在 JSP 中实现数据库操作

本章学习目标

- 熟练掌握对 MySQL 数据库的建库、建表与对数据的操作。
- 了解用 Java 语言对 MySQL 数据库访问的编码实现。
- 熟练掌握在 JSP 网页中编码实现对 MySQL 中数据的访问操作。
- 熟练掌握通用的创建数据连接类的方法及其在 JSP 中的应用。
- 熟练掌握用 JSP 开发数据库应用程序(包括对数据库的增、删、改、查询操作)的方法与过程。

应用程序中常常有对数据库的操作,即需要将处理的业务数据存储在数据库中,然后通过程序对其进行操作处理。在 Java 程序设计的课程中一般会介绍采用 JDBC 方式连接数据库,并通过该连接实现对数据库的操作。本章将介绍如何将这种方式应用到 JSP 文件中,并介绍了通用的数据库连接类的创建方法及其使用。

当数据库连接建立后,就可以对数据库进行操作了,即调用 SQL 语句新增、修改、删除、查询数据库中的数据。本章介绍了利用不带参数的 SQL 语句以及带参数的 SQL 语句进行数据库操作,并以案例的形式介绍了 JSP 中数据库操作的综合应用。

3.1 Java 访问数据库概述

在第 2 章中图 2-1、图 2-2 已经说明了基于数据库服务器、Web 应用服务器的 JSP 运行原理。一般 Web 应用软件需要用数据库系统存储数据,而 Web 应用程序则将信息在客户端、到服务器、再到数据库之间进行数据操作处理。

在学习 Java 程序设计时,已经学习了在 Java 中利用 JDBC 方式对数据库的操作,本章将介绍在 JSP 中采用相同的 JDBC 方式,对数据库进行操作并对数据进行处理以及对处理代码的封装。首先回顾一下 Java 中采用 JDBC 方式对数据库的操作。

3.1.1 数据库运行环境介绍

对数据库操作首先要有数据库运行环境,即首先安装了数据库,我们选用 MySQL 数据库系统。MySQL 是免费软件,有很多优点,适合小型的应用。为了用 Java 访问 MySQL,本教程采用以下数据库环境:

(1) 以 MySQL 数据库管理系统作为数据库服务器;

(2) 采用 MySQL-Front.exe 作为 MySQL 数据库客户端软件;

(3) MySQL 驱动程序为 mysql-connector-java-5.1.5-bin.jar。

1. MySQL 数据库简介

MySQL 是一个开放源码的小型关联式数据库管理系统,其开发者为瑞典 MySQL AB 公司。由于企业运作的原因,目前 MySQL 已成为 Oracle 公司的另一个数据库项目。

由于 MySQL 体积小、速度快、总体拥有成本低,尤其是开放源码这一特点,许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库系统。所以,MySQL 被广泛地应用在 Internet 上的中小型网站中。

与其他大型数据库系统例如 Oracle、DB2、SQL Server 等相比,MySQL 自有它的不足,但对于一般的个人使用者和中小型企业来说,MySQL 提供的功能已经绰绰有余,而且由于 MySQL 是开放源码软件,因此受到用户的广泛欢迎。

MySQL 系统具有以下一些特性:

(1) 支持 AIX、FreeBSD、Linux、Mac OS、OS/2 Wrap、Solaris、Windows 等多种操作系统。

(2) 为多种编程语言提供了应用程序接口(API),如 C、C++、Python、Java、Perl、PHP 等。

(3) 提供多语言支持,常见的编码如中文的 GB 2312、BIG5、日文的 Shift_JIS 等。

(4) 提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径。

(5) 提供用于管理、检查、优化数据库操作的管理工具。

(6) 支持大型的数据库。可以处理拥有上千万条记录的大型数据库。

本教程采用 MySQL 作为数据库环境,对于大型数据库如 Oracle、DB2、SQL Server 等,用 Java 进行访问的原理与方法类似。MySQL 的官方网站为:http://dev.mysql.com/,在这里可以直接下载最新版本的 MySQL 数据库软件。

在 MySQL 安装之后,我们可以通过类似于 Oracle 的 SQL-Plus 命令行工具来使用它。可是这样的话,我们就必须熟练掌握大量的命令。对于大多数的用户来说,GUI(图形用户界面)总是比较受欢迎的。MySQL-Front 等 MySQL 客户端管理软件可以解决这个问题。

MySQL-Front 软件操作简单,是一款非常不错的 MySQL 管理软件,它非常容易上手,其他的 MySQL 的 GUI 使用类似。

2. MySQL 客户端管理软件

为了避免记忆大量的命令,简化学习过程,我们可以选择 MySQL 的 GUI 工具(图形用户界面)。MySQL 的 GUI 很多,例如 Navicat、MySQL GUI Tools、MySQL-Front 等,为了介绍方便,本教程选择 MySQL-Front 作为 MySQL 的 GUI。

例如,MySQL-Front 是一个小巧的数据库管理软件。它使用起来很简单,操作也很方便。

MySQL-Front 主要特性包括：多文档界面，语法突出，拖曳方式的数据库和表格；可编辑、增加、删除数据库和表，可编辑、插入、删除数据库记录；可执行的 SQL 脚本，提供与外程序接口，保存数据到 CSV 文件等。

3. 用 JDBC 连接 MySQL 数据库

Java 数据库连接(Java Data Base Connectivity，JDBC)是 Sun 公司(已被 Oracle 公司收购)制定的一个可以用 Java 语言连接数据库的技术。JDBC 是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 为数据库开发人员提供了一个标准的 API，据此可以构建更高级的工具和接口，使数据库开发人员能够用纯 Java API 编写数据库应用程序，并且可跨平台运行，并且不受数据库供应商的限制。

为了实现 Java 对 MySQL 的访问，需要下载 MySQL 支持 JDBC 的驱动程序，其下载的官方网站地址是：http://www.mysql.com/products/connector/。如果手上有则可跳过这一步直接使用。本教程使用的 MySQL 的 JDBC 驱动程序是：mysql-connector-java-5.1.5-bin.jar。

3.1.2 编写 Java 程序访问 MySQL 数据库

如果安装好了 MySQL 数据库及其客户端程序(GUI)，准备好了 MySQL 的 JDBC 驱动程序，就可以编写 Java 程序对 MySQL 数据库进行数据访问与操作了。

下面介绍用 Java 编写程序访问 MySQL 数据库中的数据。

【案例 3-1】 编写 Java 程序访问并显示数据库中的用户信息。

1. 建立被访问的数据库环境

用 Java 程序访问数据库，首先要建立被访问的数据库及表，并添加数据。在 MySQL-Front 中建立 MySQL 数据库：mydatabase，编码为 utf-8(支持中文)；再建立一个学生表：user，并在其中添加两个用户信息(如表 3-2 中的李国华、张淑芳所示)。学生表结构见表 3-1，添加的数据如表 3-2 所示。

表 3-1 学生表 user 结构

字段名	类型	是否为空	中文含义	备注
id	int(11)	No	编码	主键、自增
name	varchar(255)	Yes	姓名	
password	varchar(255)	Yes	密码	

表 3-2 在 student 表中添加两个学生信息

id	name	password	id	name	password
1	李国华	admin	3	张淑芳	zsf

完成上述操作后，便建立好数据库及供访问的数据了，可以从 MySQL-Front 中浏览到这些数据，如图 3-1 所示。

id	name	password
1	李国华	admin
3	张淑芳	zsf

图 3-1　创建好数据库后显示的数据

可以将上述操作“导出(Export)”为一个 SQL 脚本文件以便今后用该 SQL 脚本文件方便地重构该数据库(见以下代码)。

```
DROP DATABASE IF EXISTS 'mydatabase';
CREATE DATABASE 'mydatabase' /*!40100 DEFAULT CHARACTER SET gbk */;
USE 'mydatabase';
CREATE TABLE 'user' (
  'Id' int(11) NOT NULL auto_increment,
  'name' varchar(255) default NULL,
  'password' varchar(255) default NULL,
  PRIMARY KEY  ('Id')
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=gbk;
INSERT INTO 'user' VALUES (1,'李国华','admin');
INSERT INTO 'user' VALUES (3,'张淑芳','zsf');
UNLOCK TABLES;
```

2. 编写 Java 程序访问数据库

下面编写 Java 程序显示如图 3-1 所示的数据库中的数据。该数据库为 mydatabase，数据在用户表 user 中。

首先创建一个 Web 项目如 myweb，在其中创建一个 Java 类 researchdb.java，并编写其代码如下：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class researchdb {
    public static void main(String[] args) throws Exception {
        try {
            Class.forName("com.mysql.jdbc.Driver");     //注册 MySQL 驱动
            //获取数据库连接,设置数据库名为 mydatabase,用户名与密码分别为 root、
            //root。需根据自己的数据库环境修改这些参数
            Connection conn=DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/mydatabase", "root", "root");
            Statement stat=conn.createStatement();
                                        //创建 Statement 对象,准备执行 SQL 语句
            ResultSet rs=stat.executeQuery("select * from user");//执行 SQL
            while (rs.next()) {             //显示结果集对象中的数据
                System.out.print(rs.getString("name")+"  ");
                System.out.println(rs.getString("password"));
```

```
            }
            rs.closs();                    //释放资源
            stat.closs();
            conn.closs();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

编写好以上 researchdb.java 程序后，还需要在开发环境中加载 MySQL 数据库的驱动程序 mysql-connector-java-5.1.5-bin.jar，有两种方法：

(1) 直接将驱动程序 mysql-connector-java-5.1.5-bin.jar 复制到项目 myweb 的 WebRoot\Web-INF\lib 文件夹中。

(2) 在 MyEclipse 中鼠标右击 myweb 项目名，依次选择 Bulid Path->Add External Archives，在出现的对话框中寻找到自己存放的驱动程序 mysql-connector-java-5.1.5-bin.jar 并打开。

无论上述哪种方法，均能在 myweb 项目中加载 MySQL 数据库连接驱动程序。然后可以运行(Run As->Java Application) researchdb.java 程序，则在控制台显示数据库中的数据，如图 3-2 所示。

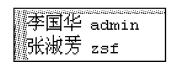


图 3-2 执行 Java 程序显示的数据

3. 代码解释

researchdb.java 程序代码需完成以下一些任务。

1) 在 Java 程序中加载驱动程序

在 Java 程序中，可以通过 Class.forName("指定数据库的驱动程序")方式来加载添加到开发环境中的驱动程序，加载 MySQL 的数据驱动程序的代码为：Class.forName(com.mysql.jdbc.Driver)。

2) 创建数据连接对象

通过 DriverManager 类创建数据库连接 Connection 对象。DriverManager 类作用于 Java 程序和 JDBC 驱动程序之间，用于检查所加载的驱动程序是否可以建立连接，然后通过它的 getConnection 方法，根据数据库的 URL、用户名和密码，创建一个 JDBC Connection 对象，如 Connection conn = DriverManager.getConnection("连接数据库的URL", "用户名", "密码")。其中，

URL=协议名+IP 地址(域名)+端口+数据库名称

用户名和密码是指登录数据库时所使用的用户名和密码。本案例中创建 MySQL 的数据库连接代码如下：

```
Connection conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/
mydatabase","root","root");
```

3）创建 Statement 对象

Statement 类主要是用于传递静态 SQL 语句（不带参数），并可以调用其执行的 executeQuery()方法执行 SQL 对数据库的操作，返回的结果存放到 ResultSet 对象中。通过 Connection 对象的 createStatement()方法可以创建一个 Statement 对象。本案例中创建 Statement 对象的代码如下：

```
Statement stat=conn.createStatement();
```

4）调用 Statement 对象的相关方法执行相应的 SQL 语句

可通过 execuUpdate()方法来实现数据的更新，包括插入和删除等操作，例如本案例查询 Student 表中的数据的代码为

```
ResultSet rs=stat.executeQuery("select * from user");
```

通过调用 Statement 对象的 executeQuery()方法进行数据的查询，而查询结果会得到 ResultSet 对象，ResultSet 表示执行查询数据库后返回的数据集合，ResultSet 对象具有可以指向当前数据行的指针。通过该对象的 next()方法，使得指针指向下一行，然后将数据以列号或者字段名取出。如果用 next()方法返回 null，则表示下一行中没有数据存在。

同样，如果是插入的 SQL 语句，则可以为

```
stat.excuteUpdate("INSERT INTO user(name, password) VALUES ('王武', ,'ww')");
```

5）关闭数据库连接以释放资源

使用完数据库或者不需要访问数据库时，通过 Connection 的 close() 方法及时关闭数据连接以释放资源，其代码如下：

```
rs.closs();
stat.closs();
conn.closs();
```

在此，介绍了编写 Java 程序访问及显示数据库中的数据。简单介绍了 MySQL 数据库驱动程序的安装，关于如何加载数据库驱动程序、如何进行程序编码是重要的操作技能，需要读者通过实践掌握。

4. 数据库连接简介

在 Java 进行数据库应用程序编码中已经讲过数据库连接了，由于其在 Java、JSP 中进行数据库操作时是必不可少的，所以这里就数据库连接的相关概念进行简单介绍。

其实，连接是编程语言与数据库交互的一种方式，如果没有数据库连接，我们就无法对数据库进行操作。获取数据库连接主要通过下列两条语句：

```
(1) Class.forName("com.mysql.jdbc.Driver");
(2) Connection con
    =DriverManager.getConnection ("jdbc:mysql://localhost:3306/mydatabase",
```

```
    "root", "root");
```

在此，我们创建了一个 Connection 对象 con，即所谓的“连接”对象，通过它可以对数据库进行各种操作。但是该连接对象在创建时内部执行了什么呢？其实，上述两条语句中，DriverManager 用于检查并注册驱动程序；com. mysql. jdbc. Driver 就是我们注册了的驱动程序（此处为 MySQL 数据库对应的驱动程序）后，它就会在驱动程序类中调用 connect（url…）方法，该方法根据我们请求的连接地址（如"jdbc: mysql://localhost: 3306/mydatabase", "root", "root"），创建一个“Socket 连接”，连接到 IP 为 localhost: 3306、默认端口为 3306 的数据库。这里 localhost 是指本地机的服务器，否则需要指定远程数据库服务器的 IP 地址。

最后，创建的 Socket 连接将被用来查询我们指定的数据库，并最终让程序返回得到一个结果，这个结果就是今后数据库访问要用到的数据库连接对象。

3.1.3 在 JSP 中编写 Java 代码段访问数据库

【案例 3-2】 在 JSP 中编写 Java 代码显示数据库中的用户信息。

将案例 3-1 中的 Java 代码以＜% %＞的形式放到 JSP 文件中，则可以在 JSP 中访问并显示数据库中的数据了。

在原案例 3-1 的 myweb 项目中编写 researchdb. jsp 代码如下：

```
<%@page language="java" import="java.util.*,java.sql.*"
pageEncoding="gbk"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>数据库查询</title>
   </head>
  <body>
<%                                          //访问数据库的 Java 代码段
  try {
          Class.forName("com.mysql.jdbc.Driver");
          Connection conn=DriverManager.getConnection(
                  "jdbc:mysql://localhost:3306/mydatabase", "root", "root");
          Statement stat=conn.createStatement();
          ResultSet rs=stat.executeQuery("select * from user");
          while (rs.next()) {              //循环显示结果集中的数据
              out.print(rs.getString("name")+" ");
              out.print(rs.getString("password")+"<br>");
          }
       rs.close();
       stat.close();
       conn.close();
       } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    %>
  </body>
</html>
```

在 Web 项目中加载 MySQL 驱动程序(如果已经加载了可跳过该步),部署该 Web 项目,启动 Tomcat 服务器,在 IE 地址栏中输入

```
http://localhost:8080/myweb/researchdb.jsp
```

则在页面中显示了所访问的数据库的数据,如图 3-3 所示。

图 3-3 在 JSP 页面中显示数据库中的数据

在 Web 方式中 researchdb.jsp 显示数据库的数据,其实只要在案例 3-1 的基础上做简单的修改。

(1) 创建 JSP 文件:researchdb.jsp。

(2) 将案例 3-1 的 Java 代码的 try-catch 中的代码,以<% %>的形式放到 JSP 的<body> </body>标记中。

(3) 将 java.sql.* 加到 import 语句中,即该语句改为:import="java.util.*,java.sql.*"。

(4) 将 System.out.print 语句改为 out.print 语句。

然后部署该项目,启动服务器后运行,则在 JSP 中显示了数据库中的数据(如图 3-3 所示)。

3.2 编写可重用的类封装数据库处理代码

3.2.1 在 JSP 中连接数据库编码的缺陷

从案例 3-2 中可以看出,JSP 文件实现了数据库的操作。但是上述 JSP 对数据库的操作有以下一些缺陷:

(1) 代码累赘,大量的 Java 代码段在 JSP 中,显得 JSP 文件复杂且不利于修改;

(2) 代码重用性差,因为每个 JSP 都需要进行相同的重复操作,不利于代码复用;

(3) 影响性能,由于通过 JSP 进行数据库连接操作,需要从客户端到服务器进行交互,影响软件的性能;

(4) 安全性差,用户的验证代码在客户端的 JSP 程序中,具有不安全性;

(5) 连接数据库的代码大量冗余,不利于系统维护。

如果采用 Java 类进行封装,在 JSP 需要的时候进行调用,就可以改进上述存在的问题。

3.2.2 通过Java类封装数据库处理代码

案例3-2的JSP程序researchdb.jsp中访问数据库的Java代码可以分为以下几个部分：

(1) 加载数据库驱动并获取数据库连接；

(2) 定义SQL语句并执行、处理数据；

(3) 关闭连接等以释放资源。

上述第(1)、第(3)两部分对于相同的数据库(如MySQL数据库)均相同，所以可以定义一个Java类，将它们封装起来以重复使用。

第(2)部分包括定义一个statement对象、SQL语句并执行，以及执行完后对获取的结果集数据的处理。这部分往往与具体的业务处理有关，所以可以将这些处理的代码放到另一个处理类中进行封装。

另外，对处理的数据需要用一个实体类进行封装以便进行各个阶段的处理。

最后，通过主程序调用这些代码以完成数据处理的功能。下面通过一个案例说明如何进行数据库处理Java代码的封装。

【案例3-3】 编写Java类封装数据库处理代码，以供JSP等程序复用。

根据上述分析，将处理数据库中数据的JSP程序分为以下4个部分：

(1) 封装数据库连接的共享Java类(共享工具类)；

(2) 封装数据的Java类(实体类)；

(3) 封装业务处理的Java类(模型类)；

(4) JSP主程序(主控程序)。

本案例的程序结构的设计见表3-3。

表3-3 案例3-3程序结构设计

序号	名称	包/文件夹	程序	说　明
1	共享工具类	src/dbutil	Dbconn.java	包含获取连接、关闭连接两个方法
2	实体类	src/entity	User.java	三个属性及它们的setter/getter方法
3	业务处理模型类	src/model	Model.java	包含业务处理的SQL语句的定义与执行，返回执行的结果
4	主控程序(JSP程序)	根文件夹	listUsers.jsp	将上述三个程序结合到一起完成整个功能

下面分别介绍表3-3中4个部分的实现。

1. 封装获取数据库类的编写

由于每个数据库操作都需要获取数据库连接、关闭数据库连接操作，这些代码可以抽象出来共享，这样可以大大减少代码的冗余，提高开发效率。

用dbutil.Dbconn.java类封装获取数据库连接及关闭数据库连接的代码，其代码如下：

```
package dbutil;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
//数据库连接处理类的定义,包括获取连接、关闭连接两个处理方法的定义
public class Dbconn {
    //获取连接方法的代码
    private Connection conn;
    public  Connection getConnection() throws SQLException{
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/
            mydatabase","root","");
        } catch (ClassNotFoundException e) {
            System.out.println("找不到服务!!");
            e.printStackTrace();
        }
        return conn;
    }
    //关闭连接方法的代码
    public void closeAll(Connection conn,Statement stat,ResultSet rs){
        if(rs!=null){
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }finally{
                if(stat!=null){
                    try {
                        stat.close();
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }finally{
                        if(conn!=null){
                            try {
                                conn.close();
                            } catch (SQLException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }
```

```
            }
        }
    }
}
```

在上述代码中，定义 getConnection()方法获取数据库连接并返回 Connection 类型的连接；定义 closeAll(Connection conn, Statement stat, ResultSet rs)方法关闭连接、Statement 对象和 ResultSet 对象，以释放资源。程序中对数据库操作时，均可以共享该代码。

2. 编写封装数据的实体类

通过实体类封装需要处理的数据库中的数据。本案例中数据库表 user 有三个字段：id，name，password 分别表示用户的编号、姓名、密码。在实体类中，也定义对应的三个属性，见实体类 entity. User. java 中的代码。

```
package entity;
//封装数据的实体类,定义的三个属性与数据库表的三个字段对应
public class User {
    private int id;
    private String name;
    private String password;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id=id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name=name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password=password;
    }
}
```

实体类中还要创建三个属性的 setter/getter 方法(setter/getter 方法的创建可以在 MyEclipse 中自动完成)。

其实，该实体类被称为是一种 JavaBean，即是一种用于封装数据并满足某种标准的

Java 类。封装数据的 JavaBean 应满足的标准包括类是公共的、有无参构造器，要求属性是 private 且需通过 setter/getter 方法取值等。

3. 编写封装业务处理的类

对业务的处理跟业务本身有关，它对应软件中的一个业务逻辑处理模型。而一个软件模块代码有界面、业务处理、数据库处理等部分。数据库处理已经抽取出来，但业务逻辑处理代码如果放到界面中，就会加大界面程序的负担，可以将它抽象到模型中，本案例将其放到 model. Model. java 类中。model. Model. java 类充当了模块处理的逻辑模型，其代码如下：

```
package model;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import dbutil.Dbconn;                          //引入数据库连接类
import entity.User;                            //引入实体类

//业务处理过程封装到一个模型中(业务处理类),通过定义的 userSelect()方法实现
public class Model {
    private Statement stat;
    private ResultSet rs;
    Dbconn s=new Dbconn();
    //定义返回查询处理后获取的对象集合并返回
    public List<User>userSelect(){
        List users=new ArrayList();
        try {
            Connection conn=s.getConnection();
            String sql="select * from user";
            stat=conn.createStatement();
            rs=stat.executeQuery(sql);
            User user;
            while(rs.next()){
                user=new User();
                user.setId(rs.getInt("id"));
                user.setName(rs.getString("name"));
                user.setPassword(rs.getString("password"));
                users.add(user);
            }
                s.closeAll(conn,stat,rs);
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

上述类程序中定义了SQL语句及对其进行了执行,同时对执行的结果数据进行了处理(保存到User类型的对象集合中)。该类封装了业务处理,它对实体JavaBean进行了操作,也是一种JavaBean,只是封装处理的JavaBean。这些JavaBean往往充当程序中的处理模型,构成了模型层。

模型本身只有通过调用才能执行,才具有实际意义。在主程序中,包括对模型的调用、返回结果的处理等。

4. 主程序的编写

上述三个程序不能单独运行,只有在主程序中才能形成一个完整的处理过程。本案例的主程序listUsers.jsp包括调用模型(Model.java)并将获取的结果在页面中进行显示。listUsers.jsp代码如下:

```
<%@page language="java" import="java.util.*,dbutil.*,entity.*,model.*"
pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>显示数据页面</title>
  </head>
  <body>
    <%
     Model model=new Model();                    //调用模型
    List<User>list=model.userSelect();  //执行模型中的查询方法,并返回结果
          %>
          //数据库中的所有用户
          <table border="1">
       <%for(int i=0;i<list.size();i++){%>//循环显示获得的结果(用户信息)
              <tr>   //从集合中取出对象的属性进行显示
              <td><%=list.get(i).getId()%></td>
              <td><%=list.get(i).getName() %></td>
              <td><%=list.get(i).getPassword() %></td>
              </tr>
       <%
       }
     %>
     </table>
```

```
  </body>
</html>
```

图 3-4　在 JSP 页面中调用 Java 类显示数据

在上述代码中，先实例化模型，然后调用其 userSelect()方法并将返回的结果存放到 list 中(list 类型的集合)。然后通过 Java 的 for 循环显示 list 中各个对象的属性(显示的结果见图 3-4)。注意：在 JSP 中要引入 dbutil.*，entity.*，model.*等类。

主程序集成其他程序形成一个完整的处理过程，其程序结构如表 3-3 所示。

通过上述步骤 1～4 的程序代码，介绍了案例 3-3 要求的用 Java 类封装数据库处理代码，供 JSP 程序调用以完成数据库处理功能。

3.2.3　JavaBean 是可重用的封装数据或处理的类

通过案例 3-3，可将封装数据的实体类，以及封装数据库处理的类抽象出来，这些类可以被复用，即可以不仅仅提供一次使用，如果程序的其他地方需要用到该处理程序也可以调用它，这就是所谓的 JavaBean 的一种类型(封装数据的 JavaBean)。

JavaBean 其实也是一种类，但它是可以重用的 Java 类(或称为软件部件)，为了满足应用的要求，对 JavaBean 类有一些要求与约定。比如 JavaBean 类要求是公共的、并提供无参的公有的构造方法；要求属性私有；具有公共的访问属性的 setter/getter 方法。

Java 之父 James Gosling 在设计 Java 语言，为 Java 组件中封装数据的 Java 类进行命名时，看见桌子上的咖啡豆，于是灵机一动就把它命名为 JavaBean。Bean 在中文中表示“豆子”的含义。

JavaBean 是 Java 语言中开发的可跨平台的可复用组件，它在 Web 应用的服务器应用中具有强大的生命力，在 JSP 程序中一般用来封装业务逻辑、封装数据库操作等。所以，一般将 JavaBean 技术作为模型层技术。JavaBean 包括封装数据和封装业务处理两类。本书后面所说的“模型”，就是一种封装业务的 JavaBean，而实体类则是封装数据的 JavaBean。

综上所述，JavaBean 实质上是一个 Java 类，但是它有自己独有的特点。这些特点主要表现在以下几个方面。

(1) JavaBean 需定义为公共的类。

(2) JavaBean 构造函数没有输入参数。

(3) 属性必须声明为 private，而方法必须是 public 类型。

(4) 用一组 setter 方法设置内部属性，而用一组 getter 方法获取内部属性。

(5) JavaBean 中是没有主方法(main())的类，一般的 Java 类默认继承 Object 类，而 JavaBean 则不需要此继承。

3.3 数据库操作交互模型的实现

3.3.1 预编译SQL语句的使用

1. PreparedStatement对象

在前面的案例3-1～案例3-3中，访问数据库是通过Statement对象实现的，其代码如下。

```
Statement stat=conn.createStatement();
ResultSet rs=stat.executeQuery("select * from user");
```

在此，操作数据库的SQL语句是固定的，即如果不改变程序，则该SQL语句执行同一操作。如果要查询满足某个用户的记录，则需加一个where＜条件＞，如果换一个用户则需要修改where中的条件，即需要修改程序。如果要求不修改程序就能满足查询不同的用户，则需要在该SQL语句中设置参数，再在执行SQL语句前根据情况设置不同的值，从而达到查询不同用户的目的。

这种SQL语句称为动态SQL语句，它会根据用户的不同操作动态生成不同的SQL语句从而查询不同的结果；而案例3-1～案例3-3中的SQL语句均是不变的，所以称其为静态SQL语句。

静态SQL语句由Statement对象定义并执行。

动态SQL语句则由PreparedStatement对象定义与执行，它使用预编译SQL语句，该SQL语句中允许有一个或多个输入参数(用“?”表示)。在执行带参数的SQL语句前，必须对“?”进行赋值。为了对“?”赋值，PreparedStatement对象中有大量的setXXX方法完成对输入参数的赋值。

PreparedStatement对象的使用与Statement对象类似，例如以下代码：

```
PreparedStatement psm=conn.prepareStatement("select * from user where name
=?");
psm.setString(1,"Tom"); //1为参数号,即这里为第1个参数,Tom为要查询的姓名
ResultSet rs=psm. executeQuery();
```

然后对结果集对象rs进行操作，其操作同案例3-3。

下面通过一个案例说明PreparedStatement对象的使用。

2. PreparedStatement对象的应用案例

【案例3-4】 查询3号操作员的信息。

实现该案例可以用Statement对象的SQL语句：select * from user where id＝3；也可以用PreparedStatement对象的SQL语句：select * from user where name＝?，然后再通过setXXX方法对“?”进行赋值。

在案例3-3的程序代码基础上修改完成。

修改 Model.java 代码，在其中加入一个 load()方法，其代码如下：

```
public User load(Integer id) {
        User user=null;
        String sql="select * from user where id=?";
        try {
            Connection conn=s.getConnection();
            ps=conn.prepareStatement(sql);
            ps.setInt(1, id.intValue());
              rs=ps.executeQuery();
              if(rs.next()){
                user=new User();
                user.setId(rs.getInt("Id"));
                user.setName(rs.getString("name"));
                user.setPassword(rs.getString("password"));
            }
            s.closeAll(conn,stat,rs);
        } catch (Exception e) {
            e.printStackTrace();
        }
    return user;
}
```

在上述代码中要用到 PreparedStatement 对象，即需要加下列引入包语句。

```
import java.sql.PreparedStatement;
```

下一步修改 JSP 文件，使其显示一个编号为 3 的学生姓名。JSP 程序文件为 showUser.jsp，其代码如下。

```
<%@page language="java" import="java.util.*,dbutil.*,entity.*,model.*"
pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>显示数据页面</title>
  </head>
  <body>
    <%
     Model model=new Model();
      User user=model.load(3);     //参数为 3,即查询 id 为 3 号的用户
    %>
    3 号用户姓名是：<br>
    <%=user.getName()%>
  </body>
</html>
```

本案例的其余代码复用案例 3-3 中的相应代码，即不需要修改直接使用。它们包括实体类 User.java、数据库连接工具类 Dbconn.java。部署运行 showUser.jsp 的结果显示如图 3-5 所示。

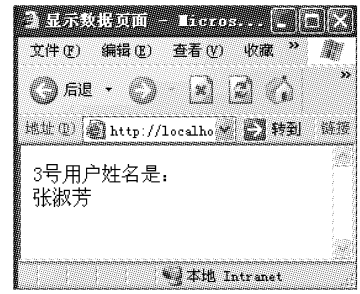


图 3-5 在 JSP 页面中显示 3 号用户的姓名

除了 setInt 外，preparedStatement 还提供了 setLong、setString、setBoolen、setShort、setByte 对不同类型的数据进行赋值，还提供了几个特殊的 setXXX 方法以处理特殊数据，如空值 null 等。

3.3.2 数据库操作交互模型的实现

一个应用软件系统应提供与用户的交互功能，即用户输入数据由系统进行处理，然后通过界面将结果返回给用户。

【案例 3-5】 用户输入一个用户 id，查询该 id 的用户信息并显示出来。

案例实现思路是：根据案例 3-4 中的代码，加一个输入界面，用于输入用户的 id，然后调用一个查询显示该用户信息的界面，调用 model 中的 load(id)方法，返回查询到的 user(用户信息)，并显示它们(运行效果见图 3-6(a)、图 3-6(b))。程序结构设计如表 3-4 所示。

表 3-4 案例 3-5 程序结构设计

序号	名称	包/文件夹	程序	说 明
1	共享工具类	src/dbutil	Dbconn.java	同案例 3-3
2	实体类	src/entity	User.java	同案例 3-3
3	业务处理模型类	src/model	Model.java	同案例 3-3 的 Model.java 类，但其中需定义 load(id)方法并返回 user 对象
4	JSP 程序	根文件夹	input.jsp	输入 id 界面
			research.jsp	调用模型中 load(id)，并显示结果

下面重点介绍需新增的代码，主要是：输入 id 的界面 input.jsp、Model.java 中的 load(id)方法，以及查询数据并显示结果的界面 research.jsp。

1. 输入 id 的界面

输入 id 的界面为 input.jsp，其代码如下：

```
<%@page language="java" import="java.util.*" pageEncoding="gbk"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>查询用户</title>
  </head>
  <body>
    <form action="research.jsp" method="post">
```

```
        请输入你要查询的 id 号：<input type="text" name="id"><br>
        <input type="submit" value="提交">
    </form>
  </body>
</html>
```

输入 id 界面主要是输入 id，并转到 research.jsp 中进行处理。而 research.jsp 将获取该 id，调用 Model 中的 load(id)方法，获取 user 对象，并显示其中的数据。

2. 模型中 load(id)方法的定义与实现

在 model.Model.java 中定义 load(id)方法，实现数据库查询并返回 user 类型的对象。关于 Model 中对数据库处理的方法在前面的案例中已经介绍过，在这里封装了动态 SQL 语句的定义与执行，返回结果为 user 类型的对象。

Model.java 中定义 load()方法在案例 3-4 中已经介绍，此处略。

3. 调用方法 load()并显示查询结果

在 research.jsp 中调用方法 load()并显示查询结果。research.jsp 的代码如下：

```
<%@page language="java"
import="java.util.*,dbutil.*,entity.*,model.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>显示数据页面</title>
  </head>
  <body>
  你查询的数据是：
    <%
     Model model=new Model();
     int id=Integer.parseInt(request.getParameter("id"));
      User user=model.load(id);
out.print(user.getId()+" "+user.getName()+" "+user.getPassword());
     %>
   </body>
</html>
```

在 research.jsp 中，获取 input.jsp 中输入的 id，调用模型中的 load(id)方法，获取用户对象 user，再将结果显示出来。

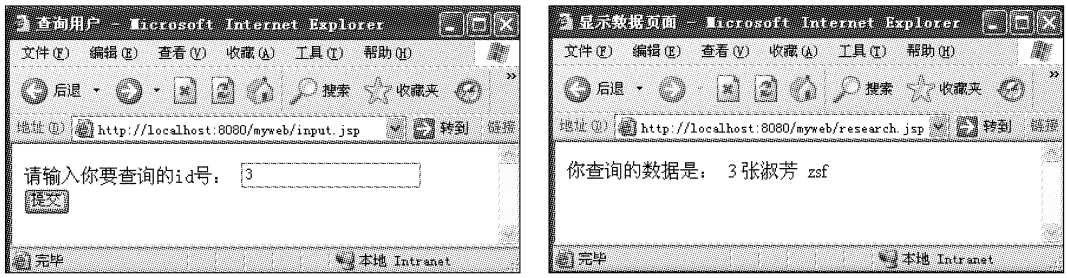
注意：如果 load()中的参数为 int 类型，则需要通过以下语句进行转换：

```
int id=Integer.parseInt(request.getParameter("id"));
```

然后才能进行调用。

```
User user=model.load(id);
```

案例 3-5 运行的结果如图 3-6 所示。



(a) 输入用户id　　(b) 根据id查询出的信息

图 3-6　动态输入用户 id 显示用户信息

3.4　综合案例：用户管理综合功能的实现

通过上面介绍的教学内容与案例，已经可以通过 JSP、Java 代码对 MySQL 数据库进行查询操作了，其操作过程为常见的分层模式。

而对于数据库的操作，除了查询显示外，还有增加、删除、修改等，只有这些功能齐全，才能满足用户的操作要求。

下面通过案例介绍对用户信息进行增加、删除、修改操作。

【案例 3-6】 综合运用前面学习的实现技术，编写程序实现对用户信息的增加、删除、修改、信息显示操作。

3.4.1　实现思路

实现思路是：分别编写程序实现对用户在数据库中的信息的增加、删除、修改操作。这些操作分为不同的功能，由不同的功能代码完成。

在编写这些对数据库操作中，前面案例的代码有些可以复用，它们是：

(1) 实体类，entity. User. java。

(2) 数据库连接工具，dbutil. Dbconn. java。

另外，每个功能需要增加一个逻辑处理模型及其处理方法，并且还要分别编写这些方法的调用程序，如表 3-5 所示。

表 3-5　对用户信息进行操作对应的程序

功　能	处理模型程序	处 理 方 法	方法调用程序
增新记录	model. Model. java	insert(Integer id, String name, String password)	insert. jsp insertShow. jsp
修改记录	model. Model. java	update(Integer id, String name, String password)	update. jsp updateShow. jsp
删除记录	model. Model. java	delete(Integer id)	dele. jsp deleShow. jsp
显示全部	model. Model. java	ArrayList userSelect()	showUser. jsp

3.4.2 实现代码提示

在对数据库中的记录进行增加、修改、查询时，分别需要用 insert、update、delete 对数据库进行操作。由于操作时需要动态地传递新增数据、修改数据、删除的记录，所以需使用 PreparedStatement 对象对数据库进行操作，具体代码见下面的提示。

新增操作代码片段如下（在 model. Model insert(Integer id, String name, String password)方法中）。

```
⋮
String sql="insert user values(?,?,?)";          //定义新增 SQL 语句
ps=conn.prepareStatement(sql);
ps.setInt(1, id);
ps.setString(2, name);
ps.setString(3, password);
a=ps.executeUpdate();                            //执行 SQL 语句
⋮
```

修改操作代码片段如下（在 model. Model update(Integer id, String name, String password)方法中）。

```
⋮
String sql="update user set name=?,password=?where id=?";
ps=conn.prepareStatement(sql);
ps.setInt(3, id);
ps.setString(1, name);
ps.setString(2, password);
a=ps.executeUpdate();
⋮
```

删除操作代码片段如下（在 model. Model delete(Integer id)方法中）。

```
⋮
String sql="delete from user where id=?";        //定义删除 SQL 语句
ps=conn.prepareStatement(sql);
ps.setInt(1, id);
a=ps.executeUpdate();                            //执行 SQL 语句
⋮
```

在上述对数据库进行增加、修改、删除操作的代码中，分别用到了 SQL 预处理对象 PreparedStatement 定义 SQL 语句，并传递方法中的参数到 SQL 语句中，调用 executeUpdate()方法执行 SQL 语句实现对数据库的操作。

将这些处理代码与 JSP 界面代码结合起来，便可以完整地实现用户对数据库的增、删、改的操作请求。