Java Web 编程技术

题解与实验指导

第 1 章　Java Web 技术概述

1.1　本章要点

Java Web 技术是用 Java 技术来解决 Web 相关领域问题的技术总和，通常包括 Web 服务器端技术和 Web客户端技术两部分。

Web 是 Internet 的最重要应用，它是基于客户/服务器（C/S）的一种体系结构，客户在计算机上使用浏览器向 Web 服务器发出请求，服务器响应客户请求，向客户回送所请求的网页，客户在浏览器窗口上显示网页的内容。

服务器端技术包括 HTTP、Web 服务器和 Web 容器、动态 Web 页面技术等。客户端技术包括 HTML 和 XML、CSS、JavaScript 以及 Ajax 等，

超文本传输协议（Hypertext Transfer Protocol，HTTP）是Web使用的协议。它是一个基于请求-响应的无状态协议，这种请求-响应的过程如图1-1所示。

图 1-1　HTTP 请求-响应示意图

客户首先通过浏览器程序建立到 Web 服务器的连接并向服务器发送 HTTP 请求消息。服务器接收到客户的请求后，对请求进行处理，然后向客户发送回 HTTP 响应。客户接收服务器发送的响应消息，对消息进行处理并关闭连接。

HTML 称为超文本标记语言，它用来编写 Web 文档。文档是由一些标签（tag）组成的文本文件，标签标识了内容和类型，描述了如何格式化文档。Web 浏览器通过解析这些标签显示文档。

CSS 是能够真正做到网页表现与内容分离的一种样式设计语言。JavaScript 是一种广泛用于客户端Web开发的脚本语言，常被用来编写校验表单数据的代码等。

Web 文档是一种重要的 Web 资源，它通常是使用某种语言（如 HTML，JSP 等）编写的页面文件，因此也称为 Web 页面。Web 文档又分为静态文档和动态文档。

Servlet 是用 Servlet API 以及相关的类编写的 Java 程序，这种程序运行在 Web 容器中，主要用来扩展 Web 服务器的功能。JSP（JavaServer Pages）页面是在 HTML 页面中嵌入 JSP 元素的页面，这些元素称为 JSP 标签。JSP 元素具有严格定义的语法并包含完成各种任务的语法元素，比如声明变量和方法、JSP 表达式、指令和动作等。

Servlet 和 JSP 需要运行在 Servlet 容器（或 Web 容器）中，它可以是 Web 服务器的一

个模块，也可以是单独的模块。Tomcat 服务器是最常用的 Web 容器，最新版本 Tomcat 7.0.39 实现了 Servlet 3.0 和 JSP 2.2 的规范，另外它本身具有作为 Web 服务器运行的能力，

在 Web 应用开发中建议采用 MVC（Model-View-Controller）设计模式，该设计模式将 Web 组件分为模型（Model）、视图（View）和控制器（Controller）三种类型，其中控制器使用 Servlet 或 Filter 实现、模型使用 JavaBeans 或 JOPO 实现，视图使用 JSP 页面实现。使用 MVC 设计模式可以实现表示逻辑与业务逻辑分离，从而使 Web 应用程序的开发和维护变得容易。

1.2 实验指导

【实验目的】

1. 掌握用 Eclipse 开发动态 Web 项目。
2. 掌握简单的 Servlet 和 JSP 页面的开发。

【实验内容】

实验题目 1：使用 Eclipse IDE 创建动态 Web 项目。

（1）在 Eclipse 中配置 Tomcat。选择 Window→Preferences 命令，在打开的对话框左边列表框中选择 Server 节点中的 Runtime Environments。单击窗口右侧的 Add 按钮，打开 New Server Runtime Environmen 对话框，在该对话框中可选择服务器的类型和版本，这里使用的是 Apache Tomcat v 7.0。

（2）在 Eclipse 中选择 File→New→Dynamic Web Project，打开新建动态 Web 项目对话框。在 Project name 文本框中输入项目名，如 helloweb，下面的选项采用默认值即可。

（3）单击 Next 按钮，打开 Web Module 对话框，在这里需要指定 Web 应用程序上下文根目录名称和 Web 内容存放的目录，这里采用默认值，选中 Generate web.xml deployment descriptor 复选框，由 Eclipse 产生部署描述文件，如图 1-2 所示。最后单击 Finish 按钮，结束项目的创建。

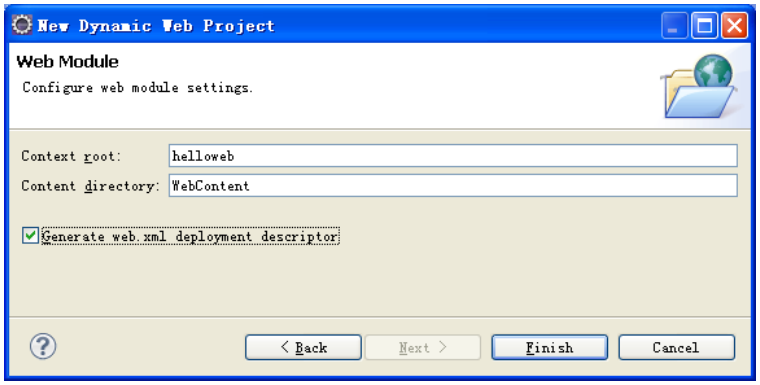

图 1-2　Web Module 对话框

实验题目 2：使用 Eclipse IDE 创建和运行 Servlet。

（1）右击 helloweb 项目，从弹出菜单中选择 New→Servlet，打开 Create Servlet 对话框。在 Java package 文本框中输入包名，如 com.demo，在 Class name 文本框中输入类名 HelloServlet。

（2）单击 Next 按钮，进入下一对话框。这里需要指定 Servlet 在部署描述文件中的信息，主要包括 Servlet 名称和 URL 映射名的定义。这里，将 Servlet 名称修改为 helloServlet，将 URL 映射名称修改为/helloServlet.do。

（3）单击 Next 按钮，在出现的对话框中指定 Servlet 实现的接口以及自动生成的方法。最后单击 Finish 按钮，Eclipse 将生成该 Servlet 的部分代码并在编辑窗口中打开，修改后完整代码如下。

```
package com.demo;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;

@WebServlet(name = "helloServlet", urlPatterns = { "/helloServlet.do" })
public class HelloServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body><head><title>当前时间</title></head>");
        out.println("<h3>Hello,World!</h3>");
        out.println("现在的时间是:"+new java.util.Date());
        out.println("</body>");
        out.println("</html>");
    }
}
```

（4）在 Eclipse IDE 中右击代码部分，在弹出菜单中选择 Run As→Run on Server 即可执行该 Servlet。

实验题目 3：使用 Eclipse IDE 创建一个 JSP 页面。

（1）右击 helloweb 项目的 WebContent 节点，从弹出菜单中选择 New→JSP File，打开 New JSP File 对话框。选择 JSP 页面存放的目录，这里为 WebContent。在 File name 文本框中输入文件名 hello.jsp。

（2）单击 Next 按钮，打开选择 JSP 模板对话框，从模板列表中选择要使用的模板，这里选择 New JSP File(html)模板，然后单击 Finish 按钮。Eclipse 创建 hello.jsp 页面并在工作

区中打开该文件，可以在<body>标签中插入代码。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<html>
<head><title>简单的JSP页面</title></head>
<body>
    <h1>Hello,World!</h1>
    现在的时间是：<%=new java.util.Date() %>
</body>
</html>
```

（3）要运行 JSP 页面，在 JSP 页面编辑区中右击鼠标，在打开的菜单中选择 Run As →Run on Server 即可执行该 JSP 页面。

【思考题】

1. 如何在 Eclipse 中配置 Tomcat？
2. 如何在 Eclipse 中创建动态 Web 项目？如何创建和运行 Servlet 和 JSP？

1.3 习题解析

1. 下面哪个是 URL?(　　)

A. www.tsinghua.edu.cn　　B. http://www.baidu.com

C. 121.52.160.5　　D. /localhost:8080/webcourse

【答】 B。A 是一个主机名，C 是一个 IP 地址，D 是一个 URI。

2. 要在页面中导入 css/layout.css 样式单文件，下面哪两个是正确的？（　　）。

A. <link type="text/css" href="css\layout.css" rel="stylesheet" />

B. <script type="text/javascript" src="css\layout.css "></script>

C. <style type="text/css">@import url(css/layout.css);</style>

D. <meta http-equiv="Content-Type" content=" css\layout.css; charset=UTF-8">

【答】 A、C。

3. 若访问的资源不存在，服务器向客户发送一个错误页面，该页面中显示的 HTTP 状态码是（　　）。

A. 500　　B. 200　　C. 404　　D. 401

【答】 C。状态码 500 表示服务器内部错误，403 表示页面禁止访问，200 表示请求成功。

4. 下面哪个不是服务器页面技术？（　　）。

A. JSP　　B. ASP　　C. PHP　　D. JavaScript

【答】 D。JavaScript 是一种脚本语言，可用来编写脚本代码实现客户端动态页面技术。JSP、ASP 和 PHP 都是服务器端动态页面技术。

5. Servlet 必须在什么环境下运行？（　　）

A. 操作系统　　B. Java 虚拟机

C. Web 容器　　D. Web 服务器

【答】　C。Servlet 必须在 Web 容器或 Servlet 容器中运行。

6. MVC 设计模式不包括下面哪个？（　　）

A. 模型　　B. 视图

C. 控制器　　D. 数据库

【答】　D。MVC 设计模式是模型（Model）、视图（View）和控制器（Controller）。

7. 什么是 URL，什么是 URI，它们都由哪几部分组成？URL 与 URI 有什么关系？

【答】　URL 称为统一资源定位符，URL 通常由 4 部分组成：协议名称、主机的 DNS 名、可选的端口号和资源的名称。URI 称为统一资源标识符，是以特定语法标识一个资源的字符串。URI 由模式和模式特有的部分组成，它们之间用冒号隔开，一般格式如下：

```
schema:schema-specific-part
```

URI 是 URL 和 URN 的超集。

8. 下面是 URL 的为（　　），是 URI 的为（　　），是 URN 的为（　　）。

① http://www.myserver.com/hello

② files/sales/report.html

③ ISBN:1-930110-59-6

【答】　①是 URL，①和②都是 URI，③是 URN

9. 动态 Web 文档技术有哪些？服务器端动态文档技术和客户端动态文档技术有何不同？

【答】　动态 Web 文档技术包括服务器端动态文档技术和客户端动态文档技术，前者包括 CGI 技术、服务器扩展技术和 HTML 页面中嵌入脚本技术。其中 HTML 页面中嵌入脚本技术包括 ASP、PHP 和 JSP 技术。

最流行的客户端动态文档技术是在 HTML 页面中嵌入 JavaScript 脚本代码。使用 JavaScript 可以设计交互式页面。与服务器端动态文档不同，JavaScript 脚本是在客户端执行的。

10. 什么是 Servlet？什么是 Servlet 容器？它的主要作用是什么？

【答】　Servlet 是用 Servlet API 开发的 Java 程序，它运行在 Servlet 容器中。Servlet 容器是运行 Servlet 的软件，主要用来扩展 Web 服务器的功能。

11. 什么是 MVC 设计模式，它有什么优点？

MVC 设计模式称为模型-视图-控制器，在这种模式中，将 Web 组件分为模型（Model）、视图（View）和控制器（Controller），每种组件完成各自的任务。该模式的最大优点是将业务逻辑和数据访问从表示层分离出来。

第 2 章　Servlet 技术模型

2.1　本章要点

Servlet API 定义了若干接口和类，它是 Java Web 应用开发的基础，它由四个软件包组成：javax.servlet、javax.servlet.http、javax.servlet.annotation 和 javax.servlet.descriptor。

Servlet接口是Servlet API中的核心接口，每个Servlet必须直接或间接实现该接口,该接口定义了init()、service()和destroy()生命周期方法以及getServletInfo()与getServletConfig()。

HttpServlet类用来实现针对HTTP协议的Servlet，它扩展了GenericServlet抽象类，我们编写的Servlet通常继承HttpServlet类。在HttpServlet中针对不同的HTTP请求方法定义了不同的处理方法，如处理GET请求的doGet()格式如下：

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
```

该方法的两个参数一个是请求对象，一个是响应对象。HttpServletRequest 接口对象是请求对象，使用它可以检索客户请求信息，如使用 getParameter()可以获取请求参数、getMethod()可以获取请求的 HTTP 方法（如 GET 或 POST）、getRequestURI()返回请求 URI 等。

HttpServletResponse 接口对象是响应对象，通过它可向客户端发送响应消息，如 getWriter()返回 PrintWriter 对象，它可以向客户发送文本数据，setContentType()设置响应的内容类型，setHeader()设置响应头，sendRedirect()响应重定向等。

在 Servlet 初始化时，容器将调用 init(ServletConfig)，并为其传递一个 ServletConfig 对象，该对象称为 Servlet 配置对象，使用该对象可以获得 Servlet 初始化参数、Servlet 名称、ServletContext 对象等。

在客户端发生下面的事件，浏览器就向 Web 服务器发送一个 HTTP 请求。

- 用户在浏览器的地址栏中输入URL并按回车键。
- 用户点击了HTML页面中的超链接。
- 用户在HTML页面中添写一个表单并提交。

如果需要从 Servlet 中将请求转发到其他资源（Servlet 或 JSP），可以通过请求对象的 getRequestDispatcher()得到 RequestDispatcher 对象，然后调用它的 forward()转发请求。请求对象是一个作用域对象，通过它的 setAttribute()将一个对象作为属性存储到请求对象中，然后可以在请求作用域的其他资源中使用 getAttribute()检索出属性。

Web 应用程序具有严格定义的目录结构，每个 Web 应用程序都有一个文档根目录，目录名就是 Web 应用程序名，该目录中可以定义其他目录存放应用程序资源。每个 Web 应

用程序在它的根目录中都必须有一个 WEB-INF 目录，该目录中主要存放供服务器访问的资源，其中 classes 目录存放支持该 Web 应用程序的类文件，lib 目录存放 Web 应用程序使用的全部库文件，该目录下还应该有一个 web.xml 文件，它叫 Web 应用程序部署描述文件，其中可以定义有关资源（如 Servlet、监听器、安全性约束等）。

在 Servlet 3.0 的 javax.servlet.annotation 包中定义了若干注解，使用@WebServlet 注解可以定义 Servlet。

下面一行是为 HelloServlet 添加的注解。

```
@WebServlet(name="HelloServlet",urlPatterns={"/hello.do"})
```

这里，name 元素指定 Servlet 名称，urlPatterns 元素指定 URL。该注解还可包含其他元素。注解在应用程序启动时被 Web 容器处理，容器根据具体的元素配置将相应的类部署为 Servlet。

2.2 实验指导

【实验目的】

1. 掌握通过 Servlet 处理表单数据。
2. 掌握通过 Servlet 处理业务逻辑，实现请求转发等。
3. 掌握使用 Servlet 3.0 的新增功能实现文件上传。

【实验内容】

实验题目 1：处理表单数据。建立下面名为 register.jsp 页面，其运行结果如图 2-1 所示。

图 2-1　register.jsp 页面运行结果

register.jsp 页面代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>注册页面</title>
</head>
<body>
<h3>用户注册</h3>
<form action="register.action" method="post">
<table>
<tr><td>用户名：</td>
    <td><input type="text" name="username" size="15"></td></tr>
<tr><td>密码：</td>
    <td><input type="password" name="password" size="16"></td></tr>
<tr><td>性别：</td>
    <td><input type="radio" name ="sex" value="male">男
      <input type="radio" name ="sex" value="female">女</td></tr>
<tr><td>年龄：</td>
  <td><input type="text" name="age" size="5"></td></tr>
<tr><td>兴趣：</td>
<td><input type="checkbox" name="hobby" value="read">文学
     <input type="checkbox" name="hobby" value="sport">体育
     <input type="checkbox" name="hobby" value="computer">电脑</td></tr>
<tr><td>学历：</td>
<td><select name="education">
     <option value="bachelor">学士</option>
     <option value="master">硕士</option>
     <option value="doctor">博士</option>
     </select>
   </td></tr>
<tr><td>邮件地址：</td><td><input type="text" name="email"
                          size="20"></td></tr>
<tr><td>简历：</td><td><textarea rows="5" cols="30"></textarea></td></tr>
<tr><td><input type="submit" name="submit" value="提交"></td>
<td><input type="reset" name="reset" value="重置"></td></tr>
<table>
</form>
</body>
</html>
```

编写 FormServlet 读取请求参数并显示用户输入信息，运行结果如图 2-2 所示。

```
package com.lab;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet(name = "FormServlet", urlPatterns = { "/register.action" })
public class FormServlet extends HttpServlet {
    private static final long serialVersionUID = 54L;
    private static final String TITLE = "用户信息";

    @Override
    public void doPost(HttpServletRequest request,
           HttpServletResponse response)
           throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       out.println("<html>");
       out.println("<head>");
       out.println("<title>" + TITLE + "</title></head>");
       out.println("</head>");
       out.println("<body><h1>" + TITLE + "</h1>");
       out.println("<table>");
       out.println("<tr><td>用户名</td>");
       String username = request.getParameter("username");
        out.println("<td>" + username + "</td></tr>");
       out.println("<tr><td>密码:</td>");
       out.println("<td>" + request.getParameter("password")
              + "</td></tr>");
       out.println("<tr><td>性别:</td>");
       out.println("<td>" + request.getParameter("sex")
              + "</td></tr>");
       out.println("<tr><td>年龄:</td>");
       out.println("<td>" + request.getParameter("age")
              + "</td></tr>");
       out.println("<tr><td>爱好:</td>");
       out.println("<td>");
       String[] hobbys = request.getParameterValues("hobby");
       if(hobbys!=null){
        for(String hobby:hobbys){
            out.println(hobby + "<br/>");
        }
       }
       out.println("</td></tr>");
       out.println("<tr><td>学历:</td>");
       out.println("<td>" + request.getParameter("education")
              + "</td></tr>");
       out.println("<tr><td>邮件地址:</td>");
       out.println("<td>" + request.getParameter("email")
              + "</td></tr>");
       out.println("<tr><td>简历:</td>");
       out.println("<td>" + request.getParameter("resume")
              + "</td></tr>");
       out.println("</table>");
       out.println("<div style='border:1px solid #ddd;" +
            "margin-top:40px;font-size:90%'>");
```

```
            out.println("用户信息<br/>");
            Enumeration<String> parameterNames = request
                    .getParameterNames();
            while (parameterNames.hasMoreElements()) {
                String paramName = parameterNames.nextElement();
                out.println(paramName + ": ");
                String[] paramValues = request
                        .getParameterValues(paramName);
                for (String paramValue : paramValues) {
                    out.println(paramValue + "<br/>");
                }
            }
            out.println("</div>");
            out.println("</body>");
            out.println("</html>");
        }
    }
```

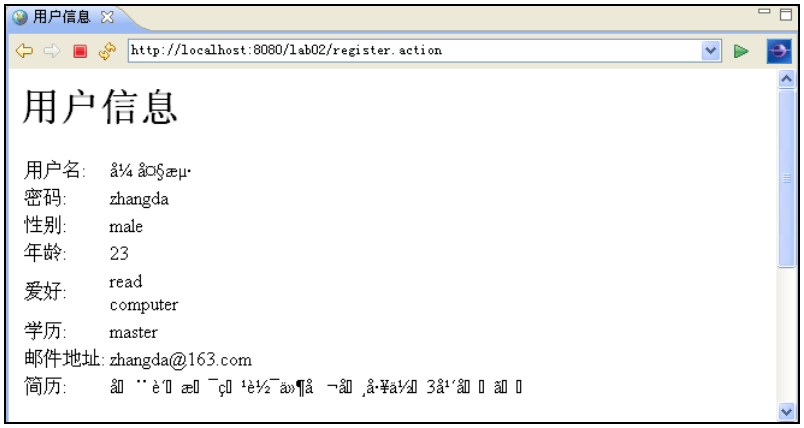

图 2-2　FormServlet 运行结果

注意，这里显示的结果中“用户名”和“简历”的值为乱码，请使用下面方法将从客户端读取的中文重新编码。

```
String username = request.getParameter("username");
username = new String(username.getBytes("ISO-8859-1"),"UTF-8");
```

实验题目 2：修改 FormServlet 类，去掉其中的注解，使用部署描述文件 web.xml 定义 Servlet，重新运行程序。

实验题目 3：简单的综合应用，实现 Student 对象的简单管理。学生信息使用 Student 类对象存放，该类包含 4 个成员，如下所示：

```
public class Student{
   private String id;
   private String name;
   private int age;
   private String qq;
```

```
    // 无参构造方法和带参数构造方法
    // 成员的 setter 方法和 getter 方法
}
```

该应用包含一个 StudentServlet，首先显示 Student 信息，如图 2-3 所示。StudentServlet 类的主要代码如下。

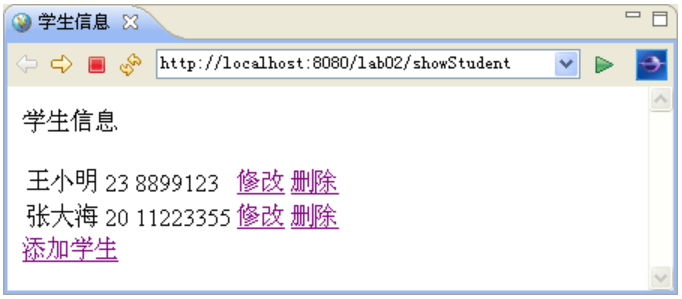

图 2-3　StudentServlet 运行结果

```
@WebServlet(name = "/StudentServlet",
          urlPatterns={"/showStudent","/editStudent","/updateStudent",
          "/insertStudent","/deleteStudent","/addStudent"})
public class StudentServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private List<Student> students = new ArrayList<Student>();
    // 初始化方法创建两个 Student 对象并存储在 List 对象中
    public void init() throws ServletException{
        Student stud1 = new Student("101","王小明",23,"8899123");
        Student stud2 = new Student("102","张大海",20,"11223355");
        students.add(stud1);
        students.add(stud2);
    }
    // 当接收到 GET 请求时，根据请求 URI 不同执行不同的方法
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        String uri = request.getRequestURI();
        if(uri.endsWith("showStudent")){
            showStudent(response);
        }else if(uri.endsWith("editStudent")){
            editStudent(request,response);
        }else if(uri.endsWith("deleteStudent")){
            deleteStudent(request,response);
        }else if(uri.endsWith("addStudent")){
            addStudent(request,response);
        }
    }
    // 根据 id 返回一个 Student 对象
    private Student getStudent(String id){
        for(Student student:students){
            if (student.getId().equals(id)){
                return student;
            }
```

```
        }
        return null;
    }
    // 显示学生信息方法
    public void showStudent(HttpServletResponse response)
                        throws ServletException,IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>学生信息</title></head>");
        out.println("<body><p>学生信息</p>");
        out.println("<table>");
        for(Student student:students){
           out.println("<tr><td>" + student.getName() + "</td><td>"
                    + student.getAge() + "</td><td>"
                    + student.getQq()+"</td>"
                    + "<td><a href='editStudent?id="+student.getId()
                     + "'>修改 </a></td>"
                    + "<td><a href='deleteStudent?id="+student.getId()
                    +"'>删除</a></td></tr>");
        }
        out.println("</table>");
        out.println("<a href='addStudent'>添加学生 </a>");
        out.println("</body></html>");
    }
```

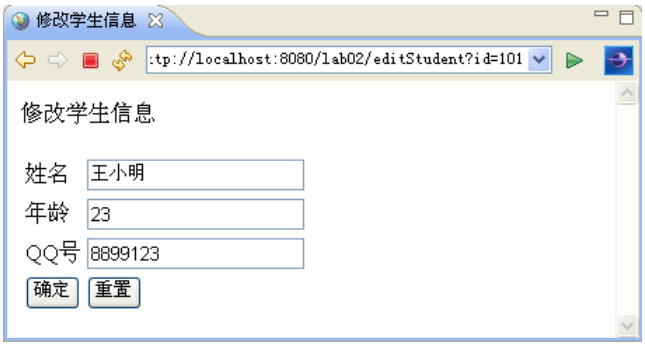

图 2-4　修改学生界面

```
// 修改学生信息方法
public void editStudent(HttpServletRequest request,
                        HttpServletResponse response)
                          throws ServletException, IOException{
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    // 根据请求参数找到学生信息并显示在表单中
    String id = request.getParameter("id");
    Student student = getStudent(id);
    if(student!=null){
        out.println("<html><head><title>修改学生信息</title></head>");
        out.println("<body><p>修改学生信息</p>");
        out.println("<form method='post' action='updateStudent'>");
```

```
            out.println("<input type='hidden' name='id'
                            value='"+id+"'/>");
            out.println("<table><tr><td>姓名</td><td><input type='text'
                      name='name' value='"+student.getName()+"'/></td>");
            out.println("<tr><td>年龄</td><td><input type='text'
                      name='age' value='"+student.getAge()+"'/></td>");
            out.println("<tr><td>QQ 号</td><td><input type='text' name='qq'
                      value='"+student.getQq()+"'/></td>");
            out.println("<tr><td><input type='submit' value='确定'/>
                     </td>"+ "<td><input type='reset' value='重置'/></td>");
            out.println("</table>");
        }else{
                out.println("没有找到学生！");
        }
    }
```

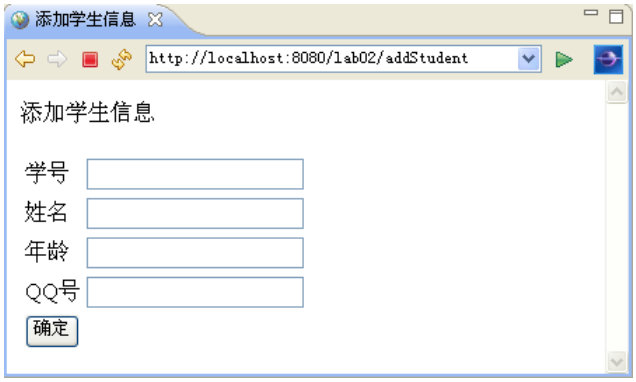

图 2-5　添加学生界面

```
// 添加学生信息表单
public void addStudent(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException{
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>添加学生信息</title></head>");
    out.println("<body><p>添加学生信息</p>");
    out.println("<form method='post' action='insertStudent'>");
    out.println("<table><tr><td>学号</td><td><input type='text'
                      name='id'/></td>");
    out.println("<tr><td>姓名</td><td><input type='text'
                      name='name'/></td>");
    out.println("<tr><td>年龄</td><td><input type='text'
                      name='age'/></td>");
    out.println("<tr><td>QQ 号</td><td><input type='text'
                      name='qq'/></td>");
    out.println("<tr><td><input type='submit' value='确定'/></td>");
    out.println("</table>");
}
// 根据请求 URI 执行修改学生或插入学生
public void doPost(HttpServletRequest request,
```

```
                        HttpServletResponse response)
                         throws ServletException, IOException {
    String uri = request.getRequestURI();
    if(uri.endsWith("updateStudent")){
        String id = request.getParameter("id");
        Student student = getStudent(id);
        if(student!=null){
            // 将请求参数编码转换为 UTF-8 编码
            String name = new String(
               request.getParameter("name").getBytes("iso-8859-1"),
               "UTF-8");
            student.setName(name);
            student.setAge(Integer.parseInt(
                            request.getParameter("age")));
            student.setQq(request.getParameter("qq"));
        }
   }else if(uri.endsWith("insertStudent")){
        String id = request.getParameter("id");
        String name = new String(
                request.getParameter("name").getBytes("iso-8859-1"),
                  "UTF-8");
        int age = Integer.parseInt(request.getParameter("age"));
        String qq = request.getParameter("qq");
        Student student = new Student(id,name,age,qq);
        students.add(student);
    }
    showStudent(response);
}
```

实验题目 4：编写一个程序实现文件上传功能。假设学生上传作业，需要指定学号和上传的文件，fileUpload.jsp 代码如下。

```
<%@ page contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<html><head><title>作业上传</title></head>
<body>
<p>作业上传</p>
<form action="fileUpload.action" enctype="multipart/form-data"
      method="post">
  学号：<input type="text" name="snumber"/><br/>
  文件： <input type="file" name="filename"/><br/>
    <input type="submit" value="提交"/>
</form>
</body>
</html>
```

实现文件上传功能的 FileUploadServlet 类代码如下：

```
package com.lab;

@WebServlet(urlPatterns = { "/fileUpload.action" })
@MultipartConfig
```

```
public class FileUploadServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                      throws ServletException,IOException {
        // 声明 Part 对象接收上传文件
        Part part = request.getPart("filename");
        String snumber = request.getParameter("snumber");
        // 将文件存储到 student 目录中
        String path = this.getServletContext().getRealPath("/WEB-INF");
        path = path + "\\student\\" +snumber;
        File f = new File(path);
        if( !f.exists()){  // 若目录不存在，则创建目录
           f.mkdirs();
        }
        String header = part.getHeader("content-disposition");
        // 得到文件名
        String fname = header.substring(header.lastIndexOf("\\")+1,
                       header.length()-1);
        part.write(path + "\\"+ fname);
        // 返回客户信息
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<br/>你的学号是: " + snumber);
        out.print("<br/>上传的文件名为: " + fname);
        out.print("<br/>文件大小: " + part.getSize());
    }
}
```

该 Servlet 将上传的文件存储到 WEB-INF\student 目录中，然后显示上传文件的信息。

【思考题】

1. 为实验题目 1 的 register.jsp 页面编写校验代码，使用 JavaScript 脚本为实现对其中的表单域校验，通过警告框为用户显示信息。
2. 如何理解 Servlet 的生命周期，如何处理 HTTP 请求和响应？
3. 在 Servlet 中如何实现请求转发？它与响应重定向有何异同？
4. 如何实现文件上传？

2.3 习题解析

1. 下面哪个方法不是 Servlet 生命周期方法？(　　)

A. public void destroy()　　B. public void service()

C. public ServletConfig getServletConfig()　　D. public void init()

【答】 C。Servlet 生命周期方法包括 init()、service()和 destroy()。

2. 要使向服务器发送的数据不在浏览器的地址栏中显示，应该使用什么方法？（）

A. POST　　B. GET　　C. PUT　　D. HEAD

【答】 A。

3. 考虑下面的HTML页面代码：

请求

当用户在显示的超链接上点击时将调用HelloServlet的哪个方法？

A. doPost()　　B. doGet()

C. doForm()　　D. doHref()

【答】　B。点击超链接向服务器发送 GET 请求。

4. 有一个URL，http://www.myserver.com/hello?userName=John，问号后面的内容称为什么？

【答】　查询串。通过查询串可以向服务器传递一个或多个请求参数。

5. HTTP请求结构由哪几部分组成？请求行由哪几部分组成？

【答】　HTTP 请求结构由请求行、请求头、空行和请求数据组成。请求行由方法名、请求资源的 URI 和使用的 HTTP 版本 3 部分组成。

6. HTTP响应结构由哪几部分组成？状态行由哪几部分组成？

【答】　HTTP 响应结构由状态行、响应头和响应数据 3 部分组成。状态行由 HTTP 版本、状态码和简短描述 3 部分组成。

7. GET请求和POST请求有什么异同？

【答】　GET 请求主要用来从服务器检索资源，POST 请求主要用来向服务器发送数据。它们的详细比较请参阅教材的表 2.7。

8. 将一个Student类的对象student用名称studobj存储到请求作用域中，下面代码哪个是正确的？（　　）

A. request.setAttribute("student",studobj)　　B. request.addAttribute("student",studobj)

C. request.setAttribute("studobj",student)　　D. request.getAttribute("studobj",studend)

【答】　C。

9. 使用RequestDispatcher的forward()转发请求和使用响应对象的sendRedirect()重定向有何异同？

【答】　用 forward()转发请求，存储在请求对象中的属性在转发到的资源中可用，用响应对象的 sendRedirect()，存储在请求对象中的属性在新的资源中不可用，但存储在会话对象中的属性可用。

10. 在Servlet中如果需要获得一个页面的表单中的请求参数，又不知道参数名时如何做？

【答】　可先通过请求对象的 getParameterNames()得到 Enumeration 对象，然后在其上得到每个请求参数名，再通过 getParameter()得到请求参数值。

11. 如果需要向浏览器发送一个GIF文件，何时调用response.getOutputStream()？

A. 在调用response.setContentType("image/gif")之前

B. 在调用response.setContentType("image/gif")之后

C. 在调用response.setDataType("image/gif")之前

D. 在调用response.setDataType("image/gif")之后

【答】 B。先设置响应的内容类型，后获得响应的输出流。

12. 如果需要向浏览器发送Microsoft Word文档，应该使用下面哪个语句创建out对象？（ ）

A. PrintWriter out = response.getServletOutput();

B. PrintWriter out = response.getPrintWriter();

C. OutputStream out = response.getWriter();

D. OutputStream out = response.getOutputStream();

【答】 D。

13. 完成下列功能需使用哪个方法？

① 向输出中写HTML标签。（ ）

② 指定响应的内容为二进制文件。（ ）

③ 向浏览器发送二进制文件。（ ）

④ 向响应中添加响应头。（ ）

⑤ 重定向浏览器到另一个资源。（ ）

下面是选项：

A. 使用HttpServletResponse的sendRedirect(String urlstring)。

B. 使用HttpServletResponse的setHeader("name", "value")。

C. 使用ServletResponse的getOutputStream()，然后使用OutputStream的write(bytes)。

D. 使用ServletResponse的setContentType(String contenttype)。

E. 首先使用ServletResponse的getWriter()获得PrintWriter对象，然后调用PrintWriter的print()。

【答】 ① E ② D ③ C ④ B ⑤ A

14. 假设客户使用 URL http://www.hacker.com/myapp/cool/bar.do 请求一个名为“bar.do”的 Servlet，该 Servlet 中使用 senRedirect("foo/stuff.html");语句将响应重定向，则重定向后新的 URL 为：_____

如果在 Servlet 中使用 senRedirect("/foo/stuff.html");语句将响应重定向，则重定向后新的 URL 为：_____

【答】 http://www.hacker.com/myapp/foo/stuff.html

http://www.hacker.com/foo/stuff.html

15. 通过哪两种方法可以获得 ServletConfig 对象？

【答】 覆盖 Servlet 的 init(ServletConfig config)，然后把容器创建的 ServletConfig 对象保存到一个成员变量中，另一种方法是在 Servlet 中直接使用 getServletConfig()获得 ServletConfig 对象。

16. 在部署描述文件中<servlet>元素的子元素<load-on-startup>的功能是什么？使用注

解如何指定该元素？

【答】 在 web.xml 文件中如果将<servlet>元素的子元素<load-on-startup>设置为一个正整数，则在应用程序启动时载入该 Servlet，否则在该 Servlet 被请求时才载入。若使用注解实现该功能，需通过@WebServlet 的 loadOnStartup 元素指定。

17. 完成下面的综合应用，其运行结果如图 2-6 和 2-7 所示。

（1）创建一个名为 input.jsp 的 JSP 页面，其中包括一个表单，表单中包含两个文本域，分别供用户输入学号和姓名，该页面也包含提交和重置按钮。

（2）定义一个名为 com.demo.Student 类，其中包括学号 sno 和姓名 name 两个 private 的成员变量，定义访问和修改 sno 和 name 的方法。

（3）编写名为 FirstServlet 的 Servlet，要求当用户在 input.jsp 中输入信息后点击“登录”按钮，请求 FirstServlet 对其处理。在 FirstServlet 中使用表单传递的参数（学号和姓名）创建一个 Student 对象并将其作为属性存储在请求对象中，然后通过请求对象的 getRequestDispatcher()获得 RequestDispatcher()对象，将请求转发到 SecondServlet。

（4）在 SecondServlet 中取出请求对象上存储的 Student 对象，并显示输出该学生的学号和姓名。在 SecondServlet 的输出中应该包含一个超链接，点击该连接可以返回 input.jsp 页面。

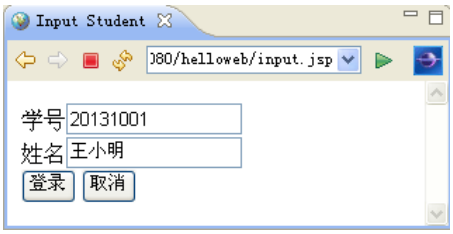

图 2-6 input.jsp 页面显示结果

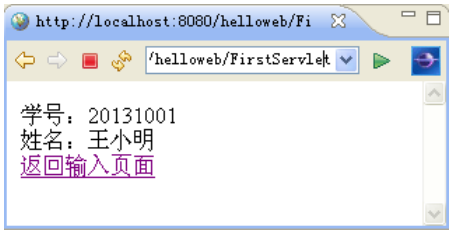

图 2-7 SecondServlet 显示结果

参考答案如下：

input.jsp 页面主要代码如下：

```
<body>
<form action="FirstServlet" method="post">
   学号<input type="text" name="sno" size="15" /><br>
   姓名<input type="text" name="sname" size="15"/><br>
   <input type="submit" value="登录" />
   <input type="reset" value="取消" />
</form>
</body>
```

Student 类代码如下：

```
package com.demo;
public class Student {
    private String sno;
    private String name;
    public Student(String sno, String name) {
      this.sno = sno;
```

```
     this.name = name;
    }
    // 省略属性的 setter 方法和 getter 方法
}
```

FirstServlet 类主要代码如下：

```
@WebServlet("/FirstServlet")
public class FirstServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                          throws ServletException, IOException {
    String sno = request.getParameter("sno");
    String name = request.getParameter("sname");
    Student student = new Student(sno,name);
    request.setAttribute("student", student);
    RequestDispatcher rd = request.getRequestDispatcher("/SecondServlet");
    rd.forward(request, response);
  }
}
```

SecondServlet 类主要代码如下：

```
@WebServlet("/SecondServlet")
public class SecondServlet extends HttpServlet {
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
    Student student= (Student)request.getAttribute("student");
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("学号："+student.getSno()+"<br>");
    out.println("姓名："+new String(
       student.getName().getBytes("iso-8859-1"),"UTF-8")+"<br>");
    out.println("<a href='input.jsp'>返回输入页面</a>");
  }
}
```

第 3 章　Servlet 容器模型

3.1　本章要点

Web容器在启动时会加载每个Web应用程序，并为每个Web应用程序创建一个唯一的ServletContext实例对象，该对象一般称为Servlet上下文对象。

在 Servlet 中可以直接调用 getServletContext()得到 ServletContext 引用。调用该对象的 getInitParameter()可以检索 Servlet 上下文初始化参数，调用 getResource()可以获得服务器上的 URL 资源，调用 getResourceAsStream()可以获得一个输入流对象，调用 log()可以登录日志，调用 getRequestDispatcher()可得到转发器对象。

ServletContext 对象也是一个作用域对象，它是 JSP 四个作用域中最大的作用域对象，在其上也可以使用 setAttribute()存储属性，在其他资源中使用 getAttribute()返回属性值。

HTTP 协议是一种无状态的协议，在 Web 应用中通常使用会话维护状态。会话是客户与服务器之间的不间断的请求响应序列。容器通过 HttpSession 接口抽象会话的概念，通过会话机制可以实现购物车应用。

使用 HttpServletRequest 请求对象的下面两个方法可以创建或返回 HttpSession 对象：

```
public HttpSession getSession(boolean create)
public HttpSession getSession()
```

HttpSession 接口中定义的 getServletContext()返回该会话所属的 ServletContext 对象，getId()返回为该会话指定的唯一标识符，invalidate()使会话对象失效。

HttpSession 接口对象也是作用域对象，在其上调用 setAttribute ()存储一个属性，在其他资源中使用 getAttribute()返回属性值。

Cookie也是实现会话管理的一种技术，Cookie是客户访问Web服务器时，服务器在客户硬盘上存放的一小段文本信息。使用Cookie类的构造方法创建Cookie对象，使用setValue()设置Cookie的值，使用getName()和getValue()返回Cookie名和Cookie值，用setMaxAge()设置Cookie在浏览器中的最长存活时间，单位为秒。

要将 Cookie 对象发送到客户端，需要调用响应对象的 addCookie()将 Cookie 添加到 Set-Cookie 响应头。要从客户端读入 Cookie，应该调用请求对象的 getCookies()，该方法返回一个 Cookie 对象的数组，对该数组迭代就可得到要找的 Cookie。

提示：在 Web 应用中实现会话管理还可以通过 URL 重写和隐藏表单域的技术，请参考有关文献。

3.2 实验指导

【实验目的】

1. 掌握 ServletContext 获得资源的方法。
2. 掌握 HttpSession 对象的应用。
3. 了解 Cookie 对象的应用

【实验内容】

实验题目 1：开发一个 FileDownloadServlet 实现文件下载，要求只有登录用户才能下载指定的文件，若用户没有登录，将请求转发到登录页面 login.jsp，用户输入用户名和密码后，控制转到 LoginServlet，如果用户合法再把控制转到 FileDownloadServlet。

FileDownloadServlet 主要代码如下：

```
package com.demo;
// 此处省略若干 import 语句

@WebServlet(urlPatterns = {"/download"})
public class FileDownloadServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                   throws ServletException,IOException {
        HttpSession session = request.getSession();
        if (session == null ||
               session.getAttribute("loggedIn") == null) {
            RequestDispatcher dispatcher =
                   request.getRequestDispatcher("/login.jsp");
            dispatcher.forward(request, response);
            return;   // 该语句是必须的
        }
        String dataDirectory = request.
               getServletContext().getRealPath("/WEB-INF/data");
        File file = new File(dataDirectory, "servlet-spec.pdf");
        if (file.exists()) {
            // 设置响应的内容类型为 PDF 文件
            response.setContentType("application/pdf");
            response.addHeader("Content-Disposition",
                   "attachment; filename=servlet-spec.pdf");
            byte[] buffer = new byte[1024];
            FileInputStream fis = null;
            BufferedInputStream bis = null;
            try {
                fis = new FileInputStream(file);
                bis = new BufferedInputStream(fis);
                OutputStream os = response.getOutputStream();
                int i = bis.read(buffer);
```

```
                while (i != -1) {
                    os.write(buffer, 0, i);
                    i = bis.read(buffer);
                }
            } catch (IOException ex) {
                System.out.println (ex.toString());
            } finally {
                if (bis != null) {
                    bis.close();
                }
                if (fis != null) {
                    fis.close();
                }
            }
        }else{
            response.setContentType("text/html;charset=UTF-8");
            PrintWriter out = response.getWriter();
            out.println("文件不存在！");
        }
    }
}
```

登录页面 login.jsp 代码如下。

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head><title>登录页面</title></head>
<body>
<form action="login" method="post">
    <table>
    <tr> <td>用户名:</td>
         <td><input type="text" name="userName"/></td>
    </tr>
    <tr> <td>密码:</td>
         <td><input type="password" name="password"/></td>
    </tr>
    <tr> <td colspan="2">
           <input type="submit" value="登录"/>
         </td>
    </tr>
    </table>
</form>
</body>
</html>
```

验证用户的 LoginServlet 代码如下。

```
package com.demo;
// 此处省略若干 import 语句

@WebServlet(urlPatterns = { "/login" })
```

```
public class LoginServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
           HttpServletResponse response) throws ServletException,
           IOException {
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");
        if (userName != null && userName.equals("member")
               && password != null && password.equals("member01")) {
           HttpSession session = request.getSession(true);
           session.setAttribute("loggedIn", Boolean.TRUE);
           response.sendRedirect("download");
           return;
        } else {
           RequestDispatcher dispatcher =
                  request.getRequestDispatcher("/login.jsp");
           dispatcher.forward(request, response);
        }
    }
}
```

当用户直接或者通过链接访问 FileDownloadServlet 时，该类首先检查用户会话中是否包含 loggedIn 属性，若无则将控制转到如图 3-1 所示的登录页面，输入用户名和密码（分别为 member 和 member01），单击“登录”按钮后，控制转到 LoginServlet，其中检查用户名和密码，如果合法则将控制转到 FileDownloadServlet 执行文件下载功能。

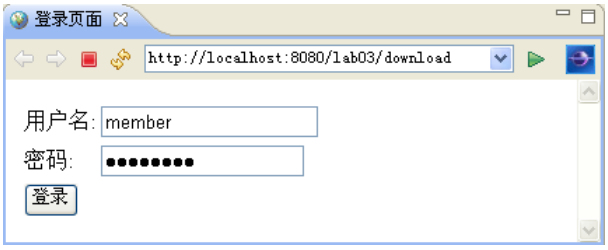

图 3-1　用户登录页面

实验题目 2：编写程序实现简单购物车应用。

该应用使用 Product 类对象存储商品信息，使用 ShoppingItem 类存储购买条目信息，ShoppingCartServlet 类是控制器类，它处理商品显示、购物车显示等功能。

存储商品信息的 Product 类代码如下：

```
package com.lab;
public class Product {
    private int id;                 // 商品编号
    private String name;            // 商品名称
    private String description;     // 商品描述
    private float price;            // 商品价格
    // 构造方法
    public Product(int id, String name, String description, float price) {
        this.id = id;
        this.name = name;
```

```
        this.description = description;
        this.price = price;
    }
    // 各属性的 setter 方法和 getter 方法
    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 getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public float getPrice() {
        return price;
    }
    public void setPrice(float price) {
        this.price = price;
    }
}
```

购物车中的每个条目使用 ShoppingItem 对象存放，代码如下。

```
package com.lab;
import java.io.Serializable;

public class ShoppingItem implements Serializable {
    private Product product;  // 商品信息
    private int quantity;     // 商品数量

    public ShoppingItem(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }
    // 属性的 getter 方法和 setter 方法
    public Product getProduct() {
        return product;
    }
    public void setProduct(Product product) {
        this.product = product;
    }
    public int getQuantity() {
```

```
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

ShoppingCartServlet 类是控制器类，它处理商品显示、购物车显示等功能，在 init()方法中创建了几种商品对象并存储到 List 中，然后使用 showPruductList()显示。

```
@WebServlet(name = "ShoppingCartServlet", urlPatterns = {
       "/products", "/viewProductDetails",
       "/addToCart", "/viewCart", "/deleteItem"  })
public class ShoppingCartServlet extends HttpServlet {
    // products 是存放所有商品的 List 对象
    private List<Product> products = new ArrayList<Product>();

    @Override
    public void init() throws ServletException {
        products.add(new Product(1, "单反相机",
               "尼康性价比最高的单反相机",4159.95F));
        products.add(new Product(2, "三星手机",
               "三星公司的最新 iPhone5 产品", 1199.95F));
        products.add(new Product(3, "笔记本电脑",
               "联想公司的新一代产品",5129.95F));
        products.add(new Product(4, "平板电脑",
               "苹果公司的新产品",1239.95F));
    }

    @Override
    public void doGet(HttpServletRequest request,
           HttpServletResponse response) throws ServletException,
           IOException {
        String uri = request.getRequestURI();
        // 根据请求 URI 决定调用哪个方法
        if (uri.endsWith("/products")) {
            showProductList(response);
        } else if (uri.endsWith("/viewProductDetails")) {
            showProductDetails(request, response);
        } else if (uri.endsWith("viewCart")) {
            showCart(request, response);
        } else if (uri.endsWith("deleteItem")) {
            deleteItem(request, response);
        }
    }
    // 该方法用于显示所有商品信息
    private void showProductList(HttpServletResponse response)
           throws IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>商品列表</title>" +
```

```
            "</head><body><p>商品列表</p>");
        out.println("<ul>");
        for (Product product : products) {
            out.println("<li>" + product.getName() + "("
                    + product.getPrice()
                    + ") (" + "<a href='viewProductDetails?id="
                    + product.getId() + "'>详细信息</a>)");
        }
        out.println("</ul>");
        out.println("<a href='viewCart'>查看购物车</a>");
        out.println("</body></html>");
    }
  …
}
```

当用户首次访问该 Servlet 时，执行 showProductList()，显示商品信息，如图 3-2 所示。

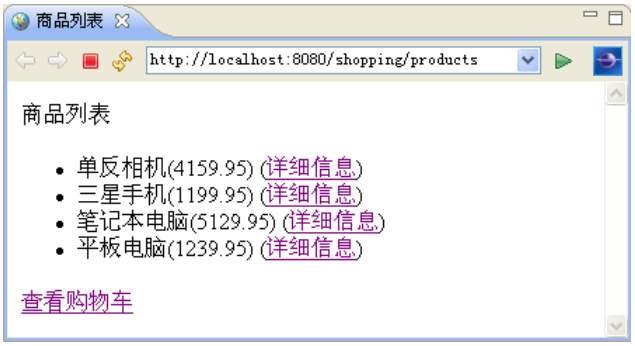

图 3-2　显示商品列表页面

当用户单击某种商品的“详细信息”链接，将进入如图 3-3 所示的页面，在该页面中输入商品数量，单击“购买”按钮，则将该商品添加到购物车中。

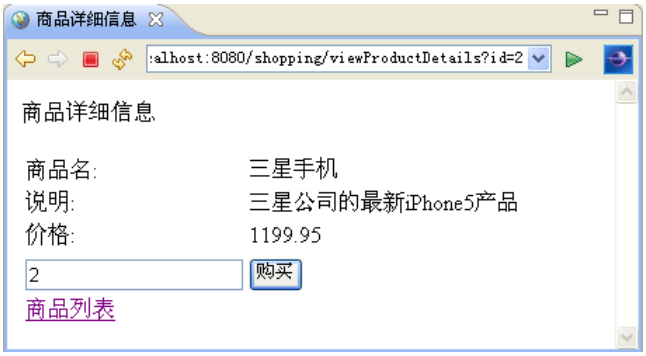

图 3-3　显示商品详细信息页面

显示商品详细信息的方法 showProductDetails()如下所示：

```
private void showProductDetails(HttpServletRequest request,
        HttpServletResponse response) throws IOException {
    response.setContentType("text/html;charset=UTF-8");
```

```
    PrintWriter out = response.getWriter();
    int productId = 0;
    try {
       productId = Integer.parseInt(
              request.getParameter("id"));
    } catch (NumberFormatException e) {
    }
    // 根据商品号返回商品对象
    Product product = getProduct(productId);

    if (product != null) {
       out.println("<html><head>"
              + "<title>商品详细信息</title></head>"
              + "<body><p>商品详细信息</p>"
              + "<form method='post' action='addToCart'>");
       // 这里使用隐藏表单域存储商品号信息
       out.println("<input type='hidden' name='id' "
            + "value='" + productId + "'/>");
       out.println("<table>");
       out.println("<tr><td>商品名:</td><td>"
              + product.getName() + "</td></tr>");

       out.println("<tr><td>说明:</td><td>"
              + product.getDescription() + "</td></tr>");
       out.println("<tr><td>价格:</td><td>"
              + product.getPrice() + "</td></tr>");

       out.println("<tr>" + "<tr>"
              + "<td><input name='quantity'/></td>"
              + "<td><input type='submit' value='购买'/>"
              + "</td>"
              + "</tr>");
       out.println("<tr><td colspan='2'>"
              + "<a href='products'>商品列表</a>"
              + "</td></tr>");
       out.println("</table>");
       out.println("</form></body>");
    } else {
       out.println("No product found");
    }
 }
// 根据商品号返回商品对象方法
private Product getProduct(int productId) {
    for (Product product : products) {
       if (product.getId() == productId) {
           return product;
       }
    }
   return null;
}
```

当用户输入购买数量，单击“购买”按钮后，请求由 doPost()处理，将用户购买的商品添加到购物车中，购物车也是用 List 实现。

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                 throws ServletException, IOException {
    // 将购买的商品添加到购物车中
    int productId = 0;
    int quantity = 0;
    try {
       productId = Integer.parseInt(
                request.getParameter("id"));
       quantity = Integer.parseInt(request
                .getParameter("quantity"));
    } catch (NumberFormatException e) { }

    Product product = getProduct(productId);
    if (product != null && quantity >= 0) {
       // 创建一个商品条目
       ShoppingItem shoppingItem = new ShoppingItem(
                    product, quantity);
       HttpSession session = request.getSession();
       // 在会话对象中查找购物车对象
       List<ShoppingItem> cart = (List<ShoppingItem>) session
                .getAttribute("cart");
       if (cart == null) {
          // 如果在会话对象上找不到购物车对象，则创建一个
          cart = new ArrayList<ShoppingItem>();
          // 将购物车对象存储到会话对象上
          session.setAttribute("cart", cart);
        }
        // 将商品添加到购物车对象中
        cart.add(shoppingItem);
     }
     showProductList(response);
 }
```

当用户向购物车中添加一些商品后，在显示商品列表页面中单击“查看购物车”链接后，显示如图 3-4 所示的页面。

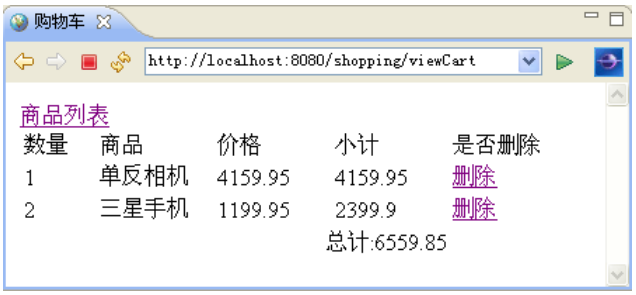

图 3-4　显示购物车信息页面

显示购物车信息代码如下：

```
private void showCart(HttpServletRequest request,
          HttpServletResponse response) throws IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>购物车</title></head>");
    out.println("<body><a href='products'>" +
           "商品列表</a>");
    HttpSession session = request.getSession();
    List<ShoppingItem> cart = (List<ShoppingItem>) session
             .getAttribute("cart");
    if (cart != null) {
       out.println("<table>");
       out.println("<tr><td style='width:50px'>数量"
           + "</td>"
           + "<td style='width:80px'>商品</td>"
           + "<td style='width:80px'>价格</td>"
           + "<td style='width:80px'>小计</td>"
           + "<td style='width:80px'>是否删除</td></tr>");

       double total = 0.0;
       for (ShoppingItem shoppingItem : cart) {
          Product product = shoppingItem.getProduct();
          int quantity = shoppingItem.getQuantity();
          if (quantity != 0) {
              float price = product.getPrice();
              out.println("<tr>");
              out.println("<td>" + quantity + "</td>");
              out.println("<td>" + product.getName()
                     + "</td>");
              out.println("<td>" + price  + "</td>");
              // 计算小计并实现四舍五入
             double subtotal = ((int)(price * quantity*100+0.5))/100.00;
              out.println("<td>" + subtotal + "</td>");
              out.println("<td><a href=deleteItem?id=" +
                    product.getId()+">"+"删除</a>" + "</td>");
              total += subtotal;
              out.println("</tr>");
           }
        }
        out.println("<tr><td colspan='4' "
              + "style='text-align:right'>"
              + "总计:" + total  + "</td></tr>");
        out.println("</table>");
     }
     out.println("</table></body></html>");
}
```

在图 3-4 的购物车信息页面，每件商品还包括一个“删除”链接，其功能是点击该链接，从购物车中删除该商品，请读者自己编写一个 deleteItem()实现该功能。

【思考题】

1. 在 ServletContext 中存储的对象如何被 Web 应用程序的其他组件使用？
2. 实现会话跟踪的方法有哪些？简述 HttpSession 的工作机制。

3.3 习题解析

1. 下面哪个方法用于从ServletContext中检索属性？（ ）

A. String getAttribute(int index)　　B. String getObject(int index)

C. Object getAttribute(int index)　　D. Object getObject(int index)

E. Object getAttribute(String name)　　F. String getAttribute(String name)

【答】 E。

2. 下面哪个方法用来检索ServletContext初始化参数？（ ）

A. Object getInitParameter(int index)

B. Object getParameter(int index)

C. Object getInitParameter(String name)

D. String getInitParameter(String name)

E. String getParameter(String name)

【答】 D。

3. 为Servlet上下文指定初始化参数，下面的web.xml片段哪个是正确的？（ ）

A.
```
<context-param>
    <name>country</name>
    <value>China</value>
</context-param>
```

B.
```
<context-param>
    <param name="country" value="China" />
</context-param>
```

C.
```
<context>
    <param name="country" value="China" />
</context>
```

D.
```
<context-param>
    <param-name>country</param-name>
    <param-value>China</param-value>
</context-param>
```

【答】 D。

4.下面哪个接口或类检索与用户相关的会话对象？( )

A. HttpServletResponse　　B. ServletConfig

C. ServletContext　　D. HttpServletRequest

【答】 D。

5. 给定 request 是一个 HttpServletRequest 对象，下面哪两行代码会在不存在会话的情况下创建一个会话？（ ）

A. request.getSession() B. request.getSession(true)

C. request.getSession(false) D. request.createSession()

【答】 A、B。

6. 关于会话属性，下面哪两个说法是正确的？（ ）

A. HttpSession的getAttribute(String name)返回类型为Object

B. HttpSession的getAttribute(String name)返回类型为String

C. 在一个HttpSession上调用setAttribute("keyA","valueB")时，如果这个会话中对应键keyA已经有一个值，就会导致抛出一个异常

D. 在一个HttpSession上调用setAttribute("keyA","valueB")时，如果这个会话中对应键keyA已经有一个值，则这个属性的原先值会被valueB替换

【答】 A、D。

7. 调用下面哪个方法将使会话失效？( )

A. session.invalidate(); B. session.close();

C. session.destroy(); D. session.end();

【答】 A。

8. 是否能够通过客户机的IP地址实现会话跟踪？

【答】 不能。因为在一个局域网中不同机器的 IP 地址相同，所以不能唯一标识客户。

9．如何理解会话失效与超时？如何通过程序设置最大失效时间？如何通过 Web 应用程序部署描述文件设置最大超时时间？二者有什么区别？

参考答案：如果客户在指定时间内没有访问服务器，则该会话超时。对超时的会话对象，服务器使其失效。通过会话对象的setMaxInactiveInterval()设置会话最大超时时间。

web.xml文件使用<session-config>元素的子元素<session-timeout>设置最大超时时间，如下所示。

```
<session-config>
   <session-timeout>20</session-timeout>
</session-config>
```

这里的最大超时时间是对整个应用程序的所有会话有效，<session-timeout>元素指定的时间单位是分钟。setMaxInactiveInterval()参数单位是秒。

10. 关于HttpSession对象，下面哪两个说法是正确的？（ ）

A. 会话的超时时间设置为-1，则会话永远不会到期

B. 一旦用户关闭所有浏览器窗口，会话就会立即失效

C. 在部署描述文件中定义的超时时间之后，会话会失效

D. 可以调用HttpSession的invalidateSession()使会话失效

【答】 A、C。

11. 给定一个会话对象 s，有两个属性，属性名分别为 myAttr1 和 myAttr2，下面哪行（段）代码会把这两个属性从会话中删除？（ ）

A. s.removeAllValues();

B. s.removeAllAttributes();

C. s.removeAttribute("myAttr1");
s.removeAttribute("myAttr2");

D. s.getAttribute("myAttr1",UNBIND);
s. getAttribute("myAttr2",UNBIND);

【答】 C。

12. 将下面哪个代码片段插入到 doGet()中可以正确记录用户的 GET 请求的数量？（ ）

A.
```
HttpSession session = request.getSession();
int count = session.getAttribute("count");
session.setAttribute("count", count++);
```

B.
```
HttpSession session = request.getSession();
int count = (int) session.getAttribute("count");
session.setAttribute("count", count++);
```

C.
```
HttpSession session = request.getSession();
int count = ((Integer) session.getAttribute("count")).intValue();
session.setAttribute("count", count++);
```

D.
```
HttpSession session = request.getSession();
int count = ((Integer) session.getAttribute("count")).intValue();
session.setAttribute("count", new Integer(++count));
```

【答】 C、D。

13. 以下哪段代码能从请求对象中获取名为"ORA-UID"的Cookie的值？（ ）

A.
```
String value = request.getCookie("ORA-UID");
```

B.
```
String value = request.getHeader("ORA-UID");
```

C.
```
Cookie[] cookies = request.getCookies();
String cName=null;
String value = null;
if(cookies !=null){
   for(int i = 0 ;i<cookies.length; i++){
     cName = cookies[i].getName();
     if(cName!=null && cName.equalsIgnoreCase("ORA_UID")){
       value = cookies[i].getValue();
     }
   }
}
```

D. Cookie[] cookies = request.getCookies();
if(cookies.length >0){
String value = cookies[0].getValue();
}

【答】 C。

第 4 章　JSP 技术模型

4.1　本章要点

JSP（JavaServer Pages）是一种动态页面技术，一般来说在 JSP 页面中可以包含的元素如表 4-1 所示。

表 4-1　JSP 页面元素

JSP 页面元素	简要说明	标签语法
声明	声明变量与定义方法	<%! Java 声明 %>
小脚本	执行业务逻辑的 Java 代码	<% Java 代码 %>
表达式	用于在 JSP 页面输出表达式的值	<%= 表达式 %>
指令	指定转换时向容器发出的指令	<%@ 指令 %>
动作	向容器提供请求时的指令	<jsp:动作名 />
EL 表达式	JSP 2.0 引进的表达式语言	${applicationScope.email}
注释	用于文档注释	<%-- 任何文本 --%>
模板文本	HTML 标签和文本	同 HTML 规则

表 4.1 中前三种元素称为 JSP 脚本元素，如果要编写无脚本的 JSP 页面则不应该使用这几个元素。

JSP 页面本质上也是 Servlet，但若仅实现表示逻辑编写 JSP 页面要比编写 Servlet 容易。JSP 页面也在容器中运行。当 JSP 页面第一次被访问时，Web 容器解析 JSP 文件并将其转换成页面实现类，该类实现了 javax.servlet.jsp.JspPage 接口，该接口是 javax.servlet.Servlet 的子接口。接下来，Web 容器编译该类并将其装入内存，然后与其他 Servlet 一样执行并将其输出结果发送到客户端。

之后对该页面再次请求时，容器检查该页面自上次转换后是否被修改，若已修改则重新转换、编译、加载和执行，若没有修改则直接执行。

在 JSP 的脚本元素中可以使用九个隐含变量，它们分别是：application、session、request、response、page、pageContext、out、config 和 exception 等。

在JSP页面可以使用的指令有三种类型：page指令、include指令和taglib指令。三种指令的语法格式如下：

```
<%@ page attribute-list %>
<%@ include attribute-list %>
<%@ taglib attribute-list %>
```

page 指令通知容器关于 JSP 页面的总体特性，include 指令实现把另一个文件（HTML、JSP 等）的内容包含到当前页面中，taglib 指令用来指定在 JSP 页面中使用标准标签或自定义标签的前缀与标签库的 URI。

在 JSP 页面中有四个作用域对象，它们的类型分别是 ServletContext、HttpSession、HttpServletRequest 和 PageContext，这四个作用域分别称为应用（application）作用域、会话（session）作用域、请求（request）作用域和页面（page）作用域，如表 4-2 所示。

表 4-2　JSP 四种作用域对象

作用域名	对应的对象	存在性和可访问性
应用作用域	application	在整个 Web 应用程序有效
会话作用域	session	在一个用户会话范围内有效
请求作用域	request	在用户的请求和转发的请求内有效
页面作用域	pageContext	只在当前的页面（转换单元）内有效

JavaBeans 是 Java 平台的组件技术，在 Java Web 开发中常用 JavaBeans 来存放数据、封装业务逻辑等，在 JSP 页面中使用 JavaBeans 主要是通过三个 JSP 标准动作实现的，它们分别是：

- <jsp:useBean>动作用来在JSP页面中查找或创建一个bean实例。
- <jsp:setProperty>动作用来给bean实例的属性赋值。
- <jsp:getProperty>动作用来检索并向输出流中打印bean的属性值。

4.2　实验指导

【实验目的】

1. 掌握编写 JSP 页面的语法元素，理解 JSP 页面与 Servlet 的关系。
2. 掌握 page 指令属性的使用和 JSP 隐含变量的使用。
3. 掌握使用<%@ include>指令和<jsp:include>动作实现页面布局。
4. 掌握四种作用域对象的含义。
5. 理解 JavaBeans 及其有关动作的使用。

【实验内容】

实验题目 1：输入并执行下面的 todayDate.jsp 页面，完成后面的操作。

```
<%@page import="java.util.Date" %>
<%@page import="java.text.DateFormat"%>
<%@page contentType="text/html;charset=UTF-8" %>
<html>
<head><title>Today's date</title></head>
<body>
<%
   DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.FULL);
```

```
    String s = dateFormat.format(new Date());
%>
今天的日期是：<%=s%>
</body>
</html>
```

（1）打开 Tomcat 安装目录的\work\Catalina\localhost\helloweb\org\apache\jsp 目录中的 todayDate_jsp.java 文件，查看隐含对象是如何定义的。完成下面的填空：

JSP 页面转换后定义的类名为：（　　）

该类继承了哪个类：（　　）

隐含对象 request 的类型为：（　　）

隐含对象 response 的类型为：（　　）

隐含对象 pageContext 的类型为：（　　）

隐含对象 session 的类型为：（　　）

隐含对象 application 的类型为：（　　）

隐含对象 config 的类型为：（　　）

隐含对象 out 的类型为：（　　）

隐含对象 page 的类型为：（　　）

哪几个是作用域对象（　　）

（2）在页面实现类中找到 dateFormat 和 s 两个变量的声明位置。

实验题目 2：使用包含指令或包含动作实现页面布局。

本实验包括 4 个文件 header.jspf、body.jsp、footer.jspf 和 main.jsp。在 main.jsp 文件中使用 include 指令包含其他页面实现页面布局。

（1）创建 header.jspf 文件

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<p style="color:#ff0000;">
新世纪 网上书店</p>
<hr/>
```

（2）创建 body.jsp 文件

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<table border=0 cellspacing=5 cellpadding=5 width="100%">
  <tr><td>
    <p align="center"><b>欢迎光临新世纪网上书店！</b></p>
   </td>
  </tr>
  <tr>
   <td>
    <p align="center"><b><a href="/bookstore/catalog">开始购买图书</a></b>
   </td>
  </tr>
</table>
```

（3）创建 footer.jspf 文件

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<hr />
<center>Copyright &copy; 2013 New Century Web Bookstore, Inc.</center>
```

（4）创建 main.jsp 文件代码

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head><title>新世纪在线书店</title></head>
<body>
<%@ include file="header.jspf" %>
<%@ include file="body.jsp" %>
<%@ include file="footer.jspf" %>
</body>
</html>
```

执行 main.jsp 文件，结果如图 4-1 所示。

图 4-1　main.jsp 页面的运行结果

（5）修改上面程序，使用<jsp:include>动作实现页面布局。

实验题目 3：在 Servlet 和 JSP 页面中使用 JavaBeans 对象。

本实验在 JSP 页面 inputProduct.jsp 中输入商品信息，将请求转到 ProductServlet，创建一个 Product 对象，然后将请求转发到 dispalyProduct.jsp 页面，在其中使用<jsp:getProperty>动作输出信息。

Product 类的主要代码如下：

```
package com.demo;
public class Product{
   private String id;          // 商品号
   private String name;        // 商品名
   private double price;       // 价格
}
```

其他代码请读者自己定义，运行该程序在输入页面中输入商品号 202，商品名“iPhone 5 手机”，价格输入 2200，最后显示的页面如下。

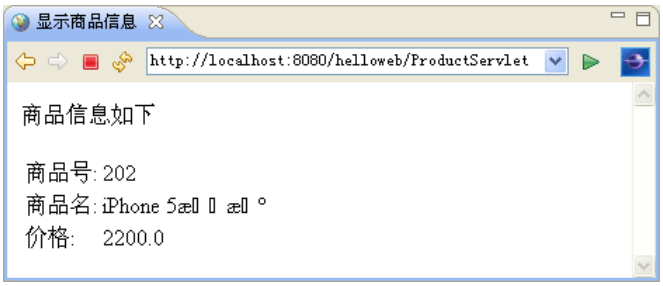

图 4-2　displayProduct.jsp 页面的运行结果

提示：当在输入页面中输入的商品名中包含中文时，结果页面出现乱码（见图 4-2）。若出现乱码，修改 ProductServlet 类使用下面代码转换字符编码：

```
name = new String(name.getBytes("ISO-8859-1"),"UTF-8");
```

【思考题】

1. 如何理解 JSP 页面的生命周期？
2. 在 JSP 页面中可以使用的隐含对象有哪几个？
3. 在 JSP 页面中可以使用的作用域对象有哪几个？
4. 在 JSP 页面中使用 JavaBean 的动作有哪几个？

4.3　习题解析

1.下面JSP代码输出结果是什么？为什么？

```
<% int x = 3; %>
<%! int x = 5; %>
<%! int y = 6; %>
x 与 y 的和是：<%=x+y%>
```

【答】　x 与 y 的和是：9。变量 x 将被声明两次：一次是作为类的全局变量，因为使用了<%! int x = 5; %>语句，另一次是在_jspService()中声明的局部变量，因为使用的代码是<% int x = 3; %>。

2. 下面JSP代码有什么错误？

```
<!% int i = 5; %>
<!% int getI() { return i; } %>
```

【答】　正确声明应为：

```
<%! int i = 5; %>
<%! int getI() { return i; } %>
```

3. 假设myObj是一个对象的引用，m1()是该对象上一个合法的方法。下面的JSP结构哪个是合法的哪个是不合法的？（　　）

A. <% myObj.m1() %>　　B. <%= myObj.m1() %>

C. <%　=myObj.m1() %>　　D. <% =myObj.m1(); %>

【答】　B 是合法的。解析：JSP 表达式中百分号和等号之间不能有空格。

4. 说明下面代码是否是合法的JSP结构？（　　）

A. <%=myObj.m1(); %>

B. <% int x=4, y=5; %>

<%=x=y%>

C. <% myObj.m1(); %>

【答】　B、C。

A.非法：等号表明它是表达式，但表达式不能以分号结束。

B.合法：<%=x=y%>将被转换成：

```
out.print(x=y);  // y 的值 5 赋给 x 并将其打印输出
```

C. 合法：这是合法的小脚本，因为在方法调用语句的后面有分号。即使该方法返回一个值，它也是合法的，因为返回的值将被忽略。

5. 下面哪个page指令是合法的？（　　）

A. <% page language="java" %>　　B. <%! page language="java" %>

C. <%@ page language="java" %>　　D. <%@ Page language="java" %>

【答】　C。

6. 下面的page指令哪个是合法的？（　　）

A. <%@ page import="java.util.* java.text.* " %>

B. <%@ page import="java.util.*", "java.text.* " %>

C. <%@ page buffer="8kb", session="false" %>

D. <%@ page import="com.manning.servlets.* " %>

<%@ page session="true" %>

<%@ page import="java.text.*" %>

E. <%@ page bgcolor="navy" %>

F. <%@ page buffer="true" %>

G. <%@ Page language='java' %>

【答】　D。解析：A 中 import 的属性值中应该有逗号。B 的 import 属性值应该在一个字符串中指定。C，属性之间不允许有逗号。E，bgcolor 不是合法的属性名。F，true 不是 buffer 属性合法值。G，指令名、属性名和值都是大小写敏感的，Page 应为 page。

7. 下面哪些是合法的 JSP 隐含变量？（　　）

A. stream　　B. context

C. exception　　D. listener

E. application

【答】　C、E。

8. 下面是 JSP 生命周期的各个步骤，正确的顺序应该是。（　　）

① 调用_jspService()

② 把 JSP 页面转换为 Servlet 源代码

③ 编译 Servlet 源代码

④ 调用 jspInit()

⑤ 调用 jspDestroy()

⑥ 实例化 Servlet 对象

【答】　②③⑥④①⑤。

9. 有下面 JSP 页面，给出该页面每一行在转换的 Servlet 中的代码。

```
<html><body>
  <% int count = 0 ;%>
  The page count is now:
  <%= ++count %>
</body></html>
```

【答】

```
out.write("<html><body>\r\n");
int count = 0 ;
out.write("   The page count is now:\r\n");
out.print( ++count );
out.write("</body></html>\r\n");
```

10. 有下列名为 counter.jsp 的页面，其中有 3 处错误。执行该页面，从浏览器输出中找出错误，修改错误直到页面执行正确。

```
<%@ Page contentType="text/html;charset=UTF-8" %>
<html><body>
<%! int count = 0 %>
<% count++; %>
该页面已被访问 <%= count ;%> 次。
</body></html>
```

【答】

```
Page 改为 page            // page 的 p 应为小写
<%! int count = 0 %>     // 声明缺少分号
<% count++; %>           // 去掉分号
```

11. 下面左边一栏是 JSP 元素类型，右边是对应名称，请连线。

```
<% Float one = new Float(88.88) %>                指令
<%! int y = 3; %>                                 EL 表达式
<%@ page import="java.util.*" %>                  声明
<jsp:include page="foo.jsp" />                    小脚本
<%=pageContext.getAttribute("foo") %>             动作
email:${applicationScope.mail}                    表达式
```

【答】

```
<% Float one = new Float(88.88) %>          小脚本
<%! int y = 3; %>                           声明
<%@ page import="java.util.*" %>            指令
<jsp:include page="foo.jsp" />              动作
<%=pageContext.getAttribute("foo") %>       表达式
email:${applicationScope.mail}              EL 表达式
```

12. 以下关于 JSP 生命周期方法，哪个是正确的？（　　）

A. 只有 jspInit()可以被覆盖

B. 只有 jspdestroy()可以被覆盖

C. jspInit()和 jspdestroy()都可以被覆盖，但_jspService()不可以被覆盖

D. jspInit()、_jspService()和 jspdestroy()都可以被覆盖

【答】 C。

13. 下面哪个JSP标签可以在请求时把另一个JSP页面的结果包含到当前页面中？

A. <%@ page import %>　　B. <jsp:include>

C. <jsp: plugin>　　D. <%@ include %>

【答】 B。

14. 在一个JSP页面中要把请求转发到view.jsp页面，下面哪个是正确的？（　　）

A. <jsp:forward file="view.jsp" />　　B. <jsp:forward page="view.jsp" />

C. <jsp:dispatch file="view.jsp" />　　D. <jsp:dispatch page="view.jsp" />

【答】 B。

15. 下面的代码有什么错误？

```
<jsp:useBean id="customer" class="com.model.Customer"
   beanName="businessData.visitorCustomeres.John" />
```

【答】 不能在同一个<jsp:useBean>声明中同时使用beanName和class属性。

16. 下面的代码有什么错误？

```
<jsp:setProperty name="customer"
                 param="phone"  value="FL" />
```

【答】 必须使用 property 指定 bean 的属性。param 用来指定请求参数，并且不能在同一个<jsp:setProperty>动作中同时指定 param 和 value 属性。

17. 给定下面的MyBean定义：

```
package com.example;
public class MyBean{
   private int value;
   public MyBean(){value = 42;}
   public int getValue(){return value;}
   public void setValue(int value){this.value=value;}
```

}

假设还没有创建MyBean实例，以下哪个标准动作可以创建一个bean实例，并将它存储在请求作用域？（　　）

A. <jsp:useBean id="mybean" type="com.example.MyBean" />

B. <jsp:makeBean id="mybean" type="com.example.MyBean" />

C. <jsp:useBean id="mybean" class="com.example.MyBean" scope="request"/>

D. <jsp:makeBean id="mybean" class="com.example.MyBean" scope="request"/>

【答】　C。

18. 简述实现 MVC 设计模式的一般步骤。

参考答案：MVC 模式称为模型-视图-控制器模式。该模式将 Web 应用的组件分为模型、视图和控制器，每种组件完成各自的任务。该模型将业务逻辑和数据访问从表示层分离出来。实现 MVC 模式的一般步骤：（1）定义 JavaBeans 表示数据；（2）使用 Servlet 处理请求；（3）填写 JavaBeans 对象数据；（4）将结果存储在作用域对象中；（5）将请求转发到 JSP 页面；（6）最后在 JSP 页面中从 JavaBeans 中取出数据。

第 5 章　表达式语言

5.1　本章要点

表达式语言（Expression Language，EL）是 JSP 2.0 的一个重要特征，使用 EL 可以方便地访问应用数据。在 JSP 页面中，表达式语言的使用形式如下。

```
${expression}
```

该结构可以出现在 JSP 页面的模板文本中，也可以出现在 JSP 标签的属性值中，下面是在 JSP 模板文本中使用 EL 表达式。

```
<ul>
   <li>客户名：${customer.custName}
   <li>Email 地址：${customer.email}
</ul>
```

下面是在 JSP 标准动作的属性中使用 EL 表达式。

```
<jsp:include page = "${expression1}" />
<c:out value = "${expression2} " />
```

使用 EL 可以很方便地访问作用域变量，JavaBeans 的属性和集合的元素值。此外，EL 还提供了隐含变量。

（1）访问作用域变量。要输出作用域变量的值，只需在 EL 中使用变量名即可，例如：

```
${variable_name}
```

（2）访问 JavaBeans 属性。可以通过点号（.）或方括号（[]）运算符访问 JavaBeans 的属性，如下所示：

```
${beanName.propertyName}
${beanName[propertyName]}
```

如果一个属性是另一个 JavaBean 对象，则还可以访问属性的属性，例如：

```
${employee.address.zipCode}
```

（3）访问集合对象元素。在 EL 中可以访问各种集合对象的元素，集合可以是数组、List 对象或 Map 对象。这需要使用数组记法的运算符([])。例如：

```
${attributeName[entryName]}
```

（4）访问 EL 隐含变量。在 EL 中定义了下面的隐含变量：pageContext、param、paramValues、header、headerValues、initParam、cookie、pageScope、requestScope、sessionScope 和 applicationScope。下面是使用隐含变量的两个例子：

```
${pageContext.request.requestURL}
${cookie.jsessionid.value}
```

EL 语言还提供了若干运算符实现简单运算。主要包括：

- 算术运算符：加（+）、减（-）、乘（*）、除（/或 div）和求余数（%或 mod）。
- 关系运算符：相等（==或 eq）、不相等（!=或 ne）、大于（>或 gt）、小于（<或 lt）大于等于（>=或 ge）和小于等于（<=或 le）。
- 逻辑运算符：逻辑非（!或 not）、逻辑与（&&或 and）、逻辑或（||或 or）。
- 条件运算符：${condition?A:B}，当 condition 的值为 true 返回 A，否则返回 B。
- 空运算符：${empty X}，当 X 为 null、空字符串、空 Map 对象、空数组、空集合时返回 true，否则返回 false。

无脚本的 JSP 页面。使用 EL、JavaBeans 和自定义标签使得编写无脚本（声明、小脚本、表达式）的页面成为可能。在 web.xml 中通过设置可以禁止在 JSP 页面中使用脚本，如下所示。

```
<jsp-config>
    <jsp-property-group>
       <url-pattern>*.jsp</url-pattern>
       <scripting-invalid>true</scripting-invalid>
    </jsp-property-group>
</jsp-config>
```

也可以禁止在 JSP 页面中使用 EL，这适用于早期编写的 JSP 页面。有多种方法禁用 JSP 中使用 EL。

如果禁止在一个页面使用 EL，可使用 page 指令，如下：

```
<%@ page isELIgnored="true" %>
```

如果禁用多个 JSP 页面使用 EL，可以使用<jsp-property-group>元素，如下：

```
<jsp-config>
    <jsp-property-group>
       <url-pattern>*.jsp</url-pattern>
       <el-ignored>true</el-ignored>
    </jsp-property-group>
</jsp-config>
```

提示：在web.xml文件中至多只能有一个<jsp-config>元素，该元素可有多个<jsp-property-group>子元素。

5.2 实验指导

【实验目的】

1. 了解表达式语言的功能。
2. 掌握表达式语言的使用。

【实验内容】

实验题目 1：输入并执行下面 JSP 页面 elDemo.jsp，体会 EL 运算符的使用。

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
    <title>JSP 2.0 Expression Language - Basic Operators</title>
</head>
<body>
<h1>JSP 2.0 表达式语言 - 基本算术运算符</h1>
<hr>
该例说明了基本的表达式语言的算术运算符的使用，其中包括加(+),减(-), 乘(*),除(/ 或
div), 取余 (% 或 mod)。
<br>
<table border="1">
   <tr><td><b>EL 表达式</b></td><td><b>结果</b></td>
   </tr>
  <tr> <td>\${1 + 2}</td> <td>${1 + 2}</td> </tr>
  <tr> <td>\${1.2 + 2.3}</td> <td>${1.2 + 2.3}</td> </tr>
  <tr> <td>\${1.2E4 + 1.4}</td> <td>${1.2E4 + 1.4}</td> </tr>
  <tr> <td>\${-4 - 2}</td> <td>${-4 - 2}</td> </tr>
  <tr> <td>\${21 * 2}</td> <td>${21 * 2}</td> </tr>
  <tr> <td>\${3/4}</td> <td>${3/4}</td> </tr>
  <tr> <td>\${3 div 4}</td> <td>${3 div 4}</td> </tr>
  <tr> <td>\${3/0}</td> <td>${3/0}</td> </tr>
  <tr> <td>\${10%4}</td> <td>${10%4}</td> </tr>
  <tr> <td>\${10 mod 4}</td> <td>${10 mod 4}</td> </tr>
   <tr> <td>\${(1==2) ? 3 : 4}</td> <td>${(1==2) ? 3 : 4}</td> </tr>
</table>
</body>
</html>
```

实验题目 2：访问作用域变量。

（1）编写一个名为Employee的JavaBeans，其中包括3个属性：eno表示员工号、ename表示员工名和ecompany表示员工公司名。代码如下。

```
package com.model;

public class Employee{
  private String eno = "";
  private String ename = "";
  private String ecompany = "";

  public Employee() { }
  // 这里省略了属性的 setter 和 getter 方法
```

```
}
```

（2）编写下面的JSP页面inputEmployee.jsp，在其中通过表单输入员工信息，将请求转发到一个Servlet。

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<body>
请输入员工信息:
<form action="employee.do" method="post">
<table>
<tr><td>员工号:</td><td><input type="text" name="eno"></td></tr>
<tr><td>员工名:</td><td><input type="text" name="ename"></td></tr>
<tr><td>公司名:</td><td><input type="text" name="ecompany"></td></tr>
</table>
<input type="submit" value="提交">
</form>
</body>
</html>
```

（3）下面的Servlet从JSP页面得到员工信息

```
package com.demo;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.model.Employee;
import javax.servlet.annotation.WebServlet;

@WebServlet(urlPatterns = {"/employee.do"})
public class EmployeeServlet extends HttpServlet{
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                  throws ServletException,IOException{

      String eno = request.getParameter("eno");
      String ename = request.getParameter("ename");
      String ecompany = request.getParameter("ecompany");
      Employee emp = new Employee();
      emp.setEno(eno);
      emp.setEname(ename);
      emp.setEcompany(ecompany);

      request.setAttribute("employee", emp);
      RequestDispatcher view =
          request.getRequestDispatcher("/displayEmployee.jsp");
      view.forward(request, response);
   }
}
```

（4）下面的JSP页面displayEmployee.jsp使用EL表达式显示员工的信息

```
<%@ page contentType="text/html;charset=gb2312"%>
<html><body>
员工的信息如下：<br>
<ul>
  <li>员工号:${employee.eno}
  <li>员工名:${employee.ename}
  <li>公司名:${employee.ecompany}
</ul>
</body></html>
```

实验题目 3：EL 隐含对象的使用。下面的 JSP 页面 implicit.jsp 演示了 EL 隐含对象的使用。

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
  <head>
    <title>EL implicit objects</title>
  </head>
  <body>
    <p>JSP 2.0 表达式语言-隐含对象</p>
    <hr>
    <p>输入 foo 参数值</p>
      <form action="implicit.jsp" method="GET">
      foo= <input type="text" name="foo" value="${param['foo']}" />
         <input type="submit" />
      </form>
      <br>
    <table border="1">
       <tr><td><b>EL 表达式</b></td><td><b>结果</b></td>
      </tr>
      <tr>
        <td>\${param.foo}</td>
        <td>${param.foo} </td>
      </tr>
      <tr>
        <td>\${param["foo"]}</td>
        <td>${param["foo"]} </td>
      </tr>
      <tr>
        <td>\${header["host"]}</td>
        <td>${header["host"]}</td>
      </tr>
      <tr>
        <td>\${header["accept"]}</td>
        <td>${header["accept"]}</td>
      </tr>
      <tr>
        <td>\${header["user-agent"]}</td>
        <td>${header["user-agent"]}</td>
```

```
        </tr>
</table>
</body>
</html>
```

【思考题】

1. 使用 EL 能否定义作用域变量？能否实现流程控制？
2. EL 是否是通用的程序设计语言？

5.3 习题解析

1. 简述表达式语言的主要功能。

【答】 表达式语言是 JSP 页面中使用的一种简洁的数据访问语言。它定义了运算符实现算术、关系等运算；可以对作用域变量、JavaBeans 对象、集合的元素、请求参数、Cookie 等进行简单的访问；还可以访问 Java 语言定义的函数（静态方法）。

2. 属性与集合的访问运算符的点（.）运算符与方括号（[]）运算符有什么不同？

【答】 使用点（.）运算符可以访问 Map 对象一个键的值和 bean 对象的属性值。使用方括号（[]）运算符还可以 List 对象和数组对象的元素。

3. 在 EL 中都可以访问哪些类型的数据？

【答】 （1）作用域变量；（2）JavaBeans 的属性；（3）访问集合元素；（4）访问隐含变量。

4. 有下面 JSP 页面，叙述正确的是（　　）。

```
<html><body>
   ${(5 + 3 + a > 0) ? 1 : 2}
</body></html>
```

A. 语句合法，输出1　　B. 语句合法，输出2

C. 因为a没有定义，因此抛出异常　　D. 因为表达式语法非法，因此抛出异常

【答】 A。

5. 表达式${(10 le 10) && !(24+1 lt 24) ? "Yes" :"No"}的结果是什么？（　　）

A. Yes　　B. No

C. true　　D. false

【答】 A。

6. 下面哪个变量不能用在 EL 表达式中？（　　）

A. param　　B. cookie

C. header　　D. pageContext

E. contextScope

【答】 E。

7. 下面哪两个表达式不能返回 header 的 accept 域?（　　）

A. ${header.accept}　　B. ${header[accept]}

C. ${header['accept']}　　D. ${header["accept"]}

E. ${header.'accept'}

【答】　B、E。

8. 如果使用EL显示请求的URI，下面哪个是正确的？（　　）

A. ${pageScope.request.requestURI}

B. ${pageContext.request.requestURI}

C. ${ request.requestURI}

D. ${requestScope.request.requestURI}

【答】　B。

9. 如果 result 是一个合法的参数名，如何使${paramValues.result}成为一个合法的表达式？

【答】　由于 paramValues 返回的是包含 String[]的 Map，所以需要访问数组的单个元素，${paramValues.result[0]}和${paramValues.result["0"]}。paramValues 和 headerValues 返回 String[]的都是 Map 对象。

10. 给定一个 HTML 表单，其中使用了有一个名为 hobbies 的复选框，如下所示：

```
兴趣：<input type="checkbox" name="hobbies" value="reading">文学
    <input type="checkbox" name="hobbies" value="sport">体育
    <input type="checkbox" name="hobbies" value="computer">电脑<br>
```

下面哪些表达式能够计算并得到 hobbies 参数的第一个值？（　　）

A. ${param.hobbies }　　B. ${ paramValues. hobbies }

C. ${paramValues.hobbies[0]}　　D. ${ paramValues. hobbies[1]}

E. ${ paramValues.[hobbies][0]}

【答】　A、C

11. 一个Web站点将管理员的Email地址存储在一个名为master-email的ServletContxt参数中，如何使用EL得到这个值？（　　）

A. email me

B. email me

C. email me

D. email me

【答】　C

12. 下面页面的输出结果是什么？

```
<%@ page isELIgnored="true"%>
<html><head>
   ${(5 + 3 > 0) ? true : false}
```

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

【答】 ${(5 + 3 > 0) ? true : false}，因为使用 page 指令禁用了表达式语言。

第 6 章　JSP 标签技术

6.1　本章要点

JSP 标签技术主要包括自定义标签、JSTL 和标签文件。从 JSP1.1 版开始就可以在 JSP 页面中使用标签了，使用标签不但可以实现代码重用，而且可以使 JSP 代码更简洁。

在 JSP 页面中可以使用两类自定义标签。一类是简单的（simple）自定义标签，一类是传统的（classic）自定义标签。本书只讨论简单标签。

简单标签 API 包括 SimpleTag 接口和其实现类 SimpleTagSupport，自定义标签可以实现 SimpleTag 接口或继承 SimpleTagSupport 类。

创建和使用自定义标签一般包含下面三个步骤：

- 创建标签处理类。
- 创建标签库描述文件 TLD。
- 在 JSP 页面中引入标签库和使用标签。

简单自定义标签的生命周期如下：

（1）JSP 容器调用标签处理类的无参数构造方法创建一个实例。

（2)JSP 容器调用 setJspContext()，为其传递一个 JspContext 对象。调用该对象的 getOut()返回 JspWriter 对象，使用它向客户发送响应。

（3） 如果自定义标签嵌套在另一个标签内，JSP 容器调用 setParent()设置父标签，该方法签名如下：

```
public void setParent(JspTag parent)
```

（4）如果标签带属性，JSP 容器调用每个属性的 setter 方法设置属性值。

（5）如果标签带标签体，JSP 容器调用 setJspBody()，将标签体内容作为 JspFragment 对象传递给该方法，调用其 invoke(null)计算并输出标签体。

（6）最后，JSP 容器调用 dotag()向 JSP 输出信息。

JSP 标准标签库（JSP Standard Tag Library，JSTL）是为实现 Web 应用程序常用功能而开发的标签库，它是由一些专家和用户开发的。在使用 JSTL 前，首先应该获得 JSTL 包，并安装到 Web 应用程序中。JSTL 共提供了 5 个库，每个子库提供了一组实现特定功能的标签，具体来说，这些子库包括：

- 核心标签库，包括通用处理的标签。
- XML 标签库，包括解析、查询和转换 XML 数据的标签。
- 国际化和格式化库，包括国际化和格式化的标签。
- SQL 标签库，包括访问关系数据库的标签。

- 函数库，包括管理 String 和集合的函数。

在JSP页面中使用JSTL，必须使用taglib指令来引用标签库，例如，要使用核心标签库，必须在JSP页面中使用下面的taglib指令。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

从 JSP 2.0 开始，还可以开发和使用标签文件，它不需要编写标签处理类和标签库描述文件，它简化了自定义标签的开发，只需用 JSP 语法就可开发标签文件。关于标签文件的详细信息请参阅本书第 1 版。

6.2 实验指导

【实验目的】

1. 了解什么是自定义标签。
2. 掌握使用 SimpleTag 和 SimpleTagSuppot 开发简单标签。
3. 掌握 JSTL 核心库中标签的使用。

【实验内容】

实验题目 1：定义一个简单的标签，向浏览器输出一段信息。

（1）创建标签处理类。

```
package com.mytag;

import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
import javax.servlet.jsp.tagext.SimpleTag;

public class MyFirstTag implements SimpleTag {
    JspContext jspContext;
    public void doTag() throws IOException, JspException {
        System.out.println("执行 doTag()方法");
        jspContext.getOut().print("这是我的第一个标签。");
    }
    public void setParent(JspTag parent) {
        System.out.println("执行 setParent()方法");
    }
    public JspTag getParent() {
        System.out.println("执行 getParent()方法");
        return null;
    }
    public void setJspContext(JspContext jspContext) {
        System.out.println("执行 setJspContext()方法");
        this.jspContext = jspContext;
```

```
    }
    public void setJspBody(JspFragment body) {
        System.out.println("执行 setJspBody()方法");
    }
}
```

（2）创建标签库描述文件 mytaglib.tld。

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    web-jsptaglibrary_2_1.xsd"
    version="2.1">
    <description>Simple tag examples</description>
    <tlib-version>1.0</tlib-version>
    <short-name>My First Taglib Example</short-name>
    <uri>http://www.mydomain.com/sample</uri>
    <tag>
        <name>firstTag</name>
        <tag-class>com.mytag.MyFirstTag</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

（3）编写 JSP 页面使用标签。

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.mydomain.com/sample" prefix="demo"%>
<html>
<head><title>第一个标签</title>
</head>
<body>
您好!!!!
<br/>
<demo:firstTag></demo:firstTag>
</body>
</html>
```

实验题目 2：开发一个带属性的标签。下面 math 标签带一个 double 型属性 x，标签功能是求 x 的平方根。

（1）创建标签处理类代码。

```
package com.mytag;

import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class MathTag extends SimpleTagSupport {
    double x;
    public void setX(String x){
```

```
        double num =0;
        try{
           num = Double.parseDouble(x);
        }catch(NumberFormatException nfe){}
        this.x = num;
    }
    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();
        out.print( x +" 的平方根是 : " + Math.sqrt(x));
    }
}
```

（2）在 TLD 文件 mytaglib.tld 添加下面代码。

```
<tag>
    <description>Compute Square Root</description>
    <name>sqrt</name>
    <tag-class>com.mytag.MathTag</tag-class>
    <body-content>empty</body-content>
    <attribute>
        <name>x</name>
        <required>true</required>
    </attribute>
</tag>
```

（3）编写 JSP 页面 math.jsp 访问该标签。

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.mydomain.com/sample" prefix="demo"%>
<html>
<head><title>sqrt 标签</title></head>
<body>
   <demo:sqrt x="100"/><br>
   <demo:sqrt x="200"/><br>
   <demo:sqrt x="0"/><br>
</body></html>
```

该页面的运行结果如图 6-1 所示。

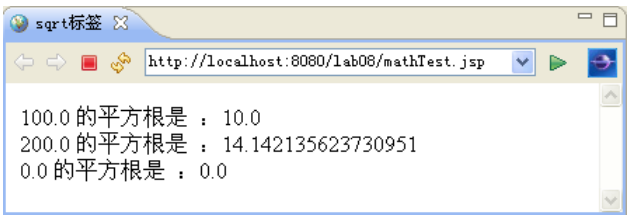

图 6-1　mathTest.jsp 页面运行结果

实验题目 3：使用 JSTL 的<c:forEach>标签对 Map 对象迭代。该实验包括 BigCitiesServlet 类和 bigCities.jsp 页面。

在 BigCitiesServlet 类中创建一个 Map<String,String>对象 capitals，键为国家名称，值为首都名称，添加几个对象。另外创建一个 Map<String,String[]>对象 bigCities，键为国家名称，值为 String 数组包含该国家的几个大城市。在 doGet()中使用 RequestDispatcher 对象将请求转发到 bigCities.jsp 页面。

```
package com.demo;

// 此处省略若干 import 语句

@WebServlet(urlPatterns = {"/bigCities"})
public class BigCitiesServlet extends HttpServlet {
   private static final int serialVersionUID = 112233;
   @Override
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                     throws ServletException, IOException {
      Map<String, String> capitals =
                          new HashMap<String, String>();
      capitals.put("俄罗斯", "莫斯科");
      capitals.put("日本", "东京");
      capitals.put("中国", "北京");
      request.setAttribute("capitals", capitals);

      Map<String, String[]> bigCities =
                           new HashMap<String, String[]>();
      bigCities.put("澳大利亚", new String[] {"悉尼",
                       "墨尔本", "布里斯班"});
      bigCities.put("美国", new String[] {"纽约",
                         "洛杉矶", "加利福尼亚"});
      bigCities.put("中国", new String[] {"北京",
                        "上海", "深圳"});

      request.setAttribute("capitals", capitals);
      request.setAttribute("bigCities", bigCities);
      RequestDispatcher rd =
             request.getRequestDispatcher("/bigCities.jsp");
      rd.forward(request, response);
   }
}
```

bigCities.jsp 页面代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>Big Cities</title></head>
<body>
<table>
   <tr style="background:#448755;color:white;font-weight:bold">
```

```
        <td>国家</td>
        <td>首都</td>
    </tr>
    <c:forEach items="${requestScope.capitals}" var="mapItem">
    <tr>
        <td>${mapItem.key}</td>
        <td>${mapItem.value}</td>
    </tr>
    </c:forEach>
</table>
<br/>
<table>
    <tr style="background:#448755;color:white;font-weight:bold">
        <td>国家</td>
        <td>城市</td>
    </tr>
    <c:forEach items="${requestScope.bigCities}" var="mapItem">
    <tr>
        <td>${mapItem.key}</td>
        <td>
            <c:forEach items="${mapItem.value}" var="city"
                       varStatus="status">
                ${city}<c:if test="${!status.last}">,</c:if>
            </c:forEach>
        </td>
    </tr>
    </c:forEach>
</table>
</body>
</html>
```

访问 BigCitiesServlet，显示的 bigCities.jsp 页面如图 6-2 所示。

图 6-2　bigCites.jsp 页面运行结果

【思考题】

1. 试述开发自定义标签的基本步骤。

2. 若要在 JSP 页面中使用 JSTL 标签在 Web 应用中添加什么库文件？

6.3 习题解析

1. 下面哪个是 SimpleTag 接口的 doTag()的返回值？（ ）

A. EVAL_BODY_INCLUDE B. SKIP_BODY

C. void D. EVAL_PAGE

【答】 C。

2. 下面哪个类提供了 doTag()的实现？（ ）

A. TagSupport B. SimpleTagSupport

C. IterationTagSupport D. JspTagSupport

【答】 B。

3. JspContext.getOut()返回的是哪一种对象类型？（ ）

A. ServletOutputStream B. PrintWriter

C. BodyContent D. JspWriter

【答】 D。

4. 下面哪个方法不能直接被 SimpleTagSupport 的子类使用？（ ）

A. getJspBody() B. getJspContext().getAttribute("name");

C. getParent() D. getBodyContent()

【答】 D。

5. 简单标签的 TLD 文件的<body-content>元素内容，下面哪个是不合法的？（ ）

A. JSP B. scriptless C. tagdependent D. empty

【答】 A。

6. 下面哪个是合法的taglib指令? ( )

A. <% taglib uri="/stats" prefix="stats" %>

B. <%@ taglib uri="/stats" prefix="stats" %>

C. <%! taglib uri="/stats" prefix="stats" %>

D. <%@ taglib name="/stats" prefix="stats" %>

【答】 B。

7. 下面哪个是合法的taglib指令? ( )

A. <%@ taglib prefix="java" uri="sunlib"%>

B. <%@ taglib prefix="jspx" uri="sunlib"%>

C. <%@ taglib prefix="jsp" uri="sunlib"%>

D. <%@ taglib prefix="servlet" uri="sunlib"%>

E. <%@ taglib prefix="sunw" uri="sunlib"%>

F. <%@ taglib prefix="suned" uri="sunlib"%>

【答】 F，其他的名称都不能作为前缀名。

8. 一个标签库有一个名为printReport的标签，该标签可以接受一个名为department的属性，它不能接受动态值。下面哪两个是该标签的正确使用?（ ）

A. <mylib:printReport/>

B. <mylib:printReport department="finance"/>

C. <mylib:printReport attribute="department" value="finance"/>

D. <mylib:printReport attribute="department"
attribute-value="finance"/>

E. <mylib:printReport>
<jsp:attribute name="department" value="finance" />
</mylib:printReport>

【答】 B、D。

9. 下面哪个是将一个标签嵌套在另一个标签中的正确用法?（ ）

A.
```
<greet:hello>
<greet:world>
</greet:hello>
</greet:world>
```

B.
```
<greet:hello>
<greet:world>
</greet:world>
</greet:hello>
```

C.
```
<greet:hello
<greet:world/>
/>
```

D.
```
<greet:hello>
</greet:hello>
<greet:world>
</greet:world>
```

【答】 B。

10. 一个标签库有一个名为getMenu的标签，该标签有一个名为subject的属性，该属性可以接受动态值。下面哪两个是对该标签的正确使用?（ ）

A. <mylib:getMenu />

B. <mylib:getMenu subject="finance"/>

C. <% String subject="HR";%>
<mylib:getMenu subject="<%=subject%>"/>

D. <mylib:getMenu> <jsp:param subject="finance"/> </mylib:getMenu>

E. <mylib:getMenu>
<jsp:param name="subject" value="finance"/>
</mylib:getMenu>

【答】 B、C。

11. 在web.xml文件中的一个合法的<taglib>元素需要哪两个元素？( )

A. uri　　B. taglib-uri　　C. tagliburi

D. tag-uri　　E. location　　F. taglib-location

【答】 B、F。

12. 考虑下面一个Web应用程序部署描述文件中的<taglib> 元素：

```
<taglib>
   <taglib-uri>/accounting</taglib-uri>
   <taglib-location>/WEB-INF/tlds/SmartAccount.tld</taglib-location>
</taglib>
```

下面在JSP页面中哪个正确指定了上述标签库的使用?（　　）

A. <%@ taglib uri="/accounting" prefix="acc"%>

B. <%@ taglib uri="/acc" prefix="/accounting"%>

C. <%@ taglib name="/accounting" prefix="acc"%>

D. <%@ taglib library="/accounting" prefix="acc"%>

【答】　A。

13. 下面有三个文件分别是JSP页面、标签处理类和TLD文件的部分代码，根据已有内容在方框中填上正确的内容，并请指出它们之间的关系。

标签处理类LoginTagHandler.java部分代码如下：

```
public class LoginTagHandler{
   public void doTag(){
         // 标签逻辑
   }
   public void setUser(String user){
        this.user = user;
   }
}
```

TLD文件的部分代码如下：

```
<taglib …>
  <uri>randomthings</uri>
  <tag>
        <name>advice</name>
        <tag-class>foo.LoginTagHandler</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>          </name>
            <required>true</required>
            <rtexprvalue>       </rtexprvalue>
        </attribute>
  </tag>
</taglib>
```

JSP页面代码如下：

```
<html><body>
<%@ taglib prefix="       "  uri="            "%>
  Login page<br>
  <mine:          user = "${foo}" />
</body><html>
```

【答】 user true mine randomthings advice

14. 把下面哪个代码放入简单标签的标签体中不可能输出 9？（ ）

A. ${3+3+3} B. "9"

C. <c:out value="9"> D. <%=27/3>

【答】 D。简单标签的标签体中不能包含脚本元素。

15. 下面 JSP 页面中使用了 JSTL 标签，它的运行结果如何？

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html><body>
  <c:forEach var="x" begin="0" end="30" step="3">
    ${x}
  </c:forEach>
</body></html>
```

【答】 在浏览器中输出下面一行。

0 3 6 9 12 15 18 21 24 27 30

16. 下面哪个与<%= var %>产生的结果相同?（ ）

A. <c:set value=var> B. <c:var out=${var}>

C. <c:out value=${var}> D. <c:out var="var">

E. <c:expr value=var>

【答】 C。

17. 下面代码的输出结果为?（ ）

```
<c:set value="3" var="a" />
<c:set value="5" var="b" />
<c:set value="7" var="c" />
${a div b}+${b mod c}
```

A. 5.6 B. 0.6+5

C. a div b + b mod c D. 3 div 5 + 5 mod 7

【答】 B。

18. <c:if>的哪个属性指定条件表达式? ( )

A. cond B. value

C. check D. expr

E. test

【答】 E。

19. 下面哪个JSTL的 <c:forEach>标签是合法的? ( )

A. <c:forEach varName="count" begin="1" end="10" step="1">

B. <c:forEach var="count" begin="1" end="10" step="1">

C. <c:forEach test="count" beg="1" end="10" step="1">

D. <c:forEach varName="count" val="1" end="10" inc="1">

E. <c:forEach var="count" start="1" end="10" step="1">

【答】 B。

20. 在JSTL的< c:choose>标签中可以出现哪两个标签? ( )

A. case　　B. choose

C. check　　D. when

E. otherwise

【答】 D、E。

21. 为下面各段代码添入合法的属性名或标签名。

① <c:forEach var="movie" items="{movieList}" [ ] ="foo">
${movie}
</c:forEach>

② <c:if [ ] ="${userPref= ='safety'}" >
Mybe you should just walk…
</c:if>

③ <c:set var="userLevel" scope="session" [ ] ="foo" />

④ <c:choose>
<c:[ ] [ ] ="${userPref = = 'performance'}">
Now you can stop even if you do drive insanely fast.
</c:[ ] >
<c:[ ] >
Our brakes are the best.
</c:[ ] >
</c:choose>

【答】 ① varStatus ② test ③ value ④ when, test,when, otherwise, otherwise

第 7 章　JDBC 数据库访问

7.1　本章要点

几乎所有的 Web 应用程序都需要访问数据库。在 Java 应用程序中通过 JDBC 访问数据库，JDBC 是 Java 程序访问数据库的 API，其基本功能包括：建立与数据库的连接；发送 SQL 语句；处理数据库操作结果。

JDBC API 由 java.sql 和 javax.sql 两个包组成，其中定义了若干接口和类。传统的数据库连接的步骤包括：（1）加载驱动程序。（2）使用 DriverManager 类的 getConnection()建立数据库 Connection 连接对象。（3）通过 Connection 对象创建语句 Statement 对象。（4）执行 SQL 语句使用 Statement 对象的方法。（5）用 close()关闭使用完的对象。

下面代码可创建数据库连接对象。

```
Connection dbconn = null;
String driver = "org.postgresql.Driver";
String dburl = "jdbc:postgresql://127.0.0.1:5432/postgres";
String username = "postgres";
String password = "postgres";
try{
    Class.forName(driver); // 加载驱动程序
    // 创建连接对象
    dbconn = DriverManager.getConnection(
                          dburl,username,password);
}catch(ClassNotFoundException e1){
    System.out.println(e1);
}catch(SQLException e2){}
```

如果在 Servlet 的 doGet()或 doPost()中使用上述代码连接数据库，对每次 HTTP 请求都要创建一个连接对象，Web 容器建立连接对象、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，这将增大请求的响应时间。此外，有些数据库支持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。

Web 应用程序在启动时事先建立若干连接存放到连接池中，当程序执行需要一个连接对象时就使用数据源对象的 getConnection()从连接池中取出一个建立对象，使用完后当关闭连接对象时，系统并不将连接对象立即关闭，而是将该连接对象返回给连接池供其他请求使用。

数据源是通过 javax.sql.DataSource 接口对象实现的，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。

通过数据源获得数据库连接对象不能直接在应用程序中通过创建一个实例的方法来生成 DataSource 对象，而是需要采用 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。

在 Tomcat 中可以建立全局数据源和局部数据源。建立局部数据源是在 Web 应用程序的 META-INF 目录中建立一个 context.xml 文件，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>

<Context reloadable = "true">
<Resource
     name="jdbc/sampleDS"
     type="javax.sql.DataSource"
     maxActive="4"
     maxIdle="2"
     username="postgres"
     maxWait="5000"
     driverClassName="org.postgresql.Driver"
     password="postgres"
     url="jdbc:postgresql://localhost/postgres"/>
</Context>
```

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，如下面代码可以获得 jdbc/sampleDS 数据源的引用。

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

得到 DataSource 对象的引用后，就可以通过它的 getConnection()获得数据库连接对象 Connection。

DAO（Data Access Object）称为数据访问对象，它是一种 Java 设计模式。主要目的是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分离，从而使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。

7.2 实验指导

【实验目的】

1. 了解和掌握传统数据库连接方法。
2. 掌握连接池和使用数据源对象访问数据库方法。
3. 掌握 DAO 设计模式。

【实验内容】

实验题目 1：修改第 3 章实验题目 2，用数据库存放商品信息。首先建立一个与 Product 类对应的数据库表 product，代码如下：

```
CREATE TABLE product(
  id integer PRIMARY KEY ,
  name character varying(20),
  description character varying(100),
  price real
)
```

修改 ShoppingCartServlet 类的 init()，代码如下：

```
private Connection conn = null;
@Override
public void init() throws ServletException {
    String username="postgres";
    String password="postgres";
    String driver = "org.postgresql.Driver";
    String url="jdbc:postgresql://localhost:5432/postgres";
    String sql = "SELECT * FROM product";
    try{
       Class.forName(driver);   // 加载驱动程序
       // 创建连接对象
       conn = DriverManager.getConnection(url,username, password);
    }catch(Exception e){
       System.out.println(e);
    }
    try{
       PreparedStatement pstmt = conn.prepareStatement(sql);
       ResultSet rst = pstmt.executeQuery();
       while(rst.next()){
           int id = rst.getInt(1);
           String name = rst.getString(2);
           String description = rst.getString(3);
           float price = rst.getFloat(4);
           Product product = new Product(id,name,description,price);
           products.add(product);
       }
    }catch(SQLException e){
       e.printStackTrace();
    }
}
```

提示：必须将数据库驱动程序复制到 WEB-INF\lib 目录中。

实验题目 2：开发一个实现下面功能的 Web 项目，在表单中输入学生信息，存储到数据库表中，同时在下方显示表中所有记录，并提供一个“删除”链接，当点击该链接时将该记录删除，如图 7-1 所示。

图 7-1　addStudent.jsp 页面

本应用采用 DAO 设计模式访问数据库，数据库只包含一个 students 表，代码如下：

```
CREATE TABLE students(
   id serial NOT NULL,
   name character varying(20),
   age integer,
   email character varying(40),
   CONSTRAINT pkey PRIMARY KEY (id)
)
```

Student 类作为模型类，包含 4 个属性：id、name、age 和 email。Student 类的代码如下。

```
package com.demo;

public class Student {
    private int id;
    private String name;
    private int age;
    private String email;
    // 这里省略属性的 getter 和 setter 方法
}
```

BaseDao 类实现数据库连接，StudentDao 接口定义了数据库访问方法，StudentDaoImpl 类是实现类，StudentServlet 是控制器类。

BaseDao 类的代码如下。

```
package com.dao;

import java.sql.Connection;
import java.sql.DriverManager;
public class BaseDao{
    public Connection getConnection(){
        String username="postgres";
        String password="postgres";
```

```
        String driver = "org.postgresql.Driver";
        String url="jdbc:postgresql://localhost:5432/postgres";
        try{
            Class.forName(driver);
            return DriverManager.getConnection(url,username, password);
        }catch(Exception e){
            System.out.println(e);
        }
        return null;
    }
    public static void main(String[] args){
      Connection conn = new BaseDao().getConnection();
      System.out.println(conn);
    }
}
```

StudentDao 接口的代码如下。

```
package com.dao;
import java.util.*;
import com.bhu.Student;

public interface StudentDao {
    public boolean addStudent(Student s);
    public List<Student> listStudent();
    public int removeStudent(int id);
}
```

StudentDaoImpl 类的代码如下。

```
package com.dao;

import java.util.*;
import java.sql.*;
import com.bhu.Student;

public class StudentDaoImpl extends BaseDao implements StudentDao {
    Connection conn = getConnection();
    // 添加学生方法
    public boolean addStudent(Student s){
       String sql = "INSERT INTO students(name,age,email) VALUES (?,?,?)";
       try{
         PreparedStatement pstmt = conn.prepareStatement(sql);
         pstmt.setString(1, s.getName());
         pstmt.setInt(2, s.getAge());
         pstmt.setString(3,s.getEmail());
         pstmt.executeUpdate();
         return true;
       }catch(SQLException sqle){
          System.out.println(sqle);
          return false;
        }
```

```
    }
    // 检索学生方法
    public List<Student> listStudent(){
       String sql = "SELECT * FROM students";
       List<Student> list = new ArrayList<Student>();
       try{
          PreparedStatement pstmt = conn.prepareStatement(sql);
          ResultSet rs = pstmt.executeQuery();
          while(rs.next()){
            int id = rs.getInt("id");
            String name = rs.getString(2);
            int age = rs.getInt(3);
            String email = rs.getString(4);
            Student s = new Student();
            s.setId(id);
            s.setName(name);
            s.setAge(age);
            s.setEmail(email);
            list.add(s);
          }
          return list;
       }catch(SQLException sqle){
          System.out.println(sqle);
          return null;
       }
    }
    // 删除学生方法
    public int removeStudent(int id){
        String sql = "DELETE FROM students WHERE id=?";
        try{
            PreparedStatement pstmt = conn.prepareStatement(sql);
           pstmt.setInt(1, id);
           return pstmt.executeUpdate();
        }catch(SQLException sqle){
            System.out.println(sqle);
            return 0;
        }
    }
}
```

StudentServlet 类是控制器，实现学生记录的添加、显示和删除，代码如下。

```
package com.demo;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.dao.*;
```

```
import java.util.*;
@WebServlet(name = "addStudentServlet", urlPatterns = { "/addStudent.do" })
public class StudentServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;
   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
       doPost(request,response);
   }
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                         throws ServletException, IOException {
     String action = request.getParameter("action");
     if(action!=null&&action.equals("addStudent")){
         addStudent(request,response);
     }else if(action.equals("remove")){
         removeStudent(request,response);
     }else{
        listStudent(request,response);
     }
   }
   // 添加学生方法
   public void addStudent(HttpServletRequest request,
                               HttpServletResponse response)
         throws ServletException, IOException {
    String name =new String(
          request.getParameter("name").getBytes("iso-8859-1"),"utf-8");
    int age = Integer.parseInt(request.getParameter("age"));
    String email = new String(
          request.getParameter("email").getBytes("iso-8859-1"),"utf-8");
    Student s = new Student();
    s.setName(name);
    s.setAge(age);
    s.setEmail(email);

    StudentDao dao = new StudentDaoImpl();
    boolean success= dao.addStudent(s);
    if(success){
        String message = "插入记录成功";
        request.setAttribute("msg", message);
        listStudent(request,response);
    }else{
     RequestDispatcher rd = request.getRequestDispatcher("error.jsp");
     rd.forward(request, response);
    }
  }
  // 显示学生信息
  public void listStudent(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
    StudentDao dao = new StudentDaoImpl();
```

```
    List<Student> list= dao.listStudent();
    request.setAttribute("studentList", list);
    RequestDispatcher rd = request.getRequestDispatcher("addStudent.jsp");
     rd.forward(request, response);
  }
  public void removeStudent(HttpServletRequest request,
                            HttpServletResponse response)
                            throws ServletException, IOException {
     int id = Integer.parseInt(request.getParameter("id"));
     StudentDao dao = new StudentDaoImpl();
     int success=dao.removeStudent(id);
     if(success>0){
         listStudent(request,response);
     }
  }
}
```

下面的 addStudent.jsp 页面用来显示学生信息。

```
<%@ page contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"%>
<%@ page import="java.util.*,com.demo.*" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>添加学生信息</title></head>
<body>

<form action="addStudent.do?action=addStudent" method="post">
<p>请输入学生信息</p>
 ${msg}<br>
 姓名<input type="text" name="name" value="王小明"/><br>
 年龄<input type="text" name="age" value="20"/><br>
 Email<input type="text" name="email" value="wangxiaoming@163.com"/><br>
 <input type="submit" value="确定"/><input type="reset" value="重置"/>
</form>
<hr/>
<table>
<tr><td>学号</td><td>姓名</td><td>年龄</td>
     <td>邮件地址</td><td>是否删除</td></tr>
<c:forEach var="s" items="${studentList}">
  <tr>
     <td>${s.id}</td><td>${s.name}</td><td>${s.age }</td>
     <td>${s.email}</td>
     <td><a href="addStudent.do?action=remove&id=${s.id}" >删除</a></td>
  </tr>
</c:forEach>
</table>
</body>
</html>
```

实验题目 3：将上述实验的数据库连接代码改为使用数据源对象连接数据库。

（1）首先在 META-INF 目录下建 context.xml 文件，内容如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<Context reloadable = "true">
<Resource
      name="jdbc/sampleDS"
      type="javax.sql.DataSource"
      maxActive="4"
      maxIdle="2"
      username="postgres"
      maxWait="5000"
      driverClassName="org.postgresql.Driver"
      password="postgres"
      url="jdbc:postgresql://localhost:5432/postgres"/>
</Context>
```

这里，jdbc/asmpleDS 即为数据源名。

（2）修改 BaseDao 类的 getConnection()，使用数据源对象创建连接。

【思考题】

1. 使用数据源对象访问数据库比传统方法有哪些优点？
2. 简述 DAO 设计模式。

7.3 习题解析

1. 简述 Java 数据库访问的两层和三层模型。

【答】 两层模型即客户机/服务器模型，在两层模型中应用程序直接通过JDBC驱动程序访问数据库。三层模型是浏览器/应用服务器/数据库服务器结构，在该结构中浏览器向应用服务器发出请求，应用服务器通过JDBC驱动程序访问数据库。

2. 简述传统的数据库连接的步骤，这种方法有什么缺点？

【答】 传统的数据库连接的一般步骤是：（1）加载 JDBC 驱动程序。（2）建立连接对象。（3）创建语句对象。（4）执行 SQL 语句得到结果集对象，调用 ResultSet 的有关方法就可以完成对数据库的操作。（5）关闭建立的各种对象。

缺点是每次访问数据库都要建立连接对象，请求结束需关闭连接对象。这将耗费大量的时间，可能导致增大请求的响应时间。

3. 使用 Class 类的 forName()加载驱动程序需要捕获什么异常？（　　）

A. SQLException　　B. IOException

C. ClassNotFoundException　　D. DBException

【答】 C。

4. 程序若要连接 Oracle 数据库，请给出连接代码。数据库驱动程序名是什么？数据库 JDBC URL 串的内容是什么？

【答】 连接 Oracle 数据库代码如下。

```
Class.forName("oracle.jdbc.driver.OracleDriver");
String dburl = "jdbc:oracle:thin:@127.0.0.1:1521:ORCL";
Connection conn = Drivermanager.getConnection(dburl, "scott", "tiger");
```

上述代码中，oracle.jdbc.driver.OracleDriver 为 JDBC 驱动程序名，jdbc:oracle:thin:@127.0.0.1:1521:ORCL 为 JDBC URL。

5. 试说明使用数据源对象连接数据库的优点是什么？通过数据源对象如何获得连接对象？

【答】 使用数据源是目前 Web 应用开发中建立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，存放在连接池中。当应用程序需要一个连接对象时就从连接池中取出一个，使用完后再放回连接池。这样就可避免每次请求都创建连接对象，从而降低请求的响应时间，提高效率。

使用数据源建立连接是通过 JNDI 技术实现的。这需要首先配置数据源（可以是局部数据源或全局数据源），然后在应用程序中通过 Context 对象查找数据源对象。假设已经配置了名为 sampleDS 的数据源，建立连接代码如下：

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

6. 试说明什么是可滚动和可更新的结果集对象，如何得到可更新且可滚动的 ResultSet 对象？

【答】 可滚动的 ResultSet 是指在结果集对象上不但可以向前访问结果集中的记录，还可以向后访问结果集中的记录。可更新的 ResultSet 是指不但可以访问结果集中的记录，还可以通过结果集对象更新数据库。

要创建可滚动、可更新的 ResultSet 对象，必须使用 Connection 对象的带两个参数的 createStatement()创建的 Statement，第一个参数用下面两个常量之一：

- ResultSet.TYPE_SCROLL_SENSITIVE
- ResultSet.TYPE_SCROLL_INSENSITIVE

第二个参数使用下面常量：

- ResultSet.CONCUR_UPDATABLE

7. 编写一个 Servlet，查询 books 表中所有图书的信息并在浏览器中通过表格的形式显示出来。

参考程序如下：

```
package com.control;
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class BookQueryServlet extends HttpServlet{
  Connection dbconn;
  public void init() {
    String driver = "org.postgresql.Driver";
    String dburl = "jdbc:postgresql://127.0.0.1:5432/bookstore";
    String username = "bookstore";
    String password = "bookstore";
    try{
      Class.forName(driver);
      dbconn = DriverManager.getConnection(
                          dburl,username,password);
    }catch(ClassNotFoundException e1){
    }catch(SQLException e2){}
  }
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException,IOException{
    response.setContentType("text/html;charset=gb2312");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("<table>");
    try{
      String sql="SELECT * FROM books";
      Statement stmt = dbconn.createStatement();
      ResultSet rst = stmt.executeQuery(sql);
      while(rst.next()){
        out.println("<tr><td>"+rst.getString(1)+"</td>");
        out.println("<td>"+rst.getString(2)+"</td>");
        out.println("<td>"+rst.getString(3)+"</td>");
        out.println("<td>"+rst.getString(4)+"</td>");
        out.println("<td>"+rst.getDouble(5)+"</td></tr>");
      }
    }catch(SQLException e){
      e.printStackTrace();
    }
    out.println("</table>");
    out.println("</body></html>");
  }
  public void destroy(){
    try {
       dbconn.close();
    }catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

8. 修改本章程序 7.4 的 displayProduct.jsp 页面，使用表达式语言显示商品信息。修改后的 displayProduct.jsp 页面代码如下：

```
<%@ page contentType="text/html; charset=utf-8" %>
```

```
<html>
<head><title>商品信息</title></head>
<body>
<table>
<tr><td>商品号：</td><td>${product.prod_id} /></td>
</tr>
<tr><td>商品名： </td><td>${product.pname} /></td>
</tr>
<tr><td>价格：</td><td>${product.price} /></td>
</tr>
<tr><td>库存量：</td><td>${product.stock} /></td>
</tr>
</table>
</body></html>
```

9. 修改本章程序 7.5 的 displayAllProduct.jsp 页面，使用 JSP 标准标签显示商品信息，使该页面成为无脚本页面。

修改后的 displayAllProduct.jsp 页面代码如下：

```
<%@ page contentType="text/html; charset=UTF-8"
                          pageEncoding="UTF-8"%>
<%@ page import="java.util.*,com.demo.Product" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>显示所有商品</title></head>
<body>
<table>
<tr><td>商品号</td><td>商品名</td><td>价格</td><td>数量</td></tr>
<c:forEach var="product" items="${sessionScope.productList}"
           varStatus="status">
   <%--为奇数行和偶数行设置不同的背景颜色--%>
   <c:if test="${status.count%2==0}">
     <tr style="background:#eeeeff">
   </c:if>
   <c:if test="${status.count%2!=0}">
     <tr style="background:#dedeff">
   </c:if>
   <%--用 EL 访问作用域变量的成员--%>
   <td>${product.prod_id}</td>
   <td>${product.pname}</td>
   <td>${product.price}</td>
   <td>${product.stock}</td>
   </tr>
</c:forEach>
</table>
</body></html>
```

10. 请为本章的 CustomerDao.java 程序增加两个方法实现删除和修改客户信息，这两个方法的格式为：

```
public boolean deleteCustomer(String custName)
```

```
public boolean updateCustomer(Customer customer)
```

首先在 SampleDAO 类中定义下面两个字符串常量：

```
private static final String DELETE_SQL =
             "DELETE FROM customer WHERE custName = ?";
private static final String UPDATE_SQL =
             "UPDATE customer SET email=? , phone=? WHERE custName=?";
```

下面是删除客户和修改客户的方法：

```
// 按姓名删除客户记录
public boolean deleteCustomer(String custName){
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rst = null;
    CustomerBean  customer =null;
    try{
      conn = dataSource.getConnection();
      pstmt = conn.prepareStatement(DELETE_SQL);
      pstmt.setString(1,custName);
      int n = pstmt.executeUpdate();
      if(n ==1){
         return true;
      }else{
         return false;
      }
    }catch(SQLException se){
        return false;
    }finally{
       try{
         pstmt.close();
         conn.close();
       }catch(SQLException se){}
     }
   }

   // 修改客户记录
   public boolean updateCustomer(CustomerBean customer){
      Connection conn = null;
      PreparedStatement pstmt = null;
      try{
        conn = dataSource.getConnection();
        pstmt = conn.prepareStatement(UPDATE_SQL);
        pstmt.setString(1,customer.getEmail());
        pstmt.setString(2,customer.getPhone());
        pstmt.setString(3,customer.getCustName());
        int n = pstmt.executeUpdate();
        if(n ==1){
           return true;
        }else{
           return false;
```

```
        }
      }catch(SQLException se){
         return false;
      }finally{
        try{
          pstmt.close();
          conn.close();
        }catch(SQLException se){}
      }
    }
```

11. 编写一个名为 SelectCustomerServlet 的 Servlet，在其中使用 CustomerDao 类的 findByName()，实现客户查询功能，然后将请求转发到 displayCustomer.jsp 页面，显示查询结果。

参考代码如下：

```
@WebServlet("/SelectCustomerServlet")
public class SelectCustomerServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                         throws ServletException, IOException {
       CustomerDao dao = new CustomerDao();
       Customer customer = null;
        String cname = request.getParameter("custName");
        customer= dao.findByName(cname);
       request.setAttribute("customer", customer);
       RequestDispatcher rd =
               request.getRequestDispatcher("/displayCustomer.jsp");
       rd.forward (request,response);
       return;
    }
}
```

12. 开发一个如图 7-2 所示的应用程序，其功能是插入和删除数据库表中学生记录。当插入一条学生记录后，程序应显示表中所有记录。要求数据库连接使用数据源，数据库操作通过 DAO 设计模式实现。

图 7-2　添加学生记录页面

【答】　略。

13. 编写一个简单的在线考试程序。要求试题和答案存放在数据库中，程序运行时通过一个链接进入考试页面，考试页面显示题目，答案选项使用单选按钮。答题结束按提交按钮，系统自动给出成绩。

【答】　略。

第 8 章　Servlet 高级应用

8.1　本章要点

本章介绍了几种 Servlet 高级应用，其中包括 Web 监听器、Web 过滤器、Servlet 的多线程问题以及 Servlet 3.0 新增的异步处理问题。

Web应用程序运行过程中可能发生各种事件，如ServletContext事件、会话事件及请求有关的事件等，Web容器采用监听器模型处理这些事件。Web应用程序中的事件主要发生在三个对象上：ServletContext、HttpSession和ServletRequest对象。表8-1列出了所有事件类和监听器接口。

表 8-1　Web 事件类与监听器接口

监听对象	事件	监听器接口
ServletContext	ServletContextEvent	ServletContextListener
	ServletContextAttributeEvent	ServletContextAttributeListener
HttpSession	HttpSessionEvent	HttpSessionListener
		HttpSessionActivationListener
	HttpSessionBindingEvent	HttpSessionAttributeListener
		HttpSessionBindingListener
ServletRequest	ServletRequestEvent	ServletRequestListener
	ServletRequestAttributeEvent	ServletRequestAttributeListener
	AsyncEvent	AsyncListener

事件的类型主要包括对象的生命周期事件和属性改变事件。生命周期事件是作用域对象创建和销毁时发生的事件，属性改变事件是在作用域对象上添加属性、删除属性或替换属性时发生的事件。

处理这些事件应实现相应的监听器接口，在实现的方法中编写处理代码。要使监听器对象起作用，还必须注册监听器。在 Servlet 3.0 的容器中可以使用两种方法注册监听器，一是使用@WebListener 注解，二是在 web.xml 中使用<listener>及子元素<listener-class>注册监听器。

过滤器（Filter）是Web服务器上的组件，过滤器用于拦截传入的请求或传出的响应，并监视、修改或以某种方式处理这些通过的数据流。

Filter接口是过滤器API的核心，所有的过滤器都必须实现该接口。该接口声明了三个方法，分别是init()、doFilter()和destroy()，它们是过滤器的生命周期方法。此外，还包括

FilterConfig接口和FilterChain接口。

Servlet 规范中提到的过滤器的一些常见应用包括：验证过滤器、登录和审计过滤器、数据压缩过滤器、加密过滤器、XSLT 过滤器等。

@WebFilter 注解用于将一个类声明为过滤器，该注解在部署时被容器处理，容器根据具体的配置将相应的类部署为过滤器。

还可以在web.xml文件中使用<filter>和<filter-mapping>元素配置过滤器。每个<filter>元素向Web应用程序引进一个过滤器，每个<filter-mapping>元素将一个过滤器与一组请求URI关联。

在Web应用程序中，一个Servlet在一个时刻可能被多个用户同时访问，Web容器将为每个用户创建一个线程。如果Servlet不涉及共享资源的问题，不必关心多线程问题，但如果Servlet需要共享资源，则要保证Servlet是线程安全的。

下面是编写线程安全的Servlet的一些建议：

（1）用方法的局部变量保存请求中的专有数据。

（2）只用 Servlet 的成员变量来存放那些不会改变的数据。

（3）如果 Servlet 访问外部资源，那么需要对这些资源同步。

在 Servlet 3.0 中可以使用异步线程对请求进行处理，在 Servlet 中一般需要完成下面操作：

- 调用ServletRequest对象的startAsync()，该方法返回AsyncContext对象，它是异步处理的上下文对象。
- 调用AsyncContext对象的setTimeout()，传递一个毫秒时间设置容器等待指定任务完成的时间。如果没有设置超时时间，容器将使用默认时间。在指定的时间内任务不能完成将抛出异常。
- 调用AsyncContext对象的start()，为其传递一个要在异步线程执行的Runnable对象。
- 当任务结束时在线程对象中调用AsyncContext对象的complete()或dispatch()。

8.2 实验指导

【实验目的】

1. 掌握常用 Web 监听器的编写方法。
2. 掌握简单过滤器的开发方法。

【实验内容】

实验题目 1：编写一个会话监听器，用来记录访问当前应用程序的会话数量。该程序使用 Integet 对象存储会话数量，作为 ServletContext 的一个属性。

```
package com.listener;

import java.util.concurrent.atomic.AtomicInteger;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
```

```
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

@WebListener
public class MySessionListener implements HttpSessionListener,
                                  ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext servletContext = sce.getServletContext();
        servletContext.setAttribute("userCounter",
               new Integer(0));
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
    }
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        HttpSession session = se.getSession();
        ServletContext servletContext = session.getServletContext();
        Integer userCounter = (Integer) servletContext
               .getAttribute("userCounter");
        int userCount = ++userCounter;
        System.out.println("当前用户数为 :" + userCount);
    }
    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        HttpSession session = se.getSession();
        ServletContext servletContext = session.getServletContext();
        Integer userCounter = (Integer) servletContext
               .getAttribute("userCounter");
        int userCount = --userCounter;
        System.out.println("当前用户数为 :" + userCount);
    }
}
```

实验题目 2：编写一个监听器程序，用来记录请求一个资源所花费的时间。该监听器应该实现 ServletRequestListener 接口，当请求开始时执行 requestInitialized()，当请求结束时调用 requestDestroyed()，代码如下。

```
package com.listener;

import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpServletRequest;

@WebListener
```

```
public class PerformanceListener implements ServletRequestListener {
    @Override
    public void requestInitialized(ServletRequestEvent sre) {
        ServletRequest servletRequest = sre.getServletRequest();
        servletRequest.setAttribute("start", System.nanoTime());
    }
    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        ServletRequest servletRequest = sre.getServletRequest();
        Long start = (Long) servletRequest.getAttribute("start");
        Long end = System.nanoTime();
        HttpServletRequest httpServletRequest =
               (HttpServletRequest) servletRequest;
        String uri = httpServletRequest.getRequestURI();
        System.out.println("请求  " + uri +
               " 花费的时间是:"  + ((end - start) / 1000) + " 毫秒");
    }
}
```

实验题目 3：创建一个过滤器改变请求编码。

```
package com.filter;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;

@WebFilter(filterName="EncodingFilter",urlPatterns={"/*"},
           initParams={@WebInitParam(name="encoding",value="UTF-8")}
           )
public class EncodingFilter implements Filter {
    protected String encoding = null;
    protected FilterConfig config;
    public void init(FilterConfig filterConfig) throws ServletException {
        this.config = filterConfig;
        // 得到在过滤器的初始化参数
        this.encoding = filterConfig.getInitParameter("encoding");
    }
    public void doFilter(
        ServletRequest request,
        ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        if (request.getCharacterEncoding() == null) {
            // 得到指定的编码
            String encode = getEncoding();
            if (encode != null) {
                //设置 request 的编码
                request.setCharacterEncoding(encode);
                response.setCharacterEncoding(encode);
            }
```

```
        }
        chain.doFilter(request, response);
    }
    protected String getEncoding() {
        return encoding;
    }
    public void destroy() {
    }
}
```

该 Filter 被应用在所有请求（/*），它将把所有的请求和响应的编码设置为“UTF-8”。

【思考题】

1. 简述在 Web 项目中可以使用的监听器有哪些？
2. 简述过滤器的主要作用？

8.3 习题解析

1. Web应用程序的哪些对象上可以发生事件，如何实现监听器接口，如何注册事件监听器？

【答】 3个对象上可发生事件：ServletContext、HttpSession和HttpRequest。针对不同的事件，应实现不同的监听器接口。如对ServletContextEvent应实现ServletContextListener接口。注册事件监听器可以使用@WebListener注解或在web.xml文件中使用<listener>元素及其子元素<listener-class>实现。

2. Web应用程序启动时将通知应用程序的哪个事件监听器？

【答】 Web应用程序启动时将通知ServletContextListener事件监听器。

3. 在Web部署描述文件web.xml中注册监听器时需要使用<listener>元素，该元素的唯一一个子元素是（ ）。

A. <listener-name> B. <listener-class>

C. <listener-type> D. <listener-class-name>

【答】 B。

4. 假设你编写了一个名为MyServletRequestListener的类监听ServletRequestEvent事件，如何在部署描述文件中配置该类？

【答】

```
<listener>
   <listener-class>MyServletRequestListener</listener-class>
</listener>
```

5. 下面代码是实现了ServletRequestAttributeListener接口的类的部分代码，且该监听器已在DD中注册：

```
public void attibuteAdded(ServletRequestAttributeEvent ev){
   getServletContext().log("A: "+ ev.getName()+" ->" +ev.getValue());
}
public void attibuteRemoved(ServletRequestAttributeEvent ev){
   getServletContext().log("M: "+ ev.getName()+" ->" +ev.getValue());
}
public void attibuteReplaced(ServletRequestAttributeEvent ev){
   getServletContext().log("P: "+ ev.getName()+" ->" +ev.getValue());
}
```

下面是一个Servlet中doGet()的代码：

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
            throws IOException,ServletException{
   request.setAttibute("a", "b");
   request.setAttibute("a", "c");
   request.removeAttibute("a");
}
```

试问如果客户访问该Servlet，在日志文件中生成的内容为（　　）。

A. A: a->b P: a->b　　B. A: a->b M: a->c

C. A: a->b P: a->b M: a->c　　D. A: a->b M: a->b P: a->c M: a->c

【答】 C 。

6. 在部署描述文件中的<filter-mapping>元素中可以使用哪三个元素？（　　）

A. <servlet-name>　　B. <filter-class>　　C. <dispatcher>

D. <url-pattern>　　E. <filter-chain>

【答】 A、C、D。

7. 下面代码有什么错误？（　　）

```
public void doFilter(ServletRequest req, ServletResponse, res,
                     FilterChain chain)
                throws ServletException, IOException {
    chain.doFilter(req, res);
    HttpServletRequest request = (HttpServletRequest)req;
    HttpSession session = request.getSession();
    if (session.getAttribute("login") == null) {
      session.setAttribute("login"", new Login());
    }
 }
```

A. doFilter()格式不正确，应该带的参数为HttpServletRequest和HttpServletResponse

B. doFilter()应该抛出FilterException异常

C. chain.foFilter(req,res)调用应该为this.doFilter(req,res,chain)

D. 在chain.doFilter()之后访问request对象将产生IllegalStateException异常

E. 该过滤器没有错误

【答】 E。

8. 给定下面过滤器声明：

```
<filter-mapping>
  <filter-name>FilterOne</filter-name>
  <url-pattern>/admin/*</url-pattern>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
<filter-mapping>
  <filter-name>FilterTwo</filter-name>
  <url-pattern>/users/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>FilterThree</filter-name>
  <url-pattern>/admin/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>FilterTwo</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

在浏览器中输入请求/admin/index.jsp，将以哪个顺序调用过滤器？（ ）

A. FilterOne, FilterThree　　B. FilterOne, FilterTwo, FilterThree

C. FilterTwo, FilterThree　　D. FilterThree, FilterTwo

E. FilterThree

【答】 D。

9. 编写线程安全的 Servlet，下面哪个是最佳方法？（ ）

A. 实现 SingleThreadModel 接口　　B. 对 doGet()或 doPost()同步

C. 使用局部变量存放用户专有数据　　D. 使用成员变量存放所有数据

【答】 C。

10. 下面哪种方法可在Servlet中启动一个异步线程?（ ）

A. 调用请求对象的 startAsync()　　B. 调用 AsyncContext 的 start()

C. 调用线程对象的 run()　　D. 调用 AsyncContext 的 complete()

【答】 B。

11. 简述开发支持异步线程调用的Servlet的一般步骤。

参考答案：（1）调用request对象的startAsync()返回AsyncContext对象，它是异步处理的上下文对象。（2）调用AsyncContext对象的setTimeout()，传递一个毫秒时间设置容器等待指定任务完成的时间。在指定的时间内任务不能完成将抛出异常。（3）调用AsyncContext对象的start()，为其传递一要异步执行的Runnable对象。（4）当任务结束时在线程对象中

调用AsyncContext对象的complete()或dispatch()。

第 9 章　Web 安全性入门

9.1　本章要点

Web 应用程序通常包含许多资源，这些资源可被多个用户访问，有些资源要求用户必须具有一定权限才能访问。安全性已成为 Web 应用程序开发中关键问题之一。

Web 应用的安全性措施主要包括下面 4 个方面：身份验证、授权、数据保密性和数据完整性。在 Servlet 规范中定义了如下 4 种用户验证的机制：① HTTP Basic 验证；② HTTP Digest 验证；③ HTTPS Client 验证；④ HTTP FORM-based 验证。

实施 Web 应用程序的安全性可以有两种方法：声明式安全和编程式安全。声明式安全是通过 web.xml 文件而不是通过程序指定安全约束。要使用声明式安全首先必须定义角色和用户，它们可以存储在文件中或数据库表中。在 Tomcat 中，角色和用户信息可以存储在 conf\tomcat-users.xml 文件中，在该文件中定义了一些角色和用户，还可以在该文件中增加或修改角色和用户以及用户和角色的映射，

为 Web 资源定义安全约束是通过 web.xml 文件实现的，这里主要配置哪些角色可以访问哪些资源。安全约束的配置主要是通过<security-constraint>、<login-config>和<security-role>三个元素实现。

如果用户访问使用基本验证保护的资源，服务器首先返回 401（Unauthorized）响应消息，该消息包含一个 WWW-Authorization 头，下面是典型例子：

```
HTTP/1.1 401 Authorization Require
Servler:Apache-Coyote/1.1
Date:Wed, 21 Dec 2011 11:32:49 GMT
WWW-Authorization:Basic realm="Security Test"
```

浏览器收到上述消息后首先显示一个对话框要求用户输入用户名和密码，将其使用 Base64 算法进行编码发送到服务器，若用户名和密码正确，服务器将资源发送给用户。

为了提高用户名和密码的安全性，可以使用 HTTP 摘要验证，它除了口令是以加密的方式发送的，其他与基本验证都一样，但比基本验证安全。

要使用摘要验证，应在<login-config>的<auth-method>指定为 DIGEST。

```
<login-config>
  <auth-method>DIGEST</auth-method>
  <realm-name>Digest authentication</realm-name>
</login-config>
```

基于表单的验证需要提供登录页面和错误页面，并将<login-config>的<auth-method>指定为 FORM。

```
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>Security Test</realm-name>
   <form-login-config>
     <form-login-page>/login.jsp</form-login-page>
     <form-error-page>/error.jsp</form-error-page>
   </form-login-config>
</login-config>
```

编程式安全可以提供更精细的安全性管理，它主要通过请求对象的有关方法实现，这些方法包括：getRemoteUser()、isUserInRole()、authenticate()、login()和 logout()等。

9.2 实验指导

【实验目的】

1. 了解 Web 应用的安全性措施。
2. 掌握基本身份验证和基于表单的身份验证的方法。
3. 了解编程式身份验证。

【实验内容】

实验题目 1：基于声明式的基本身份验证。

（1） 修改 Tomcat 的用户配置文件。打开 conf\tomcat-users.xml 文件，在其中添加下面的角色和用户的定义：

```
<role rolename="manager"/>
<role rolename="member"/>
<user name="mary" password="mmm" roles="manager,member" />
<user name="bob" password="bbb" roles="member" />
```

这里增加了两个角色 manager 和 member，增加了两个用户 mary 和 bob，其中 mary 具有 manager 和 member 角色。

注意：修改了用户配置文件后需要重新启动 Tomcat。

（2）编写一个 SampleServlet 作为资源，代码如下。

```
package com.demo;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.RequestDispatcher;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(urlPatterns={"/sample"})
public class SampleServlet extends HttpServlet{
```

```
    public void doGet(HttpServletRequest request,
                HttpServletResponse response)
                  throws IOException,ServletException {
       RequestDispatcher rd =
            request.getRequestDispatcher("/jsp/demo.jsp");
       rd.forward(request, response);
    }
}
```

（3）在部署描述文件 web.xml 中定义所保护的资源。

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>JSP pages</web-resource-name>
     <url-pattern>*.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint />
</security-constraint>

<security-constraint>
   <web-resource-collection>
     <web-resource-name>sample servlet</web-resource-name>
     <url-pattern>/sample</url-pattern>
   </web-resource-collection>
   <auth-constraint>
     <role-name>manager</role-name>
     <role-name>member</role-name>
   </auth-constraint>
   <user-data-constraint>
     <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
</security-constraint>

<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>Security Test</realm-name>
</login-config>
```

（3）访问 SampleServlet，首先显示标准对话框提示输入用户名和密码，若输入正确，显示正常页面，否则显示错误页面。

实验题目 2：在上一实验的基础上完成基于表单的验证。

（1）首先需要建立登录页面和出错页面。

登录页面 login.jsp 的代码如下。

```
<%@ page contentType="text/html;charset=UTF-8" %>
<html><head>
<title>登录页面</title></head>
<body bgcolor="white">
<p>请您输入用户名和口令：</p>
<form method="POST" action="j_security_check">
```

```
  <table>
    <tr>
      <td>用户名：</td>
      <td><input type="text" name="j_username"><td>
    </tr>
    <tr>
      <td>密  码：</td>
      <td><input type="password" name="j_password"></td>
    </tr>
    <tr>
      <td><input type="submit" value="登录"></td>
      <td><input type="reset" value="重置"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

错误页面 error.jsp 的代码如下。

```
<%@ page contentType="text/html;charset=UTF-8"%>
<html>
<body>
  <p style="color:red">对不起，你的用户名和口令不正确！</p>
</body>
</html>
```

（2）修改 web.xml 文件，指定采用基于表单的验证方法。

```
<login-config>
   <auth-method>FORM</auth-method>
   <realm-name>Security Test</realm-name>
   <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/error.jsp</form-error-page>
   </form-login-config>
</login-config>
```

（3）访问 SampleServlet 首先显示用户定制的 login.jsp 页面，输入正确的用户名和密码，则显示请求的资源，否则显示错误页面 error.jsp。

【思考题】

1. Web 应用安全性包括哪些方面？
2. 试说明声明式安全和编程式安全？

9.3 习题解析

1. Web应用的安全性主要涉及哪些方面？假如你要进入一栋大楼，需要向保卫人员出示有关证件，这属于哪方面的安全问题？

【答】 Web应用的安全性主要包括4个方面：（1）身份验证；（2）授权；（3）数据完整性；（4）数据保密性。进入大楼出示证件属于身份验证。

2. Servlet规范中定义了哪4种验证用户的机制？各有什么优缺点？

【答】 验证用户的机制包括：（1）HTTP基本验证。优点：实现简单。缺点：用户名和口令没有加密。（2）HTTP摘要验证。优点：用户名和口令加密，比基本验证安全。（3）HTTPS客户证书验证。优点：是最安全的。缺点：需要授权机构的证书。（4）基于表单的验证。优点：实现容易。缺点：用户名和口令不加密。

3. 试比较Web应用程序的声明式的安全性与编程式的安全性有何异同？

【答】 声明式安全和程序式安全是Web应用实现安全性的两种方法。声明式安全是在程序外配置安全信息，程序式安全是在程序内实施安全措施。

4. 下面哪一条正确地定义了数据完整性？（ ）

A. 它保证信息只能被某些用户访问

B. 它保证信息在服务器上以加密的形式保存

C. 它保证信息在客户和服务器之间传输时不被无意的用户读取

D. 它保证信息在客户和服务器之间传输时不被修改

【答】 D。

5. 在Web应用程序部署描述文件中下面哪个元素用来指定验证机制？（ ）

A. security-constraint　　B. auth-constraint

C. login-config　　D. web-resource-collection

【答】 C。

6. 下面哪三个元素用来定义安全约束？只选择是<security-constraint>元素直接子元素的元素。（ ）

A. login-config　　B. role-name

C. role　　D. transport-guarantee

E. user-data-constraint　　F. auth-constraint

G. authorization-constraint　　H. web-resource-collection

【答】 E、F、H。

7. 在下面哪两个web.xml 文件片段能正确标识sales目录下所有HTML文件？（ ）

A.
```
<web-resource-collection>
    <web-resource-name>reports</web-resource-name>
    <url-pattern>/sales/*.html</url-pattern>
</web-resource-collection>
```

B.
```
<resource-collection>
    <web-resource-name>reports</web-resource-name>
    <url-pattern>/sales/*.html</url-pattern>
</resource-collection>
```

C. <resource-collection>
<resource-name>reports</resource-name>
<url-pattern>/sales/*.html</url-pattern>
</resource-collection>

D. <web-resource-collection>
<web-resource-name>reports</web-resource-name>
<url-pattern>/sales/*.html</url-pattern>
<http-method>GET</http-method>
</web-resource-collection>

【答】 A、D。

8. 下面关于验证机制的叙述哪两个是正确的？（ ）

A. HTTP Basic 验证传输的用户名和密码是以明文传输的

B. HTTP Basic 验证使用HTML 表单获得用户名和口令

C. Basic和FORM机制验证的传输方法是相同的

D. Basic和FORM机制获得用户名和口令的方法是相同的

【答】 A、C。

9. 假设Web应用程序要采用基于表单的验证机制，下面是登录页面的部分代码和web.xml文件的部分代码，请在方框中填上正确的内容。

登录页面代码如下：

```
Please input your name and password:
<form method="post" action = [          ] >
  <input type = "text" name = [          ] >
  <input type = "password" name = "j_password" >
  <input type = "submit" value="Enter" >
</form>
```

web.xml 文件代码如下：

```
<login-config>
  <auth-method> [          ] </auth-method>
  <form-login-config>
    < [          ]>/loginPage.jsp< [          ]>
    <form-error-page>/errorPage.jsp</form-error-page>
 </form-login-config>
 </login-config>
```

【答】 ① j_security_check ② j_username ③ FORM ④ form-login-page ⑤/form-login-page

10. 关于未验证的用户，下面哪两个叙述是正确的？（ ）

A. HttpServletRequest.getUserPrincipal() 返回null

B. HttpServletRequest.getUserPrincipal() 抛出SecurityException异常

C. HttpServletRequest.isUserInRole(rolename) 返回false

D. HttpServletRequest.getRemoteUser() 抛出SecurityException异常

【答】 A、C。

第 10 章　Ajax 技术基础

10.1　本章要点

Ajax 技术是 Web 2.0 阶段系列技术和相关产品服务中非常重要的一种技术。Ajax 是英文 Asynchronous JavaScript and XML 的缩写，意思为异步 JavaScript 与 XML。作为一种客户端技术，Ajax 应用实现客户浏览器与服务器的异步交互。

Ajax 不仅仅只包含异步 JavaScript 和 XML，它实际上是包含多种技术的一个综合技术，其中包括 JavaScript 脚本、XHTML、CSS、DOM、XML、XSTL 以及最重要的 XMLHttpRequest 对象。实际上，Ajax 是包含允许浏览器与服务器异步通信而无需完全刷新当前页面的所有技术。

XMLHttpRequest 是浏览器中定义的对象，它是 Ajax 技术中的核心对象。通过 JavaScript 脚本可以创建 XMLHttpRequest 对象。XMLHttpRequest 对象定义了若干属性和方法，通过这些属性和方法就可以向服务器发出异步请求和处理响应结果。

下面代码展示了通过跨浏览器的 JavaScript 脚本创建 XMLHttpRequest 对象。

```
var xmlHttp;
function createXMLHttpRequest(){
   if(window.ActiveXObject){
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }else if(window.XMLHttpRequest){
      xmlHttp = new XMLHttpRequest();
   }
}
```

从上述代码可以看到，创建 XMLHttpRequest 对象非常容易，可以使用全局变量引用 XMLHttpRequest 对象。

XMLHttpRequest 对象通过各种属性和方法为客户提供服务。常用的属性有 status、onreadystatechange、readyState、responseText、responseXML 等，常用的方法有 open()、send() 等。

图10-1说明了Ajax应用中标准的交互模式。

Ajax 的交互模式从客户触发事件开始，然后创建 XMLHttpRequest 对象。在向服务器发出请求之前，应该通过 XMLHttpRequest 对象的 onreadystatechange 属性设置回调函数。当 XMLHttpRequest 对象的内部状态改变时调用回调函数，因此回调函数是处理响应的地方。

```
function startRequest() {
```

```
    createXMLHttpRequest();
    xmlHttp.onreadystatechange = handleStateChange;
    xmlHttp.open("GET", "simpleResponse.xml", true);
    xmlHttp.send(null);
}
```

这里 handleStateChange 就是回调函数，通过回调函数可以对响应结果进行处理。在回调函数中首先应该检查 XMLHttpRequest 对象的 readyState 属性和 status 属性的值。当 readyState 属性值为 4、status 属性的值为 200 时表示响应完成，这时才能使用 XMLHttpRequest 对象的 responseText 或 responseXML 检索请求结果。例如，下面是一个回调函数：

```
function handleStateChange() {
    if(xmlHttp.readyState == 4) {
        if(xmlHttp.status == 200) {
            alert("The server replied with: " + xmlHttp.responseText);
        }
    }
}
```

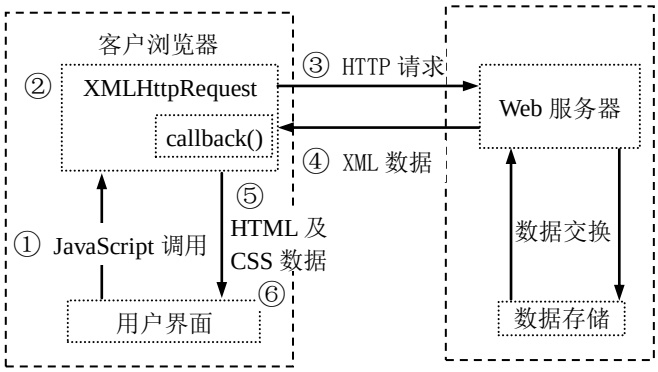

图 10-1　Ajax 的工作原理

Ajax 技术在 Web 应用开发中有很多应用，本书通过一些例子介绍几种常见的应用，其中包括数据验证、动态加载列表框、创建工具提示、动态更新 Web 页面等。

10.2　实验指导

【实验目的】

1. 了解 Ajax 相关的各种技术，包括 JavaScript、XML、DOM 等技术。
2. 掌握开发基于 Ajax 技术的应用的基本步骤。

【实验内容】

实验题目：编写一个使用邮箱的注册页面，邮箱信息存储在数据库中，当用户用邮箱

注册，焦点离开文本框时引发一个动作，通过 Ajax 技术向服务器发送请求，检查该邮箱名是否已被使用，若是给出提示，否则可以注册。

首先建立一个数据库表 emailtab，其中包含两个字段 email 和 password 分别表示 Email 地址和密码。

register.jsp 页面如下：

```
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head><title>用户注册</title>
<script type="text/javascript">
  var xmlHttp;
  function createXMLHttpRequest() {
    if (window.ActiveXObject) {
       xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    } else if (window.XMLHttpRequest) {
       xmlHttp = new XMLHttpRequest();
    }
  }

  function validate() {
     createXMLHttpRequest();
     var myemail = document.getElementById("myemail");
     var url = "validation.do?myemail=" + escape(myemail.value);
     xmlHttp.open("GET", url, true);
     xmlHttp.onreadystatechange = handleStateChange;
     xmlHttp.send(null);
  }

  function handleStateChange() {
    if(xmlHttp.readyState == 4){
      if(xmlHttp.status == 200){
        var message = xmlHttp.responseXML.
            getElementsByTagName("message")[0].firstChild.data;
        var messageArea = document.getElementById("results");
        messageArea.innerHTML = "<p>" + message + "</p>";
      }
    }
  }
 </script>
</head>
<body>
<form action="register.action" method="post">
<p>个人用户注册</p>
<table>
<tr>
   <td>*我的邮箱：</td>
   <td><input type="text" id="myemail" size="20" onblur="validate();"></td>
</tr>
<tr>
```

```
   <td></td>
   <td><div id="results">请输入邮箱地址作为登录账号</div></td>
</tr>

<tr>
   <td>*请输入密码：</td>
   <td><input type="text" id="password" size="20"></td>
</tr>
<tr>
   <td><input type="submit" name="submit" value="提交"></td>
   <td><input type="reset" name="reset" value="重置"></td>
   <td></td>
</tr>
</table>
</form>
</body>
</html>
```

该页面运行结果如图 10-2 所示。

图 10-2　register.jsp 页面运行结果

当用户输入邮箱焦点离开文本框时调用 validate()函数，该函数向 ValidationServlet 发送请求检查该邮箱地址是否被使用，若已被使用则向客户返回信息。

ValidationServlet 代码如下。

```
package ajax.demo;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
import java.sql.*;

@WebServlet(name = "validationServlet", urlPatterns = { "/validation.do" })
public class ValidationServlet extends HttpServlet{
    Connection dbconn = null;
    public void init() {
        String driver = "org.postgresql.Driver";
        String dburl = "jdbc:postgresql://127.0.0.1:5432/postgres";
        String username = "postgres";
        String password = "postgres";
```

```
        try{
           Class.forName(driver);
           dbconn = DriverManager.getConnection(
                                dburl,username,password);
         }catch(ClassNotFoundException e1){
             System.out.println(e1);
         }catch(SQLException e2){}
      }

    public void doGet(HttpServletRequest request,
                   HttpServletResponse response)
                throws ServletException,IOException{
      response.setContentType("text/xml;charset=UTF-8");
      response.setHeader("Cache-Control","no-cache");

      String message = "该邮箱可以被使用";
      String myemail = request.getParameter("myemail");
      String sql = "SELECT * FROM emailtab WHERE email=?";
      try{
         PreparedStatement pstmt = dbconn.prepareStatement(sql);
         pstmt.setString(1, myemail);
         ResultSet rst = pstmt.executeQuery();
         if(rst.next())
            message = "该邮箱已被使用，请更换其它邮箱！";
      }catch(SQLException sqle){
         System.out.println(sqle);
      }
      // 向客户发送 XML 响应数据
      PrintWriter out = response.getWriter();
      out.println("<response>");
      out.println("<message>"+message+"</message>");
      out.println("</response>");
    }
}
```

【思考题】

1. Ajax 技术主要应用在什么情况下？
2. 使用 Ajax 有什么好处？

10.3 习题解析

1. 什么是 Ajax？它主要实现什么功能？

【答】 Ajax是英文Asynchronous JavaScript and XML的缩写，意思为异步JavaScript与XML。该技术主要实现客户与服务器的异步通信，实现页面的部分刷新。

2. 如何创建 XMLHttpRequest 对象？

【答】 可通过跨浏览器的 JavaScript 脚本创建 XMLHttpRequest 对象。

```
var xmlHttp;
function createXMLHttpRequest(){
   if(window.ActiveXObject){
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }else if(window.XMLHttpRequest){
      xmlHttp = new XMLHttpRequest();
   }
}
```

3. 调用 XMLHttpRequest 对象的哪个方法向服务器发出异步请求？（　　）

A. send()　　B. open()

C. getRequestHeader()　　D. abort()

【答】　A。

4. 使用 XMLHttpRequest 对象的哪个属性可以得到从服务器返回的 XML 数据？（）

A. responseText　　B. responseXML

C. responseBody　　D. statusTex

【答】　B。

5. 若返回文档中由 id 指定的元素，应该使用文档对象 element 的什么方法？（　　）

A. getElementByTagName()　　B. getElementById()

C. getAttribute()　　D. hasChildNodes()

【答】　B。

第 11 章　Hibernate 框架基础

11.1　本章要点

Hibernate 是一个对象/关系映射（ORM）框架，它用来解决面向对象语言中对象与关系数据库中关系的不匹配问题。Hibernate 通过映射文件和配置文件把 Java 持久化对象 PO 映射到数据库中，然后通过操作 PO，对数据表进行插入、删除、修改和查询等操作。

Hibernate 的运行过程如图 11-1 所示。

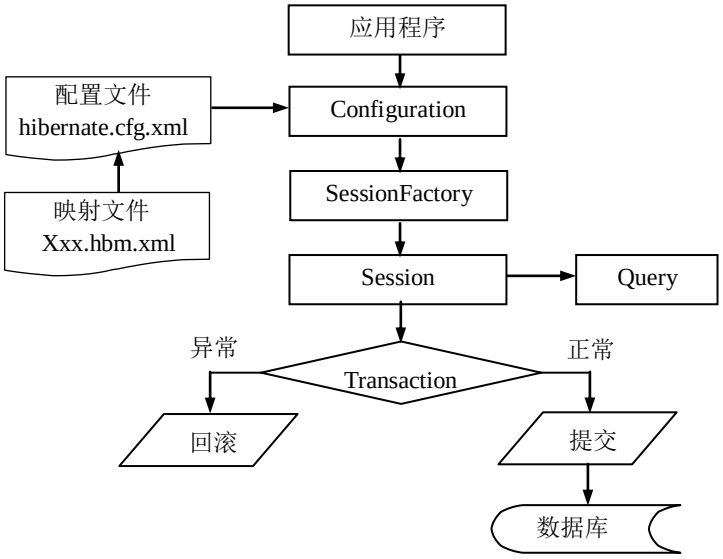

图 11-1　Hibernate 应用的执行过程

应用程序首先创建 Configuration 对象，该对象读取 Hibernate 配置文件和映射文件的信息，并用这些信息创建一个 SessionFactory 会话工厂对象，然后从 SessionFactory 对象生成一个 Session 会话对象，并用 Session 对象生成 Transaction 事务对象。最后，通过 Session 对象的 save()、get()、load()、update()和 delete()等方法对 PO 进行操作，Transaction 对象将把这些操作结果提交到数据库中。如果要进行查询，可以通过 Session 对象生成一个 Query 对象，然后调用 Query 对象的 list()或 iterate()执行查询操作，在返回的 List 对象或 Iterator 对象上迭代，即可访问数据库数据。

持久化类是一种轻量级的持久化对象，它通常与关系数据库中的表对应，每个持久化对象与表中的一行对应。Hibernate 的持久化对象分为三种状态：临时态（transient）、持久态（persistent）和脱管态（detached）。

Hibernate 的映射文件把一个 PO 与一个数据表映射起来。每个持久化类都应该有一个映射文件。在 Hibernate 中有多种类型的映射，常用的关联映射有四种类型：一对一、一对多、多对一和多对多。此外，本书还介绍了组件属性映射和继承映射。

Hibernate 配置文件用来配置 Hibernate 运行的各种信息，在 Hibernate 应用开始运行时要读取配置文件信息。配置文件可以使用属性文件的格式，文件名为 hibernate.properies，也可以使用 XML 文件格式，文件名为 hibernate.cfg.xml，在 Hibernate 系统中使用后者比较方便一些。配置文件中最重要的是数据库连接属性的指定。

```
<property name="connection.driver_class">
            org.postgresql.Driver</property>
<property name="connection.url">
            jdbc:postgresql://localhost:5432/postgres</property>
<property name="connection.username">postgres</property>
<property name="connection.password">postgres</property>
```

此外，在配置文件中还经常需要配置数据库方言、数据库连接池、会话管理特征、缓存特征、以及是否自动建表等。

数据查询是 Hibernate 的最常见操作。Hibernate 提供了多种查询方法：HQL、条件查询、本地 SQL 查询和命名查询等。HQL（Hibernate Query Language）称为 Hibernate 查询语言，它是 Hibernate 提供的一种功能强大的查询语言。HQL 与 SQL 类似，用来执行对数据库的查询，它的很多语法都来自 SQL 语言，如支持 where 子句、order by 子句、group by 子句、允许使用聚集函数、支持带参数查询以及连接查询等。

HQL 的查询结果是 Query 对象，调用该对象的 list()返回 List，调用 Query 对象的 iterate()返回 Iterator 对象，之后就可以在 List 或 Iterator 对象上迭代，返回查询结果对象。

下面代码说明如何使用 list()返回 List 对象，然后通过其 get()检索每个 Student 持久类实例。

```
String query_str="from Student as s";
Query query = session.createQuery(query_str);
List<Student> list = query.list();
for(int i = 0; i < list.size(); i ++){
    Student stud =(Student)list.get(i);
    System.out.println("学号: " + stud.getStudentNo());
    System.out.println("姓名: " + stud.getStudentName());
}
```

HQL 查询语句中可带参数，在执行查询语句之前需要设置参数。如果使用的是命名参数，应该使用 setParameter()设置，如果使用的是占位符（？）参数，则应该使用 setXxx()设置。

对来自多个表的数据，可以使用连接查询。在 Hibernate 中则使用关联映射来处理底层数据表之间的连接。一旦建立了正确的关联映射后，就可利用 Hibernate 的关联来进行连接。

HQL 支持两种关联连接形式：显式（explicit）连接和隐式（implicit）连接。显式连接需要使用 join 关键字，这与 SQL 连接表类似。隐式连接不需要使用 join 关键字，仅需使

用“点号”来引用相关实体。隐式连接可使用在任何 HQL 语句中，但在最终的 SQL 语句中仍以 inner join 的方式出现。

除 HQL 外 Hibernate 还提供了其他查询技术，包括条件查询、使用本地 SQL 语句查询以及命名查询等。

11.2 实验指导

【实验目的】

1. 掌握 Hibernate 开发环境的构建以及映射文件的定义与配置文件的建立。
2. 掌握 Hibernate 核心 API 的使用，包括 Session、Transaction 等。
3. 掌握关联映射、组件映射以及继承映射。
4. 掌握 HQL 语言的使用，了解其他查询语言。

【实验内容】

实验题目：开发一个 Hibernate 项目实现对 Student 对象操作。

（1） 新建项目，将 Hibernate 软件包解压目录的 lib/requried 目录中的 JAR 文件复制到 WEB-INF/lib 目录中，将数据库驱动包也复制到该目录中。

（2）在 PostgreSQL 中创建 student 数据库表，代码如下。

```
CREATE TABLE student(
   id bigserial PRIMARY KEY,
   student_no  bigint,
   student_name  character varying(50),
   sage   integer,
   major  character varying(15) DEFAULT NULL
);
```

（3）定义持久化类 Student

```
package com.hibernate;
public class Student {
   private Long id;
    private long studentNo;
    private String studentName;
    private int sage;
    private String major;
    public Student() { }
    public Student(long studentNo, String studentName, int sage,
                     String major){
       this.studentNo = studentNo;
       this.studentName = studentName;
       this.sage = sage;
       this.major = major;
    }
    public Long getId(){
```

```
        return id;
    }
    public void setId(Long id){
        this.id = id;
    }
    public long getStudentNo(){
        return studentNo;
    }
    public void setStudentNo(long studentNo){
        this.studentNo = studentNo;
    }
    public String getStudentName(){
        return studentName;
    }
    public void setStudentName(String studentName){
        this.studentName = studentName;
    }
    public int getSage(){
        return sage;
    }
    public void setSage(int sage){
        this.sage = sage;
    }
    public String getMajor(){
        return major;
    }
    public void setMajor(String major){
        this.major = major;
    }
}
```

（3）定义映射文件 Student.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="com.hibernate">
     <class name="Student" table="student">
         <id name="id" column="id">
              <generator class="sequence" />
         </id>
         <property name="studentNo" type="long" column="student_no" />
         <property name="studentName" type="string" column="student_name" />
         <property name="sage" type="integer" column="sage" />
         <property name="major" type="string" column="major" />
     </class>
</hibernate-mapping>
```

映射文件保存在与持久化类相同的目录中。

（4）创建配置文件 hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
   <!-- 指定数据库连接参数 -->
   <property name="connection.driver_class">
            org.postgresql.Driver</property>
   <property name="connection.url">
            jdbc:postgresql://localhost:5432/postgres
   </property>
   <property name="connection.username">postgres</property>
   <property name="connection.password">postgres</property>

   <!-- 指定JDBC 连接池-->
   <property name="connection.pool_size">1</property>
   <!-- 指定SQL 方言 -->
   <property name="dialect">
       org.hibernate.dialect.PostgreSQLDialect
   </property>
   <!-- 指定将所有执行的SQL语句回显到stdout -->
   <property name="show_sql">true</property>
   <!-- 指定在启动时对表进行检查 -->
   <property name="hibernate.hbm2ddl.auto">validate</property>
   <!-- 指定映射文件，若有多个映射文件，使用多个mapping元素指定 -->
   <mapping resource="com/hibernate/Student.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

（5）创建 HibernateUtil.java 工具类文件，代码如下。

```
package com.util;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;
import org.hibernate.service.ServiceRegistryBuilder;

public class HibernateUtil {
    private static SessionFactory factory;
    private static ServiceRegistry serviceRegistry;
    static{
       try{
          Configuration configuration = new Configuration().configure();
          serviceRegistry = new ServiceRegistryBuilder()
             .applySettings(configuration.getProperties())
             .buildServiceRegistry();
```

```
            factory = configuration.buildSessionFactory(serviceRegistry);
        }catch(HibernateException e){
           e.printStackTrace();
        }
    }
  // 返回会话工厂对象
  public static SessionFactory getSessionFactory() {
      return factory;
  }
  // 返回一个会话对象
  public static Session getSession() {
      Session session = null;
      if(factory !=null)
          session = factory.openSession();
      return session;
  }
  // 关闭指定的会话对象
  public static void closeSession(Session session){
     if(session !=null){
         if(session.isOpen())
             session.close();
     }
  }
}
```

（6）编写 listStudent.jsp 页面，代码如下。

```
<%@ page contentType="text/html; charset=UTF-8"
          pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>添加学生信息</title></head>
<body>
<form action="studentManage?action=addStudent" method="post">
<p>请输入学生信息</p>
   学号<input type="text" name="sno" />
   姓名<input type="text" name="name" /><br>
   年龄<input type="text" name="age" />
   专业<input type="text" name="major"/><br>
 <input type="submit" value="确定"/><input type="reset" value="重置"/>
</form>
 ${msg}<hr/>
<table>
<tr><td>学号</td><td>姓名</td><td>年龄</td><td>专业</td>
            <td>删除</td><td>修改</td></tr>
<c:forEach var="s" items="${studentList}">
  <tr>
     <td>${s.studentNo}</td><td>${s.studentName}</td>
     <td>${s.sage }</td><td>${s.major}</td>
     <td><a href="studentManage?action=delete&id=${s.id}" >删除</a></td>
     <td><a href="studentManage?action=edit&id=${s.id}" >修改</a></td>
```

```
  </tr>
</c:forEach>
</table>
</body>
</html>
```

运行该页面结果如图 11-2 所示。

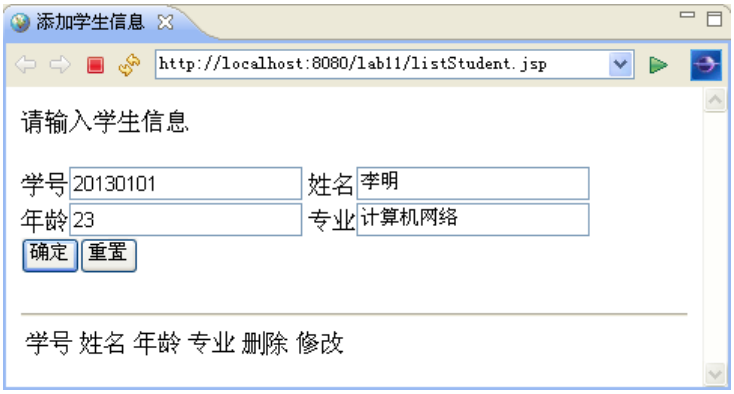

图 11-2　listStudent.jsp 页面运行结果

（7）编写 StudentServlet 类实现对 Student 对象的插入、删除和修改

```
package com.hibernate;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.HibernateException;
import com.util.HibernateUtil;
import java.io.PrintWriter;

@WebServlet(name = "StudentServlet", urlPatterns = { "/studentManage" })
public class StudentServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // GET请求和POST请求都由doPost()处理
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                         throws ServletException, IOException {
       doPost(request,response);
     }

    protected void doPost(HttpServletRequest request,
```

```
                    HttpServletResponse response)
                throws ServletException, IOException {
      String action = request.getParameter("action");
      // 根据action请求参数的不同执行不同的方法
      if(action!=null&&action.equals("addStudent")){
          addStudent(request,response);
      }else if(action.equals("delete")){
          deleteStudent(request,response);
      }else if(action.equals("update")){
         updateStudent(request,response);
       }else if(action.equals("edit")){
          editStudent(request,response);
       }else{
          listStudent(request,response);
    }
}
// 插入学生记录方法
protected void addStudent(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
    Session session = HibernateUtil.getSession();
     Transaction tx = null;
     Long studentID = null;
    try{
      tx = session.beginTransaction();
      long sno = Long.parseLong(request.getParameter("sno"));
      // 字符编码转换
      String name =new String(request.getParameter("name")
                   .getBytes("iso-8859-1"),"utf-8");
      int age = Integer.parseInt(request.getParameter("age"));
      String major = new String(request.getParameter("major")
                   .getBytes("iso-8859-1"),"utf-8");
      Student student = new Student(sno,name,age,major);
      // 持久化student对象
      studentID = (Long) session.save(student);
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
    }finally {
      session.close();
    }
    if(studentID!=null){
       String message = "插入记录成功! ";
       request.setAttribute("msg", message);
       listStudent(request,response);
    }else{
        RequestDispatcher rd = request.getRequestDispatcher("error.jsp");
        rd.forward(request, response);
    }
}
```

```
// 显示学生信息
public void listStudent(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
    Session session = HibernateUtil.getSession();
    Transaction tx = null;
    List students=null;
    try{
       tx = session.beginTransaction();
       // 查询学生信息
       students = session.createQuery("FROM Student").list();
       tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   }finally {
      session.close();
   }
   // 将控制转发到listStudent.jsp页面
   if(students!=null){
      request.setAttribute("studentList", students);
      RequestDispatcher rd =
            request.getRequestDispatcher("listStudent.jsp");
      rd.forward(request, response);
    }
}
// 编辑学生信息
public void editStudent(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {
    Long studentID = Long.parseLong(request.getParameter("id"));
    Student s=null;
    Session session = HibernateUtil.getSession();
    Transaction tx = null;
    try{
      tx = session.beginTransaction();
      s = (Student)session.get(Student.class, studentID);
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
       e.printStackTrace();
    }finally {
       session.close();
    }
    // 显示要修改的学生信息
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>修改学生信息</title></head>");
    out.println("<body>");
    out.println("<p>修改学生信息</p>");
    out.println("<form action='studentManage?action=update'
```

```
                        method='post'>");
    out.println("<input type='hidden' name='id' value='"+s.getId()+"'/>");
    out.println("学号<input type='text' name='sno' value='"+
                s.getStudentNo()+"'/>");
    out.println("姓名<input type='text' name='name' value='"+
                s.getStudentName()+"'/><br>");
    out.println("年龄<input type='text' name='age' value='"+
                s.getSage()+"'/>");
    out.println("专业<input type='text' name='major' value='"+
                s.getMajor()+"'/><br>");
    out.println("<input type='submit' value='修改'/>"
                +"<input type='reset' value='重置'/>");
    out.println("</form>");
    out.println("</body>");
    out.println("</html>");
}
// 修改学生信息
public void updateStudent(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
    Long studentID = Long.parseLong(request.getParameter("id"));
    Student student=null;
    Session session = HibernateUtil.getSession();
    Transaction tx = null;
    try{
      tx = session.beginTransaction();
      student = (Student)session.get(Student.class, studentID);
      long sno = Long.parseLong(request.getParameter("sno"));
      String name =new String(request.getParameter("name").
                   getBytes("iso-8859-1"),"utf-8");
      int age = Integer.parseInt(request.getParameter("age"));
      String major = new String(request.getParameter("major")
                  .getBytes("iso-8859-1"),"utf-8");
      // 用表单输入值修改student对象属性值
      student.setStudentNo(sno);
      student.setStudentName(name);
      student.setSage(age);
      student.setMajor(major);
      // 更新持久化对象
      session.update(student);
      tx.commit();
    }catch (HibernateException e) {
      if (tx!=null) tx.rollback();
       e.printStackTrace();
    }finally {
      session.close();
    }
     listStudent(request,response);
  }
  // 删除学生记录
  protected void deleteStudent(HttpServletRequest request,
```

```
                                HttpServletResponse response)
                                throws ServletException, IOException {
    Long studentID = Long.parseLong(request.getParameter("id"));
    Session session = HibernateUtil.getSession();
    Transaction tx = null;
    try{
      tx = session.beginTransaction();
      Student student = (Student)session.get(Student.class, studentID);
      // 删除持久化对象
      session.delete(student);
      tx.commit();
    }catch (HibernateException e) {
       e.printStackTrace();
       String message = "删除记录失败！";
       request.setAttribute("msg", message);
       if (tx!=null) tx.rollback();
    }finally {
      session.close();
    }
    listStudent(request,response);
  }
}
```

插入一些记录后显示的页面如图 11-3 所示。

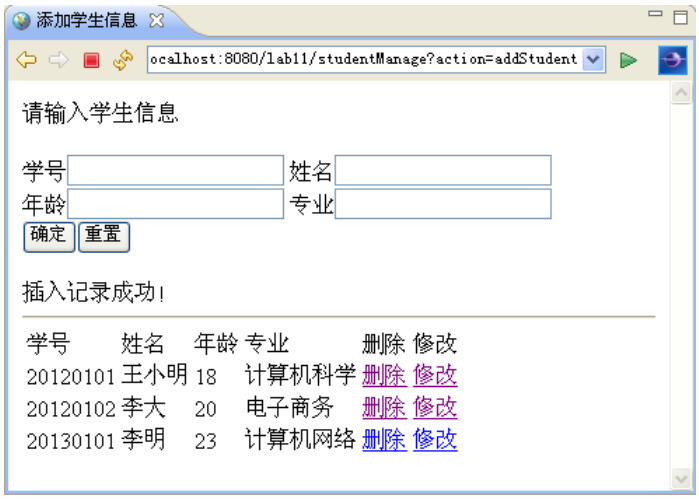

图 11-3　添加记录后页面显示结果

当单击某个记录的“修改”按钮后显示的页面如图 11-4 所示。

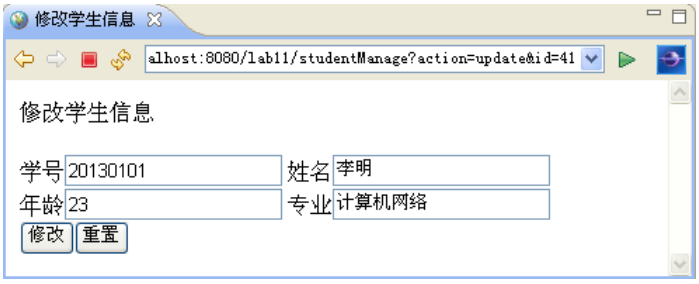

图 11-4　修改记录的页面

【思考题】

1. 使用 ORM 映射框架操作数据库与传统方法操作数据库相比有什么优点？
2. 在 Hibernate 中有哪些查询方法？

11.3　习题解析

1. 什么是 ORM，它要解决什么问题？

【答】　ORM是Object/Relation Mapping的缩写，称为对象/关系映射。它主要解决面向对象语言中的对象与关系数据库中关系的不匹配问题。Hibernate是实现对象/关系映射的一个框架。

2. 映射文件的作用是（　　）。

A. 定义数据库连接参数
B. 建立持久化类和数据表之间的对应关系
C. 创建持久化类
D. 自动建立数据库表

【答】　B。

3. Hibernate 的配置文件的主要作用是什么？

【答】　Hibernate 的配置文件用来配置 Hibernate 运行的各种信息，包括：数据库连接信息、数据库方言、连接池、缓存策略、映射文件、是否自动建表等信息。

4. 在 Hibernate 中一个持久化类对象可能处于三种状态之一，下面哪个是不正确的？（　　）。

A. 持久状态　　B. 临时状态
C. 固定状态　　D. 脱管状态

【答】　C。

5. 假设有一个 Student 持久化类，它的映射文件名是（　　）。

A. Student.mapping.xml　　B. Student.hnm.xml
C. hibernate.properties　　D. hibernate.cfg.xml

【答】　B。

6. 要让 Hibernate 自动创建数据表，应在配置文件中如何设置？

【答】 在配置文件中将 hibernate.bhm2ddl.auto 属性值指定为 create，如下所示。

```
<property name="hibernate.hbm2ddl.auto">create</property>
```

7. 若建立两个持久化类的双向关联，需要（　　）。

A. 在一方添加多方关联的属性

B. 在多方添加一方关联的属性

C. 在一方和多方都添加对方的属性

D. 不需要在某一方添加对象的属性

【答】 C。

8. 在 Hibernate 的继承映射中有三种实现方式，不包括下面哪一种（　　）？

A. 所有类映射成一张表　　B. 每个子类映射成一张表

C. 每个具体子类映射成一张表　　D. 只将超类映射成一张表

【答】 D。

9. HQL 支持带参数的查询语句，下面哪个是正确的（　　）？

A. HQL 只支持命名参数　　B. HQL 只支持占位符（?）参数

C. HQL 支持命名参数和占位符参数　　D. HQL 不支持动态参数

【答】 C。

10. 如果使用 Hibernate 命名查询，SQL 语句应该定义在（　　）中？

A. 持久化类文件　　B. *.hnm.xml 映射文件

C. hibernate.properties　　D. hibernate.cfg.xml

【答】 B。

第 12 章　Struts 2 框架基础

12.1　本章要点

Struts 2 是基于 MVC 设计模式的 Web 应用程序开发框架，它主要包括控制器、Action 对象、视图 JSP 页面和配置文件等。

- 控制器：控制器由过滤器、拦截器或 Action 组件实现。
- 模型：模型由 JavaBeans 实现，它可实现业务逻辑。
- 视图：通常由 JSP 页面实现，也可以由 Velocity Template、FreeMarker 或其他表示层技术实现。
- 配置文件：Struts 2 框架提供 struts.xml 配置文件，使用它来配置应用程序中的组件。
- Struts 2 标签：Struts 2 提供了一个功能强大的标签库，该库提供了大量标签，使用这些标签可以简化 JSP 页面的开发。

在 Struts 中一切活动都是从用户触发动作开始的，用户触发动作有多种方式：在浏览器的地址栏中个输入一个 URL，点击页面的一个链接，填写表单并单击提交按钮。所有这些操作都可以触发一个动作。

动作类的任务就是处理用户动作，在 Struts 2 中充当控制器。当发生一个用户动作时，请求将经由过滤器发送到 Action 动作类。Struts 将根据配置文件 struts.xml 中的信息确定要执行 Action 对象的哪个方法。通常是调用 Action 对象的 execute()执行业务逻辑或数据访问逻辑，Action 类执行后根据结果选择一个资源发送给客户。资源可以是视图页面，也可能是 PDF 文件、Excel 电子表格等。动作类可以实现 Action 接口，但通常继承 ActionSupport 类。

配置文件用来建立动作 Action 类与视图的映射关系。当客户请求 URL 与某个动作名匹配时，Struts 将使用 struts.xml 文件中的映射处理请求。动作映射在 struts.xml 文件中使用<action>标签定义。在该文件中为每个动作定义一个映射，Struts 根据动作名确定执行哪个 Action 类，根据 Action 类的执行结果确定请求转发到哪个视图页面。

OGNL（Object-Graph Navigation Language）称为对象-图导航语言，它是一种简单的、功能强大的表达式语言。使用 OGNL 表达式语言可以访问存储在 ValueStack 和 ActionContext 中的数据。

访问 ValueStack 中的数据可用 Struts 的下面语法。

```
<s:property value="[0].user.username"/>
<s:property value="user.username"/>
<s:property value="[0]['message']"/>
```

Stack Context 中包含下列对象：application、session、request、parameters、attr。这些对象的类型都是 Map，可在其中存储“键/值”对数据。

要访问 Stack Context 中的对象需要给 OGNL 表达式加上一个前缀字符“#”，“#”相当于 ActionContext.getContext()，可以使用以下几种形式之一：

```
#object.propertyName
#object['propertyName']
#object["propertyName"]
```

Struts 2 框架提供了一个标签库，使用这些标签很容易地在页面中动态访问数据，创建动态响应。Struts 2 的标签可以分为两大类：通用标签和用户界面（UI）标签，如表 12-1 所示。

表 12-1 Struts 2 的常用标签

标签分类		标 签
通用标签	数据标签	a、action、bean、date、debug、i18n、include、param、push、set、text、url、property
	控制标签	if、elseif、else、append、generator、iterator、merge、sort、subset
UI 标签	表单标签	checkbox、checkboxlist、combobox、doubleselect、file、form、hidden、label、optiontransferselect、optgroup、password、radio、reset、select、submit、textarea、textfield、token、updownselect
	非表单标签	actionerror、actionmessage、component、fielderror、table、
	Ajax 标签	a、autocompleter、datetimepicker、div、head、submit、tabbedPanel、tree、treenode

要使用 Struts 2 的标签，应该使用 taglib 指令导入标签库：

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

通常需要编写有关代码实现输入数据校验，在 Struts 2 中有多种方法实现用户输入校验。

- 使用 Struts 2 校验框架。这种方法是基于 XML 的简单的校验方法，可以对用户输入数据自动校验，甚至可以使用相同的配置文件产生客户端脚本。
- 在 Action 类中执行校验。这是最强大和灵活的方法。Action 中的校验可以访问业务逻辑和数据库等。但是，这种校验可能需要在多个 Action 中重复代码，并要求自己编写校验规则。而且，需要手动将这些条件映射到输入页面。
- 使用注解实现校验。可以使用 Java 5 的注解功能定义校验规则，这种方法的好处是不用单独编写配置文件，所配置的内容和 Action 类放在一起，这样容易实现 Action 类中的内容和校验规则保持一致。
- 客户端校验。客户端校验通常是指通过浏览器支持的各种脚本来实现用户输入校验，这其中最经常使用的就是 JavaScript。在 Struts 2 中可以通过有关标签产生客户端 JavaScript 校验代码。

Web 应用程序可被来自世界不同国家、使用不同语言的人们访问，因此 Web 应用程序应提供国际化的支持。Struts 2 对国际化的支持采用属性（资源）文件来存储国际化信息，具体应用中可在 Action 类中通过 getText()获得国际化信息，也可在 JSP 页面中通过<s:text>标签和<s:i18n>标签实现国际化。

Apache Tiles 是一个模板框架，它用来简化 Web 应用的用户界面的开发。它允许定义页面片段来组装完整的页面。具体步骤如下：

（1）创建模板页面。模板页面是创建其他 JSP 页面的模板，在其中使用 Tiles 的<tiles:insertAttribute>标签作为占位符，这些占位符在定义文件中用具体的值或 JSP 页面替换。

（2）创建定义文件。定义文件用来定义具体的视图页面。通常是先用模板页面定义一个基本布局视图（baseLayout），然后定义逻辑视图继承该基本布局视图。

（3）在 struts.xml 文件中的<package>元素中应添加调用 tiles 的标签处理请求，在动作的定义中将结果的 type 属性指定为“tiles”，结果指定为逻辑视图名。

12.2 实验指导

【实验目的】

1. 掌握 Struts 2 的框架的组成、环境构建和开发步骤。
2. 掌握动作类的开发和如何在配置文件中配置。
3. 掌握 OGNL 的概念和常用 Struts 2 标签的使用。
4. 掌握使用 Tiles 设计页面布局的方法。

【实验内容】

实验题目 1：整合 Struts 2 和 Hibernate 开发一个注册/登录系统。

该应用的需求是：把在注册页面中输入用户名、密码、年龄和 Email 地址等信息存入数据库表。如果我们采用 MVC 设计模式，各层的组件如下：

- 视图层：包括用户注册页面 register.jsp，注册/登录成功页面 success.jsp 和失败页面 error.jsp。
- 控制层：使用 Action 实现控制组件，文件为 RegisterAction.java。
- 模型层：其中包括 PO 类 User.java、工具类 HibernateUtil.java、配置文件 hibernate.cfg.xml 和映射文件 User.hbm.xml。
- 数据层：数据库采用 PostgreSQL，客户表为 customer。

（1）构建 Struts 2 和 Hibernate 开发环境。在 WEB-INF/lib 中添加有关库文件和驱动程序文件。在 web.xml 文件中声明核心过滤器。

（2）在数据库中建立一个名为 userinfo 的数据表，该表有 username、password、age 和 email 字段。

```
CREATE TABLE userinfo(
  id bigserial PRIMARY KEY,             -- ID
```

```
   username varchar(20),                -- 用户名
   password varchar(8) NOT NULL,        -- 口令
   age int,                             -- 年龄
   email varchar(50) UNIQUE             -- Email 地址
);
```

（3）创建持久化类 User

```
package com.demo;

public class User{
    private Long id;
    private String username;
    private String password;
    private int age;
    private String email;
    public User(String username,String password,
                    int age, String email){
       this.username = username;
       this.password = password;
       this.age = age;
       this.email = email;
    }
    // 这里省略属性的 getter 方法和 setter 方法
}
```

（4）创建映射文件 User.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="com.hibernate">
     <class name="User" table="userinfo">
       <id name="id" column="id">
              <generator class="identity" />
       </id>
       <property name="username" type="string" column="username" />
       <property name="password" type="string" column="password" />
       <property name="age" type="integer" column="age" />
       <property name="email" type="string" column="email" />
     </class>
</hibernate-mapping>
```

（5）创建配置文件 hibernate.cfg.xml 中添加映射文件

```
<mapping resource="com/demo/User.hbm.xml"/>
```

（6）编写工具类 HibernateUtil.java 代码同第 11 章实验。

（7）创建动作类 RegisterAction

```
package com.action;
import com.demo.User;
import com.util.HibernateUtil;
import com.opensymphony.xwork2.ActionSupport;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.Query;
import java.util.List;

public class RegisterAction extends ActionSupport {
    private User user;
    public User getUser() {
      return user;
    }
    public void setUser(User user) {
      this.user = user;
    }
    @Override
    public String execute() throws Exception {
      return SUCCESS;
    }
    public String register() throws Exception {
      try{
        Session session = HibernateUtil.getSession();
        Transaction tx = session.beginTransaction();
        session.save(user);  // 将 user 对象持久化到数据表中
        tx.commit();
        return SUCCESS;
      }catch(Exception e){
        e.printStackTrace();
        HibernateUtil.getSession().close();
        return ERROR;
      }
    }

    public String login() throws Exception {
      try{
         Session session = HibernateUtil.getSession();
         Transaction tx = session.beginTransaction();
         Query query = session.createQuery(
                     "from User where username=:uname");
         query.setParameter("uname", user.getUsername());
         List list = query.list();
         tx.commit();
         if(list.size()==1){
           return SUCCESS;
         }else
           return ERROR;
       }catch(Exception e){
          e.printStackTrace();
          HibernateUtil.getSession().close();
```

```
            return ERROR;
        }
    }
}
```

（8）创建结果视图页面

下面是 register.jsp 页面代码：

```
<%@ page contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head><title>用户注册</title></head>
<body>
<p>注册一个新用户</p>
<s:form action="Register">
   <s:actionerror /> <s:fielderror />
   <s:textfield name="user.username" label="用户名" />
   <s:password name="user.password" label="口令" />
   <s:textfield name="user.age"  label="年龄"  />
   <s:textfield name="user.email"  label ="Email 地址"/>
   <s:submit value="注册"/>
</s:form>
</body>
</html>
```

下面是 login.jsp 页面代码：

```
<%@ page contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head><title>登录页面</title></head>
<body>
<p>请输入用户名和密码：</p>
<s:form action="Login">
  <s:textfield name="user.username" label="用户名"
      tooltip="输入用户名" labelposition="left" />
  <s:password name="user.password" label="密码"
      tooltip="输入密码" labelposition="left" />
  <s:submit value="登录" align="center" />
</s:form>
</body>
</html>
```

下面是 success.jsp 页面代码：

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head><title>注册成功页面</title></head>
```

```
<body>
   <p>注册成功！</p>
   <s:property value="user" />
   <p><a href="<s:url action='index' />" >返回 首页</a></p>
</body>
</html>
```

下面是 welcome.jsp 页面代码：

```
<%@ page contentType="text/html;charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head><title>登录成功</title></head>
<body>
    <p align="center"><font color="#000080" size="5">
         欢迎登录本系统</font></p>
</body>
</html>
```

（9）创建配置文件 struts.xml

在 struts.xml 配置文件中添加下面的动作定义。

```
<action name="registerInput">
      <result>/register.jsp</result>
</action>
<action name="loginInput">
      <result>/login.jsp</result>
</action>
<action name="Register" class="com.action.RegisterAction"
        method="register">
     <result name="success">/success.jsp</result>
     <result name="error">/error.jsp</result>
</action>
<action name="Login" class="com.action.RegisterAction"
        method = "login">
    <result name="success">/welcome.jsp</result>
    <result name="error">/error.jsp</result>
</action>
```

（10）运行应用程序

实验题目 2：Struts 2 的 UI 标签的使用。

下面实例演示了几个 UI 标签的使用，动作类 RegisterAction 的代码如下。

```
package com.action;
import java.util.ArrayList;
import com.model.City;
import com.opensymphony.xwork2.ActionSupport;

public class RegisterAction extends ActionSupport {
   private String username;
   private String password;
```

```
   private String gender;
   private String about;
   private String city;            // 存放在页面中选中的城市
   private String[] language;      // 存放在页面中选中的语言
   private ArrayList<City> cityList;
   private ArrayList<String> langList;
   private Boolean marry;
   // 省略属性的 setter 和 getter 方法
   public String populate() {
      cityList = new ArrayList<City>();
      cityList.add(new City(1, "北京"));
      cityList.add(new City(2, "上海"));
      cityList.add(new City(3, "广州"));

      langList = new ArrayList<String>();
      langList.add("Java");
      langList.add(".Net");
      langList.add("Object C");
      langList.add("C++");
      marry = false;
      return "populate";
   }
  public String execute() {
     return SUCCESS;
  }
}
```

在 struts.xml 文件中添加下面的 action 定义：

```
<action name="*Register" method="{1}" class="com.action.RegisterAction">
    <result name="populate">/register.jsp</result>
    <result name="input">/register.jsp</result>
    <result name="success">/success.jsp</result>
</action>
```

下面是 register.jsp 页面代码。

```
<%@ page contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8"%>
<%@taglib uri="/struts-tags" prefix="s"%>
<html>
<head><title>注册页面</title></head>
<body>
<s:form action="Register">
   <s:textfield name="username" label="用户名" />
   <s:password name="password" label="口令" />
   <s:radio name="gender" label="性别" list="{'男','女'}" />
   <s:select name="city" list="cityList"
             listKey="cityId" listValue="cityName"
             headerKey="0" headerValue="城市" label="请选择城市" />
   <s:textarea name="about" label="简历" />
   <s:checkboxlist name="language" list="langList"
```

```
                    label="精通语言" />
    <s:checkbox name="marry" label="婚否?" />
    <s:submit value="提交"/>
</s:form>
</body>
</html>
```

应该通过 populateRegister.action 动作请求执行 RegisterAction 类的 populate()，这样才执行 register.jsp 页面，显示如图 12-1 所示。

图 12-1　register.jsp 页面运行结果

在页面中输入或选择选项后，单击“提交”按钮请求 Register 动作，首先使用从页面获得的属性值填充属性，然后执行 RegisterAction 类的 execute()，将控制转发到 success.jsp 页面，代码如下。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="/struts-tags"  prefix="s"%>
<html>
<head><title>Details Page</title>
</head>
<body>
   用户名: <s:property value="username" /><br>
   性别: <s:property value="gender" /><br>
   城市: <s:property value="city" /><br>
   简历: <s:property value="about" /><br>
   精通语言: <s:property value="language" /><br>
   婚否: <s:property value="marry" />
</body>
</html>
```

该页面运行结果如图 12-2 所示。

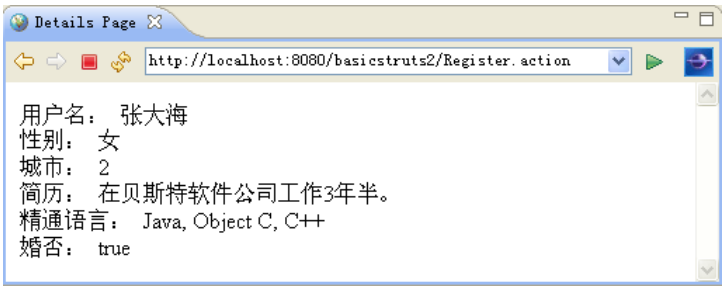

图 12-2　success.jsp 页面运行结果

实验题目 3：用 Tiles 实现页面布局。

（1）安装所需的 Tiles 库。将 struts-2.3.8-all.zip 解压文件中下列文件复制到 WEB-INF\lib 目录中。

- commons-beanutil-1.8.0.jar
- commons-collections-3.2.jar
- commons-digester-2.0.jar
- commons-logging-1.1.1.jar
- commons-logging-api-1.1.jar
- struts2-tiles-plugin-2.3.4.jar
- tiles-api-2.0.6.jar
- tiles-core-2.0.6.jar
- tiles-jsp-2.0.6.jar

（2）在 web.xml 中配置 Tiles 监听器和 tilesDefinitions 参数

```
<listener>
    <listener-class>
        org.apache.struts2.tiles.StrutsTilesListener
    </listener-class>
</listener>
<context-param>
    <param-name>tilesDefinitions</param-name>
    <param-value>/WEB-INF/tiles.xml</param-value>
</context-param>
```

（3）创建模板页面 baseLayout.jsp

```
<%@ page contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title><tiles:insertAttribute name="title" ignore="true" /></title>
<style type="text/css">@import url(css/style.css);</style>
</head>
```

```
<body>
<div id="container">
    <div id="header">
       <tiles:insertAttribute name="header" /></div>
    <div id="mainContent">
       <div id="leftmenu"><tiles:insertAttribute name="leftmenu" /></div>
       <div id="content"><tiles:insertAttribute name="content" /></div>
    </div>
    <div id="footer"><tiles:insertAttribute name="footer" /></div>
</div>
</body>
</html>
```

（3）创建 tiles.xml 定义文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
      "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
      "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
    <definition name="baseLayout" template="/baseLayout.jsp">
         <put-attribute name="title" value="" />
         <put-attribute name="header" value="/header.jsp" />
         <put-attribute name="leftmenu" value="/leftmenu.jsp" />
         <put-attribute name="content" value="" />
         <put-attribute name="footer" value="/footer.jsp" />
    </definition>

    <definition name="/welcome.tiles" extends="baseLayout">
        <put-attribute name="title" value="欢迎页面" />
        <put-attribute name="content" value="/welcome.jsp" />
     </definition>

    <definition name="/error.tiles" extends="baseLayout">
        <put-attribute name="title" value="错误页面" />
        <put-attribute name="content" value="/error.jsp" />
    </definition>
</tiles-definitions>
```

（4）创建 LinkAction 类文件

```
package com.action;

import com.opensymphony.xwork2.ActionSupport;
public class LinkAction extends ActionSupport {
    private static final long serialVersionUID = -2613425890762568273L;
   public String welcome(){
        return "welcome";
    }
    public String friends(){
        return "friends";
    }
```

```
    public String office(){
        return "office";
    }
}
```

（5）修改 struts.xml 文件，为布局创建一个新结果类型。

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="default" extends="struts-default">
        <result-types>
          <result-type name="tiles"
              class="org.apache.struts2.views.tiles.TilesResult" />
        </result-types>
        <action name="*Link" method="{1}"
                class="com.action.LinkAction">
.           <result name="welcome" type="tiles">welcome</result>
            <result name="friends" type="tiles">friends</result>
.           <result name="office" type="tiles">office</result>
        </action>
    </package>
</struts>
```

（6）访问 welcomeLink.action，运行结果如图 12-3 所示。

图 12-3　welcomeLink.action 运行结果

【思考题】

1. 简述 Struts 2 框架的基本构成。
2. 简述 Struts 2 如何与 Hibernate 集成。

12.3　习题解析

1. 简述 Struts 2 框架的组成。

【答】 Struts 2是基于MVC设计模式的Web应用程序开发框架，它主要包括控制器（过滤器、拦截器）、Action对象、视图JSP页面和配置文件struts.xml以及标签库等。

2. 说明在 Struts 2 框架中实现 MVC 的模型、视图和控制器都是使用什么组件实现的。

【答】 模型由JavaBeans实现，它可实现业务逻辑。视图通常由JSP页面实现，也可以由Velocity Template、FreeMarker或其他表示层技术实现。控制器由过滤器、拦截器或Action组件实现。

3. 下面哪个常量不是在 Action 接口中声明的？（ ）

A. SUCCESS　　B. ERROR

C. INPUT　　D. LOGOUT

【答】 D。

4. 下面哪个方法是在 Action 接口中声明的？（ ）

A. String lonin()　　B. String execute()

C. void register()　　D. String validate()

【答】 B。

5. 要在JSP页面中使用Struts 2的标签，应该在页面中使用什么指令？（ ）

A. <% page taglib="/struts-tags"%>

B. <%@ taglib prefix="s" uri="/struts-tags" %>

C. <%@ taglib prefix="c" uri=" http://java.sun.com/jsp/jstl/core " %>

D. <%@ taglib prefix="/struts-tags " uri=" s" %>

【答】 B。

6. 试说明 Struts 2 的 struts.xml 配置文件的作用。

【答】 主要用来建立动作Action类与视图的映射。根元素<struts>的直接子元素包括package、constant、bean和include，它们分别用来定义包、常量、bean和包含其他配置文件。

7. 在 JSP 页面中如何访问值栈中的动作属性？

【答】 假设 message 是一个动作属性，有多种方法使用<s:property>标签访问动作属性，如下所示：

```
<s:property value="message"/>
<s:property value="[0]['message']"/>
<s:property value="getMessage()"/>
```

8. 要访问 Stack Context 的 application 对象中的 userName 属性，下面哪个是正确的？（ ）

A. <s:property value="#application.userName" />

B. <s:property value="application.userName" />

C. <s:property value="${application.userName}" />

D. <s:property value="%{application.userName}" />

【答】 A。

9. 下面哪个标签可以在集合对象上迭代？（ ）

A. <s:bean> B. <s:iterator>

C. <s:generator> D. <s:sort>

【答】 B。

10. 表单 UI 标签默认使用的主题是（ ）。

A. simple B. css_xhtml

C. xhtml D. ajax

【答】 C。

11. 若要开发国际化的 Struts 2 应用，可以使用哪几种属性文件？

【答】 Action级别属性文件、包级别属性文件和全局属性文件。

12. 如果属性文件中包含非西欧字符，应该如何转换？

【答】 可使用JDK的native2ascii工具进行转换。

13. 假设使用 Struts 2 的校验框架为 LoginAction 动作类定义校验规则，校验规则文件名应为（ ）。

【答】 LoginAction-validation.xml。

14. 下面哪种校验不能使用校验框架实现？（ ）

A. 限制一个字段的长度 B. 指定口令包含的字符

C. Email 地址是否合法 D. 日期数据是否合法

【答】 B。

15. 简述用 Tiles 框架设计页面布局需要编写哪些文件？

【答】 在web.xml文件中注册StrutsTilesListener监听器、定义一个tilesDefinition的上下文参数；创建模板页面文件（如baseLayout.jsp）；编写定义文件（tiles.xml）；修改struts.xml文件用<result-types>指定全局结果类型。