# JSTL标签库

## JSTL标签库概述

### JSTL标签库概述

JSTL全称为JavaServer Pages Standard Tag Library

由JCP（Java Community Process）指定标准

是提供给 Java Web 开发人员一个标准通用的标签函数库

可以和 EL 配合来取代传统直接在页面上嵌入 Java 程序（Scripting）的做法，以提高程序可读性、维护性和方便性

在javaee4.0需要导入JSTL相关的jar包，在javaee5.0开始，默认已经包含了此jar包。还要需要用<%@taglib%>指令引入标签库

## JSTL标签库子库

### JSTL标签库子库

核心标签库 (core) --- c

国际化标签 fmt

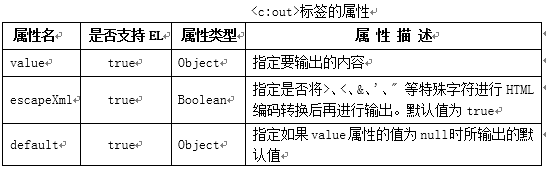
数据库标签 sql --Servlet

XML标签 xml

JSTL函数(EL函数) el

## JSTL核心标签库

### <c:out>



输出固定值 <c:out value="Hello c out~"/> 输出变量 <c:out value="${m}"/> 转义html,默认为转义，可以设置escapeXml属性为false，禁止转义 <c:out value="<a href='#'>link</a>" escapeXml="false"/> 允许输出默认值，如果给定值不存在，输出默认值 <c:out value="${city}" default="北京"/>

### <c:set>

如图-7所示：

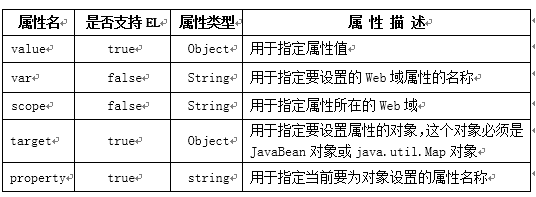


图-7

标签用于把某一个对象存在指定的域范围内，或者设置Web域中的java.util.Map类型的属性对象或JavaBean类型的属性对象的 属性。

在指定域中增加、修改属性 <c:set var="name" value="value" scope="page"/> 设置一个保存在四个范围的java对象的属性值，cset不能设置对象进域，但是可以更改域中对象的属性 <c:set targe="${person}" property="name" value="lisi"/>

### <c:remove>

标签用于删除各种Web域中的属性

<c:remove var="varName" [scope="{page|request|session|application}"] />

### <c:catch>

标签用于捕获嵌套在标签体中的内容抛出的异常，其语法格式如下：

<c:catch var="e">

<%=1/0 %>

</c:catch>

${e.message}

### <c:if>

如图-8所示：

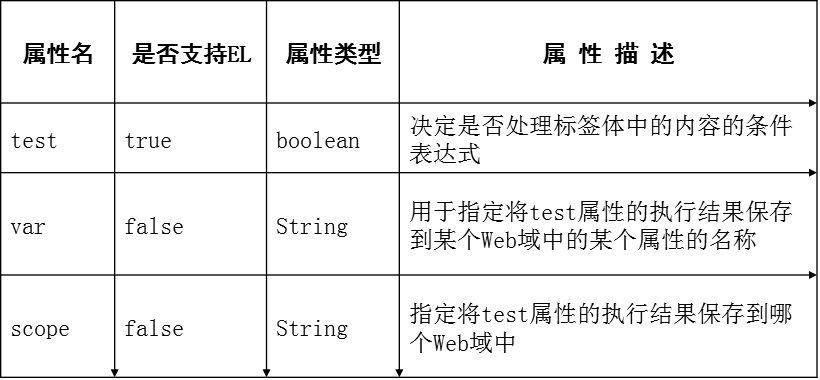


图-8

此标签可以构造简单的“if-then”结构的条件表达式 。

### <c:choose>

标签用于指定多个条件选择的组合边界，它必须与<c:when>和<c:otherwise>标签一起使用。使用<c:choose>，<c:when>和<c:otherwise>三个标签，可以构造类似 “if-else if-else” 的复杂条件判断结构。

<c:choose> <c:when test="${weekday==1}"> 星期一 </c:when> <c:when test="${weekday==1}"> 星期二 </c:when> <c:when test="${weekday==1}"> 星期三 </c:when> <c:otherwise> 输入无效 </c:otherwise> </c:choose>

### <c:forEach>

如图-9所示：

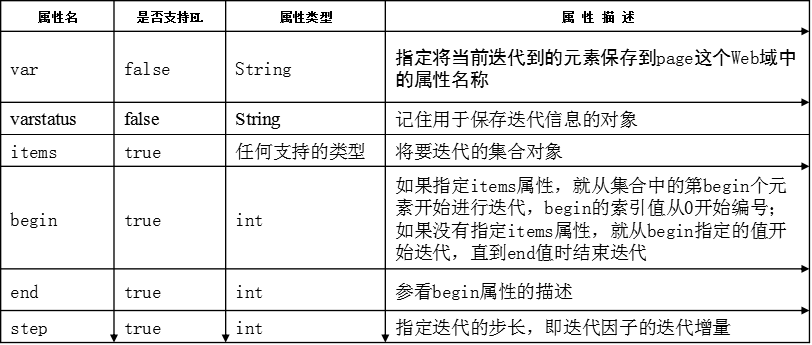


图-9

varStatus属性，如图-10所示：



图-10

实验：遍历10到100的偶数，如果数字所在的位置是3的倍数，显示成红色

### <c:forTokens>

用来浏览一字符串中所有的成员，其成员是由定义符号所分隔的

## URL相关的标签概述

### <c:param>

在JSP页面进行URL的相关操作时，经常要在URL地址后面附加一些参数。<c:param>标签可以嵌套在<c:import>、<c:url>或<c:redirect>标签内，为这些标签所使用的URL地址附加参数。

### <c:import>

标签,实现include操作，如图-11所示：



图-11

### <c:url>(了解)

如图-12所示：

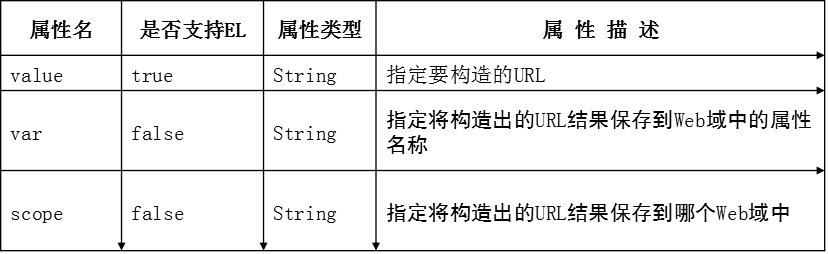


图-12

标签用于在JSP页面中构造一个URL地址，其主要目的是实现URL重写。URL重写就是将会话标识号以参数形式附加在URL地址后面

### <c:redirect>

此标签用于实现请求重定向。如图-13所示：

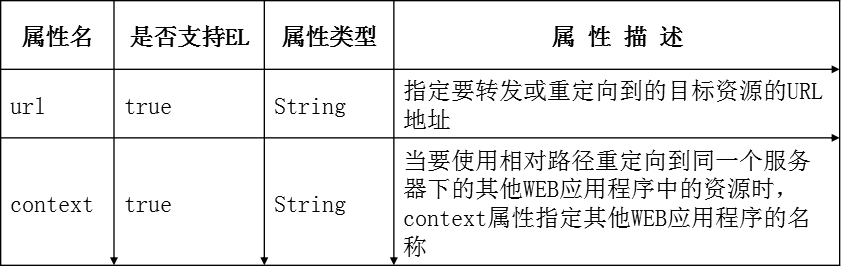


图-13

# 自定义标签技术

## 自定义标签概述

### 自定义标签概述

虽然有第三方组织提供了很多标签，但是这些都是一些通用标签，开发中常常需要根据业务需求使用jsp页面，这个时候通用的标签就不够用了，我们需要自己去开发标签库。

## 开发自定义标签

### 开发步骤

想要开发自定义标签需要两个步骤：

写一个类实现SimpleTag接口

在tld文件中描述该标签，在jsp页面中通过<%@taglib %>引入该标签库使用标签

## 写一个类实现SimpleTag接口

### 写一个类实现SimpleTag接口

SimpleTag接口的继承结构如图-14所示：



图-14

我们写一个类实现SimpleTag接口，发现其中包含如下方法：

public void doTag() throws JspException, IOException {

}

public void setParent(JspTag parent) {

}

public JspTag getParent() {

return null;

}

public void setJspContext(JspContext pc) {

}

public void setJspBody(JspFragment jspBody) {

}

### SimpleTag接口方法详解

当jsp页面执行到自定义标签时，首先创建出自定义标签处理类对象，创建出来后会立即调用setJspContext方法将当前页面的PageContext传入。如果当前标签具有标签体，则将标签的标签体封装到JspFragment对象中调用setJspBody方法，如果没有标签体此方法不执行。如果当前自定义标签有父标签，则调用setParent方法将父标签传入。最后调用doTag方法执行自定义标签的处理逻辑。

所以我们在开发自定义标签时，将核心代码写入doTag方法即可。

更多的时候我们不会直接实现SimpleTag接口而是继承SimpleTagSupport。

SimpleTagSupport实现了SimpleTag接口，对立面的方法都做了实现，实现了setJspContext方法，将JspContext保存成了类的成员，并提供了getJspContext()方法获取该对象。实现了setJspBody(JspFragment)方法，将JspFragment保存成了类的成员，并提供了getJspBody()方法获取该对象。实现了doTag方法，做了空实现，我们作为实现类，只需要覆盖这个方法，在其中编写标签处理代码。

## 实现自定义标签的基本功能

### 控制是否执行标签体

控制标签体不执行:什么都不做标签体默认就不执行

控制标签体执行:获取封装了标签体的JSPFragment对象,执行invoke方法,输出到out输出流中即可

JspFragment fragment = this.getJspBody();

fragment.invoke(this.getJspContext().getOut());

fragment.invoke(null);

### 控制是否执行标签之后的内容

控制标签之后的内容执行:什么都不做,就执行

控制标签之后的内容不执行:抛出一个特殊的异常SkipPageException就可以阻止标签之后的内容执行

throw new SkipPageException();

### 控制标签体重复执行

JspFragment fragment = this.getJspBody();

for(int i = 0;i<5 ;i++){

fragment.invoke(null);

}

### 修改标签体输出

StringWriter writer = new StringWriter();

JspFragment fragment = this.getJspBody();

fragment.invoke(writer);

String str = writer.toString();

str = str.toUpperCase();

this.getJspContext().getOut().write(str);

## 在tld文件中描述该标签

### tld文件详解

tld是专门用来描述自定义标签的文件，可以在其中描述自定义标签。

tld文件必须放置在WEB-INF目录除classes和lib目录外的其他目录下。

其格式如下，如图-15所示：

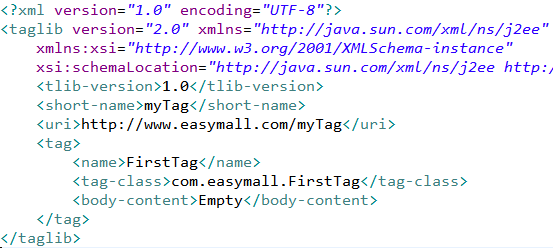
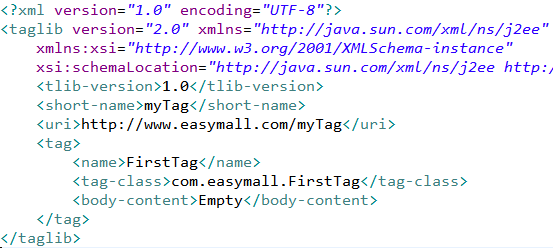


图-15

其中<name>是该自定义标签的名字

<tag-class>是该自定义标签处理类的全路径名

<body-content>是标签体类型，可取的值为Empty表示不能含有标签体。scriptless表示可以包含标签体但是不能包含java源代码。JSP表示可以包含任意内容的标签体。

## 在jsp页面中引入自定义标签使用

### 在jsp页面中引入自定义标签

在jsp页面中使用<%@taglib %>引入自定义标签。

<%@ taglib uri="http://www.easymall.com/myTag" prefix="myTag" %>

## 开发带有属性的自定义标签

### 开发带有属性的自定义标签

想要开发带有属性的标签，需要做如下两件事：

在自定义标签处理类中定义javabean属性并提供setXxx()方法，如图-16所示：

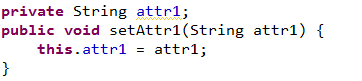


图-16

在tld文件中描述该属性，如图-17所示：

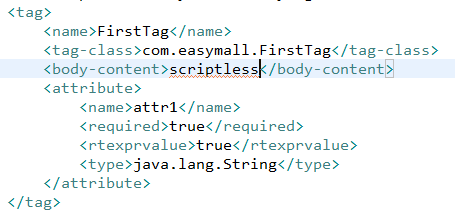


图-17

其中,描述属性的标签如图-18所示：

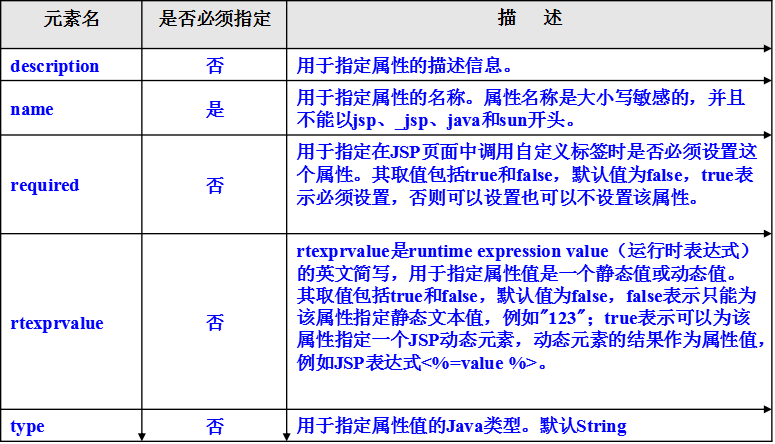


图-18

# 用JSTL+EL改造注册、登录页面

## login.jsp

### 改造login.jsp

之前的代码十分混乱，如图-19所示：

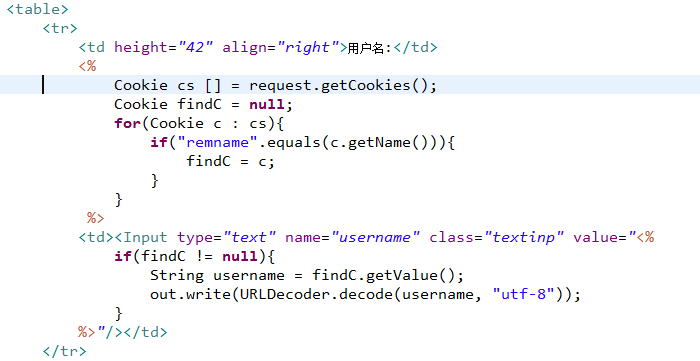


图-19

用el+jstl改造，如图-20所示：

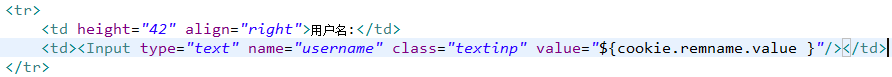


图-20

### 自定义标签实现URL解码

经过测试发现显示的用户名是经过URL编码的，而JSTL并没有提供现成的URL解码用的标签,而我们不希望在jsp页面中写java代码，对于这种非常常见的功能，我们可以开发一个自定义标签来进行处理。代码如图-21所示：

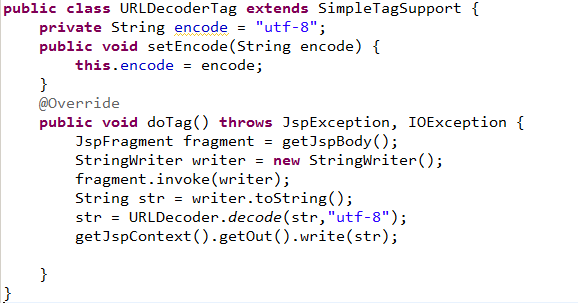


图-21

在tld文件中描述该标签，如图-22所示：



图-22

在regist.jsp中引入此tld文件，通过该自定义标签进行URL解码，如图-23所示：



图-23

经过测试，可以正确的显示中文，如图-24所示：



图-24

## index.jsp

### 改造index.jsp

如图-25所示：



图-25

# 开发模式

## Javaee开发模式

### 遇到的问题

回顾我们目前写的EasyMall代码，我们发现一个非常简单的功能都需要非常多的代码，代码混杂在一起，造成Servlet中代码庞杂不易维护。可以想象，如果想要用这种开发方式开发一个完整的web应用，一定是十分困难的。如图-26所示：



图-26

### 开发模式的发展

既然这种开发模式已经进行不下去了，那么应该如何组织整个工程的架构呢？在整个web开发发展的历史中，产生了多种开发模式。如图-27所示：

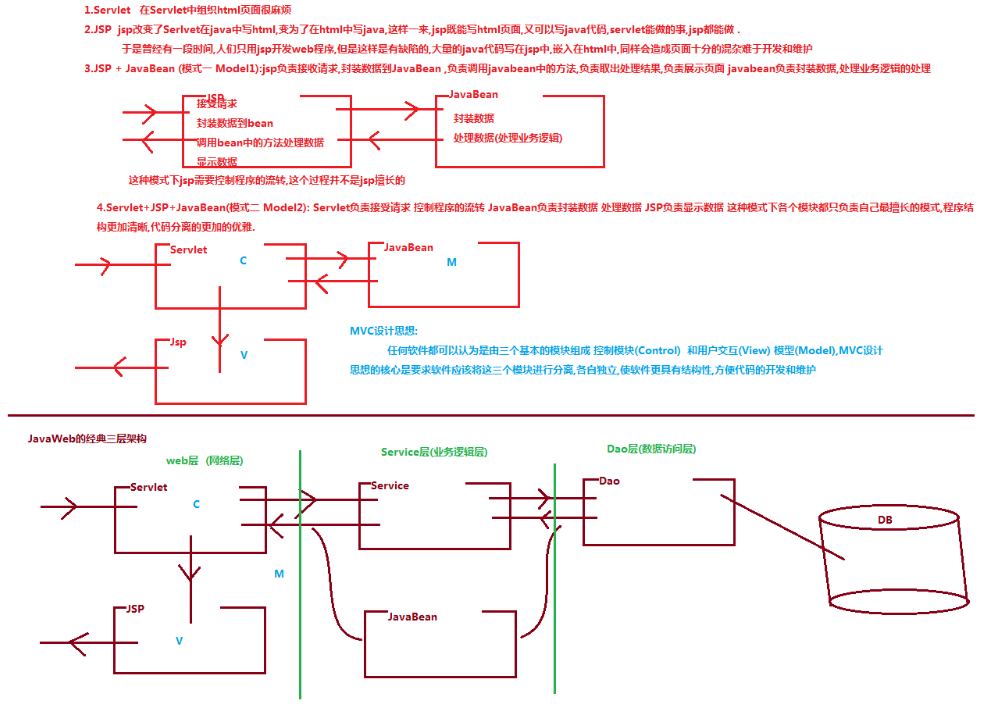


图-27

现在的javaweb应用基本都使用的是javaee的经典三层架构。

我们就使用这种架构改造EasyMall项目。

# JAVAEE经典三层架构改造EasyMall

## 改造EasyMall

### 分包

我们重构EasyMall项目。首先进行分包。创建UserService、UserDao。创建User javabean。如图-28所示：

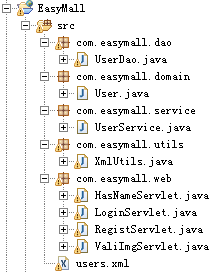


图-28

### 实现Dao

将操作数据库相关的代码写到UserDao中。如图-29所示：

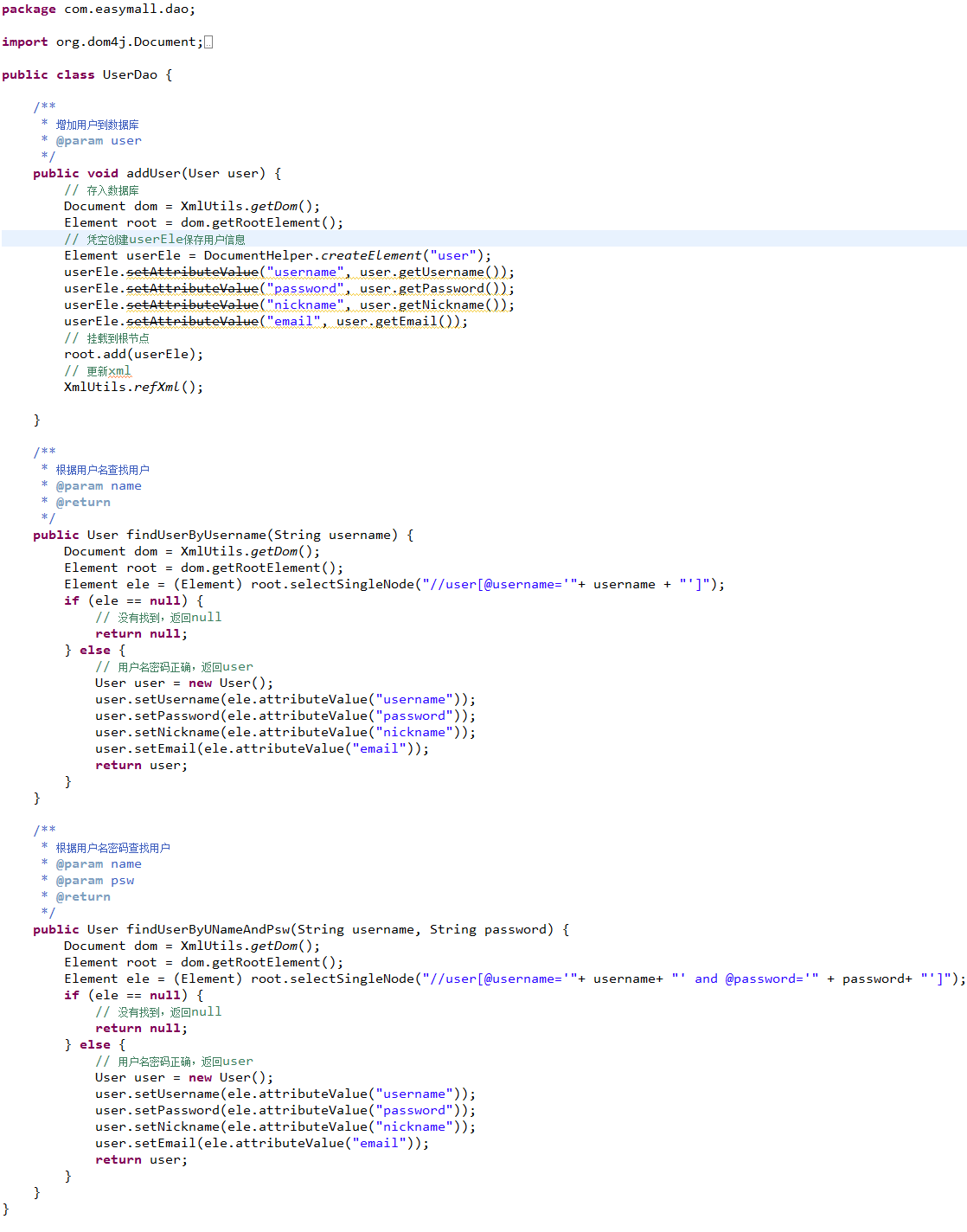


图-29

### 实现Service

将业务逻辑代码写到UserService中。

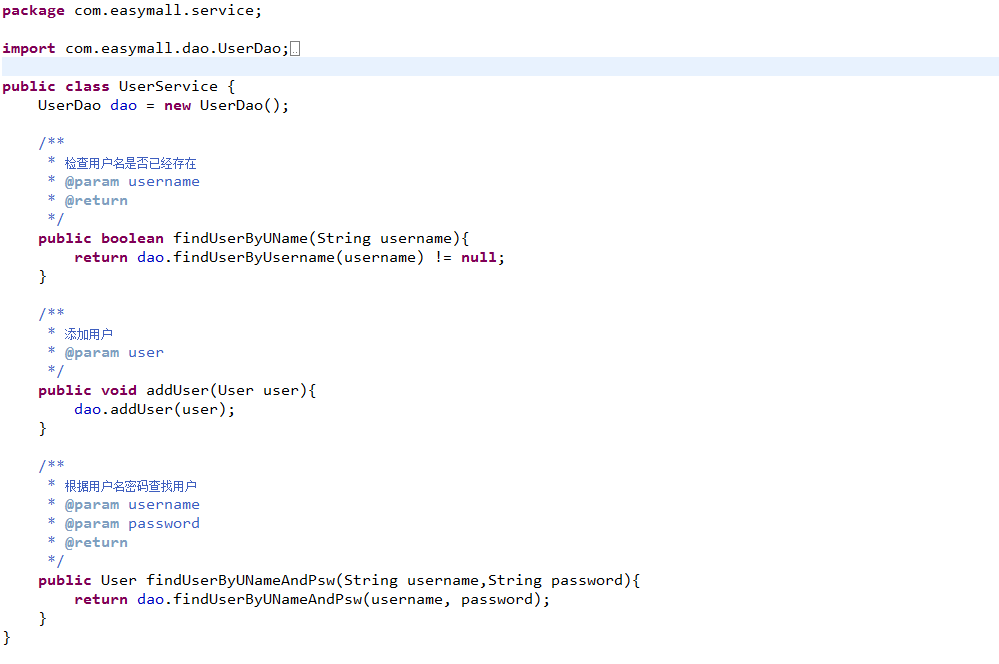


图-30

### 实现web层

web层只存放视图相关代码。如图-31所示：



图-31