第5章　Servletからのデータベース操作

これまでのサーブレットでは通常のファイルに永続的なデータを記録していたが、実際には、データベースを利用すると便利な場合が多く、実際に実用的なプログラムの多くがデータベースを利用する。この章では、Javaプログラムからデータベースを利用する方法を紹介する。

5.1　JDBCとは

JDBCとは **J**ava **D**ata**B**ase **C**onnectivityの略で、Javaからデータベースを利用するための統一インタフェース（API）である。JDBCを利用することにより、利用するデータベースサーバに依存せずにデータベースを利用するJavaプログラムを作成することが可能になる。

Oracle, PostgreSQL, MySQLなど、ほとんどのメジャーなデータベースサーバはJavaからJDBC経由で利用可能である。

本演習ではServletから利用するデータベースサーバとして、OpenOffice.org（ver. 2.0以降）でも採用されているHSQLDBを利用する。HSQLDBは100% Javaで記述された軽量でインストールが容易なデータベースサーバである。なお、他のデータベースサーバを利用する場合も、プログラム自体はほとんど変更する必要はない。

HSQLDBのインストール方法については別ドキュメントで説明する。また、SQLの基本的な書き方など関係データベースそのものに関する事柄は既知のものとして扱う。

5.2　JDBCの利用方法

Javaのプログラムからデータベースサーバを利用するためには、通常次のような手順を踏む必要がある。

1. JDBCドライバのロード
2. データベースサーバへの接続を確立
3. SQL文の送信
4. SQL文実行結果の取出し
5. データベースサーバへの接続を切断

次のサーブレットプログラム（DBSelect.java）でこれらを順に説明する。このDBSelect.javaはcustomerというテーブルから`"SELECT * FROM customer"`というSQLコマンドで全ての内容を取り出し、それを単にHTMLの表として整形するプログラムである。

まず、`java.sql.*`をimportしていることに注意する。JDBC関係のクラスは主にこのパッケージに定義されている。

ファイル DBSelect.java

```
import java.io.*;
import java.sql.*;
import javax.servlet.http.*;

public class DBSelect extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException {

    Connection conn = null;

    response.setContentType("text/html; charset=Windows-31J");
    PrintWriter out = response.getWriter();

    out.println("<html><head></head><body>");
    out.println("<table border='true'>");
    out.println("<tr><th>id</th><th>first</th><th>last</th>"
                +"<th>street</th><th>city</th></tr>");

    try {
      Class.forName("org.hsqldb.jdbcDriver");       // JDBCドライバの登録
      String url = "jdbc:hsqldb:hsql://localhost"; // HSQLDB用のURL
      String user = "sa", password="";              // HSQLDBの既定値
      conn = DriverManager.getConnection(url, user, password);    // 接続の確立
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM customer"); // SQLの送信

      while (rs.next()) {
        out.print("<tr>");
        out.print("<td>"+rs.getString("id")+"</td>");
        out.print("<td>"+rs.getString("firstname") + "</td>");
        out.print("<td>"+rs.getString("lastname") + "</td>");
        out.print("<td>"+rs.getString("street") + "</td>");
        out.print("<td>"+rs.getString("city") + "</td>");
        out.print("</tr>");
      }
    } catch (ClassNotFoundException e) {
      out.println("クラスが見つかりません。");
    } catch (SQLException e) {
      out.println("データベース操作中にエラーがありました。");
      e.printStackTrace(out);
    } finally {
      try {
        if (conn != null) { conn.close(); /* 接続の切断 */ }
      } catch (SQLException e) {}
    }
    out.println("</table>");
    out.println("</body></html>");
    out.close();
  }
}
```

5.2.1 JDBCドライバのロード

JDBC を利用するプログラムでは、通常、JDBC ドライバ（個々のデータベースサーバにアクセスするための固有のライブラリ）を動的に（つまりプログラム実行時に）ロードする。クラスを動的にロードするためには Class.forName というクラスメソッドを使用する。

HSQLDB の場合、動的にロードすべきクラス名は"org.hsqldb.jdbcDriver"である[1]ので、

```
Class.forName("org.hsqldb.jdbcDriver");
```

という呼出しをする。これで、JDBC ドライバがロード・登録され、利用可能な状態になる。

このメソッドは与えられたクラス名に対応する class ファイルを見付けられなかった場合、ClassNotFoundException を発生する。そのため ClassNotFoundException を catch ブロックで処理している。

5.2.2 接続の確立

次にデータベースサーバとの通信路を確立する。このためには、DriverManager.getConnection というクラスメソッドを利用する。このメソッドの第 1 引数は、接続のための URL 表記で、データベースを利用するための JDBC ドライバの種類とデータベースのある場所を示している。ローカルホスト（Servlet コンテナと同一のホスト）で動作している HSQLDB サーバにアクセスする場合は、この URL は "jdbc:hsqldb:hsql://localhost"とする。第 2 引数はユーザ名、第 3 引数はパスワードで、HSQLDB の場合、既定値のユーザ名・パスワードはそれぞれ"sa"と ""である。データベースへの接続を表すメソッドの戻り値は java.sql.Connection 型であり、以降はこの Connection オブジェクトを利用してデータベースを操作する。

> **参考:** データベースへの接続は重い処理なので、サーブレットの場合、プーリングという技法を用いて、一度確立した接続を何度も再利用するのが普通である。この場合、サーブレットの設定ファイルに、JDBC ドライバのクラス名や URL を記述する。本演習ではこの方法の説明は割愛する。

5.2.3 SQL 文の送信と結果の取出し

データベースサーバに SQL 文を送信するためにまず、java.sql.Statement クラスのオブジェクトを用意する。この Statement オブジェクトは Connection オブジェクトの createStatement メソッドで生成される。次に Statement オブジェクトの executeQuery メソッドの引数に SQL の SELECT 文を文字列として渡し、データベースサーバに送信する。データベースサーバでこの SQL 文が実行され、結果が Java のオブジェクトに変換されて Java 側に返される。この executeQuery メソッドの戻り値は java.sql.ResultSet というクラスのオブジェクトである。

なお、これらのデータベース操作はエラーを起こす可能性がある。データベース操作のときのエラーは、ほとんどの場合 SQLException クラスに属するので、catch ブロックでこの例外を処理している。

---

[1] 他のデータベースサーバを利用する場合のクラス名は、対応する JDBC ドライバのドキュメントを参照する必要がある。

ResultSet オブジェクトはクエリの結果の表に対応するデータを保持する。ResultSet オブジェクトの next メソッドは現在行を進める。初期状態では現在行は最初の行の前に位置付けられていて、next メソッドの最初の呼出しによって、最初の行が現在行になる。また、これ以上、行がない場合は、next メソッドは false を返す。この ResultSet オブジェクトの getString メソッドを、列名を引数として呼び出すと、現在行の当該の列の内容が String 型として取り出される。この例題の場合、HSQLDB のテストデータ中の customer というテーブルをアクセスする。このテーブルには、id, firstname, lastname, street, city という 5 つの列がある。ResultSet クラスには getString 以外にも getInt など他の型のデータを取り出す getter メソッドが多く存在する。またそれぞれの getter には列を名前 (文字列) ではなく列のインデックス (整数) で指定するバージョンもある。詳しくは java.sql.ResultSet クラスのドキュメント (***(J2SEAPI)***/java/sql/ResultSet.html) を参照すること。

問 5.2.1 *java.sql.PreparedStatement* クラスの使用法を調べよ。

5.2.4 接続の切断

データベースを利用する場合、接続の切断は確実に行なう必要がある。（そうしないとネットワークその他の資源が無駄使いされてしまう可能性がある。）DBSelect.java では切断を確実に行なうために、データベースに関する操作を全て **try** ブロックの中で行ない、それに対応する **finally** ブロック内で接続の切断（conn.close()）を行なっている。

5.3 データベースの更新

次に示す例はデータベースの内容を更新するサーブレットの例である。ここでは前節で利用したものと同じ customer テーブルを用い、顧客の住所を変更するサーブレットを示す。

このサーブレットは次のようなフォームから利用することを想定している。

ファイル DBUpdate.html

```
<html><head></head>
<body>
<form action='servlet/DBUpdate' method='POST'>
更新する値を入力してください。<br>
<table border>
<tr><th>列名</th>  <th>値</th></tr>
<tr><td>ID</td>    <td><input type='text' size='5' name='id'></td></tr>
<tr><td>STREET</td><td><input type='text' size='32' name='street'></td></tr>
<tr><td>CITY</td>  <td><input type='text' size='32' name='city'></td></tr>
</table>
<input type='submit' value='送信'>
</form>
</body>
</html>
```

単なるクエリではなく、データベースの内容を書き換える時には executeQuery メソッドの代わりに、executeUpdate メソッドを用いる。このプログラムでは SQL の UPDATE 文を使っているが、行を挿入する INSERT 文や削除する DELETE 文などを使う場合でも同様に executeUpdate メソッドを用い

ファイル DBUpdate.java

```
import java.io.*;
import java.sql.*;
import javax.servlet.http.*;

public class DBUpdate extends HttpServlet {
  @Override
  public void doPost(HttpServletRequest request, HttpServletResponse response)
              throws IOException {

    Connection conn = null;

    response.setContentType("text/html; charset=Windows-31J");
    PrintWriter out = response.getWriter();

    out.println("<html><head></head><body>");

    String id     = request.getParameter("id");
    String street = request.getParameter("street");
    String city   = request.getParameter("city");

    try {
      String user = "sa", password="";

      Class.forName("org.hsqldb.jdbcDriver");
      String url = "jdbc:hsqldb:hsql://localhost";
      conn = DriverManager.getConnection(url, user, password);
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("UPDATE customer SET street = '"+street
                         +"', city ='"+city+"' WHERE id = "+id);

      out.println("テーブルを更新しました。<br>");
      out.println("id="+id+", street='"+street+"', city='"+city+"'");
    } catch (ClassNotFoundException e) {
      out.println("クラスが見つかりません。");
    } catch (SQLException e) {
      out.println("テーブルの更新に失敗しました。");
      e.printStackTrace(out);
    } finally {
      try {
        if (conn != null) { conn.close(); }
      } catch (SQLException e) {}
    }
    out.println("</body></html>");
    out.close();
  }
}
```

る。executeUpdate メソッドは実行した SQL が対象とした行数を戻り値とする (ただし DBUpdate.java ではこの戻り値は利用していない)。それまでのデータベースへ接続する部分などは、DBSelect.java と全く変わらない。

このサーブレットはデータベースを操作したあと、「テーブルを更新しました」というメッセージと更新内容 (失敗したときは「テーブルの更新に失敗しました」というメッセージ) を表示する。

問 5.3.1 *X* 国では *price* が単価 *15* 以上のものに対して *10%* の間接税がかかるという。*HSQLDB* のテストデータの中の *product* テーブルの *price* は税抜の単価を示すものとする。*product* テーブルの内容を *X* 国での間接税を加えた値段を表に成形して出力するサーブレットを作成せよ。
（当然、データベース中のデータは変更してはいけない。）

問 5.3.2 *HSQLDB* のテストデータの中の *customer* テーブルに対して、既存の顧客のデータを変更するだけでなく、新規の顧客を追加する／顧客を削除することも可能な入力用の *HTML* ページと処理用のサーブレットを作成せよ。

キーワード :

JDBC, Class.forName メソッド, DriverManager.getConnection メソッド, Connection クラス, createStatement メソッド, Statement クラス, executeQuery メソッド, executeUpdate メソッド, ResultSet クラス, next メソッド, getString メソッド,