今まで知らなかった 5 つの事項: Java Database Connectivity

JDBC API に対する認識を改める

Ted Neward
Principal
Neward & Associates

Alex Theedom
Senior Java developer
Consultant

2017年 8月 31日
(初版 2010年 8月 10日)

JDBC、つまり Java Database Connectivity は、JDK 全体の中で最も頻繁に使われるパッケージの 1 つでありながら、JDBC の機能をフルに利用している Java 開発者や JDBC の最新機能を利用している Java 開発者はごく稀です。この記事では Ted Neward が JDBC の新機能を紹介します。具体的には、自動的にスクロールし、オンザフライで更新できる ResultSet、データベース接続が開いているか否かによらず動作する Rowset、ネットワークを 1 度素早く往復するだけで複数の SQL 文を実行できるバッチ更新、などについて説明します。

このシリーズの他の記事を見る

今日の多くの Java 開発者は、Hibernate や Spring などのデータ・アクセス・プラットフォームを通じて JDBC (Java Database Connectivity) API を理解しています。しかし JDBC はデータベース接続の裏側で働く以上の機能を備えています。JDBC について理解すればするほど、RDBMS とのやり取りが効率的になるはずです。

今回の記事では、最近のバージョンで JDBC に導入された新機能をいくつか説明します。これらの新機能は最近のソフトウェア開発での課題を念頭に設計されており、アプリケーションのスケーラビリティーと開発者の生産性という、今日の Java 開発者が直面する共通の 2 つの課題をサポートしています。

この連載について

皆さんは自分が Java プログラミングについて知っていると思うかもしれません。しかし実際には、ほとんどの開発者は Java プラットフォームの表面的な部分しか扱っておらず、当面の作業を完了するために十分なことしか学んでいません。この連載では、Ted Neward が Java プラットフォームのコア機能を深く掘り下げ、非常に厄介な Java プログラミングの難題の解決にも役立つ、ほとんど知られていない事実を紹介します。

© Copyright IBM Corporation 2010, 2017

1. スカラー関数

さまざまな RDBMS 実装では、正規サポートではではないものの、開発者の作業を楽にするために設計された SQL や付加価値機能をサポートしています。例えば、SQL には `COUNT()` というスカラー操作があることはよく知られています。COUNT() により、特定の SQL フィルター基準 (つまり `WHERE` 述語) を満たす行数を返すことができます。しかしそうした機能以上のことをしようとして SQL によって返される値を変更するのは面倒であり、また現在の日付と時刻をデータベースから取得しようとすると、かなり辛抱強い JDBC 開発者でさえ気が狂いそうになります (そして頭が禿げてしまうかもしれません)。

そうした場合のために、JDBC 仕様にはさまざまな RDBMS 実装に対してその一部を切り離したり、変更したりするといった加工をある程度までできる機能が、スカラー関数として用意されています。JDBC 仕様には、サポートされている操作の一覧が含まれており、JDBC ドライバーはそれらの操作を認識し、必要に応じて特定のデータベース実装に対応させる必要があります。従って、現在の日付や時刻を返す機能をサポートするデータベースの場合、それを実行するためのコードはリスト 1 のように単純なものになります。

リスト 1. 現在の日時を取得する

```
Connection conn = ...; // get it from someplace
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("{fn CURRENT_DATE()}");
```

JDBC API で認識されるスカラー関数の一覧は、完全な形で JDBC の仕様の付録に記載されています (「関連トピック」を参照)。しかしドライバーやデータベースによっては、一覧に挙げられたすべての関数をサポートしているわけではない場合があります。指定された JDBC 実装でサポートされている関数を取得するには、`Connection` オブジェクトのメソッドによって返される `DatabaseMetaData` オブジェクトを利用することができます (リスト 2)。

リスト 2. サポートされている関数を取得する

```
Connection conn = ...; // get it from someplace
DatabaseMetaData dbmd = conn.getMetaData();
```

スカラー関数の一覧は、さまざまな `DatabaseMetaData` メソッドから返される `String` をカンマで区切ったものです。例えば、`getNumericFunctions()` を呼び出すと、すべての数値型スカラー関数の一覧がカンマで区切った形で返されます。その結果に対して `String.split()` を実行すると、`equals()` によって即座に調べることのできるリストのできあがりです。

指摘する価値がある点として付け加えておくと、JDBC 4 に加えられた改善の 1 つとして、`Class.forName("some driver class")` 文を安全に削除できるようになっています。

2. スクロール可能な ResultSet

`Connection` オブジェクトを作成 (または既存の Connection オブジェクトを取得) し、そのオブジェクトを使って `Statement` を作成する手続きは、JDBC では非常に一般的です。`Statement` で `SQL` の `SELECT` 文を指定して実行すると、`ResultSet` が返されます。そして `ResultSet` は `while`

ループ (Iterator と似ていないこともありません) で処理され、ResultSet が空になるまで、while ループの本体によって 1 度に 1 列ずつ、左から右の順に列が抽出されます。

この操作全体はあまりにも一般的なため、ほとんど神聖なお決まりの操作になっています。つまりこの操作は、こうして行うものだから、という理由で行われています。しかしこの操作はまったく不要なのです。

ResultSet のタイプを検証する

DatabaseMetaData の中に、スクロール可能な ResultSet をサポートすると記載されているにもかかわらず、そのドライバーが実際にはスクロール機能をサポートしていないと思える場合には、getType() を呼び出して ResultSet のタイプを検証することができます。もちろん、皆さんが非常に疑い深くなっているなら、getType() によって返される値すら信用しないかもしれません。そのような人には次のように言っておきましょう。もし getType() が返す ResultSet の情報に嘘があるのであれば、それは getType() が実際には ResultSet のタイプを取得する代わりに、疑い深い皆さんを捕まえに出ているから正しい情報が返されないのです。

スクロール可能な ResultSet を導入する

JDBC の機能は長年にわたって大幅に強化されてきており、JDBC の新しいバージョンや新しいリリースに反映されていますが、そうした事実を多くの開発者は認識していません。この記事の執筆時点で、JDBC はバージョン 4.2 になっています。

JDBC に対して行われた興味深い (ただし無視されがちな) 機能強化の 1 つに、ResultSet を「スクロール」できる機能があります。つまり必要に応じて、順方向にも逆方向にも、さらには両方向に進むこともできます。ただし、そのためにはそれを見越したコードの作成が必要で、JDBC を呼び出して Statement を作成する際に、スクロール可能な ResultSet が必要であることを示す必要があります。

JDBC ドライバーがスクロール機能をサポートしている場合は、その Statement のメソッドを実行した結果として、スクロール可能な ResultSet が返されます。ただしスクロール機能を要求する前に、ドライバーがスクロール機能をサポートしているかどうかを明らかにしておくとよいでしょう。スクロール機能に関して問い合わせる場合には DatabaseMetaData オブジェクトを利用しますが、DatabaseMetaData オブジェクトは先ほど説明したとおり、Connection オブジェクトのメソッドを使って取得することができます。

DatabaseMetaData オブジェクトを取得したら、getJDBCMajorVersion() を呼び出すことで、そのドライバーがサポートしている JDBC のメジャー・バージョン番号を確認できます。もちろん、ドライバーは指定された仕様のサポート・レベルに関して誤った情報を返す場合があるので、特に確実を期すためには、必要とする ResultSet タイプを使って supportsResultSetType() メソッドを呼び出します (ResultSet タイプは ResultSet クラスの定数です。それぞれの値に関しては、このすぐ後に説明します)。

リスト 3. スクロール機能を問い合わせる

```
int JDBCVersion = dbmd.getJDBCMajorVersion();
boolean srs = dbmd.supportsResultSetType(ResultSet.TYPE_SCROLL_INSENSITIVE);
if (JDBCVersion > 2 || srs == true)
{
    // scroll, baby, scroll!
}
```

スクロール可能な ResultSet を要求する

スクロール機能をサポートする、とドライバーが答えたとすると (サポートしていない場合には新しいドライバーまたはデータベースが必要です)、スクロール可能な `ResultSet` を要求するためには、`Connection.createStatement()` の呼び出しに 2 つの引数を渡します (リスト 4)。

リスト 4. スクロール機能を要求する

```
Statement stmt = con.createStatement(
                       ResultSet.TYPE_SCROLL_INSENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
ResultSet scrollingRS = stmt.executeQuery("SELECT * FROM whatever");
```

`createStatement()` を呼び出す場合、最初の引数と 2 番目の引数はどちらも `int` であるため、特に注意する必要があります。`createStatement()` には (不適切な定数値を含め)、どのような `int` 値でも使用することができます。

最初の引数は `ResultSet` に必要な「スクロール機能」を示し、受け付けられる値は以下の 3 つのうちの 1 つです。

- `ResultSet.TYPE_FORWARD_ONLY:` この値はデフォルトであり、私たちがよく知っていて愛着のある、消火ホース・スタイルのカーソルです。
- `ResultSet.TYPE_SCROLL_INSENSITIVE:` この `ResultSet` によって逆方向にも順方向にも繰り返し操作が可能ですが、データベースの中のデータが変更されても、その変更は `ResultSet` に反映されません。このスクロール可能な `ResultSet` は最も一般的に必要とされるタイプです。
- `ResultSet.TYPE_SCROLL_SENSITIVE:` 作成される `ResultSet` では両方向の繰り返し操作が可能なだけではなく、データベースの中のデータが変更されると、その変更はすぐに ResultSet に反映されます。

2 番目の引数については次のヒントで説明するので、それまでお待ちください。

方向性を持ったスクロール

`Statement` のメソッドを実行して `ResultSet` を取得したら、`previous()` を呼び出せばその ResultSet を逆方向にスクロールすることができます。previous() を呼び出すと、`next()` のように順方向に進むのではなく、逆方向に 1 行戻ります。あるいは、`first()` を呼び出すと `ResultSet` の先頭に戻り、`last()` を呼び出すと `ResultSet` の最後に行きます。これで感覚がつかめると思います。

`relative()` メソッドと `absolute()` メソッドも便利です。relative() メソッドは指定された行数だけ移動し (値が正の場合は順方向、値が負の場合は逆方向)、absolute() メソッドはカーソルの位置にかかわらず、`ResultSet` の中の指定された行まで移動します。もちろん、現在の行番号は `getRow()` で取得することができます。

特定の方向へのスクロールを大量に行う予定がある場合には、`setFetchDirection()` を呼び出して方向を指定することで、`ResultSet` を補助することができます。(`ResultSet` はどちらにスク

ロールする場合も動作しますが、事前に方向がわかっているとデータの取得を最適化することができます。)

3. 更新可能な ResultSet

JDBC は両方向にスクロール可能な `ResultSet` をサポートしているだけではなく、`ResultSet` のインプレース更新をすることもできます。つまり、新しい SQL 文を作成し、データベースに現在保存されている値を変更する代わりに、`ResultSet` の中に保持されている値を変更するだけで、その値は自動的にデータベースに送信され、該当する行の該当する列の値が変更されます。

更新可能な `ResultSet` を要求するプロセスは、スクロール可能な `ResultSet` を要求するプロセスと同様です。実際、ここで `createStatement()` に対する 2 番目の引数を使います。2 番目の引数として `ResultSet.CONCUR_READ_ONLY` を指定する代わりに、`ResultSet.CONCUR_UPDATEABLE` を送信します (リスト 5)。

リスト 5. 更新可能な ResultSet を要求する

```
Statement stmt = con.createStatement(
                      ResultSet.TYPE_SCROLL_INSENSITIVE,
                      ResultSet.CONCUR_UPDATEABLE);
ResultSet scrollingRS = stmt.executeQuery("SELECT * FROM whatever");
```

更新可能なカーソル (このカーソルも JDBC 仕様の機能であり、であり、「実際の」データベースの大部分がサポートするようになるはずです) をドライバーがサポートしている場合には、`ResultSet` の中にある任意の値を更新することができます。それには、その値の行までナビゲートし、その値に対して `update...()` メソッドの 1 つを呼び出します (リスト 6)。ResultSet に対する `get...()` メソッドと同様、`ResultSet` の実際の列の型に合わせて `update...()` は置き換えられます。つまり「`PRICE`」という浮動小数点型の列を変更するためには、`updateFloat("PRICE")` を呼び出します。ただし、その呼び出しによって更新されるのは `ResultSet` の中の値のみです。その値をデータベースに書き込むためには、`updateRow()` を呼び出します。ユーザーが価格 (price) の変更に関して気が変わった場合には、`cancelRowUpdates()` を呼び出すと、途中段階の更新がすべて中止されます。

リスト 6. より良い方法

```
Statement stmt = con.createStatement(
                      ResultSet.TYPE_SCROLL_INSENSITIVE,
                      ResultSet.CONCUR_UPDATEABLE);
ResultSet scrollingRS =
    stmt.executeQuery("SELECT * FROM lineitem WHERE id=1");
scrollingRS.first();
scrollingRS.udpateFloat("PRICE", 121.45f);
// ...
if (userSaidOK)
    scrollingRS.updateRow();
else
    scrollingRS.cancelRowUpdates();
```

JDBC 4.0 は単なる更新以上の機能をサポートしています。まったく新しい行を追加したい場合、新しい `Statement` を作成して `INSERT` を実行する代わりに、`moveToInsertRow()` を呼び出し、各列に対して `update...()` を呼び出し、それから `insertRow()` を呼び出すと、新しい行の追加は完了です。列の値が指定されていない場合には、値は `SQL NULL` と見なされます (この場合、データ

ベース・スキーマによってその列に NULL が許可されていない場合には、SQLException が発生するかもしれません)。

当然ですが、ResultSet が行の更新をサポートしている場合には、ResultSet は deleteRow() による行の削除もサポートする必要があります。

また、忘れないうちに言っておきますが、こうしたスクロール機能や更新機能は PreparedStatement の場合にも同じように当てはまります (上記のパラメーターを prepareStatement() メソッドに渡します)。通常の Statement は常に SQL インジェクション攻撃を受ける危険があるため、PreparedStatement を使った方がはるかに適切です。

4. Rowset

こうしたすべての機能が 10 年近くも JDBC に備わっていたのなら、なぜ大部分の開発者は相変わらず順方向スクロールの ResultSet や接続のないアクセスに固執しているのでしょう。

その主な理由はスケーラビリティーです。データベース接続を最低限にとどめることは、インターネットによって企業の Web サイトに接続される膨大な数のユーザーをサポートする上で非常に重要です。通常、ResultSet のスクロールや更新にはネットワーク接続を開いたままにする必要があるため、多くの開発者はスクロールや更新を使おうとしません (あるいは使うことができません)。

幸いなことに、データベース接続を開いたままにしなくても、ResultSet で行える多くのことを行うための別の手段があります。

Rowset の概念は、基本的に ResultSet と同じですが、Rowset の場合は、接続モデルか非接続モデルのいずれかを使用することができます。Rowset を使用するために必要なことは、Rowset を作成して ResultSet を参照させ、Rowset へデータを追加することです。データの追加が終了したら ResultSet と同じように使えばよいのです (リスト 7)。

リスト 7. Rowset によって ResultSet を置き換える

```
Statement stmt = con.createStatement(
                      ResultSet.TYPE_SCROLL_INSENSITIVE,
                      ResultSet.CONCUR_UPDATEABLE);
ResultSet scrollingRS = stmt.executeQuery("SELECT * FROM whatever");
if (wantsConnected)
    JdbcRowSet rs = new JdbcRowSet(scrollingRS); // connected
else
    CachedRowSet crs = new CachedRowSet(scrollingRS); disconnected
```

JDBC には、Rowset インターフェースの「実装」(つまり拡張されたインターフェース) が 5 つあります。JdbcRowSet は接続型の Rowset 実装であり、以下に挙げる他の 4 つは非接続型です。

- CachedRowSet は単なる非接続型 Rowset です。
- WebRowSet は CachedRowSet のサブクラスであり、キャッシュされた結果を XML に変換することと、その逆変換をすることができます。
- JoinRowSet は、データベースに接続しなくても SQL JOIN と同じことができる WebRowSet です。

- `FilteredRowSet` は、データベースに接続しなくてもデータをさらにフィルタリングして返すこともできる `WebRowSet` です。

`Rowset` には JavaBeans が多用されています。つまり `Rowset` はリスナー・スタイルのイベントをサポートしているため、必要に応じて `Rowset` への変更をキャッチしたり検証したり、あるいは変更に基づいてアクションを実行したりすることができます。実際、`Rowset` が `Username`、`Password`、`URL`、`DatasourceName` といったプロパティー・セットを持つ (つまり Rowset が `DriverManager.getConnection()` を使って接続を作成する) か、あるいは `Datasource` プロパティー・セット (おそらく JNDI を使って取得されたもの) を持つ場合には、Rowset によってデータベースに対する全てのアクションを管理することさえできます。すると、実行すべき SQL を `Command` プロパティーの中で指定し、`execute()` を呼び出し、その結果に対して作業を開始することができ、他の操作は必要ありません。

`Rowset` の実装は通常、JDBC ドライバーによって提供されているため、実際の名前やパッケージは使用される JDBC ドライバーに依存します。`Rowset` の実装が標準的なディストリビューションの一部となってからしばらく経っているので、`...RowsetImpl()` を作成するだけで、必要なことを行えるはずです (万が一、ドライバーが Rowset の実装を提供していない場合は、Sun がリファレンス実装を提供しています。「関連トピック」のリンクを参照してください)。

5. バッチ更新

`Rowset` は便利ですが、場合によると、必要なことをすべて満たすことはできず、やはり皆さんが直接 SQL 文を作成しなければならない場合があります。そうした場合 (特に処理を大量に抱えている場合には)、バッチ更新を行える機能が役立ちます。バッチ更新では、ネットワークを 1 度往復するだけで、データベースに対して複数の SQL 文を実行することができます。

JDBC ドライバーがバッチ更新をサポートしているかどうかを判断するためには、`DatabaseMetaData.supportsBatchUpdates()` を呼び出すことで、サポートの有無を示すブール値が返されるようにします。バッチ更新がサポートされている場合 (`SELECT` 以外の文が使われていることでもわかります)、SQL 文をキューイングして一気に実行します (リスト 8)。

リスト 8. データベースをバッチ更新する

```
conn.setAutoCommit(false);

PreparedStatement pstmt = conn.prepareStatement("INSERT INTO lineitems VALUES(?,?,?,?)");
pstmt.setInt(1, 1);
pstmt.setString(2, "52919-49278");
pstmt.setFloat(3, 49.99);
pstmt.setBoolean(4, true);
pstmt.addBatch();

// rinse, lather, repeat

int[] updateCount = pstmt.executeBatch();
conn.commit();
conn.setAutoCommit(true);
```

`setAutoCommit()` の呼び出しが必要な理由は、ドライバーはデフォルトで、提供されるすべての文をコミットしようとするためです。それを除くと、このコードの他の部分は非常に単純で

す。`Statement` または `PreparedStatement` によって通常の SQL 操作を行いますが、`execute()` を呼び出す代わりに、`executeBatch()` を呼び出します。executeBatch() は SQL 文を即座に送信せずにキューイングします。

すべての文の実行準備が整うと、それらの文をすべて、`executeBatch()` によってデータベースに対して実行します。すると一連の整数値が返され、それぞれの値は `executeUpdate()` が使われた場合と同じ結果を保持しています。

バッチの中の 1 つの文が失敗した場合や、ドライバーがバッチ更新をサポートしていなかったり、バッチの中の 1 つの文によって `ResultSet` が返されたりした場合、ドライバーは `BatchUpdateException` をスローします。場合によると、例外がスローされた後、ドライバーはそれらの文を実行しようと試み続けていたかもしれません。JDBC の仕様では具体的な動作を規定していません。そのため、ドライバーの動作を正確に理解できるように、事前にドライバーを試してみるようお勧めします (しかしもちろん、皆さんはユニット・テストを実行するはずなので、問題になるよりもはるかに以前に、このエラーを発見できるはずです。)

まとめ

JDBC API は、Java 開発に不可欠な要素として、すべての Java 開発者が十分に理解する必要のあるものです。奇妙なことに、ほとんどの開発者は、何年にもわたって行われてきた JDBC API の機能強化の最新情報を知りません。その結果、この記事で説明したような、時間を節約する工夫を活用できずにいます。

もちろん、JDBC の新しい機能を使うかどうかは皆さん次第です。考慮すべき重要な点は、皆さんのシステムのスケーラビリティーです。システムのスケーリングに対する要求が高ければ高いほど、データベースの使用制限は厳しくなり、従ってデータベースに対するネットワーク・トラフィックを一層減らす必要があります。そうした場合には、`Rowset`、スカラー呼び出し、バッチ更新が有効です。それ以外の場合には、スクロール可能な `ResultSet` と更新可能な `ResultSet` を試し (`ResultSet` は `Rowset` ほどメモリーを使用しません)、スケーラビリティーへの影響を調べてみてください。結果は皆さんが想像するほど悪くはないはずです。

著者について

Ted Neward

Ted Neward has written over 250 articles and a dozen books across many different technologies, including .NET, iOS, Java, Android, and JavaScript. He resides in Seattle with his wife, two kids, nine laptops, fourteen mobile devices, and two cats. Email him if you're interested in having him or his company work with you.

Alex Theedom

May 2017

© Copyright IBM Corporation 2010, 2017
(www.ibm.com/legal/copytrade.shtml)
商標
(www.ibm.com/developerworks/jp/ibm/trademarks/)