

# JSTL

## jsp常用标签

为了方便开发，省事

## JSTL

### 开头需要注意的地方

在Maven中集成springMVC时，在页面引入EL表达式，但是总是失效，在网上找资料后发现可以在页面中添加`<%@page isELIgnored="false"%>`，试验后发现果然问题解决。

主要原因是EL表达式无法被解析到。

其实从后台取值并传值到前台来根本就没有错，而前台JSP页面EL表达式无效，解析不到EL表达式，引起的原因是web.xml中：

```
<web-app version="2.5" xmlns="http://Java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
```

注意里面的web-app\_2\_5.xsd，就是这个引起的，在web-app\_2\_4.xsd中就不会出现这种问题(这个版本的isELIgnored默认设置为false)。在不改变web.xml2.5版本的情况下解决办法是：在jsp页面头加：`<%@page isELIgnored="false"%>` 问题得以解决。

还有就是：`<%@page isELIgnored="false"%>`的优先级要高于web.xml中的设置，所以在JSP中的设置会盖掉web.xml中的设置。

这个必须在之前加载的，

由于在Idea中我不知道怎么回事

这个IsElgonred是默认true的

所以也需要这样

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

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

由于JSTL标签和Servlet及JSP页面有着比较严格的版本对应关系

版本对应不正确很容易抛出异常。

### JSTL的基本配置

使用`<c:out>`输出标签

#### 核心标签

-最常用。最重要，也是本节课的重点

#### 格式化标签

#### SQL标签

#### XML标签

#### JSTL函数

--也很常用



### JSTL的好帮手EL表达式

#### 什么是EL表达式

全名为Expression Language

--经常与JSTL配合使用，使得JSP页面更直观，写法更简单

---普通写法：`<%=session.getValue("name")%>`

--EL表达式写法：`<c:out value="${sessionScope.name}" />`

#### EL表达式的格式

## **`${表达式}`**

--用美元符号"\$"界定，内容包括在花括号 "{}" 中；

","与"[]"运算符

通常情况是通用的 `${user.sex}` `${user["sex"]}`

"[]" 还可以用来进行集合元素中的定位 `${booklist[0].price}`



除2种情况外

1 包含特殊字符：`.${user.first-name}` 这是错误的  `${user["first-name"]}` 这是正确的

2 通过变量动态取值的时候  `${user[param]}` 这是正确的写法，不能用点

param可以是name of sex or others

"." 的写法是  `${user.name}` or  `${user.sex}` or..

EL变量

| JSP内置对象     | EL名称             |
|-------------|------------------|
| Page        | PageScope        |
| Request     | RequestScope     |
| Session     | SessionScope     |
| Application | ApplicationScope |

列如：我们在输出

```
<c:out value="${empty username}"></c:out>
```

没给范围时会从小到大得一个范围去找这个变量值



如果没有，就会输出空，不显示

EL的自动类型转换

这是我们用的以前的类型转换

```
- String str_count = request.getParameter("count")
- int count = Integer.parseInt(str_count);
- count = count + 20;
```

而EL会自动进行类型转换，只需要这样写

```
-- ${param.count+20}
```

EL隐式对象

| 序号 | 隐式对象             | 意义                             |
|----|------------------|--------------------------------|
| 1  | pageContext      | PageContext 实例对应于当前页面的处理       |
| 2  | pageScope        | 与页面作用域属性的名称和值相关联的Map类          |
| 3  | requestScope     | 与请求作用域属性的名称和值相关联的Map类          |
| 4  | sessionScope     | 与会话作用域属性的名称和值相关联的Map类          |
| 5  | applicationScope | 与应用程序作用域属性的名称和值相关联的Map类        |
| 6  | param            | 按名称存储请求参数的主要值的 Map 类           |
| 7  | paramValues      | 将请求参数的所有值作为 String 数组存储的 Map 类 |
| 8  | Header           | 按名称存储请求头主要值的 Map 类             |
| 9  | headerValues     | 将请求头的所有值作为 String 数组存储的 Map 类  |
| 10 | cookie           | 按名称存储请求附带的 cookie 的 Map 类      |
| 11 | initParam        | 按名称存储 Web 应用程序上下文初始化参数的Map类    |

## EL的运算符

运算符运行对数据和文字进行组合和比较

EL运算符：

| 类别    | 运算符   |
|-------|---|
| 算术运算符 | +、-、*、/ ( 或 div ) 和 % ( 或 mod )                 |
| 关系运算符 | ==(或eq)、!=(或ne)、<(或lt)、>(或gt)、<=(或le) 和 >=(或ge) |
| 逻辑运算符 | &&(或 and)、  (或or)和 !(或 not)                     |
| 验证运算符 | empty   |

EL符号：empty//用于判断变量是否为空

用法：\${empty usernam}//为空或""返回true，否则返回false

## JSTL的核心标签库标签共13个，从功能上分为4类，：

表达式控制标签：out set catch

流畅控制标签：if choose when otherwise

循环标签：forEach forTokens

URL操作标签：import url redirect

### out标签的用法

-输出常量

可在value属性中直接赋值

输出变量

变量不存在时刻配合default属性输出默认值，还可通过escapeXml控制转义字符的输出方式

```

<!--使用out标签输出常量-->
<c:out value="this is our  xxxx"></c:out>
<!--使用out标签输出变量-->
<%
session.setAttribute("name","jESSICA");
%>
<c:out value="${name}" default="error "></c:out>
<!--当变量不存在时 通过default输出默认值-->

<!--设置转义字符后的字符需要escapeXml属性值设置为Flase-->
<c:out value="&ltout标签&gt" escapeXml="false"></c:out>

```

### Set标签的用法

--存值到scope中，

可将值以变量形式存放在指定的范围中（2种方式）

```

<!--存值到scope中 -->
<c:set value="today" var="day" scope="session"></c:set>
<c:out value="${day}"></c:out>

```

--存值到Javabeans的属性中  
需要配合target属性指定对象并且通过property属性指定要赋值给对象的哪个属性

```
<!--if的标签的用法-->
<form action="FirstDemo.jsp" method="post">
    <input type="text" name="score" value="${param["score"]}">
    <input type="submit" value="提交"/>
</form>
<!--优秀 score>=90-->
<c:if test="${param.score}>=90}" var="result">
<c:out value="恭喜，您的成绩是优秀!"></c:out>
</c:if>
<c:if test="${param.score}>=80} && ${param.score}<=90}" var="result2">
<c:out value="恭喜，您的成绩是良好!"></c:out>
</c:if>

<c:out value="是不是90, ${result}"></c:out>
<c:out value="是不是80, ${result2}"></c:out>
```

### choose.when.otherwise标签的用法

--通常这三个标签放在一起配合使用

--<c:choose>标签嵌套在<c:when>和<c:otherwise>标签的外面作为他们的父标签使用

--其中choose和when标签也可以一起组合使用

综上所述通常有以下两种语法结构

用法一：3个标签同时使用

这里的注释是有问题的，是不能加在里面的，我只是为了方便观看而设置的  
而在实际使用中，是必须得去掉的

```
${param.score<60 && param.score>0}
```

这样的写法是在when标签中特有的，在if标签中会出错，if标签还是上面我写的方法

```
<c:choose>
<!--当输入的成绩大于等于90，小于等于100的时候 优秀-->
<c:when test="${param.score}>=90 && param.score<=100}">
<c:out value="优秀"></c:out>
</c:when>
<!--当输入的成绩大于等于80，小于90的时候 良好-->
<c:when test="${param.score}>=80 && param.score<90}">
<c:out value="良好"></c:out>
</c:when>
<!--当输入的成绩大于等于70，小于80的时候 中等-->
<c:when test="${param.score}>=70 && param.score<80}">
<c:out value="中等"></c:out>
</c:when>
<!--当输入的成绩大于等于60，小于70的时候 及格-->
<c:when test="${param.score}>=60 && param.score<70}">
<c:out value="及格"></c:out>
</c:when>
<!--当输入的成绩小于60，大于0的时候 不及格-->
<c:when test="${param.score}<60 && param.score>0}">
<c:out value="不及格"></c:out>
</c:when>
<!--当输入的成绩小于0，大于100的时候 -->
<c:otherwise>
<c:out value="您的输入有问题"></c:out>
</c:otherwise>
</c:choose>
```

用法二：只使用choose标签和when标签

```
<c:choose>
<c:when test="${param.score==100}">
<c:out value="你是第一名"></c:out>
</c:when>
</c:choose>
```

### forEach标签的用法

---根据循环条件遍历集合（Collection）中的元素

var设定变量名用于存储从集合中取出元素（必须无默认值）

items指定要遍历的集合（必须无默认值）

begin.end用于指定遍历的起始位置和终止位置

step指定循环的步长（有默认）

varStatus通过index。count。first。last几个状态值，描述begin和end子集中的元素的状态

常用用法一：全部遍历

```
<!--forEach标签的用法-->
<!--全部遍历-->
<%List<String> fruits=new ArrayList<String>();
    fruits.add("apple");
    fruits.add("orange");
    fruits.add("pear");
    fruits.add("watermelon");
    fruits.add("banana");
    fruits.add("grape");
    request.setAttribute("fruits",fruits);
%>
```

```
<c:forEach var="fruit" items="{fruits}">
<c:out value="{fruit}"></c:out><br />

</c:forEach>
<c:out value="===== "></c:out>
```

结果:

```
orange
pear
watermelon
banana
grape
```

## 常用用法二：部分遍历

```
<!--用法二：部分遍历-->
<c:forEach var="fruit" items="{fruits}" begin="1" end="4">
<c:out value="{fruit}"></c:out>
</c:forEach>
```

结果：

```
orange pear watermelon banana
```

## 常用方法三：部分遍历中指定步长

```
<!--用法三：部分遍历并且指定步长-->
<c:forEach var="fruit" items="{fruits}" begin="1" end="4" step="2">
<c:out value="{fruit}"></c:out>
</c:forEach>
```

结果：orange watermelon

## 常用用法四：部分遍历时输出元素的状态

```
<!--用法四：部分遍历时输出元素的状态-->
<c:forEach var="fruit" items="{fruits}" begin="1" end="4" varStatus="fru">
<c:out value="{fruit} 的四个属性值"></c:out><br>
<c:out value="index属性{fru.index}"></c:out><br>
<c:out value="count属性{fru.count}"></c:out><br>
<c:out value="first属性{fru.first}"></c:out><br>
<c:out value="last属性{fru.last}"></c:out><br>
<c:out value="-----"></c:out>
</c:forEach>
```

结果:

```
orange 的四个属性值
```

```
index属性1
count属性1
first属性true
last属性false
```

```
----- pear 的四个属性值
```

```
index属性2
count属性2
first属性false
last属性false
```

```
----- watermelon 的四个属性值
```

```
index属性3
count属性3
first属性false
last属性false
```

```
----- banana 的四个属性值
```

```
index属性4
count属性4
first属性false
last属性true
```

```
-----
```

## 循环控制标签详解：forTokens

forTokens 标签的用法:

--用于浏览器字符串，并根据指定的字符将字符串截取



items指定被迭代的字符串  
delims指定使用的分隔符  
var指定用来存放遍历到的元素

- **begin**指定遍历的开始位置 (int型从取值0开始)
- **end**指定遍历结束的位置 (int型, 默认集合中最后一个元素)
- **step**遍历的步长 (大于0的整型)
- **varStatus**通过index、count、first、last几个状态值, 描述begin和end子集中的元素的状态

```
<!--forTokens标签的用法-->
<c:forTokens items="010-880967789-123" delims="-" var="num">
<c:out value="{num}"></c:out>
</c:forTokens>
```

结果: ----- 010 880967789 123

```
<!--forTokens标签的用法-->
<c:forTokens items="010-880967789-123-123232" delims="-" var="num" begin="1" end="2"
varStatus="tel">
<c:out value="{num}"></c:out><BR>
<c:out value="index属性{tel.index}"></c:out><BR>
</c:forTokens>
<BR>
```

结果: 880967789

index属性1

123

index属性2

URL操作标签

import标签的用法:

可以把其他静态页面文件包含到本JSP页面

同<jsp:include>的区别在于: 只能包含一个web应用的文件, 而<c:import>可以包含其他web应用的文件, 甚至是网络上的资源

url属性被导入资源的URL地址

context相同服务器下的其他web工程, 必须以"/"开头

var以String类型存入被包含文件的内容

Scope var变量的JSP范围

charEncoding被导入文件的编码格式

varReader以Reader类型存储被包含文件内容

```
<c:import url="" context="" var=""
scope="" charEncoding="" varReader="">
</c:import>
```

```
<!--import标签的用法-->
<!--导入网络的绝对路径-->
<c:catch var="error09">
<c:import url="http://www.imooc.com/"></c:import>
</c:catch>
<c:out value="{error09}"></c:out><BR>
<br>
<!--导入相对路径的文件-->
<c:catch var="error08">
<c:import var="22text" scope="session" url="22.txt" charEncoding="gbk"></c:import>
</c:catch>
<c:out value="{error08}"></c:out>
<br>

<%
String xt=(String) session.getAttribute("22text");
%>
<%=xt%>
```

context引入的内容是webapps下的其他Web project

---1 修改Tomcat的发布路径

----2修改%TOMCAT\_HOME%/conf/context.xml的context标签的属性  
crossContext;

```
<!--context属性的用法-->
<c:catch var="error08">
<c:import url="/success.jsp" context="/maven-strtus2"></c:import>
</c:catch>
<c:out value="${error08}"></c:out>
```

## URL操作标签详解

### redirect标签的用法

--该标签来实现请求的重定向,同时可以在url中加入指定的参数

url 指定重定向的地址,可以是一个string类型的绝对地址或相对地址

context用于导入其他web项目的页面

这是SecondDemo.jsp

```
<!--redirect标签的用法-->
<c:redirect url="FirstDemo.jsp">
<c:param name="username">Lily</c:param>
<c:param name="password">aaaaaa</c:param>
</c:redirect>
```

这是FirstDemo.jsp 获取到传过来的参数

```
<c:out value="username是${param.username}"></c:out>
<br><c:out value="password是${param.password}"></c:out>
```

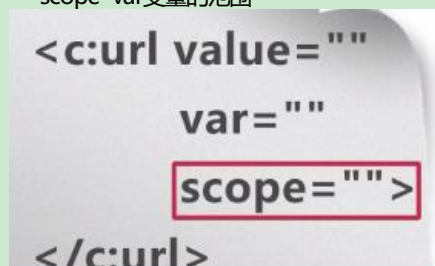
### url标签的用法

该标签用于动态生成一个String类型的URL,可以同<c:param>标签共同使用,也可以使用html的<a>标签实现超链接

value表示url路径

var 将url路径存储在变量中

scope var变量的范围



```
<c:url value=""
var=""
scope="">
</c:url>
```

```
<!--url标签-->
<c:if test="${1<3}">
<c:set var="partUrl" >aa</c:set>
</c:if>
<c:url value="http://localhost:8080/${partUrl}" var="newUrl" scope="session"></c:url>
<a href="${newUrl}">新的URL</a>
```

## JSTL的常用函数

使用JSTL函数之前 必须加的 其中第一个是因为在idea中的maven出现了问题,在上面有说

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

contains函数,是否包含指定字符串

```
<c:out value=""`你好, Hello World! `字符串中, 是否包含`Hello`: ${fn:contains('你好, Hello World! ', 'Hello')} "></c:out><br>
<c:out value=""`你好, Hello World! `字符串中, 是否包含`aaaa`: ${fn:contains('你好, Hello World! ', 'aaaa')} "></c:out><br>
```

结果:“你好, Hello World!”字符串中, 是否包含“Hello”: true

“你好, Hello World!”字符串中, 是否包含“aaaa”: false



containsIgnoreCase函数，是否包含，忽略大小写

```
<c:out value="\你好, Hello World! "字符串中, 是否包含"hello": ${fn:containsIgnoreCase('你好, Hello World! ', 'hello')} "></c:out><br>
```

结果：“你好, Hello World!”字符串中, 是否包含“hello”: true

**endsWith("字符串1","字符串2")**  
用于判断字符串2是否在字符串1结尾,  
**startsWith("字符串1","字符串2")**  
用于判断字符串2是否在字符串1开头

```
<c:out value="\你好, Hello World! "字符串中, 是否以"World!"结尾: ${fn:endsWith('你好, Hello World!', 'World!')} "></c:out><br>
```

结果：“你好, Hello World!”字符串中, 是否以“World!”结尾: true

输出我们想输出的xml的函数 fixescapeXml, 一般情况是不输出xml的

```
<book>冰与火之歌</book><br>  
<c:out value="${fn:escapeXml(' <book>冰与火之歌</book> ')} "></c:out><br>
```

结果: 冰与火之歌

&lt;book&gt;冰与火之歌&lt;/book&gt;

```
<c:out value="<book>冰与火之歌</book>"></c:out>
```

这是直接输出的, 不转义

结果: <book>冰与火之歌</book>

还有一些常用的函数

```
<c:out value="\你好, Hello World! "字符串中, llo的位置index值": ${fn:indexOf('你好, Hello World! ', 'llo')} "></c:out><br>  
<c:out value="\你好, Hello World! "字符串的长度是: ${fn:length('你好, Hello World! ')} "></c:out><br>  
<c:out value="\你好, Hello World! "字符串的第3-7个字符: ${fn:substring('你好, Hello World! ', 3, 7)} "></c:out><br>
```

结果：“你好, Hello World!”字符串中, llo的位置index值”: 5

“你好, Hello World!”字符串的长度是: 15

“你好, Hello World!”字符串的第3-7个字符: Hell

还需要多加练习才能熟练使用