# JSP

## jsp概述

sun公司提供的动态web资源开发技术之一。

看起来非常像HTML，但是其中可以编写java代码，是一种动态web资源。它的出现是为了解决Servlet不适合向外输出页面内容的问题。JSP非常适合编写HTML页面，其中又可以写java代码，所以非常完好的解决了输出页面，和书写逻辑的问题。

如果说Serlvet是在java代码中嵌入HTML，那么JSP计是在HTML中嵌入java代码。

jsp会在第一次访问时被翻译成Serlvet，对jsp的访问其实是翻译过来的Servlet在向外进行输出。

## JSP基本语法

### 模版元素

直接书写在JSP中的HTML内容，看起来就像写HTML一样的方便，但是最终会在被翻译成Servlet的过程中 out.write()直接输出

### 脚本表达式

<%= 表达式%> 接受的是一段java表达式，在JSP翻译到Servlet的过程中，将会计算表达式的值，利用out.write()输出出去

### 脚本片段

<% %>直接可以在脚本片段中书写java源代码，其中的源代码将会直接拷贝到翻译过来的servlet中的响应位置上。

### JSP声明

<%! %>在其中可以写java代码，其中的源代码会被拷贝到servlet中的service方法之外，可以利用它来为servlet增加成员方法、成员变量、静态代码块

### JSP注释

<%-- --%>被jsp注释包围起来的内容将不会被服务器翻译到servlet之中，要注意区分html注释和java注释的区别

jsp注释不会被翻译到servlet中，会在翻译时遗弃

java注释不会被编译到class文件中，会在编译时遗弃 //java注释

html注释将会当作模版元素，直接输出到浏览器中，浏览器将不会显示html注释中的内容 <!—html注释 -->

### page指令

用来通知翻译引擎，如何翻译当前的JSP

[ language="java" ] 当前JSP使用的开发语言

[ extends="package.class" ] 当前jsp翻译成servlet后要继承的类，注意此值必须是一个servlet的子类，一般情况下不要改

[ import="{package.class | package.\*}, ..." ] 导入需要使用到的包 java.lang.\*;javax.servlet.\*;javax.servlet.jsp.\*;javax.servlet.http.\*;

[ session="true | false" ] 用来指定当前页面是否使用session，如果设置为true，则翻译过来的servlet中将会有对session对象的引用，于是可以直接在jsp中使用session隐式对象。但是这将导致一旦访问jsp就会调用request.getSession()方法，可能导致不必要的空间浪费。如果确定jsp中不需要session可以设为false

[ isThreadSafe="true | false" ] 翻译过来的servlet是否实现SingleThreadModel

[ errorPage="relative\_url" ] 如果页面出错，将要跳转到的页面，除了在jsp中使用此属性指定错误页面外也可以在web.xml中配置整个web应用的错误页面，如果两个都设置则jsp中的此属性起作用

[ isErrorPage="true | false" ] 如果设置此属性为true,翻译过来的servlet中将含有Exception隐式对象,其中封装的就是上一个页面中抛出的异常对象

[ contentType="mimeType [ ;charset=characterSet ]" | "text/html ; charset=ISO-8859-1" ] 和jsp乱码相关的指令,用来指定jsp输出时,设置的Content-Type响应头用来指定浏览器打开的编码

[ pageEncoding="characterSet | ISO-8859-1" ] 服务器翻译jsp时使用的编码集.如果向防止jsp乱码,应该保证文件的保存编码和jsp翻译成servlet用的编码以及输出到浏览器后浏览器打开的编码一致.此属性一旦设置好,翻译引擎会间接帮我们设置content-type属性.

[ isELIgnored="true | false" ] 当前页面是否使用el表达式,设置为false时表示启用el,j2ee4.0开始默认支持,j2ee4.0一下做开发时,如果要使用el表达式,需将此属性设置为false

[ buffer="none | 8kb | sizekb" ] out隐式对象所使用的缓冲区的大小

[ autoFlush="true | false" ] out隐式对象是否自动刷新缓冲区，默认为true，不需要更改；如果设置为false，需要手动调用out.flush()方法否则报错。

### include指令

<%@ incluede file=""%> 静态引入其他页面的内容

\*静态引入：

在源文件级别进行合并，多个jsp生成一个servlet，最终由这一个servlet生成响应。推荐使用。

\*动态引入：

在运行时将多个输出进行合并，多个jsp分别生成多个servlet，最终由这多个servlet生成响应，组成一个输出流，提供响应。执行效率没有静态引入高。

### taglib指令

<%@ taglib uri="" prefix=""%>用来引入标签库。

uri指定被引入.tld文件的名称空间

prefix 对该名称空间的一个缩写

## Jsp标签技术概述

在之前我们编写的程序中，JSP页面中大量的写入了java代码，我们发现在jsp页面中嵌入大量java代码会造成页面的可读性、可维护性下降。

如图-1所示：



图-1

那么该如何解决这种问题呢？sun公司考虑到这种问题，为我们提供了JSP标签技术，通过标签替换掉jsp页面中的java代码，从而提高jsp页面的可读性、可维护性。

JSP中的标签技术我们介绍如下四种：

JSP标签

EL表达式

JSTL标签

自定义标签

# JSP标签

由sun公司提供，属于jsp规范中的内容，不需要引入第三方标签库，我们介绍其中的三个。

<jsp:include>用来替代request.getRequestDispatcher().include()

<jsp:forward>用来替代request.getRequestDispatcher().forward()

<jsp:param>配合前两个标签使用，可以在包含或转发时，带一些参数过去

# 九大隐式对象

在jsp翻译成Serlvet时，jsp翻译引擎在翻译过来的Serlvet中，预先定义了一些变量，可以不需要在JSP中预先定义就可以直接使用，共有9个，统称为jsp的九大隐式对象。分别是：

page

config

application

response

request

session

out

exception

pageContext

其中其他的七个我们都学习过了，只需要再学习一下out和pageContext即可。

### out对象

可以将他理解成response.getWriter()获得的PrintWriter.

它自带一个缓冲区，其大小收page指令中的buffer的设定限制。当缓冲区满或缓冲区被关闭或当前jsp页面结束，则此缓冲区中的内容将被刷新到response.getWriter（）的缓冲区中。

### PageContext对象

(1)作为入口对象获取其它八大隐式对象。

getException方法返回exception隐式对象

getPage方法返回page隐式对象

getRequest方法返回request隐式对象

getResponse方法返回response隐式对象

getServletConfig方法返回config隐式对象

getServletContext方法返回application隐式对象

getSession方法返回session隐式对象

getOut方法返回out隐式对象

(2)作为入口对象获取其他域中的数据

pageContext操作所有域中属性的方法

getAttribute(String name,int scope)

setAttribute(String name, Object value,int scope)

removeAttribute(String name,int scope)

其中pageContext中代表域的常量：

PageContext.APPLICATION\_SCOPE

PageContext.SESSION\_SCOPE

PageContext.REQUEST\_SCOPE

PageContext.PAGE\_SCOPE

findAttribute方法：

在四大域中搜寻属性，搜寻的顺序是page域、request域、session域、application域，从小域到大域开始搜索，如果搜索到就直接获取该值，如果所有域中都找不到，返回一个null

(3)作为域对象使用

作用的范围：真个jsp页面，是四大作用域中最小的一个。

生命周期：当对jsp的请求开始时生成，当响应结束时销毁。

作用：在jsp页面范围内共享数据

(4)跳转到其他资源

其身上提供了forward和include方法，简化重定向和转发的操作

# el表达式

## EL概述

### EL概述

EL 全名为Expression Language，用来替代<%= %>脚本表达式。

在j2ee1.4以前默认是不支持el，如果需要需要指定page指令[isELIgnored="true | false" ]为false，J2EE4.0后默认支持el

EL具有获取数据、执行运算、获取常用开发对象、调用java方法这四方面的功能

## 获取数据

### 获取常量

EL支持数据、字符串、布尔类型的常量

${23}

${“asdf”}

${true}

### 获取变量：在四大作用域中搜寻属性

${propName}在四个域中搜寻proName属性，输出该值到输出流中

搜寻域的顺序为由小到大。

如果四大域中都搜寻不到，则什么都不输出。

### 获取变量：获取指定域中的属性

${pageScope/requestScope/sessionScope/applicationScope.proName}

获取指定域中的属性

### 获取变量：获取数组中的数据

${attr[0]}获取域中的数组的指定元素

### 获取变量：获取集合中的数据

${list[0]}获取list中的指定位置元素

### 获取变量：获取map中的数据

${map.keyName}获取map中指定键的值

### 获取变量：获取javabean属性

${bean.propName}获取javaBean的属性，可以认为是调用了javaBean的getXXX方法，

最重要的一个应用场景：

在写路径的时候最好不要把web应用名称写死，java中应该用request.getContextPath去获取，jsp中就用el获取：

${pageContext.request.contextPth}

### 获取数据细节

（1）和[]区别：使用点的地方都可以用中括号，如果属性名是数字或包含特殊符号（.-）就必须使用中括号。例子：${map["first.name"]}

（2）EL只能获取不能设置

（3）EL只能获取不能遍历

## 进行简单运算

### 算数运算

${3+2}

${“3”+2}

${“3”+2}

${1+”a”}

所有参与元算的元素都会被转成数字，如果不能转就报错，空元素参与运算当作没参与。

### 关系运算：

如图-2所示：



图-2

${3 > 2}

${3 gt 2}

### 逻辑运算:

如图-3所示：



图-3

${3>2 && 4>5}

### Empty运算符

empty/not empty判断对象是否为null，判断集合是否为空,数组长度是否为0，判断字符串是否为空串，判断域中是否没有任何属性。

### 三元表达式

${name == null ? "张三" : name;}

## 获取web开发常用对象

### EL的内置对象

El中预先定义了11个内置对象，不需要提前存入域中个，可以直接在el使用。

如图-4、图-5所示：

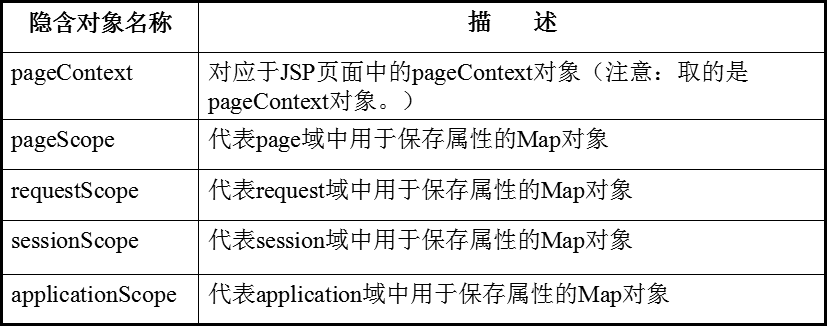


图-4

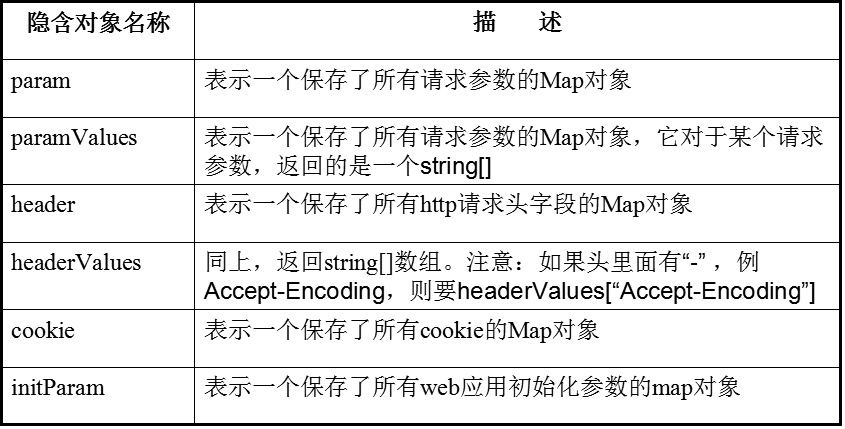


图-5

### pageContext

代表pageContext对象,通过他可以获取其他八大隐式对象

### 四大作用域中域属性组成的map

pageScope：代表page域，可以用来获取page域中的属性

reqeustScope：代表reqeust域，可以用来获取reqeust域中的属性

sessionScope：代表session域，可以用来获取session域中的属性

applicationScope：代表application域，可以用来获取application域中的属性

### 请求参数组成的map

param 代表请求参数组成的map集合${param.userName}

paramValues 代表请求参宿组成的map集合，但是此集合的value是String[]，用来获取一名多值的param

### 请求头组成的map

header 获取请求头组成的map

headerValues 获取请求头组成的map但是value是一个String[]，用来获取一名多值的head

### 所有cookie信息组成的map

cookie ：获取cookie组成的map对象，键为cookie的名，值为Cookie对象。

${cookie.cookieName.cookieValue}

### web应用初始化参数组成的map

initParam 以map封装的web.xml中配置的整个web应用的初始化参数

## 调用java方法

### 调用java方法

略。

# 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

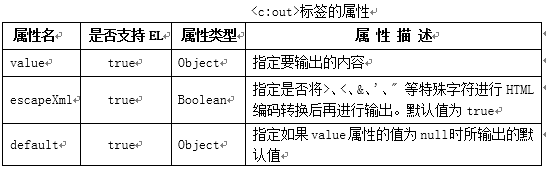
数据库标签 sql --Servlet

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：

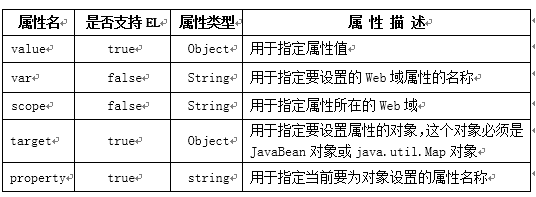


图-7

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：

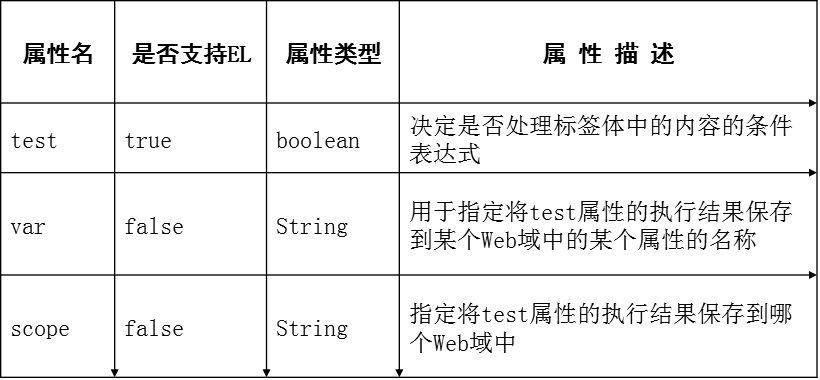


图-8

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>

标签用于对一个集合对象中的元素进行循环迭代操作，或者按指定的次数重复迭代执行标签体中的内容。

如图-9所示：

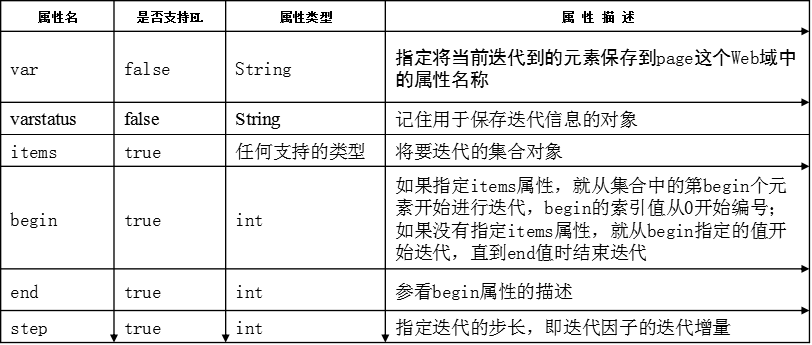


图-9

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>

如图-12所示：

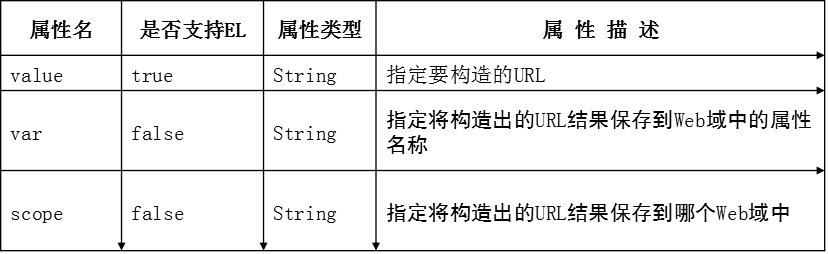
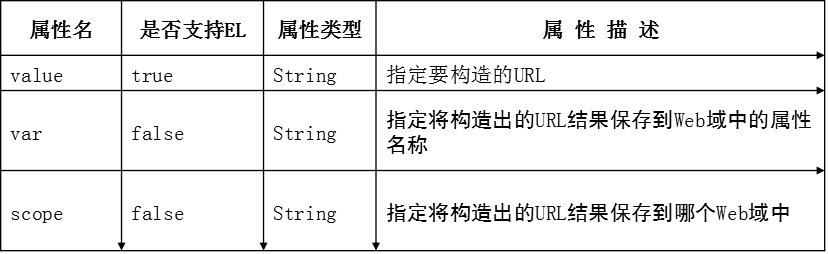


图-12

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：

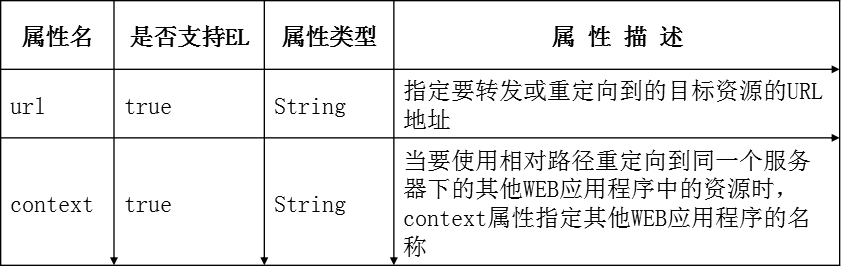
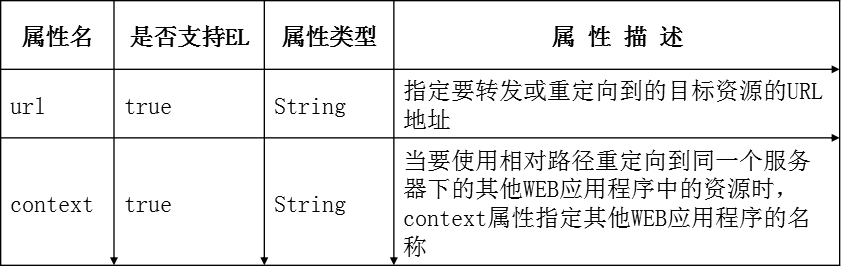


图-13

# 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：

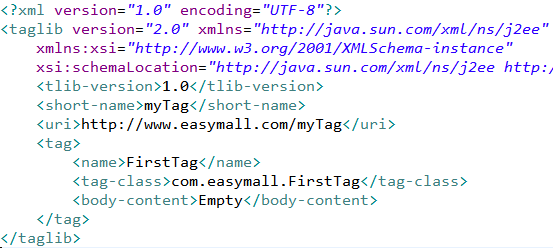
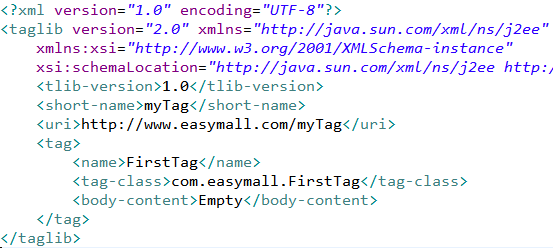


图-15

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：

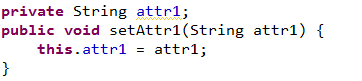


图-16

在tld文件中描述该属性，如图-17所示：

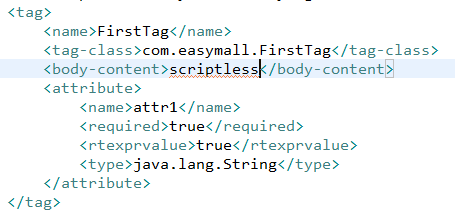


图-17

其中,描述属性的标签如图-18所示：

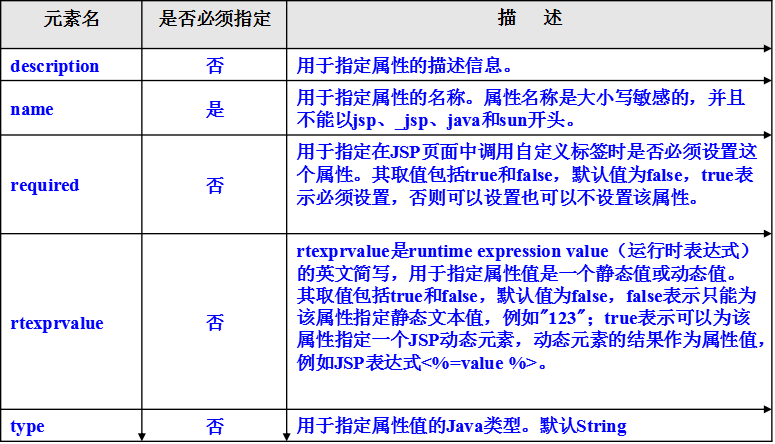


图-18

## 打包自定义标签为jar包

### 打包自定义标签为jar包

略。

# 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：

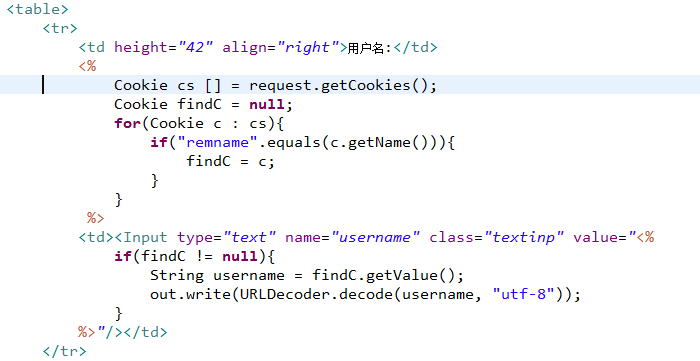


图-19

用el+jstl改造，如图-20所示：

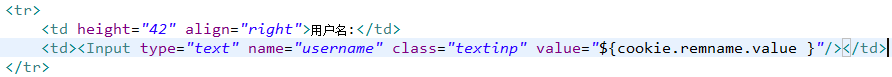


图-20

### 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：

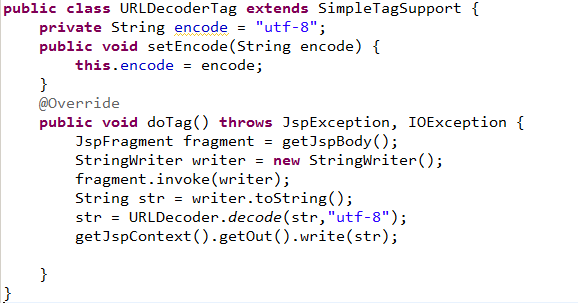


图-21

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