Bulk API 開発者ガイド

バージョン 37.0, Summer '16

@salesforcedocs

最終更新日: 2016/04/22

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

第 1 章　　Bulk API の概要

Bulk API を使えば、プログラムを介して、組織のデータを Salesforce にすばやく読み込むことができます。API を使用するには、ソフトウェア開発、Web サービス、および Salesforce ユーザインターフェースについての基本的な知識が必要です。

説明されている機能は、組織で Bulk API 機能が有効化されている場合にのみ使用できます。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。

Bulk API を使用するケース

Bulk API は、REST 規則に基づいており、大規模データセットの読み込みまたは削除用に最適化されています。Bulk API を使用して複数のバッチを送信することにより、多数のレコードを非同期でクエリ、挿入、更新、更新/挿入または削除できます。バッチはバックグラウンドで処理されます。

一方、SOAP API は、一度に少数のレコードを更新するリアルタイムのクライアントアプリケーション用に最適化されています。SOAP API を使用しても多数のレコードを処理することはできますが、数十万のレコードがデータセットに含まれている場合には実用性に欠けます。Bulk API は、千から百万単位のレコードのデータを簡単に読み込めるように設計されています。

Bulk API の最も簡単な使用方法は、CSV ファイルを使ってデータローダでレコードの処理ができるようにすることです。データローダを使用すると、独自のクライアントアプリケーションを作成する必要がなくなります。

SOAP API を使用するケース

SOAP API では、Salesforce を操作するための、強力で便利な、使いやすい SOAP ベースの Web サービスインターフェースを提供します。SOAP API を使用して、レコードを作成、取得、更新、または削除できます。また、SOAP API を使用して、検索の実行などを行うことができます。SOAP API は、Web サービスをサポートしている言語で使用できます。

たとえば、SOAP API を使用して、Salesforce を組織の ERP や会計システムと統合できます。また、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することもできます。

REST API を使用するケース

REST API では、Salesforce を操作するための、強力で便利な、使いやすい REST ベースの Web サービスインターフェースを提供します。インテグレーションや開発が容易になるという利点があり、モバイルアプリケーションや Web プロジェクトで使用するための技術の選択としては最適です。ただし、処理するレコード件数が多

い場合、REST 規則に基づいており、大規模データセットの処理用に最適化されている Bulk API を使用することを検討してください。

メタデータ API を使用するケース

メタデータ API を使用して、組織のカスタマイズを取得、リリース、作成、更新、または削除します。最も一般的な使い方は、Sandbox またはテスト組織から本番環境への変更の移行です。メタデータ API は、カスタマイズを管理し、データ自体ではなくメタデータモデルを管理可能なツールを構築することを目的とします。

Force.com IDE または Force.com 移行ツールを使用すると、最も簡単にメタデータ API の機能にアクセスできます。どちらのツールもメタデータ API の上位に構築され、メタデータ API を使用した作業を簡略化するために標準の Eclipse および Ant ツールが使用されます。

- Force.com IDE は、Eclipse プラットフォーム上に構築され、統合された開発環境に慣れているプログラマを対象としています。IDE 内でコード記述、コンパイル、テスト、リリースを行います。
- Force.com 移行ツールは、スクリプトまたはコマンドラインを使用してローカルディレクトリと Salesforce 組織間でメタデータを移動する場合に最適です。

Bulk API によって可能になること

REST Bulk API では、大量のレコードを非同期でクエリ、挿入、更新、更新/挿入、削除できます。レコードには、Attachment オブジェクトや Salesforce CRM Content などのバイナリ添付ファイルを含めることができます。まず、HTTP POST を使用して、サーバに一連のバッチを送信します。バッチを受け取ったサーバは、バックグラウンドで処理を実行します。バッチの処理が行われている間は、HTTP GET コールを使用してジョブの状況を確認し、進行状況を追跡できます。すべての処理では、HTTP GET メソッドまたは HTTP POST メソッドを使用して、CSV データ、XML データ、または JSON データが送受信されます。

重要: 現在、Bulk API のクエリで base64 項目の使用はサポートされていません。

Bulk API のしくみ

一連のレコードを処理するには、1 つ以上のバッチを含むジョブを作成します。このジョブは、処理されるオブジェクトと使用されるアクションのタイプ (クエリ、挿入、更新/挿入、更新、または削除) を指定します。バッチは、HTTP POST 要求でサーバに送信されるレコードセットです。各バッチはサーバによって独自に処理されます。受信した順序で処理されるとは限りません。バッチは並列処理が可能です。データセット全体をどのように適切な数のバッチに分割するかは、クライアント側で決定されます。

ジョブは JobInfo リソースで表されます。このリソースは、新規ジョブの作成、既存のジョブの状況の取得、ジョブの状況の変更に使用します。バッチを作成するには、レコードのセット、およびバイナリ添付ファイルへの参照を表す、CSV、XML、または JSON を HTTP POST 要求で送信します。作成したバッチの状況は BatchInfo リソースで表されます。バッチの処理が完了すると、各レコードの結果が結果セットのリソースとして提供されます。

データを処理する一般的な手順は、次のとおりです。

1. オブジェクトとアクションを指定し、新しいジョブを作成します。

2. 複数のバッチに分割されたデータをサーバに送信します。
3. すべてのデータが送信されたら、ジョブを終了します。ジョブの終了後は、そのジョブの一部として新たにバッチを送信することはできません。
4. 適切な間隔ですべてのバッチの状況を確認します。状況確認では、その都度各バッチの状態が返されます。
5. すべてのバッチの処理が完了または失敗した時点で、各バッチの結果を取得します。
6. 結果セットと元のデータセットを照合して、失敗したレコードと成功したレコードを特定し、適切な処理を行います。

このプロセス中には、いつでもジョブを中止できます。ジョブを中止すると、その時点で未処理のバッチは処理されなくなります。すでに処理されたバッチが元に戻されることはありません。

データローダを使用した CSV の処理については、*『データローダガイド』*を参照してください。

関連トピック:

- ジョブでの作業
- バッチでの作業

CORS を使用した、サポートされた Salesforce API、Apex REST、および Lightning Out へのアクセス

Chatter REST API、REST API、Lightning Out、Bulk API、および Apex REST では、CORS (クロスオリジンリソース共有) がサポートされます。Web ブラウザで JavaScript からこれらの API にアクセスするには、スクリプトを提供するオリジンを CORS ホワイトリストに追加します。

「CORS」は、Web ブラウザが他のオリジンからのリソースを要求 (クロスオリジン要求) できるようにする W3C 勧告です。たとえば、CORS を使用すると、`https://www.example.com` にある JavaScript スクリプトで `https://www.salesforce.com` からのリソースを要求できます。

CORS をサポートするブラウザが、Salesforce CORS ホワイトリスト内のオリジンに要求を行うと、Salesforce はオリジンを含む `Access-Control-Allow-Origin` HTTP ヘッダーと、追加の CORS HTTP ヘッダーを返します。オリジンがホワイトリストにない場合は、Salesforce が HTTP 状況コード 403 を返します。

1. [設定] から、[クイック検索] ボックスに *「CORS」* と入力し、**[CORS]** を選択します。
2. **[新規]** を選択します。
3. オリジンの URL パターンを入力します。

   オリジンの URL パターンには、HTTPS プロトコル (localhost を使用しない場合) とドメイン名が含まれている必要があり、ポートが含まれることもあります。ワイルドカード文字 (*) はサポートされますが、第2レベルドメイン名の前にある必要があります。たとえば、`https://*.example.com` により、`example.com` のすべてのサブドメインがホワイトリストに追加されます。

エディション

使用可能なエディション: Salesforce Classic および Lightning Experience

使用可能なエディション: **Developer** Edition、**Enterprise** Edition、**Performance** Edition、**Unlimited** Edition

ユーザ権限

作成、参照、更新、および削除する
- 「すべてのデータの編集」

オリジンの URL パターンに IP アドレスを使用できます。ただし、IP アドレスと、同じアドレスに解決するドメインは同じオリジンではないため、CORS ホワイトリストには別々のエントリとして追加する必要があります。

重要: OAuth トークンが必要な要求では、OAuth トークンを渡す必要があります。

第 2 章　クイックスタート

トピック:

このサンプルを使用して、REST ベースの Bulk API で新しい取引先責任者レコードを挿入する HTTP 要求を作成します。ログインから、レコードの送信、状況の確認、結果の取得まで、手順を順番に説明します。

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。

- 製品ドキュメントに従って、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。このガイドの他の部分を確認しておくと、用語や概念を把握できます。

Salesforce Developer Edition 組織の設定

まず、Salesforce Developer Edition 組織を取得し、Bulk API を有効にします。

1. Salesforce Developer Edition 組織を取得します。

   まだ開発者コミュニティのメンバーでない場合、developer.salesforce.com/signup にアクセスし、Developer Edition アカウントのサインアップの説明に従ってください。すでに Enterprise Edition、Unlimited Edition、または Performance Edition のアカウントを所有している場合でも、組織の使用中のデータを保護するために、サンプルデータに対するソリューションの開発、ステージングおよびテストには Developer Edition を使用することを強くお勧めします。これは、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) データをクエリ、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。

2. Bulk API を有効にします。

   「API の有効化」権限が必要です。この権限は、システム管理者プロファイルで有効にします。

クライアントアプリケーションの設定

Bulk API は HTTP GET メソッドおよび HTTP POST メソッドを使用して CSV、XML、および JSON のコンテンツを送受信します。そのため、任意のツールや言語を使用してクライアントアプリケーションを非常に簡単に構築することができます。このクイックスタートでは、HTTP 要求と応答の送受信を単純化するために、cURL というコマンドラインツールを使用します。

cURL は、多くの Linux システムや Mac システムにあらかじめインストールされています。Windows バージョンは、curl.haxx.se/ からダウンロードできます。Windows で HTTPS を使用する場合、システムが SSL の cURL 要件を満たしていることを確認してください。

メモ: cURL はオープンソースのツールで、Salesforce ではサポートされていません。

Mac および Linux システムでのセッション ID のエスケープまたは一重引用符の使用

REST リソースで cURL の例を実行するとき、セッション ID 引数の感嘆符の特殊文字によって、Mac および Linux システムでエラーが発生する場合があります。このエラーの発生を回避するには、次のいずれかを実行します。

- セッション ID が二重引用符で囲まれている場合、セッション ID の感嘆符 (!) 特殊文字の前にバックスラッシュを挿入して (\!) エスケープします。たとえば、この cURL コマンドのセッション ID 文字列では、感嘆符 (!) がエスケープされています。

```
curl https://instance_name.salesforce.com/services/data/v37.0/
-H "Authorization: Bearer
00D50000000IehZ\!AQcAQH0dMHZfz972Szmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1E6
LYUfiDUkWe6H34r1AAwOR8B8fLEz6n04NPGRrq0FM"
```

- セッション ID を一重引用符で囲みます。次に例を示します。

```
curl https://instance_name.salesforce.com/services/data/v37.0/
-H 'Authorization: Bearer sessionID'
```

cURL を使用した HTTP 要求の送信

cURL の設定が完了したら、Bulk API に HTTP 要求を送信できるようになります。Bulk API で処理を実行するには、URI に HTTP 要求を送信します。

HTTP 要求の送信先の URI の形式は次のとおりです。

Web_Services_SOAP_endpoint_instance_name/services/async/*APIversion*/*Resource_address*

API のバージョンより後ろの部分 (*Resource_address*) は、処理するジョブやバッチによって変わります。

Bulk API の最も簡単な使用方法は、CSV ファイルを使用してデータローダでレコードの処理ができるようにすることです。データローダを使用すれば、HTTP 要求を独自に作成したり、クライアントアプリケーションを新たに開発したりする必要はありません。Java を使用してクライアントアプリケーションを開発する例は、「Java を使用したサンプルクライアントアプリケーション」 (ページ 94)を参照してください。

1. ステップ 1: SOAP API を使用したログイン
   Bulk API はログイン処理をサポートしていません。ログインは、SOAP API を使用して実行する必要があります。
2. ステップ 2: ジョブの作成
   データを読み込むには、まずジョブを作成します。ジョブには、読み込むオブジェクトの種別(取引先責任者など) と、実行する処理 (クエリ、挿入、更新、更新/挿入、削除など) を指定します。ジョブでは、ユーザがデータ読み込みのプロセスをある程度制御できます。たとえば、進行中のジョブを中止することなどが可能です。
3. ステップ 3: ジョブへのバッチの追加
   ジョブを作成したら、次に取引先責任者レコードのバッチを作成します。データをバッチにまとめ、複数の HTTP POST 要求に分割して送信します。各要求の URI は、ジョブの作成で使用したものとよく似ていますが、今回は URI の末尾に ***jobId***/batch を追加します。
4. ステップ 4: ジョブの終了
   Salesforce へのバッチの送信が完了したら、ジョブを終了します。ジョブを終了すると、このジョブのバッチはこれ以上送信されないことが Salesforce に通知され、監視ページに、ジョブの処理状況に関する正確な統計データが表示されます。
5. ステップ 5: バッチ状況の確認
   次の cURL コマンドを実行すると、個々のバッチの状況を確認できます。
6. ステップ 6: バッチ結果の取得
   単一のバッチを実行して、その状況が Completed と表示されたら、そのバッチ結果を取得して個々のレコードの状況を確認する必要があります。

関連トピック:

URI について

ステップ 1: SOAP API を使用したログイン

Bulk API はログイン処理をサポートしていません。ログインは、SOAP API を使用して実行する必要があります。

1. `login.txt` という名前のテキストファイルを作成し、次のテキストを含めます。

```
<?xml version="1.0" encoding="utf-8" ?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <n1:login xmlns:n1="urn:partner.soap.sforce.com">
      <n1:username>your_username</n1:username>
      <n1:password>your_password</n1:password>
    </n1:login>
  </env:Body>
</env:Envelope>
```

2. *`your_username`* と *`your_password`* の箇所を、Salesforce のユーザ名とパスワードで置き換えます。
3. コマンドラインウィンドウを使用して、次の cURL コマンドを実行します。

```
curl https://login.salesforce.com/services/Soap/u/37.0 -H "Content-Type: text/xml;
charset=UTF-8" -H "SOAPAction: login" -d @login.txt
```

URI の `Soap/u/` 部分は Partner WSDL を指定しています。これを `Soap/c/` に変更すると、Enterprise WSDL を指定できます。

4. Salesforce は、`<sessionId>` 要素と `<serverUrl>` 要素を含む XML 応答を返します。`<sessionId>` 要素の値と、`<serverUrl>` 要素内のホスト名の最初にあるインスタンス (***`yourInstance`***`-api` など) をメモしておいてください。これらの値は、後続の処理で Bulk API に要求を送信するときに使用します。

関連トピック:

セッションヘッダーの設定

ステップ 2: ジョブの作成

データを読み込むには、まずジョブを作成します。ジョブには、読み込むオブジェクトの種別 (取引先責任者など) と、実行する処理 (クエリ、挿入、更新、更新/挿入、削除など) を指定します。ジョブでは、ユーザがデータ読み込みのプロセスをある程度制御できます。たとえば、進行中のジョブを中止することなどが可能です。

1. `job.txt` という名前のテキストファイルを作成し、次のテキストを含めます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <operation>insert</operation>
    <object>Contact</object>
    <contentType>CSV</contentType>
</jobInfo>
```

警告: この operation 項目の値はすべて小文字で入力してください。たとえば、`insert` の代わりに `INSERT` と入力すると、エラーが発生します。

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行します。

```
curl https://instance.salesforce.com/services/async/37.0/job -H "X-SFDC-Session:
sessionId" -H "Content-Type: application/xml; charset=UTF-8" -d @job.txt
```

instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x0000000005LAAQ</id>
  <operation>insert</operation>
  <object>Contact</object>
  <createdById>005x0000000wPWdAAM</createdById>
  <createdDate>2009-09-01T16:42:46.000Z</createdDate>
  <systemModstamp>2009-09-01T16:42:46.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>37.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

3. <id> 要素内に返されたジョブ ID の値をメモしておいてください。後続の処理で使用します。

関連トピック:

ジョブの作成

ステップ 3: ジョブへのバッチの追加

ジョブを作成したら、次に取引先責任者レコードのバッチを作成します。データをバッチにまとめ、複数の HTTP POST 要求に分割して送信します。各要求の URI は、ジョブの作成で使用したものとよく似ていますが、今回は URI の末尾に ***jobId***/batch を追加します。

バイナリ添付ファイルを含めない場合は、データの形式を CSV、XML、または JSON に設定します。バイナリ添付ファイルについては、「バイナリ添付ファイルの読み込み」 (ページ 28)を参照してください。バッチサイズの制限については、「バッチサイズ」 (ページ 91)を参照してください。

この例では、推奨される形式である CSV を使用します。データセットは、バッチサイズの制限内に収まるように適宜分割してください。ここでは、説明をわかりやすくするために少数のレコードのみを使用します。

ジョブにバッチを追加する手順は、次のとおりです。

1. data.csv という名前の CSV ファイルを作成し、次のような 2 件のレコードを含めます。

```
FirstName,LastName,Department,Birthdate,Description
Tom,Jones,Marketing,1940-06-07Z,"Self-described as ""the top"" branding guru on the West
 Coast"
Ian,Dury,R&D,,"World-renowned expert in fuzzy logic design.
Influential in technology purchases."
```

   最後の行の Description 項目の値は 2 行にわたっているため、二重引用符で囲みます。

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行します。

```
curl https://instance.salesforce.com/services/async/37.0/job/jobId/batch -H
"X-SFDC-Session: sessionId" -H "Content-Type: text/csv; charset=UTF-8" --data-binary
@data.csv
```

   instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。*jobId* は、ジョブ作成時に返されたジョブ ID です。

   Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>751x000000000079AAA</id>
 <jobId>750x0000000005LAAQ</jobId>
 <state>Queued</state>
 <createdDate>2009-09-01T17:44:45.000Z</createdDate>
 <systemModstamp>2009-09-01T17:44:45.000Z</systemModstamp>
 <numberRecordsProcessed>0</numberRecordsProcessed>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>0</totalProcessingTime>
 <apiActiveProcessingTime>0</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

   Salesforce では CSV のコンテンツは解析しませんが、代わりに後続の処理でバッチを検証します。この応答は単にバッチが受信されたことを知らせるためのものです。

3. <id> 要素内で返されたバッチ ID の値をメモしておいてください。このバッチ ID は、後でバッチの状況を確認するときに使用します。

関連トピック:

- CSV ファイルの準備
- ジョブへのバッチの追加
- Bulk API の制限

ステップ 4: ジョブの終了

Salesforceへのバッチの送信が完了したら、ジョブを終了します。ジョブを終了すると、このジョブのバッチはこれ以上送信されないことがSalesforceに通知され、監視ページに、ジョブの処理状況に関する正確な統計データが表示されます。

1. close_job.txt という名前のテキストファイルを作成し、次のテキストを含めます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <state>Closed</state>
</jobInfo>
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行します。

```
curl https://instance.salesforce.com/services/async/37.0/job/jobId -H "X-SFDC-Session: sessionId" -H "Content-Type: application/xml; charset=UTF-8" -d @close_job.txt
```

instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。*jobId* は、ジョブ作成時に返されたジョブ ID です。

この cURL コマンドは、ジョブリソースの状況を Open から Closed に更新します。

関連トピック:

ジョブの終了

ステップ 5: バッチ状況の確認

次の cURL コマンドを実行すると、個々のバッチの状況を確認できます。

```
curl https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId -H "X-SFDC-Session: sessionId"
```

instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。*jobId* は、ジョブ作成時に返されたジョブ ID です。*batchId* は、ジョブにバッチを追加したときに返されたバッチ ID です。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x00000000079AAA</id>
  <jobId>750x0000000005LAAQ</jobId>
  <state>Completed</state>
  <createdDate>2009-09-01T17:44:45.000Z</createdDate>
  <systemModstamp>2009-09-01T17:44:45.000Z</systemModstamp>
  <numberRecordsProcessed>2</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>5820</totalProcessingTime>
  <apiActiveProcessingTime>2166</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Salesforce でバッチのコンテンツを読み取れない場合や、バッチにエラーが含まれている場合 (例: CSV ファイルのヘッダー行に無効な項目名が含まれていた場合など)、バッチの状態は Failed になります。バッチ内のすべてのレコードが処理されると、バッチの状態は Completed になります。ただし、バッチの中には処理に失敗したレコードが含まれている場合があります。個々のレコードの状況を確認するために、バッチ結果を取得する必要があります。

各バッチの状況を個別に確認する必要はありません。次の cURL コマンドを実行すると、ジョブに含まれているすべてのバッチの状況を確認できます。

```
curl https://instance.salesforce.com/services/async/37.0/job/jobId/batch -H
"X-SFDC-Session: sessionId"
```

関連トピック:

- バッチに関する情報の取得
- ジョブ内のすべてのバッチに関する情報の取得
- バッチの状態の解釈

ステップ 6: バッチ結果の取得

単一のバッチを実行して、その状況が `Completed` と表示されたら、そのバッチ結果を取得して個々のレコードの状況を確認する必要があります。

単一のバッチの結果を取得するには、次の cURL コマンドを実行します。

```
curl https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId/result
-H "X-SFDC-Session: sessionId"
```

instance は、ログインの応答でメモした `<serverUrl>` 要素の一部分です。*sessionId* は同様にメモした `<sessionId>` 要素です。*jobId* は、ジョブ作成時に返されたジョブ ID です。*batchId* は、ジョブにバッチを追加したときに返されたバッチ ID です。

Salesforce により、次のようなデータを含む応答が返されます。

```
"Id","Success","Created","Error"
"003x0000004ouM4AAI","true","true",""
"003x0000004ouM5AAI","true","true",""
```

レスポンスボディは、バッチ要求の各行に対応する行が含まれた CSV ファイルです。レコードが作成されると、行に ID が挿入されます。レコードが更新されると、`Created` 列の値が「false」になります。レコードの処理が失敗すると、`Error` 列にエラーメッセージが挿入されます。

関連トピック:

- バッチ結果の取得
- バッチの失敗したレコードの処理

第 3 章　一括データ読み込みの計画

トピック:

ほとんどの場合、大量のレコードを読み込むには、SOAP ベースの API よりも Bulk API のほうが短時間で処理できます。ただし、読み込みのパフォーマンスは、読み込むデータの種類と、バッチ内のオブジェクトに関連付けられているワークフロールールやトリガによって変わってきます。この章では、読み込み時間を最小限にするためのポイントを解説します。

データ読み込みに関する一般的なガイドライン

最小限の処理時間でデータ読み込みが実行されるように計画を立てるためのヒントを紹介します。データ読み込みを行う場合は、事前に必ずSandbox組織でテストを実行します。ただし、Sandbox組織と本番組織とでは、処理時間が異なる可能性があります。

できる限り並列モードを使用する

バッチを並列モードで処理することにより、Bulk API のメリットを最大限に引き出すことができます。並列モードはデフォルトで選択されており、より迅速なデータ読み込みを可能にします。ただし、並列モードを使用すると、レコードに対するロック競合が発生することがあります。これを回避するには、逐次モードを使用して処理を実行します。ただし、並列モードだとロックタイムアウトが発生し、バッチを再構成してもロックを回避できないとわかっている場合以外は、逐次モードでデータ処理を実行しないでください。

処理モードはジョブレベルで設定します。1つのジョブに含まれるすべてのバッチが並列モード、逐次モードのいずれかで処理されます。

ロック競合を回避できるようにバッチを再構成する

たとえば、AccountTeamMember レコードの作成や更新を行う場合、このレコードが参照する取引先レコードは、トランザクションが完了するまでロックされます。AccountTeamMember レコードのバッチを複数読み込む場合に、これらのレコードに同一の取引先への参照が含まれていると、すべてのバッチで同じ取引先をロックしようとするため、ロックタイムアウトが発生することが予想されます。このような場合、バッチへのデータの振り分けを見直すことで、ロックタイムアウトを回避できる可能性があります。たとえば、`AccountId` 別に AccountTeamMember レコードを分類して、同じ取引先を参照するすべてのレコードを 1 つのバッチにまとめると、複数のバッチによるロック競合のリスクを減らすことができます。

Bulk API では、ロックが発生するとただちにエラーが返されるわけではありません。ロックが解除されるのを数秒間待ち、解除されない場合はレコードを Failed としてマークします。1 つのバッチ内で、100 を超えるレコードに対してロックが確認された場合、そのバッチの未処理のリクエストはいったんキューに戻されます。後で再度同じバッチが処理されるとき、Failed としてマークされたレコードは再試行されません。これらのレコードを処理するには、別のバッチにまとめて再送信する必要があります。

再度の処理も正常に完了しなかった場合、バッチは再びキューに戻され、最大10回まで処理が試行されます。それでも完了できない場合は、バッチ処理が完全に失敗したとみなされます。バッチの状態がFailedであっても、一部のレコードは正常に処理されている可能性があります。バッチ結果を取得して、個々のレコードの処理状況を確認する方法については、「バッチ結果の取得」 (ページ 55)を参照してください。エラーが回避されない場合は、別のジョブを作成し、データを逐次モードで処理します。逐次モードでは、バッチが 1 つずつ順番に処理されます。

ロック競合を引き起こしやすい処理に注意する

次に示すのは、ロック競合を引き起こしやすい処理です。これらの処理では必要に応じて逐次モードを使用します。

- 新規ユーザの作成
- 非公開で共有されているレコードの所有者の更新
- ユーザロールの更新
- テリトリー階層の更新

これらの処理に関連するエラーが発生した場合は、別のジョブを作成し、逐次モードでデータを処理してください。

メモ: データモデルは組織ごとに異なるため、ロック競合が発生する条件は、上記で説明したものと一致しない場合があります。

項目の数を最小限に抑える
各レコードの読み込む項目数が少ないほど、処理時間は短くなります。外部キー項目、参照関係項目、積み上げ集計項目が含まれると、処理時間が長くなる傾向にあります。項目の数を減らすことが難しい場合もあるかと思いますが、可能であれば実行してください。これにより、読み込み時間を短縮できます。

ワークフローアクションの数を最小限に抑える
ワークフローアクションの数が増えると処理時間が長くなります。

トリガの数を最小限に抑える
トリガが関連付けられたオブジェクトでは、それらのトリガが他の並列トランザクションに悪影響を及ぼさないようであれば、並列モードを使用してもかまいません。ただし、Salesforce では、複雑なトリガが関連付けられたオブジェクトを含む、サイズの大きなバッチを読み込むことは推奨していません。代わりに、すべてのデータが読み込まれた後に実行される一括処理 Apex ジョブとして、トリガのロジックを記述し直すことをお勧めします。

バッチサイズを最適化する
Salesforce では、すべての顧客組織間で処理リソースが共有されます。各組織がバッチの処理を長時間待たなくても済むように、処理に 10 分以上かかるバッチはいったんキューに戻され、後で処理されます。そのため、処理時間を短縮する最善の方法は、10 分以内に処理が完了するバッチを送信することです。一括処理のタイミングの監視についての詳細は、「バッチの監視」 (ページ 48)を参照してください。

バッチサイズは処理時間に基づいて調整する必要があります。まず 5,000 件のレコードを含むバッチで処理を試行し、処理時間に応じてサイズを調整します。1 つのバッチの処理に 5 分以上かかるようなら、バッチサイズを小さくしたほうがよいでしょう。数秒で完了するようであれば、バッチサイズを大きくします。バッチの処理時にタイムアウトエラーが発生する場合は、バッチをより小さなサイズに分割して再試行します。詳細は、「Bulk API の制限」 (ページ 90)を参照してください。

メモ: 一括クエリの場合、バッチサイズはクエリ結果セットと取得データサイズのどちらにも適用されません。一括クエリの処理に時間がかかりすぎる場合、クエリステートメントを絞り込んで、返されるデータ量を減らす必要があります。

非同期キュー内の一括処理数を最小限に抑える
Salesforce では、キューベースのフレームワークを使用して、Apex の future ジョブや一括処理、Bulk API 一括処理などのソースからの非同期プロセスを処理します。このキューは、組織間で要求ワークロードを調整するために使用します。キュー内で 1 つの組織の未処理要求が 2,000 を上回ると、その組織からの以降の要求は遅延し、その間に他の組織からの要求が処理されます。一括処理がキュー内で遅延しないようにするには、一度に送信できる一括処理の数を最小限に抑えます。

レスポンスの圧縮の使用

API バージョン 27.0 以降では、Bulk API は、レスポンスデータを圧縮することができます。これにより、ネットワークのトラフィックが削減され、応答時間が短縮されます。

レスポンスは、クライアントが `gzip` の値を持つ `Accept-Encoding` ヘッダーを使用して要求を行う場合に圧縮されます。Bulk API はレスポンスを gzip 形式で圧縮し、`Content-Encoding: gzip` 応答ヘッダーを持つクライアントにレスポンスを送信します。`gzip` 以外の値を持つ `Accept-Encoding` ヘッダーを使用して要求が行われると、エンコードタイプは無視され、レスポンスは圧縮されません。

たとえば、`Accept-Encoding: gzip` ヘッダーを使用して Batch Results 要求が行われた場合、レスポンスは次のようになります。

```
HTTP/1.1 200 OK
Date: Tue, 09 Oct 2012 18:36:45 GMT
Content-Type: text/csv; charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked

...compressed response body...
```

Bulk API は、レスポンスの圧縮の HTTP 1.1 標準に従います。ほとんどのクライアントは圧縮レスポンスを自動的にサポートします。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。

第 4 章　データファイルの準備

トピック:

Bulk API は、CSV ファイル、XML ファイル、または JSON ファイルのいずれかに格納されたレコードを処理します。

メモ: 最高のパフォーマンスを得るには、データファイルを CSV、JSON、XML の優先順位に設定することをお勧めします。

バイナリ添付ファイルを含むレコードの読み込みについては、「バイナリ添付ファイルの読み込み」 (ページ 28)を参照してください。

サードパーティソースからのデータの読み込みについては、「データ項目の対応付け」 (ページ 108)を参照してください。

項目名の確認

クライアントの設定が完了したら、Bulk API を使用するクライアントアプリケーションを構築できます。次のサンプルを使用して、クライアントアプリケーションを作成します。各セクションで、コードの特定部分を順に説明していきます。詳細なサンプルは最後に記載します。

実行できる操作

- 『*SOAP API 開発者ガイド*』に記載されている `describeSObjects()` コールを使用する
- Salesforce の [設定] ページを使用する
- *オブジェクト参照*の、該当するオブジェクトの項目を参照する (各オブジェクトの項目名、データ型、説明がリストされています)

Salesforce の [設定] ページを使用する

オブジェクトの項目名を検索する手順は、次のとおりです。

1. オブジェクトの管理設定から、項目領域に移動します。
2. 項目の［項目の表示ラベル］をクリックします。

標準項目の場合は、項目の詳細ページで［項目名］を確認します。この値が CSV ファイルの列ヘッダーとして使用されます。

カスタム項目の場合は、［API 参照名］を確認します。この値が、CSV ファイル内の項目列ヘッダー、また、XML または JSON ファイル内の項目名 ID として使用されます (API 参照名を確認するには、項目名をクリックします)。

関連トピック:

Salesforce ヘルプ: オブジェクト管理設定の検索

レコードで有効な日付形式

`dateTime` 型の項目を使用するときは、*`yyyy-MM-ddTHH:mm:ss.SSS+/-HHmm`* または *`yyyy-MM-ddTHH:mm:ss.SSSZ`* という形式を使用します。

- `yyyy` は 4 桁の年号
- `MM` は 2 桁の月 (01 ～ 12)
- `dd` は 2 桁の日付 (01 ～ 31)
- 「T」はこの後に時刻が記述されることを示す区切り文字
- `HH` は 2 桁の時間 (00 ～ 23)
- `mm` は 2 桁の分 (00 ～ 59)
- `ss` は 2 桁の秒 (00 ～ 59)
- `SSS` は省略可能な 3 桁のミリ秒 (000 ～ 999)
- `+/-HHmm` は、Zulu (UTC) タイムゾーンオフセット

- 「Z」はタイムゾーンが UTC であることを示す

UTC の日付時刻にタイムゾーンオフセット値を追加すると、日付と時刻をそのタイムゾーンの値で示せます。たとえば、「2002-10-10T12:00:00+05:00」は「2002-10-10T12:00:00Z」より 5 時間早いことを示します (UTC の「2002-10-10T07:00:00Z」)。「2002-10-10T00:00:00+05:00」は「2002-10-10T00:00:00Z」より 5 時間早いことを示します (UTC の「2002-10-09T19:00:00Z」)。「W3C XML Schema Part 2: DateTime Datatype」を参照してください。

`date` 型の項目を使用するときは、*`yyyy-MM-dd+/-HHmm`* または *`yyyy-MM-ddZ`* という形式を使用します。「W3C XML Schema Part 2: Date Datatype」を参照してください。

CSV ファイルの準備

CSV ファイルのヘッダー行には、処理するオブジェクトの項目名を記述します。2 行目以降は、Salesforce の各レコードに対応します。レコードは一連の項目で構成され、項目と項目の間はカンマで区切られます。1 つの CSV ファイルには複数のレコードを含めることができ、これがバッチで使用されます。

CSV ファイル内のレコードは、すべて同一オブジェクトのレコードである必要があります。オブジェクトは、バッチに関連付けられたジョブで指定します。ある特定のジョブに関連付けられたすべてのバッチに含まれるレコードは、すべて同一オブジェクトのものである必要があります。

Bulk API で CSV ファイルを処理する場合は、次の点に注意します。

- Bulk API はカンマ以外の区切り文字をサポートしません。
- Bulk API は、大量データの処理用に最適化されており、CSV ファイルの形式を厳格に定義しています。「CSV のレコード行の有効な形式」(ページ 21)を参照してください。CSV ファイルを処理する最も簡単な方法は、データローダで Bulk API を有効にすることです。
- レコードを作成するときは、必須項目はもれなく含める必要があります。その他の項目は、必要に応じて含めることができます。
- レコードの更新では、CSV ファイルで定義されていない項目は無視されます。
- ファイルでは UTF-8 形式を使用します。

ヒント: これらのルールを満たしていない CSV ファイルからデータをインポートするには、CSV ファイルのデータ項目を Salesforce データ項目に対応付けます。「データ項目の対応付け」(ページ 108)を参照してください。

ヘッダー行のリレーション項目

Salesforce のオブジェクトの多くは、別のオブジェクトに関連付けられています。たとえば、取引先は取引先責任者の親オブジェクトです。CSV ファイルでは、列ヘッダーでリレーションを示すことにより、関連付けられたオブジェクトへの参照を追加できます。Bulk API でレコードを処理する場合、CSV ファイルの列ヘッダーで ***`RelationshipName.IndexedFieldName`*** という構文を使用して、オブジェクトとその親オブジェクトのリレーションを記述します。*`RelationshipName`* は項目のリレーション名、*`IndexedFieldName`* は親レコードを一意に識別するインデックス化された項目の名前を示します。項目の `relationshipName` プロパティの値を取得するには、SOAP ベースの SOAP API で `describeSObjects()` コールを使用します。

別のオブジェクトではなく、同じオブジェクトに関連付けられるというケースもあります。たとえば、取引先責任者の［上司］項目は、別の取引先責任者を参照します。このような場合、列ヘッダーで`ReportsTo.Email` と記述すると、［メール］項目の値に基づいて、各取引先責任者の［上司］項目を一意に識別できます。列ヘッ

ダーの ReportsTo の部分は、[上司] 項目の relationshipName プロパティの値です。リレーションを使用した CSV ファイルの例を次に示します。

```
FirstName,LastName,ReportsTo.Email
Tom,Jones,buyer@salesforcesample.com
```

CSV ファイルのヘッダー行でリレーションを参照する場合は、次の点に注意します。

- 子から親へのリレーションを使用できます。親から子へのリレーションは使用できません。
- 子から親へのリレーションを拡張して使用することはできません。つまり、子から親へのリレーションを参照し、さらに親からその親へのリレーションを参照することはできません。
- 親の識別では、インデックス化された項目のみを使用できます。カスタム項目は、外部 ID 属性が選択されている [外部 ID] 項目である場合、インデックス化されています。標準項目は、idLookup プロパティの値が true に設定されている場合、インデックス化されています。詳細は、「標準オブジェクト」で、各オブジェクトの項目リストの「項目のプロパティ」列を参照してください。

カスタムオブジェクトのリレーション項目

カスタムオブジェクトでは、カスタム項目を使用してオブジェクト間のリレーションを追跡します。2 つのカスタムオブジェクト間のリレーションを示すには、__r (アンダースコア 2 つと「r」) で終わるリレーション名を使用します。列ヘッダーでこのリレーションを指定して、関連オブジェクトへの参照を追加できます。

たとえば、子オブジェクトに、あるカスタムオブジェクトを親として参照する Mother_Of_Child__c という [API 参照名] を持つカスタム項目があり、親となるオブジェクトに External_ID__c という [API 参照名] を持つ項目がある場合は、Mother_Of_Child__r.External_ID__c という列ヘッダーを記述することにより、親オブジェクトの [外部 ID] 項目を使用して、[Mother Of Child] 項目を一意に識別できます。列ヘッダー内のリレーション名では、子オブジェクトのカスタム項目の__cを__rで置き換えた値を入力します。リレーションについての詳細は、www.salesforce.com/us/developer/docs/soql_sosl/index.htm にある『*Salesforce SOQL および SOSL リファレンスガイド*』の「リレーション名について」を参照してください。

リレーションを使用した CSV ファイルの例を次に示します。

```
Name,Mother_Of_Child__r.External_ID__c
CustomObject1,123456
```

多態的な項目でのリレーション

多態的な項目では、2 種類以上のオブジェクトを親として参照できます。たとえば、取引先責任者またはリードを ToDo の親にすることができます。つまり、ToDo の WhoId 項目に取引先責任者またはリードの ID を含めることができます。多態的な項目は柔軟性が高く、列ヘッダーの構文に、親オブジェクトの種別を定義する要素を追加できるようになっています。この構文は ***ObjectType*:*RelationshipName*.*IndexedFieldName*** です。後続のサンプルには、次の 2 つの参照項目が含まれています。

1. WhoId: これは多態的な項目であり、Who という relationshipName が使用されています。ここでは、リードを参照し、インデックス化された [メール] 項目を使用して親レコードを一意に識別します。

2. OwnerId: この項目は多態性を持たず、Owner という relationshipName が使用されています。ユーザを参照し、インデックス化された Id 項目を使用して親レコードを一意に識別します。

```
Subject,Priority,Status,Lead:Who.Email,Owner.Id
Test Bulk API polymorphic reference field,Normal,Not
Started,lead@salesforcesample.com,005D0000001AXYz
```

警告: 項目の列ヘッダーで使用する *ObjectType*: という要素は、多態的な項目でのみ必須となります。多態的な項目でこの要素を省略するとエラーが返されますが、それ以外の項目でこの要素を使用した場合もエラーが返されます。

CSV のレコード行の有効な形式

Bulk API は、大量データの処理用に最適化されており、項目の値の形式に厳格な制限を設けています。

Salesforce レコードを含む CSV ファイルを作成する場合は、次の点に注意してください。

- 行内で項目の値を区切る場合は、必ずカンマを使用します。
- 値にカンマ、改行文字、二重引用符などが含まれる場合は、二重引用符で囲む必要があります。たとえば、"Director of Operations, Western Region"のように記述します。
- 値に二重引用符が含まれる場合は、それぞれの二重引用符の前にもう 1 つ二重引用符を挿入してエスケープする必要があります。たとえば、"This is the ""gold"" standard" のように記述します。
- 値は切り詰められることはありません。区切り文字のカンマの前後に空白文字が挿入されている場合、それらも値の一部とみなされます。ただし、二重引用符の前後に空白文字が挿入されている場合は、その行がエラーになります。たとえば、John,Smith は有効な値です。John, Smith も、空白文字が含まれていますが有効です。一方、" Smith"、"John", "Smith" は無効な値となります。
- 項目の値が空である場合、レコードの更新時にはその項目は無視されます。値を null に設定するには、項目値 #N/A を使用します。
- double 型の項目には、小数値を含めることができます。W3C XML Schema Part 2: Datatypes Second Edition に記載されているとおり、相応に大きな数値の場合は、値を科学的記数法で保存できます。

サンプルの CSV ファイル

次に示すサンプルの CSV ファイルには、取引先責任者オブジェクトのレコードが 2 つ含まれています。各レコードの項目は6つです。処理対象のオブジェクトの項目であればすべて追加できます。このファイルを使用して既存のレコードを更新する場合、CSV ファイル内に定義されていない項目は無視されます。レコードを作成するときは、必須項目はもれなく含める必要があります。

```
FirstName,LastName,Title,ReportsTo.Email,Birthdate,Description
Tom,Jones,Senior Director,buyer@salesforcesample.com,1940-06-07Z,"Self-described as ""the
 top"" branding guru on the West Coast"
Ian,Dury,Chief Imagineer,cto@salesforcesample.com,,"World-renowned expert in fuzzy logic
design.
Influential in technology purchases."
```

2 件目のレコードの `Description` 項目の値には改行が含まれているため、二重引用符で囲んでいます。

関連トピック:

- サンプルの XML ファイル
- サンプルの JSON ファイル

XML ファイルおよび JSON ファイルの準備

Bulk API は、XML ファイル、JSON ファイル、または CSV ファイルに記述されたレコードを処理します。1 つの XML ファイルまたは JSON ファイルには複数のレコードを含めることができ、これがバッチで使用されます。XML ファイルでは、レコードは `sObjects` タグで定義されます。

XML ファイルまたは JSON ファイル内のレコードは、すべて同一オブジェクトのレコードである必要があります。オブジェクトは、バッチに関連付けられたジョブで指定します。ある特定のジョブに関連付けられたすべてのバッチに含まれるレコードは、すべて同一オブジェクトのものである必要があります。

Bulk API で XML ファイルまたは JSON ファイルを処理する場合は、次の点に注意します。

- レコードを作成するときは、必須項目はもれなく含める必要があります。その他の項目は、必要に応じて含めることができます。
- レコードの更新では、ファイルで定義されていない項目は無視されます。
- ファイルでは UTF-8 形式を使用します。

レコード内のリレーション項目

Salesforce のオブジェクトの多くは、別のオブジェクトに関連付けられています。たとえば、取引先は取引先責任者の親オブジェクトです。別のオブジェクトではなく、同じオブジェクトに関連付けられるというケースもあります。たとえば、取引先責任者の［上司］項目は、別の取引先責任者を参照します。

XML ファイルまたは JSON ファイルでは、次に示す構文を使用して、関連付けられたオブジェクトへの参照を追加できます。*`RelationshipName`* は項目のリレーション名、*`IndexedFieldName`* は親レコードを一意に識別するインデックス化された項目の名前を示します。

```
<RelationshipName>
    <sObject>
        <IndexedFieldName>rwilliams@salesforcesample.com</IndexedFieldName>
    </sObject>
</RelationshipName>
```

項目の `relationshipName` プロパティの値を取得するには、SOAP ベースの SOAP API で `describeSObjects()` コールを使用します。さらに、親レコードを一意に識別するために、インデックス化された項目を指定する必要があります。標準項目は、`idLookup` プロパティの値が `true` に設定されている場合、インデックス化されています。

次のサンプルには、［上司］項目を含む取引先責任者レコードが記述されています。この項目は、別の取引先責任者レコードを参照します。`ReportsTo` は、［上司］項目の `relationshipName` プロパティの値です。この例では、取引先責任者レコードの［上司］項目が参照する親レコードも取引先責任者レコードであり、`Email` 項目の値に基づいて親レコードを識別します。`Email` 項目の `idLookup` プロパティの値は `true` であるため、

この項目はインデックス化されています。項目に idLookup プロパティが存在するかどうかを確認するには、「標準オブジェクト」で、各オブジェクトの項目リストの「項目のプロパティ」列を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <FirstName>Ray</Name>
      <LastName>Riordan</Description>
      <ReportsTo>
        <sObject>
          <Email>rwilliams@salesforcesample.com</Email>
        </sObject>
      </ReportsTo>
   </sObject>
</sObjects>
```

XML ファイルまたは JSON ファイルでリレーションを使用する場合は、次の点に注意します。

- 子から親へのリレーションを使用できます。親から子へのリレーションは使用できません。
- 子から親へのリレーションを拡張して使用することはできません。つまり、子から親へのリレーションを参照し、さらに親からその親へのリレーションを参照することはできません。

カスタムオブジェクトのリレーション項目

カスタムオブジェクトでは、カスタム項目を使用してオブジェクト間のリレーションを追跡します。2つのカスタムオブジェクト間のリレーションを示すには、__r （アンダースコア 2 つと「r」）で終わるリレーション名を使用します。関連オブジェクトへの参照は、インデックス化された項目を使用して追加できますが、カスタム項目は、外部 ID 属性が選択されている［外部 ID］項目である場合、インデックス化されています。

たとえば、子オブジェクトに、あるカスタムオブジェクトを親として参照する Mother_Of_Child__c という［API 参照名］を持つカスタム項目があり、親となるオブジェクトに External_ID__c という［API 参照名］を持つ項目がある場合は、項目のリレーション名である Mother_Of_Child__r の relationshipName プロパティによって、リレーションを参照していることを示します。さらに、親オブジェクトの［外部 ID］項目を使用して、［子の母親］項目を一意に識別します。リレーション名には、子オブジェクトのカスタム項目の__cを__rで置き換えた値を使用します。リレーションについての詳細は、www.salesforce.com/us/developer/docs/soql_sosl/index.htm にある『*Salesforce SOQL および SOSL リファレンスガイド*』の「リレーション名について」を参照してください。

リレーションを使用した XML ファイルの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <Name>CustomObject1</Name>
      <Mother_Of_Child__r>
        <sObject>
          <External_ID__c>123456</External_ID__c>
        </sObject>
        </Mother_Of_Child__r>
   </sObject>
</sObjects>
```

多態的な項目でのリレーション

多態的な項目では、2 種類以上のオブジェクトを親として参照できます。たとえば、取引先責任者またはリードを ToDo の親にすることができます。つまり、ToDo の `WhoId` 項目に取引先責任者またはリードの ID を含めることができます。多態的な項目は柔軟性が高く、リレーション項目の構文に、親オブジェクトの種別を定義する要素を追加できるようになっています。次に XML ファイルの例を示します。*`RelationshipName`* は項目のリレーション名、*`ObjectTypeName`* は親レコードのオブジェクトの種別、*`IndexedFieldName`* は親レコードを一意に識別するインデックス化された項目の名前を示します。

```
<RelationshipName>
    <sObject>
        <type>ObjectTypeName</type>
        <IndexedFieldName>rwilliams@salesforcesample.com</IndexedFieldName>
    </sObject>
</RelationshipName>
```

後続のサンプルには、次の 2 つの参照項目が含まれています。

1. `WhoId`: これは多態的な項目であり、`Who` という `relationshipName` が使用されています。ここでは、リードを参照し、インデックス化された `[メール]` 項目を使用して親レコードを一意に識別します。
2. `OwnerId`: この項目は多態性を持たず、`Owner` という `relationshipName` が使用されています。ユーザを参照し、インデックス化された `Id` 項目を使用して親レコードを一意に識別します。

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <Subject>Test Bulk API polymorphic reference field</Subject>
      <Priority>Normal</Priority>
      <Status>Not Started</Status>
      <Who>
        <sObject>
          <type>Lead</type>
          <Email>lead@salesforcesample.com</Email>
        </sObject>
      </Who>
      <Owner>
        <sObject>
          <Id>005D0000001AXYz</Id>
        </sObject>
      </Owner>
   </sObject>
</sObjects>
```

警告: `<type>ObjectTypeName</type>` 要素は、多態的な項目でのみ必須となります。多態的な項目でこの要素を省略するとエラーが返されますが、それ以外の項目でこの要素を使用した場合もエラーが返されます。

XML ファイルおよび JSON ファイルでのレコードの有効な形式

1 つの XML ファイルまたは JSON ファイル内のバッチ要求には、1 つのオブジェクト種別のレコードのみ含めることができます。XML ファイル内の各バッチでは、`sObject` タグを使用してレコードを記述します。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <field_name>field_value</field_name>
      ...
   </sObject>
   <sObject>
      <field_name>field_value</field_name>
      ...
   </sObject>
</sObjects>
```

JSON ファイル内の各バッチでは、JSON オブジェクトを使用してレコードを記述します。次に例を示します。

```
[
   {
      "field_name" : "field_value"
      ...
   }
   {
      "field_name" : "field_value"
      ...
   }
]
```

メモ: 多態的な項目ではすべてのバッチで `type` タグを含めます。それ以外の項目ではこれを除外する必要があります。この規則に従わない場合、バッチは失敗します。多態的な項目では、2 種類以上のオブジェクトを親として参照できます。たとえば、取引先責任者またはリードを ToDo の親にすることができます。つまり、ToDo の `WhoId` 項目に取引先責任者またはリードの ID を含めることができます。

XML ファイルまたは JSON ファイルでレコードを作成するときは、次の点に注意します。

- レコードの更新では、ファイルで定義されていない項目は無視されます。XML で値を `null` に設定するには、項目で `xsi:nil` の値を `true` に設定します。たとえば、`<description xsi:nil="true"/>` と記述すると、Description 項目の値が `null` に設定されます。JSON では、単に項目値を null に設定します。たとえば、「`"description" : null`」などです。
- `double` 型の項目には、小数値を含めることができます。W3C XML Schema Part 2: Datatypes Second Edition に記載されているとおり、相応に大きな数値の場合は、値を科学的記数法で保存できます。

サンプルの XML ファイル

次に示すサンプルの XML ファイルには、Account オブジェクトのレコードが 2 つ含まれています。各レコードの項目は3つですが、処理対象のオブジェクトの項目であればすべて追加できます。このファイルを使用して既存のレコードを更新する場合、XML ファイル内に定義されていない項目は無視されます。レコードを作成するときは、必須項目はもれなく含める必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <Name>Xytrex Co.</Name>
      <Description>Industrial Cleaning Supply Company</Description>
      <Account Number>ABC15797531</Account Number>
   </sObject>
   <sObject>
      <Name>Watson and Powell, Inc.</Name>
      <Description>Law firm. New York Headquarters</Description>
      <Account Number>ABC24689753</Account Number>
   </sObject>
</sObjects>
```

関連トピック:

- サンプルの CSV ファイル
- サンプルの JSON ファイル

サンプルの JSON ファイル

次に示すサンプルの JSON ファイルには、Account オブジェクトのレコードが 2 つ含まれています。各レコードの項目は3つですが、処理対象のオブジェクトの項目であればすべて追加できます。このファイルを使用して既存のレコードを更新する場合、JSON ファイル内に定義されていない項目は無視されます。レコードを作成するときは、必須項目はもれなく含める必要があります。

次に示すサンプルの JSON ファイルには、Account オブジェクトのレコードが 2 つ含まれています。各レコードの項目は3つですが、処理対象のオブジェクトの項目であればすべて追加できます。このファイルを使用して既存のレコードを更新する場合、JSON ファイル内に定義されていない項目は無視されます。レコードを作成するときは、必須項目はもれなく含める必要があります。

```
[
   {
      "Name" : "Xytrex Co.",
      "Description" : "Industrial Cleaning Supply Company",
      "Account Number" : "ABC15797531"
   }
   {
      "Name" : "Watson and Powell, Inc.",
      "Description" : "Law firm. New York Headquarters",
      "Account Number" : "ABC24689753"
```

```
    }
]
```

関連トピック:

- サンプルの CSV ファイル
- サンプルの XML ファイル

第 5 章　バイナリ添付ファイルの読み込み

トピック:

Bulk API では、バイナリ添付ファイルの読み込みも実行できます。添付ファイルオブジェクトと Salesforce CRM Content のコンテンツがサポートされます。

request.txt ファイルの作成

バイナリ添付ファイルの読み込みでは、バッチは zip ファイルにまとめられます。この zip ファイルには、バイナリ添付ファイルへの参照が含まれた request.txt という名前のファイルが、CSV、XML、または JSON のいずれかの形式で格納されます。バイナリ添付ファイル自体も格納されます。バイナリ添付ファイルを含まない CSV、XML、または JSON バッチファイルとの違いに注意してください。こうしたバッチファイルでは、zip ファイルや request.txt ファイルは必要ありません。

request.txt ファイルは zip のベースディレクトリに格納されます。バイナリ添付ファイル自体は、ベースディレクトリに格納することも、サブディレクトリに整理して格納することもできます。request.txt ファイルは、zip に格納されたバイナリ添付ファイルのマニフェストファイルであり、バイナリ添付ファイルを参照する各レコードのデータを保存します。

メモ: CSV、XML、または JSON のいずれを使用する場合も、バッチデータファイルの名前は request.txt になります。

添付ファイルオブジェクトでは、次の項目を適切な表記で記述することが非常に重要です。

- Name:バイナリ添付ファイルのファイル名を記述します。バッチに含める添付ファイルについて、一意の名前を取得するもっとも簡単な方法は、ベースディレクトリからバイナリ添付ファイルへの相対パスを使用することです。たとえば、attachment1.gif または subdir/attachment2.doc です。
- Body:バイナリ添付ファイルへの相対パスの先頭に # 記号を付けて記します。たとえば、#attachment1.gif または #subdir/attachment2.doc です。
- ParentId: 添付ファイルの親レコード (取引先、ケースなど) を表す ID を記述します。

バッチファイルには、上記以外の添付ファイルオブジェクトの項目 (例: Description など) も含めることができます。詳細は、「Attachment」を参照してください。

サンプルの CSV request.txt ファイル

次に示す CSV のサンプルファイルには、添付ファイルオブジェクトの 2 つのレコードが含まれています。1 つ目のレコードは、zip ファイルのベースディレクトリにあるバイナリ添付ファイル attachment1.gif を参照しています。2 つ目のレコードは、zip ファイルのサブディレクトリ subdir にあるバイナリ添付ファイル attachment2.doc を参照しています。ParentId 項目を見ると、どちらの添付ファイルレコードも親レコードとして取引先を参照していることがわかります。なお、*Account ID* の部分は、親となる取引先レコードの Id によって置き換えます。

```
Name,ParentId,Body
attachment1.gif,Account Id,#attachment1.gif
subdir/attachment2.doc,Account Id,#subdir/attachment2.doc
```

サンプルの XML request.txt ファイル

次に示す XML のサンプルファイルには、先ほどの CSV のサンプルファイルと同じ 2 つのレコードが含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects
```

```
   xmlns="http://www.force.com/2009/06/asyncapi/dataload"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sObject>
    <Name>attachment1.gif</Name>
    <ParentId>Account Id</ParentId>
    <Body>#attachment1.gif</Body>
  </sObject>
  <sObject>
    <Name>subdir/attachment2.doc</Name>
    <ParentId>Account Id</ParentId>
    <Body>#subdir/attachment2.doc</Body>
  </sObject>
</sObjects>
```

サンプルの JSON `request.txt` ファイル

次に示す JSON のサンプルファイルには、先ほどの例と同じ 2 つのレコードが含まれています。

```
[
  {
    "Name" : "attachment1.gif",
    "ParentId" : "Account Id",
    "Body" : "#attachment1.gif"
  }
  {
    "Name" : "subdir/attachment2.doc",
    "ParentId" : "Account Id",
    "Body" : "#subdir/attachment2.doc"
  }
]
```

関連トピック:

バイナリ添付ファイルを含む Zip バッチファイルの作成

バイナリ添付ファイルを含む Zip バッチファイルの作成

バイナリ添付ファイルをバッチとして送信する zip バッチファイルを作成できます。

1. バイナリ添付ファイルを含むベースディレクトリを作成します。ファイルはサブディレクトリに整理して格納することもできます。
2. ベースディレクトリ内に CSV、XML、または JSON のいずれかの形式で、`request.txt` という名前でファイルを作成します。`request.txt` ファイルは、zip ファイルに格納されたバイナリ添付ファイルのマニフェストファイルであり、バイナリ添付ファイルを参照する各レコードのデータを保存します。
3. ベースディレクトリとサブディレクトリ (作成した場合) から zip ファイルを作成します。

関連トピック:

request.txt ファイルの作成

バイナリ添付ファイルを含むバッチに対するジョブの作成

添付ファイルレコードを含んだバッチを処理するためのジョブを、cURL を使用して作成できます。

詳細は、「cURL を使用した HTTP 要求の送信」(ページ 7)を参照してください。

1. job.txt という名前のテキストファイルを作成し、次のテキストを含めます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <operation>insert</operation>
    <object>Attachment</object>
    <contentType>ZIP_CSV</contentType>
</jobInfo>
```

   メモ: このジョブで処理するバッチは CSV 形式のデータを含むため、contentType が ZIP_CSV と設定されています。バッチが XML または JSON 形式のデータを含む場合は、それぞれ ZIP_XML または ZIP_JSON と設定します。

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行します。

   ```
   curl https://instance.salesforce.com/services/async/37.0/job -H "X-SFDC-Session:
   sessionId" -H "Content-Type: application/xml; charset=UTF-8" -d @job.txt
   ```

   instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。ログインについての詳細は、「ステップ 1:SOAP API を使用したログイン」(ページ 7)を参照してください。

   Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D000000001SRIAY</id>
  <operation>insert</operation>
  <object>Attachment</object>
  <createdById>005D0000001B0VkIAK</createdById>
  <createdDate>2010-08-25T18:52:03.000Z</createdDate>
  <systemModstamp>2010-08-25T18:52:03.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>ZIP_CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>37.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

3. <id> 要素内に返されたジョブ ID の値をメモしておいてください。後続の処理で使用します。

関連トピック:

バイナリ添付ファイルを含むバッチの作成
ジョブの作成

バイナリ添付ファイルを含むバッチの作成

ジョブを作成したら、次に添付ファイルレコードのバッチを作成します。データをバッチにまとめ、複数の HTTP POST 要求に分割して送信します。ここでは、バッチを 1 つのみ作成して送信します。

データを複数のバッチに分割する方法については、「データ読み込みに関する一般的なガイドライン」(ページ 14)を参照してください。

1. zip のバッチファイルを作成します。この例では、request.zip という名前を付けます。
2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行します。

```
curl https://instance.salesforce.com/services/async/37.0/job/jobId/batch -H
"X-SFDC-Session: sessionId" -H "Content-Type:zip/csv" --data-binary @request.zip
```

instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。*jobId* は、ジョブ作成時に返されたジョブ ID です。

メモ: POST 要求の Content-type は zip/csv です。バッチが XML または JSON 形式のデータを含む場合は、zip/xml または zip/json と設定します。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751D000000003uwIAA</id>
  <jobId>750D000000001TyIAI</jobId>
  <state>Queued</state>
  <createdDate>2010-08-25T21:29:55.000Z</createdDate>
  <systemModstamp>2010-08-25T21:29:55.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Salesforce では CSV のコンテンツは解析しませんが、代わりに後続の処理でバッチを検証します。この応答は単にバッチが受信されたことを知らせるためのものです。

3. <id> 要素内で返されたバッチ ID の値をメモしておいてください。このバッチ ID は、後でバッチの状況を確認するときに使用します。

ジョブを終了後、バッチの状況を確認してバッチ結果を取得する手順は、「クイックスタート」を参照してください。

関連トピック:

- バイナリ添付ファイルを含むバッチに対するジョブの作成
- ジョブへのバッチの追加

第 6 章　要求の基本事項

トピック:

処理を実行するときに使用する URI の形式や、セッションヘッダーを使用して要求を認証する方法など、Bulk API 要求の基本事項について説明します。

URI について

Bulk API で処理を実行するには、URI に HTTP 要求を送信します。

HTTP 要求の送信先の URI の形式は次のとおりです。

Web_Services_SOAP_endpoint_instance_name/services/async/*APIversion*/*Resource_address*

URI の API のバージョンの部分までが、すべての処理に共通するベース URI になります。API のバージョンより後の部分(*Resource_address*)は、処理対象のジョブやバッチによって変わってきます。たとえば、インスタンスでバージョン 37.0 の Bulk API を使用している場合、実際のベース URI は `https://`***yourInstance***`.salesforce.com/services/async/37.0` となります。

組織のインスタンス名は、LoginResult の `serverUrl` 項目で返されます。

関連トピック:

- ジョブでの作業
- バッチでの作業

セッションヘッダーの設定

すべての HTTP 要求には、SOAP API の `login()` コールで取得した有効な API セッション ID を含める必要があります。セッション ID は、SessionHeader によって返されます。

`login()` コールで有効なセッション ID を取得したら、必要な情報を指定します。次に例を示します。

```
POST /service/async/37.0/job/ HTTP/1.1
Content-Type: application/xml; charset=UTF-8
Accept: application/xml
User-Agent: Salesforce Web Service Connector For Java/1.0
X-SFDC-Session: sessionId
Host: yourInstance.salesforce.com
Connection: keep-alive
Content-Length: 135

<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <operation>insert</operation>
 <object>Account</object>
</jobInfo>
```

関連トピック:

- クイックスタート
- Java を使用したサンプルクライアントアプリケーション

第 7 章　ジョブでの作業

トピック:

レコードセットを処理するには、1 つ以上のバッチを含むジョブを作成します。このジョブは、処理されるオブジェクトと使用されるアクションの種別 (クエリ、挿入、更新/挿入、更新、または削除) を指定します。

ジョブは JobInfo リソースで表されます。このリソースは、新規ジョブの作成、既存のジョブの状況の取得、ジョブの状況の変更に使用します。

ジョブの作成

ジョブを作成するには、次のようなURI宛てにPOST要求を送信します。リクエストボディでは、関連付けられたすべてのバッチで処理されるオブジェクトの種別を指定します。

URI

```
https://instance_name—api.salesforce.com/services/async/APIversion/job
```

XML リクエストボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <operation>insert</operation>
 <object>Account</object>
 <contentType>CSV</contentType>
</jobInfo>
```

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000004SkLIAU</id>
  <operation>insert</operation>
  <object>Account</object>
  <createdById>005D0000001b0fFIAQ</createdById>
  <createdDate>2015-12-15T21:41:45.000Z</createdDate>
  <systemModstamp>2015-12-15T21:41:45.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

JSON リクエストボディの例

```
{
  "operation" : "insert",
  "object" : "Account",
  "contentType" : "CSV"
}
```

JSON レスポンスボディの例

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 0,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005D0000001b0fFIAQ",
   "createdDate" : "2015-12-15T20:45:25.000+0000",
   "id" : "750D00000004SkGIAU",
   "numberBatchesCompleted" : 0,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 0,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 0,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Open",
   "systemModstamp" : "2015-12-15T20:45:25.000+0000",
   "totalProcessingTime" : 0
}
```

これらのサンプルの `contentType` 項目は、CSV形式のバッチがジョブに関連付けられることを示しています。別の形式 (XML や JSON など) については、「JobInfo」 (ページ 77)を参照してください。

警告: この operation 項目の値はすべて小文字で入力してください。たとえば、`insert` の代わりに `INSERT` と入力した場合、エラーが発生します。

関連トピック:

- バイナリ添付ファイルを含むバッチに対するジョブの作成
- ジョブの詳細の取得
- ジョブの終了
- ジョブの中止
- ジョブへのバッチの追加
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- JobInfo
- クイックスタート

ジョブの監視

Salesforceでは、Bulk APIのジョブを監視できます。監視ページでは、データローダや独自開発のアプリケーションを含むすべてのクライアントアプリケーションで作成したジョブとバッチを追跡できます。

処理中または最近完了したデータの一括読み込みジョブ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「*一括データ読み込みジョブ*」と入力し、[一括データ読み込みジョブ] を選択します。

詳細は、Salesforce オンラインヘルプの「一括データ読み込みジョブの監視」を参照してください。

関連トピック:

- ジョブの作成
- ジョブの詳細の取得
- ジョブの終了
- ジョブの中止
- ジョブへのバッチの追加
- ジョブとバッチの有効期限
- Bulk API の制限

ジョブの終了

ジョブを終了するには、次のような URI 宛てに POST 要求を送信します。要求 URI では、終了するジョブを指定します。終了したジョブに、それ以上バッチを追加することはできません。

URI

`https://`***instance_name***`—api.salesforce.com/services/async/`***APIversion***`/job/`***jobId***

XML リクエストボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <state>Closed</state>
</jobInfo>
```

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>750D0000000002lIAA</id>
 <operation>insert</operation>
 <object>Account</object>
 <createdById>005D0000001ALVFIA4</createdById>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
 <state>Closed</state>
 <concurrencyMode>Parallel</concurrencyMode>
 <contentType>XML</contentType>
```

```
 <numberBatchesQueued>0</numberBatchesQueued>
 <numberBatchesInProgress>0</numberBatchesInProgress>
 <numberBatchesCompleted>1</numberBatchesCompleted>
 <numberBatchesFailed>0</numberBatchesFailed>
 <numberBatchesTotal>1</numberBatchesTotal>
 <numberRecordsProcessed>2</numberRecordsProcessed>
 <numberRetries>0</numberRetries>
 <apiVersion>36.0</apiVersion>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>3647</totalProcessingTime>
 <apiActiveProcessingTime>2136</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

JSON リクエストボディの例

```
{
   "state" : "Closed"
}
```

JSON レスポンスボディの例

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 5059,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005xx000001SyhGAAS",
   "createdDate" : "2015-11-19T01:45:03.000+0000",
   "id" : "750xx000000000GAAQ",
   "numberBatchesCompleted" : 10,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 10,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 100,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Closed",
   "systemModstamp" : "2015-11-19T01:45:03.000+0000",
```

```
   "totalProcessingTime" : 5759
}
```

関連トピック:

- ジョブの作成
- ジョブの監視
- ジョブの詳細の取得
- ジョブの中止
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- JobInfo
- クイックスタート

ジョブの詳細の取得

既存のジョブのすべての詳細を取得するには、次のような URI 宛てに GET 要求を送信します。

URI

```
https://instance_name—api.salesforce.com/services/async/APIversion/job/jobId
```

リクエストボディの例

リクエストボディは記述しません。

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000004SkLIAU</id>
  <operation>insert</operation>
  <object>Account</object>
  <createdById>005D0000001b0fFIAQ</createdById>
  <createdDate>2015-12-15T21:41:45.000Z</createdDate>
  <systemModstamp>2015-12-15T21:41:45.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
```

```
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

JSON レスポンスボディの例

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 0,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005D0000001b0fFIAQ",
   "createdDate" : "2015-12-15T20:45:25.000+0000",
   "id" : "750D00000004SkGIAU",
   "numberBatchesCompleted" : 0,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 0,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 0,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Open",
   "systemModstamp" : "2015-12-15T20:45:25.000+0000",
   "totalProcessingTime" : 0
}
```

関連トピック:

- ジョブの作成
- ジョブの監視
- ジョブの終了
- ジョブの中止
- ジョブへのバッチの追加
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- JobInfo
- クイックスタート

ジョブの中止

既存のジョブを中止するには、次のような URI 宛てに POST 要求を送信します。要求 URI では、中止するジョブを指定します。ジョブが中止されると、それ以降レコードは処理されません。なお、すでに確定されたデータ変更が元に戻ることはありません。

URI

https://***instance_name***—api.salesforce.com/services/async/***APIversion***/job/***jobId***

XML リクエストボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <state>Aborted</state>
</jobInfo>
```

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>750D0000000002lIAA</id>
 <operation>insert</operation>
 <object>Account</object>
 <createdById>005D0000001ALVFIA4</createdById>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:16:00.000Z</systemModstamp>
 <state>Aborted</state>
 <concurrencyMode>Parallel</concurrencyMode>
 <contentType>XML</contentType>
 <numberBatchesQueued>0</numberBatchesQueued>
 <numberBatchesInProgress>0</numberBatchesInProgress>
 <numberBatchesCompleted>1</numberBatchesCompleted>
 <numberBatchesFailed>0</numberBatchesFailed>
 <numberBatchesTotal>1</numberBatchesTotal>
 <numberRecordsProcessed>2</numberRecordsProcessed>
 <numberRetries>0</numberRetries>
 <apiVersion>36.0</apiVersion>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>3647</totalProcessingTime>
 <apiActiveProcessingTime>2136</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

JSON リクエストボディの例

```
{
   "state" : "Aborted"
}
```

JSON レスポンスボディの例

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 2166,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005D0000001b0fFIAQ",
   "createdDate" : "2015-12-15T21:54:29.000+0000",
   "id" : "750D00000004SkVIAU",
```

```
    "numberBatchesCompleted" : 2,
    "numberBatchesFailed" : 0,
    "numberBatchesInProgress" : 0,
    "numberBatchesQueued" : 0,
    "numberBatchesTotal" : 2,
    "numberRecordsFailed" : 0,
    "numberRecordsProcessed" : 2,
    "numberRetries" : 0,
    "object" : "Account",
    "operation" : "insert",
    "state" : "Aborted",
    "systemModstamp" : "2015-12-15T21:54:29.000+0000",
    "totalProcessingTime" : 2870
}
```

関連トピック:

- ジョブの詳細の取得
- ジョブの作成
- ジョブの監視
- ジョブの終了
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- JobInfo

ジョブとバッチの有効期限

ジョブやバッチは、作成後 7 日を超えると削除されます。

- ジョブやバッチが作成後 7 日を超えてから削除されるまでには、最大で 24 時間かかることがあります。
- 作成後 7 日を超えたジョブの中に、作成後 7 日を経過していないバッチが含まれている場合、そのジョブとそのジョブに関連付けられているすべてのバッチは、最も新しいバッチの有効期限までは削除されません。
- ジョブとバッチは状況とは無関係に削除されます。
- いったん削除されたジョブやバッチをプラットフォームから取得することはできません。

制限についての詳細は、「Bulk API の制限」 (ページ 90)を参照してください。

関連トピック:

- ジョブの作成
- ジョブの監視
- ジョブの詳細の取得
- ジョブの終了
- ジョブの中止
- ジョブへのバッチの追加
- Bulk API の制限
- URI について
- JobInfo
- クイックスタート

第 8 章　バッチでの作業

トピック:

バッチは、HTTP POST 要求でサーバに送信されるレコードセットです。各バッチはサーバによって独自に処理されます。受信した順序で処理されるとは限りません。

バッチを作成するには、レコードのセット、およびバイナリ添付ファイルへの参照を表す、CSV、XML、または JSON を HTTP POST 要求で送信します。作成したバッチの状況は BatchInfo リソースで表されます。バッチの処理が完了すると、各レコードの結果が結果セットのリソースとして提供されます。

バッチは並列処理が可能です。データセット全体をどのように適切な数のバッチに分割するかは、クライアント側で決定されます。

バッチサイズは処理時間に基づいて調整する必要があります。まず5,000件のレコードを含むバッチで処理を試行し、処理時間に応じてサイズを調整します。1つのバッチの処理に 5 分以上かかるようなら、バッチサイズを小さくしたほうがよいでしょう。数秒で完了するようであれば、バッチサイズを大きくします。バッチの処理時にタイムアウトエラーが発生する場合は、バッチをより小さなサイズに分割して再試行します。

ジョブへのバッチの追加

ジョブに新しいバッチを追加するには、次のような URI 宛てに POST 要求を送信します。リクエストボディには、処理するレコードのリストが含まれます。

URI

https://***instance_name***—api.salesforce.com/services/async/***APIversion***/job/***jobid***/batch

メモ: すべてのバッチ処理で使用する、URI 内で指定された API のバージョンは、関連付けられたジョブの API のバージョンと一致している必要があります。

XML リクエストボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 0 (batch 0)</name>
  </sObject>
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 1 (batch 0)</name>
  </sObject>
</sObjects>
```

このサンプルでは、関連付けられたジョブの contentType 項目が XML に設定されているため、バッチデータは XML 形式になっています。バッチデータの別の形式 (CSV や JSON など) については、「JobInfo」 (ページ 77)を参照してください。

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>751D0000000004rIAA</id>
 <jobId>750D0000000002lIAA</jobId>
 <state>Queued</state>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
 <numberRecordsProcessed>0</numberRecordsProcessed>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>0</totalProcessingTime>
 <apiActiveProcessingTime>0</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

JSON リクエストボディの例

```
[
   {
      "Name":"[Bulk API] Account 0 (batch 0)",
      "description" : "Created from Bulk API"
   },
   {
      "Name":"[Bulk API] Account 1 (batch 0)",
```

```
        "description" : "Created from Bulk API"
    }
]
```

このサンプルでは、関連付けられたジョブの `contentType` 項目が `JSON` に設定されているため、バッチデータは JSON 形式になっています。

JSON レスポンスボディの例

```
{
   "apexProcessingTime":0,
   "apiActiveProcessingTime":0,
   "createdDate":"2015-12-15T21:56:43.000+0000",
   "id":"751D00000004YGZIA2",
   "jobId":"750D00000004SkVIAU",
   "numberRecordsFailed":0,
   "numberRecordsProcessed":0,
   "state":"Queued",
   "systemModstamp":"2015-12-15T21:56:43.000+0000",
   "totalProcessingTime":0
}
```

メモ: 一括処理ジョブは、Bulk API に準拠していない CSV ファイルを使用して追加できます。「データ項目の対応付け」 (ページ 108)を参照してください。

関連トピック:

- バイナリ添付ファイルを含むバッチの作成
- バッチに関する情報の取得
- バッチの監視
- ジョブ内のすべてのバッチに関する情報の取得
- バッチの状態の解釈
- バッチ要求の取得
- バッチ結果の取得
- ジョブでの作業
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- BatchInfo
- クイックスタート

バッチの監視

Salesforce では、Bulk API のバッチを監視できます。

一括データ読み込みジョブとそのジョブに関連付けられたバッチ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「*一括データ読み込みジョブ*」と入力し、[一括データ読み込みジョブ]を選択します。[ジョブ ID] をクリックすると、ジョブの詳細ページが表示されます。

ジョブの詳細ページには、ジョブの全バッチの関連リストも表示されます。関連リストには、各バッチについて [要求を表示] および [応答を参照] リンクが表示されます。バッチが CSV ファイルの場合、これらのリンクは CSV 形式で要求または応答を返します。バッチが XML または JSON ファイルの場合、これらのリンクはそれぞれ XML あるいは JSON 形式で要求または応答を返します。これらのリンクは、API バージョン 19.0 以降で作成されたバッチに使用できます。

詳細は、Salesforce オンラインヘルプの「一括データ読み込みジョブの監視」を参照してください。

関連トピック:

- バッチに関する情報の取得
- ジョブへのバッチの追加
- ジョブ内のすべてのバッチに関する情報の取得
- バッチの状態の解釈
- バッチ要求の取得
- バッチ結果の取得
- バッチの失敗したレコードの処理
- ジョブでの作業
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- BatchInfo
- クイックスタート

バッチに関する情報の取得

既存のバッチに関する情報を取得するには、次のような URI 宛てに GET 要求を送信します。

URI

https://***instance_name***–api.salesforce.com/services/async/***APIversion***/job/***jobid***/batch/***batchId***

リクエストボディの例

リクエストボディは記述しません。

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>751D0000000004rIAA</id>
 <jobId>750D0000000002lIAA</jobId>
 <state>InProgress</state>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
```

```
 <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
 <numberRecordsProcessed>0</numberRecordsProcessed>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>0</totalProcessingTime>
 <apiActiveProcessingTime>0</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

JSON レスポンスボディの例

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 0,
   "createdDate" : "2015-12-15T22:52:49.000+0000",
   "id" : "751D00000004YGeIAM",
   "jobId" : "750D00000004SkVIAU",
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 0,
   "state" : "InProgress",
   "systemModstamp" : "2015-12-15T22:52:49.000+0000",
   "totalProcessingTime" : 0
}
```

関連トピック:

- ジョブへのバッチの追加
- バッチの監視
- ジョブ内のすべてのバッチに関する情報の取得
- バッチの状態の解釈
- バッチ要求の取得
- バッチ結果の取得
- ジョブとバッチの有効期限
- Bulk API の制限
- BatchInfo
- URI について
- ジョブでの作業
- クイックスタート

ジョブ内のすべてのバッチに関する情報の取得

ジョブに含まれるすべてのバッチに関する情報を取得するには、次のようなURI宛てにGET要求を送信します。

URI

https://***instance_name***—api.salesforce.com/services/async/***APIversion***/job/***jobid***/batch

メソッド

GET

リクエストボディの例

リクエストボディは記述しません。

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfoList
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <batchInfo>
  <id>751D0000000004rIAA</id>
  <jobId>750D0000000002lIAA</jobId>
  <state>InProgress</state>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:16:09.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D0000000004sIAA</id>
  <jobId>750D0000000002lIAA</jobId>
  <state>InProgress</state>
  <createdDate>2009-04-14T18:16:00.000Z</createdDate>
  <systemModstamp>2009-04-14T18:16:09.000Z</systemModstamp>
  <numberRecordsProcessed>800</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>5870</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>2166</apexProcessingTime>
 </batchInfo>
</batchInfoList>
```

JSON レスポンスボディの例

```
{
   "batchInfo" : [
      {
         "apexProcessingTime" : 0,
         "apiActiveProcessingTime" : 0,
         "createdDate" : "2015-12-15T21:56:43.000+0000",
         "id" : "751D00000004YGZIA2",
         "jobId" : "750D00000004SkVIAU",
         "numberRecordsFailed" : 0,
         "numberRecordsProcessed" : 0,
         "state" : "Queued",
         "systemModstamp" : "2015-12-15T21:57:19.000+0000",
         "totalProcessingTime" : 0
      },
      {
         "apexProcessingTime" : 0,
         "apiActiveProcessingTime" : 2166,
         "createdDate" : "2015-12-15T22:52:49.000+0000",
         "id" : "751D00000004YGeIAM",
```

```
            "jobId" : "750D00000004SkVIAU",
            "numberRecordsFailed" : 0,
            "numberRecordsProcessed" : 800,
            "state" : "Completed",
            "systemModstamp" : "2015-12-15T22:54:54.000+0000",
            "totalProcessingTime" : 5870
        }
    ]
}
```

関連トピック:

- ジョブへのバッチの追加
- バッチの監視
- バッチに関する情報の取得
- バッチの状態の解釈
- バッチ要求の取得
- バッチ結果の取得
- ジョブとバッチの有効期限
- Bulk API の制限
- BatchInfo
- URI について
- ジョブでの作業
- クイックスタート

バッチの状態の解釈

ここでは、バッチのさまざまな状態(または状況)について解説します。バッチの状態に基づいて、結果を取得する、待機する、要求に関連したエラーを修正するなど、後続の作業を判断できます。

Queued
バッチ処理はまだ開始されていません。このバッチに関連付けられているジョブが中止された場合は、バッチは処理されず、状態は `Not Processed` に設定されます。

InProgress
バッチは処理中です。バッチに関連付けられているジョブが中止された場合でも、バッチは完了するまで処理されます。バッチで処理を完了できるようにするため、バッチに関連付けられているジョブを閉じる必要があります。

Completed
バッチ処理は完了済みで、結果のリソースを使用できます。一部のレコードの処理に失敗した場合は、結果のリソースに示されます。バッチは、一部のレコードまたはすべてのレコードの処理に失敗した場合でも完了します。レコードのサブセットの処理に失敗した場合、正常に処理されたレコードはロールバックされません。

Failed
要求がサポートされない形式に圧縮されている、内部のサーバエラーが発生したなどの予期しないエラーにより、このバッチは要求全体の処理に失敗しました。バッチの状態がFailedであっても、一部のレコードは正常に処理されている可能性があります。レスポンスボディの `numberRecordsProcessed` 項目に 0 以外の数値が表示されている場合は、結果を取得して、処理されたレコードと、その処理結果を確認する必要があります。

Not Processed
バッチはこれ以降処理されることはありません。この状態は、バッチがキューに入っている間にジョブが中止された場合に適用されます。一括クエリのジョブで PK Chunking が有効になっている場合、後続のバッチが作成されるときに、クエリを含む元のバッチにこの状態が割り当てられます。元のバッチがこの状態に変更された後に、後続のバッチを監視し、各バッチの完了時にその結果を取得できます。

関連トピック:

- ジョブへのバッチの追加
- バッチの監視
- ジョブ内のすべてのバッチに関する情報の取得
- バッチ要求の取得
- バッチ結果の取得
- バッチの失敗したレコードの処理
- ジョブとバッチの有効期限
- Bulk API の制限
- BatchInfo
- URI について
- ジョブでの作業
- クイックスタート

バッチ要求の取得

バッチ要求を取得するには、次のような URI 宛てに GET 要求を送信します。これは、API バージョン 19.0 以降で使用できます。

または、Salesforce でバッチ要求を取得することもできます。一括データ読み込みジョブとそのジョブに関連付けられたバッチ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*「一括データ読み込みジョブ」*と入力し、[一括データ読み込みジョブ] を選択します。[ジョブ ID] をクリックすると、ジョブの詳細ページが表示されます。ジョブの詳細ページには、ジョブの全バッチの関連リストも表示されます。関連リストには、各バッチについて [要求を表示] および [応答を参照] リンクが表示されます。バッチが CSV ファイルの場合、これらのリンクは CSV 形式で要求または応答を返します。バッチが XML または JSON ファイルの場合、これらのリンクはそれぞれ XML あるいは JSON 形式で要求または応答を返します。

URI

`https://`***`instance_name`***`-api.salesforce.com/services/async/`***`APIversion`***`/job/`***`jobid`***`/batch/`***`batchId`***`/request`

リクエストボディの例

リクエストボディは記述しません。

XML レスポンスボディの例

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 0 (batch 0)</name>
  </sObject>
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 1 (batch 0)</name>
  </sObject>
</sObjects>
```

JSON レスポンスボディの例

```
[
   {
      "Name" : "[Bulk API] Account 0 (batch 0)",
      "description" : "Created from Bulk API"
   },
   {
      "Name" : "[Bulk API] Account 1 (batch 0)",
      "description" : "Created from Bulk API"
   }
]
```

関連トピック:

- バッチに関する情報の取得
- バッチの監視
- ジョブ内のすべてのバッチに関する情報の取得
- バッチの状態の解釈
- バッチ結果の取得
- ジョブでの作業
- ジョブとバッチの有効期限
- Bulk API の制限
- URI について
- BatchInfo
- クイックスタート

バッチ結果の取得

処理が完了したバッチの結果を取得するには、次のようなURI宛てにGET要求を送信します。バッチがCSVファイルの場合、応答はCSV形式で返されます。バッチがXMLまたはJSONファイルの場合、応答はそれぞれXMLまたはJSON形式で返されます。

または、Salesforce で Bulk API のバッチを監視することもできます。一括データ読み込みジョブとそのジョブに関連付けられたバッチ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「*一括データ読み込みジョブ*」と入力し、[一括データ読み込みジョブ] を選択します。[ジョブ ID] をクリックすると、ジョブの詳細ページが表示されます。

ジョブの詳細ページには、ジョブの全バッチの関連リストも表示されます。関連リストには、各バッチについて [要求を表示] および [応答を参照] リンクが表示されます。バッチがCSVファイルの場合、これらのリンクはCSV形式で要求または応答を返します。バッチがXMLまたはJSONファイルの場合、これらのリンクはそれぞれXMLあるいはJSON形式で要求または応答を返します。これらのリンクは、APIバージョン19.0以降で作成されたバッチに使用できます。[応答を参照] リンクで返される結果は、次のURIリソースの結果と同じです。

URI

https://***instance_name***-api.salesforce.com/services/async/***APIversion***/job/***jobid***/batch/***batchId***/result

リクエストボディの例

リクエストボディは記述しません。

レスポンスボディの例

XML バッチの場合

```
<?xml version="1.0" encoding="UTF-8"?>
<results xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <result>
    <id>001D000000ISUr3IAH</id>
    <success>true</success>
    <created>true</created>
  </result>
  <result>
    <id>001D000000ISUr4IAH</id>
    <success>true</success>
    <created>true</created>
  </result>
</results>
```

JSON バッチの場合

```
[
   {
      "success" : true,
      "created" : true,
      "id" : "001xx000003DHP0AAO",
      "errors" : []
   },
   {
      "success" : true,
      "created" : true,
      "id" : "001xx000003DHP1AAO",
```

```
      "errors" : []
   }
]
```

CSV バッチの場合

```
"Id","Success","Created","Error"
"003D000000Q89kQIAR","true","true",""
"003D000000Q89kRIAR","true","true",""
"","false","false","REQUIRED_FIELD_MISSING:Required fields are missing:
[LastName]:LastName --"
```

メモ: このバッチ結果は、`LastName` 項目が欠落していたために、最後のレコードが正常に処理されなかったことを示しています。`Error` 列にはエラー情報が表示されます。各結果行の `Success` 項目をチェックして、すべての行が正常に処理されたことを確認する必要があります。詳細は、「バッチの失敗したレコードの処理」 (ページ 56)を参照してください。

関連トピック:

ジョブへのバッチの追加
バッチの監視
バッチ要求の取得
バッチに関する情報の取得
ジョブ内のすべてのバッチに関する情報の取得
バッチの状態の解釈
ジョブとバッチの有効期限
Bulk API の制限
BatchInfo
URI について
ジョブでの作業
クイックスタート

バッチの失敗したレコードの処理

バッチの状態は、一部またはすべてのレコードが失敗した場合でも、`Completed` になることがあります。レコードのサブセットの処理に失敗した場合、正常に処理されたレコードはロールバックされません。同様に、バッチの状態が `Failed` になっている場合や、ジョブが途中で中止された場合でも、一部のレコードは正常に処理されている可能性があります。

バッチ結果を取得したら、各結果行の `Success` 項目の値をチェックして、すべての行が正常に処理されたかどうかを確認する必要があります。レコードが正常に処理されなかった場合は、`Error` 列に失敗の詳細情報が表示されます。

失敗したレコードを特定し、エラーファイルにログを記録する手順は、次のとおりです。

1. バッチ処理が完了するのを待ちます。「バッチに関する情報の取得」 (ページ 49)および「バッチの状態の解釈」 (ページ 52)を参照してください。

2. バッチ結果を取得します。

   次のCSV形式のバッチ結果のサンプルは、`LastName` 項目が欠落していたために、最後のレコードでエラーが発生したことを示しています。

```
"Id","Success","Created","Error"
"003D000000Q89kQIAR","true","true",""
"003D000000Q89kRIAR","true","true",""
"","false","false","REQUIRED_FIELD_MISSING:Required fields are missing:
[LastName]:LastName --"
```

3. 各レコードの結果を解析します。

   a. 各結果レコードのレコード番号を追跡します。各結果レコードはバッチに含まれるレコードに対応してします。結果は、バッチ要求のレコードと同じ順序で返されます。結果のレコード番号を追跡することで、バッチ要求内で処理に失敗した関連レコードを特定できます。

   b. `Success` 項目の値が `false` の場合、行は正常に処理されていません。それ以外の場合は、レコードは正常に処理されているので、後続のレコードの結果を確認します。

   c. `Error` 列の内容を取得します。

   d. バッチ要求内の対応するレコードの内容をコンピュータ上のエラーファイルに書き込みます。`Error` 列の情報を追加します。送信したバッチ要求がキャッシュされていない場合は、Salesforceからバッチ要求を取得できます。

各レコードの結果を確認したら、エラーファイルにある各レコードを手動で修正し、修正済みのレコードを新しいバッチとして送信できます。その後、上記の手順を繰り返して、各レコードが正常に処理されたかどうかを確認します。

関連トピック:

- ジョブへのバッチの追加
- エラー
- Bulk API の制限

第 9 章　一括クエリ

トピック:

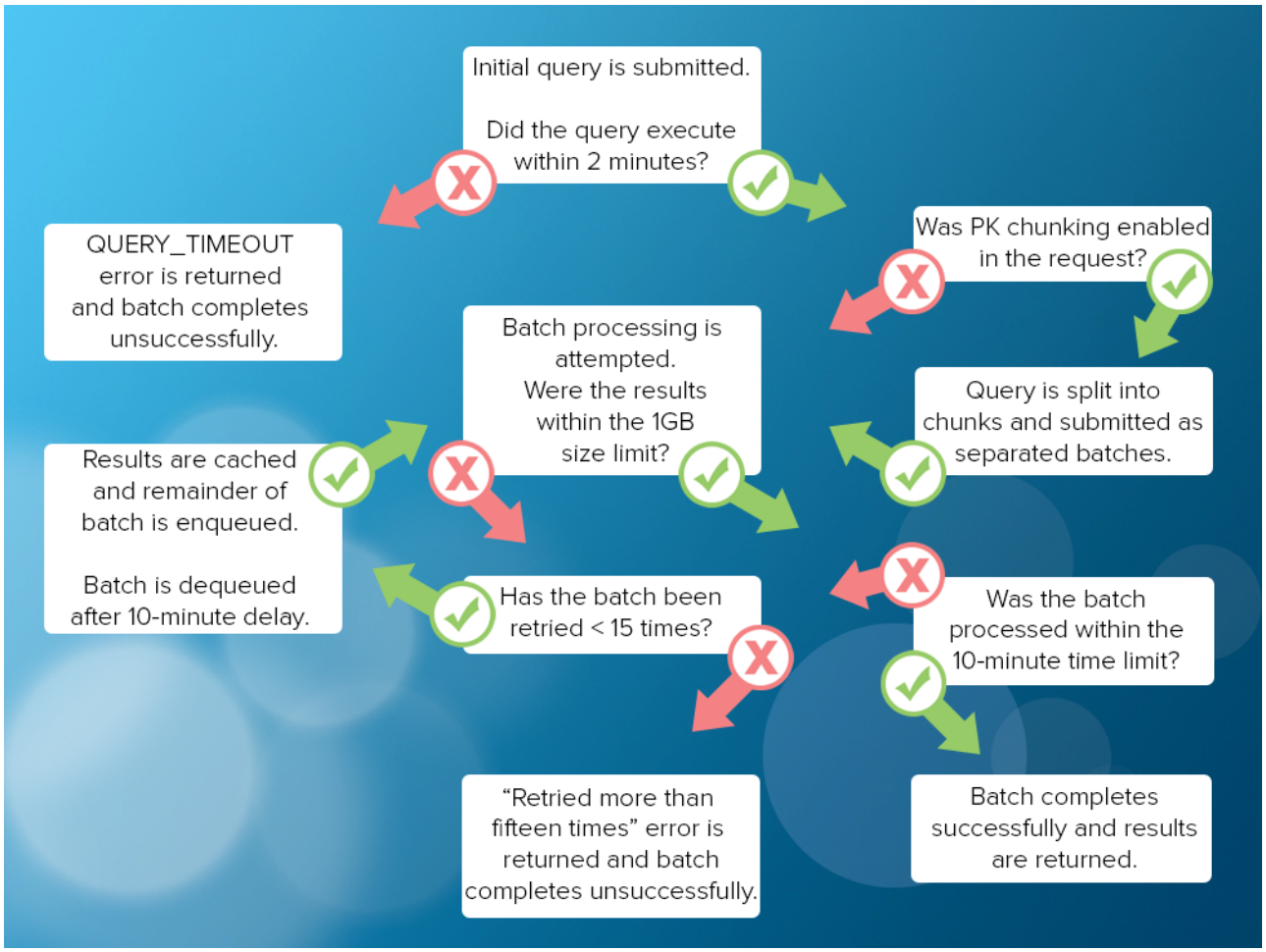
一括クエリを使用すると、大きなデータセットをより効率的にクエリし、API要求の数を減らすことができます。一括クエリでは、最大 15 GB のデータを 15 個の 1 GB のファイルに分割して取得できます。サポートされるデータ形式は、CSV、XML、および JSON です。

一括クエリの処理方法

一括クエリを処理すると、Salesforce によってクエリの実行が試行されます。標準のタイムアウト制限 (2 分) 内にクエリが実行されないと、ジョブは失敗し、QUERY_TIMEOUT エラーが返されます。この場合は、より単純なクエリに書き換えて、バッチを再送信してください。

クエリが正常に実行されると、Salesforce により結果の取得が試行されます。結果が 1 GB のファイルサイズ制限を超えているか、取得に 10 分以上かかる場合は、処理が完了した結果がキャッシュされ、別の試行が行われます。試行回数が 15 回を超えると、ジョブが失敗し、「Retried more than fifteen times (試行回数が 15 回を超えました)」というエラーメッセージが返されます。この場合は、PK Chunking ヘッダーを使用して、クエリ結果を小規模なチャンクに分割することを検討してください。試行に成功すると、結果が返され、7 日間保存されます。

次のフローチャートに、一括クエリの処理方法を示します。



関連トピック:

- 一括クエリの使用
- PK Chunking ヘッダー
- 一括クエリサンプルの説明

一括クエリの使用

一括クエリジョブにバッチを追加するときに、要求のヘッダーの Content-Type には、ジョブの作成時に指定されたコンテンツタイプに応じて、`text/csv`、`application/xml`、または `application/json` のいずれかを指定する必要があります。バッチに使用される実際の SOQL ステートメントは、平文テキスト形式で表されます。

URI

`https://`***`instance_name`***`—api.salesforce.com/services/async/`***`APIversion`***`/job/`***`jobid`***`/batch`

一括クエリの要求

```
POST baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
Content-Type: [either text/csv, application/xml, or application/json depending on job]

[plain text SOQL statement]
```

メモ: Bulk API クエリでは、次の SOQL はサポートされていません。

- COUNT
- ROLLUP
- SUM
- GROUP BY CUBE
- OFFSET
- ネストされた SOQL クエリ
- リレーション項目

また、Bulk API は、複合住所項目または複合地理位置情報項目へのアクセスやクエリができません。

例: 要求および応答

一括クエリの要求と応答の例を次に示します。

一括クエリバッチ作成の HTTP 要求

```
POST baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
Content-Type: text/csv; charset=UTF-8

SELECT Name, Description__c FROM Merchandise__c
```

一括クエリバッチ作成の HTTP レスポンスボディ

```
<?xmlversion="1.0" encoding="UTF-8"?>
<batchInfo
  xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <id>751x00000000079AAA</id>
    <jobId>750x00000000014</jobId>
    <state>Queued</state>
    <createdDate>2009-09-01T17:44:45.000Z</createdDate>
    <systemModstamp>2009-09-01T17:44:45.000Z</systemModstamp>
    <numberRecordsProcessed>0</numberRecordsProcessed>
    <numberRecordsFailed>0</numberRecordsFailed>
    <totalProcessingTime>0</totalProcessingTime>
    <apiActiveProcessingTime>0</apiActiveProcessingTime>
    <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

ジョブの HTTP 要求内のすべてのバッチに関する情報の取得 (PK Chunking が有効な場合に使用)

```
GET baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
```

ジョブの HTTP レスポンスボディ内のすべてのバッチに関する情報の取得

```
<?xml version="1.0" encoding="UTF-8"?><batchInfoList
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <batchInfo>
  <id>751D00000004YjwIAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>NotProcessed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk1IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk2IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
```

```
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk6IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk7IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>50000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>500</totalProcessingTime>
  <apiActiveProcessingTime>500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
</batchInfoList>
```

バッチ結果取得の HTTP 要求

```
GET baseURI/job/750x00000000014/batch/751x00000000030/result
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
```

バッチ結果取得の HTTP レスポンスボディ

```
<result-list
xmlns="http://www.force.com/2009/06/asyncapi/dataload"><result>752x000000000F1</result></result-list>
```

一括クエリ結果取得の HTTP 要求

```
GET baseURI/job/750x00000000014/batch/751x00000000030/result/752x000000000F1
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
```

一括クエリ結果取得の HTTP レスポンスボディ

```
"Name", "Description__c"
"Merchandise1", "Description for merchandise 1"
"Merchandise2", "Description for merchandise 2"
```

例: WSC を使用した Java の例

次の例では、Partner API を使用して組織にログインし、Partner API ログインからサービスエンドポイントを使用して `BulkConnection` オブジェクトをインスタンス化します。

```
public boolean login() {
  boolean success = false;

  String userId = getUserInput("UserID: ");
  String passwd = getUserInput("Password: ");
  String soapAuthEndPoint = "https://" + loginHost + soapService;
  String bulkAuthEndPoint = "https://" + loginHost + bulkService;
  try {
    ConnectorConfig config = new ConnectorConfig();
    config.setUsername(userId);
    config.setPassword(passwd);
    config.setAuthEndpoint(soapAuthEndPoint);
    config.setCompression(true);
    config.setTraceFile("traceLogs.txt");
    config.setTraceMessage(true);
    config.setPrettyPrintXml(true);
    System.out.println("AuthEndpoint: " +
        config.getRestEndpoint());
    PartnerConnection connection = new PartnerConnection(config);
    System.out.println("SessionID: " + config.getSessionId());
    config.setRestEndpoint(bulkAuthEndPoint);
    bulkConnection = new BulkConnection(config);
    success = true;
  } catch (AsyncApiException aae) {
    aae.printStackTrace();
  } catch (ConnectionException ce) {
    ce.printStackTrace();
  } catch (FileNotFoundException fnfe) {
    fnfe.printStackTrace();
  }
  return success;
}

public void doBulkQuery() {
  if ( ! login() ) {
    return;
  }
  try {
    JobInfo job = new JobInfo();
    job.setObject("Merchandise__c");

    job.setOperation(OperationEnum.query);
    job.setConcurrencyMode(ConcurrencyMode.Parallel);
    job.setContentType(ContentType.CSV);

    job = bulkConnection.createJob(job);
    assert job.getId() != null;

    job = bulkConnection.getJobStatus(job.getId());
```

```
      String query =
          "SELECT Name, Id, Description__c FROM Merchandise__c";

      long start = System.currentTimeMillis();

      BatchInfo info = null;
      ByteArrayInputStream bout =
          new ByteArrayInputStream(query.getBytes());
      info = bulkConnection.createBatchFromStream(job, bout);

      String[] queryResults = null;

      for(int i=0; i<10000; i++) {
        Thread.sleep(i==0 ? 30 * 1000 : 30 * 1000); //30 sec
        info = bulkConnection.getBatchInfo(job.getId(),
            info.getId());

        if (info.getState() == BatchStateEnum.Completed) {
          QueryResultList list =
              bulkConnection.getQueryResultList(job.getId(),
                  info.getId());
          queryResults = list.getResult();
          break;
        } else if (info.getState() == BatchStateEnum.Failed) {
          System.out.println("-------------- failed ----------"
              + info);
          break;
        } else {
          System.out.println("-------------- waiting ----------"
              + info);
        }
      }

      if (queryResults != null) {
        for (String resultId : queryResults) {
          bulkConnection.getQueryResultStream(job.getId(),
              info.getId(), resultId);
        }
      }
    } catch (AsyncApiException aae) {
      aae.printStackTrace();
    } catch (InterruptedException ie) {
      ie.printStackTrace();
    }
  }
```

関連トピック:

- 一括クエリの処理方法
- PK Chunking ヘッダー
- 一括クエリサンプルの説明

一括クエリサンプルの説明

このコードサンプルは、cURL を使用していくつかの取引先レコードをクエリします。

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。
- テストクライアントアプリケーションの作成を開始する前に、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も確認します。

ジョブの作成

1. 次のテキストを含む、`create-job.xml` というファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <operation>query</operation>
  <object>Account</object>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
</jobInfo>
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、新しいジョブを作成します。

```
curl -H "X-SFDC-Session: sessionId" -H "Content-Type: application/xml; charset=UTF-8"
-d @create-job.xml https://instance.salesforce.com/services/async/37.0/job
```

instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
```

```
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

ジョブへのバッチの追加

1. SOQL クエリステートメントを含む、`query.txt` というファイルを作成します。

```
SELECT Id, Name FROM Account LIMIT 10
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ジョブにバッチを追加します。

```
curl -d @query.txt -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv;
charset=UTF-8" https://instance.salesforce.com/services/async/37.0/job/jobId/batch
```

jobId は、ジョブ作成の応答で返されたジョブ ID です。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Queued</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T00:59:47.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

メモ: 一括クエリジョブにバッチを追加するときに、要求のヘッダーの Content-Type には、ジョブの作成時に指定されたコンテンツタイプに応じて、`text/csv`、`application/xml`、または `application/json` のいずれかを指定する必要があります。バッチに使用される実際の SOQL ステートメントは、平文テキスト形式で表されます。

ジョブとバッチの状況の確認

1. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ジョブの状況を確認します。

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/37.0/job/jobId
```

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
```

```
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>1</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>1</numberBatchesTotal>
  <numberRecordsProcessed>10</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、バッチの状況を確認します。

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId
   ```

   batchId は、バッチ作成の応答で返されたバッチ ID です。

   Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>10</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

結果の取得

1. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、バッチ結果のリストを取得します。

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId/result
```

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<result-list xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <result>752x00000004CJE</result>
</result-list>
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、クエリの結果を取得します。

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId/result/resultId
```

resultId は、バッチ結果リストの要求に対する応答で返された結果 ID です。

Salesforce により、次のようなデータを含む CSV 応答が返されます。

```
"Id","Name"
"001x000xxx4TU4JAAW","name161268--1296595660659"
"001x000xxx4TU4KAAW","name161269--1296595660659"
"001x000xxx4TU4LAAW","name161270--1296595660659"
"001x000xxx4TU4MAAW","name161271--1296595660659"
"001x000xxx4TU4NAAW","name161272--1296595660659"
"001x000xxx4TU4OAAW","name161273--1296595660659"
"001x000xxx4TU4PAAW","name161274--1296595660659"
"001x000xxx4TU4QAAW","name161275--1296595660659"
"001x000xxx4TU4RAAW","name161276--1296595660659"
"001x000xxx4TU4SAAW","name161277--1296595660659"
```

ジョブの終了

1. 次のテキストを含む、close-job.xml というファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <state>Closed</state>
</jobInfo>
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ジョブを終了します。

```
curl -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv; charset=UTF-8" -d
@close-job.xml https://instance.salesforce.com/services/async/37.0/job
```

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
```

```
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Closed</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>1</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>1</numberBatchesTotal>
  <numberRecordsProcessed>10</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

関連トピック:

- 一括クエリの処理方法
- 一括クエリの使用
- PK Chunking を使用した一括クエリサンプルの説明

PK Chunking を使用した一括クエリサンプルの説明

このコードサンプルは、cURLを使用して、いくつかの取引先レコードでPKChunkingが有効な一括クエリを実行します。

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。

- 製品ドキュメントに従って、開発プラットフォームをインストールする。
- テストクライアントアプリケーションの作成を開始する前に、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も確認します。

PK Chunking が有効なジョブの作成

1. 次のテキストを含む、`create-job.xml` というファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <operation>query</operation>
  <object>Account</object>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
</jobInfo>
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、PK Chunking が有効なジョブを作成します。

```
curl -H "X-SFDC-Session: sessionId" -H "Content-Type: application/xml; charset=UTF-8"
-H "Sforce-Enable-PKChunking: true" -d @create-job.xml
https://instance.salesforce.com/services/async/37.0/job
```

instance は、ログインの応答でメモした <serverUrl> 要素の一部分です。*sessionId* は同様にメモした <sessionId> 要素です。

> **メモ:** Salesforce は、レコード数が 1,000 万を超えるテーブルをクエリする場合や、一括クエリがいつもタイムアウトしてしまう場合に PK Chunking を有効化することをお勧めします。この例では、この推奨件数よりも大幅に少ないレコード数をクエリしている場合に、そのレコード数よりも小さい数を chunkSize に設定します。たとえば、「Sforce-Enable-PKChunking: chunkSize=1000」などです。こうすることで、PK Chunking の動作、およびクエリが複数のバッチに分割されることを確認できます。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

ジョブへのバッチの追加

1. SOQL クエリステートメントを含む、query.txt というファイルを作成します。

```
SELECT Id, Name FROM Account
```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ジョブにバッチを追加します。

```
curl -d @query.txt -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv;
charset=UTF-8" https://instance.salesforce.com/services/async/37.0/job/jobId/batch
```

jobId は、ジョブ作成の応答で返されたジョブ ID です。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Queued</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T00:59:47.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

メモ: 一括クエリジョブにバッチを追加するときに、要求のヘッダーの Content-Type には、ジョブの作成時に指定されたコンテンツタイプに応じて、`text/csv`、`application/xml`、または `application/json` のいずれかを指定する必要があります。バッチに使用される実際の SOQL ステートメントは、平文テキスト形式で表されます。

ジョブとバッチの状況の確認

1. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ジョブの状況を確認します。

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/37.0/job/jobId
   ```

   Salesforce により、次のようなデータを含む XML 応答が返されます。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <jobInfo
       xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <id>750x000000009tvAAA</id>
     <operation>query</operation>
     <object>Account</object>
     <createdById>005x0000001WR0lAAG</createdById>
     <createdDate>2016-01-10T00:53:19.000Z</createdDate>
     <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
     <state>Open</state>
     <concurrencyMode>Parallel</concurrencyMode>
     <contentType>CSV</contentType>
     <numberBatchesQueued>0</numberBatchesQueued>
     <numberBatchesInProgress>0</numberBatchesInProgress>
     <numberBatchesCompleted>4</numberBatchesCompleted>
     <numberBatchesFailed>0</numberBatchesFailed>
     <numberBatchesTotal>4</numberBatchesTotal>
     <numberRecordsProcessed>350000</numberRecordsProcessed>
   ```

```
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>3500</totalProcessingTime>
  <apiActiveProcessingTime>3500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

PK Chunking が有効化されているため、クエリ全体を処理するための追加バッチが自動的に作成されます。

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、元のバッチの状況を確認します。

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId
```

batchId は、バッチ作成の応答で返されたバッチ ID です。

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Not Processed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

PK Chunking が有効化されているため、元のバッチの状況は `Not Processed` になります。クエリは残りのバッチで処理されます。

残りのバッチの ID の取得

コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、残りのバッチを取得します。

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/37.0/job/jobId/batch
```

Salesforce により、次のようなデータを含む XML 応答が返されます。

```
<?xml version="1.0" encoding="UTF-8"?><batchInfoList
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <batchInfo>
  <id>751D00000004YjwIAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>NotProcessed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
```

```
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk1IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk2IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk6IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk7IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>50000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>500</totalProcessingTime>
  <apiActiveProcessingTime>500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
</batchInfoList>
```

結果の取得

残りの各バッチで次の手順を実行します。

1. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、バッチ結果のリストを取得します。

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId/result
   ```

   Salesforce により、次のようなデータを含む XML 応答が返されます。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <result-list xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <result>752x00000004CJE</result>
   </result-list>
   ```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、クエリの結果を取得します。

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/37.0/job/jobId/batch/batchId/result/resultId
   ```

   resultId は、バッチ結果リストの要求に対する応答で返された結果 ID です。

   Salesforce により、次のようなデータを含む CSV 応答が返されます。

   ```
   "Id","Name"
   "001x000xxx4TU4JAAW","name161268--1296595660659"
   "001x000xxx4TU4KAAW","name161269--1296595660659"
   "001x000xxx4TU4LAAW","name161270--1296595660659"
   "001x000xxx4TU4MAAW","name161271--1296595660659"
   "001x000xxx4TU4NAAW","name161272--1296595660659"
   "001x000xxx4TU4OAAW","name161273--1296595660659"
   "001x000xxx4TU4PAAW","name161274--1296595660659"
   "001x000xxx4TU4QAAW","name161275--1296595660659"
   "001x000xxx4TU4RAAW","name161276--1296595660659"
   "001x000xxx4TU4SAAW","name161277--1296595660659"
   ...
   ```

ジョブの終了

1. 次のテキストを含む、close-job.xml というファイルを作成します。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <state>Closed</state>
   </jobInfo>
   ```

2. コマンドラインウィンドウを使用して、次の cURL コマンドを実行し、ジョブを終了します。

   ```
   curl -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv; charset=UTF-8" -d
   @close-job.xml https://instance.salesforce.com/services/async/37.0/job
   ```

   Salesforce により、次のようなデータを含む XML 応答が返されます。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <jobInfo
   ```

```
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Closed</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>4</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>4</numberBatchesTotal>
  <numberRecordsProcessed>350000</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>3500</totalProcessingTime>
  <apiActiveProcessingTime>3500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

関連トピック:

- 一括クエリの処理方法
- 一括クエリの使用
- PK Chunking ヘッダー
- 一括クエリサンプルの説明

第 10 章　リファレンス

トピック:

Bulk API でサポートされているリソース、エラーの詳細、処理に関する制限事項について解説します。

Schema

Bulk API サービスは XML スキーマドキュメント (XML Schema Document: XSD) ファイルで記述されています。

次のような URI を使用して、利用中のバージョンの API に対応するスキーマファイルをダウンロードできます。

Web_Services_SOAP_endpoint_instance_name/services/async/*APIversion*/AsyncApi.xsd

たとえば、インスタンスでバージョン 37.0 の Bulk API を使用している場合、実際の URI は次のようになります。

https://***yourInstance***.salesforce.com/services/async/37.0/AsyncApi.xsd

組織のインスタンス名は、LoginResult の `serverUrl` 項目で返されます。

スキーマと API のバージョン

スキーマファイルは、最新リリースより前のバージョンの API に対して提供されています。現在、バージョン 18.0 以降の API バージョンに対応するファイルがダウンロード可能です。たとえば、API のバージョン 18.0 に対するスキーマファイルをダウンロードする場合は、次の URI を使用します。

https://***yourInstance***.salesforce.com/services/async/18.0/AsyncApi.xsd

関連トピック:

- JobInfo
- BatchInfo
- エラー

JobInfo

ジョブには、Salesforce に送信して処理するデータのバッチが 1 つ以上含まれます。ジョブを作成すると、Salesforce によってジョブの状態が自動的に `Open` に設定されます。

以下のリストに示す JobInfo のリソースを使用すると、ジョブの新規作成、ジョブに関する情報の取得、ジョブの終了、ジョブの中止を実行できます。

項目

名前	型	要件	説明
`apiVersion`	string	参照のみ。新しいジョブでの指定は不可	ジョブが作成された時点で URI に設定されるジョブの API のバージョン。現在は、17.0 以降のバージョンがサポートされています。
`apexProcessingTime`	Long	新しいジョブでの指定は不可	ジョブデータに関連するトリガおよび他のプロセスの処理時間(ミリ秒)。ジョブ内のすべてのバッチの処理時間の合計に相当します。非同期およびバッチの Apex 操作の処理時間は含まれません。トリガがない場合、

名前	型	要件	説明
			この値は 0 になります。「apiActiveProcessingTime」および「totalProcessingTime」も参照してください。 この項目は API バージョン 19.0 以降で使用できます。
apiActiveProcessingTime	Long	新しいジョブでの指定は不可	有効なジョブの処理時間 (ミリ秒)。apexProcessingTime の時間を含みますが、ジョブがキューで処理を待機している時間や、逐次化および並列化に要する時間は含まれません。ジョブ内のすべてのバッチの処理時間の合計に相当します。「apexProcessingTime」および「totalProcessingTime」も参照してください。 この項目は API バージョン 19.0 以降で使用できます。
assignmentRuleId	string	作成後は変更不可	ケースまたはリード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。割り当てルールの ID を取得するには、SOAP ベースの SOAP API を使用して AssignmentRule オブジェクトをクエリします。
concurrencyMode	ConcurrencyModeEnum		ジョブの同時実行モード。有効な値は、次のとおりです。 • Parallel: バッチを並列モードで処理します。これはデフォルト値です。 • Serial: バッチを逐次モードで処理します。並列処理を行うと、データベースの競合が生じる可能性があります。競合が激しいと、読み込みが失敗することがあります。このような場合は、ジョブを serial 同時実行モードで送信します。これにより、バッチを 1 つずつ確実に処理できます。ただし、このオプションを使用すると、ジョブの処理時間が大幅に増えることがあります。

名前	型	要件	説明
contentType	ContentType		ジョブのコンテンツタイプ。有効な値は、次のとおりです。 • CSV — CSV 形式のデータ • JSON — JSON 形式のデータ • XML — XML 形式のデータ (デフォルトのオプション) • ZIP_CSV — バイナリ添付ファイルを含む zip ファイルに入っている CSV 形式のデータ • ZIP_JSON — バイナリ添付ファイルを含む zip ファイルに入っている JSON 形式のデータ • ZIP_XML — バイナリ添付ファイルを含む zip ファイルに入っている XML 形式のデータ
createdById	string	システム項目	ジョブを作成したユーザのID。すべてのバッチは同じユーザによって作成される必要があります。
createdDate	dateTime	システム項目	ジョブが作成された日時をUTCのタイムゾーンで示します。
externalIdFieldName	string	更新/挿入では必須	upsert() に使用される外部 ID 項目の名前です。
id	string	新しいジョブでの指定は不可	このジョブの一意の ID。 この値は、すべてのGET処理の結果で返されます。
numberBatchesCompleted	int	新しいジョブでの指定は不可	このジョブの完了したバッチ数。
numberBatchesQueued	int	新しいジョブでの指定は不可	このジョブのキューにあるバッチ数。
numberBatchesFailed	int	新しいジョブでの指定は不可	このジョブの失敗したバッチ数。

名前	型	要件	説明
numberBatchesInProgress	int	新しいジョブでの指定は不可	このジョブの処理中のバッチ数。
numberBatchesTotal	int	新しいジョブでの指定は不可	ジョブに現在含まれているバッチの数の合計。この値は、ジョブにバッチが追加されるに従って増加します。ジョブの state が Closed または Failed になっている場合は、取得された値が最終的な合計値になります。 numberBatchesCompleted と numberBatchesFailed の合計が numberBatchesTotal の値と等しくなったら、ジョブは完了です。
numberRecordsFailed	int	新しいジョブでの指定は不可	このジョブで正常に処理されなかったレコードの数。 この項目は API バージョン 19.0 以降で使用できます。
numberRecordsProcessed	int	新しいジョブでの指定は不可	すでに処理済みのレコード数。この数値は、処理されたバッチ数が増えると大きくなります。
numberRetries	int		Salesforce が処理結果の保存を試行した回数。再試行の繰り返しはロックの競合などの問題によって発生します。
object	string	必須項目	処理中のデータのオブジェクト種別。ジョブに含まれるすべてのデータは、1 種類のオブジェクト種別でなければなりません。
operation	OperationEnum	必須項目	ジョブに含まれるバッチすべてに対する処理操作。有効な値は、次のとおりです。 • delete • insert • query • upsert • update • hardDelete 警告: この operation 項目の値はすべて小文字で入力してください。たとえ

名前	型	要件	説明
			ば、insert の代わりに INSERT と入力すると、エラーが発生します。 参照の整合性を確保するため、delete操作ではカスケード処理がサポートされています。親レコードを削除すると、各子レコードが削除可能な場合は自動的に削除されます。たとえば、Case レコードを削除すると、Bulk API はそのケースに関連付けられた CaseComment、CaseHistory、CaseSolution などの子レコードも自動的に削除します。ただし、CaseComment が削除可能でない場合、または現在使用中の場合、親 Case レコードの delete 操作は失敗します。 警告: hardDelete 値を指定した場合、削除されたレコードはごみ箱に保存されません。代わりに、即座に削除の対象となります。この操作の権限である「Bulk API の物理削除」は、デフォルトでは無効になっており、システム管理者が有効にする必要があります。物理削除を行うには、Salesforce ユーザライセンスが必要です。
state	JobStateEnum	ジョブの作成、終了、中止の各操作で必須	ジョブの現在の処理状況を次のような値によって示します。 • Open: ジョブが作成され、このジョブにバッチを追加できます。 • Closed: このジョブには新しいバッチを追加できません。ジョブに関連付けられているバッチは、ジョブが完了した後に処理できます。完了したジョブを編集または保存することはできません。 • Aborted: ジョブが中止されました。ジョブを中止するには、自分がそのジョブの作成者であるか、「データインテグレーションの管理」権限が有効になっている必要があります。 • Failed: ジョブが失敗しました。成功したレコードの処理が元に戻ることはありません。ジョブに含まれるすべてのバッチのリストはバッチ情報リスト

名前	型	要件	説明
			(BatchInfoList) に出力されます。BatchInfoList の結果から完了済みのバッチの結果を取得できます。この結果でどのレコードが処理されたかを確認できます正常に処理されなかったレコードの数は numberRecordsFailed 項目に出力されます。
systemModstamp	dateTime	システム項目	ジョブが完了した日時を UTC のタイムゾーンで示します。
totalProcessingTime	Long	新しいジョブでの指定は不可	ジョブの処理時間(ミリ秒)。ジョブ内のすべてのバッチの処理時間の合計に相当します。「apexProcessingTime」および「apiActiveProcessingTime」も参照してください。 この項目は API バージョン 19.0 以降で使用できます。

関連トピック:

- ジョブでの作業
- クイックスタート

BatchInfo

BatchInfo には、Salesforce に送信して処理する 1 件のバッチの情報が格納されます。

BatchInfo

名前	型	要件	説明
apexProcessingTime	Long	システム項目	バッチデータに関連するトリガおよび他のプロセスの処理時間 (ミリ秒)。トリガがない場合、この値は 0 になります。非同期およびバッチの Apex 操作の処理時間は含まれません。「apiActiveProcessingTime」および「totalProcessingTime」も参照してください。 この項目は API バージョン 19.0 以降で使用できます。
apiActiveProcessingTime	Long	システム項目	有効なバッチの処理時間 (ミリ秒)。apexProcessingTime の時間を含みます。バッチがキューで処理を待機している時間や、逐次化および並

名前	型	要件	説明
			列化に要する時間は含まれません。「totalProcessingTime」も参照してください。 この項目は API バージョン 19.0 以降で使用できます。
createdDate	dateTime	システム項目	UTC タイムゾーンでのバッチの作成日時。これは処理が開始した時間ではなく、バッチがジョブに追加された時間です。
id	string	必須項目	バッチの ID。グローバルにユニークの場合がありますが、必ずしもそうである必要はありません。
jobId	string	必須項目	バッチに関連付けられたジョブを一意に識別する 18 文字から成る ID。
numberRecordsFailed	int	システム項目	このバッチで正常に処理されなかったレコードの数。 この項目は API バージョン 19.0 以降で使用できます。
numberRecordsProcessed	int	システム項目	要求が送信された時点で、このバッチで処理済みのレコード数。この数値は、処理されたバッチ数が増えると大きくなります。
state	BatchStateEnum	システム項目	バッチの現在の処理状態。 • Queued: バッチ処理はまだ開始されていません。このバッチに関連付けられているジョブが中止された場合は、バッチは処理されず、状態は Not Processed に設定されます。 • InProgress: バッチは処理中です。バッチに関連付けられているジョブが中止された場合でも、バッチは完了するまで処理されます。バッチで処理を完了できるようにするため、バッチに関連付けられているジョブを閉じる必要があります。 • Completed: バッチ処理は完了済みで、結果のリソースが使用可能です。一部のレコードの処理に失敗した場合は、結果のリソースに示されます。バッチは、一部のレコードまたはすべてのレコードの処理に失敗した場合でも完了します。レコードのサブセットの処理に失敗した場合、正常に処理されたレコードはロールバックされません。 • Failed: 要求がサポートされない形式に圧縮されている、内部のサーバエラーが発生したなどの予期しないエラーにより、このバッチは要求全体の処理に失敗しました。エラーの詳細は、stateMessage 要素に出力されます。バッチの状態が Failed であって

名前	型	要件	説明
			も、一部のレコードは正常に処理されている可能性があります。処理されたレコードの数は `numberRecordsProcessed` 項目に出力されます。正常に処理されなかったレコードの数は `numberRecordsFailed` 項目に出力されます。 • `Not Processed`: バッチはこれ以降処理されることはありません。この状態は、バッチがキューに入っている間にジョブが中止された場合に適用されます。一括クエリのジョブで PK Chunking が有効になっている場合、後続のバッチが作成されるときに、クエリを含む元のバッチにこの状態が割り当てられます。元のバッチがこの状態に変更された後に、後続のバッチを監視し、各バッチの完了時にその結果を取得できます。その後安全にジョブを終了できます。
`stateMessage`	string	システム項目	状態 (state) に関する詳細が出力されます。たとえば、`state` の値が `Failed` の場合にはエラーの原因が出力されます。なお、複数のエラーが発生した場合には、最大長を超えたメッセージが切り捨てられることがあります。そのような場合は、確認できたエラーを修正してからバッチを再送信してください。バッチの状態が Failed であっても、一部のレコードは正常に処理されている可能性があります。
`systemModstamp`	dateTime	システム項目	UTC タイムゾーンでの処理の終了日時。状態が `Completed` の場合のみ有効です。
`totalProcessingTime`	Long	システム項目	バッチの処理時間(ミリ秒)。バッチがキューで処理を待機している時間は除外されます。 「`apexProcessingTime`」および「`apiActiveProcessingTime`」も参照してください。 この項目は API バージョン 19.0 以降で使用できます。

HTTP の BatchInfoList

名前	型	説明
`batchInfo`	BatchInfo	ジョブに関連付けられた各バッチに関する BatchInfo のリソースの要素。BatchInfo の構成については、「BatchInfo」 (ページ 82)を参照してください。

関連トピック:

- バッチでの作業
- バッチの状態の解釈
- クイックスタート

ヘッダー

Bulk API に使用されるカスタム HTTP の要求ヘッダーと応答ヘッダーを次に示します。

- Content Type ヘッダー
- Batch Retry ヘッダー
- Line Ending ヘッダー
- PK Chunking ヘッダー

Content Type ヘッダー

ContentTypeヘッダーを使用して、要求と応答の形式を指定します。作業するジョブの `contentType` に合わせてこのヘッダーの値を設定します。

`contentType` が CSV のジョブの場合、CSV で返される一括クエリ結果以外は XML が応答形式として使用されます。確実に JSON で応答を取得するには、`contentType` が JSON のジョブを作成し、バッチペイロードで JSON を使用します。確実に XML で応答を取得するには、`contentType` が XML または CSV のジョブを作成し、バッチペイロードで同じ形式を使用します。

ジョブの `contentType` が使用できない場合 (たとえば、ジョブの作成時や不正なジョブ ID での要求の送信時など)、応答にはこのヘッダーの値が適用されます。このヘッダーが含まれていない場合、応答はデフォルトで XML になります。

ヘッダーの項目名と値

項目名

`Content-Type`

項目値

- `application/json` (JSON 形式を推奨。)
- `application/xml` (XML 形式を推奨。)

- `text/csv` (CSV 形式を推奨。応答が XML で返される一括クエリ結果は除く。)

例

```
Content-Type: application/json
```

Batch Retry ヘッダー

一括ジョブを作成するとき、Batch Retry 要求ヘッダーを使用して、ジョブに含まれる未完了の一括処理の再試行を無効にできます。このヘッダーを使用して、常にタイムアウトする一括処理の一括処理時間を制限します。

ヘッダーの項目名と値

項目名

```
Sforce-Disable-Batch-Retry
```

項目値

- `TRUE`。このジョブに含まれる未完了の一括処理は再試行されません。
- `FALSE`。このジョブに含まれる未完了の一括処理は標準回数再試行されます (一括クエリは 15 回、一括アップロードは 10 回)。要求でヘッダーが指定されていない場合は、これがデフォルト値になります。

例

```
Sforce-Disable-Batch-Retry: TRUE
```

Line Ending ヘッダー

一括アップロードジョブを作成するとき、Line Ending 要求ヘッダーを使用すると、データ型が［テキストエリア］および［ロングテキストエリア］の項目について、行末を改行 (LF) または行頭復帰/改行 (CRLF) のどちらとして読み取るかを指定できます。

ヘッダーの項目名と値

項目名

```
Sforce-Line-Ending
```

項目値

- `LF`。行末を LF として読み取ります。
- `CRLF`。行末を CRLF として読み取ります。

例

```
Sforce-Line-Ending: CRLF
```

PK Chunking ヘッダー

PK Chunking 要求ヘッダーを使用すると、一括クエリジョブで自動的に主キー (PK) チャンクを作成できます。PK Chunking では、非常に大きなテーブルに対する一括クエリを、クエリされるレコードのレコード ID (主キー) に基づいたチャンクに分割します。各チャンクは、別個のバッチとして処理されて 1 日あたりのバッチ制限にカウントされます。また、各バッチの結果は別個にダウンロードする必要があります。PK Chunking は、Account、Campaign、CampaignMember、Case、Contact、Lead、LoginHistory、Opportunity、Task、User の各オブジェクトとカスタムオブジェクトでサポートされます。

PK Chunking は、`WHERE` 句でレコード ID 境界をクエリに追加し、クエリ結果を小規模なチャンクの合計結果数に制限することで動作します。残りの結果は、連続した境界を含む追加クエリで取得されます。各チャンクの ID 境界内にあるレコード件数は、チャンクサイズと呼ばれます。最初のクエリでは、指定された開始 ID と、開始 ID にチャンクサイズを加えた値の間にあるレコードが取得されます。次のクエリではレコードの次のチャンクが取得されます。

たとえば、10,000,000 件のレコードを含む Account テーブルの次のクエリで PK Chunking を有効にするとします。

```
SELECT Name FROM Account
```

チャンクサイズを 250,000、開始レコード ID を `001300000000000` と想定した場合、クエリは次の 40 個に分割されます。各クエリは別個のバッチとして送信されます。

```
SELECT Name FROM Account WHERE Id >= 001300000000000 AND Id < 00130000000132G
SELECT Name FROM Account WHERE Id >= 00130000000132G AND Id < 00130000000264W
SELECT Name FROM Account WHERE Id >= 00130000000264W AND Id < 00130000000396m
...
SELECT Name FROM Account WHERE Id >= 00130000000euQ4 AND Id < 00130000000fxSK
```

各クエリは、base-62 の ID 境界で指定された 250,000 件のレコードのチャンクで実行されます。

PK Chunking は、テーブル全体からデータを抽出するように設計されていますが、絞り込み条件が設定されたクエリにも使用できます。各クエリの結果からレコードが絞り込まれる可能性があるため、各チャンクで返される結果の件数はチャンクサイズを下回る場合があります。また、クエリがチャンクに分割されるときに、論理削除されたレコードの ID はカウントされますが、そのレコードは結果から除外されます。したがって、論理削除されたレコードが特定のチャンクの ID 境界内にある場合、返される結果の件数はチャンクサイズを下回ります。

デフォルトのチャンクサイズは 100,000 で、最大サイズは 250,000 です。デフォルトの開始 ID は、テーブルの最初のレコードです。ただし、異なる開始 ID を指定して、チャンクに分割されたバッチ間で失敗したジョブを再開することができます。

クエリがチャンクに正常に分割されると、元のバッチの状況が `NOT_PROCESSED` になります。チャンク分割に失敗すると、元のバッチの状況は `FAILED` になりますが、チャンク分割時に正常に分割されてキューに入れられたバッチは通常どおり処理されます。元のバッチの状況が `NOT_PROCESSED` に変更された場合は、後続のバッチを監視します。各後続のバッチの完了後に、そのバッチの結果を取得できます。その後安全にジョブを終了できます。

Salesforce は、レコード数が 1,000 万を超えるテーブルをクエリする場合や、一括クエリがいつもタイムアウトしてしまう場合に PK Chunking を有効化することをお勧めします。ただし、PK Chunking の有効性は、クエリおよびクエリされるデータの詳細に応じて異なります。

ヘッダーの項目名と値

項目名

`Sforce-Enable-PKChunking`

項目値

- `TRUE` — キューに入れられたテーブルの最初のレコード ID で開始し、デフォルトのチャンクサイズで PK Chunking を有効にします。
- `FALSE` — PK Chunking を無効にします。要求にヘッダーが指定されていない場合、デフォルト値は `FALSE` です。
- `chunkSize` — 各チャンクの ID 境界内にあるレコード件数を指定します。デフォルトは 100,000 で、最大サイズは 250,000 です。クエリに絞り込み条件が設定された、あるいは論理削除されたレコードが含まれている場合、各チャンクで返される結果の件数はチャンクサイズを下回る場合があります。
- `parent` — 共有オブジェクトのクエリで PK Chunking を有効にする場合に、親オブジェクトを指定します。チャンクは、共有オブジェクトのレコードではなく、親オブジェクトのレコードを基にしています。たとえば、AccountShare をクエリする場合は、Account を親オブジェクトとして指定します。PK Chunking は、親オブジェクトがサポートされていれば、共有オブジェクトでもサポートされます。
- `startRow` — 最初のチャンクの境界の下限として使用する 15 文字または 18 文字のレコード ID を指定します。バッチ間で失敗したジョブを再開するときに、このパラメータを使用して開始 ID を指定します。

例

```
Sforce-Enable-PKChunking: chunkSize=50000; startRow=00130000000xEftMGH
```

HTTP 状況コード

Bulk API で処理を実行すると、HTTP 状況コードが返されます。次に、一般的な状況コードと、その原因と思われる Bulk API アクションを示します。

HTTP 200

処理が正常に完了しました。

HTTP 400

要求が不正であったため、処理が正常に完了しませんでした。

HTTP 405

GET、POST 以外の HTTP メソッドが URI に送信されました。

HTTP 415

サポートされていない圧縮形式が設定されている可能性があります。有効な圧縮形式の値は `gzip` のみです。圧縮は必須ではありませんが、できる限り行うことをお勧めします。

HTTP 500

ほとんどの場合、サーバエラーが発生しています。

エラー

Bulk API での処理の実行中、エラーコードが返されることがあります。次に、一般的なエラーコードと、その原因と思われる Bulk API アクションを示します。

ClientInputError
クライアント側での不明なエラーにより処理が失敗しました。

バイナリ添付ファイルの場合、要求のコンテンツは入力ストリームと添付ファイルの両方を含む形で提供する必要があります。

ExceededQuota
作成しようとしているジョブ (またはバッチ) の数が、24 時間以内に作成できる上限数を超過しました。

FeatureNotEnabled
組織で Bulk API が有効になっていません。

InvalidBatch
バッチの更新またはバッチのクエリで指定された ID が無効です。

このエラーコードは、バイナリ添付ファイルの処理において、zip ファイルの形式が不正である場合や、次の条件が該当する場合に返されます。

- `request.txt` ファイルが無効である。具体的には、見つからない、読み取れない、別のディレクトリに保存されている、無効なコンテンツを含んでいるなど。
- 特定のバイナリ添付ファイルの圧縮解除時のサイズが大きすぎる。
- zip ファイルのサイズが大きすぎる。
- 複数のバイナリ添付ファイルの圧縮解除時の合計サイズが大きすぎる。

メモ: 次の条件に該当する場合、`INVALID_FIELD` という StatusCode が返されます。

- バッチデータ内で参照しているバイナリファイルが見つからない、または別のディレクトリに保存されている。
- バッチデータ内でバイナリファイルを参照するパスの先頭に `#` が付けられていない。

バイナリ添付ファイルの制限についての詳細は、「バイナリ型のコンテンツ」 (ページ 91)を参照してください。

InvalidJob
ジョブのクエリや更新、バッチの作成、更新、クエリで指定されたジョブ ID が無効です。

このエラーは、バージョン 19.0 以前の API で zip コンテンツを含むジョブを作成しようとした場合にも返されます。

InvalidJobState
ジョブの更新処理で指定されたジョブの状況が無効です。

InvalidOperation
ジョブの URI で指定された処理が無効です。URI に含まれている「job」のスペルをチェックしてください。

InvalidSessionId
指定されたセッション ID が無効です。

InvalidUrl
指定された URI が無効です。

InvalidUser
Bulk API 要求を送信しているユーザが適切な権限を持っていないか、または、別のユーザが作成したジョブやバッチを指定しています。

InvalidXML
リクエストボディに含まれている XML が無効です。

Timeout
接続がタイムアウトになりました。このエラーは、Salesforce でのバッチの処理時間がタイムアウト値を超過した場合に発生します。タイムアウト制限についての詳細は、「バッチの処理時間」 (ページ 91)を参照してください。バッチの処理時にタイムアウトエラーが発生する場合は、バッチをより小さなサイズに分割して再試行します。

TooManyLockFailure
多数のロック競合が発生してバッチの処理に失敗しました。このエラーはバッチの処理中に返されます。問題を解決するには、バッチを分析してロック競合を回避する必要があります。「データ読み込みに関する一般的なガイドライン」 (ページ 14)を参照してください。

Unknown
原因不明の例外が発生しました。

Bulk API では、これまで取り上げてきたコードの他に、SOAP API と共通の状況コードと例外コードを使用します。れらのコードについての詳細は、 *『SOAP API 開発者ガイド』*の「ExceptionCode」を参照してください。

関連トピック:

HTTP 状況コード

バッチの失敗したレコードの処理

Bulk API の制限

Bulk API を使用する場合は、次の制限事項に注意してください。

API の使用制限
Bulk API には、標準の API の使用制限が適用されます。使用制限の計算では、各 HTTP 要求が 1 コールとカウントされます。

バッチのコンテンツ
各バッチには、単一オブジェクトのレコードから成る CSV ファイル、XML ファイル、または JSON ファイルが 1 つだけ含まれている必要があります。この制限が守られていない場合、バッチは処理されず、`stateMessage` が更新されます。オブジェクトレコードの形式を正しいものとするためには、Enterprise WSDL を使用します。

バッチの制限
24 時間内に送信可能なバッチ数は最大で 5,000 件に制限されています。作成後 24 時間を超えたジョブに新しいバッチを関連付けることはできません。

バッチの有効期限
作成後 7 日を超えたバッチやジョブは、ジョブの状況にかかわらずキューから削除されます。この有効期限は、ジョブに関連付けられた最も新しいバッチの作成日時によって判断されます。バッチが含まれていないジョブでは、ジョブの作成日時が基準になります。作成後 24 時間を超えたジョブに新しいバッチを関連付けることはできません。

バッチサイズ

- データ読み込みのバッチは 10 MB 以下の単一の CSV ファイル、XML ファイル、または JSON ファイルで構成できます。
- 1 つのバッチには、最大で 10,000 件のレコードを含めることができます。
- 1 つのバッチには、最大で 10,000,000 文字のデータを含めることができます。
- 1 つの項目には、最大で 32,000 文字を含めることができます。
- 1 つのレコードには、最大で 5,000 項目を含めることができます。
- 1 つのレコードに含まれる項目には、合計で最大 400,000 文字を含めることができます。
- バッチには何らかのコンテンツが必要です。バッチが空の場合はエラーが返されます。

バイナリ型のコンテンツの制限については、「バイナリ型のコンテンツ」(ページ 91)を参照してください。

バッチの処理時間

バッチはチャンクで処理されます。チャンクサイズは、API バージョンによって異なります。API バージョンが 20.0 以前の場合、チャンクサイズは 100 レコードです。API バージョンが 21.0 以降の場合、チャンクサイズは 200 レコードです。各チャンクの処理時間には、5 分という制限があります。また、バッチ全体の処理に 10 分以上かかる場合、Bulk API ではバッチの残りがキューに戻され、後で処理されます。その後の処理でも 10 分以内に完了できない場合、バッチがキューに戻され最大 10 回まで処理が試行されます。それでも完了できない場合は、バッチ処理が完全に失敗したとみなされます。

バッチの状態が Failed であっても、一部のレコードは正常に処理されている可能性があります。バッチ結果を取得して、個々のレコードの処理状況を確認する方法については、「バッチ結果の取得」(ページ 55)を参照してください。バッチの処理時にタイムアウトエラーが発生する場合は、バッチをより小さなサイズに分割して再試行します。

> メモ: 一括クエリの場合、バッチ処理制限とは別に、実際のクエリの処理時間に 2 分という制限が追加されます。「query の制限」を参照してください。

バイナリ型のコンテンツ

- ファイル名の最大長は 512 バイトです。
- zip ファイルの最大サイズは 10 MB です。
- コンテンツの最大合計サイズは、圧縮解除した状態で 20 MB です。
- 1 つの zip ファイルに含めることができるファイル数は最大で 1,000 ファイルです。ディレクトリはファイル数にはカウントされません。

圧縮

有効な圧縮形式の値は `gzip` のみです。圧縮は必須ではありませんが、できる限り行うことをお勧めします。なお、圧縮を行っても、「バッチサイズ」で説明した文字数の制限は変わりません。

ジョブの中止

ジョブの中止は、適切な権限があればどのユーザでも実行できます。ジョブの終了は、そのジョブを作成したユーザのみが実行できます。

ジョブの終了

ジョブの終了は、そのジョブを作成したユーザのみが実行できます。ジョブの中止は、適切な権限があればどのユーザでも実行できます。

ジョブのコンテンツ
それぞれのジョブでは、1 つの処理と 1 つのオブジェクトを指定できます。このジョブに関連付けられたバッチには、1つのオブジェクトのレコードが含まれます。ジョブでは、必要に応じて逐次処理モードを指定することもできますが、このモードは、以前に送信した非同期ジョブでロックによる競合が発生した場合に使用します。これは、Salesforce が推奨した場合にのみ使用してください。

ジョブの外部 ID
JobInfo では外部 ID 項目の値を編集することはできません。外部 ID 項目を指定した場合、実行される処理は必ず更新/挿入となります。作成や更新の処理で使用しようとするとエラーが生成されます。

ジョブの有効期限
作成後 7 日を超えたバッチやジョブは、ジョブの状況にかかわらずキューから削除されます。この有効期限は、ジョブに関連付けられた最も新しいバッチの作成日時によって判断されます。バッチが含まれていないジョブでは、ジョブの作成日時が基準になります。作成後 24 時間を超えたジョブに新しいバッチを関連付けることはできません。

ジョブの Open 状態の時間
ジョブの状態が Open であるのは、作成後最大 24 時間までです。Bulk API では、バッチを毎時間に 1 つずつ、長時間にわたって追加するようなクライアントはサポートされません。

ジョブ履歴に表示されるジョブの状況
ジョブの状況とバッチの結果セットは、ジョブの完了後 7 日間保存され、それを過ぎると完全に削除されます。

ジョブの状況の変更
ジョブの状況を変更して POST の本文を送信する場合、指定できるのは `status` 項目の値のみとなります。`operation` 項目や `entity` 項目の値を指定した場合はエラーが発生します。

ポータルユーザ
カスタマーポータル、セルフサービスポータル、パートナーポータルなどのポータルユーザは、「API の有効化」権限を割り当てられていたとしても Bulk API にはアクセスできません。

`query` の制限
Bulk API クエリには、次の制限があります。

機能	詳細
取得されるファイルサイズ	1 ギガバイト。
取得されるファイル数	15 個。クエリで返されるファイルが 15 個を超える場合は、クエリを絞り込んで、返されるデータ量を減らす必要があります。一括バッチサイズは、一括クエリには使用されません。
クエリの試行回数	10 分ごとに 15 回、バッチを処理します。クエリの処理時間には 2 分という制限もあります。クエリの試行回数が 15 回を超えると、「Tried more than ten times (試行回数が 15 回を超えました)」というエラーメッセージが返されます。クエリの処理時間が 2 分を超えると、QUERY_TIMEOUT エラーが返されます。

機能	詳細
結果が保持される期間	7 日間。

Bulk API クエリでは、次の SOQL はサポートされていません。

- COUNT
- ROLLUP
- SUM
- GROUP BY CUBE
- OFFSET
- ネストされた SOQL クエリ
- リレーション項目

付録 A　Java を使用したサンプルクライアントアプリケーション

このコードサンプルを使用して、REST ベースの Bulk API で大量の取引先レコードを挿入するテストクライアントアプリケーションを作成します。

ステップバイステップの手順に加え、簡単にコピーして利用できる完全なコードをこのセクションの最後に記載します。

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。

- 製品ドキュメントに従って、開発プラットフォームをインストールする。
- テストクライアントアプリケーションの作成を開始する前に、すべての手順に目を通す。このガイドの他の部分を確認しておくと、用語や概念を把握できます。

1. クライアントアプリケーションの設定

   Bulk API は、HTTP GET メソッドおよび HTTP POST メソッドを使用して XML または JSON コンテンツを送信および取得するため、任意の言語で簡単にクライアントアプリケーションを構築できます。このタスクでは、Java サンプルと、開発作業の簡便化のために Salesforce が提供している Salesforce Web Service Connector (WSC) ツールキットを使用します。WSC は、ストリーミングパーサーを使用して実装された、高パフォーマンスの Web サービスクライアントスタックです。このツールキットには、Bulk API で使用される基本的な処理とオブジェクトに対するサポートが組み込まれています。

2. サンプルコードの説明

   クライアントの設定が完了したら、Bulk API を使用するクライアントアプリケーションを構築できます。次のサンプルを使用して、クライアントアプリケーションを作成します。各セクションで、コードの特定部分を順に説明していきます。詳細なサンプルは最後に記載します。

クライアントアプリケーションの設定

Bulk API は、HTTP GET メソッドおよび HTTP POST メソッドを使用して XML または JSON コンテンツを送信および取得するため、任意の言語で簡単にクライアントアプリケーションを構築できます。このタスクでは、Java サンプルと、開発作業の簡便化のために Salesforce が提供している Salesforce Web Service Connector (WSC) ツールキットを使用します。WSC は、ストリーミングパーサーを使用して実装された、高パフォーマンスの Web サービスクライアントスタックです。このツールキットには、Bulk API で使用される基本的な処理とオブジェクトに対するサポートが組み込まれています。

WSC のライブラリは以下のページから確認できます。

https://github.com/forcedotcom/wsc

Salesforce WSC ツールキットをダウンロードする手順は、次のとおりです。

1. http://mvnrepository.com/artifact/com.force.api/force-wsc にアクセスします。
2. 使用している API バージョンと一致する [使用可能なバージョン] リンクをクリックします。
3. **[ダウンロード (JAR)]** をクリックし、ローカルディレクトリにファイルを保存します。

Bulk API はログイン処理をサポートしていないため、ログインは SOAP API を使用して実行する必要があります。

Partner WSDL をダウンロードし、WSC ツールキットで Java クライアントにコンパイルする手順は、次のとおりです。

1. Developer Edition Salesforce のアカウントにログインします。管理者または「すべてのデータの編集」権限を持つユーザ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。詳細は、Salesforce オンラインヘルプの「ユーザが Salesforce にログインできる範囲と時間帯の制限」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]** を選択します。
3. **[Partner WSDL]** を右クリックしてブラウザの保存オプションを表示し、Partner WSDL をローカルディレクトリに保存します。Partner WSDL については、「Partner WSDL の使用」を参照してください。
4. WSC のコンパイルツールを使用して、WSDL から Partner API のコードをコンパイルします。

```
java -classpath pathToJar\wsc.jar com.sforce.ws.tools.wsdlc pathToWSDL\wsdlFilename
.\wsdlGenFiles.jar
```

たとえば、`wsc.jar` が `C:\salesforce\wsc` にインストールされていて、Partner WSDL が `C:\salesforce\wsdl\partner` に保存されている場合は、次のようになります。

```
java -classpath C:\salesforce\wsc\wsc.jar com.sforce.ws.tools.wsdlc
C:\salesforce\wsdl\partner\partner.wsdl .\partner.jar
```

後続のセクションにあるコード例のクラスパスで必要なライブラリは、`wsc.jar` と、ここで生成された `partner.jar` のみです。

サンプルコードの説明

クライアントの設定が完了したら、Bulk API を使用するクライアントアプリケーションを構築できます。次のサンプルを使用して、クライアントアプリケーションを作成します。各セクションで、コードの特定部分を順に説明していきます。詳細なサンプルは最後に記載します。

次のコードは、WSC ツールキット内のパッケージとクラスを設定し、さらに Partner WSDL から生成されたコードを設定します。

```
import java.io.*;
import java.util.*;

import com.sforce.async.*;
import com.sforce.soap.partner.PartnerConnection;
```

```
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;
```

main() メソッドの設定

次のコードは、クラスの `main()` メソッドを設定します。main() メソッドは、サンプルの処理ロジックを含む `runSample()` メソッドを呼び出します。`runSample()` で呼び出されるメソッドについては、後続のセクションで取り上げます。

```
    public static void main(String[] args)
      throws AsyncApiException, ConnectionException, IOException {
        BulkExample example = new BulkExample();
        // Replace arguments below with your credentials and test file name
        // The first parameter indicates that we are loading Account records
       example.runSample("Account", "myUser@myOrg.com", "myPassword", "mySampleData.csv");

    }

    /**
     * Creates a Bulk API job and uploads batches for a CSV file.
     */
    public void runSample(String sobjectType, String userName,
              String password, String sampleFileName)
            throws AsyncApiException, ConnectionException, IOException {
        BulkConnection connection = getBulkConnection(userName, password);
        JobInfo job = createJob(sobjectType, connection);
        List<BatchInfo> batchInfoList = createBatchesFromCSVFile(connection, job,
            sampleFileName);
        closeJob(connection, job.getId());
        awaitCompletion(connection, job, batchInfoList);
        checkResults(connection, job, batchInfoList);
    }
```

ログインと BulkConnection の設定

次のコードは、パートナー接続 (PartnerConnection) を使用してログインし、セッションを再利用して Bulk API 接続 (BulkConnection) を作成します。

```
    /**
     * Create the BulkConnection used to call Bulk API operations.
     */
    private BulkConnection getBulkConnection(String userName, String password)
          throws ConnectionException, AsyncApiException {
        ConnectorConfig partnerConfig = new ConnectorConfig();
        partnerConfig.setUsername(userName);
        partnerConfig.setPassword(password);
       partnerConfig.setAuthEndpoint("https://login.salesforce.com/services/Soap/u/37.0");

        // Creating the connection automatically handles login and stores
```

```
        // the session in partnerConfig
        new PartnerConnection(partnerConfig);
        // When PartnerConnection is instantiated, a login is implicitly
        // executed and, if successful,
        // a valid session is stored in the ConnectorConfig instance.
        // Use this key to initialize a BulkConnection:
        ConnectorConfig config = new ConnectorConfig();
        config.setSessionId(partnerConfig.getSessionId());
        // The endpoint for the Bulk API service is the same as for the normal
        // SOAP uri until the /Soap/ part. From here it's '/async/versionNumber'
        String soapEndpoint = partnerConfig.getServiceEndpoint();
        String apiVersion = "37.0";
        String restEndpoint = soapEndpoint.substring(0, soapEndpoint.indexOf("Soap/"))
            + "async/" + apiVersion;
        config.setRestEndpoint(restEndpoint);
        // This should only be false when doing debugging.
        config.setCompression(true);
        // Set this to true to see HTTP requests and responses on stdout
        config.setTraceMessage(false);
        BulkConnection connection = new BulkConnection(config);
        return connection;
    }
```

この BulkConnection **インスタンスは、**Bulk API **を使用するための基盤であり、アプリケーションのライフサイクルにわたって繰り返し再利用できます。**

ジョブの作成

接続が作成できたら、ジョブを作成します。データは常にジョブ単位で処理されます。ジョブは、処理するデータの詳細、つまり実行する処理の種類(挿入、更新、更新/挿入、削除)やオブジェクト種別を指定します。次のコードは、Account **オブジェクトを対象とした挿入処理のジョブを新規作成します。**

```
    /**
     * Create a new job using the Bulk API.
     *
     * @param sobjectType
     *            The object type being loaded, such as "Account"
     * @param connection
     *            BulkConnection used to create the new job.
     * @return The JobInfo for the new job.
     * @throws AsyncApiException
     */
    private JobInfo createJob(String sobjectType, BulkConnection connection)
          throws AsyncApiException {
        JobInfo job = new JobInfo();
        job.setObject(sobjectType);
        job.setOperation(OperationEnum.insert);
        job.setContentType(ContentType.CSV);
        job = connection.createJob(job);
        System.out.println(job);
```

```
        return job;
    }
```

作成したばかりのジョブの状態は、Open になります。ジョブがこの状態にある間は、新しいバッチをジョブに追加できます。ジョブの状態が Closed になると、それ以上バッチを追加することはできません。

ジョブへのバッチの追加

データは一連のバッチ要求を通じて処理されます。各要求は、リクエストボディに XML 形式のデータセットを含む HTTP POST です。「Bulk API の制限」 (ページ 90)で説明されているバッチサイズと、1 日あたりの処理バッチ数の上限を超過しない限り、データセット全体をどの程度に分割して処理するかは、クライアントアプリケーション側で決定できます。

各バッチの処理にはオーバーヘッドが伴います。オーバーヘッドの処理コストを最小限に抑えられるよう、バッチを処理と転送に適したサイズに調整する必要があります。レコード数 1,000 ～ 10,000 件の範囲が、適切なバッチサイズとみなされます。

次のコードは、CSV ファイルを小さいバッチファイルに分割して、Salesforce にアップロードします。

```
/**
 * Create and upload batches using a CSV file.
 * The file into the appropriate size batch files.
 *
 * @param connection
 *            Connection to use for creating batches
 * @param jobInfo
 *            Job associated with new batches
 * @param csvFileName
 *            The source file for batch data
 */
private List<BatchInfo> createBatchesFromCSVFile(BulkConnection connection,
      JobInfo jobInfo, String csvFileName)
        throws IOException, AsyncApiException {
    List<BatchInfo> batchInfos = new ArrayList<BatchInfo>();
    BufferedReader rdr = new BufferedReader(
        new InputStreamReader(new FileInputStream(csvFileName))
    );
    // read the CSV header row
    byte[] headerBytes = (rdr.readLine() + "\n").getBytes("UTF-8");
    int headerBytesLength = headerBytes.length;
    File tmpFile = File.createTempFile("bulkAPIInsert", ".csv");

    // Split the CSV file into multiple batches
    try {
        FileOutputStream tmpOut = new FileOutputStream(tmpFile);
        int maxBytesPerBatch = 10000000; // 10 million bytes per batch
        int maxRowsPerBatch = 10000; // 10 thousand rows per batch
        int currentBytes = 0;
        int currentLines = 0;
        String nextLine;
        while ((nextLine = rdr.readLine()) != null) {
            byte[] bytes = (nextLine + "\n").getBytes("UTF-8");
```

```
            // Create a new batch when our batch size limit is reached
            if (currentBytes + bytes.length > maxBytesPerBatch
              || currentLines > maxRowsPerBatch) {
                createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
                currentBytes = 0;
                currentLines = 0;
            }
            if (currentBytes == 0) {
                tmpOut = new FileOutputStream(tmpFile);
                tmpOut.write(headerBytes);
                currentBytes = headerBytesLength;
                currentLines = 1;
            }
            tmpOut.write(bytes);
            currentBytes += bytes.length;
            currentLines++;
        }
        // Finished processing all rows
        // Create a final batch for any remaining data
        if (currentLines > 1) {
            createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
        }
    } finally {
        tmpFile.delete();
    }
    return batchInfos;
}

/**
 * Create a batch by uploading the contents of the file.
 * This closes the output stream.
 *
 * @param tmpOut
 *            The output stream used to write the CSV data for a single batch.
 * @param tmpFile
 *            The file associated with the above stream.
 * @param batchInfos
 *            The batch info for the newly created batch is added to this list.
 * @param connection
 *            The BulkConnection used to create the new batch.
 * @param jobInfo
 *            The JobInfo associated with the new batch.
 */
private void createBatch(FileOutputStream tmpOut, File tmpFile,
  List<BatchInfo> batchInfos, BulkConnection connection, JobInfo jobInfo)
          throws IOException, AsyncApiException {
    tmpOut.flush();
    tmpOut.close();
    FileInputStream tmpInputStream = new FileInputStream(tmpFile);
    try {
        BatchInfo batchInfo =
          connection.createBatchFromStream(jobInfo, tmpInputStream);
        System.out.println(batchInfo);
        batchInfos.add(batchInfo);
```

```
        } finally {
            tmpInputStream.close();
        }
    }
```

サーバは、受け取ったバッチをただちに処理待ちのキューに格納します。バッチの送信時には、形式のエラーは報告はされません。こうしたエラーはバッチの処理が完了した後、結果データとして報告されます。

ヒント: バイナリ添付ファイルをインポートするには、以下のメソッドを使用して、`batchContent` パラメータにバッチのコンテンツ (CSV、XML、または JSON のいずれか) を指定するか、添付ファイルに `request.txt` を追加して、`batchContent` パラメータには `null` を渡します。これらのメソッドは、`com.async.BulkConnection` クラス内に含まれます。

- `createBatchFromDir()`
- `createBatchWithFileAttachments()`
- `createBatchWithInputStreamAttachments()`
- `createBatchFromZipStream()`

ジョブの終了

すべてのバッチをジョブに追加したら、ジョブを終了します。ジョブを終了すると、すべてのバッチの処理が確実に完了します。

```
    private void closeJob(BulkConnection connection, String jobId)
           throws AsyncApiException {
        JobInfo job = new JobInfo();
        job.setId(jobId);
        job.setState(JobStateEnum.Closed);
        connection.updateJob(job);
    }
```

バッチの状況の確認

バッチはバックグラウンドで処理されます。バッチが完了するまでの処理時間は、データセットのサイズによって異なります。処理の実行中に、すべてのバッチの状況を取得して、バッチが完了しているかどうかを確認することができます。

```
    /**
     * Wait for a job to complete by polling the Bulk API.
     *
     * @param connection
     *            BulkConnection used to check results.
     * @param job
     *            The job awaiting completion.
     * @param batchInfoList
     *            List of batches for this job.
     * @throws AsyncApiException
```

```
     */
    private void awaitCompletion(BulkConnection connection, JobInfo job,
          List<BatchInfo> batchInfoList)
            throws AsyncApiException {
        long sleepTime = 0L;
        Set<String> incomplete = new HashSet<String>();
        for (BatchInfo bi : batchInfoList) {
            incomplete.add(bi.getId());
        }
        while (!incomplete.isEmpty()) {
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException e) {}
            System.out.println("Awaiting results..." + incomplete.size());
            sleepTime = 10000L;
            BatchInfo[] statusList =
              connection.getBatchInfoList(job.getId()).getBatchInfo();
            for (BatchInfo b : statusList) {
                if (b.getState() == BatchStateEnum.Completed
                  || b.getState() == BatchStateEnum.Failed) {
                    if (incomplete.remove(b.getId())) {
                        System.out.println("BATCH STATUS:\n" + b);
                    }
                }
            }
        }
    }
```

バッチは、状況がFailedかCompletedのいずれかになった場合に終了となります。このコードでは、ジョブのすべてのバッチが終了するまでループ処理を実行します。

ジョブの結果の取得

すべてのバッチの処理が完了したら、各バッチの結果を取得できます。バッチが成功した場合も、失敗した場合も、またジョブが途中で中止された場合も、必ず結果を取得してください。結果セットを取得しないと、個々のレコードの状況を確認できません。各レコードの結果を正しく取得するには、バッチと対応する元のデータセットをコード内で正確に追跡する必要がありまが、そのためには、バッチの作成時のリストを保持し、結果の取得に使用するようにします。次のコードはそのための処理を記述しています。

```
    /**
     * Gets the results of the operation and checks for errors.
     */
    private void checkResults(BulkConnection connection, JobInfo job,
              List<BatchInfo> batchInfoList)
            throws AsyncApiException, IOException {
        // batchInfoList was populated when batches were created and submitted
        for (BatchInfo b : batchInfoList) {
            CSVReader rdr =
              new CSVReader(connection.getBatchResultStream(job.getId(), b.getId()));
            List<String> resultHeader = rdr.nextRecord();
            int resultCols = resultHeader.size();
```

```
            List<String> row;
            while ((row = rdr.nextRecord()) != null) {
                Map<String, String> resultInfo = new HashMap<String, String>();
                for (int i = 0; i < resultCols; i++) {
                    resultInfo.put(resultHeader.get(i), row.get(i));
                }
                boolean success = Boolean.valueOf(resultInfo.get("Success"));
                boolean created = Boolean.valueOf(resultInfo.get("Created"));
                String id = resultInfo.get("Id");
                String error = resultInfo.get("Error");
                if (success && created) {
                    System.out.println("Created row with id " + id);
                } else if (!success) {
                    System.out.println("Failed with error: " + error);
                }
            }
        }
    }
```

このコードは、各レコードの結果を取得し、処理が成功したか失敗したかを報告します。レコードでエラーが発生した場合にはエラーを出力します。

クイックスタートサンプルの完全版

ジョブやバッチについての理解が深まったでしょうか。クイックスタートサンプルのコード全体を次に示します。コピーして活用してください。

```

import java.io.*;
import java.util.*;

import com.sforce.async.*;
import com.sforce.soap.partner.PartnerConnection;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;


public class BulkExample {


    public static void main(String[] args)
      throws AsyncApiException, ConnectionException, IOException {
        BulkExample example = new BulkExample();
        // Replace arguments below with your credentials and test file name
        // The first parameter indicates that we are loading Account records
       example.runSample("Account", "myUser@myOrg.com", "myPassword", "mySampleData.csv");

    }

    /**
     * Creates a Bulk API job and uploads batches for a CSV file.
     */
    public void runSample(String sobjectType, String userName,
```

```
            String password, String sampleFileName)
        throws AsyncApiException, ConnectionException, IOException {
    BulkConnection connection = getBulkConnection(userName, password);
    JobInfo job = createJob(sobjectType, connection);
    List<BatchInfo> batchInfoList = createBatchesFromCSVFile(connection, job,
        sampleFileName);
    closeJob(connection, job.getId());
    awaitCompletion(connection, job, batchInfoList);
    checkResults(connection, job, batchInfoList);
}



/**
 * Gets the results of the operation and checks for errors.
 */
private void checkResults(BulkConnection connection, JobInfo job,
          List<BatchInfo> batchInfoList)
        throws AsyncApiException, IOException {
    // batchInfoList was populated when batches were created and submitted
    for (BatchInfo b : batchInfoList) {
        CSVReader rdr =
          new CSVReader(connection.getBatchResultStream(job.getId(), b.getId()));
        List<String> resultHeader = rdr.nextRecord();
        int resultCols = resultHeader.size();

        List<String> row;
        while ((row = rdr.nextRecord()) != null) {
            Map<String, String> resultInfo = new HashMap<String, String>();
            for (int i = 0; i < resultCols; i++) {
                resultInfo.put(resultHeader.get(i), row.get(i));
            }
            boolean success = Boolean.valueOf(resultInfo.get("Success"));
            boolean created = Boolean.valueOf(resultInfo.get("Created"));
            String id = resultInfo.get("Id");
            String error = resultInfo.get("Error");
            if (success && created) {
                System.out.println("Created row with id " + id);
            } else if (!success) {
                System.out.println("Failed with error: " + error);
            }
        }
    }
}



private void closeJob(BulkConnection connection, String jobId)
      throws AsyncApiException {
    JobInfo job = new JobInfo();
    job.setId(jobId);
    job.setState(JobStateEnum.Closed);
    connection.updateJob(job);
}
```

```
/**
 * Wait for a job to complete by polling the Bulk API.
 *
 * @param connection
 *            BulkConnection used to check results.
 * @param job
 *            The job awaiting completion.
 * @param batchInfoList
 *            List of batches for this job.
 * @throws AsyncApiException
 */
private void awaitCompletion(BulkConnection connection, JobInfo job,
      List<BatchInfo> batchInfoList)
        throws AsyncApiException {
    long sleepTime = 0L;
    Set<String> incomplete = new HashSet<String>();
    for (BatchInfo bi : batchInfoList) {
        incomplete.add(bi.getId());
    }
    while (!incomplete.isEmpty()) {
        try {
            Thread.sleep(sleepTime);
        } catch (InterruptedException e) {}
        System.out.println("Awaiting results..." + incomplete.size());
        sleepTime = 10000L;
        BatchInfo[] statusList =
          connection.getBatchInfoList(job.getId()).getBatchInfo();
        for (BatchInfo b : statusList) {
            if (b.getState() == BatchStateEnum.Completed
              || b.getState() == BatchStateEnum.Failed) {
                if (incomplete.remove(b.getId())) {
                    System.out.println("BATCH STATUS:\n" + b);
                }
            }
        }
    }
}



/**
 * Create a new job using the Bulk API.
 *
 * @param sobjectType
 *            The object type being loaded, such as "Account"
 * @param connection
 *            BulkConnection used to create the new job.
 * @return The JobInfo for the new job.
 * @throws AsyncApiException
 */
private JobInfo createJob(String sobjectType, BulkConnection connection)
```

```
      throws AsyncApiException {
    JobInfo job = new JobInfo();
    job.setObject(sobjectType);
    job.setOperation(OperationEnum.insert);
    job.setContentType(ContentType.CSV);
    job = connection.createJob(job);
    System.out.println(job);
    return job;
}



/**
 * Create the BulkConnection used to call Bulk API operations.
 */
private BulkConnection getBulkConnection(String userName, String password)
      throws ConnectionException, AsyncApiException {
    ConnectorConfig partnerConfig = new ConnectorConfig();
    partnerConfig.setUsername(userName);
    partnerConfig.setPassword(password);
   partnerConfig.setAuthEndpoint("https://login.salesforce.com/services/Soap/u/37.0");

    // Creating the connection automatically handles login and stores
    // the session in partnerConfig
    new PartnerConnection(partnerConfig);
    // When PartnerConnection is instantiated, a login is implicitly
    // executed and, if successful,
    // a valid session is stored in the ConnectorConfig instance.
    // Use this key to initialize a BulkConnection:
    ConnectorConfig config = new ConnectorConfig();
    config.setSessionId(partnerConfig.getSessionId());
    // The endpoint for the Bulk API service is the same as for the normal
    // SOAP uri until the /Soap/ part. From here it's '/async/versionNumber'
    String soapEndpoint = partnerConfig.getServiceEndpoint();
    String apiVersion = "37.0";
    String restEndpoint = soapEndpoint.substring(0, soapEndpoint.indexOf("Soap/"))
        + "async/" + apiVersion;
    config.setRestEndpoint(restEndpoint);
    // This should only be false when doing debugging.
    config.setCompression(true);
    // Set this to true to see HTTP requests and responses on stdout
    config.setTraceMessage(false);
    BulkConnection connection = new BulkConnection(config);
    return connection;
}



/**
 * Create and upload batches using a CSV file.
 * The file into the appropriate size batch files.
 *
 * @param connection
 *            Connection to use for creating batches
```

```
     * @param jobInfo
     *            Job associated with new batches
     * @param csvFileName
     *            The source file for batch data
     */
    private List<BatchInfo> createBatchesFromCSVFile(BulkConnection connection,
          JobInfo jobInfo, String csvFileName)
            throws IOException, AsyncApiException {
        List<BatchInfo> batchInfos = new ArrayList<BatchInfo>();
        BufferedReader rdr = new BufferedReader(
            new InputStreamReader(new FileInputStream(csvFileName))
        );
        // read the CSV header row
        byte[] headerBytes = (rdr.readLine() + "\n").getBytes("UTF-8");
        int headerBytesLength = headerBytes.length;
        File tmpFile = File.createTempFile("bulkAPIInsert", ".csv");

        // Split the CSV file into multiple batches
        try {
            FileOutputStream tmpOut = new FileOutputStream(tmpFile);
            int maxBytesPerBatch = 10000000; // 10 million bytes per batch
            int maxRowsPerBatch = 10000; // 10 thousand rows per batch
            int currentBytes = 0;
            int currentLines = 0;
            String nextLine;
            while ((nextLine = rdr.readLine()) != null) {
                byte[] bytes = (nextLine + "\n").getBytes("UTF-8");
                // Create a new batch when our batch size limit is reached
                if (currentBytes + bytes.length > maxBytesPerBatch
                  || currentLines > maxRowsPerBatch) {
                    createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
                    currentBytes = 0;
                    currentLines = 0;
                }
                if (currentBytes == 0) {
                    tmpOut = new FileOutputStream(tmpFile);
                    tmpOut.write(headerBytes);
                    currentBytes = headerBytesLength;
                    currentLines = 1;
                }
                tmpOut.write(bytes);
                currentBytes += bytes.length;
                currentLines++;
            }
            // Finished processing all rows
            // Create a final batch for any remaining data
            if (currentLines > 1) {
                createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
            }
        } finally {
            tmpFile.delete();
        }
        return batchInfos;
    }
```

```
    /**
     * Create a batch by uploading the contents of the file.
     * This closes the output stream.
     *
     * @param tmpOut
     *            The output stream used to write the CSV data for a single batch.
     * @param tmpFile
     *            The file associated with the above stream.
     * @param batchInfos
     *            The batch info for the newly created batch is added to this list.
     * @param connection
     *            The BulkConnection used to create the new batch.
     * @param jobInfo
     *            The JobInfo associated with the new batch.
     */
    private void createBatch(FileOutputStream tmpOut, File tmpFile,
      List<BatchInfo> batchInfos, BulkConnection connection, JobInfo jobInfo)
              throws IOException, AsyncApiException {
        tmpOut.flush();
        tmpOut.close();
        FileInputStream tmpInputStream = new FileInputStream(tmpFile);
        try {
            BatchInfo batchInfo =
              connection.createBatchFromStream(jobInfo, tmpInputStream);
            System.out.println(batchInfo);
            batchInfos.add(batchInfo);

        } finally {
            tmpInputStream.close();
        }
    }


}
```

付録 B　　データ項目の対応付け

Microsoft Outlook、Google コンタクト、およびその他のサードパーティソースから直接エクスポートされたデータを Bulk API を使用してインポートするには、CSV インポートファイルのデータ項目をすべて Salesforce データ項目に対応付けます。CSV インポートファイルは、Bulk API に準拠している必要はありません。

たとえば、CSV インポートファイルに含まれている `Number` という項目を、標準の Salesforce 項目である `AccountNumber` に対応付けるとします。Bulk API を使用する一括処理ジョブを追加すると、`Number` 項目からのデータは、Salesforce の `AccountNumber` 項目にインポートされるか、項目を更新します。

データ項目を Salesforce データ項目に対応付ける一括処理ジョブを追加する手順は、次のとおりです。

1. データ項目の対応付けを定義する変換仕様 (spec.csv) を作成します (これは、データが含まれる CSV インポートファイルとは異なります)。
2. 他の Bulk API ジョブと同様に、オブジェクトとアクションを指定する新しいジョブを作成します。
3. 変換仕様をアップロードします。
4. データをバッチでサーバに送信します。

対応付けを定義する変換仕様の作成

変換仕様 (spec.csv) は、インポートファイルのデータを、Salesforce データ項目に対応付ける手順を提供します。

spec.csv ファイルには次の 4 つの項目が含まれます。

項目	説明
Salesforce 項目	対応付け先の Salesforce 項目。
Csv Header	対応付けるインポートファイル内の項目。
Value	デフォルト値。 Bulk API は、この値を次の 2 つの場合に使用します。 • Csv Header 項目で指定した項目に対する値がインポートファイル内に存在しない場合 • Csv Header 項目の値が spec.csv ファイルに定義されていない場合 この項目は省略可能です。

項目	説明
Hint	Bulk API に、インポートファイルのデータ解釈方法を指示します。 Bulk API は、この値を使用して次の 2 つの処理を実行できます。 • 日時項目の Java 形式の文字列を解釈する • 正規表現を使用し、Boolean 項目で true となる条件を定義する この項目は省略可能です。

次に、サンプルの spec.csv ファイルを示します。

```
Salesforce Field,Csv Header,Value,Hint
Name,Full Name,,
Title,Job Title,,
LeadSource,Lead Source,Import,
Description,,Imported from XYZ.csv,
Birthdate,Date of Birth,,dd MM yy
```

この spec.csv ファイルは、Bulk API に次の処理を行うように指示します。

- インポートファイルの `[Full Name]` 項目を、Salesforce の `LastName` 項目と `FirstName` 項目に対応付ける。
- インポートファイルの `Title` 項目を、Salesforce の `Title` 項目に対応付ける。
- インポートファイルの `[Lead Source]` 項目を、Salesforce の `LeadSource` 項目に対応付け、さらに、インポートファイルに値がない場合は `Import as the default value` を使用する。
- `Imported from XYZ.csv` を、Salesforce の `Description` 項目のデフォルト値として使用する。
- インポートファイルの `[Date of Birth]` 項目を、Salesforce の `Birthdate` 項目に対応付け、さらに、`dd MM yy` 形式を使用して `[Date of Birth]` 項目形式を Bulk API で受け入れ可能な形式に変換する。

インポートファイルの対応する内容は次のようになります。

```
Full Name,Job Title,Lead Source,Date of Birth,Comment
"Cat, Tom",DSH,Interview,10 Feb 40,likes Jerry
Jerry Mouse,House Mouse,,10 Feb 40,likes Tom
```

変換後の対応するリクエストボディは次のようになります。

```
LastName,FirstName,Title,LeadSource,Description,Birthdate
Cat,Tom,DSH,Interview,Imported from XYZ.csv,1940-02-10Z
Mouse,Jerry,House Mouse,Import,Imported from XYZ.csv,1940-02-10Z
```

変換仕様のアップロード

変換仕様をアップロードするには、POST 要求を次の URI に送信します。

```
https://instance_name—api.salesforce.com/services/async/APIversion/job/jobid/spec
```

考慮事項

- 変換仕様は、CSV ファイルである必要があります。XML および JSON ファイルはサポートされていません。
- 変換仕様 (spec.csv ファイル) は、UTF-8 文字コードを使用する必要があります。CSV インポートファイルは、UTF-8 文字コードを使用する必要はありません (文字コードを Content-Type ヘッダーに指定できます)。
- 変換仕様は、永続的ではなく、その範囲は現在のジョブに限定されています。

用語集

A |B |C |D |E |F |G |H |I |J |K |L |M |N |O |P |Q |R |S |T |U |V |W |X |Y |Z

A

Apex

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

App

「アプリケーション」の短縮形です。特定のビジネス要件を扱うタブ、レポート、ダッシュボードおよび Visualforce ページなどのコンポーネントの集合です。Salesforce では、セールスおよびコールセンターなどの標準アプリケーションを提供しています。お客様のニーズに合わせてこれらの標準アプリケーションをカスタマイズできます。また、アプリケーションをパッケージ化して、カスタム項目、カスタムタブ、カスタムオブジェクトなどの関連コンポーネントと共に AppExchange にアップロードできます。そのアプリケーションを AppExchange から他の Salesforce ユーザが利用できるようにすることもできます。

アプリケーションプログラムインターフェース (API)

コンピュータシステム、ライブラリ、またはアプリケーションが、その他のコンピュータプログラムがサービスを要求したりデータを交換したりできる機能を提供するインターフェースです。

非同期コール

操作に長い時間がかかるため、直ちに結果を返さないコールです。メタデータ API と Bulk API のコールは非同期です。

B

バッチ (Bulk API)

Bulk API でまとめて処理されるレコードのセットです。ファイル形式は CSV、XML、または JSON のいずれかになります。レコードセットを処理するには、1 つ以上のバッチを含むジョブを作成します。各バッチはサーバによって独自に処理されます。受信した順序で処理されるとは限りません。「ジョブ (Bulk API)」も参照してください。

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
REST ベースの Bulk API は、大規模データセットの処理用に最適化されています。Salesforce によりバックグラウンドで処理される複数のバッチを送信することにより、多数のレコードを非同期でクエリ、挿入、更新、更新/挿入または削除できます。「SOAP API」も参照してください。

C

クライアントアプリケーション
Salesforce ユーザインターフェースの外部で実行し、Force.com API または Bulk API のみを使用するアプリケーションです。通常、デスクトップまたはモバイルデバイス上で稼動します。これらのアプリケーションは、プラットフォームをデータソースとして扱い、設計されたツールおよびプラットフォームの開発モデルを使用します。

CSV (カンマ区切り値)
構造化データの共有と移動を可能にするファイル形式です。Salesforce のインポートウィザード、データローダ、Bulk API は CSV をサポートしています。CSV ファイルの各行は 1 件のレコードを表します。レコード内の各項目の値はカンマで区切られます。

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です。

D

データローダ
Salesforce 組織からデータをインポートおよびエクスポートするために使用する Force.com プラットフォームのツールです。

データベース
情報の編成された集合です。Force.com プラットフォームの基底となるアーキテクチャには、データが格納されているデータベースが含まれています。

データベーステーブル
追跡する必要のある人物、物事、またはコンセプトに関する情報のリストで、行および列で表示されます。「オブジェクト」も参照してください。

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です。たとえば、4.98 の場合は 2 となります。これ以上の桁の数値を入力した場合は、四捨五入されます。たとえば、[小数点の位置] が 2 の場合に 4.986 と入力すると、その数値は 4.99 となります。Salesforce では、round half up アルゴリズムを使用します。中間値は常に四捨五入されます。たとえば、1.45 は 1.5 に切り上げられます。–1.45 は –1.5 に切り上げられます。

連動項目
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、カスタム選択リストまたは複数選択リストの項目です。

Salesforce 開発者
Salesforce 開発者 Web サイト (developer.salesforce.com) では、サンプルコード、ツールキット、オンライン開発者コミュニティなど、プラットフォーム開発者向けの幅広いリソースを提供しています。開発向けのForce.com プラットフォーム環境も、ここから入手できます。

E

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です。

項目レベルセキュリティ
項目が、ユーザに非表示、表示、参照のみ、または編集可能であるかどうかを決定する設定です。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。

Force.com
クラウドでアプリケーションを構築するための Salesforce プラットフォーム。Force.com は、強力なユーザインターフェース、オペレーティングシステムおよびデータベースを結合して、企業全体でアプリケーションをカスタマイズおよび展開できます。

Force.com IDE
開発者がEclipse開発環境でForce.comアプリケーションを管理、作成、デバッグおよびリリースできるEclipse プラグイン。

Force.com 移行ツール
ローカルファイルシステムと Salesforce 組織との間で Force.com コンポーネントを移行する Apache Ant 開発スクリプトを作成するためのツールキット。

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です。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。

H

該当用語はありません。

I

ID
「Salesforce レコード ID」を参照してください。

インスタンス
組織のデータをホストし、アプリケーションを実行する単一の論理サーバとして示されるソフトウェアおよびハードウェアのクラスタです。Force.com プラットフォームは複数のインスタンスで稼動しますが、1つの組織のデータは常に 1 つのインスタンスに保存されています。

インテグレーションユーザ
クライアントアプリケーションまたはインテグレーションのみを対象に定義された Salesforce ユーザです。また、SOAP API コンテキストではログインユーザとも呼ばれます。

ISO コード
国際標準化機構が定める国コードで、各国を 2 文字で表します。

J

ジョブ (Bulk API)
Bulk API の「ジョブ」では、処理するオブジェクト (取引先、商談など) と、アクションのタイプ (挿入、更新、更新/挿入、削除)を指定します。レコードセットを処理するには、1つ以上のバッチを含むジョブを作成します。「バッチ (Bulk API)」も参照してください。

JSON (JavsScript Object Notation)
JSON は、データを転送するための軽量の形式です。

K

該当用語はありません。

L

ログインユーザ
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とも呼ばれます。

参照項目
別のレコードに対するリンク可能な値を含む項目の種別です。オブジェクトに別のオブジェクトとの参照関係または主従関係がある場合、ページレイアウトに参照項目を表示できます。たとえば、ケースに納入

商品との参照関係がある場合、ケース詳細ページからルックアップダイアログを使用して納入商品を選択し、ケース詳細ページから納入商品の名前をクリックできます。

M

管理パッケージ
ユニットとしてAppExchangeに投稿され、名前空間と、場合によりライセンス管理組織に関連付けられるアプリケーションコンポーネントの集合です。アップグレードをサポートするには、管理パッケージであることが必要です。組織は、他の多くの組織でダウンロードおよびインストールできる単一の管理パッケージを作成できます。管理パッケージは、未管理パッケージとは異なり、コンポーネントの一部がロックされていて、後でアップグレードできます。未管理パッケージには、ロックされたコンポーネントは含まれておらず、アップグレードはできません。また、管理パッケージでは、開発者の知的財産保護のため、登録している組織では特定のコンポーネント (Apex など) は隠されます。

共有の直接設定
レコード所有者がレコードにアクセス権を持たないユーザに参照権限および編集権限を与えることができるレコードレベルのアクセスルールです。

多対多リレーション
リレーションの両端に多くの子があるリレーションです。多対多リレーションは、連結オブジェクトを使用して実装されます。

メタデータ
組織およびいずれかの部署の構造、外観、機能に関する情報です。Force.com では、メタデータを記述するのに XML を使用します。

マルチテナンシー
すべてのユーザおよびアプリケーションが単一で共通のインフラストラクチャおよびコードベースを共有するアプリケーションモデルです。

N

ネイティブアプリケーション
Force.comの設定(メタデータ)定義で排他的に開発されたアプリケーションです。ネイティブアプリケーションには、外部サービスまたは外部インフラストラクチャは必要ありません。

O

オブジェクト
Salesforce 組織に情報を保存するために使用するオブジェクト。オブジェクトは、保存する情報の種類の全体的な定義です。たとえば、Case オブジェクトを使用して、顧客からの問い合わせに関する情報を保存できます。各オブジェクトについて、組織は、そのデータ型の具体的なインスタンスに関する情報を保存する複数のレコードを保有します。たとえば、佐藤次郎さんから寄せられたトレーニングに関する問い合わせに関する情報を保存するケースレコードと、山田花子さんから寄せられたコンフィグレーションの問題に関する情報を保存するケースレコードなどです。

オブジェクトレベルセキュリティ
: 特定のユーザに対してオブジェクト全体を非表示にできる設定です。ユーザはそうしたデータの存在を知ることもできません。オブジェクトレベルセキュリティはオブジェクト権限で指定されます。

一対多リレーション
: 1つのオブジェクトが多数のオブジェクトに関連するリレーション。たとえば、取引先に1つまたは複数の関連取引先責任者がある場合があります。

組織の共有設定
: ユーザが組織で持つデータアクセスのベースラインレベルを指定できる設定。たとえば、オブジェクト権限によって有効化されている特定のオブジェクトの任意のレコードを参照できますが、編集するには別の権限が必要となるよう、組織の共有設定を設定できます。

アウトバウンドメッセージ
: アウトバウンドメッセージでは、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。アウトバウンドメッセージは[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。

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。

P

パッケージ
: AppExchangeを介して他の組織で使用可能なForce.comのコンポーネントおよびアプリケーションのグループです。AppExchangeにまとめてアップロードできるように、パッケージを使用してアプリケーションおよび関連するコンポーネントをバンドルします。

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です。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。

Platform Edition
: セールスやサービス&サポートなどの標準Salesforceアプリケーションを含まないEnterprise Edition、Unlimited Edition、またはPerformance Editionに基づいたSalesforceエディションです。

主キー
: リレーショナルデータベースのコンセプトです。リレーショナルデータベースの各テーブルには、データ値が一意にレコードを識別する項目があります。この項目を、主キーと呼びます。2つのテーブルのリレーションは、あるテーブルの外部キーの値と、別のテーブルの主キーの値が一致することによって成り立ちます。

本番組織
実際の本番データとそれらにアクセスするライブユーザを持っている Salesforce 組織です。

Q

クエリ文字列パラメータ
通常 URL の「?」文字の後に指定されている名前 - 値のペアです。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 Force.com アプリケーションに表示されると、値はレコードの詳細ビューへのリンクとして表示されます。レコード名は自由形式のテキストまたは自動採番項目です。［レコード名］には、必ずしも一意の値を割り当てる必要はありません。

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、特定のユーザがオブジェクトを参照および編集でき、ユーザが編集できるレコードを制限できます。

ごみ箱
削除した情報を表示し、復元できるページです。ごみ箱には、サイドバー内のリンク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です。親対子および子対親の構文は、SOQL クエリでは異なります。

ロール階層
レコードレベルのセキュリティで使用される設定です。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は、組織の共有モデルとは関係なく、階層において自分よりも下位のユーザが所有しているデータ、および該当のユーザと共有しているデータに対する参照、編集権限を持つことになります。

積み上げ集計項目
主従関係の子レコードの値の集計値を自動的に提供する項目の種別です。

実行ユーザ
各ダッシュボードには*実行ユーザ*が指定され、そのユーザのセキュリティ設定によってダッシュボードに表示されるデータが決まります。実行ユーザが特定の 1 ユーザである場合、すべてのダッシュボード閲覧者には、閲覧者個々人のセキュリティ設定に関係なく、実行ユーザのセキュリティ設定に基づいてデータが表示されます。動的ダッシュボードの場合、実行ユーザをログインユーザに設定することができるため、各ユーザには独自のアクセスレベルに従ってダッシュボードが表示されます。

S

SaaS
「サービスとしてのソフトウェア (SaaS)」を参照してください。

Salesforce レコード ID
Salesforce の 1 つのレコードを識別する 15 文字または 18 文字の一意の英数字文字列です。

Salesforce SOA (サービス指向アーキテクチャ)
Apex 内から外部 Web サービスへのコールを実行できる Force.com の強力な機能です。

Sandbox
開発、テストおよびトレーニング用の、Salesforce 本番組織とほぼ同一のコピー。Sandbox のコンテンツとサイズは、Sandbox の種別および Sandbox に関連付けられた本番組織のエディションによって異なります。

セッション ID
ユーザが Salesforce に正常にログインした場合に返される認証トークンです。セッション ID を使用すると、ユーザが Salesforce で別のアクションを実行するときに毎回ログインする必要がなくなります。レコード ID または Salesforce ID と異なり、Salesforce レコードの一意の ID を示す用語です。

セッションタイムアウト
ログインしてからユーザが自動的にログアウトするまでの時間です。セッションは、前もって決定された非活動状態の期間の後、自動的に終了します。非活動状態の期間の長さは、[設定] の [セキュリティのコントロール] をクリックすることによって Salesforce で設定できます。デフォルト値は 120 分 (2 時間) です。ユーザが Web インターフェースでアクションを実行または API コールを実行すると、非活動状態タイマーが 0 にリセットされます。

設定
システム管理者が組織の設定および Force.com アプリケーションをカスタマイズおよび定義できるメニューです。組織のユーザインターフェース設定に応じて、[設定] はユーザインターフェースのヘッダーでリンクになっている場合もあれば、ユーザ名の下でドロップダウンリストになっている場合もあります。

SOAP (Simple Object Access Protocol)
XML 符号化データを渡す一定の方法を定義するプロトコルです。

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です。SaaS ベンダは、アプリケーションおよび各顧客データの日常メンテナンス、操作およびサポートを行う責任があります。このサービスで、顧客が独自のハードウェア、ソフトウェア、そして関連 IT リソースを使用してアプリケーションをインストール、構成、保守する必要性を緩和します。SaaS モデルを使用して、あらゆる市場区分にサービスを配信することができます。

SOQL (Salesforce オブジェクトクエリ言語)
Force.com データベースからデータを選択する条件を指定するために使う、単純で強力なクエリ文字列を構築できるクエリ言語です。

SOSL (Salesforce オブジェクト検索言語)
Force.com API を使用して、テキストベースの検索を実行できるクエリ言語です。

標準オブジェクト
Force.com プラットフォームに含まれる組み込みオブジェクトです。アプリケーション独自の情報を格納するカスタムオブジェクトを作成することもできます。

T

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を使用して、翻訳する言語を指定し、翻訳者を言語に割り当て、Salesforce 組織に作成したカスタマイズの翻訳を作成し、管理対象パッケージから表示ラベルと翻訳を上書きすることができます。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、海外のユーザが Salesforce のすべてを彼らの言語で使用できるようになりました。

U

V

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、タグベースのシンプルなマークアップ言語です。各タグが、ページのセクション、関連リスト、または項目など、大まかなコンポーネントときめの細かいコンポーネントのどちらにも対応しています。コンポーネントの動作は、標準の Salesforce ページと同じロジックを使用して制御することも、開発者が独自のロジックを Apex で記述されたコントローラと関連付けることもできます。

W

Web サービス
様々なプラットフォームで稼動、さまざまな言語で作成、またはお互い地理的に離れている場合であっても、2 つのアプリケーションがインターネットを経由してデータを容易に交換できるメカニズムです。

Web サービス API
Salesforce組織の情報へのアクセスを提供するWebサービスアプリケーションプログラミングインターフェース。「SOAP API」および「Bulk API」も参照してください。

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 ファイルです。開発環境の SOAP クライアントは、Salesforce Enterprise WSDL または Partner WSDL を使用して、SOAP API で Salesforce と通信します。

X

XML (拡張可能マークアップ言語)
構造化データの共有と移動を可能にするマークアップ言語です。メタデータ API を使用して取得またはリリースされるすべての Force.com コンポーネントは、XML 定義に従って表されます。

Y

該当用語はありません。

Z

該当用語はありません。