

☑ 模型层 (Model)

模型层是应用程序的核心部分，主要由 JavaBean 组件来充当，可以是一个实体对象或一种业务逻辑。之所以称之为模型，是因为它在应用程序中有更好的重用性、扩展性。

☑ 视图层 (View)

视图层提供应用程序与用户之间的交互界面。在 MVC 模式中，这一层并不包含任何的逻辑，仅提供一种与用户相交互的视图，在 Web 应用中由 JSP、HTML 界面充当。

☑ 控制层 (Controller)

控制层用于对程序中的请求进行控制，起到一种宏观调控的作用，它可以通知容器选择什么样的视图、什么样的模型组件，在 Web 应用中由 Servlet 充当。

5.3.2 JSP+Servlet+JavaBean

Model2 模式 (JSP+Servlet+JavaBean) 在 Model1 模式的基础上引入了 Servlet 技术。此种开发模式遵循 MVC 的设计理念，其中 JSP 作为视图层为用户提供与程序交互的界面，JavaBean 作为模型层封装实体对象及业务逻辑，Servlet 作为控制层接收各种业务请求，并调用 JavaBean 模型组件对业务逻辑进行处理，在视图与业务逻辑之间建立起一座桥梁。

例 5.07 Model2 模式录入商品信息。(实例位置：光盘\TM\Instances\5.07)

本实例将在例 5.06 的基础上，以 Model2 模式 (JSP+Servlet+JavaBean) 向数据库中添加商品信息。实现步骤如下：

(1) 创建模型层用到的 JavaBean 组件，本实例中放置在 com.lyq.model 包中，分别为 Goods 类与 GoodsDao 类，其中 Goods 类用于封装商品信息，GoodsDao 类封装商品对象的数据库操作，其代码参见例 5.06。

(2) 创建控制层对象 GoodsServlet 类，它是一个 Servlet，此类通过 doPost() 方法对添加商品信息请求进行处理。其关键代码如下：

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //设置 response 编码
    response.setContentType("text/html");
    response.setCharacterEncoding("GBK");
    //设置 request 编码
    request.setCharacterEncoding("GBK");
    //获取输出流
    PrintWriter out = response.getWriter();
    //获取商品信息
    String name = request.getParameter("name");
    String price = request.getParameter("price");
    String description = request.getParameter("description");
    Goods goods = new Goods();    //实例化商品对象
    //对商品对象属性赋值
    goods.setName(name);
    goods.setPrice(Double.valueOf(price));
    goods.setDescription(description);
    //实例化 GoodsDao
```



```

GoodsDao goodsDao = new GoodsDao();
goodsDao.saveGoods(goods);    //保存商品信息
out.print("保存商品信息成功！");
out.flush();
out.close();
}

```

此 Servlet 首先对获取的商品信息进行封装，然后调用相应的 JavaBean 方法保存商品数据。在这一过程中，并没有与任何的 JSP 网页相混合，也没有在网页中进行处理，完全由 Servlet 对业务请求进行控制，当 JavaBean 对象不符合要求时，只需改变相应的 JavaBean 代码或创建一个新的 JavaBean 对象即可，大大提高了程序的可扩展性及可维护性。

(3) 创建视图层页面 index.jsp，此页面放置商品信息的表单，此表单的提交地址为 GoodsServlet。其关键代码如下：

```

<form action="GoodsServlet" method="post" onsubmit="return save(this);">
  <table border="1" align="center" width="300">
    <tr>
      <td align="center" colspan="2">
        <br><h1>录入商品信息</h1>
      </td>
    </tr>
    <tr>
      <td align="right">商品名称:</td>
      <td><input type="text" name="name"></td>
    </tr>
    <tr>
      <td align="right">价 格:</td>
      <td><input type="text" name="price"></td>
    </tr>
    <tr>
      <td align="right">商品描述:</td>
      <td><textarea name="description" cols="30" rows="3"></textarea></td>
    </tr>
    <tr>
      <td align="center" colspan="2">
        <input type="submit" value="提 交">
        <input type="reset" value="重 置">
      </td>
    </tr>
  </table>
</form>

```

实例运行结果如图 5.8 所示，正确填写商品信息后，商品信息将被写入到数据库中。

5.4 两种模式的比较

Java 代码与 HTML 代码写在一起的方式使得 JSP 页面十分混乱，对于一些复杂业务的实现，将会

导致大量的问题。如一个上千行代码的 JSP 文件，程序的可读性会非常差，出现一个小小的错误，都会给程序的调试带来一定的难度，不利于代码的编写与维护。

Model1 开发模式通过 JavaBean 改变了 Java 代码与 HTML 代码混合交织的情况，但它对 JavaBean 的操作仍然在 JSP 页面中进行，甚至部分 JSP 页面只用于与 JavaBean 交互处理业务逻辑，并不包含 HTML 网页代码，JSP 又充当了控制业务逻辑的角色，使显示层与业务层混合在一起，因此这种开发模式仍然不是一种理想的状态。

Model2 开发模式的出现是程序设计方面的一次巨大进步，它以 MVC 的设计理念，将模型层（Model）、视图层（View）、控制层（Controller）相区分，使各部分独挡一面、各负其责，充分体现了程序中的层次概念，改变了 JSP 网页代码与 Java 代码深深耦合的状态，为程序提供了更好的重用性及扩展性。

Model1 开发模式虽然适用于小型项目的开发，但囿于其自身缺陷，目前已逐渐被遗弃。在开发程序时，应该积极采用 Model2 模式进行开发。在本书的后续章节中，将为读者介绍 Struts 框架，此框架是 Model2 模式一个非常经典的实现。

5.5 登录模块的实现

 视频讲解：光盘\TM\Video\5\登录模块的实现.exe

用户登录模块在网站中的应用是十分广泛的，如一个论坛型网站，需要用户登录来识别用户的权限；一个博客型网站，需要用户登录来管理博客文章。通过用户登录能够快速识别用户的身份信息，维护网站的安全。

5.5.1 模块介绍

下面以用户登录模块为例，以 Model2（JSP+Servlet+JavaBean）模式进行开发，其系统流程如图 5.12 所示。

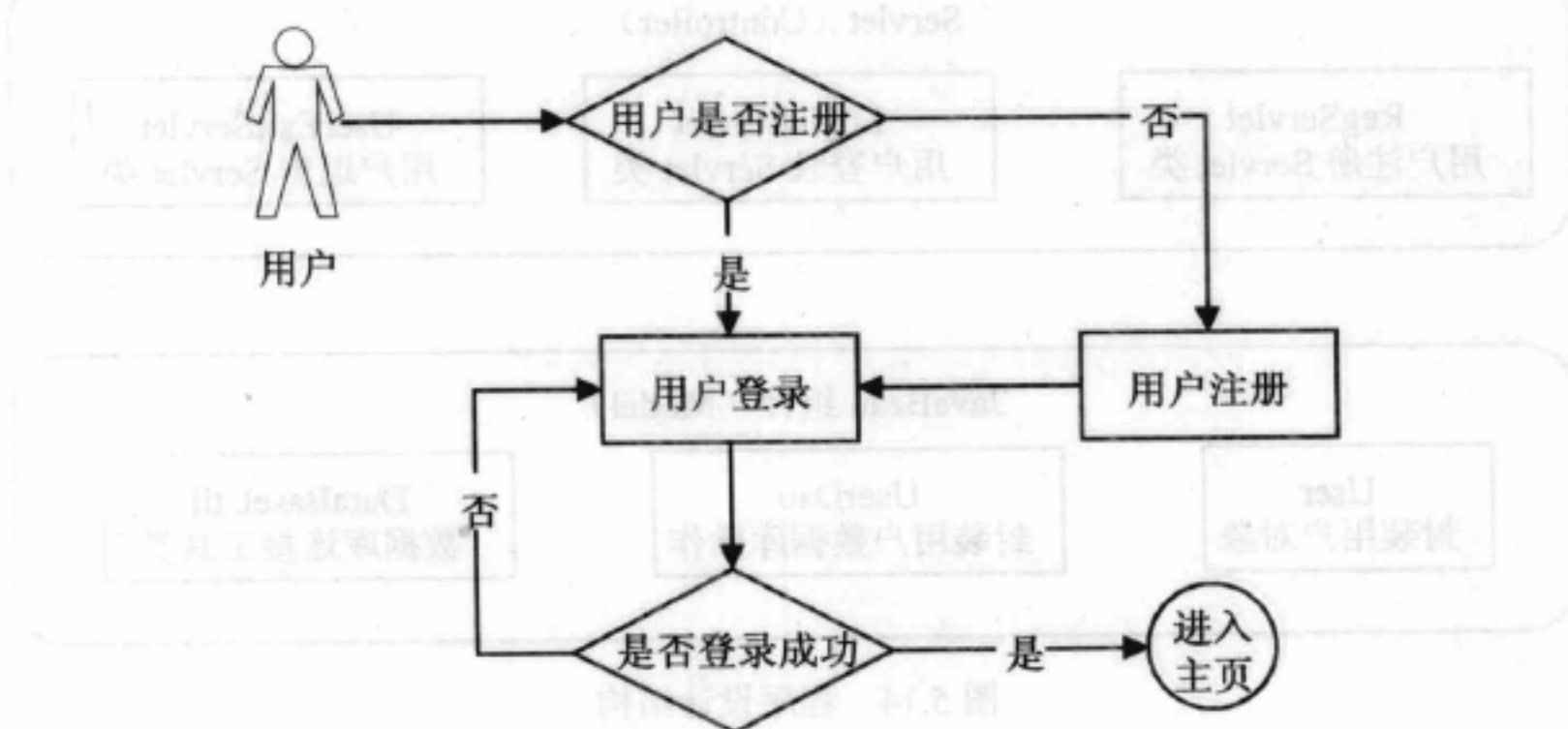


图 5.12 系统流程

用户登录之前需要进行注册,在注册成功后通过注册的用户名及密码进行登录,登录失败可以根据系统提示重新登录,登录成功后进入主页,效果如图 5.13 所示。

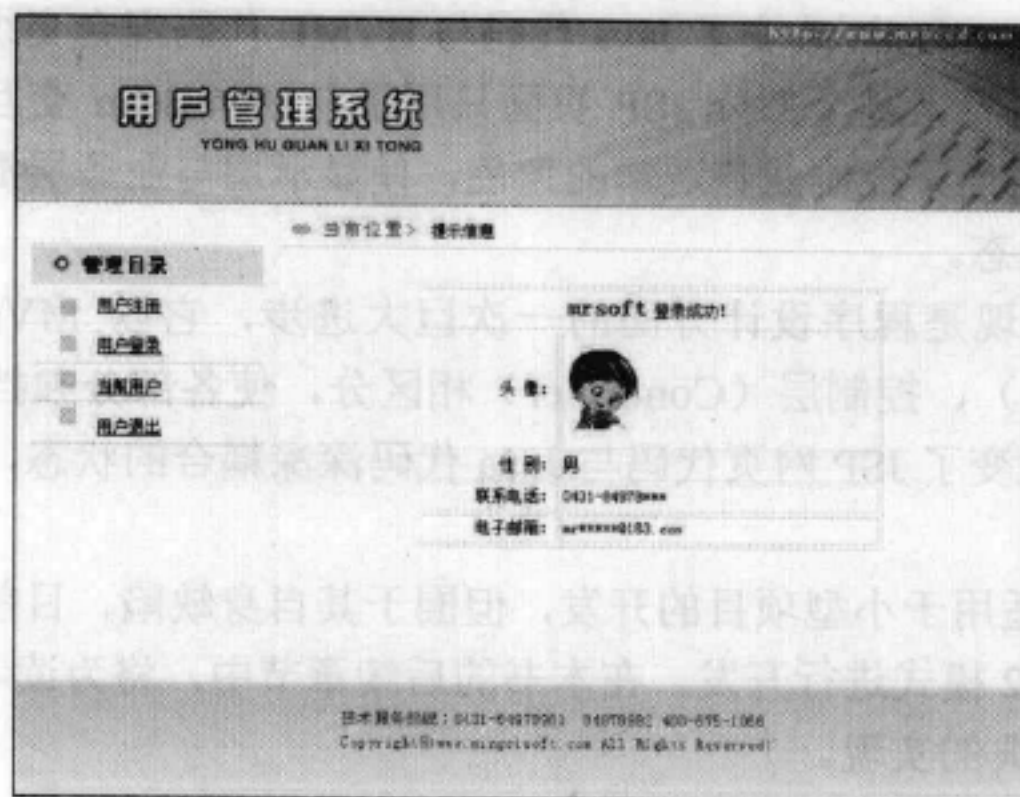


图 5.13 登录成功效果

5.5.2 关键技术

本实例为用户登录模块,采用 Model2 模式进行开发,展现了模型层 (Model)、视图层 (View)、控制层 (Controller) 的结构体系,程序设计结构如图 5.14 所示。

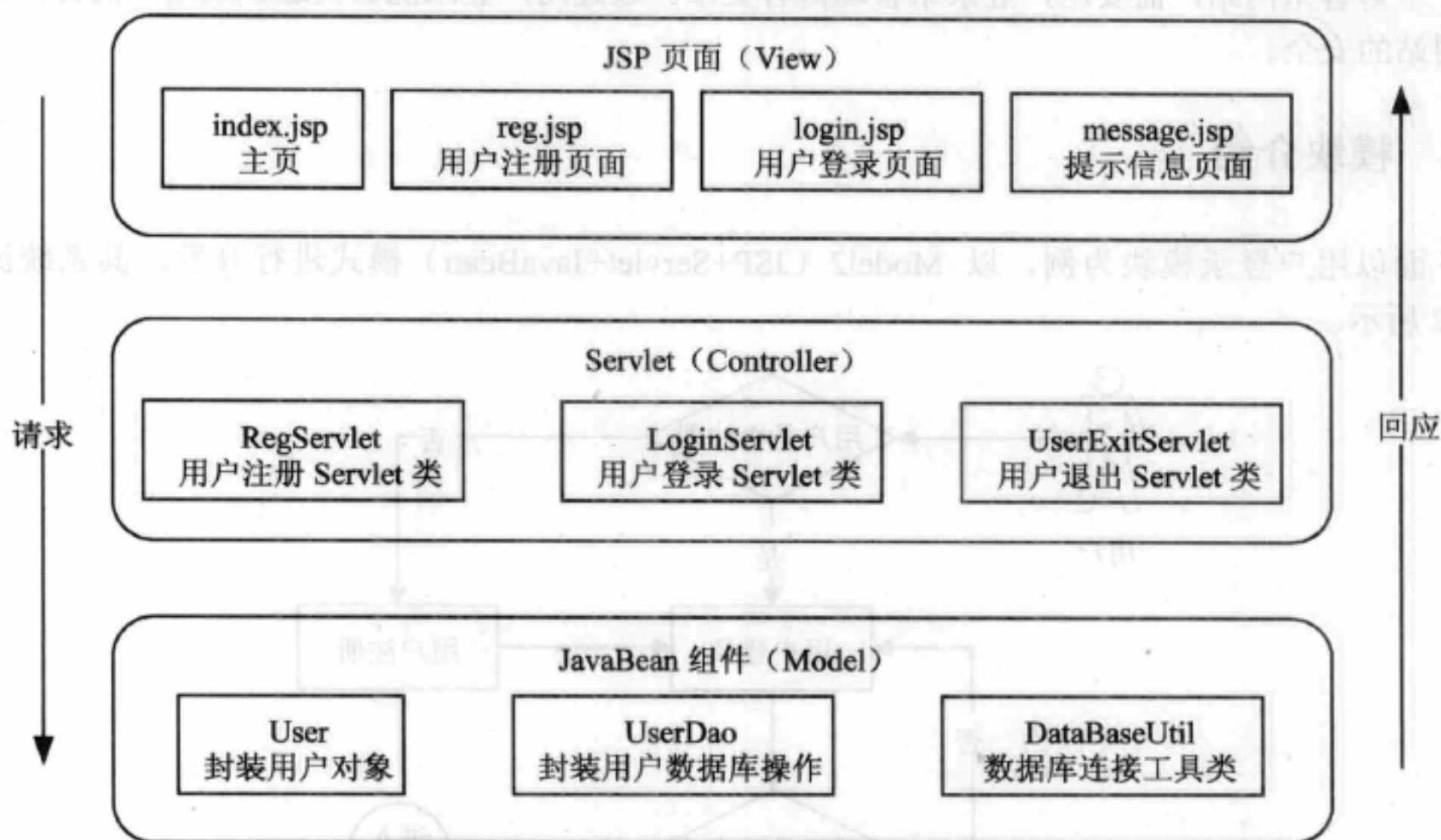


图 5.14 程序设计结构

程序为用户提供了 JSP 页面进行展示,如用户注册页面 reg.jsp、用户登录页面 login.jsp 等。这些

JSP 页面是程序的视图层 (View)，对于用户而言，通过这一层与程序进行交互，同时交互后的结果也是通过这一层回应给用户。

用户对程序的请求以及程序对用户所作出的回应由控制层 (Controller) 掌管，本实例中表现为 Servlet，如用户注册 Servlet、用户登录 Servlet 等。当用户发送一个请求时，Servlet 将判断用户的请求类型，进而提供相应的业务逻辑处理方法进行处理；请求由程序处理完毕后，又由 Servlet 控制返回处理的结果信息。此层是程序的核心部分。

对于程序的常用对象或操作，可以将其封装为一个单独的类。这样做既有利于程序的管理，又可实现代码的重用等。实例中将用户对象、用户数据库操作、数据库连接等封装为 JavaBean，从而实现程序中的模型层 (Model)。这一层为控制层提供服务，当控制层接收某一请求时，只需调用相应的 JavaBean 组件即可完成业务的处理。

5.5.3 数据库设计

本实例只涉及到一张数据表，名称为 tb_use。此表为用户信息表，用于存放用户的注册信息，其结构如图 5.15 所示。

Column Name	Datatype	PK	Null	Flags	Default Value	Comment
id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	0000	主键
username	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> BINARY	0000	用户名
password	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> BINARY	0000	密码
sex	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> BINARY	0000	性别
photo	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> BINARY	0000	头像
email	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> BINARY	0000	邮箱
tel	VARCHAR(45)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> BINARY	0000	联系电话

图 5.15 tb_user 表

5.5.4 JavaBean 设计

本实例涉及到 3 个主要的 JavaBean 组件，分别为用户实例对象 User、用户数据库操作对象 UserDao 类、数据库连接工具类 DataBaseUtil。

1. 用户实体对象

User 类用于封装用户实体对象，提供了用户对象的详细信息以及相应的 getXXX() 与 setXXX() 方法。其关键代码如下：（以下“代码位置”在光盘\TMInstance 路径下）

代码位置：UserLogin\src\com\lyq\Model\User.java

```
public class User {
    private int id;           //标识
    private String username;  //用户名
    private String password;  //密码
    private String sex;       //性别
    private String tel;       //电话
    private String photo;     //头像
    private String email;     //电子邮箱
    public int getId() {
        return id;
    }
}
```



```

    }
    public void setId(int id) {
        this.id = id;
    }
    //省略部分 getXXX()与 setXXX()方法
}

```

2. 数据库连接工具类

对于经常用到的操作可以将其封装为一个类，在类中提供相应的操作方法，从而增强代码的重用性。实例中将繁琐的获取数据库的连接与关闭操作封装到 DataBaseUtil 类中，其关键代码如下。

代码位置：UserLogin\src\com\lyq\util\DataBaseUtil.java

```

public class DataBaseUtil {
    /**
     * 获取数据库连接
     * @return Connection 对象
     */
    public static Connection getConnection(){
        Connection conn = null;
        try {
            //加载驱动
            Class.forName("com.mysql.jdbc.Driver");
            //数据库连接 url
            String url = "jdbc:mysql://localhost:3306/db_database05";
            //获取数据库连接
            conn = DriverManager.getConnection(url, "root", "111");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return conn;
    }
    /**
     * 关闭数据库连接
     * @param conn Connection 对象
     */
    public static void closeConnection(Connection conn){
        //判断 conn 是否为空
        if(conn != null){
            try {
                conn.close(); //关闭数据库连接
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

getConnection()方法用于获取数据库的连接，返回 Connection 对象；closeConnection()方法用于关闭数据库的连接。这两个方法均为静态方法，可以直接调用。

3. 用户数据库操作

与用户相关的数据库操作方法被封装在 UserDao 类中, 此类提供了实例中所用到的数据添加与查询方法, 其中 userIsExist()方法用于查询指定用户名在数据库中是否存在, 返回布尔值。其关键代码如下:

代码位置: UserLogin\src\com\lyq\model\dao\UserDao.java

```
public boolean userIsExist(String username){
    //获取数据库连接 Connection 对象
    Connection conn = DataBaseUtil.getConnection();
    //根据指定用户名查询用户信息
    String sql = "select * from tb_user where username = ?";
    try {
        //获取 PreparedStatement 对象
        PreparedStatement ps = conn.prepareStatement(sql);
        //对用户对象属性赋值
        ps.setString(1, username);
        //执行查询获取结果集
        ResultSet rs = ps.executeQuery();
        //判断结果集是否有效
        if(!rs.next()){
            //如果无效则证明此用户名可用
            return true;
        }
        //释放此 ResultSet 对象的数据库和 JDBC 资源
        rs.close();
        //释放此 PreparedStatement 对象的数据库和 JDBC 资源
        ps.close();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally{
        //关闭数据库连接
        DataBaseUtil.closeConnection(conn);
    }
    return false;
}
```

在用户提交注册信息时, 需要判断所提交的用户名是否已被注册, 如果用户名已被占用则不能再次被注册。用户名是用户信息的标识, 在提交注册信息时可以使用 userIsExist()进行判断。

用户提交注册信息后, 需要对用户信息进行持久化, 以保证用户凭其信息可以登录。这就需要在 UserDao 类中提供用户信息持久化的方法, 其名称为 saveUser()。关键代码如下:

代码位置: UserLogin\src\com\lyq\model\dao\UserDao.java

```
public void saveUser(User user){
    //获取数据库连接 Connection 对象
    Connection conn = DataBaseUtil.getConnection();
    //插入用户注册信息的 SQL 语句
    String sql = "insert into tb_user(username,password,sex,tel,photo,email) values(?,?,?,?,?,?)";
    try {
        //获取 PreparedStatement 对象
```



```

        PreparedStatement ps = conn.prepareStatement(sql);
        //对 SQL 语句的占位符参数进行动态赋值
        ps.setString(1, user.getUsername());
        ps.setString(2, user.getPassword());
        ps.setString(3, user.getSex());
        ps.setString(4, user.getTel());
        ps.setString(5, user.getPhoto());
        ps.setString(6, user.getEmail());
        //执行更新操作
        ps.executeUpdate();
        //释放此 PreparedStatement 对象的数据库和 JDBC 资源
        ps.close();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        //关闭数据库连接
        DataBaseUtil.closeConnection(conn);
    }
}

```

注册成功后，用户即可通过注册的用户名及密码进行登录。对于程序而言，此操作实质是根据用户所提供的用户名及密码查询用户信息，如果查询成功，证明在数据库中存在与之匹配的信息，则登录成功。这一操作通过 UserDao 类的 login() 方法来实现。其关键代码如下：

代码位置：UserLogin\src\com\lyq\model\dao\UserDao.java

```

public User login(String username, String password){
    User user = null;
    //获取数据库连接 Connection 对象
    Connection conn = DataBaseUtil.getConnection();
    //根据用户名及密码查询用户信息
    String sql = "select * from tb_user where username = ? and password = ?";
    try {
        //获取 PreparedStatement 对象
        PreparedStatement ps = conn.prepareStatement(sql);
        //对 SQL 语句的占位符参数进行动态赋值
        ps.setString(1, username);
        ps.setString(2, password);
        //执行查询获取结果集
        ResultSet rs = ps.executeQuery();
        //判断结果集是否有效
        if(rs.next()){
            //实例化一个用户对象
            user = new User();
            //对用户对象属性赋值
            user.setId(rs.getInt("id"));
            user.setUsername(rs.getString("username"));
            user.setPassword(rs.getString("password"));
            user.setSex(rs.getString("sex"));
            user.setTel(rs.getString("tel"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```



```

        user.setPhoto(rs.getString("photo"));
        user.setEmail(rs.getString("email"));
    }
    //释放此 ResultSet 对象的数据库和 JDBC 资源
    rs.close();
    //释放此 PreparedStatement 对象的数据库和 JDBC 资源
    ps.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    //关闭数据库连接
    DataBaseUtil.closeConnection(conn);
}
return user;
}

```

5.5.5 实现过程

用户登录模块遵循 MVC 模式进行设计，主要对用户注册、用户登录、用户退出、查看当前登录用户等请求进行实现。其实现过程如下：

1. 用户注册

(1) 创建名为 RegServlet 的类（即处理用户注册请求的 Servlet 对象），通过 doPost() 方法对用户注册请求进行处理。其关键代码如下：

代码位置：UserLogin\src\com\lyq\service\RegServlet.java

```

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //获取用户名
    String username = request.getParameter("username");
    //获取密码
    String password = request.getParameter("password");
    //获取性别
    String sex = request.getParameter("sex");
    //获取头像
    String photo = request.getParameter("photo");
    //获取联系电话
    String tel = request.getParameter("tel");
    //获取电子邮箱
    String email = request.getParameter("email");
    //实例化 UserDao 对象
    UserDao userDao = new UserDao();
    if(username != null && !username.isEmpty()){
        if(userDao.userIsExist(username)){
            //实例化一个 User 对象
            User user = new User();
            //对用户对象中的属性赋值
            user.setUsername(username);

```



```

        user.setPassword(password);
        user.setSex(sex);
        user.setPhoto(photo);
        user.setTel(tel);
        user.setEmail(email);
        //保存用户注册信息
        userDao.saveUser(user);
        request.setAttribute("info", "恭喜, 注册成功! <br>");
    } else {
        request.setAttribute("info", "错误: 此用户名已存在! ");
    }
}
//转发到 message.jsp 页面
request.getRequestDispatcher("message.jsp").forward(request, response);
}

```

在处理过程中, 首先通过 request 的 `getParameter()` 方法获取用户的注册信息, 如用户名、密码等; 然后通过 UserDao 类的 `userIsExist()` 方法判断所提交的用户名是否已被注册, 如果没有被注册则将用户提交的注册信息写入到数据库中, 否则进行错误处理; 对于用户注册的结果信息由 message.jsp 页面予以显示。

(2) 创建视图层 (为用户提供的注册页面), 名称为 reg.jsp, 在其中提供用户注册的表单。其关键代码如下:

代码位置: UserLogin\WebRoot\reg.jsp

```

<form action="RegServlet" method="post" onsubmit="return reg(this);">
  <table align="center" width="450" border="0">
    <tr>
      <td align="right">用户名: </td>
      <td>
        <input type="text" name="username">
      </td>
    </tr>
    <tr>
      <td align="right">密 码: </td>
      <td>
        <input type="password" name="password">
      </td>
    </tr>
    <tr>
      <td align="right">确认密码: </td>
      <td>
        <input type="password" name="repassword">
      </td>
    </tr>
    <tr>
      <td align="right">性 别: </td>
      <td>
        <input type="radio" name="sex" value="男" checked="checked">男
        <input type="radio" name="sex" value="女">女
      </td>
    </tr>
  </table>
</form>

```



```

</td>
</tr>
<tr>
<td align="right">头像: </td>
<td>
<select name="photo" id="photo" onchange="change();">
<option value="images/1.gif" selected="selected">头像一</option>
<option value="images/2.gif">头像二</option>
</select>

</td>
</tr>
<tr>
<td align="right">联系电话: </td>
<td>
<input type="text" name="tel">
</td>
</tr>
<tr>
<td align="right">电子邮箱: </td>
<td>
<input type="text" name="email">
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="注册">
<input type="reset" value="重置">
</td>
</tr>
</table>
</form>

```

此表单以 post 提交方式将请求发送到 RegServlet, RegServlet 类将根据用户提供的用户信息进行相应处理。reg.jsp 页面运行结果如图 5.16 所示。

图 5.16 用户注册页面

2. 用户登录

(1) 创建名为 LoginServlet 的类 (即处理用户登录请求的 Servlet), 通过 doPost() 方法对用户登

录进行处理。其关键代码如下：

代码位置：UserLogin\src\com\lyq\service>LoginServlet.java

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //获取用户名
    String username = request.getParameter("username");
    //获取密码
    String password = request.getParameter("password");
    //实例化 UserDao 对象
    UserDao userDao = new UserDao();
    //根据密码查询用户
    User user = userDao.login(username, password);
    //判断 user 是否为空
    if(user != null){
        //将用户对象放入 Session 中
        request.getSession().setAttribute("user", user);
        //转发到 result.jsp 页面
        request.getRequestDispatcher("message.jsp").forward(request, response);
    }else{
        //登录失败
        request.setAttribute("info", "错误：用户名或密码错误！");
        request.getRequestDispatcher("message.jsp").forward(request, response);
    }
}
```

在获取用户提供的用户名及密码后，通过 UserDao 类的 login()方法查询用户信息，如果查询到的用户信息不为 null，则用户登录成功，将获取到的用户对象写入到 Session 中，否则进行相应的错误处理。

(2) 创建视图层（为用户提供的登录页面），名称为 login.jsp，在其中提供用户登录表单。其关键代码如下：

代码位置：UserLogin\WebRoot\login.jsp

```
<form action="LoginServlet" method="post" onsubmit="return login(this);">
  <table align="center" width="300" border="0" class="tbl">
    <tr>
      <td align="right">用户名： </td>
      <td>
        <input type="text" name="username">
      </td>
    </tr>
    <tr>
      <td align="right">密 码： </td>
      <td>
        <input type="password" name="password">
      </td>
    </tr>
  </table>
```



```

        <td colspan="2" align="center" height="50">
            <input type="submit" value="登 录">
            <input type="reset" value="重 置">
        </td>
    </tr>
</table>
</form>

```

3. 用户退出

用户退出请求由 UserExitServlet 类进行处理, 它是一个 Servlet 对象。此类通过 doGet() 方法对退出请求进行操作, 此操作需要将存放在 Session 中的 User 对象逐出。其关键代码如下:

代码位置: UserLogin\src\com\lyq\service\UserExitServlet.java

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //获取 session
    HttpSession session = request.getSession();
    //获取用户对象
    User user = (User)session.getAttribute("user");
    //判断用户是否有效
    if(user != null){
        //将用户对象逐出 Session
        session.removeAttribute("user");
        //设置提示信息
        request.setAttribute("info", user.getUsername() + " 已成功退出!");
    }
    //转发到 message.jsp 页面
    request.getRequestDispatcher("message.jsp").forward(request, response);
}

```

4. 提示信息页面

程序在处理业务请求后, 需要告知用户处理结果, 如用户注册成功、用户登录失败等, 所以实例中提供了 message.jsp 页面, 用于用户提供系统提示信息。其关键代码如下:

代码位置: UserLogin\WebRoot\message.jsp

```

<%
    //获取提示信息
    String info = (String)request.getAttribute("info");
    //如果提示信息不为空, 则输出提示信息
    if(info != null){
        out.println(info);
    }
    //获取登录的用户信息
    User user = (User)session.getAttribute("user");
    //判断用户是否登录
    if(user != null){
%>

```



```

<table align="center" width="350" border="1" height="200" bordercolor="#E8F4CC">
  <tr>
    <td align="center" colspan="2">
      <span style="font-weight: bold;font-size: 18px;"><%=user.getUsername() %></span>
      登录成功!
    </td>
  </tr>
  <tr>
    <td align="right" width="30%">头 像: </td>
    <td>
      
    </td>
  </tr>
  <tr>
    <td align="right">性 别: </td>
    <td><%=user.getSex()%></td>
  </tr>
  <tr>
    <td align="right">联系电话: </td>
    <td><%=user.getTel()%></td>
  </tr>
  <tr>
    <td align="right">电子邮箱: </td>
    <td><%=user.getEmail()%></td>
  </tr>
</table>
<%
  }else{
    out.println("<br>对不起, 您还没有登录!");
  }
%>

```

此页面主要用于输出系统的提示信息, 包含错误提示及登录成功消息, 例如, 用户登录失败的提示信息如图 5.17 所示。

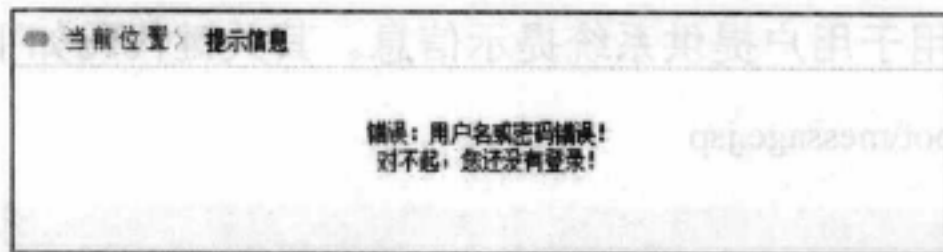


图 5.17 提示信息

5. Servlet 配置

对于 Servlet 对象的使用需要进行配置, 其配置内容包含在 web.xml 文件中。实例中主要用到了 3 个 Servlet, 其配置代码如下:

代码位置: UserLogin\WebRoot\WEB-INF\web.xml

```

<!-- 用户注册 -->
<servlet>

```



```

<servlet-name>RegServlet</servlet-name>
<servlet-class>com.lyq.service.RegServlet</servlet-class>
</servlet>
<!-- 用户登录 -->
<servlet>
<servlet-name>LoginServlet</servlet-name>
<servlet-class>com.lyq.service.LoginServlet</servlet-class>
</servlet>
<!-- 用户退出 -->
<servlet>
<servlet-name>UserExitServlet</servlet-name>
<servlet-class>com.lyq.service.UserExitServlet</servlet-class>
</servlet>
<!-- Servlet 映射 -->
<servlet-mapping>
<servlet-name>RegServlet</servlet-name>
<url-pattern>/RegServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>LoginServlet</servlet-name>
<url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>UserExitServlet</servlet-name>
<url-pattern>/UserExitServlet</url-pattern>
</servlet-mapping>

```

5.6 运行项目

项目开发完成后，就可以在 MyEclipse 中运行该项目了。具体步骤如下：

(1) 在 MyEclipse 的包资源管理器中选中 UserLogin 项目，单击鼠标右键，在弹出的快捷菜单中选择“运行方式”/MyEclipse Server Application 命令，如图 5.18 所示，此时 MyEclipse 将对项目自动部署并运行。



图 5.18 运行 UserLogin 项目

(2) 如果在 MyEclipse 中配置了多个 Web 服务器，则将弹出 Server Selection 对话框，如图 5.19 所示，从中选择所需 Web 服务器即可；如果没有配置多个 Web 服务器，则不会出现此对话框。此时在控制台中将显示启动信息，当出现如图 5.20 所示提示信息后，即表示 Web 服务器（实例中为 Tomcat）启动成功。

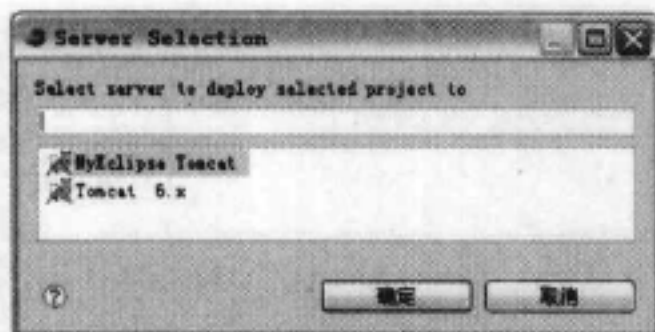


图 5.19 Server Selection 对话框

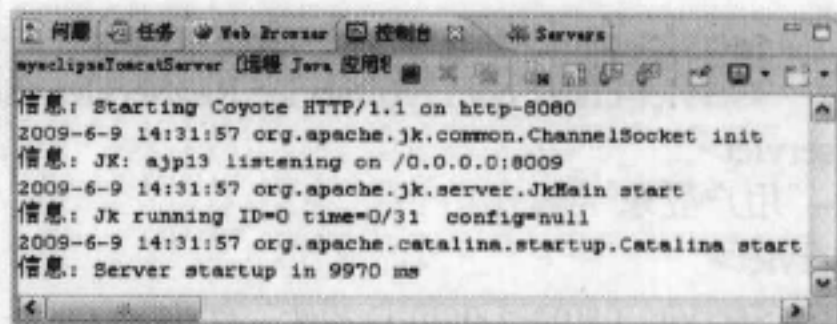


图 5.20 提示 Tomcat 启动成功的控制台提示信息

(3) 在 Web 服务器启动成功后, MyEclipse 将通过内置的浏览器 (MyEclipse Web Browser) 打开项目主页, 其运行效果如图 5.21 所示。

注意: MyEclipse Web Browser 默认通过计算机名进行访问, 其地址栏中地址以 “http://+计算机名” 开始, 其效果与 “http://+localhost” 及 “http://+IP 地址” 的访问效果相同。

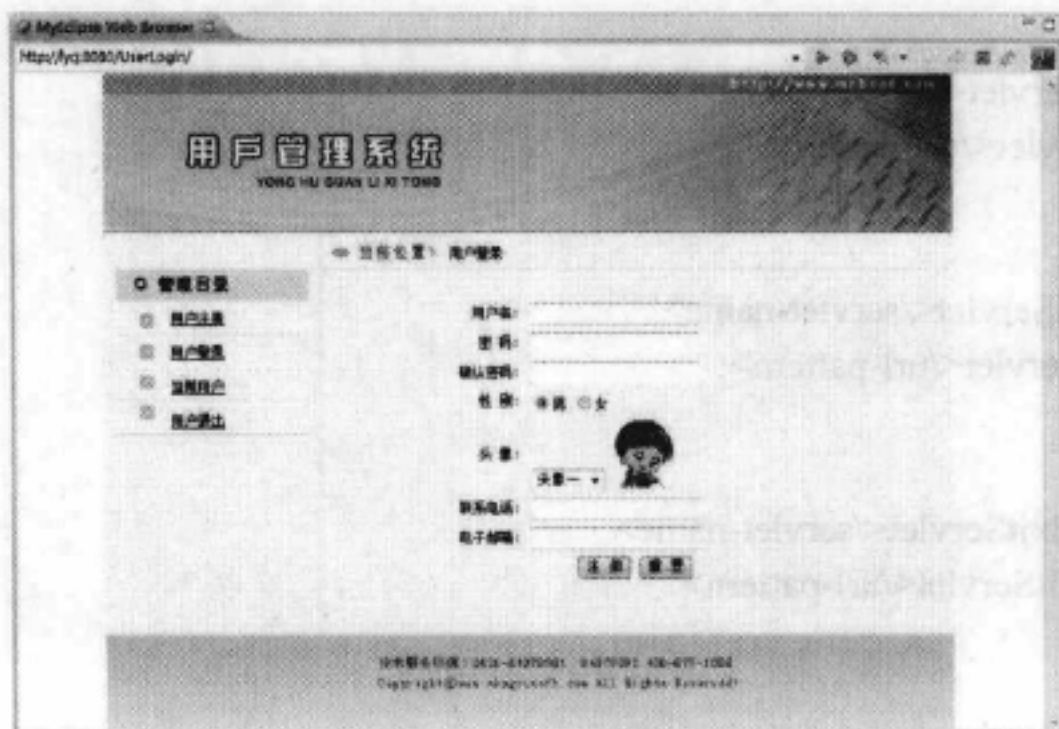


图 5.21 UserLogin 项目运行效果

5.7 本章小结

Model1 (JSP+JavaBean) 与 Model2 (JSP+Servlet+JavaBean) 是 JSP 开发 Web 应用程序的两种架构模式, Model1 开发模式比较适合小型项目的开发, 但囿于其自身缺陷, 不能达到一种理想的设计状态, 目前已很少使用, 而是采用 Model2 模式遵循 MVC 设计理念进行实际开发。在这两种架构模式中, 都离不开 JavaBean 组件的使用, 所以在学习过程中要重点掌握 JavaBean 的使用。

5.8 实战练习

1. 根据 JavaBean 的作用域, 实现简单计数器。(答案位置: 光盘\TM\Instances\5.08)
2. Model2 模式实现用户留言。(答案位置: 光盘\TM\Instances\5.09)
3. 使用 Model2 模式向数据库添加图书信息。(答案位置: 光盘\TM\Instances\5.10)

第 2 部分

高级技术

- » 第 6 章 EL 表达式语言
- » 第 7 章 JSTL 核心标签库
- » 第 8 章 结合 JSTL 与 EL 技术开发通讯录模块
- » 第 9 章 JSP 操作 XML
- » 第 10 章 JavaScript 脚本语言
- » 第 11 章 Ajax 实现用户注册模块


第2卷

高聚物

第1章 高聚物	1
第2章 高聚物的物理性质	10
第3章 高聚物的化学性质	20
第4章 高聚物的合成	30
第5章 高聚物的降解	40
第6章 高聚物的应用	50
第7章 高聚物的测试	60
第8章 高聚物的改性	70
第9章 高聚物的回收	80
第10章 高聚物的未来	90
第11章 高聚物的参考文献	100

第 6 章

EL 表达式语言

( 视频讲解: 60 分钟)

JSP 2.0 引入了一个新的内容——EL 表达式语言。通过 EL 表达式语言，可以简化在 JSP 开发中对对象的引用，从而规范页面代码，增强程序的可读性及可维护性。本章将对 EL 表达式语言的语法、基本应用、运算符及隐含对象进行详细介绍。

通过阅读本章，您可以：

- ▶▶ 了解使用 EL 表达式的前提条件
- ▶▶ 掌握 EL 表达式的基本语法
- ▶▶ 了解 EL 表达式的特点
- ▶▶ 了解 EL 表达式的存取范围
- ▶▶ 掌握 EL 表达式的各种运算符以及运算符的优先级
- ▶▶ 了解 EL 表达式中的关键字
- ▶▶ 掌握 EL 表达式的隐含对象

6.1 EL 概述


 视频讲解: 光盘\TM\Video\6\EL 概述.exe

EL 是 Expression Language 的简称,意思是表达式语言(下文中将其称为 EL 表达式)。它是 JSP 2.0 中引入的一种计算和输出 Java 对象的简单语言,为不熟悉 Java 语言页面开发的人员提供了一种开发 JSP 应用程序的新途径。

6.1.1 使用 EL 表达式的前提条件

如今 EL 表达式已经是一项成熟、标准的技术,只要安装的 Web 服务器能够支持 Servlet 2.4/JSP 2.0,就可以在 JSP 页面中直接使用 EL 表达式。

由于在 JSP 2.0 以前不存在 EL 表达式,为了和以前的规范兼容,JSP 特意提供了禁用 EL 表达式功能。

 注意: 为了保证页面能正确解析 EL 表达式,需要确认 EL 表达式没有被禁用。

JSP 中提供了以下 3 种禁用 EL 表达式的方法,只有确认 JSP 中没有通过以下 3 种方法禁用 EL 表达式,才可以正确解析 EL 表达式,否则 EL 表达式的内容将原样显示到页面中。


1. 使用斜杠“\”符号

一种比较简单的禁用 EL 表达式的方法是,在 EL 表达式的起始标记前加上“\”符号,即在“\${”之前加“\”。具体的语法如下:

```
\${expression}
```

例如,要禁用页面中的 EL 表达式\${username},可以使用以下代码。

```
\${username}
```

 说明: 该方法适合只是禁用页面的一个或几个 EL 表达式的情况。


2. 使用 page 指令

使用 JSP 的 page 指令也可以禁用 EL 表达式,其语法格式如下。

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

在上面的语法中,如果值为 true,则忽略页面中的 EL 表达式,否则将解析页面中的 EL 表达式。例如,如果想忽略页面中的 EL 表达式可以在页面的顶部添加以下代码:


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

 说明: 该方法适用于禁用一个页面中的 EL 表达式。

3. 在 web.xml 文件中配置<el-ignored>元素

在 web.xml 文件中配置<el-ignored>元素可以实现禁用服务器中的 EL 表达式。在 web.xml 文件中配置<el-ignored>元素的具体代码如下：


```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <el-ignored>true</el-ignored><!--将此处的值设置为 false，表示使用 EL 表达式-->
  </jsp-property-group>
</jsp-config>
```

 说明：该方法适用于禁用 Web 应用中所有的 JSP 页面。

6.1.2 EL 表达式的基本语法

EL 表达式语法很简单，它以“\${”开头，以“}”结束，中间为合法的表达式。具体的语法格式如下：

```
${expression}
```

 技巧：由于 EL 表达式的语法以“\${”开头，所以如果在 JSP 网页中要显示“\${”字符串，必须在前面加上“\”符号，即“\\${”，或者写成“\${'\$'}”，也就是用表达式来输出“\${”符号。

在 EL 表达式中要输出一个字符串，可以将此字符串放在一对单引号或双引号内。例如，要在页面中输出字符串“用代码书写人生”，可以使用以下代码。

```
${"用代码书写人生"}
```

6.1.3 EL 表达式的特点

EL 表达式具有以下特点：


- ☒ 在 EL 表达式中可以获得命名空间（PageContext 对象，它是页面中所有其他内置对象的最大范围的集成对象，通过它可以访问其他内置对象）。
- ☒ EL 表达式不仅可以访问一般变量，而且可以访问 JavaBean 中的属性以及嵌套属性和集合对象。
- ☒ 在 EL 表达式中可以执行关系运算、逻辑运算和算术运算等。
- ☒ 扩展函数可以与 Java 类的静态方法进行映射。
- ☒ 在表达式中可以访问 JSP 的作用域（request、session、application 以及 page）。
- ☒ EL 表达式可以与 JSTL 结合使用，也可以与 JavaScript 语句结合使用。

6.2 EL 表达式的存取范围


当 EL 表达式中的变量没有指定范围时,系统默认从 page 范围中查找,然后依次在 request、session 及 application 范围内查找。如果在此过程中找到指定的变量,则直接返回,否则返回 null。另外,EL 表达式还提供了指定存取范围的方法。在要输出表达式的前面加入指定存取范围的前缀即可指定该变量的存取范围。EL 表达式中用于指定变量使用范围的前缀如表 6.1 所示。

表 6.1 EL 表达式中使用的变量范围前缀

范 围	前 缀	举 例 说 明
page	pageScope	例如, <code>\${pageScope.username}</code> 表示在 page 范围内查找变量 username, 若找不到直接返回 null
request	requestScope	例如, <code>\${requestScope.username}</code> 表示在 request 范围内查找变量 username, 若找不到直接返回 null
session	sessionScope	例如, <code>\${sessionScope.username}</code> 表示在 session 范围内查找变量 username, 若找不到直接返回 null
application	applicationScope	例如, <code>\${applicationScope.username}</code> 表示在 application 范围内查找变量 username, 若找不到直接返回 null

 说明: 这里所说的前缀,实际上就是 EL 表达式提供的用于访问作用域范围的隐含对象。关于隐含对象的详细介绍参见 6.5 节。

6.3 EL 表达式的运算符

 视频讲解: 光盘\TM\Video\6\EL 表达式的运算符.exe

在 JSP 中,EL 表达式提供了存取数据运算符、算术运算符、关系运算符、逻辑运算符、条件运算符及 empty 运算符,下面进行详细介绍。

6.3.1 存取数据运算符“[]”和“.”

在 EL 表达式中可以使用运算符“[]”和“.”来取得对象的属性。例如, `${user.name}` 或者 `${user[name]}` 都是表示获取对象 user 中的 name 属性值。通常情况下,获取指定对象的属性使用的是“.”运算符;但是当属性名中包含一些特殊符号(如“.”或者“-”等非字母或数字符号)时,就只能使用“[]”来访问属性值了。例如, `${sessionScope.user[user-name]}` 是正确的,而 `${sessionScope.user.user-name}` 是错误的。

另外,在 EL 表达式中可以使用“[]”运算符来读取数组、Map、List 或者对象容器中的数据。下面进行详细介绍。

(1) 数组元素的获取

使用“[]”运算符可以获取数组的指定元素。例如,向 request 域中保存一个包含 4 个元素的一维

数组，并应用 EL 表达式输出该数组的第二个元素的代码如下。

```
<%
String[] arrFruit={"苹果","西瓜","芒果","荔枝"};    //定义数组
request.setAttribute("fruit",arrFruit);              //将数组保存到 request 对象中
%>
${requestScope.fruit[1]}                             <!-- 输出数组中第二个元素 -->
```

(2) List 集合元素的获取

使用“[]”运算符可以获取 List 集合中的指定元素。例如，向 session 域中保存一个包含 3 个元素的 List 集合对象，并应用 EL 表达式输出该集合的第二个元素的代码如下。

```
<%
List list = new ArrayList();
list.add("苹果");
list.add("西瓜");
list.add("芒果");
session.setAttribute("fruitList",list);    //将 List 集合保存到 session 对象中
%>
${sessionScope.fruitList[1]}              <!-- 输出集合中第二个元素 -->
```

(3) Map 集合元素的获取

使用“[]”运算符可以获取 Map 集合中的指定元素。例如，向 session 域中保存一个包含 3 个键值的 Map 集合对象，并应用 EL 表达式输出该集合的第二个元素的代码如下。

```
<%
Map map= new HashMap();
map.put("1", "苹果");
map.put("2", "西瓜");
map.put("3", "香蕉");
application.setAttribute("fruitMap",map);    //将数组保存到 application 对象中
%>
${applicationScope.fruitMap["1"]}           <!-- 输出集合中“1”键所对应的值 -->
```

6.3.2 算术运算符

EL 表达式中提供了如表 6.2 所示的算术运算符，这些运算符多数是 Java 中常用的操作符。

表 6.2 EL 表达式的算术运算符

运 算 符	功 能	举 例 说 明
+	加	$\{6+1\}$ ，返回值为 7
-	减	$\{7-1\}$ ，返回值为 6
*	乘	$\{23.5*10\}$ ，返回值为 235.0
/或 div	除	$\{16/2\}$ 或 $\{16 \text{ div } 2\}$ ，返回值为 8.0
%或 mod	求余	$\{15\%4\}$ 或 $\{15 \text{ mod } 4\}$ 返回值为 3

⚠ 注意：EL 表达式无法像 Java 一样将两个字符串用 “+” 运算符连接在一起 (“明”+“日”)，所以 `${"明"+"日"}` 的写法是错误的。但是，可以采用 `${"明"}${"日"}` 的形式来表示。

6.3.3 关系运算符

在 EL 表达式中提供了用于对两个表达式进行比较的关系运算符，如表 6.3 所示。EL 表达式的关系运算符不仅可以用来比较整数和浮点数，还可以用来比较字符串。

表 6.3 EL 表达式的关系运算符

运 算 符	功 能	举 例 说 明
==或 eq	等于	<code>\${10==10}</code> 或 <code>\${10 eq 10}</code> ，返回 true
		<code>\${"A"=="a"}</code> 或 <code>\${"A" eq "a"}</code> ，返回 false
!=或 ne	不等于	<code>\${10!=10}</code> 或 <code>\${10 ne 10}</code> ，返回 false
		<code>\${"A"!="A"}</code> 或 <code>\${"A" ne "A"}</code> ，返回 false
<或 lt	小于	<code>\${5<3}</code> 或 <code>\${5 lt 3}</code> ，返回 false
		<code>\${"A"<"B"}</code> 或 <code>\${"A" lt "B"}</code> ，返回 true
>或 gt	大于	<code>\${5>3}</code> 或 <code>\${5 gt 3}</code> ，返回 true
		<code>\${"A">"B"}</code> 或 <code>\${"A" gt "B"}</code> ，返回 false
<=或 le	小于等于	<code>\${3<=4}</code> 或 <code>\${3 le 4}</code> ，返回 true
		<code>\${"A"<="A"}</code> 或 <code>\${"A" le "A"}</code> ，返回 true
>=或 ge	大于等于	<code>\${3>=4}</code> 或 <code>\${3 ge 4}</code> ，返回 false
		<code>\${"A">="B"}</code> 或 <code>\${"A" ge "B"}</code> ，返回 false

⚠ 注意：在使用 EL 表达式关系运算符时，不能写成如下形式。

```
${param.pwd1} == ${param.pwd2}
```

或

```
`${${param.pwd1}} == ${param.pwd2}`
```

而应写成：

```
${param.pwd1} = param.pwd2}
```

6.3.4 逻辑运算符

同 Java 语言一样，EL 表达式也提供了与、或、非 3 种逻辑运算符，下面进行详细介绍。


1. “&&” 或 and 运算符

“&&” 或 and 运算符为与运算符，这与 Java 中的与运算符相同，也是只有在两个操作数的值均为 true 时，才返回 true，否则返回 false。

例如下面的 EL 表达式:

```
${username == "mr" && pwd == "mrsoft"}
```

只有当 username 的值为 mr, pwd 的值为 mrsoft 时, 返回值才为 true, 否则将返回 false。

 **说明:** 与运算符在执行的过程中, 只要表达式的值可以确定, 就会停止执行。例如, 进行多个与运算时, 如果遇到其中一个操作数的值为 false, 则停止执行, 并返回 false。


2. “||” 或 or 运算符

“||” 或 or 运算符为或运算符, 这与 Java 中的或运算符相同, 也是只要有一个操作数的值为 true, 就返回 true, 只有全部操作数均为 false 时, 才返回 false。

例如下面的 EL 表达式:

```
${username == "mr" || username == "mrsoft"}
```

只要 username 的值为 mr, 或 pwd 的值为 mrsoft, 就返回 true, 否则才返回 false。

 **说明:** 或运算符同与运算符一样, 在执行的过程中, 只要表达式的值可以确定, 就会停止执行。例如, 进行多个或运算时, 如果遇到其中一个操作数的值为 true, 则停止执行, 并返回 true。

3. “!” 或 not 运算符

“!” 或 not 运算符为非运算符, 这与 Java 中的非运算符相同, 也是对操作数进行取反。如果原来操作数的值为 true, 则返回 false; 如果原来操作数的值为 false, 则返回 true。

例如下面的 EL 表达式:

```
${! username == "mr"}
```

如果 username 的值为 mr, 则返回 false, 否则返回 true。


6.3.5 empty 运算符

在 EL 表达式中, 有一个特殊的运算符——empty。该运算符是一个前缀 (prefix) 运算符, 即 empty 运算符位于操作数前方, 用来确定一个对象或变量是否为 null 或空。empty 运算符的语法格式如下:

```
${empty expression}
```

expression: 用于指定要判断的变量或对象。

一个变量或对象为 null 或空代表的是不同的含义: null 表示这个变量没有指明任何对象; 而空表示这个变量所属的对象其内容为空, 例如空字符串、空的数组或者空的 List 容器。

 **技巧:** empty 运算符也可以与 not 运算符结合使用, 用于确定一个对象或变量是否为非空。例如, 要判断 session 域中的变量 user 不为空, 可以使用以下代码。

```
${not empty sessionScope.user}
```

6.3.6 条件运算符

在 EL 表达式中可以利用条件运算符进行条件求值，其语法格式如下。

```
${条件表达式 ? 计算表达式 1 : 计算表达式 2}
```

在上面的语法中，如果条件表达式为真，则计算表达式 1，否则计算表达式 2。但是 EL 表达式中的条件运算符功能比较弱，一般可以用 JSTL 中的条件标签 `<c:if>` 或 `<c:choose>` 替代；当然，如果处理的问题比较简单，也可以使用。EL 表达式中的条件运算符的唯一的优点便是其非常简单和方便，和 Java 语言里的用法完全一致。

例如，应用条件运算符，当变量 `user` 的值为空时，输出“`user 为空`”，否则输出 `user` 的值。具体的代码如下：

```
${empty user ? "user 为空" : user}
```

6.3.7 运算符的优先级

运算符的优先级决定了在多个运算符同时存在时，各个运算符的求值顺序。EL 表达式中各运算符的优先级如图 6.1 所示，对于同级的运算符采用从左向右计算的原则。



图 6.1 EL 表达式中各运算符的优先级

说明：使用括号 `()` 可以改变优先级，例如 `${1 * (2+4)}` 改变了先乘除、后加减的基本规则，这是因为括号的优先级高于绝大部分的运算符。在复杂的表达式中，使用括号可以使表达式更容易阅读及避免出错。

6.4 EL 表达式中的保留字

EL 表达式中定义了如表 6.4 所示的保留字，在为变量命名时，应该避免使用这些保留字。

表 6.4 EL 表达式中的保留字

and	eq	gt	true	instanceof	div	or	ne
le	false	lt	empty	mod	not	ge	null

6.5 EL 表达式中的隐含对象

 视频讲解：光盘\TM\Video\6\EL 表达式中的隐含对象.exe

为了能够获得 Web 应用程序中的相关数据，EL 表达式中定义了一些隐含对象。这些隐含对象共有 11 个，如表 6.5 所示。

表 6.5 EL 表达式的隐含对象

	隐含对象	对象类型	说明
页面上下文对象	pageContext	javax.servlet.jsp.PageContext	用于访问 JSP 内置对象
访问环境信息的隐含对象	param	java.util.Map	包含页面所有参数的名称和对应值的集合
	paramValues	java.util.Map	包含页面所有参数的名称和对应多个值的集合
	header	java.util.Map	包含每个 header 名和值的集合
	headerValues	java.util.Map	包含每个 header 名和可能的多个值的集合
	cookie	java.util.Map	包含每个 cookie 名和值的集合
	initParam	java.util.Map	包含 Servlet 上下文初始参数名和对应值的集合
访问作用域范围的隐含对象	pageScope	java.util.Map	包含 page（页面）范围内的属性值的集合
	requestScope	java.util.Map	包含 request（请求）范围内的属性值的集合
	sessionScope	java.util.Map	包含 session（会话）范围内的属性值的集合
	applicationScope	java.util.Map	包含 application（应用）范围内的属性值的集合

在对 EL 表达式中包含的隐含对象有了初步了解后，下面对各个隐含对象进行详细介绍。

6.5.1 PageContext 对象的应用

PageContext 隐含对象用于访问 JSP 内置对象，如 request、response、out、session、config 和 servletContext 等。

例如，要获取当前 session 中的变量 username 可以使用以下的 EL 表达式。

```
${PageContext.session.username}
```

6.5.2 param 和 paramValues 对象的应用

param 对象用于获取请求参数的值，而如果一个参数名对应多个值时，则需要使用 paramValues 对象获取请求参数的值。在应用 param 对象时，返回的结果为字符串；在应用 paramValues 对象时，返回

的结果为数组。

例如，在 JSP 页面中放置一个名为 user 的文本框，关键代码如下。

```
<input name="user" type="text" id="user">
```

当表单提交后，要获取 user 文本框的值，可以使用下面的 EL 表达式。

```
${param.user}
```

⚠ 注意：如果 user 文本框中可以输入中文，那么在使用 EL 表达式输出其内容前，还需使用 `request.setCharacterEncoding("GBK");` 语句设置请求的编码为 GBK，否则将产生乱码。

再例如，在 JSP 页面中放置一个名为 affect 的复选框组，关键代码如下。

```
<input name="affect" type="checkbox" id="affect" value="体育">
体育
<input name="affect" type="checkbox" id="affect" value="美术">
美术
<input name="affect" type="checkbox" id="affect" value="音乐">
音乐
```

当表单提交后，要获取 affect 的值，可以使用下面的 EL 表达式。

```
爱好为: ${paramValues.affect[0]}${paramValues.affect[1]}${paramValues.affect[2]}
```

⚠ 注意：在应用 param 或 paramValues 对象时，如果指定的参数不存在，则返回空的字符串，而不是返回 null。

6.5.3 header 和 headerValues 对象的应用

header 对象用于获取 HTTP 请求的一个具体 header 值，但是在某些情况下，可能存在同一个 header 拥有多个不同的值，这时就必须使用 headerValues 对象。

例如，要获取 HTTP 请求的 header 的 Host 属性，可以使用以下 EL 表达式。

```
${header.host}或${header[host]}
```

上面的 EL 表达式将输出如图 6.2 所示的结果。

但是，如果要获取 HTTP 请求的 header 的 Accept-Agent 属性，则必须使用以下 EL 表达式。

```
${header["user-agent"]}
```

上面的 EL 表达式将输出如图 6.3 所示的结果。

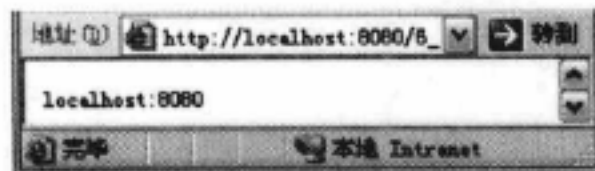


图 6.2 应用 header 对象获取的 Host 属性

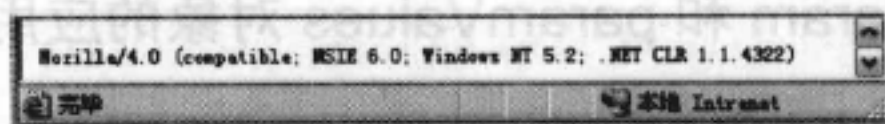


图 6.3 应用 header 对象获取的 user-agent 属性

6.5.4 访问作用域范围的隐含对象

EL 表达式中提供了 4 个用于访问作用域范围的隐含对象,即 `pageScope`、`requestScope`、`sessionScope` 和 `applicationScope`。应用这 4 个隐含对象指定查找标识符的作用域后,系统将不再按照默认的顺序 (`page`、`request`、`session` 及 `application`) 来查找相应的标识符。它们与 JSP 中的 `page`、`request`、`session` 及 `application` 内置对象类似,只不过这 4 个隐含对象只能用来取得指定范围内的属性值,而不能取得其他相关信息。

例如,要获取 `session` 范围内的 `user` 变量的值,可以使用以下 EL 表达式。

```
${sessionScope.user}
```


6.5.5 cookie 对象的应用

`cookie` 对象用于访问由请求设置的 `cookie` 名称。如果在 `cookie` 中已经设定一个名为 `username` 的值,那么可以使用 `${cookie.username}` 来获取该 `cookie` 对象;但是如果需要获取 `cookie` 中的值,则需要使用 `cookie` 对象的 `value` 属性。

例如,使用 `response` 对象设置一个请求有效的 `cookie` 对象,然后再使用 EL 表达式获取该 `cookie` 对象的值。具体代码如下:

```
<%Cookie cookie=new Cookie("user","mr");  
response.addCookie(cookie);  
%>  
${cookie.user.value}
```

运行上面的代码后,将在页面中显示 `mr`。

 **说明:** 所谓的 `cookie` 是一个文本文件,它是以 `key`、`value` 的方法将用户会话信息记录在这个文本文件内,并将其暂时存放在客户端浏览器中。

6.5.6 initParam 对象的应用

`initParam` 对象用于获取 Web 应用初始化参数的值。例如,在 Web 应用的 `web.xml` 文件中设置一个初始化参数 `author`,用于指定作者。具体代码如下:


```
<context-param>  
  <param-name>author</param-name>  
  <param-value>wgh</param-value>  
</context-param>
```

应用 EL 表达式获取该参数的代码如下:

```
${initParam.author}
```


6.6 实 战

6.6.1 应用 EL 表达式访问 JavaBean 的属性

 视频讲解：光盘\TM\Video\6\应用 EL 表达式访问 JavaBean 的属性.exe

例 6.01 在客户端的表单中填写用户注册信息后，单击“提交”按钮，应用 EL 表达式通过访问 JavaBean 属性的方法显示到页面上。（实例位置：光盘\TM\Instances\6.01）

（1）编写 index.jsp 页面，在该页面中添加用于收集用户注册信息的表单及表单元素。关键代码如下：

```
<form name="form1" method="post" action="deal.jsp">
用户名: <input name="username" type="text" id="username">
密码: <input name="pwd" type="password" id="pwd">
确认密码: <input name="repwd" type="password" id="repwd">
性别:
<input name="sex" type="radio" class="noborder" value="男">男
<input name="sex" type="radio" class="noborder" value="女">女
爱好:
<input name="affect" type="checkbox" class="noborder" id="affect" value="体育">
体育
<input name="affect" type="checkbox" class="noborder" id="affect" value="美术">
美术
<input name="affect" type="checkbox" class="noborder" id="affect" value="音乐">
音乐
<input name="affect" type="checkbox" class="noborder" id="affect" value="旅游">
旅游
<input name="Submit" type="submit" class="btn_grey" value="提交">
<input name="Submit2" type="reset" class="btn_grey" value="重置">
</form>
```

（2）编写保存用户信息的 JavaBean，将其保存到 com.wgh 包中，命名为 UserForm。具体代码如下：

```
package com.wgh;
public class UserForm {
    private String username="";           //用户名属性
    private String pwd="";               //密码属性
    private String sex="";               //性别属性
    private String[] affect=null;        //爱好属性
    public void setUsername(String username) { //设置 username 属性的方法
        this.username = username;
    }
    public String getUsername() {         //获取 username 属性的方法
        return username;
    }
    ... //此处省略了设置其他属性对应的 setXXX()和 getXXX()方法的代码
```



```

public void setAffect(String[] affect) {      //设置 affect 属性的方法
    this.affect = affect;
}
public String[] getAffect() {                //获取 affect 属性的方法
    return affect;
}
}

```

(3) 编写 deal.jsp 页面。在该页面中, 首先使用 request 内置对象的 setCharacterEncoding() 方法设置请求的编码方式为 GBK, 然后使用 <jsp:useBean> 动作指令在页面中创建一个 JavaBean 实例, 再使用 <jsp:setProperty> 动作指令设置 JavaBean 实例的各属性值, 最后使用 EL 表达式将 JavaBean 的各属性显示到页面中。具体代码如下:

```

<%@ page language="java" pageEncoding="GBK"%>
<%request.setCharacterEncoding("GBK");%>
<jsp:useBean id="userForm" class="com.wgh.UserForm" scope="page"/>
<jsp:setProperty name="userForm" property="*" />
<jsp:setProperty name="userForm" property="affect"
value="<%=request.getParameterValues("affect")%>" />
<!--显示用户注册信息-->
用户名: ${userForm.username}
密码: ${userForm.pwd}
性别: ${userForm.sex}
爱好: ${userForm.affect[0]} ${userForm.affect[1]} ${userForm.affect[2]} ${userForm.affect[3]}


```

❗ 注意: 这里一定要使用上面加粗部分的代码, 重新设置 affect 属性的值, 否则将只能获取到第一个复选框的值。

运行程序, 在页面的“用户名”文本框中输入用户名, 在“密码”文本框中输入密码, 在“确认密码”文本框中输入确认密码, 选择性别和爱好, 然后单击“提交”按钮, 如图 6.4 所示, 即可将该用户信息显示到页面中, 如图 6.5 所示。

• 用户注册	
用户名:	<input type="text" value="无话"/>
密 码:	<input type="password" value="111111"/>
确认密码:	<input type="password" value="111111"/>
性 别:	<input checked="" type="radio"/> 男 <input type="radio"/> 女
爱 好:	<input checked="" type="checkbox"/> 体育 <input checked="" type="checkbox"/> 美术 <input checked="" type="checkbox"/> 音乐 <input type="checkbox"/> 旅游
<input type="button" value="返回"/> <input type="button" value="提交"/>	

图 6.4 填写注册信息页面

• 显示用户填写的注册信息	
用户名:	无话
密 码:	111111
性 别:	女
爱 好:	体育 美术 音乐
<input type="button" value="返回"/>	

图 6.5 显示注册信息页面

6.6.2 应用 EL 表达式显示投票结果

❗ 视频讲解: 光盘\TM\Video\6\应用 EL 表达式显示投票结果.exe

例 6.02 实现投票功能并应用 EL 表达式显示投票结果。(实例位置: 光盘\TM\Instances\6.02)

(1) 编写 index.jsp 页面, 在该页面中添加用于收集投票信息的表单及表单元素。关键代码如下:

```
<form name="form1" method="post" action="PollServlet">
    • 您最需要哪方面的编程类图书?
    <input name="item" type="radio" class="noborder" value="基础教程类" checked>基础教程类
    <input name="item" type="radio" class="noborder" value="实例集锦类">实例集锦类
    <input name="item" type="radio" class="noborder" value="经验技巧类">经验技巧类
    <input name="item" type="radio" class="noborder" value="速查手册类">速查手册类
    <input name="item" type="radio" class="noborder" value="案例剖析类">案例剖析类
    <input name="Submit" type="submit" class="btn_grey" value="投票">
    <input name="Submit2" type="button" class="btn_grey" value="查看投票结果"
    onClick="window.location.href='showResult.jsp'">
</form>
```

(2) 编写完成投票功能的 Servlet, 将其保存到 com.wgh.servlet 包中, 命名为 PollServlet。在该 Servlet 的 doPost() 方法中, 首先设置请求的编码方式为 GBK, 并获取投票项; 然后判断是否存在保存投票结果的 ServletContext 对象 (该对象在 application 范围内有效), 如果存在, 则获取保存在 ServletContext 对象中的 Map 集合, 并将指定投票项的得票数加 1, 否则创建并初始化一个保存投票信息的 Map 集合, 再将保存投票结果的 Map 集合保存到 ServletContext 对象中; 最后向浏览器输出弹出提示对话框并重定向网页的 JavaScript 代码。PollServlet 的具体代码如下:

```
package com.wgh.servlet;
import java.io.IOException;
... //此处省略了导入其他包或类的代码

public class PollServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("GBK"); //设置请求的编码方式
        String item=request.getParameter("item"); //获取投票项
        //获取 ServletContext 对象, 该对象在 application 范围内有效
        ServletContext servletContext=request.getSession().getServletContext();
        Map map=null;
        if(servletContext.getAttribute("pollResult")!=null){
            map=(Map)servletContext.getAttribute("pollResult"); //获取投票结果
            map.put(item,Integer.parseInt(map.get(item).toString())+1); //将当前的投票项加 1
        }else{ //初始化一个保存投票信息的 Map 集合, 并将选定投票项的投票数设置为 1, 其他为 0
            String[] arr={"基础教程类","实例集锦类","经验技巧类","速查手册类","案例剖析类"};
            map=new HashMap();
            for(int i=0;i<arr.length;i++){
                if(item.equals(arr[i])){ //判断是否为选定的投票项
                    map.put(arr[i], 1);
                }else{
                    map.put(arr[i], 0);
                }
            }
            //初始化 Map 集合
        }
        servletContext.setAttribute("pollResult", map); //保存投票结果到 ServletContext 对象中
        //设置响应的编码方式, 如果不设置则弹出的对话框中的文字将乱码
    }
}
```



```

response.setCharacterEncoding("GBK");
PrintWriter out=response.getWriter();
out.println("<script>alert('投票成功! ');window.location.href='showResult.jsp';</script>");

}

}

```

(3) 编写 showResult.jsp 页面, 在该页面中应用 EL 表达式输出投票结果。具体代码如下:

• 您最需要哪方面的编程类图书?

基础教程类

```



```

```

(${empty applicationScope.pollResult["基础教程类"]? 0 : applicationScope.pollResult["基础教程类"]})

```

实例集锦类

```



```

```

(${empty applicationScope.pollResult["实例集锦类"]? 0 : applicationScope.pollResult["实例集锦类"]})

```

经验技巧类

```



```

```

(${empty applicationScope.pollResult["经验技巧类"]? 0 : applicationScope.pollResult["经验技巧类"]})

```

速查手册类

```



```

```

(${empty applicationScope.pollResult["速查手册类"]? 0 : applicationScope.pollResult["速查手册类"]})

```

案例剖析类

```



```

```

(${empty applicationScope.pollResult["案例剖析类"]? 0 : applicationScope.pollResult["案例剖析类"]})

```

合计:

```


${applicationScope.pollResult["基础教程类"]+applicationScope.pollResult["实例集锦类"]+applicationScope.pollResult["经验技巧类"]+applicationScope.pollResult["速查手册类"]+applicationScope.pollResult["案例剖析类"]}人投票!

```

```



```

 说明: 上面的代码中, EL 表达式 `${empty applicationScope.pollResult["案例剖析类"]? 0 : applicationScope.pollResult["案例剖析类"]}` 用于显示“案例剖析类”图书的得票数, 在该 EL 表达式中应用了条件运算符, 用于当没有投票信息时, 将得票数显示为 0。

运行程序, 将显示如图 6.6 所示的投票页面。在该页面中, 选择自己需要的编程类图书, 单击“投票”按钮, 将完成投票, 并显示投票结果, 如图 6.7 所示。在投票页面中单击“查看投票结果”按钮,

也可以查看投票结果。

您最需要什么方面的编程类图书?	
<input checked="" type="radio"/> 基础教程类	
<input type="radio"/> 实例集锦类	
<input type="radio"/> 经验技巧类	
<input type="radio"/> 速查手册类	
<input type="radio"/> 案例剖析类	
<input type="button" value="投票"/> <input type="button" value="查看投票结果"/>	

图 6.6 投票页面





您最需要什么方面的编程类图书?	
基础教程类	 (4)
实例集锦类	 (2)
经验技巧类	 (1)
速查手册类	 (6)
案例剖析类	 (1)
合计: 14人投票! <input type="button" value="返回"/>	

图 6.7 显示投票结果页面

6.7 本章小结


本章首先介绍了使用 EL 表达式的前提条件、EL 表达式的基本语法以及 EL 表达式的特点, 其中需要读者重点掌握的是使用 EL 表达式的前提条件和基本语法; 然后介绍了 EL 表达式的存取范围; 接下来又介绍了 EL 表达式的运算符, 由于 EL 表达式的运算符同 Java 的运算符类似, 所以读者可以将其与 Java 的运算符对比学习; 最后详细介绍了 EL 表达式中的隐含对象, 其中 param 对象、paramValues 对象以及访问作用域范围的隐含对象在今后的程序开发过程中会经常用到, 读者需要重点掌握。

6.8 实战练习

1. 设置当前的 Web 应用中禁止使用 EL 表达式。(答案位置: 光盘\TM\Instances\6.03)
2. 应用 EL 表达式显示客户端使用的浏览器。(答案位置: 光盘\TM\Instances\6.04)
3. 应用 EL 表达式显示客户端能够接收的内容类型。(答案位置: 光盘\TM\Instances\6.05)
4. 应用 EL 表达式判断用户是否登录, 并显示不同的提示信息。(答案位置: 光盘\TM\Instances\6.06)
5. 应用 EL 表达式判断用户名是否为空, 如果为空显示相应的提示信息。(答案位置: 光盘\TM\Instances\6.07)

第 7 章

JSTL 核心标签库

( 视频讲解: 87 分钟)

JSTL 是一个不断完善的开放源代码的 JSP 标签库, 在 JSP 2.0 中已将 JSTL 作为标准支持。使用 JSTL 可以取代在传统 JSP 程序中嵌入 Java 代码的做法, 大大提高了程序的可维护性。本章将对 JSTL 的下载和配置以及 JSTL 的核心标签进行详细介绍。

通过阅读本章, 您可以:

- » 掌握下载和配置 JSTL 的方法
- » 了解 JSTL 标签库
- » 掌握 JSTL 的核心标签库中的表达式标签
- » 掌握 JSTL 的核心标签库中的条件标签
- » 掌握 JSTL 的核心标签库中的循环标签
- » 掌握 JSTL 的核心标签库中的 URL 相关标签



图 7.1 JSP 的 JSTL 页面

7.1 JSTL 简介

JSTL (JavaServer Pages Standard Tag Library, JSP 标准标签库) 是由 Apache 的 Jakarta 小组负责维护的一个不断完善的开放源代码的 JSP 标准标签库, 它为 Java Web 开发人员提供了一个标准的、通用的标签库。

JSTL 有多个版本, 目前最新的版本是 JSTL 1.2, 必须在 JSP 2.1 版本的容器内运行。

7.1.1 下载和配置 JSTL

由于 JSTL 还不是 JSP 2.0 规范中的一部分, 所以在使用 JSTL 之前, 需要安装并配置 JSTL。可以到 Sun 的 JSTL 页面下载, 当前最新的版本为 JSTL 1.2。JSTL 的下载和配置步骤如下。

(1) 在 IE 地址栏中输入 URL 地址 “<http://java.sun.com/products/jsp/jstl/>”, 按 Enter 键, 将进入到如图 7.1 所示的页面。

(2) 单击 JSTL Project 超链接, 将进入到如图 7.2 所示的 JSTL 主页。

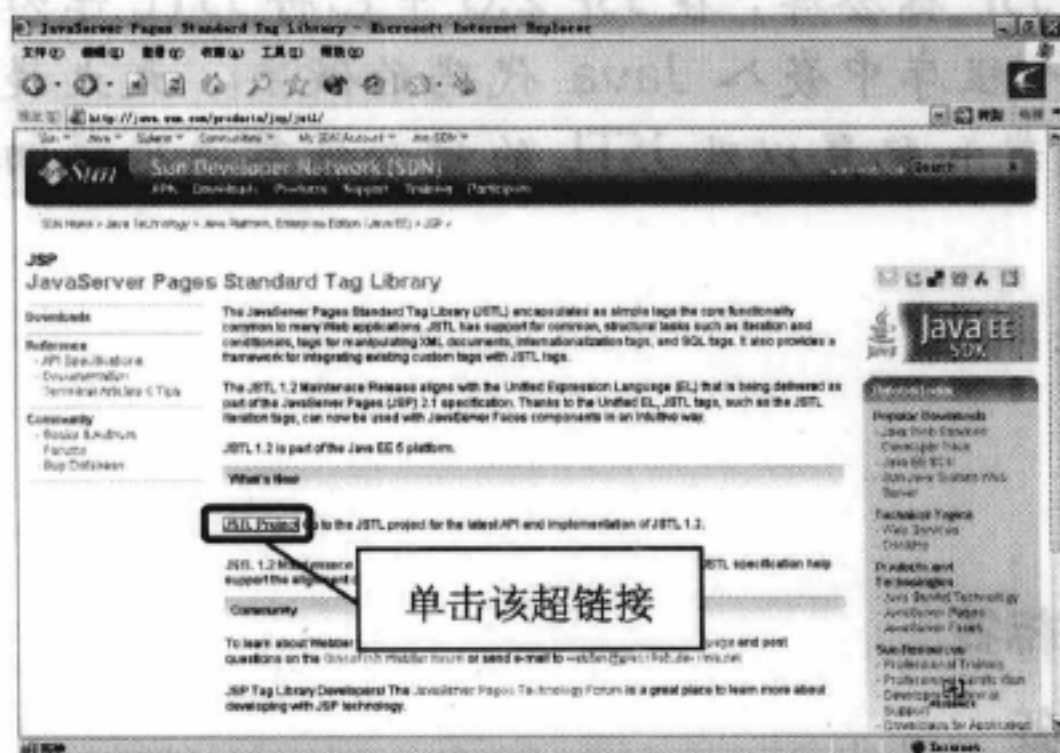


图 7.1 JSP 的 JSTL 页面

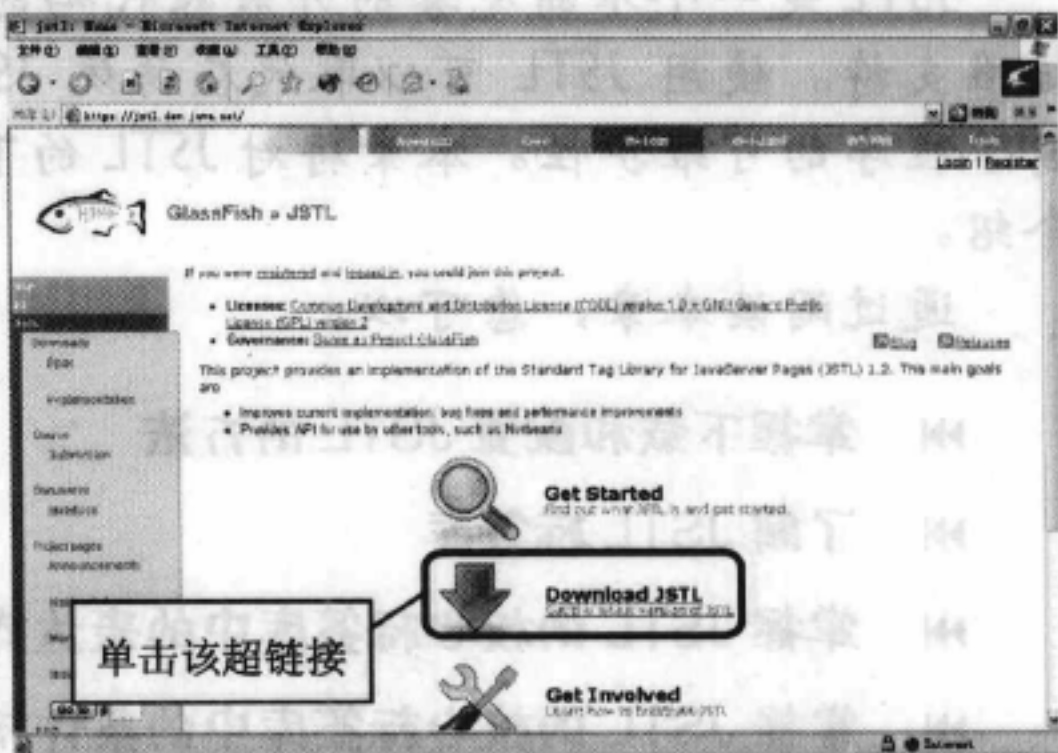


图 7.2 JSTL 的主页

(3) 单击 Download JSTL 超链接, 将进入到如图 7.3 所示的 JSTL 下载列表页面。在该页面中提供了两个下载超链接, 其中 JSTL API 用于下载 JSTL 的 API, JSTL Implementation 用于下载 JSTL 的实现品。

(4) 单击 JSTL API 超链接下载 JSTL 的 API, 下载后的文件名为 jstl-api-1.2.jar; 单击 JSTL Implementation 超链接下载 JSTL 的实现品, 下载后的文件名为 jstl-impl-1.2.jar。

(5) 当 JSTL 的标签库下载完毕后, 就可以在 Web 应用中配置 JSTL 标签库了。配置 JSTL 比较简单, 只需要将 jstl-api-1.2.jar 和 jstl-impl-1.2.jar 复制到 Web 应用的 WEB-INF\lib 目录中即可。配置 JSTL 后的 Web 应用如图 7.4 所示。

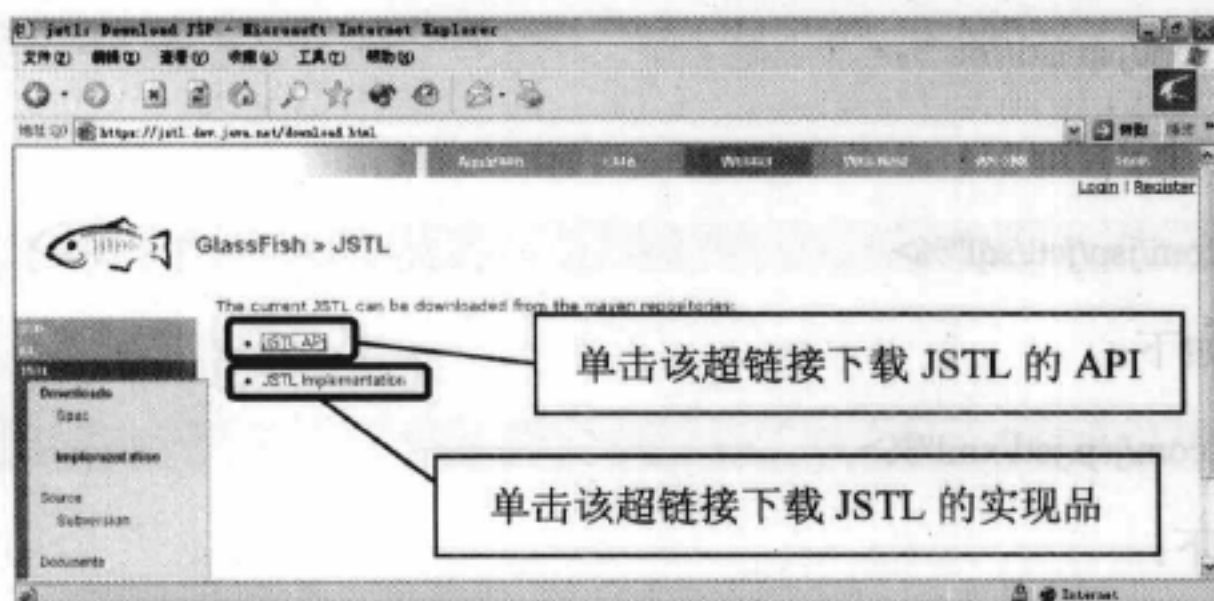


图 7.3 JSTL 下载列表页面

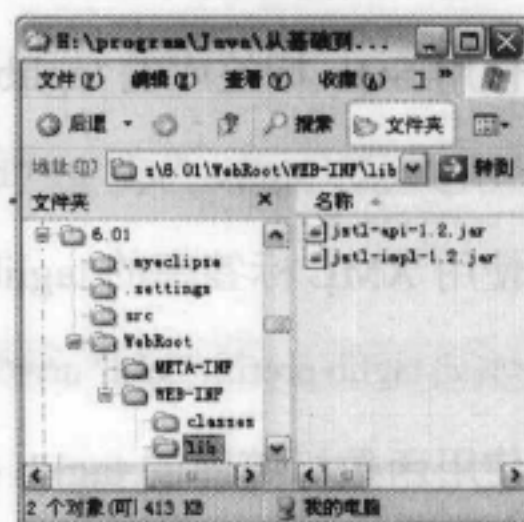


图 7.4 配置 JSTL 后的 Web 应用

说明：在 MyEclipse 7.0 中创建 Web 项目时，如果选择的 J2EE Specification Level 为 Java EE 5.0，那么 MyEclipse 将自动配置 JSTL。但是 MyEclipse 配置的 JSTL 并不包含 jstl-api-1.2.jar 和 jstl-impl-1.2.jar 两个包，而是只有一个 jstl-1.2.jar 包，这是因为在这个包中已经包含了 jstl-api-1.2.jar 和 jstl-impl-1.2.jar 两个包的内容。

7.1.2 JSTL 标签库简介

JSTL 提供了核心标签库、格式标签库、SQL 标签库、XML 标签库和函数标签库等 5 种标签库，下面进行详细介绍。

☑ 核心标签库

核心标签库主要用于完成 JSP 页面的常用功能，其中包括 JSTL 的表达式标签、条件标签、循环标签和 URL 操作共 4 种标签。

☑ 格式标签库

格式标签库提供了一个简单的国际化 (I18N) 标记，用于处理国际化相关问题，另外，格式标签库中还包含用于格式化数字和日期显示格式的标签。

☑ SQL 标签库

SQL 标签库封装了数据库访问的通用逻辑，简化了对数据库的访问。如果结合核心标签库，可以方便地获取结果集，并迭代输出结果集中的数据。

☑ XML 标签库

XML 标签库可以处理和生成 XML 标记，使用这些标记可以很方便地开发基于 XML 的 Web 应用。

☑ 函数标签库

函数标签库提供了一系列字符串操作函数，用于完成分解字符串、连接字符串、返回子串、确定字符串是否包含特定的子串等功能。

在使用这些标签之前必须在 JSP 页面的首行使用 `<%@ taglib %>` 指令定义标签库的位置和访问前缀。例如，使用核心标签库的 taglib 指令格式如下：

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

使用格式标签库的 taglib 指令格式如下：

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

使用 SQL 标签库的 taglib 指令格式如下:

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

使用 XML 标签库的 taglib 指令格式如下:

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

使用函数标签库的 taglib 指令格式如下:

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

7.2 表达式标签

 视频讲解: 光盘\TM\Video\7\表达式标签.exe

表达式标签主要包括<c:out>、<c:set>、<c:remove>、<c:catch>4 个标签,下面分别介绍其语法及应用。

7.2.1 <c:out>输出标签

<c:out>标签用于将表达式的值输出到 JSP 页面中,该标签可以替代<%=表达式%>。<c:out>标签有两种语法格式:

☒ 语法 1: 没有标签体。

```
<c:out value="expression" [escapeXml="true|false"] [default="defaultValue"]/>
```

☒ 语法 2: 有标签体。

```
<c:out value="expression" [escapeXml="true|false"]>
    defaultValue
</c:out>
```

语法 1 和语法 2 的输出结果完全相同。<c:out>标签的属性说明如表 7.1 所示。

表 7.1 <c:out>标签的属性说明

属 性	类 型	描 述	使用 EL
value	Object	用于指定将要输出的变量或表达式	可以
escapeXml	boolean	用于指定是否转换特殊字符,默认值为 true,表示转换,例如“<”转换为“<”	不可以
default	Object	用于指定当 value 属性值等于 null 时,将要显示的默认值	不可以

例 7.01 测试<c:out>标签的 escapeXml 属性及通过两种语法格式设置 default 属性时的显示结果。
(实例位置: 光盘\TM\Instances\7.01)


```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
***** 测试 escapeXml 属性 *****<br>
escapeXml 属性值为 false 时: <c:out value="<hr>" escapeXml="false"/>
escapeXml 属性值为 true 时: <c:out value="<hr>"/>
<br>
***** 测试两种语法 *****<br>
第一种语法格式: <c:out value="${user}" default="user 的值为空"/>
<br>
第二种语法格式: <c:out value="${user}">
    user 的值为空
</c:out>

```

运行结果如图 7.5 所示。



图 7.5 测试<c:out>标签的运行结果

7.2.2 <c:set>设置标签

<c:set>标签用于在指定范围（page、request、session 或 application）内定义保存某个值的变量，或为指定的对象设置属性值。使用该标签可以在页面中定义变量，而不用在 JSP 页面中嵌入打乱 HTML 排版的 Java 代码。<c:set>标签有以下 4 种语法格式。

- ☑ 语法 1：在 scope 指定的范围内将变量值存储到变量中。

```
<c:set value="value" var="name" [scope="page|request|session|application"]/>
```

- ☑ 语法 2：在 scope 指定的范围内将标签主体存储到变量中。

```

<c:set var="name" [scope="page|request|session|application"]>
    标签主体
</c:set>

```

- ☑ 语法 3：将变量值存储在 target 属性指定的目标对象的 propName 属性中。

```
<c:set value="value" target="object" property="propName"/>
```

- ☑ 语法 4：将标签主体存储到 target 属性指定的目标对象的 propName 属性中。

```

<c:set target="object" property="propName">
    标签主体
</c:set>

```

以上语法格式所涉及到的属性说明如表 7.2 所示。

表 7.2 <c:set>标签的属性说明

属 性	类 型	描 述	引用 EL
value	Object	用于指定变量值	可以
var	String	用于指定变量名	不可以
target	Object	用于指定存储变量值或者标签主体的目标对象，可以是 JavaBean 或 Map 集合对象	可以
property	String	用于指定目标对象存储数据的属性名	可以
scope	String	用于指定变量的作用域，默认值是 page	不可以

⚠ 注意：target 属性不能是直接指定的 JavaBean 或 Map，而应该是使用 EL 表达式或一个脚本表达式指定的真正对象。例如，要为 JavaBean “LinkmanForm” 的 id 属性赋值，那么 target 属性值应该是 target="\${linkman}"，而不应该是 target="linkman"。其中 linkman 为 LinkmanForm 的对象。

例 7.02 应用<c:set>标签定义不同范围的变量和为 JavaBean 的属性赋值，并通过<c:out>标签进行输出。（实例位置：光盘\TM\Instances\7.02）

(1) 编写一个名为 LinkmanForm 的 JavaBean，用于保存联系人信息。关键代码如下：

```
package com.wgh;
public class LinkmanForm {
    private int id=0;           //联系人 ID
    private String name="";     //联系人姓名
    private String tel="";      //电话
    public void setId(int id) {
        this.id = id;
    }
    public int getId() {
        return id;
    }
    ...
    //此处省略了其他属性对应的 getXXX()和 setXXX()方法
}
```

(2) 编写 index.jsp 页面，在该页面中首先应用<c:set>标签定义两个不同范围的变量，并应用 EL 表达式输出这两个变量，然后再为 JavaBean 设置属性值，并应用<c:out>标签输出 JavaBean 的属性。关键代码如下：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:useBean class="com.wgh.LinkmanForm" id="linkman"/>
应用语法 1 定义一个 session 范围内的变量 user，值为 mrsoft <br>
    <c:set var="user" value="mrsoft" scope="session"/>
    输出变量 user 的值为：${sessionScope.user}
    } 语法 1 的应用
<br>
应用语法 2 定义一个 request 范围内的变量 money，值为 12.5*6 的结果 <br>
    <c:set var="money" scope="request">
    ${12.5*6}
    </c:set>
    输出变量 money 的值为：${requestScope.money}
    } 语法 2 的应用
```



```
<br>
```

应用语法 3 为 JavaBean “LinkmanForm” 设置各属性并应用<out>标签输出各属性值


```
<c:set value="1" target="{linkman}" property="id"/>
<c:set value="wgh" target="{linkman}" property="name"/>
id 属性值为:<out value="{linkman.id}"/> <br>
name 属性值为:<out value="{linkman.name}"/>
```

} 语法 3 的应用

```
<br>
```

应用语法 4 为 JavaBean “LinkmanForm” 设置各属性并应用<out>标签输出各属性值


```
<c:set target="{linkman}" property="tel"> 84978981 </c:set>
tel 属性值为:<out value="{linkman.tel}"/>
```

} 语法 4 的应用

运行结果如图 7.6 所示。

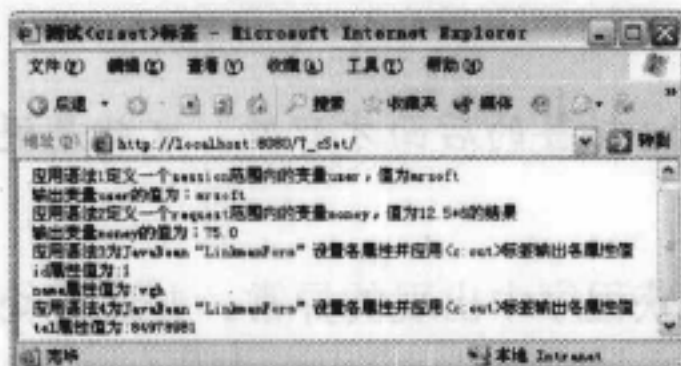


图 7.6 测试<c:set>标签的运行结果

7.2.3 <c:remove>移除标签

<c:remove>标签可以从指定的 JSP 范围内移除指定的变量，其语法格式如下。

```
<c:remove var="name" [scope="page|request|session|application"]/>
```

- ☒ var 属性：用于指定要移除的变量名称。
- ☒ scope 属性：用于指定变量的存在范围，可选值有 page、request、session、application。默认值是 page。

例 7.03 应用<c:set>标签定义一个 page 范围内的变量，然后通过<c:out>标签输出该变量，再应用<c:remove>标签移除该变量，最后通过<c:out>标签输出该变量。（实例位置：光盘\TM\Instances\7.03）

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:set var="softName" value="明日科技编程词典" scope="page"/>
移除前输出变量 softName 的值: <out value="{pageScope.softName}" default="softName 的值为空"/>
<br>
<c:remove var="softName" scope="page"/>
移除后输出变量 softName 的值: <out value="{pageScope.softName}" default="softName 的值为空"/>
```

运行结果如图 7.7 所示。

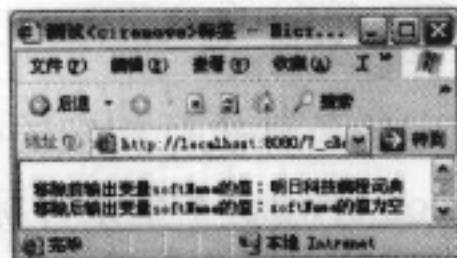


图 7.7 测试<c:remove>标签的运行结果

7.2.4 <c:catch>捕获异常标签

<c:catch>标签与 Java 程序中的 try...catch 语句类似,用于捕获程序中出现的异常;此外,它还能将异常信息保存在变量中。<c:catch>标签的语法格式如下:

```
<c:catch [var="exception"]>
...           //可能存在异常的代码
</c:catch>
```

在上面的语法中, var 属性可以指定存储异常信息的变量。这是一个可选项,如果不需要保存异常信息,则可以省略该属性。

注意: var 属性值只有在<c:catch>标签的后面才有效,也就是说,在<c:catch>标签体中无法使用有关异常的任何信息。

例 7.04 应用<c:catch>标签捕获程序中出现的异常,并通过<c:out>标签输出该异常信息。(实例位置:光盘\TM\Instances\7.04)

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:catch var="exception">
<%
int number=Integer.parseInt(request.getParameter("number"));
out.println("合计金额为: "+521*number);
%>
</c:catch>
抛出的异常信息: <c:out value="${exception}"/>
```

运行程序,页面中将显示如图 7.8 所示的异常信息;在 IE 地址栏中,将 URL 地址修改为 http://localhost:8080/7_cCatch/index.jsp?number=10,将显示“合计金额为: 5210”。



图 7.8 抛出的异常信息


7.3 条件标签

视频讲解: 光盘\TM\Video\7\条件标签.exe

在程序中,使用条件标签可以根据指定的条件执行相应的代码来产生不同的运行结果。在 JSTL 中提供了<c:if>、<c:choose>、<c:when>和<c:otherwise>4 个条件标签,使用这些条件标签可以处理程序中任何可能发生的事情。

7.3.1 <c:if>标签

<c:if>标签可以根据不同的条件处理不同的业务，即执行不同的程序代码。<c:if>标签和 Java 中的 if 语句的功能类似，但是它没有对应的 else 标签。<c:if>标签有两种语法格式：

 说明：虽然<c:if>标签没有对应的 else 标签，但是利用 JSTL 提供的<c:choose>、<c:when>和<c:otherwise>标签也可以实现 if else 的功能。

- ☑ 语法 1：可判断条件表达式，并将条件的判断结果保存在 var 属性指定的变量中，而这个变量存在于 scope 属性所指定的范围内。

```
<c:if test="condition" var="name" [scope=page|request|session|application]/>
```

- ☑ 语法 2：不但可以将 test 属性的判断结果保存在指定范围的变量中，还可以根据条件的判断结果执行标签主体。标签主体可以是 JSP 页面能够使用的任何元素，如 HTML 标记、Java 代码或者嵌入的其他 JSP 标签。

```
<c:if test="condition" var="name" [scope=page|request|session|application]>
    标签主体
</c:if>
```

以上语法格式所涉及到的属性说明如表 7.3 所示。

表 7.3 <c:if>标签的属性说明

属 性	类 型	描 述	引用 EL
test	boolean	必选属性，用于指定条件表达式	可以
var	String	可选属性，用于指定变量名。这个属性会指定 test 属性的判断结果将存放在哪个变量中，如果该变量不存在就创建它	不可以
scope	String	存储范围，该属性用于指定 var 属性所指定的变量的存在范围	不可以

例 7.05 应用<c:if>标签根据是否发表过评论，决定是否显示发表评论表单及表单元素。（实例位置：光盘\TM\Instances\7.05）

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
未来的世界是：方向比努力重要，能力比知识重要，健康比成绩重要，生活比文凭重要，情商比智商重要！
<c:if test="${empty param.comment}">
    <form name="form1" method="post" action="">
        评论：
        <textarea name="comment" cols="30" rows="4"></textarea>
        <br>
        <br>
        <input type="submit" name="Submit" value="发表评论">
    </form>
</c:if>
```

运行程序，将显示如图 7.9 所示的页面，在“评论”文本框中输入评论内容后，单击“发表评论”

按钮，页面中将不显示发表评论表单，如图 7.10 所示。

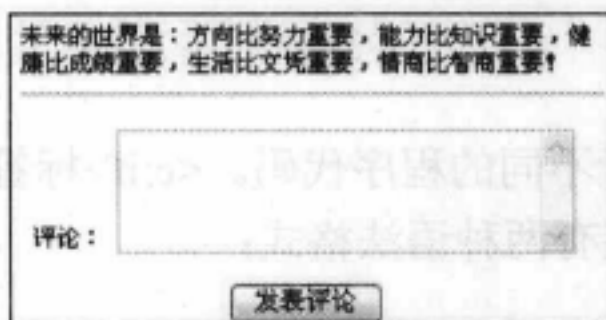


图 7.9 显示评论表单

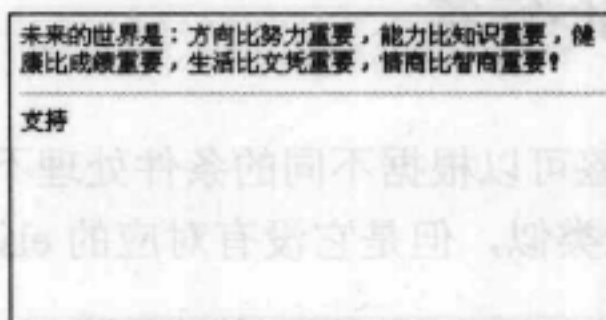


图 7.10 不显示发表评论表单

7.3.2 <c:choose>、<c:when>和<c:otherwise>标签

<c:choose>标签可以根据不同的条件完成指定的业务逻辑，如果没有符合的条件，则会执行默认条件的业务逻辑。需要注意的是，<c:choose>标签只能作为<c:when>和<c:otherwise>标签的父标签，可以在其中嵌套这两个标签完成条件选择逻辑。<c:choose>、<c:when>和<c:otherwise>标签的语法格式如下：

```
<c:choose>
  <c:when test="condition">
    业务逻辑
  </c:when>
  ... <!--多个<c:when>标签-->
  <c:otherwise>
    业务逻辑
  </c:otherwise>
</c:choose>
```

在一个<c:choose>标签中只能有一个该标签，并且应该在全部<c:when>标签的后面

在上面的语法中，<c:when>标签可以根据不同的条件执行相应的业务逻辑，<c:otherwise>标签用于指定默认条件处理逻辑，即如果没有任何一个结果满足<c:when>标签指定的条件，将会执行<c:otherwise>标签主体中定义的逻辑代码。

在一个<c:choose>标签中可以存在多个<c:when>标签来处理不同条件的业务逻辑。其中，<c:when>标签的 test 属性是必须定义的属性，用于指定条件表达式；它可以引用 EL 表达式。

说明：在<c:choose>标签中，如果发现一个 test 属性值为 true 的<c:when>标签，则其后面的<c:when>标签将不再处理。

注意：在<c:choose>标签中，必须有一个<c:when>标签，但是<c:otherwise>标签是可选的。如果省略了<c:otherwise>标签，当所有的<c:when>标签都不满足条件时，将不会处理<c:choose>标签的标签体。

例 7.06 应用<c:choose>、<c:when>和<c:otherwise>标签实现 if else 功能，即如果 session 变量为空，则显示用户登录表单，要求用户登录；当用户登录后（表示 session 变量不为空），将显示当前登录的用户。（实例位置：光盘\TM\Instances\7.06）

(1) 编写 index.jsp 页面。在该页面中应用<c:choose>、<c:when>和<c:otherwise>标签，根据保存用户名的 session 变量是否为空确定页面中显示的内容。关键代码如下：


```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:choose>
  <c:when test="{empty sessionScope.user}">
    <form name="form1" method="post" action="deal.jsp">
      用户名:
      <input name="user" type="text" id="user">
      &nbsp;
      <input type="submit" name="Submit" value="登录">
    </form>
  </c:when>
  <c:otherwise>
    欢迎您! ${sessionScope.user} [<a href="logout.jsp">退出</a>]
  </c:otherwise>
</c:choose>

```

(2) 编写 deal.jsp 页面。在该页面中, 应用<c:set>标签将提交的用户名保存到 session 范围的变量中, 并将页面重定向到 index.jsp 页面中。deal.jsp 页面的具体代码如下:

```

<%@ page language="java" pageEncoding="GBK"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%request.setCharacterEncoding("GBK");%>
<c:set var="user" scope="session" value="{param.user}"/>
<%response.sendRedirect("index.jsp");%>

```

(3) 编写 logout.jsp 页面。在该页面中, 清空全部 session 变量, 并将页面重定向到 index.jsp 页面中。具体代码如下:

```

<%@ page language="java" pageEncoding="GBK"%>
<%
session.invalidate();
response.sendRedirect("index.jsp");
%>

```

运行程序, 将显示如图 7.11 所示的用户登录页面, 在“用户名”文本框中输入用户名后, 单击“登录”按钮, 在页面中将显示当前登录的用户及“退出”超链接, 如图 7.12 所示; 单击“退出”超链接, 将返回到图 7.11 所示的用户登录页面。

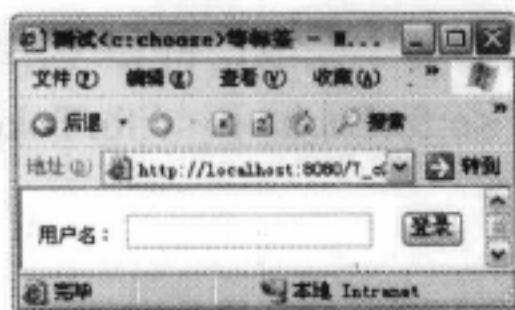


图 7.11 显示用户登录页面

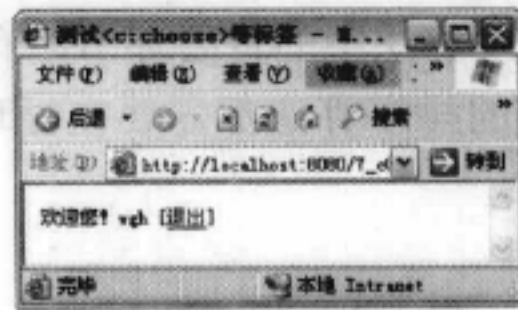


图 7.12 显示登录用户

例 7.07 应用<c:choose>、<c:when>和<c:otherwise>标签在页面中显示分时间候。(实例位置: 光盘\TM\Instances\7.07)

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<jsp:useBean id="now" class="java.util.Date"/>

```

```

<c:choose>
  <c:when test="{now.hours}>=0 && now.hours<5}">
    凌晨好!
  </c:when>
  <c:when test="{now.hours}>=5 && now.hours<8}">
    早上好!
  </c:when>
  <c:when test="{now.hours}>=8 && now.hours<11}">
    上午好!
  </c:when>
  <c:when test="{now.hours}>=11 && now.hours<13}">
    中午好!
  </c:when>
  <c:when test="{now.hours}>=13 && now.hours<17}">
    下午好!
  </c:when>
  <c:otherwise>
    晚上好!
  </c:otherwise>
</c:choose>
现在是: ${now.hours}时${now.minutes}分

```

运行程序, 页面中将显示如图 7.13 所示的提示信息。

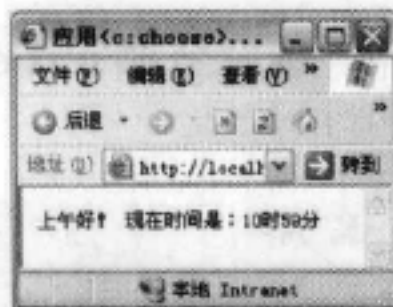


图 7.13 分时间候

7.4 循环标签

视频讲解: 光盘\TM\Video\7\循环标签.exe

循环是程序算法中的重要环节, 有很多著名的算法都需要在循环中完成, 例如递归算法、查询算法和几乎所有的排序算法都需要在循环中完成。JSTL 中提供了<c:forEach>和<c:forTokens>两个循环标签。

7.4.1 <c:forEach>标签

<c:forEach>标签可以根据循环条件遍历数组和集合类中的所有或部分数据。例如, 在使用 Hibernate 技术访问数据库时, 返回的均为数组、java.util.List 和 java.util.Map 对象, 它们封装了从数据库中查询出的数据, 而这些数据都是 JSP 页面所需要的。如果在 JSP 页面中使用 Java 代码来循环遍历所有数据, 会使页面非常混乱, 不易分析和维护; 而使用 JSTL 的<c:forEach>标签循环显示这些数据不但可以解决

JSP 页面混乱的问题，还可提高代码的可维护性。

<c:forEach>标签有以下两种语法格式。

☑ 语法 1：数字索引迭代。

```
<c:forEach begin="start" end="finish" [var="name"] [varStatus="statusName"]>
    [step="step"]
    标签主体
</c:forEach>
```

在该语法中，begin 和 end 属性是必选的属性，其他属性均为可选属性。

☑ 语法 2：集合成员迭代。

```
<c:forEach items="data" [var="name"] [begin="start"] [end="finish"] [step="step"]
    [varStatus="statusName"]>
    标签主体
</c:forEach>
```

在该语法中，items 属性是必选属性，通常使用 EL 表达式指定，其他属性均为可选属性。<c:forEach> 标签的各属性的详细介绍如表 7.4 所示。

表 7.4 <c:forEach>标签的属性

属 性	类 型	描 述	引用 EL
items	数组、集合类、字符串和枚举类型	被循环遍历的对象，多用于数组与集合类	可以
var	String	循环体的变量，用于存储 items 指定的对象的成员	不可以
begin	int	循环的起始位置，如果没有指定，则从集合的第一个值开始迭代	可以
end	int	循环的终止位置，如果没有指定，则一直迭代到集合的最后一位	可以
step	int	循环的步长	可以
varStatus	String	循环的状态信息，该属性还有表 7.5 所示的 4 个状态属性	不可以

表 7.5 varStatus 属性的状态属性

属 性	类 型	描 述
index	int	当前循环的索引值，从 0 开始
count	int	当前循环的循环计数，从 1 开始
first	boolean	是否为第一次循环
last	boolean	是否为最后一次循环

✎ 技巧：如果要在循环的过程中获取循环计数，可以应用 varStatus 属性的状态属性 count 获得。

例 7.08 应用<c:forEach>标签遍历 List 集合中第 2 个元素以后的元素（包括第 2 个元素）。（实例位置：光盘\TM\Instances\7.08）

```
<%@ page language="java" pageEncoding="GBK" import="java.util.*"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%
```

```

List list=new ArrayList();
list.add("明日科技");
list.add("编程词典");
list.add("www.bccd.com");
request.setAttribute("list",list);           //将 List 集合保存到 request 对象中
%>
利用<c:forEach>标签遍历 List 集合的结果如下: <br>
<c:forEach items="${requestScope.list}" var="keyword" varStatus="id" begin="1">
    ${id.index }&nbsp;${keyword}<br>
</c:forEach>

```

⚠ 注意: 在应用<c:forEach>标签时, var 属性指定的变量只在循环体内有效, 这一点与 Java 语言的 for 循环语句中的循环变量类似。

程序运行结果如图 7.14 所示。



图 7.14 应用<c:forEach>标签遍历 List 集合

7.4.2 <c:forTokens>标签

<c:forTokens>标签可以用指定的分隔符将一个字符串分割开, 根据分割的数量确定循环的次数。

<c:forTokens>标签的语法格式如下:

```

<c:forTokens items="String" delims="char" [var="name"] [begin="start"] [end="end"] [step="len"]
[varStatus="statusName"]>
    标签主体
</c:forTokens>

```

<c:forTokens>标签的属性说明如表 7.6 所示。

表 7.6 <c:forTokens>标签的属性

属 性	类 型	描 述	引用 EL
items	String	被循环遍历的对象, 多用于数组与集合类	可以
delims	String	字符串的分割字符, 可以同时有多个分隔字符	不可以
var	String	变量名称	不可以
begin	int	循环的起始位置	可以
end	int	循环的终止位置	可以
step	int	循环的步长	可以
varStatus	String	循环的状态变量	不可以

例 7.09 应用<c: forTokens >标签分割字符串并显示。(实例位置: 光盘\TM\Instances\7.09)


```

<%@ page language="java" pageEncoding="GBK"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:set var="sourceStr" value="编程词典软件涵盖技术、函数、控件、实例、项目、方案、界面等所有开发内容，以及所有实例程序、实用工具等内容，是程序开发人员高效编程必备的软件。"/>
原字符串: <c:out value="${sourceStr}"/>
<br>分割后的字符串: <br>
<c:forTokens items="${sourceStr}" delims="，、。" var="item">
    ${item}<br>
</c:forTokens>

```

程序运行结果如图 7.15 所示。

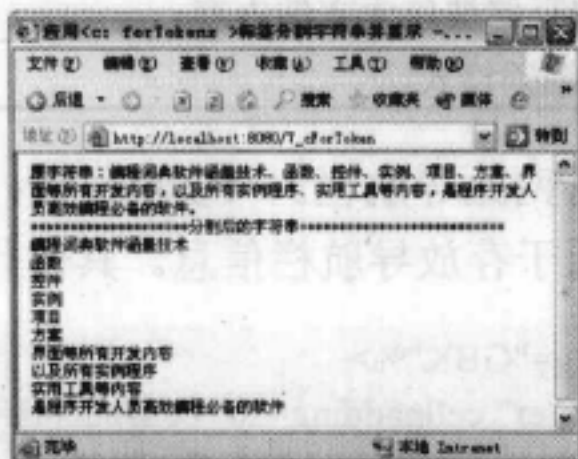



图 7.15 应用<c: forTokens>标签分割字符串并显示

7.5 URL 操作标签

 视频讲解: 光盘\TM\Video\7\ URL 操作标签.exe

URL 操作标签是指与文件导入、重定向、URL 地址生成以及参数传递相关的标签。JSTL 中提供的与 URL 相关的标签有<c:import>、<c:redirect>、<c:url>和<c:param>4 个，下面进行详细介绍。

7.5.1 <c:import>文件导入标签

<c:import>标签可以导入站内或其他网站的静态和动态文件到 Web 页面中，例如使用<c:import>标签导入其他网站的天气信息到自己的网页中；而<jsp:include>只能导入站内资源，相比之下，<c:import>的灵活性要高很多。

<c:import>标签的语法格式如下：

☒ 语法 1：

```

<c:import url="url" [context="context"] [var="name"]
    [scope="page|request|session|application"] [charEncoding="encoding"]
    标签主体
</c:import>

```

☒ 语法 2：

```

<c:import url="url" varReader="name" [context="context"]
    [charEncoding="encoding"]

```

上面语法中涉及到的属性说明如表 7.7 所示。

表 7.7 <c:import> 标签的属性说明

属 性	类 型	描 述	引用 EL
url	String	被导入的文件资源的 URL 路径	可以
context	String	上下文路径, 用于访问同一个服务器的其他 Web 工程, 其值必须以 “/” 开头; 如果指定了该属性, 那么 url 属性值也必须以 “/” 开头	可以
var	String	变量名称, 将获取的资源存储在变量中	不可以
scope	String	变量的存在范围	不可以
varReader	String	以 Reader 类型存储被包含文件内容	不可以
charEncoding	String	被导入文件的编码格式	可以

例 7.10 应用<c:import>标签包含网站导航栏。(实例位置: 光盘\TM\Instances\7.10)

(1) 编写 navigation.jsp 文件, 用于存放导航栏信息。具体代码如下:

```
<%@ page language="java" pageEncoding="GBK"%>
<table width="778" border="0" align="center" cellpadding="0" cellspacing="0">
  <tr><td height="112" valign="top" background="images/top_bg.jpg">&nbsp;</td></tr>
  <tr>
    <td height="39" align="right" valign="top" background="images/navigate_bg.jpg" bgcolor="#EEEEEE">
      <table width="100%" height="30" border="0" cellpadding="0" cellspacing="0">
        <tr>
          <td width="28%">&nbsp;</td>
          <td width="70%" align="center" valign="middle" class="word_grey"><a href="#">首页</a> |
            <a href="#">名片夹管理</a> | <a href="#">信息库管理</a> | <a href="#">收发短信</a>
            | <a href="#">邮件群发</a> | <a href="#">系统设置</a> | <a href="#">退出系统</a></td>
          <td width="2%">&nbsp;</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
```

navigation.jsp 文件的设计效果如图 7.16 所示。

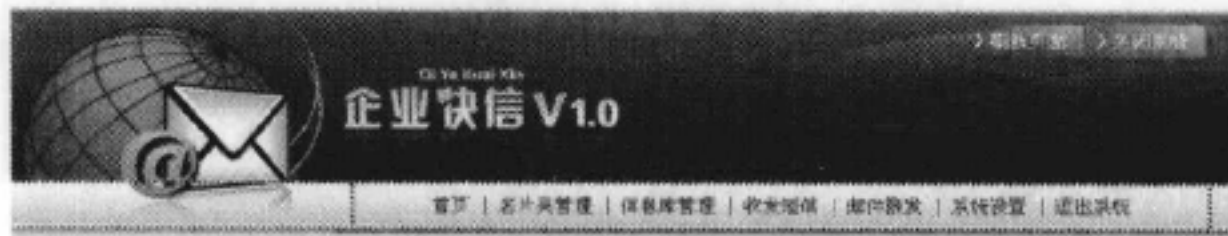


图 7.16 导航栏文件的设计效果

(2) 编写 index.jsp 页面, 在该页面中应用<c:import>标签导入步骤 (1) 中编写的导航栏文件 navigation.jsp。关键代码如下:

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

程序运行结果如图 7.17 所示。