException, MalformedURLException {
    // Array of three template bodies.
    // The second template body generates an error.
    final String[] TEMPLATE_BODIES = new String[] {
           "This is a good template body {!Contact.Name}",
           "This is a bad template body {!Opportunity.Name} {!Contact.SNARF} ",
           "This is another good template body {!Contact.Name}"};

    // Create request and add template bodies, whatId, and whoId.
    RenderEmailTemplateRequest req = new RenderEmailTemplateRequest();
    req.setTemplateBodies(TEMPLATE_BODIES);
    req.setWhatId(whatId);
    req.setWhoId(whoId);
    // An array of results is returned, one for each request.
    // We only have one request.
    RenderEmailTemplateResult[] results =  connection.renderEmailTemplate(
            new RenderEmailTemplateRequest[] { req });
    if (results != null) {
        // Check results for our one and only request.
        // Check request was processed successfully, and if not, print the errors.
        if (!results[0].isSuccess()) {
            System.out.println(
            "The following errors were encountered while rendering email templates:");
            for (Error err : results[0].getErrors()) {
                System.out.println(err.getMessage());
            }
        } else {
            // Check results for each body template and print merged body
            RenderEmailTemplateBodyResult[] bodyResults = results[0].getBodyResults();
            for( Integer i=0;i<bodyResults.length;i++) {
```

```
                RenderEmailTemplateBodyResult result = bodyResults[i];
                if (result.isSuccess()) {
                    System.out.println("\nMerged body: \n" + result.getMergedBody());
                } else {
                    System.out.println("\nErrors were found for body[" + i + "]: ");
                    for (RenderEmailTemplateError err : result.getErrors()) {
                        System.out.println(err.getMessage() + " - Field name: "
                            + err.getFieldName());
                    }
                }
            }
        }
    }
}
```

最初の引数 (*whoId*) に有効な取引先責任者 ID を指定し、2 番目の引数 (*whatId*) に `null` を指定して、このサンプルを実行するとします。2 番目のテンプレートでは、不正な差し込み項目に対して、1 つのエラーが設定されています。応答は次のようになります。

```
Merged body:
This is a good template body Howard Jones

Errors were found for body[1]:
Field Contact.SNARF does not exist. Check spelling. - Field name: Contact.SNARF


Merged body:
This is another good template body Howard Jones
```

RenderEmailTemplateRequest

名前	型	説明
`templateBodies`	string[]	`{!Account.Phone}` や `{!Contact.Name}` などの差し込み項目を含めることができるテキスト本文の配列。
`whatId`	reference	取引先、商談、キャンペーン、ケース、カスタムオブジェクトなどの人以外のオブジェクトを参照します。`whatId` は多態的です。つまり、ケースや商談など、複数の種別のオブジェクトを参照できる ID であるという意味です。
`whoId`	reference	リード、取引先責任者、ユーザーなど、人のオブジェクトを参照します。`whoId` は多態的です。つまり、複数の種別のオブジェクトを参照できる ID であるという意味です。

障害

`renderEmailTemplate()` から、次のいずれかの API 状況コードが返される場合があります。

`EMAIL_TEMPLATE_FORMULA_ERROR`

EMAIL_TEMPLATE_MERGEFIELD_ACCESS_ERROR (ページ 48)

EMAIL_TEMPLATE_MERGEFIELD_ERROR

EMAIL_TEMPLATE_MERGEFIELD_VALUE_ERROR

EMAIL_TEMPLATE_PROCESSING_ERROR

RenderEmailTemplateResult

表示されるメールテンプレートの個々の結果を含め、renderEmailTemplate() コールで処理される要求の状況およびエラーに関する情報が含まれます。

名前	型	説明
bodyResults	RenderEmailTemplateBodyResult[]	テンプレートの差し込み本文テキストを含め、renderEmailTemplate() によって要求で処理されるテンプレート本文ごとに状況およびエラーに関する情報が含まれます。
errors	Error[]	renderEmailTemplate() で要求が表示されるときに発生した 1 つ以上のエラーが含まれます。
success	boolean	要求が正常に処理されたか (true)、否か (false) を示します。

RenderEmailTemplateBodyResult

テンプレートの差し込み本文テキストを含め、renderEmailTemplate() によって要求で処理されるテンプレート本文ごとに状況およびエラーに関する情報が含まれます。

名前	型	説明
errors	RenderEmailTemplateError[]	renderEmailTemplate() で処理されたテンプレート本文に関連付けられた 1 つ以上のエラーが含まれます。
mergedBody	string	Salesforce オブジェクトの対応する値と置き換えられた差し込み項目を含むテンプレート本文のテキスト。要求の whatId および whoId 項目は、使用する Salesforce オブジェクトを参照します。 mergedBody 項目は、テンプレートの表示に成功した場合 (success が true) にのみ入力されます。success が false の場合、mergedBody は null になります。
success	boolean	テンプレート本文が正常に表示されたか (true)、否か (false) を示します。

RenderEmailTemplateError

renderEmailTemplate() でテンプレート本文を処理するときに発生したエラー。

名前	型	説明
fieldName	string[]	エラー条件に影響を与えた差し込み項目。
message	string	エラーメッセージテキスト。
offset	int	エラーの原因となった差し込み項目のテンプレート本文テキストのオフセット。オフセットは、本文テキストの先頭からの文字数として計算されます。コンテキスト情報が不十分なためオフセットを計算できない場合は、-1 になります。
statusCode	StatusCode	エラーを特徴付けるコード。statusCode の完全な一覧は組織の WSDL ファイルに記述されています (「組織の WSDL ファイルの生成」を参照してください)。

resetPassword()

一時的なシステム生成値にユーザーパスワードを変更します。

構文

```
string password = connection.resetPassword(ID userID);
```

使用方法

resetPassword() は、API で User または SelfServiceUser のパスワード変更し、ランダムな文字と数字から成るシステムが生成したパスワードを取得する場合に使用します。特定の値にパスワードを設定する場合は、setPassword() を使用します。

クライアントアプリケーションは、そのユーザーのパスワードを変更するために必要なアクセス権限でログインされている必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

ID についての詳細は、「ID データ型」を参照してください。

サンプルコード — Java

このサンプルでは、userId パラメーターで指定されたユーザーのパスワードをリセットします。この ID を使用して resetPassword() をコールし、コールの結果から仮のパスワードを取得します。この仮のパスワードをコンソールに書き込み、そのパスワードを返します。

```
public String doResetPassword(String userId) {
   String result = "";
   try {
```

```
      ResetPasswordResult rpr = connection.resetPassword(userId);
      result = rpr.getPassword();
      System.out.println("The temporary password for user ID " + userId
            + " is " + result);
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
   return result;
}
```

サンプルコード — C#

このサンプルでは、userId パラメーターで指定されたユーザーのパスワードをリセットします。このIDを使用して resetPassword() をコールし、コールの結果から仮のパスワードを取得します。この仮のパスワードをコンソールに書き込み、そのパスワードを返します。

```
public String doResetPassword(String userId)
{
   String result = "";
   try
   {
      ResetPasswordResult rpr = binding.resetPassword(userId);
      result = rpr.password;
      Console.WriteLine("The temporary password for user ID " + userId + " is " +
            result);
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                        e.Message + "\n" + e.StackTrace);
   }
   return result;
}
```

引数

名前	型	説明
userID	ID	パスワードをリセットする User または SelfServiceUser の ID。ID についての詳細は、「ID データ型」を参照してください。

応答

名前	型	説明
password	string	API によって生成された新しいパスワード。このパスワードでユーザーがログインすると、そのユーザーは新しいパスワードを提供するように求めら

名前	型	説明
		れます。このパスワードは一時的なもので、ユーザーが新しいパスワードを設定したら再利用することができないパスワードです。

障害

`InvalidIdFault`

`UnexpectedErrorFault`

関連トピック:

ユーティリティ API コール

sendEmail()

メールメッセージをただちに送信します。

構文

単一のメールメッセージでは、次のようになります。

```
SendEmailResult = connection.sendEmail(SingleEmailMessage emails[]);
```

一括メールメッセージに関して、次のようになります。

```
SendEmailResult = connection.sendEmail(MassEmailMessage emails[]);
```

使用方法

このコールは、Lightning プラットフォーム AppExchange アプリケーション、カスタムアプリケーションなどの、Salesforce 以外のアプリケーションで、メールの個別送信や一括送信を行う場合に使用します。メールでは標準的なメール属性 (件名行、BCC など) を含めることができ、Salesforce のメールテンプレートを使用することが可能です。平文テキストのほか HTML 形式もサポートされています。HTML メールの状況は、Salesforce を使って追跡できます。送信日、メールが最初に開かれた日、最後に開かれた日、開かれた回数などを確認できます。(詳細は、Salesforce ヘルプの「HTML メールの追跡」を参照してください)。

ログインユーザー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。不在の返信も含め、返ってきたメールはすべてログインユーザーに送信されます。不達管理が有効で、`SingleEmailMessage.targetObjectId` または `MassEmailMessage.targetObjectIds` が設定されている場合、不達は Salesforce によって自動的に処理され、該当するレコードが更新されます。それ以外の場合は、ログインユーザーに送信されます。不達管理は取引先責任者とリードにのみ機能します。

メモ:

- このコールで送信される単一メールメッセージは、送信側組織の 1 日の単一メール送信の制限に含まれます。この制限に達すると、SingleEmailMessage を使用する `sendEmail()` コールは拒否され、ユー

ザーには SINGLE_EMAIL_LIMIT_EXCEEDED エラーコードが返されます。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。

- このコールで送信される一括メールメッセージは、送信側組織の 1 日の一括メール送信の制限に含まれます。この制限に達すると、MassEmailMessage を使用する `sendEmail()` コールは拒否され、ユーザーには MASS_MAIL_LIMIT_EXCEEDED エラーコードが返されます。
- API バージョン 35.0 以降、`SingleEmailMessage` の `optOutPolicy` 項目を使用して、取引先責任者またはリードの [メール送信除外] 設定を適用または無視できます。`optOutPolicy` 項目は、メールの To、CC、および BCC リストの受信者に適用されます。以前のバージョンでは、デフォルトで `SingleEmailMessage` で受信者の [メール送信除外] 設定が無視され、メールがすべての受信者に送信されます。`MassEmailMessage` を使用すると、受信者の [メール送信除外] 設定が常に適用され、メールは除外した受信者には送信されず、その他のすべての受信者に送信されます。

SingleEmailMessage には、OrgWideEmailAddress のオブジェクト ID である `OrgWideEmailAddressId` という省略可能な項目があります。`OrgWideEmailAddressId` が設定されている場合、OrgWideEmailAddress `DisplayName` 項目が、ログインユーザーの [表示名] の代わりに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 の `DisplayName` および `senderDisplayName` の両方が定義されると、DUPLICATE_SENDER_DISPLAY_NAME エラーが発生します。

サンプルコード — Java

このサンプルでは、メールメッセージを作成して、宛先、CC、BCC の各受信者、件名、本文テキストを含む項目を設定します。また、`setTargetObjectId` メソッドを使用して、ログインユーザーの ID を受信者として設定します。これにより、指定されたユーザーのメールアドレスにメールが送信されます。このサンプルでは、添付ファイルを作成し、添付ファイルを含むメールメッセージを送信します。最後に、状況メッセージまたはエラーメッセージがある場合はコンソールに書き込みます。

```
public void doSendEmail() {
   try {
      EmailFileAttachment efa = new EmailFileAttachment();
      byte[] fileBody = new byte[1000000];
      efa.setBody(fileBody);
      efa.setFileName("attachment");
      SingleEmailMessage message = new SingleEmailMessage();
      message.setBccAddresses(new String[] {
         "someone@salesforce.com"
      });
      message.setCcAddresses(new String[] {
         "person1@salesforce.com", "person2@salesforce.com", "003xx00000a1b2cAAC"
      });
      message.setBccSender(true);
      message.setEmailPriority(EmailPriority.High);
      message.setReplyTo("person1@salesforce.com");
      message.setSaveAsActivity(false);
      message.setSubject("This is how you use the " + "sendEmail method.");
      // We can also just use an id for an implicit to address
      GetUserInfoResult guir = connection.getUserInfo();
      message.setTargetObjectId(guir.getUserId());
```

```
      message.setUseSignature(true);
      message.setPlainTextBody("This is the humongous body "
            + "of the message.");
      EmailFileAttachment[] efas = { efa };
      message.setFileAttachments(efas);
      message.setToAddresses(new String[] { "person3@salesforce.com" });
      SingleEmailMessage[] messages = { message };
      SendEmailResult[] results = connection.sendEmail(messages);
      if (results[0].isSuccess()) {
         System.out.println("The email was sent successfully.");
      } else {
         System.out.println("The email failed to send: "
               + results[0].getErrors()[0].getMessage());
      }
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

この例では、送信除外設定を適用してメールを送信する方法を示します。受信者は、そのIDで指定されます。SendEmailOptOutPolicy.FILTER オプションにより、メール送信を除外していない受信者にのみメールが送信されます。

```
SingleEmailMessage message = new SingleEmailMessage();
// Set recipients to two contact IDs.
// Replace IDs with valid record IDs in your org.
message.setToAddresses(new String[] { "003D000000QDexS", "003D000000QDfW5" });
message.setOptOutPolicy(SendEmailOptOutPolicy.FILTER);
message.setSubject("Opt Out Test Message");
message.setPlainTextBody("This is the message body.");
SingleEmailMessage[] messages = { message };
SendEmailResult[] results = connection.sendEmail(messages);
if (results[0].isSuccess()) {
    System.out.println("The email was sent successfully.");
} else {
    System.out.println("The email failed to send: "
          + results[0].getErrors()[0].getMessage());
}
```

サンプルコード — C#

このサンプルでは、メールメッセージを作成して、宛先、CC、BCCの各受信者、件名、本文テキストを含む項目を設定します。また、setTargetObjectId メソッドを使用して、ログインユーザーのIDを受信者として設定します。これにより、指定されたユーザーのメールアドレスにメールが送信されます。このサンプルでは、添付ファイルを作成し、添付ファイルを含むメールメッセージを送信します。最後に、状況メッセージまたはエラーメッセージがある場合はコンソールに書き込みます。

```
public void doSendEmail()
{
   try
   {
      EmailFileAttachment efa = new EmailFileAttachment();
```

```
      byte[] fileBody = new byte[1000000];
      efa.body = fileBody;
      efa.fileName = "attachment";
      SingleEmailMessage message = new SingleEmailMessage();
      message.setBccAddresses(new String[] {
         "someone@salesforce.com"
      });
      message.setCcAddresses(new String[] {
         "person1@salesforce.com", "person2@salesforce.com", "003xx00000a1b2cAAC"
      });
      message.bccSender = true;
      message.emailPriority = EmailPriority.High;
      message.replyTo = "person1@salesforce.com";
      message.saveAsActivity = false;
      message.subject = "This is how you use the " + "sendEmail method.";
      // We can also just use an id for an implicit to address
      GetUserInfoResult guir = binding.getUserInfo();
      message.targetObjectId = guir.userId;
      message.useSignature = true;
      message.plainTextBody = "This is the humongous body "
            + "of the message.";
      EmailFileAttachment[] efas = { efa };
      message.fileAttachments = efas;
      message.toAddresses = new String[] { "person3@salesforce.com" };
      SingleEmailMessage[] messages = { message };
      SendEmailResult[] results = binding.sendEmail(messages);
      if (results[0].success)
      {
         Console.WriteLine("The email was sent successfully.");
      }
      else
      {
         Console.WriteLine("The email failed to send: "
               + results[0].errors[0].message);
      }
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                        e.Message + "\n" + e.StackTrace);
   }
}
```

この例では、送信除外設定を適用してメールを送信する方法を示します。受信者は、そのIDで指定されます。`SendEmailOptOutPolicy.FILTER` オプションにより、メール送信を除外していない受信者にのみメールが送信されます。

```
SingleEmailMessage message = new SingleEmailMessage();
// Set recipients to two contact IDs.
// Replace IDs with valid record IDs in your org.
message.toAddresses = new String[] { "003D000000QDexS", "003D000000QDfW5" };
message.optOutPolicy = SendEmailOptOutPolicy.FILTER;
message.subject = "Opt Out Test Message";
message.plainTextBody = "This is the message body.";
```

```
SingleEmailMessage[] messages = { message };
SendEmailResult[] results = binding.sendEmail(messages);
if (results[0].success)
{
    Console.WriteLine("The email was sent successfully.");
} else {
    Console.WriteLine("The email failed to send: "
          + results[0].errors[0].message);
}
```

BaseEmail

次の表は、単一と一括メールの両方で使用される引数を含んでいます。

メモ: テンプレートが使用されていない場合、すべてのメールの内容は平文テキスト、HTML、またはその両方で書かれている必要があります。

名前	型	説明
bccSender	boolean	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。 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、ユーザーは 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。
saveAsActivity	boolean	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。
useSignature	boolean	そのユーザー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、ユーザーはメールに署名が含まれます。
emailPriority	picklist	省略可能。メールの優先度。 • Highest • High • Normal • Low • Lowest デフォルト値は Normal です。

名前	型	説明
replyTo	string	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。Visualforce メールテンプレートを使用して replyTo 値を指定している場合は、この引数を設定することはできません。
subject	string	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て、件名行を上書きしようすると、エラーメッセージが返されます。
templateId	ID	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。
senderDisplayName	string	省略可能。メールの From 行に表示される名前。SingleEmailMessage の OrgWideEmailAddressId に関連するオブジェクトが DisplayName 項目を定義している場合、この引数を設定することはできません。

SingleEmailMessage

次の行には、基本のメール引数に加えて、単一メールが使用する引数が含まれます。

名前	型	説明
bccAddresses	string[]	省略可能。ブラインドカーボンコピー (BCC) アドレス、またはメールの送信先となる取引先責任者、リード、ユーザーのオブジェクト ID の配列。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。この項目の最大サイズは 4,000 バイトです。メールあたりの toAddresses、ccAddresses、および bccAddresses の最大合計数は、150 です。これら 3 つの項目のすべての受信者は、Apex または API を使用して送信されるメールの制限に含まれます。 ID で追加された受信者の optOutPolicy 項目でのみメールの送信除外オプションを指定できます。 メールアドレスは検証されて、形式が正しく、これまで不達としてマークされていないことが確認されます。 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、ユーザーは 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。 すべてのメールで、次のいずれかの項目の少なくとも 1 つに受信者の値を割り当てる必要があります。 • toAddresses • ccAddresses

名前	型	説明
		• `bccAddresses` • `targetObjectId`
`ccAddresses`	string[]	**省略可能。カーボンコピー(CC)アドレス、またはメールの送信先となる取引先責任者、リード、ユーザーのオブジェクト ID の配列。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。この項目の最大サイズは 4,000 バイトです。メールあたりの `toAddresses`、`ccAddresses`、および `bccAddresses` の最大合計数は、150 です。これら 3 つの項目のすべての受信者は、Apex または API を使用して送信されるメールの制限に含まれます。** **ID で追加された受信者の `optOutPolicy` 項目でのみメールの送信除外オプションを指定できます。** **メールアドレスは検証されて、形式が正しく、これまで不達としてマークされていないことが確認されます。** **すべてのメールで、次のいずれかの項目の少なくとも 1 つに受信者の値を割り当てる必要があります。** • `toAddresses` • `ccAddresses` • `bccAddresses` • `targetObjectId`
`charset`	string	**省略可能。メール用の文字セット。この値が null の場合、ユーザーのデフォルト値が使われます。テンプレートには文字セットが指定されるため、`templateId` を指定した場合は使用できません。**
`documentAttachments`	ID[]	**非推奨。代わりに `entityAttachments` を使用します。省略可能。メールに添付する各 Document の ID をリストした配列。**
`entityAttachments`	ID[]	**省略可能。メールに添付する Document、ContentVersion、または Attachment 項目の ID の配列。** **この項目は、API バージョン 35.0 以降で使用できます。**
`fileAttachments`	EmailFileAttachment[]	**省略可能。メールに添付するバイナリとテキストファイルのファイル名をリストした配列。添付ファイルの合計が 20 MB を超えない限り、いくつでもファイルを追加できます。**

名前	型	説明
htmlBody	string	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。
inReplyTo	string	省略可能。送信メールの In-Reply-To 項目。このメールが返信しているメール (親メール) を識別します。親メールのメッセージ ID が含まれています。「RFC2822 - Internet Message Format」を参照してください。
optOutPolicy	SendEmailOptOutPolicy (string 型の列挙)	省略可能。取引先責任者、リード、または個人取引先の受信者をメールアドレスではなく ID で追加する場合、この項目によって sendEmail() コールの動作が決まります。デフォルトでは、メールアドレスで追加された受信者は、送信除外設定がオフになり、常にメールを受信します。SendEmailOptOutPolicy 列挙値の有効な値は次のとおりです。 • SEND (デフォルト) — メールはすべての受信者に送信されます。受信者の [メール送信除外] 設定は無視されます。[メールのプライバシー設定を適用] 設定は無視されます。 • FILTER — [メール送信除外] オプションが設定された受信者にはメールは送信されません。それ以外の受信者にはメールが送信されます。[メールのプライバシー設定を適用] 設定は無視されます。 • REJECT — [メール送信除外] オプションが設定された受信者がいる場合、sendEmail() がエラーを発生させ、メールは送信されません。[メールのプライバシー設定を適用] の設定が利用されます。これは、データプライバシーレコードの個人オブジェクトに基づいた選択であるためです。いずれかの受信者が [取引しない]、[処理しない]、または [この個人を除外] を選択している場合、sendEmail() によってエラーが発生し、メールは送信されません。 「営業以外のメールの送信」権限は考慮されません。 この項目は、API バージョン 35.0 以降で使用できます。
orgWideEmailAddressId	ID	省略可能。送信メールに関連付けられている OrgWideEmailAddress のオブジェクト ID。OrgWideEmailAddress.DisplayName は、

名前	型	説明
		senderDisplayName 項目がすでに設定されている場合は設定できません。
plainTextBody	string	省略可能。メールのテキスト版(送信者による指定)。
references	string	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。親メールのメッセージ ID 項目および References 項目、および場合によっては In-Reply-To 項目が含まれています。「RFC2822 - Internet Message Format」を参照してください。
targetObjectId	ID	省略可能。取引先責任者、リード、またはメールの送信先ユーザーのオブジェクト ID。入力するオブジェクト ID によりコンテキストが設定され、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。 すべてのメールで、次のいずれかの項目の少なくとも 1 つに受信者の値を割り当てる必要があります。 • toAddresses • ccAddresses • bccAddresses • targetObjectId
toAddresses	string[]	省略可能。メールアドレス、またはメールの送信先となる取引先責任者、リード、ユーザーのオブジェクト ID の配列。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。この項目の最大サイズは 4,000 バイトです。メールあたりの toAddresses、ccAddresses、および bccAddresses の最大合計数は、150 です。これら 3 つの項目のすべての受信者は、Apex または API を使用して送信されるメールの制限に含まれます。 ID で追加された受信者の optOutPolicy 項目でのみメールの送信除外オプションを指定できます。 メールアドレスは検証されて、形式が正しく、これまで不達としてマークされていないことが確認されます。 すべてのメールで、次のいずれかの項目の少なくとも 1 つに受信者の値を割り当てる必要があります。 • toAddresses • ccAddresses • bccAddresses

名前	型	説明
		• targetObjectId
treatBodiesAsTemplate	boolean	**省略可能**。true に設定すると、メールの件名、プレーンテキスト、およびHTMLテキスト本文がテンプレートデータとして扱われます。差し込み項目は、renderEmailTemplate() コールを使用して解決されます。デフォルトは false です。 この項目は、API バージョン 35.0 以降で使用できます。
treatTargetObjectAsRecipient	boolean	**省略可能**。true に設定すると、targetObjectId (取引先責任者、リード、またはユーザー)はメールの受信者になります。false に設定すると、targetObjectId は、テンプレート表示用の WhoId 項目として提供されますが、メールの受信者にはなりません。デフォルトは、true です。 この項目は、API バージョン 35.0 以降で使用できます。以前のバージョンでは、targetObjectId は常にメールの受信者でした。
whatId	ID	**省略可能**。targetObjectId 項目に取引先責任者を指定する場合、whatId も指定することができます。この項目により、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。値は、次の型のいずれかです。 • Account • Asset • Campaign • Case • Contract • Opportunity • Order • Product • Solution • Custom

MassEmailMessage

次の表には、基本のメール引数のほか、一括メール送信で使用される引数が含まれます。

名前	型	説明
description	string	一括メールキューでオブジェクトを識別するために内部で使用する値。
targetObjectIds	ID[]	取引先責任者、リード、またはメールの送信先ユーザーのオブジェクト ID の配列。入力するオブジェクト ID によりコンテキストが設定され、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。オブジェクトは同じタイプ (すべての取引先責任者、すべてのリード、またはすべてのユーザー) である必要があります。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、ユーザー、取引先責任者、またはリードに設定することが可能です。これらの ID を指定することにより、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。
whatIds	ID[]	省略可能。targetObjectIds 項目に取引先責任者の配列を指定する場合、whatIds の配列も指定可能です。これらの ID を指定することにより、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。値は、次の型のいずれかである必要があります。 • Contract • Case • Opportunity • Product 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。

EmailFileAttachment

次の表は、EmailFileAttachment が、要求の一部として渡す添付ファイルを指定するために SingleEmailMessage オブジェクトで使用する引数を示しています。なお、Document を渡す場合には documentAttachments を使用します。

プロパティ	型	説明
body	base64	添付ファイル自体。
contentType	string	省略可能。添付ファイルの内容の種類。
fileName	string	添付するファイルの名前。
inline	boolean	省略可能。Content-Dispositions がインラインか (true) 添付ファイルか (false) を示します。多くの場合、インラインコンテンツは、

プロパティ	型	説明
		メッセージ表示時にユーザーに表示されます。添付ファイルの内容は、ユーザーのアクションに基づいて表示されます。

応答

SendEmailResult

障害

次の API 状況コードが返される可能性があります。また、`sendEmail()` では、メールテンプレートの表示時に他のエラーが返される場合もあります。「`renderEmailTemplate()` の障害」を参照してください。

BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED

BCC_SELF_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED

DUPLICATE_SENDER_DISPLAY_NAME

EMAIL_ADDRESS_BOUNCED

EMAIL_NOT_PROCESSED_DUE_TO_PRIOR_ERROR

EMAIL_OPTED_OUT

ERROR_IN_MAILER

INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY

INVALID_CONTENT_TYPE

INVALID_EMAIL_ADDRESS

INVALID_ID_FIELD

INVALID_MESSAGE_ID_REFERENCE

INVALID_SAVE_AS_ACTIVITY_FLAG

LIMIT_EXCEEDED

MALFORMED_ID

MASS_MAIL_LIMIT_EXCEEDED

NO_MASS_MAIL_PERMISSION

REQUIRED_FIELD_MISSING

SINGLE_EMAIL_LIMIT_EXCEEDED

TEMPLATE_NOT_ACTIVE

UNVERIFIED_SENDER_ADDRESS

SendEmailResult

`sendEmail()` コールは、SendEmailResult オブジェクトのリストを返します。各 SendEmailResult 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
success	boolean	単一メールを送信する場合、配信するメールが、メッセージ転送エージェントによって正常に受け取られたか(true)、否か(false)を示します。success = true の場合であっても、メールがスパムブロッカーによって拒否されたかブロックされた可能性があるため、相手の受信者がそのメールを受け取ったわけではありません。また、配信するメールが、メッセージ転送エージェントによって正常に受け取られた場合でも、メール内では、個々のアドレスに関連付けられたエラー配列内にエラーが存在することがあります。 一括メール送信の場合、処理のためにメールがキューに正常に追加されたか(true)、否か(false)を示します。ただし、メールがキューに追加されても、対象受信者に配信されない処理エラーが発生する可能性は依然としてあります。
SendEmailError	Error[]	sendEmail() コールの最中にエラーが発生した場合、SendEmailError オブジェクトが返されます。単一メールでは、エラーは Salesforce でメールを配信できなかったことを示します。一括メール送信の場合、エラーは処理のためにメールがキューに正常に追加されなかったことを示します。

SendEmailError

SendEmailError には、次の属性があります。

名前	型	説明
Fields	Field[]	将来の使用のために予約されています。1 つ以上の項目名の配列。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。
Message	string	エラーメッセージテキスト。
StatusCode	statusCode	エラーを特徴付けるコード。ステータスコードの完全なリストは、組織の WSDL ファイルで確認できます。
TargetObjectId	ID	エラー発生先のオブジェクト ID。

メモ: sendEmail() で1つまたは複数の送信先にメールを送信することを妨げるエラーが発生した場合、SendEmailResult には、それらの送信先の各 TargetObjectId に関連付けられたエラーが含まれます。SendEmailResult 内に関連付けられたエラーを持たない TargetObjectId は、メールが送信先に送られたことを示します。SendEmailResult に、関連付けられる TargetObjectId がないエラーが存在する場合、メールはまったく送信されていません。

次は、一連のエラーの解析方法の例です。

```
Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
email.setToAddresses(new String[] { 'admin@acme.com' });
email.setSubject('my subject');
email.setPlainTextBody('plain text body');
List<Messaging.SendEmailResult> results =
    Messaging.sendEmail(new Messaging.Email[] { email });
if (!results.get(0).isSuccess()) {
    System.StatusCode statusCode = results.get(0).getErrors()[0].getStatusCode();
    String errorMessage = results.get(0).getErrors()[0].getMessage();
}
```

sendEmailMessage()

最大 10 件のドラフトメールメッセージをただちに送信します。

構文

Enterprise SOAP:

```
SendEmailResult[] = connection.sendEmailMessage( String[] draftEmailIds);
```

Partner SOAP:

```
SendEmailResult[] = connection.sendEmailMessage( ID[] draftEmailIds);
```

使用方法

このコールは、Lightning Platform AppExchange アプリケーション、カスタムアプリケーションなどの、Salesforce 以外のアプリケーションで、最大10個のドラフトメールメッセージの送信を行う場合に使用します。メッセージでは標準的なメール属性 (件名、BCC など) を含めることができ、Salesforce のメールテンプレートを使用することが可能です。平文テキストのほか HTML 形式もサポートされています。HTML メールの状況は、Salesforce を使って追跡できます。送信日、メールが最初に開かれた日、最後に開かれた日、開かれた回数などを確認できます。(詳細は、Salesforce ヘルプの「HTML メールの追跡」を参照してください)。

ログインユーザー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。不在の返信も含め、返ってきたメールはすべてログインユーザーに送信されます。不達管理が有効で、`SingleEmailMessage.targetObjectId` または `MassEmailMessage.targetObjectIds` が設定されている場合、不達は Salesforce によって自動的に処理され、該当するレコードが更新されます。それ以外の場合は、ログインユーザーに送信されます。不達管理は取引先責任者とリードにのみ機能します。

メモ:

- このコールで送信されるメールメッセージは、送信側組織の 1 日の単一メール送信の制限に含まれます。この制限に達すると、SingleEmailMessage を使用する `sendEmailMessage()` コールは拒否され、ユーザーには `SINGLE_EMAIL_LIMIT_EXCEEDED` エラーコードが返されます。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。

- このコールで送信される一括メールメッセージは、送信側組織の 1 日の一括メール送信の制限に含まれます。この制限に達すると、MassEmailMessage を使用する `sendEmail()` コールは拒否され、ユーザーには `MASS_MAIL_LIMIT_EXCEEDED` エラーコードが返されます。
- `AllOrNone` ヘッダーはこのコールでは優先されません。`sendEmailMessage()` では、`AllOrNone` ヘッダーが `true` に設定されている場合でも、部分的な成功が返されます。

サンプルコード — Java

このサンプルでは、ケースおよびドラフトメールメッセージを作成して、`From`、`To`、`CC`、`BCC` の各受信者、件名、本文テキストを含むメッセージ項目を設定します。また、添付ファイルを作成し、添付ファイルを含むメールメッセージを送信します。最後に、状況メッセージまたはエラーメッセージがある場合はコンソールに書き込みます。

```
public void doSendEmail() {
   try {
      //Create a case
      Case theCase = new Case();
      theCase.setSubject("Sample Case");
      SaveResult[] saveResult = connection.create(new SObject[] { theCase });
      String caseId = saveResult[0].getId();

      //Create a draft EmailMessage
      EmailMessage message = new EmailMessage();
      message.setParentId(theCase.getId());
      message.setBccAddress("bcc@email.com");
      message.setCcAddress("cc1@salesforce.com; cc2@email.com");
      message.setSubject("This is how you use the sendEmailMessage method.");
      message.setFromAddress("from@email.com");
      message.setFromName("Sample Code");
      message.setTextBody("This is the text body of the message.");
      message.setStatus("5"); //"5" means Draft
      message.setToAddress("to@email.com");
      saveResult = connection.create(new SObject[] { message });
      String emailMessageId = saveResult[0].getId();

      //Create an attachment for the draft EmailMessage
      Attachment att = new Attachment();
      byte[] fileBody = new byte[1000000];
      att.setBody(fileBody);
      att.setName("attachment");
      att.setParentId(emailMessageId);
      connection.create(new SObject[] { att });

      //Send the draft EmailMessage
      SendEmailResult[] results = connection.sendEmailMessage(messages);
      if (results[0].isSuccess()) {
         System.out.println("The email was sent successfully.");
      } else {
         System.out.println("The email failed to send: " +
            results[0].getErrors()[0].getMessage());
      }
```

```
    } catch (ConnectionException ce) {
       ce.printStackTrace();
    }
```

引数

なし。

応答

SendEmailResult[]

障害

BCC_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED

BCC_SELF_NOT_ALLOWED_IF_BCC_COMPLIANCE_ENABLED

EMAIL_NOT_PROCESSED_DUE_TO_PRIOR_ERROR

ERROR_IN_MAILER

INSUFFICIENT_ACCESS_ON_CROSS_REFERENCE_ENTITY

INVALID_CONTENT_TYPE

INVALID_EMAIL_ADDRESS

INVALID_ID_FIELD

INVALID_MESSAGE_ID_REFERENCE

LIMIT_EXCEEDED

MALFORMED_ID

REQUIRED_FIELD_MISSING

SINGLE_EMAIL_LIMIT_EXCEEDED

TEMPLATE_NOT_ACTIVE

UNVERIFIED_SENDER_ADDRESS

setPassword()

指定されたユーザーのパスワードを指定された値に設定します。

構文

```
SetPasswordResult setPasswordResult = connection.setPassword(ID userID, string password);
```

使用方法

User または SelfServiceUser のパスワードを指定の値に変更するには、setPassword() を使用します。たとえば、クライアントアプリケーションは、ユーザーに異なるパスワードを指定するように促し、次にシステム管理者によるユーザーのパスワード変更のために setPassword() を呼び出す可能性があります。API によって生成されたランダム値にパスワードをリセットする場合は、代わりに、resetPassword() を使用してください。

このコールは、組織のパスワードポリシー設定 [セルフリセットに **setPassword() API** を使用することを許可] が有効である場合に限り、ユーザーに自分のパスワード変更を許可するために使用できます。それ以外の場合は、changeOwnPassword() を使用します。変更を許可する前にユーザーの現在のパスワードを確認するため、こちらの方が安全です。

クライアントアプリケーションは、そのユーザーのパスワードを変更するために必要なアクセス権限でログインされている必要があります。詳細は、「データアクセスに影響する要素」を参照してください。

ID についての詳細は、「ID データ型」を参照してください。

パスワードが失効していた場合、このコールは、LoginResult 内で返されるセッション ID を使用可能です。詳細は、「passwordExpired」を参照してください。

サンプルコード — Java

このサンプルでは、ユーザー ID およびパスワードパラメーターを受け取り、これらを使用して指定されたユーザーのパスワードを設定する setPassword() コールを実行します。

```
public void doSetPassword(String userId, String newPasswd) {
   try {
      SetPasswordResult result = connection.setPassword(userId, newPasswd);
      System.out.println("The password for user ID " + userId + " changed to "
            + newPasswd);
   } catch (ConnectionException ce) {
      ce.printStackTrace();
   }
}
```

サンプルコード — C#

このサンプルでは、ユーザー ID およびパスワードパラメーターを受け取り、これらを使用して指定されたユーザーのパスワードを設定する setPassword() コールを実行します。

```
public void doSetPassword(String userId, String newPasswd)
{
   try
   {
      SetPasswordResult result = binding.setPassword(userId, newPasswd);
      Console.WriteLine("The password for user ID " + userId + " changed to "
            + newPasswd);
   }
   catch (SoapException e)
   {
      Console.WriteLine("An unexpected error has occurred: " +
                        e.Message + "\n" + e.StackTrace);
```

```
    }
}
```

引数

名前	型	説明
`userID`	ID	パスワードをリセットする User または SelfServiceUser の ID。ID についての詳細は、「ID データ型」を参照してください。
`password`	string	指定されたユーザーのために使用される新しいパスワード。

応答

SetPasswordResult (空)

障害

`InvalidIdFault`

`UnexpectedErrorFault`

関連トピック:

- resetPassword()
- ユーティリティ API コール
- changeOwnPassword()

第 15 章　SOAP ヘッダー

API は、クライアントアプリケーションに SOAP ヘッダーを提供します。

ヘッダー	説明
AllOrNoneHeader	正常に処理されないレコードがあった場合に、コールですべての変更をロールバックするかどうかを指定します。このヘッダーは、API バージョン 20.0 以降で使用できます。
AllowFieldTruncationHeader	API バージョン 15.0 以降のいくつかのデータ型に切り捨て動作を指定します。
AssignmentRuleHeader	Account、Case、または Lead を作成または更新する場合に使用する代入ルールを指定します。
CallOptions	API 要求のコールオプションを指定します。
DebuggingHeader	出力ヘッダー `DebuggingInfo` でデバッグログを返し、デバッグログの詳細レベルを指定します。
DisableFeedTrackingHeader	現在のコールに適用された変更をフィードで追跡するかどうかを指定します。
DuplicateRuleHeader	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。
EmailHeader	要求が処理されるときにメール通知を送信します。Salesforce ユーザーインターフェースと同等の機能を提供します。
LimitInfoHeader	コールから SOAP API に返される応答ヘッダー。このヘッダーは、組織の制限情報を返します。このヘッダーを使用して、組織にコールを行うときの API 制限を監視します。
LocaleOptions	返されるラベルの言語を指定します。値は、`de_DE`、`en_GB` など有効なユーザーロケール (言語および国) である必要があります。ロケールについての詳細は、CategoryNodeLocalization オブジェクトの `Language` 項目を参照してください。
LoginScopeHeader	`login()` コールを使用して組織のセルフサービスユーザーを認証できるように、組織 ID を指定します。

ヘッダー	説明
MruHeader	MRU アイテムのリストを更新するか (`true`)、更新しないか (`false`) を示します。
OwnerChangeOptions	添付ファイルおよびメモの所有者を指定します。
PackageVersionHeader	API バージョン 16.0 以降で、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。
QueryOptions	クエリ結果のバッチサイズを指定します。
SessionHeader	`login()` が正常に行われた後にログインサーバーから返される、セッション ID を指定します。
UserTerritoryDeleteHeader	現在の所有者がテリトリーから削除された場合に進行中の商談が割り当てられるユーザーを指定します。

AllOrNoneHeader

正常に処理されないレコードがあった場合に、コールですべての変更をロールバックできます。

AllOrNoneHeader ヘッダーがない場合、エラーのないレコードはコミットされますが、エラーのあったレコードはコールの結果では失敗とマークされます。このヘッダーは、API バージョン 20.0 以降で使用できます。

ヘッダーが有効な場合でも、エラーのあるレコードを特定するには、レコードごとにコールの結果の `success` 項目を確認する必要があります。各 `success` 項目には `true` または `false` が含まれ、コールが正常に処理されたかどうかを示します。

少なくとも 1 つのレコードに関連するエラーがある場合、レコードのコールの結果の `errors` 項目でエラーの詳細情報が提供されます。同じコールのその他のレコードにエラーがない場合、`errors` 項目は他のエラーによりレコードがロールバックされたことを示します。

API コール

`create()`、`delete()`、`undelete()`、`update()`、`upsert()`

項目

要素名	型	説明
`allOrNone`	boolean	`true` の場合、コールに失敗したレコードが存在すると、コールに対するすべての変更がロールバックされます。すべてのレコードが正常に処理されない限りレコードの変更はコミットされません。

要素名	型	説明
		デフォルトは、false です。一部のレコードは正常に処理されますが、その他のレコードはコールの結果で失敗とマークされます。

サンプルコード — Java

このサンプルでは、AllOrNoneHeader の使用方法を示します。2 つの取引先責任者の作成を試みますが、2 番目の取引先責任者について必須項目の一部が設定されていないため、作成に失敗します。次に、このサンプルでは allOrNone 項目を true に設定し、取引先責任者の作成を試みます。いずれかの取引先責任者の作成でエラーが発生するため、トランザクション全体がロールバックされ、取引先責任者は作成されません。

```
public void allOrNoneHeaderSample() {
  try {
    // Create the first contact.
    SObject[] sObjects = new SObject[2];
    Contact contact1 = new Contact();
    contact1.setFirstName("Robin");
    contact1.setLastName("Van Persie");

    // Create the second contact. This contact doesn't
    // have a value for the required
    // LastName field so the create will fail.
    Contact contact2 = new Contact();
    contact2.setFirstName("Ashley");
    sObjects[0] = contact1;
    sObjects[1] = contact2;

    // Set the SOAP header to roll back the create unless
    // all contacts are successfully created.
    connection.setAllOrNoneHeader(true);
    // Attempt to create the two contacts.
    SaveResult[] sr = connection.create(sObjects);
    for (int i = 0; i < sr.length; i++) {
      if (sr[i].isSuccess()) {
        System.out.println("Successfully created contact with id: " +
          sr[i].getId() + ".");
      }
      else {
        // Note the error messages as the operation was rolled back
        // due to the all or none header.
        System.out.println("Error creating contact: " +
          sr[i].getErrors()[0].getMessage());
        System.out.println("Error status code: " +
          sr[i].getErrors()[0].getStatusCode());
      }
    }
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
```

```
    }
}
```

AllowFieldTruncationHeader

一部の項目について、文字列が大きすぎる場合は操作が失敗するように指定します。ヘッダーがない場合、これらの項目の文字列は切り捨てられます。

`AllowFieldTruncationHeader` **ヘッダーは、次のデータ型に影響します。**

- anyType (**ここにリストされていないデータ型のいずれかを示す場合**)
- email
- encryptedstring
- multipicklist
- phone
- picklist
- string
- textarea

バージョン 15.0 **より前の** API **では、記載のいずれかの項目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、値が大きすぎると、操作は失敗し、失敗コード** `STRING_TOO_LONG` **が返されます。** `AllowFieldTruncationHeader` **ヘッダーを使用すると、**API**バージョン** 15.0**以降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。**

このヘッダーはバージョン 14.0 **以前の製品には無効です。**

API コール

convertLead()、create()、merge()、process()、undelete()、update()、**および** upsert()

Apex: `executeanonymous()`

項目

要素名	型	説明
`allowFieldTruncation`	boolean	`true` **の場合、長すぎる項目値を切り捨てます。これは**API**バージョン** 14.0 **以前の動作です。** **デフォルトは** `false` **です (変更は行われません)。** `string` **値および** `textarea` **が大きすぎる場合、操作は失敗し、失敗コード** `STRING_TOO_LONG` **が返されます。** **次のリストには、切り捨ておよびこのヘッダーの影響を受ける項目のデータ型を示しています。** • anyType(**ここにリストされていないデータ型のいずれかを示す場合**) • email

要素名	型	説明
		• encryptedstring • multipicklist • phone • picklist • string • textarea

サンプルコード — Java

［名前］項目の名前が長すぎる取引先を作成する場合、AllowFieldTruncation **ヘッダーを使用します。**

この例では、次のような作業を行います。

1. 255 **文字の項目制限を超える名前を使用して取引先オブジェクトを作成します。**
2. create **コールを送信しますが、名前項目の長さが原因で失敗します。**
3. `AllowFieldTruncationHeader` **を** `true` **に設定し、取引先の作成を再試行すると、今度は成功します。**

```
public void allowFieldTruncationSample() {
  try {
    Account account = new Account();
    // Construct a string that is 256 characters long.
    // Account.Name's limit is 255 characters.
    String accName = "";
    for (int i = 0; i < 256; i++) {
      accName += "a";
    }
    account.setName(accName);
    // Construct an array of SObjects to hold the accounts.
    SObject[] sObjects = new SObject[1];
    sObjects[0] = account;
    // Attempt to create the account. It will fail in API version 15.0
    // and above because the account name is too long.
    SaveResult[] results = connection.create(sObjects);
    System.out.println("The call failed because: "
       + results[0].getErrors()[0].getMessage());
    // Now set the SOAP header to allow field truncation.
    connection.setAllowFieldTruncationHeader(true);
    // Attempt to create the account now.
    results = connection.create(sObjects);
    System.out.println("The call: " + results[0].isSuccess());
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

AssignmentRuleHeader

AssignmentRuleHeader を、指定された割り当てルールの Case または Lead の create() コールまたは update() コールに指定する必要があります。また、適用されるテリトリー割り当てルールで Account に対して update() コールを実行する場合に指定する必要があります。

API コール

create()、merge()、update()、upsert()

項目

要素名	型	説明
assignmentRuleId	ID	Case または Lead に対して実行される特定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 オブジェクトを照会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。取引先では、すべてのテリトリールールが適用されるため、この要素は無視されます。 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、MALFORMED_ID 例外が返されます。
useDefaultRule	boolean	Case または Lead では true の場合、Case または Lead のデフォルトの (有効な) 割り当てルールを使用します。指定する場合は、assignmentRuleId を指定しないでください。取引先に対して true であれば、すべてのテリトリー割り当てルールが適用されます。取引先に対して false であれば、テリトリー割り当てルールは適用されません。

サンプルコード

コードの例については、「Lead」を参照してください。

関連トピック:

https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_assignmentrule.htm

CallOptions

特定のクライアントで使用する必要があるオプションを指定します。このヘッダーは、Partner WSDL で使用する場合にのみ有効です。

API コール

`defaultNamespace` 要素は、create()、merge()、`queryAll()`、`query()`、`queryMore()`、`retrieve()`、`search()`、update()、および upsert() のコールをサポートしています。

`client` 要素は、上記すべてのコールと `login()`、`delete()`、`describeGlobal()`、`describeLayout()`、`describeTabs()`、`describeSObject()`、`describeSObjects()`、`getDeleted()`、`getUpdated()`、process()、undelete()、`getServerTimestamp()`、`getUserInfo()`、`setPassword()`、および `resetPassword()` のコールをサポートしています。

項目

要素名	型	説明
`client`	string	クライアントを識別する文字列。
`defaultNamespace`	string	開発者の名前空間プレフィックスを識別する文字列。この項目を使用して、`fieldName` をすべての場所で完全に指定することなく、管理パッケージの項目名を解決します。 たとえば、開発者名前空間プレフィックスが `battle` で、`botId` というパッケージにカスタム項目がある場合、このヘッダーを設定して、次のようなクエリを正常に行えます。 `query("SELECT id, botId__c from Account");` この場合、実際に照会される項目は、`battle__botId__c` です。 この項目を使用すると、名前空間プレフィックスを指定せずにクライアントコードを作成することができます。この項目を指定しない場合、クエリを正常に行うには、この項目の完全名を使用する必要があります。上記の例では、`battle__botId__c` を指定する必要があります。 この項目が設定され、クエリが名前空間を指定している場合、レスポンスにはプレフィックスは含まれません。たとえば、このヘッダーを `battle` に設定し、`query("SELECT id, battle__botId__c from Account");` のようなクエリを発行すると、レスポンスは `battle_botId__c` 要素ではなく、`botId__c` 要素を使用します。 記述用の API コール (describe) はこのヘッダーを無視するため、名前空間プレフィックスを持つ項目と、プレフィックスのない同じ名前のカスタム項目との間で混乱が生じることはありません。

サンプルコード — C#

このサンプルでは、CallOptions ヘッダーの使用方法を示します。クライアント ID および開発者の名前空間プレフィックスを設定します。これは管理パッケージで項目名を解決するために使用します。次に、指定したユーザーをログインします。

```
public void CallOptionsSample()
{
    // Web Reference to the imported Partner WSDL.
    APISamples.partner.SforceService partnerBinding;

    string username = "USERNAME";
    string password = "PASSWORD";

    // The real Client ID will be an API Token provided by Salesforce
    // to partner applications following a security review.
    // For more details, see the Security Review FAQ in the online help.
    string clientId = "SampleCaseSensitiveToken/100";

    partnerBinding = new SforceService();
    partnerBinding.CallOptionsValue = new CallOptions();
    partnerBinding.CallOptionsValue.client = clientId;

    // Optionally, if a developer namespace prefix has been registered for
    // your Developer Edition organization, it may also be specified.
    string prefix = "battle";
    partnerBinding.CallOptionsValue.defaultNamespace = prefix;

    try
    {
        APISamples.partner.LoginResult lr =
            partnerBinding.login(username, password);
    }
    catch (SoapException e)
    {
        Console.WriteLine(e.Code);
        Console.WriteLine(e.Message);
    }
}
```

DisableFeedTrackingHeader

現在のコールで行われた変更をフィードで追跡することを指定します。

このヘッダーは、レコードに関連するさまざまなフィードで変更を追跡しないで大量のレコードを処理する必要がある場合に使用します。このヘッダーは、組織で Chatter 機能が有効になっている場合にのみ使用できます。

API コール

convertLead()、create()、delete()、merge()、process()、undelete()、update()、upsert()

項目

要素名	型	説明
disableFeedTracking	boolean	true の場合、現在のコールで行われた変更をフィードで追跡しません。 デフォルトは、false です。

サンプルコード — Java

このサンプルでは、DisableFeedTrackingHeader の使用方法を示します。このヘッダーを true に設定することで、フィード追跡を無効にし、多数の取引先レコードを一括して作成します。

```
public void disableFeedTrackingHeaderSample() {
  try {
    // Insert a large number of accounts.
    SObject[] sObjects = new SObject[500];
    for (int i = 0; i < 500; i++)  {
       Account a = new Account();
       a.setName("my-account-" + i);
       sObjects[i] = a;
    }
    // Set the SOAP header to disable feed tracking to avoid generating a
    // large number of feed items because of this bulk operation.
    connection.setDisableFeedTrackingHeader(true);
    // Perform the bulk create. This won't result in 500 feed items, which
    // would otherwise be generated without the DisableFeedTrackingHeader.
    SaveResult[] sr = connection.create(sObjects);
    for (int i = 0; i < sr.length; i++) {
      if (sr[i].isSuccess()) {
        System.out.println("Successfully created account with id: " +
          sr[i].getId() + ".");
      } else {
        System.out.println("Error creating account: " +
          sr[i].getErrors()[0].getMessage());
      }
    }
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

関連トピック:

https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_customobject__feed.htm

https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_entitysubscription.htm

DebuggingHeader

出力ヘッダー `DebuggingInfo` でデバッグログを返し、デバッグログの詳細レベルを指定します。

メモ: DebuggingHeader を含むコールは、24 時間以内に 1,000 件に制限されています。これらのコールは、組織の合計要求数制限に達した後も引き続き実行できます。

API のコール

`compileAndTest()`、`executeanonymous()`、`runTests()`

項目

要素名	型	説明
`categories`	LogInfo[]	デバッグログで返される情報の種別と量を指定します。
`debugLevel`	DebugLevel (string 型の列挙)	非推奨。この項目は後方互換性確保の目的でのみ提供されます。`debugLevel` と `categories` の両方の値を指定すると、`categories` の値が使用されます。 `debugLevel` 項目では、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`

LogInfo

デバッグログで返される情報の種別と量を指定します。`categories` 項目は、これらのオブジェクトのリストを取ります。LogInfo は、`category` から `level` への対応付けです。

項目

要素名	型	説明
`category`	LogCategory	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`

要素名	型	説明
		• Workflow • Validation • Callout • Apex_code • Apex_profiling • Visualforce • System • All
level	LogCategoryLevel	デバッグログに返される詳細の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)。 • NONE • ERROR • WARN • INFO • DEBUG • FINE • FINER • FINEST

DuplicateRuleHeader

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。

API コール

create()、update()、upsert()

項目

要素名	型	説明
allowSave	boolean	重複ルールでこのプロパティを true に設定すると、アラートオプションが有効になっていてもアラートがスキップされ、重複レコードが保存されます。このプロパティを false に設定すると、重複レコードが保存されなくなります。

要素名	型	説明
includeRecordDetails	boolean	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、このプロパティを 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、このプロパティを false に設定します。
runAsCurrentUser	boolean	重複ルールを実行するときに現在のユーザーの共有ルールを適用するには、このプロパティを true に設定します。要求のクラスで指定した共有ルールを使用するには、このプロパティを 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、現在のユーザーの共有ルールは適用されません。

Java のサンプル

次のサンプルは、DuplicateRuleHeader を使用して、重複ルールを使用するためのオプションを設定する方法を示します。サンプルアプリケーションの全体は、「DuplicateResult」を参照してください。

```
_DuplicateRuleHeader header = new _DuplicateRuleHeader();
        header.setAllowSave(false);
        header.setIncludeRecordDetails(true);
        header.setRunAsCurrentUser(true);

        binding.setHeader(new SforceServiceLocator().getServiceName().getNamespaceURI(),
"DuplicateRuleHeader", header);
```

関連トピック:

https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_duplicateresult.htm

https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_duplicaterule.htm

EmailHeader

Salesforceユーザーインターフェースを使用して、次の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。

- Case を作成する
- CaseComment を作成する
- Case のメールを Contact に変換する
- 新規 User のメール通知を送信する
- resetPassword() コールを実行する

API バージョン 8.0 以降では、メールを送信する API 要求も送信できます。

グループ行動とは、IsGroupEvent が true である行動です。EventRelation オブジェクトは、グループ行動に招待されているユーザー、リード、取引先責任者を追跡します。API を使用して送信されるグループの行動に関するメールでは、次のような動作に注意してください。

- ユーザー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。
- グループの行動の更新または削除時に送信されるメールは、送信対象に基づき triggerUserEmail や triggerOtherEmail の影響を受けます。

API コール

create()、delete()、resetPassword()、update()、upsert()

項目

要素名	型	説明
triggerAutoResponseEmail	boolean	リード、ケースに対して自動応答ルールをトリガーするか (true)、トリガーしないか (false) を示します。Salesforce ユーザーインターフェースで、このメールは、ケースの作成やユーザー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ーされます。この値が true に設定されている場合、Case 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	boolean	組織外のメールをトリガーするか (true)、トリガーしないか (false) を示します。Salesforce ユーザーインターフェースで、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ーされます。
triggerUserEmail	boolean	組織内のユーザーに送信されるメールをトリガーするか (true)、トリガーしないか (false) を示します。Salesforce ユーザーインターフェースで、このメールはパスワードのリセット、ユーザーの新規作成、ケースへのコメントの追加など、さまざまなイベントによって自動的にトリガーされます。

サンプルコード — Java

このサンプルでは、EmailHeader の使用方法を示します。triggerAutoResponseEmail メールヘッダー項目を true に設定します。この設定では、ケースが作成されたときにメールが送信されます。次に、ケースを作成します。このサンプルでは、ケースに自動応答ルールが設定され、ContactId で参照される取引先責任者にメールアドレスが指定されていることを前提としています。

```
public void createCaseWithAutoResponse(String contactId) {
  try {
    connection.setEmailHeader(true, false, false);
    Case c = new Case();
    c.setSubject("Sample Subject");
    c.setContactId(contactId);
    SaveResult[] sr = connection.create(new SObject[] { c });
    // Parse sr array to see if case was created successfully.
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

LimitInfoHeader

コールから SOAP API に返される応答ヘッダー。このヘッダーは、組織の制限情報を返します。このヘッダーを使用して、組織にコールを行うときの API 制限を監視します。

API コール

login() を除く、すべてのコール。

項目

要素名	型	説明
current	int	組織ですでに使用中の指定された制限種別に対するコール数。
limit	int	指定された制限種別の組織の制限。
type	string	ヘッダーで指定された制限情報の種別。 • API REQUESTS — コールが行われた組織の API 使用状況。

サンプルコード

次の例は、Merchandise レコードの SOAP 要求に対する応答を示しています。`LimitInfoHeader` には、組織の API 使用状況情報が含まれます。

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="urn:partner.soap.sforce.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:sf="urn:sobject.partner.soap.sforce.com">
    <soapenv:Header>
        <LimitInfoHeader>
            <limitInfo>
                <current>5</current>
                <limit>100000</limit>
                <type>API REQUESTS</type>
            </limitInfo>
    </soapenv:Header>
    <soapenv:Body>
        <queryResponse>
            <result xsi:type="QueryResult">
                <done>true</done>
                <queryLocator xsi:nil="true"/>
                <records xsi:type="sf:sObject">
                    <sf:type>dev_ns__Merchandise__c</sf:type>
                    <sf:Id>a00D0000008pQSNIA2</sf:Id>
                    <sf:dev_ns__Description__c>Phone Case for iPhone
                        4/4S</sf:dev_ns__Description__c>
                    <sf:dev_ns__Price__c>16.99</sf:dev_ns__Price__c>
                    <sf:dev_ns__Stock_Price__c>12.99</sf:dev_ns__Stock_Price__c>
                    <sf:dev_ns__Total_Inventory__c>108.0</sf:dev_ns__Total_Inventory__c>
                    <sf:Id>a00D0000008pQSNIA2</sf:Id>
                </records>
                <size>1</size>
            </result>
        </queryResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

LocaleOptions

返されるラベルの言語を指定します。

API コール

describeSObject()、describeSObjects()、describeDataCategoryGroups()、
describeDataCategoryGroupStructures()

項目

要素名	型	説明
language	string	返されるラベルの言語を指定します。値は、de_DE、en_GB など有効なユーザーロケール (言語および国) である必要があります。ロケールについての詳細は、CategoryNodeLocalization オブジェクトの Language 項目を参照してください。

サンプルコード — Java

このサンプルでは、LocaleOptions ヘッダーをログインユーザーのロケールに設定してから、取引先に対して describe を実行します。

```
public void localeOptionsExample() {
  try {
    connection.setLocaleOptions("en_US");
    connection.describeSObject("Account");
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

LoginScopeHeader

組織 ID を指定し、既存の login() コールを使用して組織のセルフサービスユーザーを認証できるようにします。

メモ: Spring '12 以降、新しい Salesforce 組織ではセルフサービスポータルを利用できません。既存の組織は、引き続きセルフサービスポータルを使用できます。

API コール

login()

項目

要素名	型	説明
organizationId	ID	セルフサービスユーザーを認証する組織の ID。

要素名	型	説明
portalId	ID	カスタマーポータルユーザーである場合にのみ入力します。この組織のポータルの ID。ID は Salesforce ユーザーインターフェースで次の手順によって取得できます。 • [設定]から、[クイック検索] ボックスに「*カスタマーポータル設定*」と入力し、[カスタマーポータル設定] を選択します。 • カスタマーポータル名を選択すると、[カスタマーポータルの詳細] ページに、カスタマーポータルの URL が表示されます。ポータル ID は URL に含まれています。

サンプルコード — C#

このサンプルでは、LoginScopeHeader の使用方法を示します。カスタマーポータルユーザーの組織 ID とポータル ID を設定します。また、CallOptions ヘッダーを設定します。次に、指定されたユーザーをログインします。

```
/// Demonstrates how to set the LoginScopeHeader values.
public void LoginScopeHeaderSample()
{
    // Web Reference to the imported Partner WSDL.
    APISamples.partner.SforceService partnerBinding;

    string username = "USERNAME";
    string password = "PASSWORD";

    // The real Client ID will be an API Token provided by Salesforce
    // to partner applications following a security review. For more details,
    // see the Security Review FAQ in the online help.
    string clientId = "SampleCaseSensitiveToken/100";

    partnerBinding = new SforceService();
    partnerBinding.CallOptionsValue = new CallOptions();
    partnerBinding.CallOptionsValue.client = clientId;

    // To authenticate Self-Service users, we need to set the OrganizationId
    // in the LoginScopeHeader.
    string orgId = "00ID0000OrgFoo";
    partnerBinding.LoginScopeHeaderValue = new LoginScopeHeader();
    partnerBinding.LoginScopeHeaderValue.organizationId = orgId;
    // Specify the Portal ID if the user is a Customer Portal user.
    string portalId = "00ID0000FooPtl";
    partnerBinding.LoginScopeHeaderValue.portalId = portalId;

    try
    {
        APISamples.partner.LoginResult lr =
            partnerBinding.login(username, password);
    }
```

```
    catch (SoapException e)
    {
        Console.WriteLine(e.Code);
        Console.WriteLine(e.Message);
    }
}
```

MruHeader

API バージョン 7.0 以降では、create()、update()、および upsert() コールは、ヘッダーが使用されていない限り、Salesforce ユーザーインターフェースのサイドバーの [最近使ったデータ] の最近使用した (MRU) 項目のリストは更新しません。このヘッダーを使用して最近使ったデータのリストを更新すると、パフォーマンスに悪影響を及ぼす場合があります。

API コール

create()、merge()、query()、retrieve()、update()、upsert()

項目

要素名	型	説明
updateMru	boolean	MRU アイテムのリストを更新するか (true)、更新しないか (false) を示します。 retrieve() では、結果が 1 行のみの場合、MRU を retrieve の結果の ID に更新します。 query() では、結果が 1 行のみで、ID 項目が選択されている場合、MRU を query の結果の ID に更新します。

サンプルコード — Java

このサンプルでは、MruHeader を true に設定することにより MRU リストの更新オプションをオンにします。次に、取引先を作成します。

```
public void mruHeaderSample() {
  connection.setMruHeader(true);
  Account account = new Account();
  account.setName("This will be in the MRU");
  try {
    SaveResult[] sr = connection.create(new SObject[]{account});
    System.out.println("ID of account added to MRU: " +
      sr[0].getId());
  } catch (ConnectionException ce) {
    ce.printStackTrace();
```

```
    }
}
```

OwnerChangeOptions

レコードの所有者が変更されると実行できるアクションを表します。これらのオプションは、API バージョン 35.0 以降で使用できます。

API コール

update()、upsert()

項目

要素名	型	説明
options	OwnerChangeOption[]	update または upsert コールでレコード所有者を変更するときに実行する特定のアクションのフラグを表します。

OwnerChangeOption 項目

要素名	型	説明
execute	boolean	true の場合、type 項目で表されるアクションが実行されます。false の場合、type 項目で表されるアクションがスキップされます。
type	文字列の列挙	update または upsert コールでレコード所有者を変更するときに execute 項目で指定された値に応じて実行またはスキップするアクションを表します。次の型を使用できます。 **EnforceNewOwnerHasReadAccess** true の場合、レコードの新しい所有者に少なくともレコードへの参照アクセス権が必要です。API バージョン 36.0 以降で利用できます。 **KeepAccountTeam** true の場合、取引先所有者が変更されたときに取引先と共に取引先チームが保持されます。false の場合、取引先チームは削除されます。デフォルトは false です。このアクションは、Salesforce システム管理者、取引先所有者、またはロール階層内のそれより上位のユーザーが追加したチームメンバーにのみ適用されます。グループベースのアクセス権を持つユーザーが追加したチームメンバーは、

要素名	型	説明
		`true` の場合であっても削除されます。取引先で API バージョン 45.0 以降で使用できます。 **KeepSalesTeam** `true` の場合、取引先所有者が変更されたときに商談と共に商談チームが保持されます。`false` の場合、商談チームは削除されます。デフォルトは `false` です。商談で API バージョン 45.0 以降で使用できます。 **KeepSalesTeamGrantCurrentOwnerReadWriteAccess** `true` の場合、所有者が変更された後に商談の以前の所有者の「参照・更新」アクセス権が保持されます。デフォルトは `false` です。KeepSalesTeam が `true` の場合のみ `true` に設定できます。商談で API バージョン 44.0 以降で使用できます。 **SendEmail** `true` の場合、メール通知は新しい所有者に送信されます。デフォルトは `false` です。 **TransferAllOwnedCases** `true` の場合、取引先所有者が所有するすべてのケース (オープンとクローズ) が新しい所有者に移行されます。デフォルトは `false` です。TransferAllOwnedCases が `true` の場合、TransferOwnedOpenCases も true である必要があります。取引先で API バージョン 45.0 以降で使用できます。 **TransferArticleOwnedPublishedVersion** `true` で、レコードがナレッジ記事の場合、現在のドラフトの言語に対応する記事所有者の公開バージョンが、現在のドラフトに加えて新しい所有者に移行されます。 **TransferArticleOwnedArchivedVersions** `true` で、レコードがナレッジ記事の場合、現在のドラフトの言語に対応する記事所有者のアーカイブ済みバージョンが、現在のドラフトに加えて新しい所有者に移行されます。 **TransferArticleAllVersions** `true` で、レコードがナレッジ記事の場合、現在のドラフトの言語に対応するすべての公開バージョンとアーカイブ済みバージョンが、現在のドラフトに加えて新しい所有者に移行されます。 **TransferContacts** `true` で、レコードが法人取引先の場合、取引先に関連付けられている取引先責任者が新しい所有者に移行されます。 **TransferContracts** `true` で、レコードが取引先の場合、取引先に関連付けられていて、取引先所有者が所有している契約が新しい所有者に移行されます。

要素名	型	説明
		TransferNotesAndAttachments `true` の場合、レコードのメモ、添付ファイル、および Google ドキュメントが新しいレコード所有者に移行されます。`false` の場合、元のレコード所有者が所有権を保持します。 **TransferOpenActivities** `true` の場合、レコードの活動予定が新しい所有者に移行されます。 **TransferOrders** `true` で、レコードが取引先の場合、取引先に関連付けられているスタンドアロンのドラフト注文と、取引先所有者が所有している移行済み契約に関連付けられているドラフト注文が新しい所有者に移行されます。 **TransferOthersOpenOpportunities** `true` で、レコードが取引先の場合、取引先に関連付けられていて、現在の所有者が所有していない進行中の商談が新しい所有者に移行されます。このオプションが実行される場合、TransferOwnedOpenOpportunities も実行されるよう設定する必要があります。デフォルトは `false` です。 **TransferOwnedClosedOpportunities** `true` で、レコードが取引先の場合、取引先所有者が所有している完了した商談が新しい所有者に移行されます。デフォルトは `false` です。API バージョン 45.0 以降で使用できます。 **TransferOwnedOpenCases** `true` で、レコードが取引先の場合、取引先所有者が所有しているオープンケースが新しい所有者に移行されます。デフォルトは `false` です。API バージョン 45.0 以降で使用できます。 **TransferOwnedOpenOpportunities** `true` で、レコードが取引先の場合、取引先に関連付けられていて、取引先所有者が所有している進行中の商談が新しい所有者に移行されます。

使用方法

複数の取引先の所有者を変更するときは、すべての取引先の古い所有者と新しい所有者が同じでなければなりません。所有者が異なる取引先の所有権を変更するには、個別の API 要求を使用します。

サンプルコード — Java

このサンプルは、取引先と、取引先のメモ、商談、ToDoを作成します。また、取引先と共にメモ、商談、ToDoが新しい所有者に移行されるように所有者変更オプションを設定します。

```
public void ownerChangeOptionsHeaderSample() {

    // Create account. Accounts don't transfer activities, notes, or attachments by default



    Account account = new Account();
    account.setName("Account");
    com.sforce.soap.enterprise.SaveResult[] sr = connection.create(new
com.sforce.soap.enterprise.sobject.SObject[] { account } );
    String accountId = null;

    if(sr[0].isSuccess()) {
        System.out.println("Successfully saved the account");
        accountId = sr[0].getId();

        // Create a note, a task, and an opportunity for the account

        Note note = new Note();
        note.setTitle("Note Title");
        note.setBody("Note Body");
        note.setParentId(accountId);

        Task task = new Task();
        task.setWhatId(accountId);

  Opportunity opportunity = new Opportunity();
        opportunity.setName("Opportunity");
        opportunity.setStageName("Prospecting");
        Calendar dt = connection.getServerTimestamp().getTimestamp();
        dt.add(Calendar.DAY_OF_MONTH, 7);
        opportunity.setCloseDate(dt);
        opportunity.setAccountId(accountId);

        sr = connection.create(new com.sforce.soap.enterprise.sobject.SObject[] { note,
task, opportunity } );

        if(sr[0].isSuccess()) {
           System.out.println("Successfully saved the note, task, and opportunity");


           com.sforce.soap.enterprise.QueryResult qr = connection.query("SELECT Id FROM
User WHERE FirstName = 'Jane' AND LastName = 'Doe'");
           String newOwnerId = qr.getRecords()[0].getId();
           account.setId(accountId);
           account.setOwnerId(newOwnerId);

           // Set owner change options so account's child note, task, and opportunity
transfer to new owner
```

```
        OwnerChangeOption opt1 = new OwnerChangeOption();
        opt1.setExecute(true);
        opt1.setType(OwnerChangeOptionType.TransferOwnedOpenOpportunities); // Transfer
Open opportunities owned by the account's owner

        OwnerChangeOption opt2 = new OwnerChangeOption();
        opt2.setExecute(true);
        opt2.setType(OwnerChangeOptionType.TransferOpenActivities);

        OwnerChangeOption opt3 = new OwnerChangeOption();
        opt3.setExecute(true);
        opt3.setType(OwnerChangeOptionType.TransferNotesAndAttachments);

        connection.setOwnerChangeOptions(new OwnerChangeOption[] {opt1, opt2, opt3});
           connection.update(new com.sforce.soap.enterprise.sobject.SObject[] { account }
 );

           // The account's note, task, and opportunity should be transferred to the new
owner.
        }

   } else {
       System.out.println("Account save failed: " + sr[0].getErrors().toString());
    }
}
```

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。

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] で指定されたパッケージのバージョンを使用します。[設定]から、［クイック検索］ボックスに「*API*」と入力し、[API]を選択して、［Enterprise パッケージバージョンの設定］にある [Enterprise パッケージバージョンの設定] をクリックします。

このヘッダーは、API バージョン 16.0 以降で使用できます。

関連する API コール

convertLead()、create()、delete()、describeGlobal()、describeLayout()、describeSObject()、describeSObjects()、describeSoftphoneLayout()、describeTabs()、merge()、process()、query()、retrieve()、search()、undelete()、update()、upsert()

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。パッケージバージョンは *majorNumber.minorNumber* のようになります (例: 2.1)。

項目

項目	型	説明
majorNumber	int	パッケージバージョンのメジャー番号。
minorNumber	int	パッケージバージョンのマイナー番号。
namespace	string	管理パッケージの一意の名前空間。

サンプルコード — Java

このサンプルでは、PackageVersionHeader を使用して 1 つのインストール済みパッケージのパッケージバージョンを設定します。次に、executeAnonymous Apex メソッドを使用してこのメソッドに渡されるコードを実行します。

```
public void PackageVersionHeaderSample(String code) throws Exception
{
    _PackageVersionHeader pvh = new _PackageVersionHeader();
    PackageVersion pv = new PackageVersion();
    pv.setNamespace("installedPackageNamespaceHere");
    pv.setMajorNumber(1);
    pv.setMinorNumber(0);
    // In this case, we are only referencing one installed package.
    PackageVersion[] pvs = new PackageVersion[]{pv};
    pvh.setPackageVersions(pvs);

    apexBinding.setHeader(new SforceServiceLocator().getServiceName().getNamespaceURI(),
        "PackageVersionHeader", pvh);
    // Execute the code passed into the method.
    ExecuteAnonymousResult r = apexBinding.executeAnonymous(code);
    if (r.isSuccess()) {
        System.out.println("Code executed successfully");
    }
    else {
        System.out.println("Exception message: " + r.getExceptionMessage());
        System.out.println("Exception stack trace: " + r.getExceptionStackTrace());
```

```
    }
}
```

QueryOptions

クエリの任意のバッチサイズを指定します。パフォーマンスを最大限に向上させるために、指定されたサイズより大きいまたは小さいバッチが作成される場合があります。

関連する API コール

`query()`、`queryMore()`、`retrieve()`

項目

要素名	型	説明
`batchSize`	int	`query()` コールまたは `queryMore()` コールで返されるレコード数のバッチサイズ。子オブジェクトは、バッチサイズのレコード数に含まれます。たとえば、リレーションクエリの場合、返される親行ごとに複数の子オブジェクトが返されます。 デフォルトおよび最大値は 2,000 で、最小値は 200 です。要求されるバッチサイズが、実際のバッチサイズになるとは限りません。パフォーマンスを最大化するために、返される結果の数は、レコードのサイズと複雑さに応じて変わることがあります。

サンプルコード

コード例については、『Salesforce SOQL および SOSL リファレンス』の「クエリのバッチサイズの変更」を参照してください。

SessionHeader

`login()` が正常に行われた後にログインサーバーから返される、セッション ID を指定します。このセッション ID はすべての後続コールで使用されます。

バージョン 12.0 以降では、このヘッダーに関連する SOAP メッセージに API 名前空間を追加します。名前空間は、Enterprise WSDL または Partner WSDL で定義されています。

API コール

ユーティリティ API コールを含むすべてのコール。

項目

要素名	型	説明
sessionId	string	login() コールで返され、後続のコール認証に使用されるセッション ID。

サンプルコード

login() の例を参照してください。

UserTerritoryDeleteHeader

メモ: 現在、元のテリトリー管理機能は使用できません。詳細は、「元のテリトリー管理モジュールを Summer '21 リリースで廃止」を参照してください。このトピックの情報は、元のテリトリー管理機能にのみ適用され、エンタープライズテリトリー管理には適用されません。

現在の所有者がテリトリーから削除された場合に進行中の商談が割り当てられるユーザーを指定します。このヘッダーが使用されない場合、またはこの要素の値が null の場合、該当のテリトリーの売上予測マネージャーに商談が転送されます。売上予測マネージャーが存在しない場合、テリトリーから削除されたユーザーが商談を保持します。

API コール

delete()

項目

要素名	型	説明
transferToUserId	ID	商談の所有者(ユーザー)がテリトリーから削除されたときに、そのユーザーのテリトリーの進行中の商談が割り当てられるユーザーの ID。

Salesforce 機能での API の使用

第 16 章　実装に関する考慮事項

トピック:

この章では、インテグレーションアプリケーションをはじめとするクライアントアプリケーションを開発する前に考慮すべき問題について説明します。

ほとんどの問題は、データ管理、使用制限、通信に関わるものです。

インテグレーション用のユーザーの選択

クライアントアプリケーションがAPIに接続するとき、最初にログインする必要があります。`login()` コール時にはSalesforceにログインするためのユーザーを指定する必要があります。クライアントアプリケーションは、ログインユーザーの権限および共有設定に基づいて動作します。以降のセクションでは、クライアントアプリケーション用にユーザーを構成する方法について説明します。

権限

組織のSalesforceシステム管理者は、プロファイルおよび権限セットを設定し、ユーザーをそれらに割り当てることによって、ユーザーが使用できる機能やビューを制御します。APIにアクセスし、コールを発行してコール結果を受信するには、ユーザーに「API対応」権限が与えられている必要があります。クライアントアプリケーションは、ログインユーザーの権限を介してアクセス権限を付与されるこれらのオブジェクトおよび項目のみを照会したり更新したりできます。

共有ルールによってデータへのアクセスを許可されたユーザーとしてのログインでは、APIはアクセスを確認する追加のクエリを発行する必要があります。それを避けるには、「すべてのデータの編集」権限を持つユーザーとしてログインします。これによって、コールの応答時間を短縮できます。「すべてのデータの編集」権限を許可すると、特定のユーザーには権限範囲が広すぎることがあります。その場合は、オブジェクトレベルの権限である「すべて変更」を使用して、データのアクセスをオブジェクト単位に制限することを検討してください。

制限

複数のクライアントアプリケーションが、同じ `username` 引数を使用してログインできます。ただし、この方法ではクエリ制限によってエラーになるリスクが高くなります。

複数のクライアントアプリケーションが同じユーザーによってログインした場合、そのアプリケーションすべてで同じセッションが共有されます。いずれかのクライアントアプリケーションが `logout()` をコールすると、すべてのクライアントアプリケーションのセッションが無効になります。クライアントアプリケーションごとに異なるユーザーを使用すると、こうした制限を回避しやすくなります。

メモ: ユーザー制限に加えて、組織ごとにAPI要求の制限があります。

ログインサーバーのURL

SOAP APIは、単一のログインサーバーを提供します。インスタンスをハードコード化することなく、単一のエントリポイントからどの組織へもログインできます。APIを介して組織にアクセスするには、まず、選択したWSDLに応じて次のどちらかのURLにあるログインサーバーに `login()` 要求を送信し、セッションの認証を受けます。

メモ: HTTPSは必須です。HTTPを使用する `login()` 要求、たとえば `https://login.salesforce.com/services/Soap/u/54.0` はサポートされません。

セッション中のすべての後続コールは、`login()` のレスポンスで返されたURLに対して実行する必要があります。このURLは組織のサーバーインスタンスを示しています。

ログインサーバーへのログイン

クライアントアプリケーションは、他のコールを呼び出す前にまず `login()` コールを呼び出してログインサーバーとのセッションを確立する必要があります。次に、返されたサーバーのURLを後続のAPI要求の要求先サーバーとして設定し、返されたセッションIDをSOAPヘッダーに設定して後続のAPI要求のサーバー認証が行えるようにする必要があります。Salesforceは、クライアントアプリケーションがログインしているIPアドレスを確認し、不明なIPアドレスからのログインをブロックします。詳細は、「`login()`」および「ステップ4:サンプルコードを説明する」を参照してください。

APIでブロックされたログインに関しては、Salesforceがログイン失敗エラーを返します。ログインするには、ユーザーがセキュリティトークンをユーザーパスワードの末尾に追加する必要があります。たとえば、パスワードが *`mypassword`* で、セキュリティトークンが *`XXXXXXXXXX`* の場合は、「*`mypasswordXXXXXXXXXX`*」と入力する必要があります。セキュリティトークンを取得するには、Salesforceユーザーインターフェースからパスワードを変更するか、セキュリティトークンをリセットします。ユーザーがパスワードを変更するか、セキュリティトークンをリセットすると、ユーザーのSalesforceレコードに指定されたメールアドレス宛に新しいセキュリティトークンが送信されます。セキュリティトークンは、ユーザーがセキュリティトークンをリセットするか、パスワードを変更するか、またはユーザーのパスワードがリセットされるまで有効です。トークンが無効な場合、ユーザーはログインプロセスを再度行う必要があります。再度ログインを行わないようにするには、クライアントのIPアドレスを組織の信頼できるIPアドレスのリストに追加します。

ログイン後は、APIコールを実行できます。それぞれの操作に対し、クライアントアプリケーションは同期要求をAPIに送信し、レスポンスを待ち、結果を処理します。APIは変更されたデータを自動的にコミットします。

APIコールには、次のようなものがあります。

- 基本となるAPIコール
- 記述用のAPIコール(describe)
- ユーティリティAPIコール

一般的なAPIコールシーケンス

各コールに対し、クライアントアプリケーションは一般的に次の処理を行います。

1. パラメーターを使用する場合、要求パラメーターを定義して要求の準備を行います。
2. 要求とパラメーターをLightning Platform Webサービスに渡し処理するためにコールを呼び出します。
3. APIからのレスポンスを受け取ります。
4. 返されたデータを処理するか(呼び出しが成功した場合)、エラーを処理し(呼び出しが失敗した場合)、レスポンスを処理します。

Salesforce Sandbox

Professional Edition、Enterprise Edition、Unlimited Edition、およびPerformance Editionを使用している場合、Salesforce Sandboxにアクセスできます。これは、Salesforce組織の本番データの全部または一部の複製を提供するテスト

環境です。詳細は、Salesforce コミュニティの Web サイト (www.salesforce.com/community) にアクセスするか、Salesforce ヘルプの「Sandbox の種別およびテンプレート」を参照してください。

API を介して組織の Sandbox にアクセスするには、次の URL からログイン要求を行います。

Salesforce データベースサーバーの複数インスタンス

通常組織は地理的な地域ごとに分けられていますが、組織はあらゆるインスタンスに置かれる可能性があります。

コンテンツタイプの要件

すべての要求には `Content-Type: text/xml; charset=utf-8` などの正しいコンテンツタイプの HTTP ヘッダーが含まれていなければなりません。

API 使用状況の測定

最適なパフォーマンスを維持し、すべてのお客様が Lightning Platform API を使用できるようにするために、Salesforce は次の 2 つ種類の制限を設けることによって、トランザクションの負荷を調整しています。

- 同時 API 要求数の制限
- 合計 API 要求の割り当て

コールが要求制限を超えると、エラーが返されます。

同時 API 要求数の制限

次の表は、20 秒以上の同時受信要求 (コール) 数について、さまざまな種類の組織に対する制限を示しています。

組織種別	制限
Developer Edition 組織とトライアル組織	5
本番組織と Sandbox 組織	25

実行時間が長い要求の数が制限を超えた場合、API は `REQUEST_LIMIT_EXCEEDED` 例外コードを返します。新しい同時要求の処理は、要求数が制限より少なくなるまで行われません。たとえば、本番環境では、実行時間が長い要求が 25 個未満になるまで、新しい同時要求は許可されません。

20 秒より短い同時要求の数に制限はありません。

合計 API 要求の割り当て

次の表は、組織の 24 時間あたりの受信 API 要求 (コール) 数の合計に関する制限について示しています。

Salesforce のエディション	24 時間あたりのライセンスの種類ごとの API コール数	24 時間あたりの合計コール数
Developer Edition	なし	15,000
• Enterprise Edition • Professional Edition (API アクセス有効)	• Salesforce: 1,000 • Salesforce Platform: 1,000 • Lightning Platform - One App: 200 • Customer Community: 0 • Customer Community Login: 0 • Customer Community Plus: 200 • Customer Community Plus Login: 10 • External Identity 25,000 SKU: 70,000 • External Identity 250,000 SKU: 750,000 • External Identity 1,000,000 SKU: 4,000,000 • Partner Community: 200 • Partner Community Login: 10 • Lightning Platform Starter: メンバーあたり 200 (Enterprise Edition 組織) • Lightning Platform Plus: メンバーあたり 1000 (Enterprise Edition 組織)	100,000 + (ライセンス数 x ライセンスの種類ごとのコール数) + 購入した API コールアドオン数
• Unlimited Edition • Performance Edition	• Salesforce: 5,000 • Salesforce Platform: 5,000 • Lightning Platform - One App: 200 • Customer Community: 0 • Customer Community Login: 0 • Customer Community Plus: 200 • Customer Community Plus Login: 10 • External Identity 25,000 SKU: 70,000 • External Identity 250,000 SKU: 750,000 • External Identity 1,000,000 SKU: 4,000,000 • Partner Community: 200 • Partner Community Login: 10 • Lightning Platform Starter: メンバーあたり 200 (Unlimited Edition 組織および Performance Edition 組織) • Lightning Platform Plus: メンバーあたり 5,000 (Unlimited Edition 組織および Performance Edition 組織)	100,000 + (ライセンス数 x ライセンスの種類ごとのコール数) + 購入した API コールアドオン数

Salesforce のエディション	24 時間あたりのライセンスの種類ごとの API コール数	24 時間あたりの合計コール数
Full Sandbox	なし	5,000,000 この制限は、テンプレートから作成されていない Full Sandbox にのみ適用されます。テンプレートから作成されたサンドボックスの場合は、テンプレートの値によって制限が決定されます。詳細は、*「Salesforce ヘルプ: Sandbox の種別およびテンプレート」*を参照してください。

Experience Cloud の制限については、「Experience Cloud ユーザーライセンス」を参照してください。

メモ: 負荷、パフォーマンスやその他のシステムの問題によっては、24時間すべてのコール割り当てを使用できなくなる場合があります。

この割り当てに含まれる API には、Lightning Platform REST API、Lightning Platform SOAP API、Bulk API、Bulk API 2.0 が含まれます。特定の Salesforce 接続アプリケーション (Salesforce モバイルアプリケーションなど) によって発行された API コールは計数されません。割り当てに影響を与える API を確認するには、「API 使用状況の監視」を参照してください。

DebuggingHeader を含むコールには、別途 24 時間あたり 1,000 コールの割り当て制限があります。組織の合計要求制限に達した後でも、これらのコールは引き続き実行できます。

コール数の制限および割り当ては、24 時間あたりに組織に対して行われた API コール数の集計に対して適用されます。この制限および割り当ては、ユーザーごとに適用されるものではありません。

API 使用状況の監視

組織の API 使用状況および制限をより詳細に監視するには、次のリソースを使用してください。

- [設定] の [システムの概要] ページの [API 使用状況] セクション。
- [設定] の [システムの概要] ページの [組織の詳細] セクションの [API 要求数 (この 24 時間以内)]。
- 使用量ベースのエンタイトルメントの [API Request Limit per Month (月間 API 要求制限)]。ここには、組織の API コールの 30 日間の集計が表示されます。[設定] の [組織情報] ページにあります。
- REST API の `Sforce-Limit-Info` 応答ヘッダーで返される情報。
- SOAP API の (`<type>API REQUESTS</type>` 内の) レスポンスボディで返される情報。
- Lightning Platform REST API の `/limits` コール。

API 要求が指定した割り当てコール数の割合を超えた場合に、メールで指定ユーザーに通知するように組織で設定できます。[設定] から、[クイック検索] ボックスに *「API 使用状況通知」* と入力し、[API 使用状況通知] を選択することで、この設定を実行します。

「App Development Without Limits (制限なしのアプリケーション開発)」 Trailhead モジュールの「Learn About Daily Rate Limits (1 日の転送制限について)」セクションも参照してください。

API 要求の制限に達すると、または制限を超えると何が起きるか

組織で 1 日の API 要求の制限に達するか、または制限を超えた場合、Salesforce は可能であれば、一定量の操作の続行を許可します。これにより、予期しないワークロードの急増やときどき発生するピーク期間においてワークフローがブロックされてしまうことを回避できます。プラットフォームリソースを保護し、API 要求が 1 日の上限を制限なく超えることがないようにするために、ハードキャップが設定されています。

メモ: 組織をホストしている Salesforce インスタンスの全体的な健全性を保護するために、通常の 1 日の制限の超過は常に制約の対象となります。(インスタンスの健全性は Salesforce Trust で監視できます。)

この機能は、ワークフローの中断を回避できるようにときどき使用するように設計されています。継続的にこの機能に依存することはしないでください。割り当てを増やすには、Salesforce の営業担当者にお問い合わせください。

この機能は、[有効] 状況の有料組織にのみ適用されます。トライアル組織、Developer Edition 組織、Sandbox 組織には適用されません。

API 要求活動が、契約開始日から始まる 30 日の期間で集計されます。この活動には、組織の制限を超えたコールが含まれます。

API 要求合計数の割り当ての増加

ユーザーライセンスに基づく API 要求数の計算は、ユーザー数に基づいて組織に十分な利用可能数を許可するように意図されています。要求数を増やす必要があるが、ユーザーライセンスの追加購入や Performance Edition へのアップグレードを希望しない場合は、API コールを追加購入できます。詳細は、営業担当者にお問い合わせください。

API コールを追加購入する前に、API 使用状況を精査します。クライアントアプリケーションが独自のエンタープライズアプリケーションであってもパートナーアプリケーションであっても、最適化によって、同じ処理を行うのに使用する API コールを減らせる場合があります。パートナー製品をお使いの場合、供給メーカーにお問い合わせいただき、その製品での API の使用が最適化されていることを確認してください。API の使用効率のよくない製品は、会社に不要なコストを負わせることになります。REST API 複合リソースを使用して、クライアントとサーバー間の往復回数を最小限に抑えることでアプリケーションのパフォーマンスを高めることができます。

API 使用制限の計算例

次の例は、API 使用制限の計算について、いくつかのシナリオを通して説明しています。

- Salesforce ライセンスを 15 個割り当てられている Enterprise Edition 組織の場合、要求数の制限は 115,000 件です (100,000 + 15 個のライセンス x 1,000 件のコール)。
- 水曜日の午前 5 時に 14,500 件のコールが作成され、水曜日の午後 11 時に 499 件のコールが作成された Developer Edition 組織の場合、木曜日の午前 5 時まで正常に作成できるコール数はあと 1 件だけです。

保存されたサードパーティの更新トークンとアクセストークンの長さ

Salesforce で保存されるサードパーティのアクセストークンおよび更新トークンの長さは、最大 10,000 文字です。

圧縮

API は、HTTP 1.1 の仕様で定義された標準を使用した要求とレスポンスの圧縮をサポートしています。いくつかの SOAP/WSDL クライアントでは自動的にサポートされており、他のクライアントへも手動で追加できます。クライアント別の詳細は、`https://developer.salesforce.com/page/Tools` を参照してください。

圧縮は、クライアントが明示的に指定しない限り使用されません。パフォーマンス向上のため、HTTP 1.1 の仕様に従ったクライアント側での圧縮のサポートをお勧めします。

クライアントが圧縮をサポートしていることを示すには、HTTP ヘッダー「Accept-Encoding: gzip, deflate」または同様のヘッダーを含める必要があります。クライアントのヘッダーで正しく指定されている場合、API はレスポンスを圧縮します。レスポンスには、「Content-Encoding: deflate」または「Content-Encoding: gzip」のいずれか適切な方が含まれます。「Content-Encoding: deflate」または「gzip」をヘッダーに含めることで、あらゆる要求を圧縮することができます。

ほとんどのクライアントは、たとえ企業内 LAN を使用している場合も、ある程度のネットワーク接続の制限があります。API は圧縮をサポートすることでパフォーマンスを向上します。ほとんどすべてのクライアントでは、レスポンスを圧縮することのメリットがあり、多くのクライアントでは要求の圧縮でもメリットがあります。API は HTTP 1.1 の仕様に従い deflate と gzip 圧縮をサポートします。

レスポンスの圧縮

API は、必要に応じてレスポンスを圧縮することができます。 クライアント側から Accept-Encoding ヘッダーで gzip または deflate 圧縮を指定した場合、レスポンスは圧縮されます。Accept-Encoding が指定された場合でも API ではレスポンスを圧縮する必要はありませんが、通常は圧縮されます。API がレスポンスを圧縮した場合、API は使用した圧縮アルゴリズムの名前で Content-Encoding ヘッダーも指定します (gzip または deflate のいずれか)。

要求の圧縮

クライアントは要求を圧縮することもできます。API は、処理前にすべての要求を展開します。クライアントは、適切な圧縮アルゴリズムの名前を記した Content-Encoding HTTP ヘッダーを送信しなければなりません。詳細は、以下を参照してください。

- Content-Encoding については、`www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11`
- Accept-Encoding については、`www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.3`
- Content Coding については、`www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.5`

メモ: WSC (Web Service Connector) を使用する Java クライアントに要求 SOAP 圧縮を実装するには、`Connection` オブジェクトのインスタンス化に使用する `Config` に対して `setCompression()` をコールします。

HTTP 永続接続

ほとんどのクライアントでは、HTTP 1.1 の永続接続を使用し、複数の要求のソケット接続を再利用したほうがパフォーマンスが向上します。永続接続は通常 SOAP/WSDL クライアントが自動的に処理します。詳細は、次の HTTP 1.1 の仕様を参照してください。

http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1

HTTP のチャンク

HTTP 1.1 を使用しているクライアントは、チャンクレスポンスを受け取ることがあります。チャンクは通常 SOAP/WSDL クライアントが自動的に処理します。

国際化と文字コード

API は Unicode または ISO-8859-1 のいずれかの文字を完全にサポートしています。文字コードは、組織で使用されている Salesforce インスタンスごとに異なります。組織が `ssl.salesforce.com` にログインしている場合、エンコードは `ISO-8859-1` です。その他のすべてのインスタンスは `UTF-8` を使用します。文字コードを判断するには、`describeGlobal()` をコールし、DescribeGlobalResult で返された `encoding` 値を確認します。

組織で ISO-8859-1 エンコーディングを使用している場合、API に送信されるすべてのデータを ISO-8859-1 で符号化する必要があります。有効な ISO-8859-1 の範囲外の文字は、切り捨てられるか、エラーとなります。

メモ: API レスポンスは、組織で使用されている文字コードに符号化されます (UTF-8 または ISO-8859-1)。どちらの場合も、符号化されたデータは通常、SOAP クライアントによって自動的に処理されます。

XML 準拠

API は XML をベースとしています。XML では、すべてのドキュメントが正しい形式であることが求められます。その要件の一部には、エスケープ文字などを使用してもある特定の Unicode 文字は XML ドキュメントでは許可されないと定められています。また、その他の文字も地域に応じたエンコードが必要です。通常、標準 SOAP クライアントまたは XML クライアントが処理します。クライアントは、一般的な XML エスケープシーケンスすべてを解析する機能を持ち、無効な XML 文字を渡さないようにする必要があります。

前述のとおり、文字によってはエスケープ文字などを使用していても無効です。無効な文字には、ペアになっていない Unicode サロゲートやその他のいくつかの Unicode 文字が含まれます。これらの文字は滅多に使用されず、どのデータでも通常は重要視されない制御文字であるため、多くのプログラムで問題となる場合があります。これらの文字は XML ドキュメントでは許可されていませんが、HTML ドキュメントでは許可されており、Salesforce データにも含まれている場合があります。無効な文字は、すべての API レスポンスから除外されます。

無効な文字:

- `0xFFFE`
- `0xFFFF`
- 制御文字 `0x0` から `0x19`。ただし、有効な次の文字を除く。`0x9`、`0xA`、`0xD`、タブ、改行、キャリッジリターン。

- 0xD800 - 0xDFFF、サロゲートペアを形成するために使用されている場合を除く

.NET、非文字列項目、および Enterprise WSDL

.NET と Enterprise WSDL を共に使用した場合、.NET は各非文字列項目に Boolean 型の項目を生成します。たとえば、MyDateField__c に日付値がある場合、.NET は MyDateField__cSpecified と呼ばれる Boolean 型の項目を生成します。

生成される項目値は、デフォルトでは false です。指定された項目値が false である場合、対応する元の項目の値は SOAP メッセージには含まれません。たとえば、通貨項目 annualRevenue の値をクライアントアプリケーションが生成した SOAP メッセージに含める前に、annualRevenueSpecified の値を true に設定する必要があります。

```
account.annualRevenue = 10000;
account.annualRevenueSpecified = true;
```

第 17 章 Apex 用のオブジェクト、SOAP API コール、およびヘッダー

以下に示す Salesforce オブジェクト、SOAP API コール、およびヘッダーは、Apex でデフォルトで使用できます。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。

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」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 60.0 を使用する場合、次のようにエンドポイント 60.0 を使用します。

```
https://MyDomain.salesforce.com/services/Soap/s/60.0
```

Apex では、次の Salesforce オブジェクトが使用可能です。

- ApexTestQueueItem
- ApexTestResult
- ApexTestResultLimits
- ApexTestRunResult

次の SOAP API コールを使用して Apex をリリースできます。

- `compileAndTest()`
- `compileClasses()`
- `compileTriggers()`
- `executeanonymous()`
- `runTests()`

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ーを含む Apex コードを実施します。

次の SOAP ヘッダーを Apex の SOAP API コールで使用できます。

- `DebuggingHeader`

- PackageVersionHeader

関連トピック:

Apex 開発者ガイド

第 18 章　アウトバウンドメッセージ

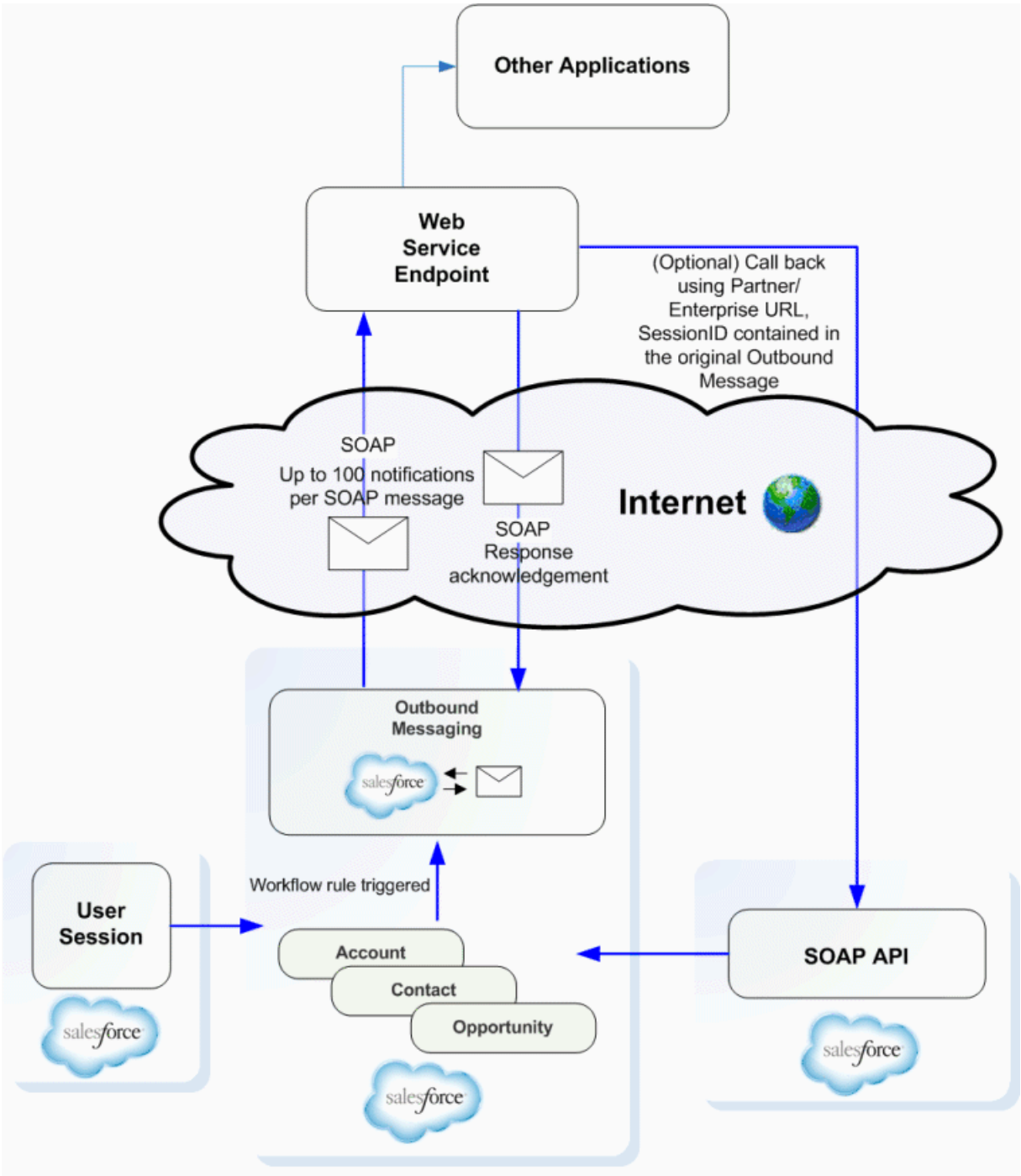
トピック:

アウトバウンドメッセージでは、Salesforce 内の項目に対する変更に基づいて、所定の外部サーバーに、項目値を含んだメッセージを送信するように指定することができます。

アウトバウンドメッセージは、Salesforceのワークフロールール機能の一部です。ワークフロールールは、項目に対する個々の変更に基づいて、メールアラートの送信、ToDo レコードの作成、アウトバウンドメッセージの送信など、自動的な Salesforce アクションをトリガーします。

アウトバウンドメッセージについて

アウトバウンドメッセージは `notifications()` コールを使用し、ワークフロールールでトリガーした場合に、SOAP メッセージを HTTP(S) を経由して指定されたエンドポイントに送信します。



アウトバウンドメッセージを設定した後、トリガーイベントが発生すると、メッセージが指定されたエンドポイントURLに送信されます。メッセージには、アウトバウンドメッセージを作成した際に指定された項目が含まれています。エンドポイントURLがメッセージを受信すると、メッセージから情報を取得して処理することができます。これを実行するには、アウトバウンドメッセージのWSDLを確認する必要があります。

通知について

1通のSOAPメッセージには、最大100件の通知を格納できます。それぞれの通知には、オブジェクトIDと、関連付けられたsObjectデータへの参照が記載されています。通知がキューされた後、送信される前にオブジェクト内の情報が変更された場合、最新の情報のみが配信され、途中の変更内容は配信されません。

複数のコールを発行すると、コールは1つ以上のSOAPにまとめられる場合があります。

メッセージはローカルでキューされます。各バックグラウンドプロセスでは実際の送信を実行します。メッセージの信頼性を保つため、次の処理が行われます。

- エンドポイントが利用できない場合、メッセージは正常に送信されるまで、または24時間が経過するまでキューに留まります。24時間を過ぎると、メッセージがキューから削除されます。
- メッセージが配信できない場合には、再試行の間隔が、最長2時間まで大幅に増えます。
- メッセージは、キュー内の順番とは無関係に再試行されます。そのため、メッセージが順番に配信されない場合もあります。
- 送信メッセージを使用して監査履歴を作成することはできません。各メッセージは通常1回配信されますが、2回以上配信される場合もあります。24時間以内に配信を行うことができない場合、メッセージはまったく配信されません。さらに、通知がキューされた後で送信される前に参照元オブジェクトが変更され場合、エンドポイントは最新のデータのみを受信し、途中の変更内容は受信しません。
- メッセージが複数回配信される場合があるため、処理を実行する前に、リスナークライアントに配信される通知の通知IDを確認してください。

メモ: Salesforceでイベントが発生した場合、以前のリリースで必要だったポーリングの代わりに、アウトバウンドメッセージを使用して実行ロジックをトリガーできます。以前のバージョンのAPIでは、クライアントアプリケーションはSalesforceにポーリングを行って関連する変更が行われたかどうかを確認する必要がありました。ルールが存在する場合、ほとんどの変更では結果としてワークフローがトリガーされます。ユーザーはワークフロールールを使用してSalesforceイベントに基づいてアクションをトリガーできます。

アウトバウンドSOAPメッセージを外部サービスに送信する`notifications()`コールの定義など、アウトバウンドメッセージに必要なメタデータは、個々のWSDLに含まれます。ワークフロールールがアウトバウンドメッセージに関連付けられると、WSDLがSalesforceユーザーインターフェースから作成され、使用可能になります。このWSDLはアウトバウンドメッセージにバインドされており、エンドポイントサービスへの到達方法に関する説明と、エンドポイントサービスに送信されるデータを記載しています。アウトバウンドメッセージの設定についての詳細は、「アウトバウンドメッセージの定義」を参照してください。

アウトバウンドメッセージの設定

アウトバウンドメッセージを使用するには、Salesforceユーザーインターフェースで次のような設定が必要になります。

ユーザープロファイルの設定

アウトバウンドメッセージでは、循環型の変更を作成できます。これにより、たとえば、インテグレーションによってあるワークフローをトリガーし、そのワークフローのアクションによって取引先の更新をトリガーし、その取引先の更新によって新しいワークフローをトリガーするというように、変更が次々にトリガーされます。こうした循環型の変更を回避するには、アウトバウンドメッセージを送信するユーザーの機能を無効にします。

次に、循環型の変更に関するもう 1 つのシナリオを示します。

1. アウトバウンドメッセージを設定して、sessionId を指定し、[送信ユーザー] 項目でユーザーを指定します。ユーザーはアウトバウンドメッセージを無効化しません。
2. 取引先責任者レコードを変更すると、指定されたユーザーからアウトバウンドメッセージ受信者へ、sessionId を含むアウトバウンドメッセージがトリガーされます。
3. アウトバウンドメッセージ受信者は Lightning プラットフォーム API を呼び出して、アウトバウンドメッセージをトリガーしたものと同じ取引先責任者のレコードを更新します。
4. 更新によりアウトバウンドメッセージがトリガーします。
5. アウトバウンドメッセージ受信者がレコードを更新します。
6. 更新によりアウトバウンドメッセージがトリガーします。
7. アウトバウンドメッセージ受信者がレコードを更新します。

ユーザーのアウトバウンドメッセージ通知を無効化するには、ユーザーの Profile で [アウトバウンドメッセージの送信] の選択を解除します。アウトバウンドメッセージに応答する 1 人のユーザーを指定し、このユーザーのアウトバウンドメッセージを無効化することをお勧めします。

アウトバウンドメッセージの定義

アウトバウンドメッセージを定義するには、Salesforce ユーザーインターフェースで次の手順を実行します。

1. [設定] から、[クイック検索] ボックスに「アウトバウンドメッセージ」と入力し、[ワークフローアクション] で [アウトバウンドメッセージ] を選択します。
2. [新規アウトバウンドメッセージ] をクリックします。
3. アウトバウンドメッセージに含める情報を持つオブジェクトを選択し、[次へ] をクリックします。
4. アウトバウンドメッセージを設定します。
   a. アウトバウンドメッセージの名前と説明を入力します。
   b. メッセージを受信するエンドポイント URL を入力します。Salesforce は、このエンドポイントに SOAP メッセージを送信します。

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。

c. [送信ユーザー] 項目でユーザー名を指定して、メッセージを送信する場合に使用する Salesforce ユーザーを選択します。選択したユーザーにより、エンドポイントに送信されるメッセージのデータのアクセス権が制御されます。

d. アウトバウンドメッセージに `sessionId` を含めるには、[送信セッション ID] を選択します。リスナーから Salesforce への API コールバックを作成する場合、`sessionId` をメッセージに含めます。`sessionId` は、ワークフローをトリガーしたユーザーではなく、前のステップで定義されたユーザーを表します。

e. アウトバウンドメッセージに含める項目を選択し、[追加] をクリックします。

5. [保存] をクリックし、アウトバウンドメッセージの詳細ページを確認します。
   - 現在では、アウトバウンドメッセージが作成されると、`[API バージョン]` 項目は自動的に現在の API バージョンに設定されます。この API バージョンは、Enterprise または Partner WSDL を使用した Salesforce への API コールバックで使用されます。`[API バージョン]` は、メタデータ API を使用してのみ変更できます。
   - [WSDL はこちら] をクリックすると、このメッセージと関連付けられている WSDL を参照できます。

     この WSDL はアウトバウンドメッセージにバインドされており、エンドポイントサービスへの到達方法に関する説明と、エンドポイントサービスに送信されるデータを記載しています。

メモ: これらのオプションが表示されない場合は、組織がアウトバウンドメッセージを有効にしていません。Salesforce に連絡して、組織のアウトバウンドメッセージを有効にします。

Salesforce クライアント証明書のダウンロード

アプリケーション (エンドポイント) サーバーの SSL/TLS を、クライアント証明書 (双方向 SSL/TLS) を要求するよう設定して、サーバーに対するクライアントのロールを取得する際に Salesforce の ID を検証することができます。Salesforce アプリケーションユーザーインターフェースから Salesforce クライアント証明書をダウンロードできます。この証明書は、Salesforce が認証のためにアウトバウンドメッセージと共に送信するクライアント証明書です。

1. [設定] から、[クイック検索] ボックスに「*API*」と入力し、[API] を選択します。
2. API WSDL ページで、[API クライアント証明書を管理] をクリックします。
3. [証明書と鍵の管理] ページの [API クライアント証明書] セクションで、[API クライアント証明書] をクリックします。
4. [証明書] ページで [証明書のダウンロード] をクリックします。ブラウザーで指定したダウンロード場所に .crt ファイルが保存されます。

   ダウンロードした証明書はアプリケーションサーバーにインポートし、そのサーバーがクライアント証明書を要求するように設定します。アプリケーションサーバーは SSL/TLS ハンドシェイクで使用する証明書がダウンロードした証明書と一致しているかを確認します。

メモ: アプリケーション(エンドポイント)サーバーは、証明書チェーンの中間証明書を送信する必要があります。そして証明書チェーンは、適切な順序でなければなりません。適切な順序は次のとおりです。

1. サーバー証明書
2. サーバーの証明書がルートの証明書による直接の署名を受けなかった場合にサーバーの証明書に署名を行った中間証明書
3. 手順 2 の中間証明書に署名を行った中間証明書
4. その他の残りの中間証明書

ルート証明書の認証機関の証明書は使用しないでください。ルート証明書はサーバーからは送信されません。Salesforce には、信頼できる証明書の一覧がファイルに収納されています。証明書チェーンには、リスト内の証明書のいずれかによって署名された証明書が含まれている必要があります。

アウトバウンドメッセージの参照

既存のアウトバウンドメッセージを表示するには、Salesforce ユーザーインターフェースで、[設定] から、［クイック検索］ボックスに「*アウトバウンドメッセージ*」と入力して、[アウトバウンドメッセージ] を選択します。

- アウトバウンドメッセージを新規に定義するには、[新規アウトバウンドメッセージ] をクリックします。
- アウトバウンドメッセージの状況を追跡するには、[メッセージ送信状況の参照] をクリックします。
- 既存のアウトバウンドメッセージの詳細や、既存のアウトバウンドメッセージを使用するワークフロールールと承認プロセスを表示するには、そのアウトバウンドメッセージを選択します。
- 既存のアウトバウンドメッセージに変更を加えるには、[編集] をクリックします。
- アウトバウンドメッセージを削除するには、[削除] をクリックします。

アウトバウンドメッセージの状況

アウトバウンドメッセージの状況を追跡するには、[設定] から、［クイック検索］ボックスに「*アウトバウンドメッセージ*」と入力して、[アウトバウンドメッセージ] を選択し、[メッセージ送信状況の参照] をクリックします。このページでいくつかの作業を実行できます。

- 送信の合計試行回数など、アウトバウンドメッセージの状況を参照する。
- ワークフローまたは承認プロセスの ID をクリックして、アウトバウンドメッセージをトリガーしたアクションを参照する。
- [次回の試行] 日を現在に変更するには、[再試行] をクリックします。このアクションにより、メッセージの配信がただちに再試行されます。
- アウトバウンドメッセージをキューから完全に削除するには、[削除] をクリックします。

セキュリティに関する考慮事項

アウトバウンドメッセージを使用するには、次のような対応を行って、Salesforceからの送信であるかのように見せかけたメッセージが第三者からエンドポイントに送信されることのないようにします。

- クライアントアプリケーションのリスナーを、Salesforce IP 範囲からの要求のみを受け入れるようロックダウンします。これにより、Salesforce からメッセージが送信されることが保証されます。ただし、別の顧客が誤ったエンドポイントをポイントしてメッセージを送信することを回避できるものではありません。Salesforce IP 範囲の最新のリストは、https://help.salesforce.com/articleView?id=000321501&type=1&mode=1 を参照してください。
- SSL/TLS を使用します。SSL/TLS を使用すると、データがインターネットで転送される間の機密保持を可能にします。SSL/TLS を使用しない場合、悪意のある第三者によってデータが侵害される危険性が生じます。この問題は、データの送信において個人情報保護が必要であり、メッセージで `SessionId` を渡す場合に特に重要です。また、接続の証明書を認証し、その証明書が有効な認証期間から発行されていることを確認し、証明書のドメインと Salesforce が接続しようとしているドメインと一致していることを確認します。この確認により、不正なエンドポイントとの通信を回避できます。
- **[送信セッション ID]** を選択する場合、セッション ID の安全な送信を確保するためにエンドポイント URL では HTTPS のみがサポートされます。Spring '19 より前に HTTPS エンドポイントを使用せずにこのオプションで作成された管理パッケージおよび未管理パッケージでも、登録者はインストールできます。Spring '19 以降、安全でないアウトバウンドメッセージオプションでパッケージを作成することはできません。
- アウトバウンドメッセージに含まれる `SessionId` の範囲は API 要求のみであり、UI 要求には適用されません。
- アプリケーション (エンドポイント) サーバーの SSL/TLS 設定が可能な場合、サーバーへのクライアントのロールを取得する際に、Salesforce クライアント証明書を使用して、Salesforce サーバーの ID を検証します。証明書のダウンロードの詳細は、「Salesforce クライアント証明書のダウンロード」を参照してください。
- 組織 `Id` が各メッセージに記載されています。ID データ型の詳細は、「ID データ型」を参照してください。クライアントアプリケーションで、メッセージに組織 `Id` が記載されていることを確認します。

送信メッセージの WSDL

このトピックの残りの部分は、送信メッセージの WSDL の関連セクションについて説明します。特定のオブジェクトの特定のイベントに送信メッセージを設定した際に選択した内容によって、ご使用の WSDL は異なります。

notifications()

このセクションでは、`notifications()` コールを定義します。このコールは特定のオブジェクトの指定された項目および値を記載した送信メッセージを作成し、指定されたエンドポイント URL に値を送信します。

```
<schema elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://soap.sforce.com/2005/09/outbound">
    <import namespace="urn:enterprise.soap.sforce.com" />
    <import namespace="urn:sobject.enterprise.soap.sforce.com" />

    <element name="notifications">
        <complexType>
            <sequence>
                <element name="OrganizationId" type="ent:ID" />
                <element name="ActionId" type="ent:ID" />
                <element name="SessionId" type="xsd:string" nillable="true" />
                <element name="EnterpriseUrl" type="xsd:string" />
                <element name="PartnerUrl" type="xsd:string" />
```

```
                    <element name="Notification" maxOccurs="100"
                      type="tns:OpportunityNotification" />
                </sequence>
            </complexType>
        </element>
    </schema>
```

次の表で、notifications メソッドの定義で指定される要素について説明します。

名前	型	説明
`OrganizationId`	ID	メッセージを送信する組織の ID。
`ActionId`	string	メッセージをトリガーするワークフロールール (アクション)。
`SessionId`	string	送信メッセージに応答するエンドポイント URL クライアントが使用するセッション ID (省略可能)。Salesforce にコールバックする受信コードによって使用します。
`EnterpriseURL`	string	Enterprise WSDL を使用して Salesforce に API コールバックするために使用する URL。
`PartnerURL`	string	Partner WSDL を使用して Salesforce に API コールバックするために使用する URL。
`Notification`	Notification	次のセクションで定義され、OpportunityNotification または ContactNotification など、オブジェクトデータ型と `Id` が指定されています。

Notification データ型は WSDL で定義されます。次の例では、`notifications()` コールの定義の Notification エントリに基づいて、商談の Notification が定義されます。

```
    <complexType name="OpportunityNotification">
        <sequence>
            <element name="Id" type="ent:ID" />
            <element name="sObject" type="ens:Opportunity" />
        </sequence>
    </complexType>
```

各オブジェクト要素 (今回の例では商談) には、送信メッセージを作成したときに選択した項目のサブセットが指定されます。各メッセージの Notification にはオブジェクト ID も指定されています。このオブジェクト ID を使用して、すでに処理された通知の再送信を追跡します。

notificationsResponse

この要素は、確認応答 (ack) を Salesforce に送信するためのスキーマです。

```
    <element name="notificationsResponse">
        <complexType>
            <sequence>
                <element name="Ack" type="xsd:boolean" />
```

```
                </sequence>
            </complexType>
        </element> //This section is the last in the types definition section.
```

複数の通知がある場合、メッセージ内ですべての通知を確認します。

リスナーの構築

送信メッセージを定義し、送信メッセージのエンドポイントを設定した後、WSDLをダウンロードしてリスナーを作成します。

1. **[WSDL はこちら]** を右クリックして、[名前を付けて保存] を選択し、適切なファイル名で WSDL をローカルディレクトリに保存します。たとえば、リードを扱う送信メッセージについて、WSDL ファイルに `leads.wsdl` という名前を付けます。
2. クライアントが Salesforce に送信するメッセージを記述する Enterprise WSDL や Partner WSDL と異なり、この WSDL は、Salesforce がクライアントアプリケーションに送信するメッセージを定義します。
3. 多くの Web サービスツールはスタブリスナーを生成します。多くは、Enterprise WSDL または Partner WSDL のクライアントスタブを生成する方法と同じです。サーバー側スタブオプションを検索します。

   たとえば .NET 2.0 の場合、次のようになります。

   a. .NET 2.0 で `wsdl.exe /serverInterface leads.wsdl` を実行します。このコマンドでは、通知インターフェースを定義する `NotificationServiceInterfaces.cs` を生成します。

   b. `NotificationServiceInterfaces.cs` を実行するクラスを作成します。

   c. インターフェースを実装するクラスを作成して、リスナーを実装します。実装するには、さまざまな方法があります。まずインターフェースを DDL にコンパイルすると最も簡単です (DLL は、ASP.NET の `bin` ディレクトリにある必要があります)。

   ```
   mkdir bin
   csc /t:library /out:bin\nsi.dll NotificationServiceInterfaces.cs
   ```

   このインターフェースを実行する ASMX ベースの Web サービスを作成します。たとえば、`MyNotificationListener.asmx` では次のようになります。

   ```
   <%@WebService class="MyNotificationListener" language="C#"%>
   class MyNotificationListener : INotificationBinding
   {            public notificationsResponse notifications(notifications n)
                {
                       notificationsResponse r = new notificationsResponse();
                       r.Ack = true;
                       return r;
                }
   }
   ```

   この例は単純な実装で、実際の実装はより複雑になります。

   d. `MyNotificationListener.asmx` を含むディレクトリの IIS の仮想ディレクトリを新規作成して、サービスを展開します。

e. ブラウザーでサービスページを参照して、サービスが展開されていることをテストすることができます。たとえば、仮想ディレクトリ `salesforce` を作成する場合、`http://localhost/salesforce/MyNotificationListener.asmx` にアクセスします。

その他の Web サービスツールのプロセスは類似しているため、Web サービスツールのマニュアルを参照してください。

リスナーは、次の要件を満たす必要があります。

- 公開インターネットから接続可能であること。
- 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。
- 有効にするために、証明書の一般名 (CN) はエンドポイントのユーザーのドメイン名に一致し、証明書は Java 2 Platform、Standard Edition (J2SE) 5.0 (JDK 1.5) で信頼された認証機関で発行されている必要があります。
- 証明書の有効期限が終了している場合、メッセージの送信ができません。

警告: 送信メッセージの無限ループによってさらに多くの送信メッセージをトリガーする変更がトリガーされることを防ぐには、オブジェクトを更新するユーザーに「送信メッセージの送信」権限が与えられていないことを確認してください。

第 19 章　データの読み込みとインテグレーション

トピック:

大容量のデータ (数十万から数百万件のレコード) を読み込む必要がある場合、考慮する必要のある要素が多数あります。このセクションのトピックを熟読して、クライアントアプリケーションのデザイン、組織の構成、およびデータローダーのベストプラクティスに関連する事項を確認してください。

クライアントアプリケーションのデザイン

大量のレコードを読み込むには Bulk API 2.0 が最適ですが、SOAP ベース API を使用することもできます。データ読み込み速度を改善するには、次のようなさまざまなアプリケーション設計方法があります。

- 永続接続を使用する。SSL/TLS ネゴシエーションからソケットステムを開く場合、時間がかかることがよくあります。SSL または TLS を使用しないと、API 要求は安全ではありません。HTTP 1.1 では、HTTP 1.0 のように要求ごとにソケットを開きなおす必要なく、要求間 (永続接続) でソケットを再利用することができます。使用するクライアントが永続接続をサポートしているかどうかは、使用している SOAP スタックによって異なります。デフォルトでは、.NET は永続接続を使用します。Apache http 共通ライブラリを使用するように設定を変更すると、クライアントは HTTP 1.1 仕様に準拠し、永続接続を使用します。

  HTTP 1.1 についての詳細は、http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1 を参照してください。

- 要求数を最小化する。クライアントが要求ごとにできる限り多くのレコードを一括処理する時間を節約するような、各要求に関連する処理がいくつかあります。batchSize を上限 2,000 に設定します。これが最も効率的なバッチサイズでない場合、API がバッチサイズを変更します。バッチサイズの詳細については、「QueryOptions」を参照してください。
- 要求サイズを最小化する。クライアントアプリケーションは、1 つの要求でできるだけ多くのレコードを送信する必要がありますが、一方で送信する要求をできるだけ小さくしてネットワーク送信時間を短縮する必要もあります。要求サイズを最小化するには、要求および応答で圧縮を使用します。Gzip が最も一般的な圧縮方法です。Lightning プラットフォーム開発者向けコミュニティボードには、さまざまな SOAP スタックで圧縮を実施する方法について解説している複数の投稿があります。Gzip の詳細な分析および検証については、Simon Fell のブログ (http://www.pocketsoap.com/weblog/2005/12/1583.html) を参照してください。
- マルチスレッドのクライアントアプリケーションを設計しない。SOAP ベース API を使用する単一クライアントアプリケーションでは、マルチスレッドは許可されません。

Salesforce の設定

ほとんどの処理はデータベース内で実行されます。これらのパラメーターを正しく設定することで、データベースの処理を高速化するのに役立ちます。

- MRU 機能を有効または無効にする。MRU (Most Recently Used) のマークが付けられたレコードは、Salesforce ユーザーインターフェースのサイドバーの [最近使ったデータ] セクションにリストされます。不要な場所でコールを有効にしていないことを確認してください。

  API バージョン 7.0 以降では、MRU 機能はデフォルトで無効になっています。MRU 機能を有効にするには、MruHeader を作成して `updateMru` を `true` に設定します。次のサンプルは、MRU 機能の使用方法を示しています。

```
public void mruHeaderSample() {
  connection.setMruHeader(true);
  Account account = new Account();
  account.setName("This will be in the MRU");
  try {
```

```
    SaveResult[] sr = connection.create(new SObject[]{account});
    System.out.println("ID of account added to MRU: " +
      sr[0].getId());
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  }
}
```

- 共有ルールを避けるために、「すべてのデータの編集」権限を持つユーザーとしてログインする。共有ルールによってデータへのアクセスを許可されたユーザーとしてのログインでは、APIはアクセスを確認する追加のクエリを発行する必要があります。それを避けるには、「すべてのデータの編集」権限を持つユーザーとしてログインします。通常、共有ルールが少ないほど所有者などのプロパティを設定する処理が少なくて済むため、読み込み速度が早くなります。

  あるいは、読み込み時間という点から、オブジェクトによっては組織の共有設定として公開/参照・更新可能などを設定することができます。詳細は、Salesforce オンラインヘルプの「組織の共有設定の設定」を参照してください。

- ワークフローまたは割り当てルールを避ける。操作後のアクションが必要な処理はすべて読み込みに時間がかかります。読み込まれたオブジェクトに適用されない場合、自動ルールを一時的に無効にすることもできます。
- カスケード更新のトリガーを避ける。たとえば、取引先の所有者を更新した場合、その取引先に関連付けられた取引先責任者と商談にも更新が必要です。単一のオブジェクトを更新する代わりに、クライアントアプリケーションは複数のオブジェクトにアクセスしなければならず、読み込みが遅くなります。

Lightningプラットフォームデータローダーは、データ読み込みのよい参考となります。MRUを無効にし、HTTP/1.1永続接続を使用し、要求とレスポンスにGZIP圧縮を使用します。データ読み込みを行う場合、または独自のJavaインテグレーションを記述する際にどこから始めればよいか分からない場合、Lightningプラットフォームデータローダーを使用し高速で安定したソリューションを実現することができます。Lightningプラットフォームデータローダーについての詳細は、Salesforceオンラインヘルプの「データローダー」を参照してください。

すべてのデータローダーでのベストプラクティス

このセクションではLightningプラットフォームデータローダーを使用したベストプラクティスについて説明しますが、基本的な原則はどのクライアントデータローダーにも当てはまります。

1. 移行するデータを特定します。

   データセット全体を移行する必要がないのであれば、移行対象のオブジェクトを指定します。たとえば、各取引先の取引先責任者情報のみ、特定のディビジョンの取引先情報のみ、といったように移行する対象を特定します。

2. データのテンプレートを作成します。

   オブジェクトごとのテンプレートを、Excelワークシートなどで作成します。

   各オブジェクトで必要な項目を指定します。標準オブジェクトごとに必要な項目を指定するだけではなく、ビジネスルールや過去のID項目などで必要な項目もあります。このガイドやSalesforceユーザーインターフェースのページレイアウト定義などを参考に、標準オブジェクトで必要な項目を見つけ出します。

   必要な項目を赤で強調表示すると、テンプレートに入力した後のデータ確認が容易になります。

順序の依存関係がある場合は指定します。オブジェクトには必須の関係が存在することがあります。たとえば、すべての取引先には所有者が存在し、またすべての商談は取引先に関連付けられています。これらの関係の依存性により、データ移行の順序が決定します。たとえば、Salesforce データではまずユーザーを読み込み、次に取引先、その後に商談を読み込みます。

依存関係を特定するには、指定されたオブジェクトのページレイアウトの関連リストと参照項目を確認し、そしてデータベースの ID (外部キー) を確認します。

3. テンプレートに入力します。

   テンプレートにデータを入力する前にデータのクリーンアップを行います。入力後はテンプレートでデータを確認します。

4. データを移行します。

   過去の ID 情報を格納するカスタム項目を作成します。必要に応じて、カスタム項目に [外部 ID] 属性を割り当てると、インデックスが付けられます。インデックスは関係を保持し、検証のためのカスタムレポートの構築に役立ちます。

   レコードを 1 つ読み込みて結果を確認してから、全レコードを読み込みます。

5. データを検証します。

   移行を検証するために、下記のすべての手法を使用します。

   - レコード件数を検証し、移行のスナップショット全体を提供するカスタムレポートを作成します。
   - データのスポットチェックを行います。
   - 例外レポートを参照し、移行されなかったデータを確認します。

6. 必要に応じてデータの再移行または更新を行います。

インテグレーションとシングルサインオン

警告: 復旧が不可能な状態に陥ることを避けるため、システム管理者アカウントのシングルサインオンを有効にしてはなりません。シングルサインオンを有効にしており、シングルサインオンのインテグレーションが失敗した場合、復旧のためのログインが不可能になります。

第 20 章　データ複製

トピック:

API はデータ複製をサポートしており、組織の関連する Salesforce データを外部で個別に複製して格納、保持します。複製されたデータは、データウェアハウス、データマイニング、カスタムレポート作成、分析、他のアプリケーションとのインテグレーションなど、特定の用途に使用できます。データ複製により、ローカルのコントロールが可能になるほか、ネットワークですべてのデータを送信することなく、データセット全体を対象に大規模な分析クエリや個別的な分析クエリを実行できるようになります。

メモ: Salesforce レコードの変更通知をリアルタイムで受け取るには、代わりに変更データキャプチャを使用してください。変更データキャプチャチャネルに登録することで、挿入、更新、削除、復元など、レコードの変更に対応した一連の変更イベントメッセージを受け取ることができます。変更データキャプチャにより、データへの幅広いアクセスが可能になり、トランザクション境界を使用して対象ストアで更新を行うことができます。変更データキャプチャは、バージョン管理されたイベントスキーマを提供し、変更イベントを一時的に保持することで後で取得できるようにします。詳細については、「変更データキャプチャの基礎」 Trailhead モジュールを参照してください。完全なリファレンスは、『変更データキャプチャ開発者ガイド』に記載されています。

このセクションのトピックを熟読して、データ複製のベストプラクティスについて理解を深めてください。

データ複製のための API コール

API は、次の API コールを使用したデータ複製をサポートしています。

API コール	説明
getUpdated()	特定のオブジェクトを対象に、指定された期間内に更新された(また追加、変更された) オブジェクトのリストを取得します。
getDeleted()	特定のオブジェクトを対象に、指定された期間内に削除されたオブジェクトのリストを取得します。

クライアントアプリケーションからこれらの API コールを呼び出し、組織内のデータの中で、指定された期間内で更新または削除されたオブジェクトを確認できます。これらの API コールは更新された(または追加、変更された) オブジェクトの ID のセットを返します。同様に、最後に更新された時間または削除された時間を示すタイムスタンプ (ローカル時間ではなく、協定世界時 (UTC)) を返します。クライアントアプリケーションはこれらの結果を処理し、必要な変更をデータのローカルコピーに適用します。

データ複製の範囲

この機能は、データ複製を目的としたメカニズムを提供します (一方向へのデータのコピー)。この機能は、データの同期 (双方向へのデータのコピー) やデータのミラーリングは提供しません。

データ複製手順

次に、オブジェクトの一般的なデータ複製手順を示します。

1. (必要に応じて実行) 直近の複製要求以降、オブジェクトの構造が変更されたかどうかを確認します (「オブジェクトの構造変更のチェック」を参照)。
2. getUpdated() をコールし、データを取得するオブジェクトと期間を渡します。

   getUpdated() は、ログインユーザーがアクセス権限を持つデータの ID を取得します。ユーザーの共有モデル外のデータは取得されません。API は、ユーザーが参照可能なオブジェクトの中で変更されたすべてのオブジェクトの ID を、変更内容に関わらず返します。ID についての詳細は、「ID データ型」を参照してください。
3. すべての ID を配列に挿入します。配列の各 ID 要素に対し、retrieve() をコールして関連するオブジェクトから必要な最新情報を取得します。その後、新しい行の挿入や既存の行の最新情報への更新などローカルデータに適切な処理を行います。
4. getDeleted() をコールし、データを取得するオブジェクトと期間を渡します。getUpdated() と同様に、getDeleted() は、ログインユーザーがアクセス権限を持つデータの ID を取得します。ユーザーの共有モデル外のデータは取得されません。API は、ユーザーが参照可能なオブジェクトの中で変更されたすべてのオブジェクトの ID を、変更内容に関わらず、使用可能な場合は SystemModstamp 項目情報に基づいて返します。ID についての詳細は、「ID データ型」を参照してください。

5. ID の配列すべてに対して反復処理を行います。その後、クライアントアプリケーションは削除されたオブジェクトをローカルデータからも削除 (または削除済みのフラグを設定) し、適切な処理を行います。クライアントアプリケーションが、取得したオブジェクト ID に一致する行をローカルデータに見つけられなかった場合、ローカルデータの行が削除されているか、作成されていなかったということになり、処理は実行されません。
6. 必要に応じて、今後の参照のために要求の期間を保存します。 これは、`getDeleted()` の latestDateCovered 値または `getUpdated()` の latestDateCovered 値を使用して行うことができます。

データ複製でのオブジェクト固有の要件

API オブジェクトは、データ複製について次の条件を満たさなければなりません。

- `getUpdated()` および `getDeleted()` コールは、ログインユーザーがアクセス権を付与されている、作成または更新されたオブジェクトの ID のみを受け取るよう、結果を絞り込みます。ID についての詳細は、「ID データ型」を参照してください。
- クライアントアプリケーションは、適切な権限が付与されている場合、任意のオブジェクトを複製できます。たとえば、組織のすべてのデータを複製するには、クライアントアプリケーションは「すべてのデータの参照」権限でログインしなければなりません。詳細は、「データアクセスに影響する要素」を参照してください。
- ログインユーザーは、オブジェクトの参照権限を付与されている必要があります。詳細は、Salesforce ヘルプの「内部組織の共有設定の設定」を参照してください。
- オブジェクトは複製可能として設定 (`replicateable` を `true` に設定) しなければなりません。特定のオブジェクトが複製可能かどうかを判断するには、アプリケーションでオブジェクトに対する `describeSObject()` コールを呼び出して、describeSObjectResult の `replicateable` プロパティを調べます。

変更のポーリング

クライアントアプリケーションは、定期的にデータ変更のポーリングを行います。ポーリングについては、次の点を考慮する必要があります。

- ポーリングの頻度は、組織の Salesforce データのローカルデータへの適用の時差の許容範囲、という業務上の要件により異なります。日に一度のポーリングで十分なクライアントアプリケーションもあれば、精度の高いデータを得るために 5 分ごとにポーリングを実施する必要があるクライアントアプリケーションもあります。
- 削除されたレコードは、`getDeleted()` からアクセス可能な削除ログに出力されます。2 時間ごとに実行されるバックグラウンドプロセスは、削除ログのレコード数が制限を超えた場合、削除ログに書き込まれてから 2 時間以上経過したレコードを消去します。最も古いレコードから順に、削除ログが制限を下回るまで消去を行います。このような処理を行うのは、大量の削除ログによる Salesforce のパフォーマンス上の問題を防ぐためです。制限値は次の数式を使用して算出します。

```
5000 * number of licenses in the organization
```

たとえば、1,000 ライセンスを所有する組織では、削除ログのレコード数が 5,000,000 (5 百万) レコード以上になると消去処理を開始します。`getDeleted()` コールを実行する前に消去処理を実行すると、

INVALID_REPLICATION_DATE エラーが返されます。この例外が発生した場合、テーブル全体に対するプル処理を実行する必要があります。

- API は、dateTime 値の秒の値を切り捨てます。たとえば、クライアントアプリケーションが 12:30:15 から 12:35:15 (協定世界時 (UTC)) までの時間を送信した場合、API は、12:30:00 から 12:35:00 (UTC) までに変更された項目 (両端の値を含む) の情報を取得します。

  メモ: 時間データの処理方法は、開発ツールごとに異なります。開発ツールによってはローカル時間を表示するものも、協定世界時(UTC)を表示するものもあります。開発ツールごとの時間の処理方法はツールのドキュメントを参照してください。

- ポーリングの頻度は、5分以上に設定することをお勧めします。アプリケーションの不備によりデータ複製の API コールが頻繁に実行されることがないようにするための制御機能が組み込みで実装されています。
- クライアントアプリケーションは、以前のデータ複製の API コールで使用した期間を保存する必要があります。それにより、データ複製が最後に正常に実行された時間をアプリケーション側で把握することができます。
- ローカルデータの整合性を確保するために、クライアントアプリケーションはポーリング中の関連する変更すべてを取得し、差異がないようにする必要があります(これにより、データを重複して処理しなければならないこともあります)。クライアントアプリケーションには、ローカルデータにすでに統合済みのデータの処理を避けるためのビジネスロジックを組み込むことができます。
- ハードウェア障害や接続エラーなど何らかの理由でクライアントアプリケーション側でデータのポーリングが失敗した場合も、データに差異が生じる場合があります。最後に正常に実行されたデータ複製とポーリングを確認し、次の処理期間を設定するためのビジネスロジックを、クライアントアプリケーションに組み込むことができます。
- 何らかの理由でローカルデータに変更が加えられた場合、ローカルデータを一から構築しなおすためのビジネスロジックを、クライアントアプリケーションに組み込むこともできます。

メモ: ここで Outbound Messaging を使用し、ポーリングの代わりにアクションをトリガーすることも可能です。

オブジェクトの構造変更のチェック

API では、データの複製はオブジェクトレコードに対して行われた変更のみを反映します。オブジェクトの構造に変更が行われたかどうかは確認しません(たとえば、カスタムオブジェクトでの項目の追加や削除など)。指定されたオブジェクトの構造が最新の更新以降変更されたかどうかを確認するのは、クライアントアプリケーションです。データを複製する前に、クライアントアプリケーションはオブジェクトに対して `describeSObjects()` をコールし、DescribeSObjectResult で返されたデータと、以前の `describeSObjects()` 呼び出しで返され、保存されたデータを比較できます。

第 21 章　機能固有の考慮事項

トピック:

API を使用してアクセスする場合、一部の Salesforce 機能では次のような特別な考慮事項が必要です。このセクションのトピックを熟読して、活動、個人取引先、売上予測上書きのビジネスルール、コールセンター、独自のアプリケーション作成についての特別な考慮事項について理解を深めてください。

アーカイブ済みの活動

Salesforce は、1 年以上経過した活動 (ToDo や行動) をアーカイブします。

アーカイブされているかどうかにかかわらず、すべての ToDo と行動のレコード上では、`queryAll()` でクエリを実行できます。また、アーカイブ済みのオブジェクトのみを探すために、`isArchived` 項目をで絞り込みを行うことが可能です。`query()` は、`isArchived` が `true` に設定されている場合、すべてのレコードを自動的に除外するため、query() は使用できません。アーカイブ済みのレコードの更新や削除は可能ですが、`isArchived` 項目を更新することはできません。API を使用して下記に示す条件を満たす活動を挿入すると、アーカイブのバックグラウンドプロセスを次に実行するときに活動がアーカイブされます。

古い行動や ToDo は、下記の基準に従ってアーカイブされます。Salesforce ユーザーインターフェースでは、ユーザーはアーカイブ済みの活動をいくつかの場所で参照できます。

- [活動履歴] 関連リストの [すべて表示] をクリックして [活動履歴] タブを開きます。[活動履歴] タブでは、エントリを並び替えたり、活動を開いたり、活動を編集または削除したりできます。
- 活動タイムラインの [すべて表示] をクリックして [すべての活動履歴] リストを開きます。アーカイブ済みレコードを含めて最大2,000件のレコードが表示されます。[すべての活動履歴] リストは、印刷に最適です。

API では、アーカイブ済み活動は `queryAll()` を使用してのみ照会することができます。

活動のアーカイブの条件:

- 365 日以上の `ActivityDateTime` または `ActivityDate` 値を持つ行動
- `IsClosed` の値が `true` であり、365 日以上の `ActivityDate` の値を持つ ToDo
- `IsClosed` の値が `true` であり、`ActivityDate` 項目が空白で、作成日が 365 日以上前の ToDo

詳細は、Salesforce ヘルプの「アーカイブ済み活動の表示」を参照してください。

個人取引先のレコードタイプ

個人取引先レコードタイプを使用すると、個人に対して営業、取引を行う B2C 機能を使用できます。たとえば、医師、美容師、不動産業者など、個人をクライアントとする業種でこうした機能を使用します。

API バージョン 8.0 以降、Account オブジェクトレコードタイプの新しいファミリー「個人取引先」レコードタイプが使用可能になりました。個人取引先についての詳細は、Salesforce ヘルプの「個人取引先」および「個人取引先を使用する場合の考慮事項」を参照してください。

Account 項目 `IsPersonAccount` が `true` に設定されている場合、レコードタイプは個人取引先レコードタイプとなります。Salesforce では 1 つのデフォルトの個人取引先レコードタイプ、PersonAccount がありますが、管理者は、追加の個人取引先レコードタイプを作成できます。それに対し、Account 項目 `IsPersonAccount` を `false` に設定しているレコードタイプは「法人取引先」レコードタイプで、通常のビジネス対ビジネス (B2B) の Salesforce 取引先です。

個人取引先が作成されると (または既存の法人取引先が個人取引先に変更されると)、対応する取引先責任者レコードも作成されます。この取引先責任者レコードは、「個人取引先責任者」といいます。個人取引先責任者を使用すると、個人取引先は、取引先および取引先責任者と同時に機能することができます。このレコードは、個人取引先と直接関連付けることができる唯一の取引先責任者レコードです。また、対応する個人取引先責任者レコードの ID は、個人取引先の `PersonContactId` 項目に保存されます。

この後、個人取引先レコードタイプを使用するにあたって注意すべき点をリストします。この機能を実際に使用する前に確認してください。

- 個人取引先機能がまだ有効でない場合、有効にするには、アカウントエグゼクティブにお問い合わせください。
- 次の例のようなクエリを使用して、個人取引先レコードタイプのすべてのレコードを検索することができます。

```
SELECT Name, SobjectType,IsPersonType
  FROM RecordType
  WHERE SobjectType='Account' AND IsPersonType=True
```

- 取引先に対して `query()` コールを発行する場合、結果は `SojbectType` 項目のルートオブジェクト種別を返します。返される値は常に Account になります。
- 個人取引先は変更できますが、作成または削除することはできません。これらの種類の取引先責任者には独自のレコード詳細ページがないため、クライアントは、ユーザーを対応する個人取引先 (Account) ページにリダイレクトする必要があります。SOSL の結果には、`IsPersonAccount` が `true` に設定されている場合に使用できる取引先責任者項目は含まれません。取引先責任者の `ReportsToId` 項目は参照できません。
- 取引先を削除すると、取引先責任者も削除されます。取引先責任者を直接削除することはできません。取引先を削除する必要があります。
- レコードタイプファミリー全体で取引先のレコードタイプを変更することができます(通常、法人取引先から個人取引先へ移行する場合に実行され、また逆の操作もサポートされています)。レコードタイプを法人取引先から個人取引先に変更すると、個人取引先が作成されます。個人取引先から法人取引先にレコードタイプを変更すると、個人の項目が null に設定され、個人取引先責任者が、変更前に同じ親取引先であった通常の取引先責任者となります。

  メモ: Salesforce ユーザーインターフェースではレコードタイプファミリーのレコードタイプを変更することはできません。

- 法人取引先のレコードタイプを、`update()` または `upsert()` のいずれかを使用して、個人取引先に変更する場合、同じコールの該当する取引先の項目に別の変更を行うことはできません。変更しようとすると、エラー `INVALID_FIELD_FOR_INSERT_UPDATE` が発生します。ただし、個人取引先レコードタイプ間、または法人取引先レコードタイプ間の変更であれば、値の変更を同じコール内の他の変更と同時に実行することができます。
- 法人取引先から個人取引先に変換する場合、それぞれの法人取引先レコードと、対応する取引先責任者レコードとの間に一対一のリレーションが必要です。さらに、`Owner` や `Currency` などの 2 つのレコードに共通する項目は、同じ値を指定する必要があります。
- ワークフローおよび入力規則数式は、個人取引先との間でレコードタイプの変更を行っている間は起動しません。
- 法人取引先を個人取引先に変更する場合、有効なレコードは変更され、無効なレコードについては結果配列にエラーが表示されます。
- 個人取引先を法人取引先に変更する場合、入力規則は実行されません。
- バージョン 7.0 以前の `describeLayout()` は、タブのデフォルトが個人取引先レコードタイプである場合も、デフォルトのレコードタイプとして法人取引先レコードタイプを返します。バージョン 8.0 以降では、デフォルトのレコードタイプは常にタブのデフォルトとなります。

- バージョン 7.0 以前では、describeLayout() は個人取引先レコードタイプは一切返しません。
- バージョン 7.0 以前の describeSObject() では、プロファイルに法人取引先レコードタイプへのアクセス権限がないため、作成できないものとして Account オブジェクトが表示されます。
- 変換後、個人取引先には、作成された取引先責任者レコードとの一意の一対一リレーションがあります。これはすべての個人取引先に当てはまるため、別の取引先責任者を個人取引先に関連付けることはできません。
- 変換後、既存の取引先項目の履歴情報は個人取引先に残ります。既存の取引先責任者項目の履歴情報は取引先責任者に保持されますが、個人取引先項目の履歴には追加されません。

個人取引先に関する詳細は、Salesforce ヘルプを参照してください。

外部オブジェクト

外部データに対する queryAll() および queryMore() コールには特殊な動作と制限が適用されます。

queryAll()

Salesforce では外部データへの変更を追跡しないため、queryAll() コールの動作は、外部オブジェクトに対する query() と同じです。

queryMore()

外部データの Salesforce Connect クエリで、大量の結果セットがバッチやページに分割されて示されることはよくあります。外部オブジェクトを照会するとき、Salesforce Connect は Web サービスコールアウト経由で外部データにリアルタイムにアクセスします。queryMore() コールを実行するたびに、Web サービスコールアウトが行われます。バッチの区切りとページサイズは、アダプターと外部データ取得元の設定方法に応じて異なります。

次の設定をお勧めします。

- 可能ならば、外部オブジェクトのクエリを、デフォルトのバッチサイズである 500 行よりも少ない行を返すように絞り込んでページングを避けます。バッチを取得するたびに queryMore() コールが必要になり、その結果、Web サービスコールアウトが行われます。
- 外部データが頻繁に変更される場合は、queryMore() コールの使用を避けてください。次の queryMore() コールまでの間に外部データが変更された場合、予期しない QueryResult になることがあります。

SELECT ステートメントの主オブジェクト(「主導」オブジェクト)が外部オブジェクトの場合、queryMore() は主オブジェクトのみをサポートし、サブクエリをサポートしません。

デフォルトでは、Salesforce Connect の OData 2.0 および 4.0 アダプターは、クライアント駆動のページングを使用します。クライアント駆動ページングにより、OData アダプターは各 queryMore() コールを、$skip および $top システムクエリオプションを使用してバッチの区切りとページサイズを指定する OData クエリに変換します。これらのオプションは、LIMIT および OFFSET 句を使用した結果セットのページ処理に似ています。

外部データソースのサーバー駆動ページングを有効にすると、要求されたページサイズ (デフォルトの queryMore() バッチサイズの 500 行を含む) が Salesforce で無視されます。バッチは、外部システムによって返されるページで決まりますが、各ページは 2,000 行を超えることはできません。

コールセンターと API

API は、`describeSoftphoneLayout()` コールで、コンピューターテレフォニーインテグレーション (CTI) コールセンターについての情報へのアクセスを提供します。組織で CTI 機能を有効にする必要があります。サポートが必要な場合はセールスフォース・ドットコムの担当者までお問い合わせください。

API は、コールセンター関連のオブジェクトへのアクセスを制限付きでサポートします。これにより、コールセンターの作成、コールセンターの追加番号の作成・更新などが可能になります。

トピック	説明
CallCenter	項目や使用方法など、コールセンターオブジェクトの説明。
AdditionalNumber	ユーザー、取引先責任者、リード、取引先、またはその他のオブジェクトとして容易に分類できない場合、追加番号を追加できるようにする構成設定。例には、電話のキューまたは会議室が含まれます。

また、いくつかの項目が既存オブジェクトに追加され、コールセンターをサポートします。次の項目では、コールセンターを操作するための構成設定を指定します。

オブジェクト名	項目名	項目の型	項目のプロパティ	説明
OpenActivity ActivityHistory Task	`CallDisposition`	string	Create (Task のみ) Filter Nillable Update (Task のみ)	「折り返しします」、「呼び出しに失敗しました」など、通話の結果を示します。最大 255 文字です。 Task オブジェクトの場合、Salesforce ユーザーインターフェースのラベル「通話結果」に対応しています。Task については、この項目の値を作成および更新できます。
OpenActivity ActivityHistory Task	`CallDurationInSeconds`	int	Create (Task のみ) Filter Nillable Update (Task のみ)	通話の時間 (秒単位)。 Task については、この項目の値を作成および更新できます。
OpenActivity ActivityHistory	`CallObject`	string	Filter Nillable	コールセンターの名前。最大 255 文字です。

オブジェクト名	項目名	項目の型	項目のプロパティ	説明
Task			Update (Task のみ)	Task については、この項目の値を作成および更新できます。
OpenActivity ActivityHistory Task	`CallType`	picklist	Create (Task のみ) Filter Nillable Restricted picklist Update	応答する通話の種類 (受信、内線、発信)。 Task については、この項目の値を作成および更新できます。
User	`CallCenterId`	reference	Create Filter Nillable Update	このユーザーに関連するコールセンターの一意の識別子。
User	`UserPermissionsCallCenterAutoLogin`	boolean	Create Update	Salesforce アプリケーションにログインする際にコールセンターに自動的にログインするか (`true`)、ログインしないか (`false`) を示します。

Lightning プラットフォームでの Salesforce インテグレーションの実行

Salesforce AppExchange アプリケーションを作成して、Lightning プラットフォームで Salesforce のインテグレーションまたはその他のクライアントアプリケーションを実行できます。

1. 外部サイトにユーザーセッション ID と API サーバー URL を渡す WebLink を作成します。

```
https://www.your_tool.com/test.jsp?sessionid={!API_Session_ID}&url={!API_Partner_Server_URL_80}
```

   `https` を使用して、セッション ID が検出できないようにします。

2. 上述の手順で参照されるページはセッション ID を取得し、それを使用して API にコールバックします。`getUserInfo()` を使用して、セッションや関連情報に関連する `userID` を返します。必要に応じて、User オブジェクトで retrieve を使用して、ユーザーに関して必要な追加情報を取得します。

3. `UserId` またはユーザー名とシステム内の対応するユーザー ID との相互参照を保持します。ユーザーがタブをクリックすることで実行される WebLink、ページレイアウト上の WebLink を利用できます。
4. Salesforce ヘルプのトピック「配布用アプリケーションの準備」の指示に従って、このアプリケーションをパッケージ化およびアップロードします。

API および OAuth を使った Salesforce データへのアクセス

Salesforce では、SOAP API 要求で OAuth 1.0.A および 2.0 がサポートされています。

すでに定義されている接続アプリケーションと OAuth プロトコルを使用することで、サードパーティは OAuth 認証フローを実装して Salesforce API と統合する琴ができます。

OAuth を使用した Salesforce API との統合の詳細な手順については、Salesforce ヘルプの「OAuth によるアプリケーションの認証」を参照してください。

認証用の OAuth コンシューマー ID の取得を希望するパートナー様は Salesforce まで連絡してください。

ナレッジ

記事の概要

重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

*記事*には、知識ベースで入手できるようにする会社の商品およびサービスに関する情報が取り込まれます。知識ベースの記事をデータカテゴリを使用して分類することにより、ユーザーは必要な記事を見つけやすくなります。管理者は、データカテゴリを使用して記事へのアクセスを制御します。

ナレッジ記事とナレッジ記事のバージョン
: 記事を操作する場合、KnowledgeArticle がすべての記事のバージョンの親レコードを表すことに注意してください。KnowledgeArticleVersion レコードは、特定の記事の各バージョンを表します。

レコードタイプと記事タイプ
: 記事は、Lightning Experience と Salesforce Classic では異なった構造で表現されます。Lightning Knowledge では、別のカスタムオブジェクトで使用できるレコードタイプと同じものを使用して、異なるタイプの記事を構造化します (Knowledge__kav の `RecordTypeId` 項目を参照)。たとえば、レコードタイプごとに異なるレイアウトを使用できます。Salesforce Classic では、この機能は記事タイプによって実現できます (KnowledgeArticleVersion の `ArticleType` 項目を参照)。記事のタイプごとに、Salesforce Classic には個別のオブジェクトがあります (たとえば、FAQ 記事タイプの場合は FAQ__kav)。Lightning Knowledge では、レコードタイプを使用して処理されるため、タイプごとに個別のオブジェクトはありません。

利用者チャネル
: 利用者はチャネルと呼ばれることもあり、記事にアクセスできるユーザーの種別を指します。Salesforce ナレッジでは、記事を利用可能にできるチャネルを 4 つ提供しています。

- 内部アプリケーション: Salesforce ユーザーが、ロールの表示設定に応じて記事にアクセスできます。
- カスタマー: カスタマーはコミュニティ、サイト、カスタマーポータルの記事にアクセスできます。カスタマーユーザーは、取引先でのマネージャーのロール表示設定を継承します。コミュニティでは、カ

スタマーコミュニティまたはカスタマーコミュニティプラスライセンスを持つユーザーのみが記事を使用できます。

- パートナー:パートナーはコミュニティ、サイト、パートナーポータルの記事にアクセスできます。パートナーユーザーは、取引先でのマネージャーのロール表示設定を継承します。コミュニティでは、パートナーコミュニティライセンスを持つユーザーのみが記事を使用できます。
- 公開知識ベース: 公開知識ベースを作成することで、記事を匿名ユーザーに表示できます。Lightning Knowledge では、ほとんどの Salesforce 組織はコミュニティを使用して知識ベースを作成します。Salesforce Classic で Salesforce Knowledge 用の公開知識ベースを作成するには、サイトと Visualforce が必要です。

公開サイクル

Salesforce ナレッジ記事は公開サイクルの作成から削除までの段階を移行していきます。公開サイクルには、3 つの異なる状況があります。`Draft` は、新しい記事が作成されるか既存の記事が更新される段階です。状況が `Online` である記事は、公開済みで、他のチャネルで利用できるようになったドラフト記事です。最後に、公開記事がサイクルの最終段階になると、`Archived` 状況に移行するか、または `Draft` に戻り後続のバージョンで更新されます。

API での記事の使用

記事は、API の KnowledgeArticleVersion オブジェクトおよび KnowledgeArticle オブジェクトを使用して利用できます。これらのオブジェクトは両方とも記事を表示しますが、異なる機能を提供します。

KnowledgeArticleVersion

Salesforce ナレッジの新しいドラフト記事には必ずバージョン番号があります。記事を公開しており、その記事を更新する必要がある場合は、個別のバージョン番号で新しい `Draft` を作成できます。各バージョンには独自の ID があります。更新されたバージョンを公開する準備が整ったら、前のバージョンと置き換え、バージョン番号を更新します。KnowledgeArticleVersion オブジェクトを使用して、記事バージョンの内容にアクセスしたり、Draft 状況または Online 状況で絞り込むことができます。たとえば、次のクエリではすべての記事タイプのすべての記事の Draft バージョンのタイトルをアメリカ英語で返します。

```
SELECT Title
FROM KnowledgeArticleVersion
WHERE PublishStatus='Draft'
AND language ='en_US'
```

また、記事には記事番号が自動的に割り当てられます。この番号は個々の記事に対しては一意の ID ではありませんが、プライマリ記事および利用可能なすべての翻訳に対しては 1 つの ID です。

メモ: プライマリバージョン (IsMasterLanguage = 1 であるナレッジの記事) および翻訳はどちらも KnowledgeArticleVersion オブジェクトです。

KnowledgeArticle

KnowledgeArticleVersion とは異なり、KnowledgeArticle レコードの ID は、記事のバージョン (状況) に関係なく同じです。KnowledgeArticleVersion オブジェクトでは記事のカスタム項目値への API アクセスを提供する場合、KnowledgeArticle オブジェクトでは記事のメタデータ項目への API アクセスを提供します。

記事レコードは、公開の状況(ドラフト、公開、アーカイブ済み)や言語を問わず、記事のすべてのバージョンの親コンテナです。KnowledgeArticle および KnowledgeArticleVersion は、知識ベースの任意の記事を表しますが、これらのオブジェクトの特定の記事には具象表現が使用されます。Lightning Knowledge では、これらの具象表現

はデフォルトで Knowledge__ka (ナレッジ記事用) および Knowledge__kav (ナレッジ記事のバージョン用) になります。Salesforce Classic では、<Article Type>__ka と <Article Type>__kav を使用します。

次の Lightning Knowledge クエリでは、公開されたすべての FAQ 記事のタイトルがアメリカ英語で返されます。

```
SELECT Title
FROM Knowledge__kav
WHERE PublishStatus='online'
AND Language = 'en_US'
AND RecordTypeId = '<specify RecordTypeId for FAQ here>'
```

次の Salesforce Classic クエリでは、公開されたすべての Offer のタイトルがアメリカ英語で返されます。

```
SELECT Title
FROM FAQ__kav
WHERE PublishStatus='online'
AND language ='en_US'
```

データカテゴリの概要

データカテゴリは、カテゴリグループ別に編成され、次のことを実行できます。

- ユーザーによるレコードの分類および検索。
- システム管理者によるレコードへのアクセスの管理。

Salesforce ナレッジではデータカテゴリを使用して記事を分類し、簡単に検索できるようにします。たとえば、営業地域や商品で記事を分類するには、[Sales Regions (営業地域)] と [Products (商品)] という 2 つのカテゴリグループを作成します。[Sales Regions (営業地域)] カテゴリグループでは、[All Sales Regions (全営業地域)] を最上位として、第 2 レベルに [北米]、[アジア] というように、地理的階層を構成することができます。[Products (商品)] グループには、[すべての商品] を最上位として、第 2 レベルに [電話]、[コンピューター]、[プリンター] を含めることができます。

API でのデータカテゴリの操作

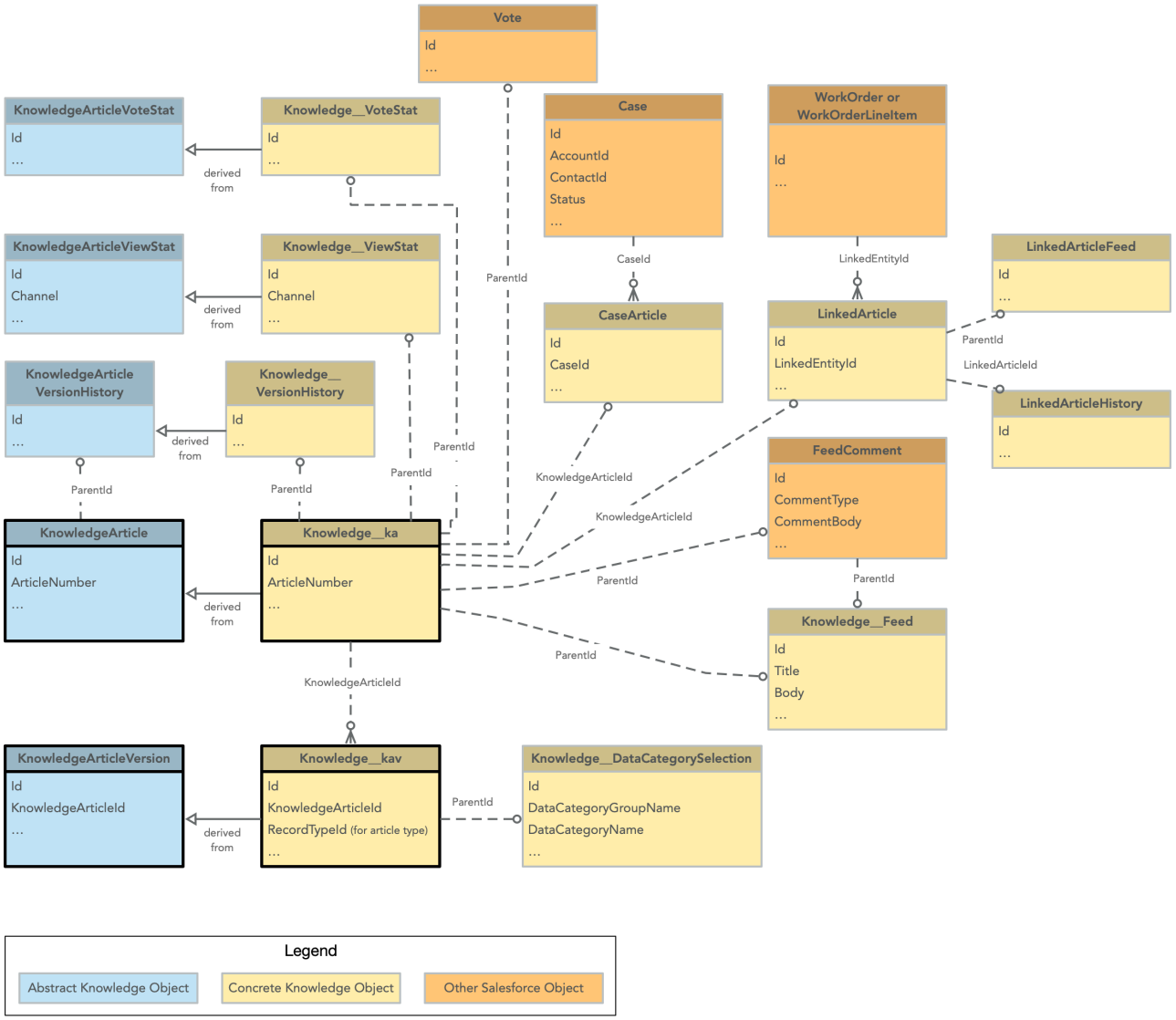
次の表に、データカテゴリを操作する API リストを示します。

名前	型	説明
Knowledge__DataCategorySelection	Object	Lightning Knowledge での記事カテゴリへのアクセスを許可します。
Article Type__DataCategorySelection	Object	Salesforce Classic のナレッジでの記事カテゴリへのアクセスを許可します。
QuestionDataCategorySelection	Object	質問カテゴリへのアクセスを許可します。
WITH DATA CATEGORY ***filteringExpression***	SOQL 句	公開サイクルおよびデータカテゴリの状況に応じて記事を分類します。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。

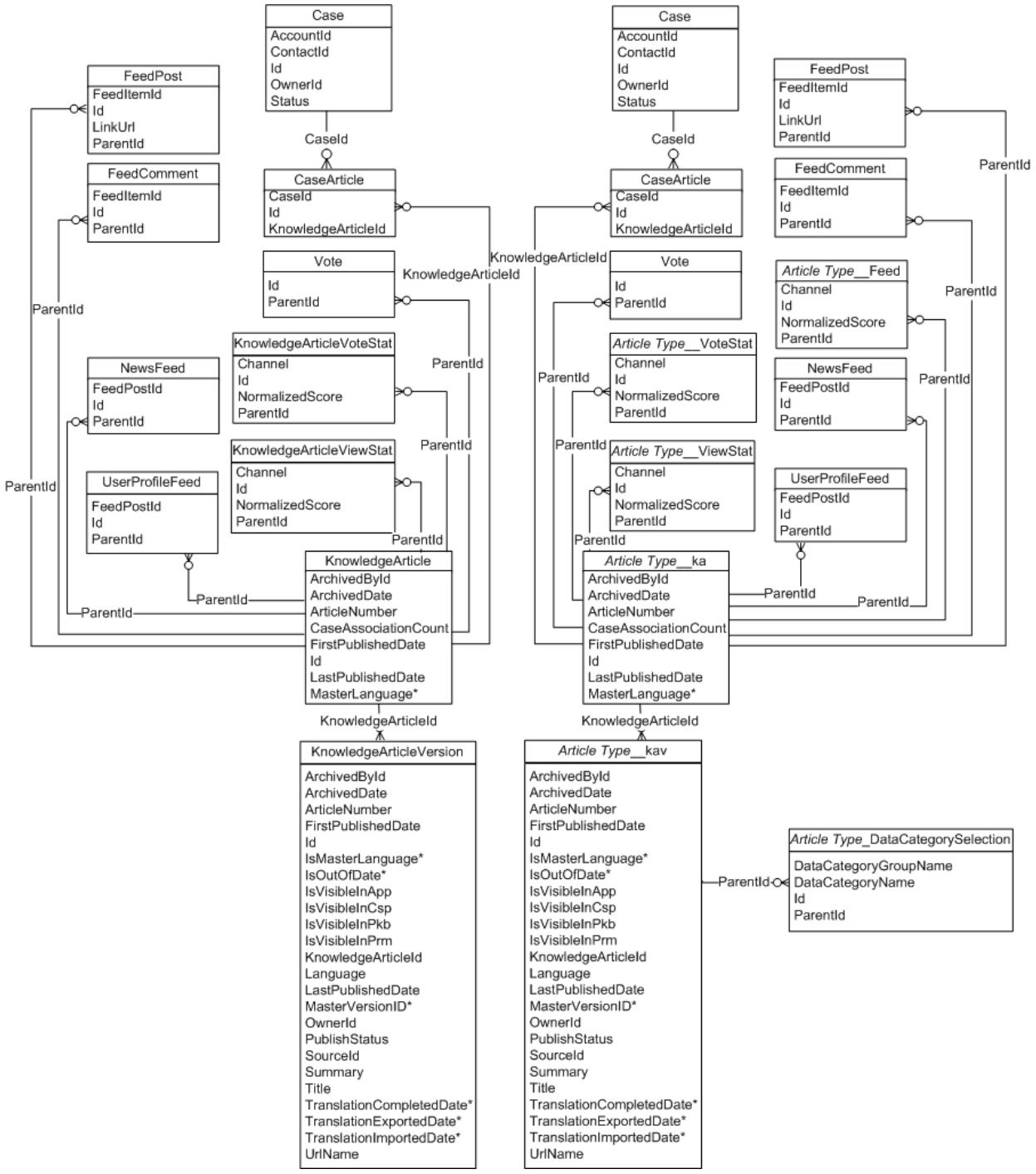
名前	型	説明
`WITH DATA CATEGORY` ***`DataCategorySpec`***	SOSL 句	カテゴリに基づいて記事を検索します。詳細は、*『Salesforce SOQL および SOSL リファレンスガイド』*を参照してください。
`describeDataCategoryGroups()`	Call	要求で指定されたオブジェクトで使用できるカテゴリグループを取得します。
`describeDataCategoryGroupStructures()`	Call	要求で指定されたオブジェクトで使用できるカテゴリグループとそのデータカテゴリ構造を返します。
`describeDataCategoryGroups`	Apex メソッド	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。*『Apex 開発者ガイド』*を参照してください。
`describeDataCategoryGroupStructures`	Apex メソッド	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。*『Apex 開発者ガイド』*を参照してください。

Salesforce ナレッジオブジェクト

このエンティティリレーションダイアグラム (ERD) は、Lightning Knowledge の Salesforce ナレッジオブジェクト間のリレーションを示します。



この ERD は、Salesforce Classic のオブジェクト間のリレーションを示します。



追加情報

API を使用した知識ベースの管理についての詳細は、『ナレッジ開発者ガイド』を参照してください。

用語集

A

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 Guide (AJAX Toolkit 開発者ガイド)』を参照してください。

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。

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。

Apex
Apex は、開発者が Lightning プラットフォームサーバーでフローとトランザクションの制御ステートメントを Lightning Platform 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 に似た構文を使用し、データベースのストアドプロシージャーのように動作する 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ーから開始できます。

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。

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、セールスおよびサービス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 ユーザー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、これを使用して Lightning Platform 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。

AppExchange のアップグレード
アプリケーションのアップグレードは、新しいバージョンをインストールするプロセスです。

アプリケーションプログラムインターフェース (API)
: コンピューターシステム、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ー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。

B

Boolean 演算子
: Boolean 演算子をレポートプロファイルで使用して、2 つの値の間の論理関係を指定できます。たとえば、2 つの値の間で AND 演算子を使用すると、両方の値を含む検索結果が生成されます。同様に、2 つの値の間で OR 演算子を使用すると、どちらかの値を含む検索結果が生成されます。

Bulk API 2.0
: REST ベースの Bulk API 2.0 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ジョブ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、ユーザー定義メソッド、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 ユーザーインターフェースの外部で実行し、Lightning プラットフォーム API または Bulk API 2.0 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の供給元として扱い、アプリケーションが設計されたあらゆるツールやプラットフォームの開発モデルを使用できます。

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。

コントローラー、**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ーを使用、またはカスタムコントローラーを使用できます。

制御項目
: 対応する 1 つ以上の連動項目で使用可能な値を制御する、標準またはカスタムの選択リストやチェックボックスの項目。

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。

カスタムリンク
: 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 サイトや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 **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ません。既存の S コントロールには影響がなく、引き続き編集できます。

  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、ブラウザー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。

D

データベース
: 情報の編成されたコレクション。Lightning 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。

データローダー
: 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 Lightning Platform 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。

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。中間値は常に切り上げられます。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。

代理認証
外部の権限を使用して Lightning プラットフォームユーザーを認証するセキュリティ処理。

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。

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 Lightning Platform 環境も、ここから入手できます。

ドキュメントライブラリ
ドキュメントの保存場所。これらのドキュメントは、取引先や取引先責任者、商談、またはその他のレコードに添付しません。

E

メールアラート
指定のメールテンプレートを使用して特定の受信者にメールを送信するアクション。

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 Lightning Platform 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 オブジェクトのエンティティリレーションダイアグラム (ERD) については、*Salesforce オブジェクトリファレンス*を参照してください。

F

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。

項目レベルセキュリティ
項目が、ユーザーに非表示、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です。使用可能なエディションは、Professional Edition、Enterprise Edition、Unlimited Edition、Performance Edition、および Developer Edition です。

検索条件
リストビューまたはレポートに含まれる項目に該当する、特定の項目に対する条件です。たとえば「都道府県」「次の文字列と一致する」「東京都」など。

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。

数式項目
: カスタム項目の一種。差し込み項目、式、またはその他の値に基づいて、値を自動的に計算します。

関数
: あらかじめ用意されている数式。入力パラメーターを使用してカスタマイズできます。たとえば、DATE 関数は、年、月、および日付から日付データ型を作成します。

G

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。

Group Edition
: ユーザー数が制限された小規模ビジネスやワークグループに設計された製品。

H

HTTP デバッガー
: 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ーで稼動するプロキシサーバーとして動作し、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。

インライン 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ません。既存のSコントロールには影響がなく、引き続き編集できます。

  各ページでなく、レコード詳細ページまたはダッシュボード内に表示される Sコントロール。

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ー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ーです。Lightning 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ユーザー
: 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ユーザーです。また、SOAP API コンテキストではログインユーザーとも呼ばれます。

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。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、「Bug(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考えられます。

K

該当用語はありません。

L

ライセンス管理アプリケーション (LMA)
無料の AppExchange アプリケーションで、AppExchange から管理パッケージ (アプリケーション) をダウンロードするすべてのユーザーのセールスリードおよび取引先を追跡できます。

ライセンス管理組織 (LMO)
パッケージをインストールしたすべての Salesforce ユーザーを追跡できる、Salesforce 組織です。ライセンス管理組織には、ライセンス管理アプリケーション (LMA) をインストールする必要があります。ライセンス管理アプリケーションは、パッケージがインストールまたはアンインストールされるたびに自動的に通知を受信するため、簡単にユーザーにアップグレードを通知できます。Enterprise Edition、Unlimited Edition、Performance Edition、または Developer Edition の組織をライセンス管理組織として指定できます。詳細は、「管理パッケージのライセンス管理」を参照してください。

Lightning プラットフォーム
クラウドでアプリケーションを構築するための Salesforce Platform。Lightning プラットフォームは、強力なユーザーインターフェース、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。

リストビュー
特定の条件による項目 (取引先、または取引先責任者など) 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。

ログインユーザー
SOAP API コンテキストで、Salesforce にログインするために使用するユーザー名です。クライアントアプリケーションは、ログインユーザーの権限および共有設定に基づいて動作します。また、インテグレーションユーザーとも呼ばれます。

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ー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。

マスター選択リスト
レコードタイプまたはビジネスプロセスで使用可能な選択リストのすべての値。

メタデータ
組織および組織の部署の、構造、表示、機能に関する情報。Lightning プラットフォームでは、メタデータを記述するのに XML を使用します。

Metadata WSDL
Lightning プラットフォームメタデータ API コールを使用するユーザーの WSDL。

マルチテナンシー
すべてのユーザー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。

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プレフィックスとそれに続く 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
Lightningプラットフォームの設定(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、ケース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、組織には、それら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が存在します。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
特定のユーザーに対してオブジェクト全体を非表示にできる設定。ユーザー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。

onClick JavaScript
ボタンまたはリンクをクリックすると実行される JavaScript コード。

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。

組織の共有設定
ユーザー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。

発信通話
Salesforce CRM Call Center のコールセンターの外部にユーザーから発信する通話。

送信メッセージ
送信メッセージでは、外部サービスなどの指定エンドポイントに情報を送信します。送信メッセージは [設定] から設定します。SOAP API を使用して外部エンドポイントを設定し、メッセージのリスナーを作成する必要があります。

フロート表示
ユーザーインターフェースの要素にマウスポインターを停止すると、フロート表示に追加情報が表示されます。フロート表示によって、マウスを移動したり、フロート表示外部をクリックしたり、または [閉じる] ボタンをクリックしたりすると、フロート表示が閉じられます。

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ー。

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 を介して他の組織から使用可能な Lightning プラットフォーム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
: この連動関係は、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。コンポーネントには、たとえば次のようなものを含めることができます。

  - 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

  権限と設定には、たとえば次のようなものを含めることができます。

  - 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、パッケージの内容が Salesforce 組織に組み込まれます。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、パッケージのコンポーネントをリリースすることで、組織のユーザーが汎用的に使用できるようにすることが可能です。

パッケージの公開
: パッケージを公開すると、AppExchange 上で公開して使用できるようになります。

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適切にアップグレードできます。そのパッケージを使用する既存の顧客のインテグレーションに影響を与えることもありません。「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。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。

個人情報
: ユーザー一般情報。個人の連絡先情報、目標、非公開グループ情報、およびデフォルトの商談チームなどが含まれます。

選択リスト
: Salesforce オブジェクトの特定の項目で選択できる選択肢。たとえば、取引先の［業種］項目など。ユーザーは、項目に直接入力せずに、選択リストから 1 つの値を選択できます。「マスター選択リスト」も参照してください。

選択リスト (複数選択)
: Salesforce オブジェクトの特定の項目で選択できる選択肢のリスト。複数選択リストを使用して 1 つまたは複数の値を選択できます。ユーザーは値をダブルクリックして選択するか、Ctrl キーを押したまま値をクリックしてスクロールリストから複数の値を選択し、矢印アイコンを使用して選択されたボックスに値を移動できます。

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ー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 ベンダーは、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 ベンダーは、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、プログラマー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ーを持っている Salesforce 組織です。

Q

キュー
処理する前にアイテムを置いておく領域です。Salesforce では、さまざまな機能やテクノロジーにキューを使用します。

クエリロケーター
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ー。

クエリ文字列パラメーター
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。

```
https://yourInstance.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 Lightning プラットフォーム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。

レコードタイプ
レコードタイプとは、そのレコードの標準およびカスタムの選択リスト項目の一部またはすべてを含めることができる特定のレコードに使用可能な項目です。レコードタイプをプロファイルに関連付けて、含まれている選択リストの値のみがそのプロファイルのユーザーに使用できるようにできます。

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、特定のユーザーがオブジェクトを参照および編集でき、ユーザーが編集できるレコードを制限できます。

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、Salesforce Classic のサイドバー内のリンクまたは Lightning Experience のアプリケーションランチャーからアクセスします。

関連オブジェクト
特定のタイプのレコードがコンソールの詳細ビューに表示されている状況で、システム管理者がエージェントコンソールのミニビューへの表示を指定できるオブジェクトです。たとえば、システム管理者は、ケースが詳細ビューに表示されているときにミニビューに表示される項目として、関連する取引先、取引先責任者、納入商品などを指定できます。

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です。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。

レポートタイプ
*レポートタイプ*は、主オブジェクトとその関連オブジェクトとの関係に基づいて、レポートで使用するレコードと項目のセットを定義するものです。レポートには、レポートタイプで定義された条件を満たすレコードのみが表示されます。Salesforce には、定義済みの標準レポートタイプのセットが用意されています。管理者がカスタムレポートタイプを作成することもできます。

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です。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ーは、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ーが所有しているデータ、および該当のユーザー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データ型。

実行ユーザー
各ダッシュボードには*実行ユーザー*が指定され、そのユーザー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。実行ユーザーが特定の 1 ユーザーである場合、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、実行ユーザー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、実行ユーザーをログインしたユーザーに設定することができるため、各ユーザー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ません。既存のSコントロールには影響がなく、引き続き編集できます。

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、ブラウザー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 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 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 Lightning プラットフォームの強力な機能です。

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。

検索語句
検索語句は、www.google.com で検索するときにユーザーが入力するクエリです。

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
ユーザーが 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、ユーザーが 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ー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。ユーザーが 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。

設定
システム管理者が組織の設定および Lightning プラットフォーム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。組織のユーザーインターフェース設定に応じて、[設定] はユーザー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、ユーザー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
Salesforce 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、ユーザーがログインする場合にユーザー名やパスワードは必要ありません。

スニペット

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ません。既存の S コントロールには影響がなく、引き続き編集できます。

スニペットは、他の S コントロールに組み込めるよう設計された S コントロールです。コードの一部で他のメソッドによって使用されるヘルパーメソッドと同様、スニペットを使用して、複数の S コントロールで再利用できる HTML や JavaScript の 1 つのコピーを保持できます。

SOAP (Simple Object Access Protocol)
XML エンコードデータを渡す一定の方法を定義するプロトコル。

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
Lightning 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 ベンダーは、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)
: 単純で強力なクエリ文字列を構築したり、Lightning プラットフォームデータベースからデータを選択する条件を指定したりできるクエリ言語です。

SOSL (Salesforce オブジェクト検索言語)
: Lightning プラットフォーム 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です。

標準オブジェクト
: Lightning 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。

シンジケーションフィード
: ユーザーに、Salesforce サイト内の変更を登録し、外部のニュースリーダーで更新を受け取る機能を提供します。

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

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 クラスメソッドです。Test メソッドは引数を取らず、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 Visual Studio Code 向け Salesforce 拡張機能のような Apex IDE で `runTests()` システムメソッドによって実行できます。

トランスレーションワークベンチ
: トランスレーションワークベンチでは、翻訳する言語を指定したり、翻訳者を言語に割り当てたり、Salesforce 組織に加えたカスタマイズの翻訳を作成したり、管理対象パッケージの表示ラベルと翻訳を上書きしたりできます。カスタム選択リスト値からカスタム項目にいたるすべてを翻訳し、海外のユーザーが Salesforce を彼らの言語で使用できるようになります。

トリガー
: データベースの特定の種別のレコードが挿入、更新、または削除される前後で実行する Apex スクリプトです。各トリガーは、トリガー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、すべてのトリガーは一括モードで実行します。つまり、一度に 1 つずつレコードを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。

トリガーコンテキスト変数
: トリガーおよびトリガー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。

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。このコンポーネントは、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ができます。また、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ーと関連付けることもできます。

W

Web コントロール
「URL Sコントロール」を参照してください。

Web サービス
2つのアプリケーションが、異なるプラットフォームで稼動していたり、異なる言語で作成されていたり、地理的に離れた場所に存在していたりする場合でも、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。

Web サービス API
Salesforce 組織の情報へのアクセスを提供する元の Salesforce Platform Web サービスのアプリケーションプログラミングインターフェース (API) を示す用語。目的の SOAP、REST、または Bulk API に関係する開発者ガイドを参照してください。

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、外部システムで使用できる Apex クラスメソッドまたは変数です。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 タブ
ユーザーがアプリケーション内から外部 Web サイトを使用できるカスタムタブ。

自動アクション
メールアラート、ToDo、項目自動更新、送信メッセージなどの自動アクションは、プロセス、ワークフロールール、承認プロセス、またはマイルストーンでトリガーできます。

ワークフローアクション
ワークフローアクション (メールアラート、項目自動更新、送信メッセージ、ToDo など) は、ワークフロールールの条件が満たされると起動します。

ワークフローメールアラート
ワークフロールールが起動したときにメールを送信するワークフローアクションです。ワークフロー ToDo と異なり、アプリケーションユーザーにのみ割り当てることができ、ワークフローアラートは有効なメールアドレスがある限り、ユーザーまたは取引先責任者に送信できます。

ワークフロー項目自動更新
: ワークフロールールが起動したときに、レコードの特定の項目の値を変更するワークフローアクション。

ワークフロー送信メッセージ
: 別のクラウドコンピューティングアプリケーションなど、外部 Web サービスにデータを送信するワークフローアクション。送信メッセージは、主にクライアントアプリケーションで使用されます。

ワークフローキュー
: 1つ以上の時間ベースワークフローアクションがあるワークフロールールに基づいて起動するようスケジュールされている、ワークフローアクションのリスト。

ワークフロールール
: ワークフロールールは、指定された条件に該当するときに、ワークフローアクションを実行します。ワークフローアクションは、ワークフロールールで指定された条件をレコードが満たすとただちに実行するか、タイムトリガーを設定して特定の日に実行するように設定することができます。

ワークフロー ToDo
: ワークフロールールが起動したときに ToDo をアプリケーションユーザーに割り当てるワークフローアクション。

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。

WSC (Web Service Connector)
: XML ベースの Web サービスフレームワークで、SOAP サーバーの Java 実装で構成されます。WSC を使用すると、開発者は、Salesforce Enterprise WSDL または Partner WSDL から生成された Java クラスを使用して、Java でクライアントアプリケーションを開発できます。

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

該当用語はありません。

Y

該当用語はありません。

Z

該当用語はありません。