Javaアプリケーション開発ガイド
一般編

第2版 2011年9月

富士通株式会社

FUJITSU

Copyright 2011 FUJITSU LIMITED

まえがき

■本書の目的

本書は、JavaアプリケーションからSymfoware Serverを利用する方法について、説明しています。

以下の内容が理解できることを目標としています。

- JDBCを用いてJavaアプリケーションからSymfoware Serverに接続する方法
  (データソースを用いてSymfoware Serverに接続する方法)
- JavaアプリケーションからSymfoware Serverにアクセスして、SQLを実行する方法
  (PreparedStatementを用いてSQLを実行する方法とResultSetの様々な使用方法)

■本書の読者

本書は、Symfoware Serverにアクセスするアプリケーションを開発される方に読んでいただくように書かれています。

本書を読むには、以下の知識が必要です。

- Symfoware Serverに関する知識
- SQLに関する知識
- Javaに関する一般知識
- JDBCに関する一般知識

※“Javaアプリケーション開発ガイド 入門編”の内容を理解できることを前提としています。

Copyright 2011 FUJITSU LIMITED

目次

Copyright 2011 FUJITSU LIMITED

Copyright 2011 FUJITSU LIMITED

第1章　JDBCドライバの種類と選択

Copyright 2011 FUJITSU LIMITED

1.1　JDBCドライバがサポートするAPI

JDBC を用いてデータベースを利用する手順は、どのデータベースソフトを使用している場合でも同じです。ただし、JDBC の規格のうち、どの API をサポートしているかは、データベースソフトによって異なります。また、データベースソフトによっては、独自に JDBC の仕様を拡張し、そのデータベースソフト固有の機能を追加している場合もあります。

Symfoware Server の JDBC ドライバがサポートしている API の一覧が、マニュアル“アプリケーション開発ガイド(JDBC 編)”に掲載されています。また、“Java API リファレンス”が製品に付属しています。クラスやメソッドの詳細は、“Java API リファレンス”に記載されています。これらを参照して、アプリケーションを作成してください。

なお、Symfoware Server では Java のアプリケーションで埋込み SQL を使用することはできません。Java のアプリケーションから Symfoware Server を利用する場合は、JDBC を用いてください。

Copyright 2011 FUJITSU LIMITED

1.2 どのJDBCドライバを使用すればよいか

JDBC のドライバには、タイプ 1 からタイプ 4 までの 4 つのタイプがあります。このうち Symfoware Server はタイプ 2 のドライバをサポートしています。

タイプ 2 は、JDBC ドライバが Symfoware Server のクライアントライブラリを利用してデータベースにアクセスするタイプです。

［補足］Symfoware Server V9 以前のバージョンで RDA-SV を使用する場合

Symfoware Server V9 以前のバージョンでは、クライアントとデータベースサーバとの通信に RDA-SV というソフトウェアを使用する方式がサポートされていました。RDA-SV 連携を使用する場合には、タイプ 4 の JDBC ドライバを使用する必要がありました。

Symfoware Server V9 以前は、JDBC のタイプを以下のように使い分けます。

- タイプ 2
  ローカルアクセスまたは RDB2_TCP 連携で Symfoware Server に接続する場合に使用されます。
- タイプ 4
  RDA-SV 連携で Symfoware Server に接続する場合に使用されます。

実際には、どちらのタイプのドライバを使用するかを選択するのではなく、接続形態を選択します。接続形態を選択すると、それに応じたドライバのタイプが使用されます。

接続形態を選択するには、以下のようにします。

- Symfoware Server に添付されているデータソース登録ツールの画面に表示されている中から選択する。
- DriverManager の getConnection メソッドに指定する URL で指定する。

Copyright 2011 FUJITSU LIMITED

1.3 JDBCの規格のバージョン

JDBC のドライバのタイプとは別に、JDBC 自体の規約のバージョンがあります。Symfoware Server V9 までは、JDBC2.X に対応しています。Symfoware Server V10 以降は、JDBC3.X と JDBC4.X にも対応しています。そのため、V10 以降では JDBC ドライバとして以下の 3 種類が提供されています。

- fjsymjdbc2.jar：JDBC2.X 対応ドライバモジュール
- fjsymjdbc3.jar：JDBC3.X 対応ドライバモジュール
- fjsymjdbc4.jar：JDBC4.X 対応ドライバモジュール

Symfoware Server V10 では、使用するドライバモジュールを選択して、そのパスを CLASSPATH に設定してください。通常は、最も新しい規格である JDBC4.X に対応したモジュールを使用します。

Symfoware Server V9 までは、JDBC2.X 対応のドライバモジュールのみです。

使用する JDBC のバージョンによって、JDK のバージョンも考慮する必要があります。新しい規約の JDBC を使用するには、新しいバージョンの JDK が必要です。詳細は、マニュアル“アプリケーション開発ガイド(JDBC ドライバ編)”を参照してください。

Copyright 2011 FUJITSU LIMITED

第2章　データソースを用いた接続

Copyright 2011 FUJITSU LIMITED

2.1 Connectionオブジェクトを作成する2つの方法

Java プログラムから Symfoware Server を利用するためには、まずデータベースに接続する必要があります。データベースに接続するには、JDBC の Connection オブジェクトを作成します。

Connection オブジェクトを作成する方法には、以下の 2 つがあります。

- DriverManager を使用する方法
- DataSource を使用する方法

DriverManager を使用する方法は、入門編で説明しました。ここでは DataSource を使用する方法について説明します。

Copyright 2011 FUJITSU LIMITED

2.2 データソース

データソースとは、データを格納する機構を一般的に表したものです。ここではもちろんSymfoware Server のデータベースを表します。

DataSource オブジェクトはデータソースを表すオブジェクトです。Java アプリケーションの中では、DataSource オブジェクトは、データベースへの接続を表す Connection オブジェクトを作成するために使用します。

DataSource オブジェクトを用いる方法には、以下の利点があります。

- JDBC ドライバに関する情報をアプリケーションの中で指定する必要がなくなります。
  どのドライバを用いてどのデータベースに誰がアクセスするのか、といった情報は、データソース情報としてアプリケーションの外部で管理されます。アプリケーション内では、外部で管理されている情報のキーとなるデータソース名を指定するだけです。これによって、接続先の変更などに柔軟に対応することができるようになります。
- コネクションプールを使うことができるようになります。
  データベースへの接続と切断は比較的コストの高い処理です。接続をキャッシュしておくことで、アプリケーションの性能向上が期待できます。

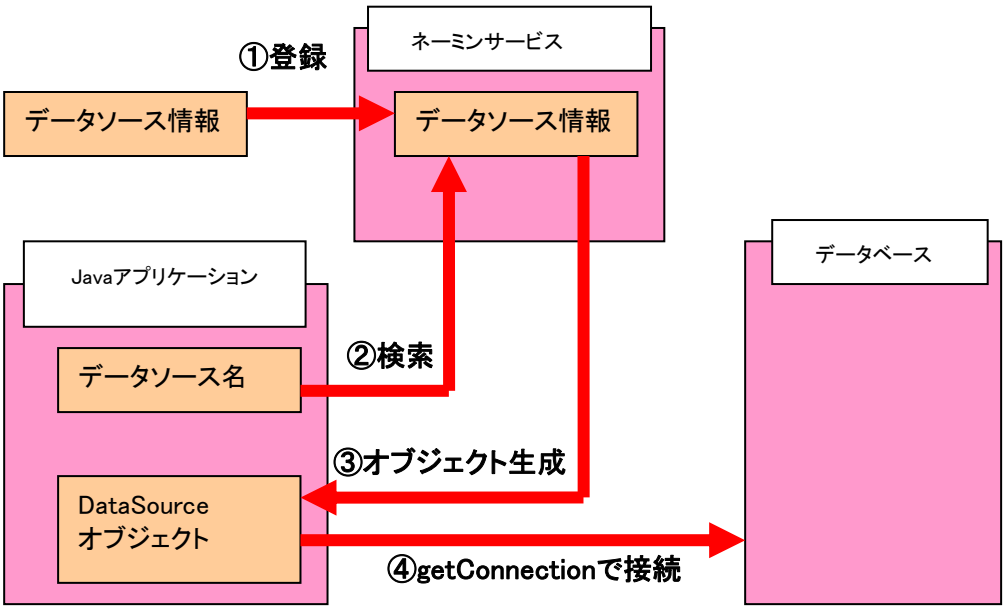
アプリケーションからはデータソース名を指定して DataSource オブジェクトを作成します。その DataSource オブジェクトを用いて Connection オブジェクトを作成します。Connection オブジェクトを作成することによって、データベースに接続します。

Copyright 2011 FUJITSU LIMITED

2.3 ネーミングサービス

データソースに関する情報は、アプリケーションとは別個に管理します。そのためにネーミングサービスを使用します。ネーミングサービスは、Java アプリケーションともデータベースとも別に動作しているプログラムで、オブジェクトと名前の対応を管理する役目を持っています。

データソースの情報は、ネーミングサービスに予め登録しておきます。Java には、アプリケーションがネーミングサービスを利用するための API が用意されています。ネーミングサービスを利用するための API を JNDI と呼びます。Java アプリケーションは、データソース名をキーにして JNDI の API を用いることで、ネーミングサービスに登録されているデータソースを利用できます。



Copyright 2011 FUJITSU LIMITED

ネーミングサービスとしては、AP サーバである Interstage が提供している機能を使用できます。ネーミングサービスへのデータソースの登録も、Interstage の機能である管理コンソールから行うことができます。

詳細は Interstage のマニュアルを参照してください。

Symfoware Server には専用のネーミングサービスとデータソース登録ツールが付属しています。Interstage を利用する場合は、Interstage のネーミングサービスを使用することを推奨します。

Copyright 2011 FUJITSU LIMITED

2.4 Symfoware Serverのデータソース登録ツール

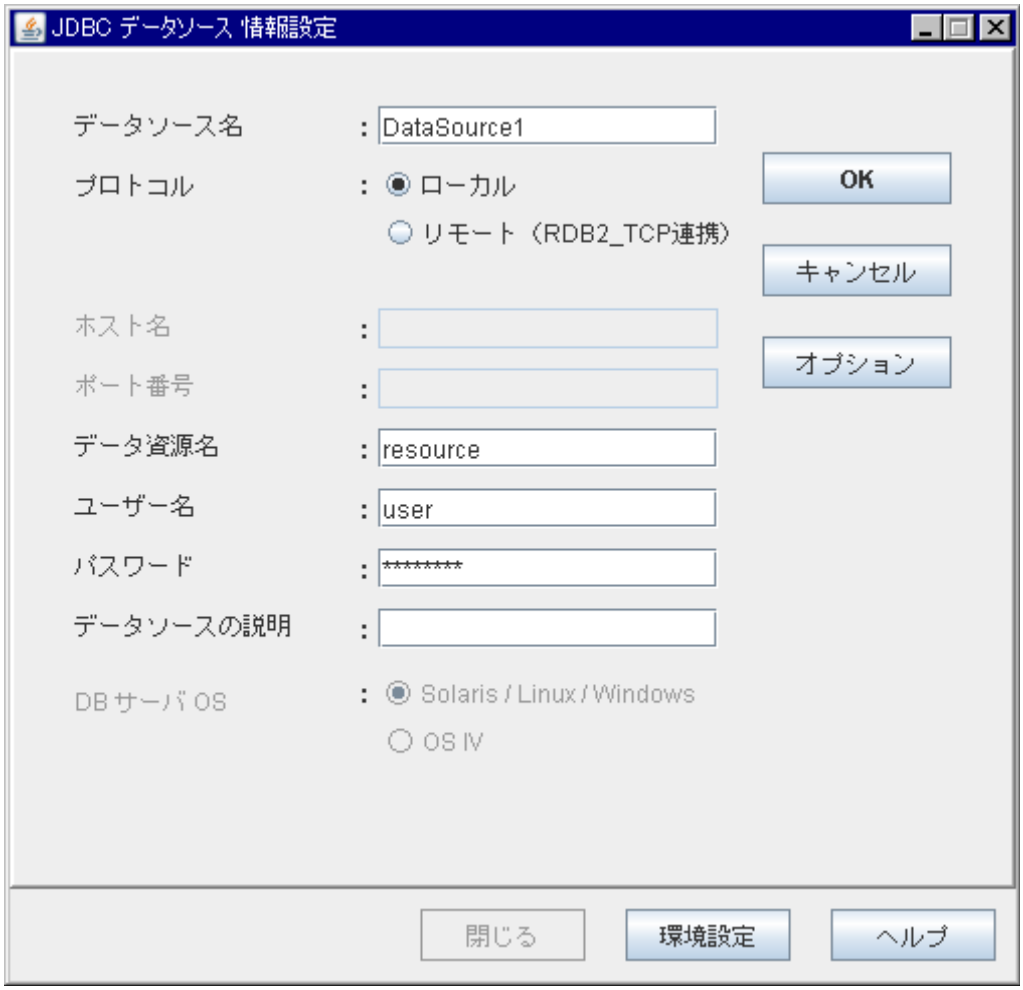
Symfoware Server には、Symfoware Server 専用のデータソース登録ツールとネーミングサービスが付属しています。

データソース登録ツールを利用することで、GUI 画面上でデータソースの情報を登録することができます。

下記の画面は、Symfoware Server に付属しているデータソース登録ツールの画面です。この画面上でデータソースの情報を設定します。

登録した情報は Symfoware Server に付属しているネーミングサービスによって管理されます。このネーミングサービスには、Java アプリケーションから JNDI を用いてアクセスできます。

Copyright 2011 FUJITSU LIMITED



詳細な手順は、マニュアル“アプリケーション開発ガイド(JDBC 編)”を参照してください。

Copyright 2011 FUJITSU LIMITED

2.5 DataSourceを使用して接続する方法

いったんデータソースを登録できたなら、Java アプリケーション内からネーミングサービスにアクセスすることによって、登録しておいたデータソースを利用することができます。

データソースは、DataSource オブジェクトとして表されます。DataSource オブジェクトの getConnection メソッドを呼び出すことで、Connection オブジェクトを作成することができます。

Connection オブジェクトが作成できたら、DriverManager を用いて作成した Connection オブジェクトの場合と同様に、Symfoware Server に処理を依頼することができます。

■Interstage を利用する場合

Interstage のネーミングサービスを利用する場合は、あらかじめ Interstage の管理コンソールからデータソースを登録しておく必要があります。登録されているデータソースを Java アプリケーションから利用するには、以下のようにします。

(例)

```
// ネーミングサービスの InitialContext を作成する。
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient");

// InitialContext を利用して DataSource を作成する。
DataSource ds = (DataSource)ctx.lookup("jdbc/DataSource1");

// DataSource を利用して Connection を作成する。
```


Copyright 2011 FUJITSU LIMITED

```
Connection con = ds.getConnection();
```

InitialContext オブジェクトの lookup メソッドで指定している「DataSource1」がデータソース名です。データソースを登録するときにつけた名前を指定します。

InitialContext オブジェクトは、ネーミングサービスの情報の起点を表すオブジェクトです。これはどのネーミングサービスを利用するのかを指定しています。Interstage の場合、利用形態によっていくつかの種類があります。上記の例は J2EE クライアントとして実行する場合です。詳細は、“Interstage Application Server JSEE ユーザーズガイド”を参照してください。

■Symfoware Server 専用のネーミングサービスを利用する場合

Symfoware Server 専用のツールを用いて登録したデータソースを使用する場合は、以下のようにします。

(例)

```
// ネーミングサービスの InitialContext を作成する。
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.fujitsu.symfoware.jdbc2.jndisp.SYMContextFactory");
env.put(Context.PROVIDER_URL,"SYM://myhost:26600");
InitialContext ctx = new InitialContext(env);

// InitialContext を利用して DataSource を作成する。
DataSource ds = (DataSource)ctx.lookup("jdbc/DataSource1");
```


Copyright 2011 FUJITSU LIMITED

```
// DataSource を利用して Connection を作成する。
Connection con = ds.getConnection();
```

InitialContext オブジェクトの lookup メソッドで指定している「DataSource1」がデータソース名です。データソースを登録するときにつけた名前を指定します。

Symfoware Server のデータソース登録ツールをネーミングサービスとして使用することを指定するには、INITIAL_CONTEXT_FACTORY に以下の値を指定します。

com.fujitsu.symfoware.jdbc2.jndisp.SYMContextFactory

Symfoware Server 専用のネーミングサービスが使用するポート番号は、デフォルトで 26600（V9 までは 10326）です。これは変更することもできます。変更する方法は、マニュアル“アプリケーション開発ガイド(JDBC 編)”を参照してください。

Copyright 2011 FUJITSU LIMITED

2.6 Connection Managerを使用する場合の接続先

Symfoware Server Standard Edition、Symfoware Server Enterprise Edition、または、Symfoware Server Enterprise Extended Edition の Connection Manager を使うと、ロードシェアのクラスタシステムでのロードバランシング機能や、ホットスタンバイシステムでのコネクションリカバリ機能などを利用することができます。

Connection Manager を利用する場合は、接続先のデータベースに関する情報を APC 動作環境ファイルの中で、SQL サーバ情報として記述しておきます。

Connection Manager を利用する環境で、Java のアプリケーションから Symfoware Server を利用する場合、接続先のデータ資源名には、APC 動作環境ファイルに記述した SQL サーバ名を指定します。

また、Connection Manager を利用する場合、アプリケーションと Symfoware Server が異なるコンピュータで動作していても、ローカル接続を指定します。

なお、マルチ RDB の場合でも RDB システム名はつけません。

Connection Manager を利用する場合の接続先は、以下のように指定します。

- DataSource を使う場合
  ネーミングサービスにデータソースを登録する時に、［JDBC データソース情報設定］画面の［データ資源名］に、SQL サーバ名を指定してください。
- DriverManager を使う場合
  アプリケーションで指定する URL のデータ資源名に SQL サーバ名を指定してください。
  (例)

```
DriverManager.getConnection("jdbc.symfold:///SS","UID","PWD")
```

  ここで、「SS」が SQL サーバ名です。APC 動作環境ファイルには、「SS」という名前の SQL サーバの情報が記載されている必要があります。

詳細は、マニュアル“Connection Manager ユーザーズガイド”の“5.1 コネクションのあて先制御”を参照してください。

Copyright 2011 FUJITSU LIMITED

第3章　SQL文の準備

Copyright 2011 FUJITSU LIMITED

3.1 SQL文を表すオブジェクト

データベースに処理を依頼するには、SQL 文を用います。Java アプリケーションで SQL 文を使用するには、Statement オブジェクトか PreparedStatement オブジェクトを作成します。

Statement オブジェクトや PreparedStatement オブジェクトには、SQL 文を実行するためのメソッドが用意されています。

SELECT 文を実行する場合は、executeQuery メソッドを用います。executeQuery メソッドを実行すると、検索結果を表す ResultSet オブジェクトが作成されます。

INSERT 文、UPDATE 文、DELETE 文などのデータ更新系の SQL 文を実行する場合は、executeUpdate メソッドを用います。

［補足］execute メソッド

Statement オブジェクトや PreparedStatement オブジェクトには、SQL 文を実行するためのメソッドとして、execute メソッドというものも用意されています。これは実行される SQL 文が参照処理なのか更新処理なのか分からない場合や、1 個の SQL 文で複数の ResultSet が作成されるようなケースで使用します。

通常は、execute メソッドを使用することはありません。

Copyright 2011 FUJITSU LIMITED

3.2 PreparedStatementクラスを使用する利点

PreparedStatement クラスには、Statement クラスに比べて主に以下のような利点があります。

- SQL 文の中に動的に変更できるパラメーターを記述することができます。
  SQL 文で指定する検索条件の値や格納するデータの値などを、「?」と記述しておき、SQL 文を実行するときに値を設定することができます。
  そのため、アプリケーションの処理を柔軟に組み立てることができます。また、SQL インジェクションを用いた不正アクセスを防ぐことにもつながり、セキュリティの向上に役立ちます。
- 同じ SQL 文を何度も実行する場合、SQL 文の解析は最初の 1 回だけで済みます。
  Statement クラスを使用する場合は、同じ SQL 文を実行する場合でも、実行するたびに SQL 文の解析が行われます。
  そのため、同じ SQL 文を何度も実行する場合には、アプリケーションの性能が向上します。

Copyright 2011 FUJITSU LIMITED

3.3 SQL文の中でパラメーターを使う

PreparedStatement オブジェクトを用いると、SQL 文の中でパラメーターを使用することができます。

SQL 文の中でパラメーターを使用する部分には、パラメーターを設定するためのプレースホルダとして「?」を記述します。「?」の部分には、SQL 文を実行する前に値を設定します。値の設定は、PreparedStatement オブジェクトの setXXXX メソッドを用いて行います。

(例)

```
// PreparedStatement オブジェクトを作成する。
PreparedStatement pstmt = con.prepareStatement(
        "SELECT ID,NAME FROM GENERAL.EMPLOYEE
        WHERE ID=?");

// 検索条件として ID 列の値を設定する。
pstmt.setInt(1, 100);

// SELECT 文を実行する。
ResultSet rs = pstmt.executeQuery();
```

ここで、「setInt(1, 100)」は、「1 番目のパラメーターに整数の 100 という値を設定する」ことを意味します。

値を設定するメソッドは、データの型ごとに用意されています。ここでいうデータの型とは、データベースの列の型ではなく、Java アプリケーション内で使用する Java のデータ型です。

Copyright 2011 FUJITSU LIMITED

もし、SQL 文中に複数のパラメーターがある場合は、前から順に 1 番目、2 番目という番号で指定します。

(例)

```
// PreparedStatement オブジェクトを作成する。
PreparedStatement pstmt = con.prepareStatement(
        "INSERT INTO GENERAL.EMPLOYEE(
        ID,NAME) VALUES(?,?)");

// 1 番目のパラメーターに 4 という数値を設定する。
pstmt.setInt(1,4);

// 2 番目のパラメーターに"monkey"という文字列を設定する。
pstmt.setString(2,"monkey");

// INSERT 文を実行する。
pstmt.executeUpdate();
```

上記の例は、最終的に以下の SQL 文を実行します。

```
INSERT INTO GENERAL.EMPLOYEE(ID,NAME) VALUES(4,'monkey')
```


Copyright 2011 FUJITSU LIMITED

3.4 同じSQL文を何度も使う

PreparedStatement オブジェクトを用いると、同じ SQL 文を何度も使いまわすことができます。

何度も同じ SQL 文を実行する場合、SQL 文の解析は最初の 1 回だけで済みます。そのため、毎回 SQL 文の解析を行う Statement オブジェクトを使用する場合に比べて、アプリケーションの性能が向上します。

(例)

```
// PreparedStatement オブジェクトを作成する。
PreparedStatement pstmt = con.prepareStatement(
        "INSERT INTO GENERAL.EMPLOYEE(
        ID,NAME) VALUES(?,?)");

for( int iID=0 ; iID<100 ; iID++ )
{
        // 1 番目のパラメーターに iID の値を設定する。
        pstmt.setInt(1,iID);
        // 2 番目のパラメーターに配列 sNAME に格納されている文字列を設定する。
        pstmt.setString(2,sNAME[iID]);
        // INSERT 文を実行する。
        pstmt.executeUpdate();
}
```


Copyright 2011 FUJITSU LIMITED

この例では、ループの中で同じ SQL 文をパラメーターを変更しながら使いまわしています。SQL 文の解析は、最初に prepareStatement メソッドを実行したときにのみ行われます。

この処理を Statement オブジェクトを用いて行うと、以下のようになります。この例では、ループ内で executeUpdate メソッドを実行するたびに、SQL 文の解析が行われます。

(例)

```
// Statement オブジェクトを作成する。
Statement stmt = con.createStatement();


for( int iID=0 ; iID<100 ; iID++ )
{
    // SQL 文を組み立てる。
    String sql = "INSERT INTO GENERAL.EMPLOYEE(ID,NAME) VALUES("
        + String.valueOf(iID) + ",'" + sNAME[iID] + "')";


    // INSERT 文を実行する。
    stmt.executeUpdate(sql);
}
```


Copyright 2011 FUJITSU LIMITED

第4章　データベースの検索

Copyright 2011 FUJITSU LIMITED

4.1 検索の実行

PreparedStatement クラスを用いて、SELECT 文の検索条件にパラメーターを用いて、何度も同じ検索を行う例を以下に示します。

SELECT 文の解析は prepareStatement メソッドを実行したときに 1 回だけ行われ、その後のループでは同じ SELECT 文を使いまわしています。

(例)

```
// Symfoware Server のネーミングサービスを利用する準備を行う。
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.fujitsu.symfoware.jdbc2.jndisp.SYMContextFactory");
env.put(Context.PROVIDER_URL,"SYM://myhost:26600");
InitialContext ctx = new InitialContext(env);

// InitialContext を利用して DataSource を作成する。
DataSource ds = (DataSource)ctx.lookup("jdbc/DataSource1");

// DataSource を利用してデータベースに接続する。
Connection con = ds.getConnection();

// 自動コミットを解除する。
con.setAutoCommit(false);
```


Copyright 2011 FUJITSU LIMITED

```
// PreparedStatement オブジェクトを作成する。
PreparedStatement pstmt = con.prepareStatement(
        "SELECT ID,NAME FROM GENERAL.EMPLOYEE WHERE ID=?");

// SELECT 文を実行して、ResultSet オブジェクトを得る。
for( int loopCount=0 ; loopCount <100 ; loopCount ++ )
{
        //  パラメーターに検索条件を設定する。
        pstmt.setInt(1, loopCount);

        // SELECT 文を実行する。
        ResultSet rs = pstmt.executeQuery();

        // ResultSet からデータを取り出す。
        while (rs.next())
        {
                // ID と NAME の値を取り出す。
                int iID = rs.getInt(1);
                String sName = rs.getString(2);

                //  取り出した値を表示させる。
                System.out.println("ID = " + iID);
                System.out.println("NAME = " + sName);
```


Copyright 2011 FUJITSU LIMITED

```
            }

            // ResultSet オブジェクトを破棄する。
            rs.close();
        }

        // PreparedStatement オブジェクトを破棄する。
        pstmt.close();

        // SELECT 文をコミットする。
        con.commit();

        //  データベースから切断する。
        con.close();
```

［補足］

PreparedStatement には利点がありますが、常に利点があるわけではありません。内容が決まっている SQL 文を 1 回だけ実行する場合など、PreparedStatement を使用する意味がない場合には、Statement を用いた方がプログラムが単純になります。

Copyright 2011 FUJITSU LIMITED

第5章　検索結果の取り出し

Copyright 2011 FUJITSU LIMITED

5.1 ResultSetの基本操作

Statement オブジェクトや PreparedStatement オブジェクトを用いて SELECT 文を実行すると、検索結果の表を表す ResultSet オブジェクトが作成されます。

ResultSet オブジェクトには、getString メソッドや getInt メソッドなど、結果表の列の値を取り出すためのメソッドが用意されています。これらのメソッドを getter メソッドと呼びます。

ResultSet オブジェクトには、結果表の特定の 1 行を参照するためのカーソルが備わっています。getter メソッドを用いることで、カーソルが位置づけられている行から、データを取り出すことができます。

ResultSet オブジェクトから検索結果を取り出す処理は、以下の手順で行います。

(1) カーソルを検索結果の表の中のある行に位置づける。
(2) ResultSet オブジェクトの getter メソッドを用いて、列の値を取り出す。

Copyright 2011 FUJITSU LIMITED

5.2 カーソルを移動させる方法

ResultSet オブジェクトが表す結果表には、行番号がつけられています。最初の行の行番号は 1、2 行目の行番号は 2 になります。

ResultSet オブジェクトが作成された時点では、カーソルは行番号 0 に位置づけられています。0 番目の行は存在しないので、この状態ではデータを取り出すことはできません。

結果表からデータを取り出すには、まずカーソルをどこかの行に移動させる必要があります。

デフォルトの ResultSet では、カーソルは順方向に 1 行ずつ移動させることのみできます。カーソルを移動させるには、next メソッドを実行します。

カーソルが最終行まで達した状態で next メソッドを実行すると、false が返ります。それによってアプリケーションは、結果表からすべてのデータを取り出し終わったことを知ることができます。

Copyright 2011 FUJITSU LIMITED

5.3 カーソルを自由に動かす方法

デフォルトのカーソルのタイプは、TYPE_FORWARD_ONLY です。ResultSet のカーソルには、全部で 3 つのタイプがあります。

- TYPE_FORWARD_ONLY
  デフォルトのカーソルタイプです。next メソッドを用いて、順方向に 1 行ずつカーソルを移動させることのみ可能です。
- TYPE_SCROLL_INSENSITIVE
  自由に移動させることができるカーソルタイプです。
- TYPE_SCROLL_SENSITIVE
  自由に移動させることができるカーソルタイプです。

Symfoware Server ではいずれのタイプを選択しても、ResultSet から取り出されるデータは、ResultSet を作成した時点のものになります。

カーソルのタイプに TYPE_SCROLL_INSENSITIVE または TYPE_SCROLL_SENSITIVE を指定すると、カーソルを自由に動かすことができるようになります。

カーソルのタイプは、createStatement メソッドまたは prepareStatement メソッドを実行するときに、引数で指定します。これらのメソッドの仕様上、カーソルのタイプを単独で指定することはできず、ResultSet の更新可能性の指定とセットで指定する必要があります。

下記の例では、カーソルのタイプに、TYPE_SCROLL_INSENSITIVE を指定しています。

(例)

```
// Statement オブジェクトを作成する。
Statement stmt = con.createStatement(
        java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE,
```


Copyright 2011 FUJITSU LIMITED

```
                java.sql.ResultSet.CONCUR_READ_ONLY);
```


Copyright 2011 FUJITSU LIMITED

5.4 カーソルを動かすためのメソッド

カーソルを移動させるには、ResultSet オブジェクトのメソッドを実行します。メソッドには以下のものがあります。

- absolute メソッド
  カーソルを ResultSet オブジェクト内の指定された行番号の行に移動します。
- afterLast メソッド
  カーソルを ResultSet オブジェクトの最終行の直後に移動します。
- beforeFirst メソッド
  カーソルを ResultSet オブジェクトの先頭行の直前に移動します。つまり、ResultSet オブジェクトを作成したときの初期位置（行番号 0）に移動します。
- first メソッド
  カーソルをこの ResultSet オブジェクト内の先頭行（行番号 1）に移動します。
- last メソッド
  カーソルを ResultSet オブジェクト内の最終行に移動します。
- next メソッド
  カーソルを現在の位置の 1 行下に移動します。
- previous メソッド
  カーソルをこの ResultSet オブジェクト内の前の行に移動します。
- relative メソッド
  カーソルを正または負の相対行数だけ移動します。

Copyright 2011 FUJITSU LIMITED

5.5 ResultSetの保持機能

ResultSet オブジェクトは、トランザクションが終了すると自動的にクローズされます。

しかし、トランザクションが終了したあとで、検索結果を参照したい場合もあります。例えば、以下のようなケースです。

- 検索結果を参照しながら、別の表を更新している。このとき、更新処理をこまめにコミットしたい。更新をコミットすると ResultSet もクローズされてしまう。これを避けるために、ResultSet を保持したい場合。
- 検索結果を順次に画面表示させるなど、検索結果のデータがいつ不要になるか分からない。ResultSet は必要だが、トランザクションは早めに終了させておきたい場合。

このような場合、トランザクション終了後も ResultSet をクローズせずに保持することができます。

ResultSet の保持機能は、Symfoware Server V10 からサポートされています。

ResulSet の保持機能を使用するかどうかは、Connection オブジェクトで設定します。以下のように、Connection オブジェクトの setHoldability メソッドで、ResultSet を保持することを宣言します。

(例)

```
// データベースに接続する。
Class.forName("com.fujitsu.symfoware.jdbc.SYMDriver");
Connection con = DriverManager.getConnection(
        "jdbc:symfold:///COMPANY","UID", "PWD");
```


Copyright 2011 FUJITSU LIMITED

```
// 自動コミットを解除し、ResultSet 保持機能の使用を宣言する。
con.setAutoCommit(false);
con.setHoldability(ResultSet.HOLD_CURSORS_OVER_COMMIT);

// 検索を実行し、トランザクションをコミットする。
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
        "SELECT ID,NAME FROM GENERAL.EMPLOYEE ");
con.commit();

// トランザクション終了後に、ResultSet を参照する。
while (rs.next())
{
        /* ID と NAME の値を取り出す。*/
        int iID = rs.getInt(1);
        String sName = rs.getString(2);

        /* 取り出した値を表示させる。*/
        System.out.println("ID = " + iID);
        System.out.println("NAME = " + sName);
}
rs.close();
```


Copyright 2011 FUJITSU LIMITED

setHoldability メソッドで、以下のいずれかを指定できます。

- CLOSE_CURSORS_AT_COMMIT
  現在のトランザクションがコミットされたときに、ResultSet オブジェクトがクローズされることを示します。デフォルトでは CLOSE_CURSORS_AT_COMMIT が指定されたことになります。
- HOLD_CURSORS_OVER_COMMIT
  現在のトランザクションがコミットされたときに、ResultSet オブジェクトがクローズされずに保持されることを示します。

Copyright 2011 FUJITSU LIMITED

5.6 ResultSetの保持機能を使う上での注意

ResultSet 保持機能を使用する場合は、以下の点に注意してください。

- ResultSet 保持機能を使用する場合、イルシデーションロックの機能は使用できません。イルシデーションロックを明示的に使用すると、構文エラーになります。
- Symfoware Server Standard Edition、Symfoware Server Enterprise Edition、または、Symfoware Server Enterprise Extended Edition において、動作環境ファイルや ctuneparam で DSO_LOCK パラメーターを設定すると、イルシデーションロックが使用できなくなります。
  ResultSet 保持機能は暗黙的にイルシデーションロックを使用します。そのため、イルシデーションロックが使用できないような設定にすると、検索処理がエラーになります。

また、ResultSet 保持機能と、ResultSet を更新可能とする機能は、同時には使用できません。

［補足］ Symfoware Server の排他制御

データベースでは同時に複数のユーザーが同じデータにアクセスすることがあります。このとき、データの検索や更新が同時に行われても、ユーザーにとって矛盾のないデータになっている必要があります。

そのために、Symfoware Server はロックを用いた排他制御を行っています。アクセス中のデータをロックすることによって、同じデータが同時に更新されることを防いだり、参照中のデータが別のユーザーによって不用意に変更されることを防いだりします。

データをロックすることによってデータの矛盾を防ぐことができますが、完全にロックしてしまうと、複数の処理を同時に実行することができなくなってしまいます。そこで、ロックの強さやロックしている期間を調整することができるようになっています。

Copyright 2011 FUJITSU LIMITED

Symfoware Server では排他制御の調整は、以下の方法で行うことができます。

- 独立性水準
  トランザクション単位にロックのかかりかたを調整できます。
  独立性水準を設定するには、Connection オブジェクトの setTransactionIsolation メソッドを用います。
- イルシデーションロック
  SQL 文単位にロックのかかりかたを調整できます。
  イルシデーションロックを設定するには、SQL 文に WITH OPTION LOCK_MODE 句を付加します。
  イルシデーションロックの設定は、独立性水準の設定よりも優先されます。

［補足］

ResultSet 保持機能を有効にした状態で SELECT 文を実行すると、SELECT 文に自動的に、以下が付加されます。

- 「WITH OPTION LOCK_MODE(FREE LOCK)」
  これは Symfoware Server のイルシデーションロックという機能です。イルシデーションロックを用いると、データの排他制御を SQL 文ごとに指定することができます。
  FREE LOCK を指定すると、SELECT 文の実行が完了した時点で、検索対象のデータに対するロックが解除されることを意味します。
  なお、Symfoware Server Standard Edition、Symfoware Server Enterprise Edition、または、Symfoware Server Enterprise Extended Edition において PRECEDENCE(1)を指定した SEQUENTIAL 構造の場合、および、Symfoware Server Lite Edition の場合は、FREE LOCK を指定すると一定条件下で資源を占有しません。
- 「CURSOR_MODE(HOLD)」
  トランザクションが終了したあともカーソルをオープンしたままにすることを指定します。

Copyright 2011 FUJITSU LIMITED

Copyright 2011 FUJITSU LIMITED

第6章　データの更新

Copyright 2011 FUJITSU LIMITED

6.1 更新可能なResultSetオブジェクトの作成

ResultSet オブジェクトは、SELECT 文による検索結果を表すオブジェクトですが、一定の条件を満たしていれば、ResultSet オブジェクトを用いて検索結果の行を更新することができます。ResultSet を用いたデータの更新機能は、Symfoware Server V10 からサポートされています。

ResultSet を用いたデータの更新を行うには、ResultSet オブジェクトを更新可能なものとして作成する必要があります。そのためには、createStatement メソッドまたは prepareStatement メソッドを実行するときに、引数で指定します。これらのメソッドの仕様上、ResultSet の更新可能性を単独で指定することはできず、カーソルのタイプの指定とセットで指定する必要があります。

ResultSet の更新可能性の指定には、以下の 2 種類があります。

- CONCUR_READ_ONLY
  ResultSet オブジェクトが更新できないことを指定します。デフォルトでは、CONCUR_READ_ONLY が指定された状態になります。
- CONCUR_UPDATABLE
  ResultSet オブジェクトが更新できることを指定します。

下記の例では、ResultSet を更新可能とするように、CONCUR_UPDATABLE を指定しています。

(例)

```
// Statement オブジェクトを作成する。
Statement stmt = con.createStatement(
        java.sql.ResultSet.TYPE_FORWARD_ONLY,
        java.sql.ResultSet.CONCUR_UPDATABLE);
```


Copyright 2011 FUJITSU LIMITED

CONCUR_UPDATABLE を指定しても、常に ResultSet が更新可能になるわけではありません。例えば、SELECT 文で複数の表を結合している場合は、ResultSet を用いた更新はできません。また、カーソルタイプにスクロール可能カーソル（TYPE_SCROLL_INSENSITIVE や TYPE_SCROLL_SENSITIVE）を指定した場合は、ResultSet は更新できません。スクロール可能カーソルを指定しても、SELECT 文に FOR UPDATE 句を付加している場合は、ResultSet を更新することができます。

ResultSet を用いたデータの更新が可能かどうかは、カーソルが更新可能かどうかによります。CONCUR_UPDATABLE を指定しても、カーソルが読取専用になっている場合には、ResultSet を更新できません。

カーソルが読取専用になる条件は、マニュアル“SQL リファレンス”の“DECLARE CURSOR(カーソル宣言)”を参照してください。

Copyright 2011 FUJITSU LIMITED

6.2 ResultSetオブジェクトを用いたデータの更新

ResultSet オブジェクトには、列の値を更新するためのメソッドが、データ型ごとに用意されています。このメソッドは updateXXXX という形式の名前になっています。例えば、文字列型のデータを更新する場合には、updateString メソッドを用います。これらのメソッドを、updater メソッドと呼びます。

ResultSet オブジェクトを用いてデータを更新するには、以下のようにします。

(1) CONCUR_UPDATABLE を指定して Statement オブジェクトや PreparedStatement オブジェクトを作成する。
(2) Statement オブジェクトや PreparedStatement オブジェクトの executeQuery メソッドを用いて検索を行い、ResultSet オブジェクトを作成する。
(3) ResultSet のカーソルを、更新したい行に位置づける。
(4) ResultSet の updater メソッド群を用いて、列の値を更新する。
(5) ResultSet の updateRow メソッドを用いて、行を実際に更新する。
(6) 更新をコミットする。

(例)

```
import java.sql.*;
import java.io.*;

public class test
{
```


Copyright 2011 FUJITSU LIMITED

```
public static void main(String args[])
{
    try
    {
        // データベースに接続する。
        Class.forName("com.fujitsu.symfoware.jdbc.SYMDriver");
        Connection con
            = DriverManager.getConnection(
                "jdbc:symfold:///DB","UID", "PWD");

        con.setAutoCommit(false);

        // Statement オブジェクトを作成する。
        Statement stmt = con.createStatement(
            ResultSet.TYPE_FORWARD_ONLY,
            ResultSet.CONCUR_UPDATABLE);

        // 検索を行い、ResultSet オブジェクトを作成する。
        ResultSet rs = stmt.executeQuery(
            "SELECT ID,NAME FROM GENERAL.EMPLOYEE");

        // ResultSet を参照して、条件に合う行を更新する。
        int iID = 0;
        String sName = null;
```


Copyright 2011 FUJITSU LIMITED

```
                while(rs.next())
                {
                        iID = rs.getInt(1);
                        sName = rs.getString(2);
                        System.out.println("ID = " + iID);
                        System.out.println("NAME = " + sName);

                        if(iID==3)
                        {
                                rs.updateString(2,"leopard");
                                rs.updateRow();
                        }
                }

                // 更新をコミットして、データベースから切断する。
                stmt.close();
                con.commit();
                con.close();
        }
        catch (SQLException e)
        {
                System.out.println("ERROR MESSAGE : " + e.getMessage());
                System.out.println("SQLSTATE : " + e.getSQLState());
                System.out.println("ERROR CODE : " + e.getErrorCode());
```


Copyright 2011 FUJITSU LIMITED

```
                    e.printStackTrace();
                }
                catch (Exception e)
                {
                    System.out.println("ERROR MESSAGE : " + e.getMessage());
                    e.printStackTrace();
                }
            }
}
```

［補足］

一般に、オブジェクトに値を設定するメソッドを setter メソッド、オブジェクトから値を取り出すメソッドを getter メソッド、オブジェクトの保持している値を更新するメソッドを updater メソッドと呼びます。

［補足］

更新可能な ResultSet を作成しても、そのデータには更新ロックは掛けられていません。更新ロックがかかるのは、updateRow メソッドを実行したときです。

実際の更新を行うまでは、独立性水準やイルシデーションロックの設定によって、ロックのかかり方が変わります。

デフォルトの状態では、読み取り用のロックがかかっています。そのため、更新可能な ResultSet を作っても、他のトランザクションから同じデータを参照することができます。データを参照する

Copyright 2011 FUJITSU LIMITED

ことによって、他のトランザクションから読み取り用のロックが掛けられるため、ResultSet を用いたデータの更新を実行すると、ロック解除待ちとなります。

SELECT 文に FOR UPDATE 句を付加すると、最初から更新ロックがかかります。他のトランザクションによる参照がロック解除待ちになるため、他のトランザクションの終了を待つことなくデータの更新を行うことができます。

Copyright 2011 FUJITSU LIMITED

6.3 カーソルを使用した更新

ResultSet を用いたデータの更新機能は、Symfoware Server V10 以降でサポートされています。ResultSet の更新機能を用いずに検索結果を参照してデータを更新するには、ResultSet オブジェクトの getCursorName メソッドを用いた方法があります。この方法は、V9 までのバージョンでも利用可能です。

UPDATE 文や DELETE 文は、カーソルを位置づけた行を更新することができます。ResultSet のカーソルを使って、UPDATE 文や DELETE 文で目的の行に位置づけることができます。

(例)

```
// 更新対象を検索
ResultSet rs1 = null;
PreparedStatement pstmt1 = con.prepareStatement(
        "SELECT ID,NAME FROM GENERAL.EMPLOYEE FOR UPDATE");
rs1 = pstmt1.executeQuery();

// カーソル名を取得
String cursorName = rs1.getCursorName();

while(rs1.next()) {
        // UPDATE 文の WHERE CURRENT OF 句にカーソル名を指定してデータ更新
        PreparedStatement pstmt2 = null;
        pstmt2 = con.prepareStatement(
```


Copyright 2011 FUJITSU LIMITED

```
                                "UPDATE GENERAL.EMPLOYEE SET NAME ='human' WHERE
CURRENT OF " + cursorName );

                        int ret = pstmt2.executeUpdate();

                        pstmt2.close();

            }
```


Copyright 2011 FUJITSU LIMITED

6.4 RowIDを使用したデータ更新

RowID とは、データベース内の任意の表の行を一意に識別することのできる値です。RowID の実体が何であるかは、データベースソフトによって異なります。Symfoware Server の場合、RowID のことを行識別子と呼びます。これは 24 バイトの値で、行の物理的なアドレスを示します。

同じデータに何度もアクセスする場合、その行の物理的なアドレスが分かっていれば、処理を高速に行うことができます。対象データがどこにあるか探さなくても、直接場所を指定してアクセスできるからです。

行識別子は SELECT 文で値を取り出したり、UPDATE 文の WHERE 句に条件として指定したりすることができます。その際の列名にあたるのは、「ROW_ID」という語です。

以下のような SQL 文で、行識別子を扱うことができます。

(例)

```
SELECT ROW_ID FROM GENERAL.EMPLOYEE WHERE NAME=lion
```

上記の例では、NAME 列が“lion”である行を検索し、その行の行識別子(ROW_ID)を取り出しています。

(例)

```
UPDATE GENERAL.EMPLOYEE SET NAME='tiger' WHERE ROW_ID=?
```

上記の例では、ROW_ID が指定した値である行の NAME 列を“tiger”に変更しています。

Copyright 2011 FUJITSU LIMITED

このようにSQL文中では、「ROW_ID」という語を用いて行識別子を扱うことができます。これをJavaアプリケーションで利用するには、ResultSetオブジェクトのgetRowIDメソッドやPreparedStatementオブジェクトのsetRowIdメソッドを用います。

ROW_IDはSQL文中では列名として扱われるので、通常の列の値と同様に行識別子を操作することができます。

行識別子を用いた更新は、Symfoware Server V10からサポートされています。

(例)

```
// SELECT 文を表す Statement オブジェクトを作成する。
Statement stmt = con.createStatement();

// 行識別子を検索する。
ResultSet rs = stmt.executeQuery
        ("SELECT ROW_ID FROM GENERAL.EMPLOYEE WHERE NAME='lion'");

RowId rowid = null;

// ResultSet の getRowId を用いて、行識別子を取り出す。
while(rs.next())
{
        rowid = rs.getRowId(1);
}

// UPDATE 文を表す PreparedStatement オブジェクトを作成する。
```


Copyright 2011 FUJITSU LIMITED

```
            PreparedStatement pstmt = con.prepareStatement

                        ("UPDATE GENERAL.EMPLOYEE SET NAME='tiger' WHERE
ROW_ID=?");



            // 検索条件のパラメーターに、先ほど取り出した行識別子を設定する。

            pstmt.setRowId(1,rowid);



            // UPDATE 文を実行する。

            pstmt.executeUpdate();



            // オブジェクトをクローズして、更新をコミットする。

            rs.close();

            stmt.close();

            pstmt.close();

            con.commit();
```


Copyright 2011 FUJITSU LIMITED

6.5 RowIDを使用したデータ更新の注意点

行識別子を使用する場合、以下の点に注意してください。

- 取り出した行識別子は、そのトランザクション内で再検索または更新に使用することができます。
  ただし、トランザクションの独立性水準によっては同一トランザクション内においても、同じ行を検索できなかったり、異なる行を更新してしまったりすることがあります。
  これは独立性水準によって、排他制御によるデータの占有期間が変わるためです。
  不都合を避けるためには、トランザクションで使用する独立性水準に、問題を起こさないものを指定するなどして対処してください。

独立性水準	行識別子を指定した操作
SERIALIZABLE	同一行に対する操作が保証されます。
REPEATABLE READ	同一行に対する操作が保証されます。
READ COMMITTED	以下のいずれかの条件で取得した行識別子は、再検索で異なる行を識別することがあります。 ・ トランザクションアクセスモードが READ ONLY ・ カーソルの更新可能性句が READ ONLY Symfoware Server Standard Edition、Symfoware Server Enterprise Edition、または、Symfoware Server Enterprise Extended Edition において PRECEDENCE(1)を指定した SEQUENTIAL 構造の場合、および、Symfoware Server Lite Edition の場合、以下のいずれかの条件で取得した行識別子は、再検索で異なる行を識別することがあります。 ・ トランザクションアクセスモードが READ ONLY ・ カーソルの更新可能性句が READ ONLY または省略
READ UNCOMMITTED	再検索で異なる行を識別することがあります。

Copyright 2011 FUJITSU LIMITED

- 取り出した行識別子を、他のトランザクションで再検索または更新に利用すると、同じ行を検索できなかったり、異なる行を更新してしまったりすることがあります。
  このような場合、行識別子を取り出したときに行のデータも同時に取り出しておき、行識別子を利用して更新する前に、行識別子で再検索して行のデータが他のトランザクションにより変更されていないかどうかを調べる必要があります。

事象	原因
検索データなし	該当行が他のトランザクションによって削除または更新されています。
前回の検索結果と値が異なる	他のトランザクションにより行が更新されています。 または、他のトランザクションにより行が削除された後、別の行が挿入されています。

Copyright 2011 FUJITSU LIMITED

6.6 バッチ更新

バッチ更新の機能を使用すると、複数の更新系の SQL 文を一括して実行することができます。

一括して実行することで、性能向上を図ることができます。

例えば、以下のような INSERT 文を 100 回実行するケースを考えてみます。

(例)

```
INSERT INTO GENERAL.EMPLOYEE(ID,NAME) VALUES(?,?)
```

これを Statement オブジェクトを用いて実行するには、100 個の INSERT 文を文字列として組み立て、1 個ずつ executeQuery メソッドを用いて実行します。格納するデータが異なるだけの同じ INSERT 文であっても、毎回 SQL 文の解析が行われます。

PreparedStatement オブジェクトを用いれば、SQL 文の解析は最初の 1 回だけで済みますが、100 個の INSERT 文を 1 個ずつ executeQuery メソッドを用いて実行しなくてはならない点は変わりません。

バッチ更新機能を用いると、100 個分の INSERT 文を溜め込んで、1 回のメソッド呼び出しで一括して実行させることができます。このとき、クライアントとデータベースサーバとの通信は 1 回だけで済みます。1 回ずつ SQL 文を送信して結果を受信する場合に比べて、性能向上を期待できます。

バッチ更新機能は、Symfoware Server V10 からサポートされています。

バッチ更新には、以下の 2 種類があります。

Copyright 2011 FUJITSU LIMITED

- 異なる SQL 文を溜め込んで、一括して実行するパターン。
- 同一の SQL のプレースホルダに設定するデータを溜め込んで、一括して実行するパターン。

バッチ更新で実行する SQL 文でエラーが発生すると、バッチ更新全体がロールバックされます。

Copyright 2011 FUJITSU LIMITED

6.7 異なるSQL文を一括実行

Statement オブジェクトを用いたバッチ更新機能です。

Statement オブジェクトの addBatch メソッドを用いて、複数の INSERT 文、UPDATE 文、DELETE 文を溜め込みます。

溜め込んだ SQL 文を一括して実行するには、Statement オブジェクトの executeBatch メソッドを用います。

以下の例では、INSERT 文、UPDATE 文、DELETE 文からなるバッチ処理を実行しています。3 個の SQL 文を addBatch メソッドで溜め込みます。その後、executeBatch メソッドを実行して、SQL 文を一括して実行しています。

(例)

```
        // 自動コミットを解除する。
        con.setAutoCommit(false);

        // Statement オブジェクトを作成する。
        Statement stmt = con.createStatement();

        // SQL 文を溜め込む。
        stmt.addBatch("INSERT INTO GENERAL.EMPLOYEE(ID,NAME) VALUES(1,'tiger')");
        stmt.addBatch("UPDATE GENERAL.EMPLOYEE SET NAME='monkey' WHERE
ID=2");
        stmt.addBatch("DELETE FROM GENERAL.EMPLOYEE WHERE ID=3");
```


Copyright 2011 FUJITSU LIMITED

```
// バッチ更新を実行する。
int[] upc = stmt.executeBatch();


// 更新をコミットして、データベースから切断する。
stmt.close();
con.commit();
con.close();
```


Copyright 2011 FUJITSU LIMITED

6.8 同一のSQL文をパラメーターを変えて一括実行

PreparedStatement オブジェクトを用いたバッチ更新機能です。

PreparedStatement オブジェクトの addBatch メソッドを用いて、プレースホルダに設定するデータの値を溜め込みます。SQL 文は 1 種類だけですが、パラメーターの値が異なる複数の SQL 文となります。

溜め込んだ SQL 文を一括して実行するには、PreparedStatement オブジェクトの executeBatch メソッドを用います。

以下の例では、100 個の INSERT 文を実行しています。INSERT 文で格納するデータはプレースホルダで記述しておき、INSERT 文 100 個分の格納データの組を addBatch メソッドで溜め込みます。その後、executeBatch メソッドを実行して、100 個の INSERT 文を一括して実行しています。

(例)

```
// 自動コミットを解除する。
con.setAutoCommit(false);


// PreparedStatement オブジェクトを作成する。
PreparedStatement pstmt = con.prepareStatement(
        "INSERT INTO GENERAL.EMPLOYEE(ID,NAME) VALUES(?,?)");


// ループしながら、INSERT 文のパラメーターを設定する。
```


Copyright 2011 FUJITSU LIMITED

```
for( int iID=0 ; iID<100 ; iID++ )
{
        // 1 番目のパラメーターに iID の値を設定する。
        pstmt.setInt(1, iID);
        // 2 番目のパラメーターに配列 sNAME に格納されている文字列を設定する。
        pstmt.setString(2, sNAME[iID]);
        //  バッチ処理として登録する。
        pstmt.addBatch();
}
//  バッチ更新を実行する。
int[] upc = pstmt.executeBatch();

//  更新をコミットして、データベースから切断する。
pstmt.close();
con.commit();
con.close();
```


Copyright 2011 FUJITSU LIMITED

第7章　例外処理と後始末

Copyright 2011 FUJITSU LIMITED

7.1 例外処理

JDBC の処理でエラーが発生すると、例外として SQLException がスローされます。この例外はアプリケーションで catch する必要があります。

ここで注意しなければならないのは、どのエラーもすべて SQLException で表される、という点です。エラーの内容を調べるには、SQLException オブジェクトから、エラー情報を取り出して調べる必要があります。

SQLException から取り出せるエラー情報と、取り出す方法は以下のとおりです。

- エラーメッセージ
  SQLException オブジェクトの getMessage メソッドを用いて取り出すことができます。これは Symfoware Server のメッセージです。メッセージ番号とメッセージ本文からなります。
  メッセージの意味と対処方法は、マニュアル“メッセージ集”から調べることができます。
- SQLSTATE
  SQLException オブジェクトの getSQLState メソッドを用いて取り出すことができます。必要に応じてアプリケーション内で SQLSTATE の値を調べ、エラー処理を行ってください。
  例えば、トランザクションをロールバックして、オブジェクトをクローズしたあと、トランザクションの再実行を行うか、あるいは接続をクローズしてアプリケーションを終了するか、といったことを決定するために、SQLSTATE の値を利用することができます。
- エラーコード
  SQLException オブジェクトの getErrorCode メソッドを用いて取り出すことができます。エラーコードはデータベースベンダの固有の情報です。Symfoware Server では、SQLSTATE が S1000 の場合に Vendor エラーメッセージとして設定されています。それ以外の場合は、メッセージ番号が設定されています。

Copyright 2011 FUJITSU LIMITED

Vendor エラーメッセージについては、マニュアル“アプリケーション開発ガイド(JDBC 編)”の“Vendor エラーメッセージ”を参照してください。

Symfowre Server V10 以降で、JDBC4.0 準拠の JDBC ドライバを使用している場合、JDBC の API のうち Symfoware Server がサポートしていない API を使用すると、例外として SQLFeatureNotSupportedExcepiton が発生します。

JDBC の API の中には Symfoware Server ではサポートしていないものもあるので、注意してください。

Copyright 2011 FUJITSU LIMITED

7.2 オブジェクトのクローズ

不要になったJDBCのオブジェクトは早めにクローズしてください。

エラーが発生した場合、エラー処理の中で必要に応じてStatementオブジェクトやConnectionオブジェクトなどのクローズ処理を行ってください。

closeメソッドでエラーが発生した場合にもSQLExceptionが発生するため、エラー処理の中でも例外をcatchする必要があります。

［補足］

アプリケーションのプロセスが終了した場合は、すべてのオブジェクトは消滅し、データベースへの接続など、Symfoware Server側のリソースも回収されます。

Copyright 2011 FUJITSU LIMITED