第5章

Web数据库编程

Web 应用一般需要访问数据库，Java 程序使用 JDBC 访问数据库，它是 Java 程序访问数据库的标准，由一组用 Java 语言编写的类和接口组成，实现对数据库的操作。

本章首先介绍 MySQL 数据库的使用，然后介绍 JDBC 访问数据库的步骤，并通过一个实例演示 Servlet 如何访问 MySQL 数据库，接下来介绍数据源和连接池的概念及应用，最后介绍使用 DAO 设计模式访问数据库。

本章内容要点

- MySQL 数据库。
- JDBC 访问数据库的步骤。
- 数据源的配置和使用。
- DAO 设计模式。

扫一扫

视频讲解

5.1 MySQL 数据库

MySQL 是一款开放源代码的关系数据库管理系统(RDBMS)，目前属于 Oracle 旗下的产品。MySQL 软件分为社区版和商业版，由于其体积小、速度快、成本低，尤其是开放源代码，一般中小型网站的开发都选择 MySQL 作为网站数据库。

5.1.1 MySQL 的下载与安装

用户可以到 Oracle 官方网站下载最新的 MySQL 软件，MySQL 提供了 Windows 系统下的安装程序，本书使用的是 MySQL 社区版(Community Server)，其下载地址如下。

```
http://dev.mysql.com/downloads/mysql
```

MySQL 的最新版本是 MySQL 8.0，下载文件名为 mysql-installer-community-8.0.29.0.msi。双击该文件即可开始安装，在安装过程中需要选择安装类型(选择 Developer Default 即可)和安装路径。

在安装结束后需要配置 MySQL，指定配置类型，这里选择 Development Machine，还需要打开 TCP/IP 网络及指定数据库的端口号，默认值为 3306。单击 Next 按钮，在出现的页面中需要指定 root 账户的密码，这里输入 123456。在下一步指定 Windows 服务名，这里指定 MySQL80。

开发 Web 应用程序使用 JDBC 访问数据库，因此还需要下载 MySQL 的 JDBC 驱动程序包，它在 MySQL 中叫作 Connector /J，在 MySQL 的同一下载页面可以找到其下载链接。在 Select Operating System 列表框中选择 Platform Independent，单击 Download 按钮即可下载。

5.1.2　使用 MySQL 命令行工具

MySQL 服务器自带了一个字符界面命令行工具和一个 MySQL Workbench 图形界面管理工具。如果要使用命令行工具，单击“开始”按钮，选择“所有程序”→“MySQL 8.0 Command Line”，打开命令行窗口，输入 root 账户的密码，此时会出现 mysql >提示符，如图 5-1 所示。

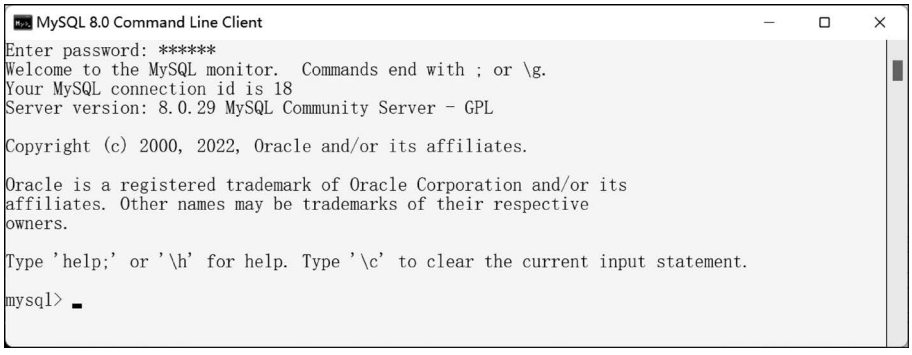

图 5-1　MySQL 命令行窗口

在 MySQL 命令提示符下可以通过命令操作数据库，使用命令可以显示所有数据库信息。

```
mysql > SHOW DATABASES;
```

使用 CREATE DATABASE 命令可以建立数据库，使用 CREATE TABLE 语句可以完成对表的创建，使用 ALTER TABLE 语句可以对创建后的表进行修改，使用 DESCRIBE 命令可以查看已创建的表的详细信息，使用 INSERT 命令可以向表中插入数据，使用 DELETE 命令可以删除表中的数据，使用 UPDATE 命令可以修改表中的数据，使用 SELECT 命令可以查询表中的数据。

1. 创建数据库

创建数据库使用 CREATE DATABASE 命令，下面创建一个名为 elearning 的数据库。

```
mysql > CREATE DATABASE elearning;
```

在默认情况下，新建的数据库属于创建它的用户，这里创建的数据库属于 root 用户。另外，也可以新建用户，并把数据库上的操作权限授予新用户。

在对数据库进行操作之前，必须使用 USE 命令打开数据库，下面打开 elearning 数据库。

```
mysql > USE elearning;
```

使用 SHOW TABLES 命令可以显示当前数据库中的表。

```
mysql > SHOW TABLES;
```

2. 使用 DDL 创建数据库对象

创建表使用 CREATE TABLE 命令，使用下面的 SQL 语句创建 students(学生)表。

```
CREATE TABLE students (
    stud_id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    gender CHAR(2) NOT NULL,
```

```
    birthday DATE,
    phone VARCHAR(14)
);
```

这里，字段 stud_id 表示学号、name 表示姓名、gender 表示性别、birthday 表示出生日期、phone 表示电话。

3. 使用 DML 操纵表

用户可以使用 SQL 的 INSERT、DELETE 和 UPDATE 语句插入、删除和修改表中的数据，使用 SELECT 语句查询表中的数据。

使用下面的语句向 students 表中插入两行数据。

```
mysql > INSERT INTO students(stud_id,name,gender,birthday,phone)
    VALUES(20220008,'张大海','男','1990 - 12 - 20',13050461188);

mysql > INSERT INTO students(name,gender,birthday,phone)
    VALUES('李清泉','女','1983 - 10 - 01',13504162222);
```

使用下面的语句可以查询 students 表中的所有信息。

```
mysql > SELECT * FROM students;
```

5.1.3 MySQL Workbench

MySQL Workbench 是 MySQL 数据库自带的一个图形界面管理工具，使用该工具可以创建数据库、创建表，以及对表数据、视图、存储过程和函数进行管理。

选择“MySQL”程序组中的“SQL Workbench 8.0 CE”打开 Workbench 界面，选择 Database 菜单中的“Connect to Database”命令连接数据库，之后在打开的界面中可以创建数据库和表，界面如图 5-2 所示。

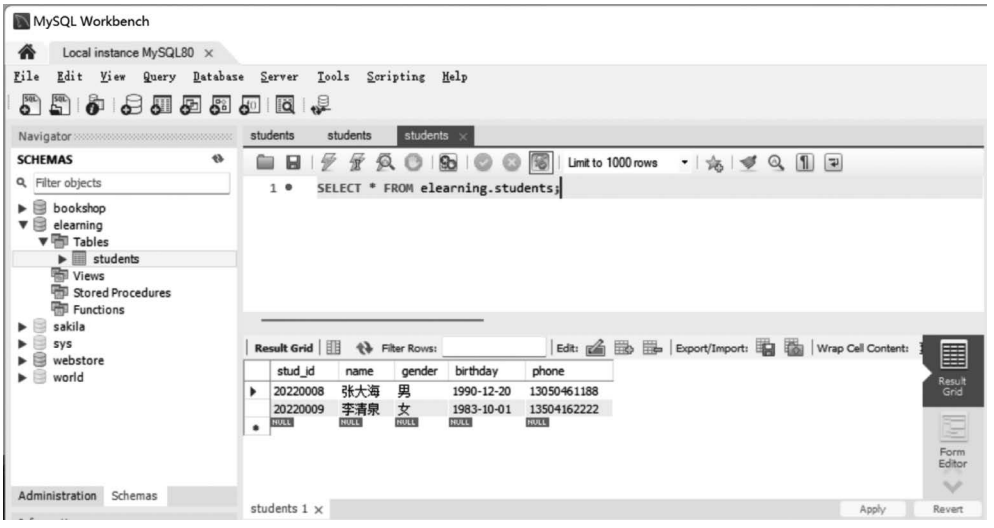

图 5-2 MySQL Workbench 工作界面

多学一招

除了可以使用 MySQL Workbench 管理 MySQL 数据库，用户还可以使用 Navicat 和 SQLyog 等工具管理 MySQL 数据库。

5.2 数据库的访问步骤

使用JDBC API访问数据库通常分为5个步骤：①加载驱动程序；②建立连接对象；③创建语句对象；④执行语句并处理结果；⑤关闭有关对象。

5.2.1 加载驱动程序

驱动程序是实现了Driver接口的类，一般由数据库厂商提供。加载JDBC驱动程序最常用的方法是使用Class类的forName()静态方法，该方法的声明格式如下。

```
public static Class <?> forName(String className)
                        throws ClassNotFoundException
```

参数className为用字符串表示的完整的驱动程序类名。该方法返回一个Class类的对象。如果找不到驱动程序将抛出ClassNotFoundException异常。

对于不同的数据库，驱动程序的类名不同。下面几行代码分别是加载MySQL数据库、Oracle数据库和PostgreSQL数据库驱动程序。

```
Class.forName("com.mysql.cj.jdbc.Driver");
Class.forName("oracle.jdbc.driver.OracleDriver");
Class.forName("org.postgresql.Driver");
```

5.2.2 建立连接对象

在驱动程序加载成功后应该使用DriverManager类的getConnection()建立数据库连接对象。下面的代码建立一个到MySQL数据库的连接。

```
String dburl = "jdbc:mysql://127.0.0.1:3306/elearning?useSSL = false&serverTimezone = UTC";
Connection dbconn = DriverManager.getConnection(dburl, "root", "123456");
```

在上述代码中127.0.0.1为本机IP地址，也可以使用localhost，3306为MySQL数据库服务器使用的端口号，数据库名为elearning，用户名为root，密码为123456。

5.2.3 创建语句对象

在JDBC API中定义了3种语句类型，包括Statement、PreparedStatement和CallableStatement。Statement用于执行简单的SQL语句，PreparedStatement用于执行带参数的SQL语句，CallableStatement用于执行数据库存储过程。下面的代码创建一个预编译的PreparedStatement对象。

```
String sql = "SELECT * FROM students";
PreparedStatement pstmt = dbconn.prepareStatement(sql);
```

下面的代码创建一个用于插入记录的语句对象。

```
String sql = "INSERT INTO students VALUES(?, ?, ?, ?,? )";
PreparedStatement pstmt = dbconn.prepareStatement(sql);
```

5.2.4 执行SQL语句并处理结果

对于查询语句，调用executeQuery()返回ResultSet。ResultSet对象保存查询的结果集，再调用ResultSet的方法可以对查询结果的每一行进行处理。

```
String sql = "SELECT * FROM students";
PreparedStatement pstmt = dbconn.prepareStatement(sql);
ResultSet rst = stmt.executeQuery();
while(rst.next()){
    out.print(rst.getString(1) + "\t");
}
```

对于 DML 语句(如 INSERT、UPDATE、DELETE)和 DDL 语句(如 CREATE、ALTER、DROP 等)需要使用语句对象的 executeUpdate()方法。该方法的返回值为整数,用来指示被影响的行数。

对于带参数的语句对象,先设置参数值,然后才能执行语句。对于前面的 INSERT 语句,可以使用下面的方法设置占位符的值。

```
LocalDate localDate = LocalDate.of(2002,Month.NOVEMBER,20);
java.sql.Date d = java.sql.Date.valueOf(localDate);
pstmt.setInt(1,20240001);
pstmt.setString(2,"刘小明");
pstmt.setString(3,"女");
pstmt.setDate(4,d);
pstmt.setString(5,"8899123");
int n = pstmt.executeUpdate();
```

5.2.5 关闭有关对象

在 Connection 接口、PreparedStatement 接口和 ResultSet 接口中都定义了 close(),当这些对象使用完毕后应该使用 close()关闭。如果使用 try-with-resources 语句,则可以自动关闭这些对象。

扫一扫

视频讲解

5.3 案例学习：使用 Servlet 访问数据库

本案例根据用户输入的学号从数据库中查询学生的信息,或者查询所有学生的信息。本应用的设计遵循了 MVC 设计模式,其中视图有 find-student.jsp、show-student.jsp、show-all-student.jsp 和 error.jsp 几个页面,Student 类实现模型,FindStudentServlet 类实现控制器。

本书使用 Maven 管理项目,需要将下面的依赖项添加到 pom.xml 文件中。

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.29</version>
</dependency>
```

本案例使用 5.1.2 节创建的 students 表,其中包含 5 个字段。根据 students 表的定义,设计下面的 Student 类存放学生的信息(代码见清单 5.1),该类实际上是一个 JavaBean,这里 Student 类的成员变量与 students 表中的字段对应。

清单 5.1 Student.java

```
package com.boda.domain;

import java.time.LocalDate;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.io.Serializable;
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor

public class Student implements Serializable{
    private Integer studId;
    private String name;
    private String gender;
    private LocalDate birthday;
    private String phone;
}
```

下面的 FindStudentServlet 连接数据库(代码见清单 5.2),当用户在 find-student.jsp 页面的文本框中输入学号,单击“确定”按钮时,将执行 doPost()方法;当用户单击“查询所有学生”链接时,将执行 doGet()方法。

清单 5.2　FindStudentServlet.java

```
package com.boda.controller;

import java.io.*;
import java.sql.*;
import java.util.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;
import com.boda.domain.Student;

@WebServlet(name = "findStudentServlet",value = "/find-student")
public class FindStudentServlet extends HttpServlet{
    public void init() {
        String driver = "com.mysql.cj.jdbc.Driver";
        try{
            Class.forName(driver);        //加载驱动程序
        }catch(ClassNotFoundException e){
            System.out.println(e);
            getServletContext().log("找不到驱动程序类!");
        }
    }

    public Connection getConnection() throws SQLException{
        String username = "root";
        String password = "123456";
        String dburl = "jdbc:mysql://127.0.0.1:3306/elearning"
                        + "?useSSL=false&serverTimezone=UTC";
        Connection conn = null ;
        try {
            conn = DriverManager.getConnection(dburl,username,password);
        }catch(SQLException e){
            throw e;
        }catch(Exception e){
            throw new RuntimeException(e);
        }
        return conn;
    }

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException,IOException{
```

```
        ArrayList<Student> studentList = new ArrayList<Student>();
        String sql = "SELECT * FROM students";
        try(
            Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql);
            ResultSet result = pstmt.executeQuery();
          )
            {
            while(result.next()){
                Student student = new Student();
                student.setStudId(result.getInt("stud_id"));
                student.setName(result.getString("name"));
                student.setGender(result.getString("gender"));
                student.setBirthday(result.getDate("birthday").toLocalDate());
                student.setPhone(result.getString("phone"));
                studentList.add(student);
            }
        }catch(SQLException e){
            e.printStackTrace();
        }
        if(!studentList.isEmpty()){
            request.getSession().setAttribute("studentList",studentList);
            response.sendRedirect("/chapter05/show-all-student.jsp");
        }else{
            response.sendRedirect("/chapter05/error.jsp");
        }
    }

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                   throws ServletException,IOException{
        String studentid = request.getParameter("stud_id");
        String sql = "SELECT * FROM students WHERE stud_id = ?";
        try(
            Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql);
          ){
            pstmt.setString(1,studentid);
            ResultSet rst = pstmt.executeQuery();
            if(rst.next()){
                Student student = new Student();
                student.setStudId(rst.getInt("stud_id"));
                student.setName(rst.getString("name"));
                student.setGender(rst.getString("gender"));
                student.setBirthday(rst.getDate("birthday").toLocalDate());
                student.setPhone(rst.getString("phone"));
                request.getSession().setAttribute("student", student);
                response.sendRedirect("/chapter05/show-student.jsp");
            }else{
                response.sendRedirect("/chapter05/error.jsp");
            }
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

该程序在init()方法中加载数据库驱动程序，用getConnection()方法返回数据库连接对象。在doGet()方法中查询数据库中所有学生的信息，将结果存储到ArrayList中，并将其存

储到会话对象中。在 doPost()方法中根据学号查询学生的信息,并将其存储到会话对象中。在这两个方法中都使用了 try-with-resources 结构创建对象,这保证了在离开 try 结构时自动关闭创建的对象。

该应用视图包含 3 个 JSP 页面,find-student.jsp 页面用于显示输入表单,show-student.jsp 页面用于显示一名学生的信息,show-all-student.jsp 页面用于显示所有学生的信息。这 3 个页面的代码见清单 5.3～清单 5.5。

清单 5.3 find-student.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head><title>学生查询</title></head>
<body>
<p><a href="find-student">查询所有学生</a></p>
<form action="find-student" method="post">
  请输入学生学号:
  <input type="text" name="stud_id" size="15">
  <input type="submit" value="确定">
</form>
</body>
</html>
```

find-student.jsp 页面的运行结果如图 5-3 所示。

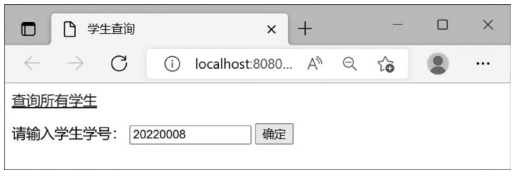

图 5-3 find-student.jsp 页面的运行结果

清单 5.4 show-student.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html>
<head>
<title>学生信息</title>
</head>
<body>
    <table border="0">
        <tr><td>学号:</td><td>${student.studId}</td></tr>
        <tr><td>姓名:</td><td>${student.name}</td></tr>
        <tr><td>性别:</td><td>${student.gender}</td></tr>
        <tr><td>出生日期:</td><td>${student.birthday}</td></tr>
        <tr><td>电话:</td><td>${student.phone}</td></tr>
    </table>
</body>
</html>
```

当单击图 5-3 中的“查询所有学生”链接时,将执行 Servlet 的 doGet()方法,查询所有学生,最后控制将转到 show-all-student.jsp 页面。

清单 5.5 show-all-student.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ page import="com.boda.domain.Student" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>所有学生信息</title></head>
<body>
<table border="1">
<tr><td>学号</td><td>姓名</td><td>性别</td>
<td>出生日期</td><td>电话</td><td>是否删除</td></tr>
<c:forEach var="student" items="${studentList}">
   <tr><td>${student.studId}</td>
```

```
            <td>${student.name}</td>
            <td>${student.gender}</td>
            <td>${student.birthday}</td>
            <td>${student.phone}</td>
            <td><a href="delete-student?id=${student.studId}">删除</a></td>
        </tr>
    </c:forEach>
    </table>
    </body></html>
```

当查询的学生不存在时，将显示 error.jsp 页面，其代码见清单 5.6。

清单 5.6　error.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html><body>
    该学生不存在。<a href="/chapter05/find-student.jsp">返回</a>
</body></html>
```

在图 5-3 所示的页面中输入学号，单击“确定”按钮，将显示如图 5-4 所示的页面；在图 5-3 所示的页面中单击“查询所有学生”链接，将显示如图 5-5 所示的页面。

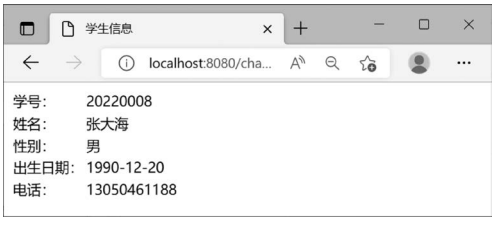

图 5-4　显示指定学生

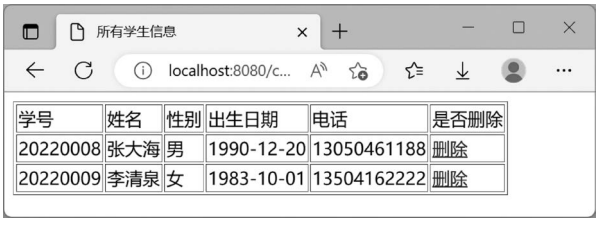

图 5-5　显示所有学生

扫一扫

视频讲解

5.4　使用数据源

在设计访问数据库的 Web 应用程序时，设计人员需要考虑的一个主要问题是如何管理 Web 应用程序与数据库的通信。一种方法是为每个 HTTP 请求创建一个连接对象，Servlet 建立数据库连接、执行查询、处理结果集、请求结束关闭连接。建立连接是比较耗费时间的操作，如果在客户每次请求时都要建立连接，这将导致增加请求的响应时间。此外，有些数据库支持的同时连接的数量要比 Web 服务器少，这种方法限制了应用程序的可缩放性。

为了提高数据库的访问效率，从 JDBC 2.0 开始提供了一种更好的方法建立数据库连接对象，即使用连接池和数据源的技术访问数据库。

5.4.1　数据源概述

数据源(data source)的概念是在 JDBC 2.0 中引入的，是目前 Web 应用开发中获取数据库连接的首选方法。这种方法是事先建立若干连接对象，将它们存放在数据库**连接池**(connection pooling)中供数据访问组件共享。使用这种技术，应用程序在启动时只需要创建少量的连接对象即可。这样就不需要为每个 HTTP 请求都创建一个连接对象，这会大大减少请求的响应时间。

数据源通过 javax.sql.DataSource 接口对象实现，通过它可以获得数据库连接，因此它是对 DriverManager 工具的一个替代。通常 DataSource 对象是从连接池中获得连接对象。连

接池预定义了一些连接，当应用程序需要连接对象时就从连接池中取出一个，当连接对象使用完毕将其放回连接池，从而可以避免在每次请求连接时都要创建连接对象。

5.4.2 配置 JNDI 数据源

下面讨论在 Tomcat 中如何配置使用 DataSource 建立数据库连接。

可以采用 Java 命名与目录接口(Java Naming and Directory Interface，JNDI)技术来获得 DataSource 对象的引用。JNDI 是一种将对象和名字绑定的技术，对象由对象工厂创建，并将其绑定到唯一的名字，外部程序可以通过名字来获得某个对象的访问。

在 javax.naming 包中提供了 Context 接口，该接口提供了将名字和对象绑定，通过名字检索对象的方法。用户可以通过该接口的一个实现类 InitialContext 获得上下文对象。

在 Tomcat 中可以配置局部数据源和全局数据源两种数据源。局部数据源只能被定义数据源的应用程序使用，全局数据源可以被所有的应用程序使用(本书不讨论全局数据源)。

建立局部数据源非常简单，首先在 Web 应用的 META-INF 目录中建立一个 context.xml 文件，在其中配置连接 MySQL 数据库的数据源，内容见清单 5.7。

清单 5.7 context.xml

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< Context reloadable = "true">
< Resource
    name = "jdbc/elearningDS"
    type = "javax.sql.DataSource"
    maxTotal = "4"
    maxIdle = "2"
    driverClassName = "com.mysql.cj.jdbc.Driver"
    url = "jdbc:mysql://127.0.0.1:3306/elearning?
          useSSL = false&amp;serverTimezone = UTC"
    username = "root"
    password = "123456"
    maxWaitMillis = "5000"/>
</Context >
```

在上述代码中，< Resource >元素的各属性的含义如下。

- name：数据源名，这里是 jdbc/elearningDS。
- driverClassName：使用的 JDBC 驱动程序的完整类名。
- url：传递给 JDBC 驱动程序的数据库 URL。
- username：数据库用户名。
- password：数据库用户的密码。
- type：指定该资源的类型，这里为 DataSource 类型。
- maxTotal：指定数据源最多的连接数。
- maxIdle：连接池中可空闲的连接数。
- maxWaitMillis：在没有可用连接的情况下，连接池在抛出异常前等待的最大毫秒数。

通过上面的设置，不用在 Web 应用程序的 web.xml 文件中声明资源的引用就可以直接使用局部数据源。

5.4.3 案例学习：使用 JNDI 数据源

在配置了数据源后，就可以使用 javax.naming.Context 接口的 lookup()查找 JNDI 数据

源，如下面的代码可以获得 jdbc/elearningDS 数据源的引用。

```
Context context = new InitialContext();
DataSource dataSource =
     (DataSource)context.lookup("java:comp/env/jdbc/elearningDS");
```

查找数据源对象的 lookup()的参数是数据源名字符串，但要加上“java:comp/env”前缀，它是 JNDI 命名空间的一部分。在得到了 DataSource 对象的引用后，就可以通过它的 getConnection()获得数据库连接对象。

对于清单 5.2 的数据库连接程序，如果使用数据源获得数据库连接对象，修改后的代码如下。

```
@WebServlet(name = "findStudentServlet" , value = "/find - student" )
public class FindStudentServlet extends HttpServlet{

   DataSource dataSource = null;

   public void init() {
     try {
       Context context = new InitialContext();
       dataSource =
          (DataSource)context.lookup("java:comp/env/jdbc/elearningDS");
     }catch(NamingException ne){
        System.out.println("Exception:" + ne);
     }
   }
   //doGet()和 doPost()方法中的 Connection 对象通过 DataSource 的 getConnection()方法获得
}
```

这段代码首先通过 InitialContext 类创建一个上下文对象 context，然后通过它的 lookup()查找 JNDI 数据源对象，最后通过数据源对象从连接池中返回一个数据库连接对象。当程序结束数据库访问后，应该调用 Connection 的 close()将连接对象返回到数据库连接池。这样就避免了每次使用数据库连接对象都要重新创建，从而可以提高应用程序的效率。

多学一招

在 Java Web 应用中也可以使用第三方数据源 API，例如，C3P0 是一个开源的 JDBC 连接池，它实现了数据源和 JNDI 的绑定，支持 JDBC 4 规范的标准扩展。用户可以到网上下载最新版本的 C3P0，将有关库复制到 WEB-INF\lib 目录中即可。在 Maven 项目中也可以通过依赖项下载，请读者参考本章的习题 14。

扫一扫

视频讲解

5.5 DAO 设计模式

DAO(Data Access Object)称为数据访问对象。DAO 设计模式可以在使用数据库的应用程序中实现业务逻辑和数据访问逻辑的分离，从而使对应用的维护变得简单。它通过将数据访问实现(通常使用 JDBC 技术)封装在 DAO 类中提高应用程序的灵活性。

DAO 模式有很多变体，这里介绍一种比较简单的形式。首先定义一个 Dao 接口，它负责建立数据库连接，然后为每种实体的持久化操作定义一个接口，如 ProductDao 接口负责 Product 对象的持久化；CustomerDao 接口负责 Customer 对象的持久化；最后定义这些接口的实现类。图 5-6 给出了 Dao 接口、ProductDao 接口和 CustomerDao 接口的关系。

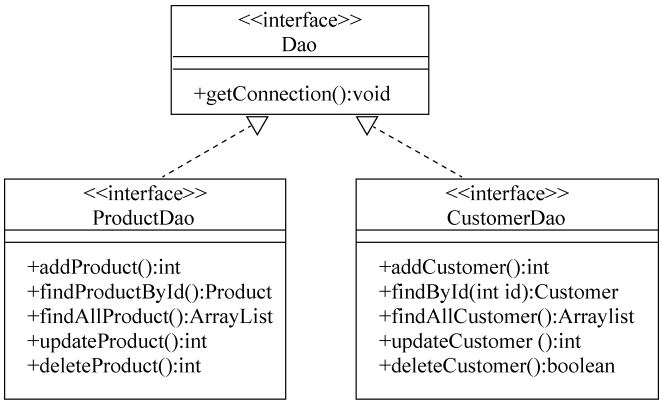

图 5-6 Dao 接口及其子接口

在 DAO 模式中,通常要为需要持久存储的每种实体类型编写一个相应的类。例如要存储 Product 信息就需要编写一个类,该实体类应该提供添加、删除、修改、检索、查找等功能。

假设存储商品信息的数据库表使用清单 5.8 中的 SQL 脚本定义,它包含 4 个字段,与 Product 类的 4 个成员对应。

清单 5.8 create_products.sql

```
CREATE TABLE products (
   id INTEGER NOT NULL PRIMARY KEY,        -- 商品号
   pname VARCHAR(20) NOT NULL,             -- 商品名
   brand VARCHAR(20) NOT NULL,             -- 品牌
   price FLOAT                             -- 价格
);
```

5.5.1 设计实体类

实体类用来存储要与数据库交互的数据。实体类通常不包含任何业务逻辑,业务逻辑由业务对象实现,因此实体类有时也叫**普通的 Java 对象**(Plain Old Java Object,POJO)。实体类必须是可序列化的,也就是它必须实现 java.io.Serializable 接口。清单 5.9 中的 Product 类就是实体类。

清单 5.9 Product.java

```
package com.boda.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.io.Serializable;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Product implements Serializable {
    private Integer id;
    private String name;
    private String brand;
    private double price;
}
```

该类用于在程序中保存应用数据,并可以实现对象与关系数据的映射。

5.5.2 设计 DAO 接口

本示例中的数据访问对象组件包含下面的接口和类。

- Dao 接口：所有接口的根接口，其中定义了默认方法建立到数据库的连接。
- ProductDao 接口和 ProductDaoImpl 实现类：提供了对 Product 对象的各种操作方法。

Dao 接口见清单 5.10，ProductDao 接口见清单 5.11，ProductDaoImpl 类见清单 5.12。

清单 5.10 Dao.java

```
package com.boda.dao;

import java.sql.Connection;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public interface Dao {
   private DataSource getDataSource(){
       DataSource dataSource = null;
       try {
         Context context = new InitialContext();
         dataSource = (DataSource)context.lookup(
                     "java:comp/env/jdbc/elearningDS");
      }catch(NamingException ne){
         System.out.println("异常:" + ne);
      }
      return dataSource;
   }

   public default Connection getConnection() throws SQLException {
       DataSource dataSource = getDataSource();
       Connection conn = null;
        try{
            conn = dataSource.getConnection();
       }catch(SQLException sqle){
        System.out.println("异常:" + sqle);
       }
       return conn;
   }
}
```

该接口的 getDataSource()方法用于查找并返回数据源对象，getConnection()方法是接口的默认方法，它通过一个数据源对象创建并返回数据库连接对象，该方法将被子接口或实现类继承。

清单 5.11 ProductDao.java

```
package com.boda.dao;

import java.util.List;
import java.sql.SQLException;
import com.boda.domain.Product;

public interface ProductDao extends Dao{
    //按 ID 查询商品方法
    public Product findProductById(int id) throws SQLException;
```

```
    //查询所有商品方法
    public List < Product > findAllProduct() throws SQLException;
    //添加商品方法
    public int addProduct(Product product) throws SQLException;
}
```

为了简单，该接口只定义了 3 个对 Product 操作的方法。addProduct()方法用来插入一个商品记录，findProductById()方法用来查询一件商品，findAllProduct()方法用来返回所有商品信息。

清单 5.12　ProductDaoImpl.java

```
package com.boda.dao;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import com.boda.domain.Product;

public class ProductDaoImpl implements ProductDao{
    //插入一条商品记录
   public int addProduct(Product product) throws SQLException{
      String sql = "INSERT INTO products VALUES(?,?,?,?)";
      int n = 0
      try(
         Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)
         ){
         pstmt.setInt(1,product.getId());
         pstmt.setString(2,product.getName());
         pstmt.setString(3,product.getBrand());
         pstmt.setDouble(4,product.getPrice());
        n = pstmt.executeUpdate();
     }catch(SQLException se){
        se.printStackTrace();
     }
     return n;
   }

   //按 ID 查询商品记录
    public Product findProductById(int id) throws SQLException{
       String sql = "SELECT id,pname,brand,price" +
                    " FROM products WHERE id = ?";
       Product product = null;
       try(
          Connection conn = getConnection();
          PreparedStatement pstmt = conn.prepareStatement(sql)
          ){
          pstmt.setInt(1,id);
          try(
             ResultSet rst = pstmt.executeQuery()
             ){
            if(rst.next()){
             product = new Product(rst.getInt("id"), rst.getString("pname"),
                  rst.getString("brand"), rst.getDouble("price"));
             }
          }
        }catch(SQLException se){
          return null;
        }
```

```
        return product;
    }

    //查询所有商品信息
    public List<Product> findAllProduct ()throws SQLException{
        Product product = null;
        ArrayList<Product> productList = new ArrayList<Product>();
        String sql = "SELECT * FROM products";
    try(
        Connection conn = getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql);
        ResultSet rst = pstmt.executeQuery()
        ){
    while(rst.next()){
            product = new Product(rst.getInt("id"),
                    rst.getString("name"),
                    rst.getString("brand"), rst.getDouble("price"));
            productList.add(product);
        }
        return productList;
    }catch(SQLException e){
        e.printStackTrace();
        return null;
    }
  }
}
```

该类没有给出修改记录和删除记录的方法，读者可以自行补充完整。

5.5.3 使用 DAO 对象

下面的 add-product.jsp 页面通过一个表单提供向数据库中插入的数据，代码见清单 5.13。

清单 5.13 add-product.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html><head><title>添加商品</title></head>
<body>
<font color=red>${result}</font>
<p>请输入一条商品记录</p>
<form action="add-product" method="post">
 <table>
   <tr><td>商品号:</td><td><input type="text" name="id"></td></tr>
   <tr><td>商品名:</td><td><input type="text" name="name"></td></tr>
   <tr><td>品牌:</td><td><input type="text" name="brand"></td></tr>
   <tr><td>价格:</td><td><input type="text" name="price"></td></tr>
   <tr><td><input type="submit" value="确定"></td>
      <td><input type="reset" value="重置"></td>
  </tr>
</table>
</form>
</body></html>
```

清单 5.14 中的 AddProductServlet 使用了 DAO 对象和持久化对象，通过 JDBC API 实现将数据插入数据库中。

清单 5.14 AddProductServlet.java

```
package com.boda.controller;

import java.io.*;
```

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import com.boda.domain.Product;
import com.boda.dao.ProductDao;
import com.boda.dao.ProductDaoImpl;
import jakarta.servlet.annotation.WebServlet;

@WebServlet(name = "addProductServlet", value = "/add-product")
public class AddProdcutServlet extends HttpServlet{
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                    throws ServletException, IOException{
      ProductDao dao = new ProductDaoImpl();
      Prodcut prodcut = null;
      String message = null;
      try{
         product = new Product(
              Integer.parseInt(request.getParameter("id")),
              request.getParameter("name"),
              request.getParameter("brand"),
              Double.parseDouble(request.getParameter("price"));
         int success = dao.addProduct(product);
         if(success == 1){
              message = "<li>成功插入一条记录!</li>";
         }else{
              message = "<li>插入记录错误!</li>";
          }
        }catch(Exception e){
           System.out.println(e);
           message = "<li>插入记录错误!</li>" + e;
        }
        request.setAttribute("result",message);
        RequestDispatcher rd =
           getServletContext().getRequestDispatcher("/add-product.jsp");
        rd.forward(request,response);
      }
}
```

该程序首先从请求对象中获得请求参数并进行编码转换，创建一个 Product 对象，然后调用 ProductDao 对象的 addProduct()方法将商品对象插入数据库中，最后根据该方法执行结果将请求再转发到 JSP 页面。

访问 add-product.jsp 页面，输入商品信息，单击“确定”按钮可以将商品信息插入数据库，如图 5-7 所示。

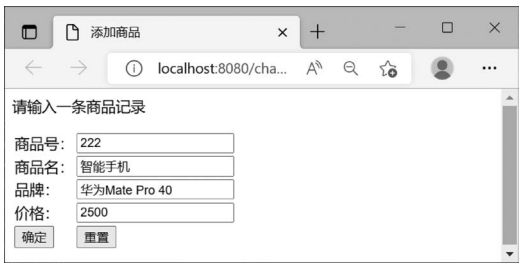

图 5-7　add-product.jsp 页面的运行结果

本章小结

Java 程序通过 JDBC API 访问数据库。JDBC API 定义了 Java 程序访问数据库的接口。访问数据库首先应该建立到数据库的连接，传统的方法是通过 DriverManager 类的 getConnection()建立连接对象。使用这种方法很容易产生性能问题，因此从 JDBC 2.0 开始提供了通过数据源建立连接对象的机制。

通过 PreparedStatement 对象可以创建预处理语句对象，它可以执行动态 SQL 语句。通过数据源连接数据库，首先需要建立数据源，然后通过 JNDI 查找数据源对象，建立连接对象，最后通过 JDBC API 操作数据库。

DAO 设计模式是数据库访问的标准方法，它是一种面向接口的设计方法，实现数据访问逻辑，通常为每个实体类设计一个接口和一个实现类。

练习与实践

扫一扫

习题

扫一扫

自测题