第4章

用 Model 2 实现用户登录注册

4.1 项目导引

经过在细化阶段的冲刺评审会议，我们得出结论，运用 Model1 开发不能满足用户的需求。在接下来的细化阶段的第三次迭代中我们采用 Model2 完成用户登录和注册模块的实现。

在 Model 2 中，控制器主要由 Servlet 来实现，所以在这一章中还将学习 Servlet 技术。

4.2 项目分析

在 Model 2 下登录和注册这两个用例的设计类图如图 4-1 所示。

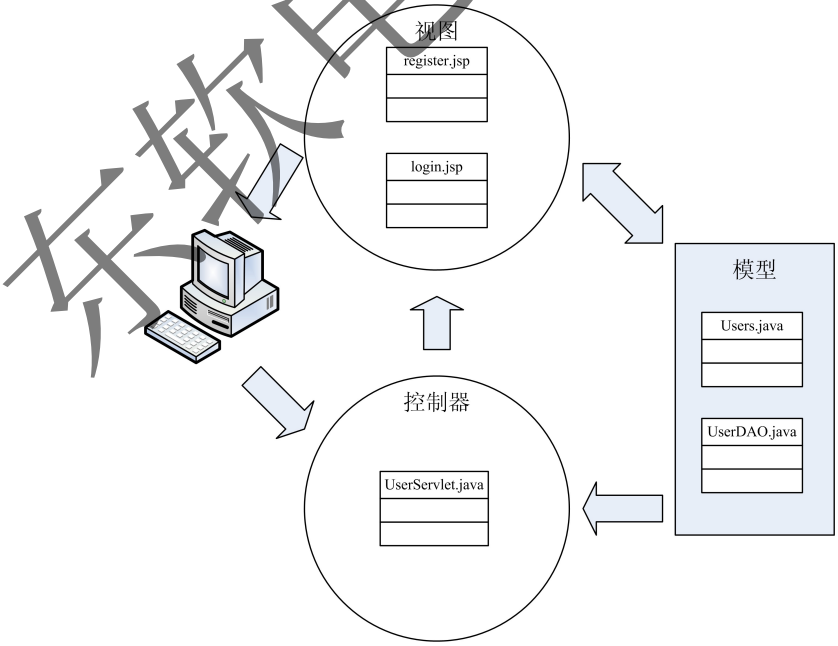


图 4-1　Model2 实现登录注册

4.3 技术准备

4.3.1 Servlet 技术简介

1.Servlet 概述

Servlet(Server Applet),全称 Java Servlet,未有中文译文,是用 Java 编写的服务器端程序。狭义的 Servlet 是指 Java 语言实现的一个接口,此接口定义了 Servlet 容器用以管理 Servlet 以及 Servlet 进行交互的方法。广义的 Servlet 是指任何实现了这个 Servlet 接口的类,一般情况下,人们将 Servlet 理解为后者。处理 HTTP 请求的 Servlet 通常会扩展 javax.servlet.http.HttpServlet 类,这个类实现了 Servlet 接口,并提供了处理 HTTP 请求的额外方法。

伴随着 JavaEE6 一起发布的 Servlet3.0 规范是 Servlet 规范历史上最重要的变革之一,它的许多特性都极大地简化了 Java web 应用的开发,提高了 Java web 应用的开发效率。

2.Servlet 容器

Apache Tomcat(或简称 Tomcat)是由 Apache Software Foundation 开发的开源 Web 服务器。随 Tomcat 4.x 发布的还有 Catalina(Servlet 容器)、Coyote(HTTP connector)和 Jasper(JSP 引擎)等组件。Jasper 解析 JSP 文件并把它们编译成 Catalina(Servlet 容器)处理的 Java 字节码。

Servlet 容器是 Web 服务器中与 Java Servlet 进行交互的组件,它定义了 Servlet 的运行时环境,提供安全、并发、生命周期管理、事务、部署和其他服务。

图 4-2 描述了在 Model 2 的软件体系结构下 Java Web 应用的请求处理过程。客户端向 Web 服务器发送 HTTP 请求,Servlet 容器创建 HTTPServletRequest 对象并将 Http 请求直接封装到 HTTPServletRequest 对象中。该对象被提交给 Servlet 和 JSP,它们与 JavaBeans 进行交互生成动态内容。最后,这些动态内容被写入 Servlet 容器创建的 HTTPServletResponse 对象中,由 Web 服务器转换成 HTTP Response 返回给客户端。

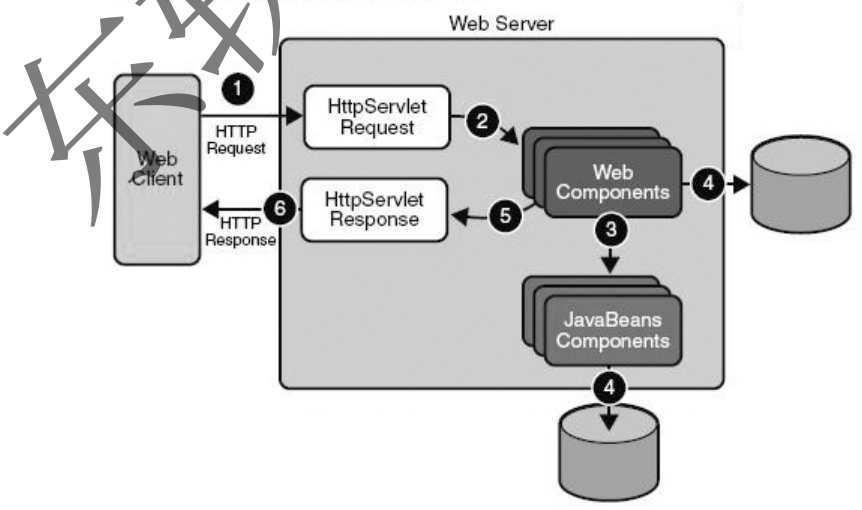
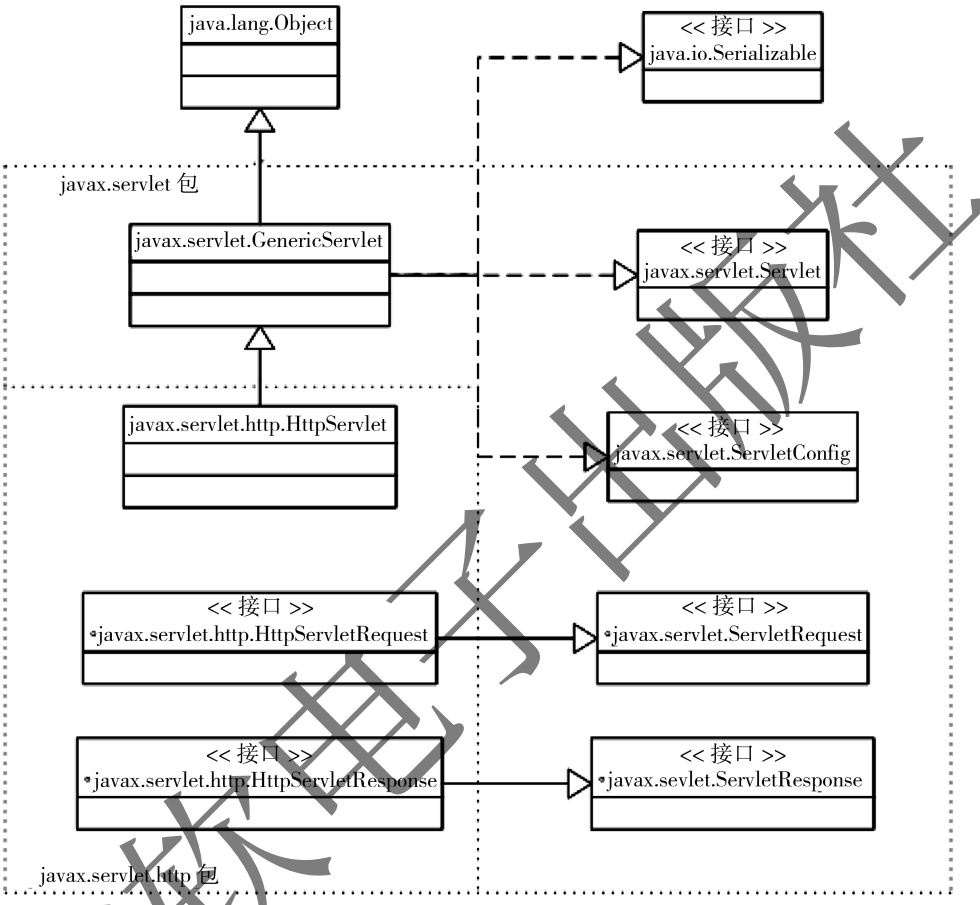


图 4-2 Java Web 应用的请求处理

4.3.2　Servlet 常用类和接口

Java API 提供了 javax. servlet 和 javax. servlet. http 包，为编写 Servlet 提供了类和接口。图 4-3 显示了这些常用类和接口。



4-3　Servlet 常用类和接口

1. Servlet 接口

Servlet 接口定义了所有 Servlet 必须实现的方法，用户编写的 Servlet 必须直接或间接地实现该接口，Servlet 接口的主要方法如表 4-1 所示。

表 4-1　　Servlet 接口常用方法

方法名	功能
void init(ServletConfig config)	Servlet 开始服务时用于初始化 servlet 的状态
void service(ServletRequest req, ServletResponse res)	处理请求
void destroy()	Servlet 退出服务
ServletConfig getServletConfig()	返回一个 ServletConfig 对象，这个配置对象是 init 方法的输入参数
String getServletInfo()	返回有关 Servlet 的信息，如作者，版本、版权等

2. HttpServlet 类

javax.servlet.http 包中定义了采用 HTTP 通信协议的 HttpServlet 抽象类，它的子类必须覆盖以下方法中的一种，如表 4-2 所示。

表 4-2　　HttpServlet 类常用方法

方法名	功能
void init(ServletConfig config)	同上
void doGet(HttpServletRequest req, HttpServletResponse resp)	处理一个 HTTP Get 请求
void doPost(HttpServletRequest req, HttpServletResponse resp)	处理一个 HTTP Post 请求
void doPut(HttpServletRequest req, HttpServletResponse resp)	处理一个 HTTP Put 请求
void doDelete(HttpServletRequest req, HttpServletResponse resp)	处理一个 HTTP Delete 请求
void destroy()	同上
StringgetServletInfo()	同上

3. HttpServletRequest 接口

javax.servlet.http.HttpServletRequest 接口是 javax.servlet.ServletRequest 的子接口，提供了获得客户端请求信息的方法。

Servlet 容器创建 HttpServletRequest 对象，并且把它作为参数传递给 Servlet 的 service 方法(和 doGet、doPost 等方法)。其常见方法如表 4-3 所示。

表 4-3　　HttpServlet 类常用方法

方法名	功能
String getContextPath()	得到请求 URI 中表示请求上下文的部分
Cookie[] getCookies()	得到客户端在此次请求中发送的所有 Cookie 对象的数组
String getMethod()	得到此次请求所使用的 HTTP 方法的名称，如 Get、Post 或 Put
String getQueryString()	得到请求 URL 中在路径后的查询字符串
String getParameter(String name)	得到请求参数的值，如果不存在返回 null
String getServletPath()	得到请求 URL 中调用 Servlet 的部分
HttpSession getSession()	得到和此次请求相关联的当前 Session，如果没有，则创建一个新的 Session
String getServerName()	得到请求被发送到的服务器的主机名

4. HttpServletResponse 接口

javax.servlet.http.HttpServletResponse 接口是 ServletResponse 的子接口，该接口提供了特定于 HTTP 的方法以返回响应到客户端。其常用方法如表 4-4 所示。

表 4-4　　HttpServletResponse 接口常用方法

方法名	功能
void addCookie(Cookie cookie)	增加一个 Cookie 到响应中
void sendRedirect(String url)	用指定的重定向 URL 发送临时的重定向响应到客户端，并且清空(用该方法的数据替代)缓冲区的数据
void setHeader(String name, String value)	使用指定的名称和值设置响应的首部
Void setContentType(String type)	设置发送到客户端的响应的内容类型

4.3.3　Servlet 生命周期

Servlet 运行在 Servlet 容器中，其生命周期由容器来管理。Servlet 生命周期包括四个阶段。

1.加载和实例化

当 Servlet 容器启动时或者在容器检测到需要这个 Servlet 来响应第一个请求时，通过类加载器加载 Servlet 类，成功加载后，容器通过 Java 的反射技术，调用 Servlet 的默认构造函数创建 Servlet 实例。因此，在编写 Servlet 时，通常不会提供有参数的构造函数。

如果需要让 Servlet 容器在启动时即加载 Servlet，可以在 web.xml 文件中配置＜load-on-startup＞元素。

2.初始化

在 Servlet 实例化之后，容器将调用 Servlet 的 init()方法初始化这个对象。初始化的目的是为了让 Servlet 对象在处理客户端请求前完成一些初始化的工作，如建立数据库的连接，获取配置信息等。对于每一个 Servlet 实例，init()方法只被调用一次。在初始化期间，Servlet 实例可以使用容器为它准备的 ServletConfig 对象从 Web 应用程序的配置信息(在 web.xml 中配置)中获取初始化的参数信息。

3.请求处理

Servlet 容器调用 Servlet 的 service()方法对请求进行处理。在 service()方法中，Servlet 实例通过 ServletRequest 对象得到客户端的相关信息和请求信息，在对请求进行处理后，调用 ServletResponse 对象的方法设置响应信息。在 service()方法执行期间，如果发生错误，Servlet 实例可以抛出 ServletException 异常或者 UnavailableException 异常。

4.服务终止

在 Servlet 退出服务之前调用 destroy()方法，该方法可以保存 Servlet 生命周期期间使用的数据到数据库，并且释放它所使用的资源。

4.3.4　Servlet 应用

接下来，通过两个案例分别介绍使用 web.xml 文件配置和 Servlet3.0 支持的 Annotaion 配置两种方法。

案例：用传统方法编写一个 Servlet，实现在客户端显示一个“Hello World!”。

第一步：新建一个 web 项目“servlettest1”，J2EE Specification Level 选择 JavaEE 5.0。

第二步：新建一个 HelloServlet 类，扩展自 HttpServlet，代码如下。

```
package demo;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
    // TODO Auto-generated method stub

    //设置客户端请求的编码方式为 UTF-8
    req.setCharacterEncoding("UTF-8");
    //设置响应的内容类型
    res.setContentType("text/html;charSet=UTF-8");
    PrintWriter out=res.getWriter();
    out.println("<HTML><BODY>");
    out.println("Hello World!");
    out.println("</BODY></HTML>");
}
}
```

第三步：在 web.xml 中配置 Servlet。

```
<servlet>
<description>This is the description of my J2EE component</description>
<display-name>This is the display name of my J2EE component</display-name>
<servlet-name>HelloServlet</servlet-name>
<servlet-class>demo.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloServlet</servlet-name>
<url-pattern>/servlet/HelloServlet</url-pattern>
</servlet-mapping>
```

说明：

(1)<servlet-name>元素中指定 Servlet 实例名。

(2)<servlet-class>元素指定 Servlet 类。

(3)<url-parttern>元素指定在 URL 中访问 Servlet 的路径。

第四步：部署项目，启动 Tomcat7。

第五步：在浏览器中输入：http://localhost:8080/servlettest/servlet/HelloServlet，查看运行效果。

开发团队按上述步骤完成，记录调试结果及遇到的问题和解决方案。

运行结果：

遇到问题及解决方案：

案例：Servlet3.0 支持的 Annotaion 配置 Servlet，实现在客户端显示一个“Hello World!”。

第一步：新建一个 web 项目“servlettest1”，在弹出窗口中“J2EE Specification Level”选择 JavaEE6.0。

第二步：新建一个 HelloServlet 类，扩展自 HttpServlet，代码如下。

```
package demo;
import java.io.IOException;
import java.io.PrintWriter;
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="helloServlet",urlPatterns={"/helloServlet"})
public class HelloServlet extends HttpServlet {
    protected void service(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        //设置客户端请求的编码方式为 UTF-8
        req.setCharacterEncoding("UTF-8");
        //设置响应的内容类型
        res.setContentType("text/html;charSet=UTF-8");
        PrintWriter out=res.getWriter();
        out.println("<HTML><BODY>");
        out.println("Hello World!");
        out.println("</BODY></HTML>");
    }
}
```

上面开发 Servlet 类时使用了@WebServlet 修饰该 Servlet 类，当使用这种方法时，不需要在 web.xml 中再配置 Servlet。使用@WebServlet 时可以指定以下属性，如表 4-5 所示。

表 4-5　　使用@WebServlet 时可以指定的属性

属性名	是否必需	说明
asynSupported	否	指定该 Servlet 是否支持异步模式
displayName	否	指定该 Servlet 显示名称
InitParams	否	用于为该 Servlet 配置参数
loadOnStartup	否	用于将该 Servlet 配置成 load-on-startup 的 Servlet
name	否	指定该 Servlet 的名称
urlPatterns/value	否	指定该 Servlet 处理的 URL

第三步：部署项目，启动 Tomcat7。

第四步：在浏览器中输入：http://localhost:8080/servlettest/helloServlet，查看运行效果。

开发团队按上述步骤完成，记录调试结果及遇到的问题和解决方案。

运行结果：

遇到问题及解决方案：

4.3.5　过滤器

1.过滤器的概念

过滤器对请求和响应的首部和体进行过滤。过滤器本身并不创建响应，它截取发送至 Servlet、JSP 页面和静态页面的请求，或截取返回给客户端的响应。过滤器的应用包括验证、日志、图像转换、数据压缩、加密、分解字符串、XML 变换等。

2.过滤器(Filter)的工作原理

过滤器的工作原理如图 4-4 所示。

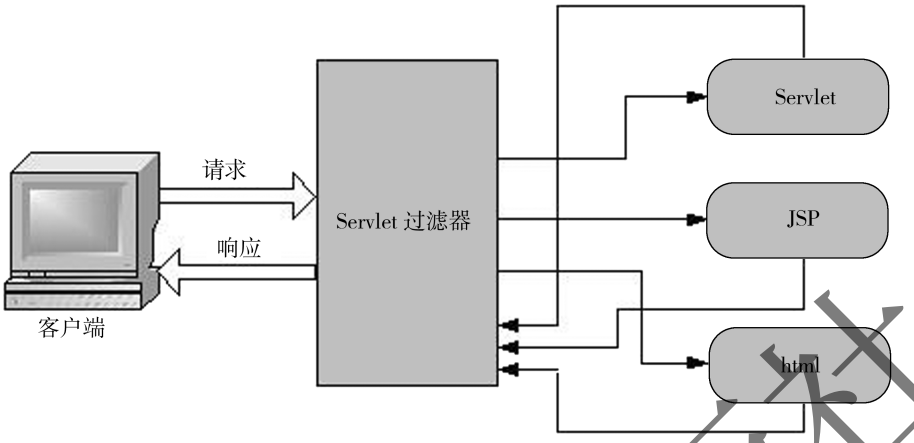


图 4-4　过滤器工作原理

FilterAPI 由 javax. servlet 包的 Filter、FilterChain 和 FilterConfig 接口组成，用户编写的过滤器必须实现 Filter 接口。该接口的最重要的方法是 doFilter(ServletRequest reg, ServletResponse res,FilterChain chain)，该方法的第一个参数为 ServletRequest 对象，该对象提供了获得客户端请求信息的方法，第二个参数为 ServletResponse 对象，该对象提供了方法以返回响应到客户端，通常在简单的过滤器中忽略此参数，最后一个参数为 FilterChain，该参数是由过滤器组成的链中。该方法执行以下操作：

(1)检查请求的首部。

(2)修改请求的首部和体。

(3)修改响应的首部和体。

(4)将请求转发给过滤器链中的下一个过滤器或 Web 资源(Servlet、JSP 页面和静态页面)。另外，还可以阻止该请求由过滤器生成响应。

(5)在将请求转发给过滤器链中的下一个过滤器后检查响应的首部。

除了 doFilter 方法，过滤器还有 init、destroy 等方法。这两个方法等同于 Servlet 的 init、destory 方法。

3. 创建过滤器的方法

Filter 可以理解为是 Servlet 的“增强版”，因此配置过滤器与配置 Servlet 非常相似，区别在于，Servlet 通常只配置一个 URL，而过滤器可以同时拦截多个请求的 URL。因此，在配置过滤器的 URL 时通常会使用模式字符串，使得过滤器可以拦截多个请求。下面介绍使用 web. xml 文件进行配置。

第一步：实现 javax. servlet. Filter 接口以及这接口中的三个抽象方法。在以下的 init 方法中初始化过滤器，并获取 web. xml 文件中配置的参数。

```
public void init(FilterConfig filterConfig) throws ServletException {
    //获取 Filter 初始化参数
    String username=filterConfig.getInitParameter("username");
}
```

第二步：在 web. xml 中配置过滤器。

配置 Servlet 过滤器包括下面两个步骤：

(1)指定过滤器的名称和相应的实现类,并且设置传递至 init 方法的初始化参数。配置代码如下。

```
<filter>
<filter-name>FilterName</filter-name>
<filter-class>package.className </filter-class>
<init-param>
<param-name>ParamName1</param-name>
<param-value>ParamValue1</param-value>
</init-param>
</filter>
```

(2)将过滤器映射至 URL 或 Servlet,这是通过<filter-mapping>元素来实现的。配置代码如下。

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<url-pattern>/path</url-pattern>
</filter-mapping>
```

<filter-name>元素和<filter>元素的子元素<filter-name>一致。<url-pattern>元素指定了过滤器的映射路径。

另外,Servlet 过滤器还可映射至 Servlet:

```
<filter-mapping>
<filter-name>FilterName</filter-name>
<servlet-name>ServletName</servlet-name>
</filter-mapping>
```

在 Web 应用程序中,解决页面传递时中文乱码,可以通过 request.setCharacterEncoding("中文编码"),也可以通过 String 类的编码转换,今天介绍的过滤器可以做到一劳永逸。

本案例中的过滤器通过 Annotation 配置来实现,使用@WebFilterAnnotation 修饰一个过滤器类。它常用的属性如表 4-6 所示。

表 4-6　　常用属性

属性名	是否必需	说明
asynSupported	否	指定该 Filter 是否支持异步模式
displayName	否	指定该 Filter 显示名
initParams	否	用于为该 Filter 配置参数
dispatcherTypes	否	指定该 Filter 仅对哪种 dispatcher 模式的请求进行过滤
filterName	否	指定该 Filter 的名称
urlPatterns/value	否	指定该 Filter 拦截处理的 URL
servletNames	否	指定多个 Servlet 名称,用于指定该 Filter 仅对这几个 Servlet 执行过滤

在 Filter 类中加@WebFilter 进行 Filter 配置的方法如下:

```
package demo;
import java.io.IOException;
import javax.servlet.Filter;
```

```
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
@WebFilter(filterName="charFilter",
urlPatterns={"/*"},
initParams={
@WebInitParam(name="encoding",value="UTF-8")})
public class CharacterFilter implements Filter {
    String encoding="";
    public void destroy() {
        // TODO Auto-generated method stub
    }
    public void doFilter(ServletRequest req, ServletResponse res,
            FilterChain chain) throws IOException, ServletException {
        // TODO Auto-generated method stub
        req.setCharacterEncoding(encoding);
        chain.doFilter(req, res);
    }
    public void init(FilterConfig config) throws ServletException {
        // 读取 web.xml 中给的初始化参数
        encoding=config.getInitParameter("encoding");
    }
}
```

开发团队利用上述方法解决第二章中的“案例：使用＜jsp:include＞动作元素包含动态和静态文件”输入中文名字出现乱码的问题，记录调试结果及遇到的问题和解决方案

运行结果：

遇到问题及解决方案：

4.4 项目实施

在细化阶段的第三次迭代中采用 Model 2 来分别实现用户登录和注册，通过本任务的学习，你应当能够：

(1)会编写控制器。

(2)会编写过滤器。

(3)会比较 Model1 和 Model 2 软件体系结构的优劣，选择一种可行的解决方案。

案例：用 Model2 实现用户登录。

在本案例中，我们采用 Model 2 来实现用户登录，涉及以下页面和类：

视图：

login.jsp 为页面来实现接收用户输入信息。

welcome.jsp 为用户登录成功跳转的页面。

控制器：UserServlet 实现控制器。

模型：

Users 类对应用户表的实体类。

UserDAO 类封装与用户相关的 DAO 操作。

DataBase 类为通用的数据库访问类。

第一步：从 SVN Repository 中导出 bookshop 项目。

第二步：设计项目中会涉及到源文件的包结构如下。

(1)UserServlet 所在的包：net.svtcc.bookshop.dispatcher。

(2)bean 所在的包：net.svtcc.bookshop.model。

(3)Users 所在的包：net.svtcc.bookshop.model.entity。

(4)UserDAO 所在的包：net.svtcc.bookshop.model.dao。

(5)DataBase 所在的包：net.svtcc.bookshop.model.db。

(6)视图所涉及的 JSP 页面。

第三步：实现 DataBase 类。

bookshop 的数据保存在数据库中，通过数据库访问类 DataBase 进行访问。基于这种策略，可以得到更易于维护的应用。因为如果数据库模式有所调整，只需在一处完成相应修改。

(1)新建属性文件 db.properties，存放数据库连接相关信息，如图 4-5 所示。

db.properties

name	value
driverName	com.microsoft.sqlserver.jdbc.SQLServerDriver
url	jdbc:sqlserver://localhost:1433;databaseName=MyBookshop
userName	java11-1
password	123456

图 4-5 属性文件 db.properties

(2)创建 DataBase 类,用于建立连接和释放连接。

```
package net.svtcc.bookshop.model.db;
import java.io.IOException;
import java.sql.*;
import java.util.*;
public class DataBase {
    Properties p;
    String driverName;
    String url;
    String userName;
    String password;
/**
     *构造函数:完成读取属性文件和加载数据库驱动
     */
    public DataBase(){
        p=new Properties();
        try {
            p.load(DataBase.class.getResourceAsStream("db.properties"));
            driverName=p.getProperty("driverName");
            url=p.getProperty("url");
            userName=p.getProperty("userName");
            password=p.getProperty("password");
            Class.forName(driverName);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
/**
     *
     * @return 得到数据库的连接
     */
    public Connection getConnection(){
        try {
            return DriverManager.getConnection(url,userName,password);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
            return null;
        }
        /* *
            * 释放数据库连接
            * @param con
            */
        public void closeConnection(Connection con){
            try {
                con.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
}
```

第四步:设计实体类 Users,存放到 net.svtcc.bookshop.model.entity 包中,该类是一个简单的 bean,属性与数据库中的 Users 表的字段相对应,为每个属性增加一个 getter 和 setter 方法。

```
package net.svtcc.bookshop.model.entity;
import java.io.Serializable;
public class Users implements Serializable {
    String loginId;
    String loginPwd;
    String name;
    String address;
    String phone;
    String mail;
    int userRoleId;
    int userStateId;
    public String getLoginId() {
        return loginId;
    }
    public void setLoginId(String loginId) {
        this.loginId=loginId;
    }
    public String getLoginPwd() {
        return loginPwd;
    }
    public void setLoginPwd(String loginPwd) {
        this.loginPwd=loginPwd;
    }
    public String getName() {
```

```
        return name;
    }
    public void setName(String name) {
        this.name=name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address=address;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone=phone;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail=mail;
    }
    public int getUserRoleId() {
        return userRoleId;
    }
    public void setUserRoleId(int userRoleId) {
        this.userRoleId=userRoleId;
    }
    public int getUserStateId() {
        return userStateId;
    }
    public void setUserStateId(int userStateId) {
        this.userStateId=userStateId;
    }
}
```

第五步：创建 UserDAO 类，使用一个 bean 来封装用户的数据库访问代码。

```
package net.svtcc.bookshop.model.dao;
import net.svtcc.bookshop.model.db.DataBase;
import net.svtcc.bookshop.model.entity.Users;
import java.sql.*;
public class UserDAO {
```

```
DataBase db;
public UserDAO(){
    db=new DataBase();
}
/**
 *
 * @param u 用户对象
 * @return 该用户是否为注册用户,如果是返回真,否则返回假
 */
public boolean doLogin(Users u){
    Connection con=db.getConnection();
    if(con!=null){
        String sql="select *" +
            "   from users" +
            "   where loginId=? and loginPwd=?";
        PreparedStatement stmt;
        try {
            stmt=con.prepareStatement(sql);
            stmt.setString(1, u.getLoginId());
            stmt.setString(2, u.getLoginPwd());
            ResultSet rs=stmt.executeQuery();
            if(rs.next())
                return true;
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    return false;
}
}
```

第六步,创建 UserServlet,该类的主要功能是接收 login.jsp 输入的用户名和密码,调用 UserDAO 的 doLogin 方法对登录的用户进行验证,然后根据返回的结果跳转到不同页面。

UserServlet 类的核心代码如下.

```
package net.svtcc.bookshop.dispatcher;
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 javax.servlet.http.HttpSession;
```

```
import net.svtcc.bookshop.model.dao.UserDAO;
import net.svtcc.bookshop.model.entity.Users;
public class UserServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        doPost(req,resp);
    }
    @Override
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        //获得视图的请求参数
        String name=req.getParameter("username");
        String password=req.getParameter("password");
        //调用模型层的业务方法
        Users u=new Users();
        u.setLoginId(name);
        u.setLoginPwd(password);
        UserDAO dao=new UserDAO();
        //获得 session 对象,用于保存用户信息
        HttpSession session=req.getSession();
        if(dao.doLogin(u)){
            session.setAttribute("name", name);
            resp.sendRedirect("../welcome.jsp");
        }
        else
            resp.sendRedirect("../login.jsp");
    }
}
```

在 web.xml 文件中配置 Servlet,在 web.xml 中加入以下代码。

```
<servlet>
<servlet-name>userServlet</servlet-name>
<servlet-class>net.svtcc.bookshop.dispatcher.UserServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>userServlet</servlet-name>
<url-pattern>/servlet/userServlet</url-pattern>
</servlet-mapping>
```

第七步:设计视图层所要用到的 JSP 页面,分别是 login.jsp,welcome.jsp。其中,login.jsp 所要用到的核心代码如下。

```
<form action="servlet/userServlet" method="post">
```

```
用户名:<input type="text" name="username"/><br/>
密码:<input type="password" name="password"/><br/>
<input type="submit" value="登录"/>
</form>
```

第八步:部署项目,测试项目运行结果。

运行结果:

遇到问题及解决方案:

案例:用 Model2 架构实现用户注册。

在本案例中,我们采用 Model 2 来实现用户注册,会涉及到以下页面和类:

视图:

regist.jsp 为页面来实现接收用户注册时输入信息(前面章节已完成)。

welcome.jsp 为用户注册成功跳转的页面(前面章节已完成)。

控制器:

UserServlet 实现控制器功能。

模型层:

DataBase 类和实体类 Users 在案例 1 中已写好。

UserDAO 类中加入用户注册的方法。

第一步:在 UserDAO 中加入 doRegister 方法,将注册的新用户信息保存到数据库中。

```
public boolean doRegister(Users u) {
    Connection con;
    try {
        con=ds.getConnection();
        if (con!=null) {
            String sql="insert into Users" +
            "  (LoginId,LoginPwd,Name,Address,phone,mail,userRoleId,userStateId) " +
            "   values(?,?,?,?,?,?,1,1)";
            PreparedStatement stmt=con.prepareStatement(sql);
            stmt.setString(1, u.getLoginId());
```

```
            stmt.setString(2, u.getLoginPwd());
            stmt.setString(3, u.getName());
            stmt.setString(4, u.getAddress());
            stmt.setString(5, u.getPhone());
            stmt.setString(6, u.getMail());
            int rowcount=stmt.executeUpdate();
            if(rowcount>0)
                return true;
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }
```

第二步:修改 UserServlet 类,加入注册控制。

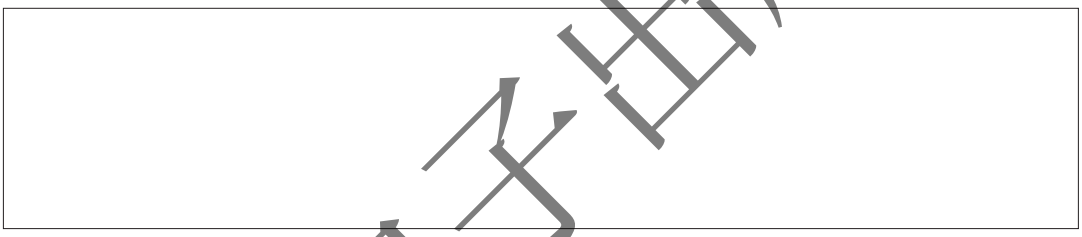
```
public void  doRegist(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException{
    //获取注册页面的请求参数
    String id=req.getParameter("loginId");
    String pwd=req.getParameter("loginPwd");
    String name=req.getParameter("name");
    String address=req.getParameter("address");
    String phone=req.getParameter("phone");
    String mail=req.getParameter("mail");
    Users u=new Users();
    u.setLoginId(id);
    u.setLoginPwd(pwd);
    u.setName(name);
    u.setAddress(address);
    u.setPhone(phone);
    u.setMail(mail);
    //调用模型层的方法
    UserDAO dao=new UserDAO();
    //根据模型层中返回的值决定页面转向
    if(dao.doRegister(u))
        resp.sendRedirect("../welcome.jsp");
    else
        resp.sendRedirect("../register.jsp");
}
@Override
public void doPost(HttpServletRequest req, HttpServletResponse resp)
```

```
        throws ServletException, IOException {
    //获得客户端请求的方法
    String str=req.getParameter("method");
    if(str.equals("regist")){
        doRegist(req, resp);
    }
}
```

第三步:部署项目,测试运行结果。

运行结果:

遇到问题及解决方案:



第四步:把这一阶段的工作成果,更新到 SVN Repository。

独立实践:请开发团队成员在上述功能的基础上,增加用户注册时输入的用户名、地址、密码、电话和邮箱等的客户端验证。

4.5 技术拓展

4.5.1 监听器

1.事件监听机制

事件监听器机制涉及到三个组件:事件源、事件监听器、事件对象。当事件源发生操作时,事件源会调用事件监听器的一个方法响应操作,并且在调用方法时还会把事件对象传递给事件处理器。事件监听器由程序员编写,程序员通过事件对象可以知道哪个事件源上发生了操作,从而可以对操作进行处理。

2.Servlet 监听器工作原理

Servlet 监听器是 Web 应用程序事件模型的一部分,当 Web 应用中的某些状态发生改变时,Servlet 容器就会产生相应的事件,比如创建 ServletContext 对象时触发 ServletContextEvent 事件,创建 HttpSession 对象时触发 HttpSessionEvent 事件,Servlet 监听

器可接收这些事件,并可以在事件发生前、发生后可以做一些必要的处理。

3.Servlet 监听器类型

Servlet 监听器类型如表 4-7 所示。

表 4-7　　Servlet 监听器类型

监听器接口	事件类
ServletContextListener	ServletContextEvent
ServletContextAttributeListener	ServletContextAttributeEvent
HttpSessionListener	HttpSessionEvent
HttpSessionActivationListener	HttpSessionEvent
HttpSessionAttributeListener	HttpSessionBindingEvent
HttpSessionBindingListener	HttpSessionBindingEvent
ServletRequestListener	ServletRequestEvent
ServletRequestAttributeListener	ServletRequestAttributeEvent

案例:用监听器统计在线人数。

第一步:编写监听器类,监听器类要实现某一监听器接口。

```
package demo;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
//第一步:创建监听器类,实现某一个监听器接口
public class OnlineListener implements HttpSessionListener {
    private int counter;
    public OnlineListener(){
    counter=0;
    }
        public void sessionCreated(HttpSessionEvent evt) {
            // TODO Auto-generated method stub
            counter++;
            evt.getSession().getServletContext().
            setAttribute("online", new Integer(counter));

        }

        public void sessionDestroyed(HttpSessionEvent evt) {
            // TODO Auto-generated method stub
            counter--;
            evt.getSession().getServletContext().
            setAttribute("online", new Integer(counter));

        }
    }
```

第二步：在 web.xml 中配置监听器。

```
<listener>
<listener-class>demo.OnlineListener</listener-class>
</listener>
```

第三步：编写 JSP 页面，在 JSP 页面中，加入以下代码，统计人数。

```
在线人数：<%=application.getAttribute("online") %><br>
```

第四步：部署项目，测试效果。

运行结果：

遇到问题及解决方案：

4.5.2 用监听器实现数据库连接池管理

ServletContextListener 接口用于监听 ServletContext 对象的创建和销毁事件。当 ServletContext 对象被创建时会激发 contextInitialized 方法。当 ServletContext 对象被销毁时，会激发 contextDestroyed 方法。ServletContextListener 监听器常常用于对 Web 应用的初始化和终止操作，例如当 Web 应用启动时加载一个数据库连接池，当 Web 应用关闭时释放数据库连接池。

接下来，我们可以在 BookShop 网站中运用 ServletContextListener 监听器来完成数据库连接池的管理。具体步骤如下：

第一步，如前一节案例所述：在 Tomcat 服务器中配置数据源，并将 JDBC 驱动程序放到 Tomcat 6.0 安装文件夹的 lib 文件夹下。

第二步：编写一个类用于在使用 DataSource 时，获得 DataSource，具体代码如下：

```
package net.svtcc.bookshop.listener;
import javax.sql.DataSource;
public class DataSourceProvider {
    private DataSource ds;
    private static DataSourceProvider instance;
    private DataSourceProvider(){
    }
    /**
```

```
        * 用单例模式来获得 DataSourceProvider 的实例
        * @return
        */
    public static  DataSourceProvider getInstance(){
        if(instance==null){
            instance=new DataSourceProvider();
        }
        return instance;
    }
    /**
        * 初始化 DataSource 实例
        * @param ds
        */
    public void initDataSource(DataSource ds){
        this.ds=ds;
    }
    /**
        * 得到一个 DataSource
        * @return
        */
    public DataSource getDataSource(){
        return ds;
    }
}
```

第三步：编写一个监听器类，用于管理数据库连接池，具体代码如下。

```
package net.svtcc.bookshop.listener;
import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.sql.DataSource;
public class ContextListener implements ServletContextListener {
    /**
        * 当 web 应用程序销毁时，关闭连接。
        */
        @Override
        public void contextDestroyed(ServletContextEvent evt) {
            // TODO Auto-generated method stub
            DataSource ds=DataSourceProvider.getInstance().getDataSource();
            try {
```

```
                ds.getConnection().close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        /**
         * 当加载上下文时,获得个数据源的连接
         */
        @Override
        public void contextInitialized(ServletContextEvent evt) {
            // TODO Auto-generated method stub
            try {
                Context ctx=new InitialContext();
                DataSource ds=(DataSource)ctx.lookup("java:comp/env/jdbc/dbpooling");
                DataSourceProvider.getInstance().initDataSource(ds);
            } catch (NamingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println("web 应用程序启动");
        }
    }
```

第四步:在 web.xml 中配置监听器,代码如下。

```
<listener>
<listener-class>net.svtcc.bookshop.listener.ContextListener</listener-class>
</listener>
```

第五步:修改 BookDAO 类和 UserDAO 类中获得数据库连接的方式,核心代码如下。

```
...
public class BookDAO {
    DataSource ds;
    public BookDAO(){
//    try {
//        Context ctx=new InitialContext();
//        ds=(DataSource) ctx.lookup("java:comp/env/jdbc/dbpooling");
//    } catch (NamingException e) {
//        //TODO Auto-generated catch block
//        e.printStackTrace();
//    }
      ds=DataSourceProvider.getInstance().getDataSource();
    }
}
```

第六步:部署应用程序,测试运行结果。

运行结果：

遇到问题及解决方案：

4.6　本章小结

本章主要介绍了 Servlet 的生命周期，Servlet 中常见的类的接口，以及如何创建部署一个 Servlet。并运用 Model2 实现用户登录和注册模块。还介绍了如何运用过滤器解决中文乱码的问题以及运用监听器管理数据库连接池对象。

4.7　强化练习

一、选择题

1. 下列哪一个 XML 标签表示某一个 servlet 的类_____。

A. <servlet-class>com. abciinc. OrderServlet. class</servlet-class>

B. <class>com. abcinc. OrderServlet</class>

C. <servlet>com. abcinc. OrderServlet</servlet>

D. <servletclass> com. abcinc. OrderServlet</servletclass>

2. 下列哪一个方法用于设置 HttpServletResponse 的内容类型_____。

A. setParameter()　　B. setContentType()

C. setAttribute()　　D. doPost()

E. None of the above

3. 在 web. xml 中下面哪一个标签表示 servlet 的 URL 的映射_____。

A. mapping　　B. servlet-url

C. url_mapping　　D. url_pattern

E. servlet-mapping

4. 下列哪一个方法用于从 session 中得到对象_____。

A. Session 接口的 getAttribute()方法

B. HttpSession 接口的 getValue 方法

C. Session 接口的 getValue()方法

D. Session 接口的 get ()方法

E. HttpSession 接口的 getAttribute()方法

5. 下列那个方法在 ServletContext 被初始化时调用_____。

A. ServletContextListener 接口的 contextInitialized()方法

B. ServletContextListener 接口的 contextCreated()方法

C. ServletContextListener 接口的 contextStateChanged()方法

D. ServletContextListener 接口的 Init()方法

E. ServletContextListener 接口的 initialized()方法

6. 下面是 web. xml 中的片段：

```
<context>
<param-name>user</param-name>
<param-value>test</param-name>
</context>
```

在 servlet 中要得到上面的参数，下面哪个表达式是正确的_____。

A. getServletConfig(). getAttribute("user")

B. getServletContext(). getAttribute("user")

C. getServletConfig(). getInitParameter("user")

D. getServletContext(). getInitParameter("user")

7. 下面那个表达式表示会话永不过期_____。

A. setTimeout(0)

B. setTimeout(－1)

C. setMaxInactiveInterval(0)

D. setMaxInactiveInterval(－1)

E. setTimeout(Integer. MAX_VALUE)

F. setMaxInactiveInterva (Integer. MAX_VALUE)

二、简答题

1. 简述在 Tomcat7 中 Servlet 的配置方法。

2. 使用监听器实现 session 超时用户自动注销功能。

3. 简述过滤器的作用。