# 标准标签库(JSTL)

## 8.1 准备实验环境

本章中所有的例子程序都在名为JSTL的WEB应用程序中运行，使用的Tomcat服务器版本为Tomcat5.5。为了简化代码和突出重点，本章中所有的JSP网页文件的例子程序都省略了<html>、<head>、<title>、<body>等HTML全局架构标签。本章中用于描述标签的语法的各种符号的意义如下：

* […]：表示方括号中的内容是可选的；
* {option1|option2|option3|…}：表示花括号中用“|”字符分隔的各个选项不能同时出现，只能出现其中的一项；
* 如果属性的值字体加粗，表示这个值是对应属性的默认值；

如果属性的值用斜体字表示，表示该属性可以是表达式；如果标签体的内容用斜体字表示，表示标签体的内容可以是JSP代码。

为了本章后面讲解的方便，这里先创建一个名为UserBean的JavaBean程序，如例程8-1所示。

例程 8-1 UserBean.java

package org.it315;

public class UserBean

{

private String userName;

private String password;

private String address;

private int visitCount;

public void setUserName(String userName)

{

this.userName = userName;

}

public String getUserName()

{

return userName;

}

public void setPassword(String password)

{

this.password = password;

}

public String getPassword()

{

return password;

}

public void setAddress(String address)

{

this.address = address;

}

public String getAddress()

{

return address;

}

public void setVisitCount(int visitCount)

{

this.visitCount = visitCount;

}

public int getVisitCount()

{

return visitCount;

}

}

编译这个源程序，将编译后的UserBean.class文件放在<TOMCAT\_HOME>\webapps\JSTL\WEB-INF\classes\org\it315目录中。

## 8.2 JSTL概述

### 8.2.1 JSTL简介

从JSP 1.1规范开始JSP就支持使用自定义标签，使用自定义标签大大降低了JSP页面的复杂度，同时增强了代码的重用性，因此自定义标签在WEB应用中被广泛使用。许多WEB应用厂商都开发出了自己的一套标签库提供给用户使用，这导致出现了许多功能相同的标签，令网页制作者无所适从，不知道选择哪一家的好。为了解决这个问题，Apache Jakarta小组归纳汇总了那些网页设计人员经常遇到的问题，开发了一套用于解决这些常用问题的自定义标签库，这套标签库被SUN公司定义为标准标签库（The JavaServer Pages Standard Tag Library），简称JSTL。使用JSTL可以解决用户选用不同WEB厂商的自定义标签时的困惑，JSP规范同时也允许WEB容器厂商按JSTL标签库的标准提供自己的实现，以获取最佳性能。

由于JSTL是在JSP 1.2规范中定义的，所以JSTL需要运行在支持JSP 1.2及其更高版本的WEB容器上，例如，Tomcat 5.5。

**1．JSTL的逻辑组成**

笔者在写本章内容时的JSTL规范的最新版本是JSTL 1.1，它是JSP 2.0最新规范中的一个重要组成部分。JSTL 1.1中包含四个标签库和一组EL函数。**为方便用户使用，JSP规范中描述了JSTL的各个标签库的URI地址和建议使用的前缀名，如表8.1所示。本章中在使用JSTL标签时，使用的都是这些建议的前缀。**

表8.1 JSTL

|  |  |  |
| --- | --- | --- |
| **标签库功能描述** | **标签库的URI** | **建议前缀** |
| 核心标签库 | http://java.sun.com/jsp/jstl/core | c |
| XML标签库 | http://java.sun.com/jsp/jstl/xml | x |
| 国际化/格式化标签库 | <http://java.sun.com/jsp/jstl/fmt> | fmt |
| 数据库标签库 | <http://java.sun.com/jsp/jstl/sql> | sql |
| EL自定义函数 | http://java.sun.com/jsp/jstl/functions | fn |

**下面对JSTL的各个标签库进行简单的介绍：**

（1）核心标签库中包含了实现WEB应用中的通用操作的标签。例如，用于输出一个变量内容的<c:out>标签、用于条件判断的<c:if>标签、用于迭代循环的<c:forEach>标签。

（2）国际化/格式化标签库中包含实现WEB应用程序的国际化的标签。例如，设置JSP页面的本地信息、设置JSP页面的时区、绑定资源文件，使本地敏感的数据（例如数值、日期等）按照JSP页面中设置的本地格式显示。

（3）数据库标签库中包含用于访问数据库和对数据库中的数据进行操作的标签。例如，从数据源中获得数据库连接、从数据库表中检索数据等。由于在软件分层的开发模型中，JSP页面仅用作表现层，我们一般不在JSP页面中直接操作数据库，而是在业务逻辑层或数据访问层操作数据库，所以，JSTL中提供的这套数据库标签库没有多大的实用价值。

（4）XML标签库中包含对XML文档中的数据进行操作的标签。例如，解析XML文档、输出XML文档中的内容，以及迭代处理XML文档中的元素。因为[XML](http://www.kissjava.com/zhuanti/struts/2005-06-05/)广泛应用于WEB开发，对[XML](http://www.kissjava.com/zhuanti/struts/2005-06-05/)文档的处理非常重要，XML标签库使处理XML文档变得简单方便，这也是[JSTL](http://www.kissjava.com/zhuanti/struts/2005-06-05/)的一个重要特征。

（5）JSTL中提供的一套EL自定义函数包含了JSP页面制作者经常要用到的字符串操作。例如，提取字符串中的子字符串、获取字符串的长度和处理字符串中的空格等。

**2．JSTL的物理组成**

完整的JSTL应包含SUN公司提供的jstl.jar 包和WEB容器产商提供的JSTL实现包，以Apache Jakarta 小组提供的JSTL实现包为例，完整的JSTL包含jstl.jar、standard.jar 和 xalan.jar 三个 jar 包。sun 公司提供的jstl.jar 包封装了 JSTL所要求的一些API 接口和类，Apache Jakarta 小组编写的 JSTL API 实现类封装在standard.jar包中。由于 JDK 在 JDK 1.5 版本中才引入了 XPath API，而 apache Jakarta 小组开发的 JSTL API 是在 JDK1.5 之前推出的，所以 apache Jakarta 小组在 JSTL 中使用的是他们自己开发的 XPath API，这些 API 封装在 xalan.jar 包中。Standard.jar 包中包括核心标签库、国际化/格式化标签库、数据库标签库中的标签和标准的EL自定义函数的实现类，xalan.jar 包中包括 JSTL 解析 XPath 的相关 API 类。

### 8.2.2 安装和测试JSTL

**1．下载JSTL包**

可以从Apache的网站下载JSTL的jar包。进入<http://jakarta.apache.org/taglibs/> 页面，根据相关的提示信息下载 zip压缩文件，这个压缩文件中包含JSTL的三个jar包。

**提示：**由于网站的不断升级，下载JSTL 的jar包的网址可能改变。读者在下载这些文件时，如果在指定的网址没有下载的链接，可以进入Apache的网站首页，然后根据相关的提示下载所需的文件。例如，笔者在编写JSTL初稿时，从<http://jakarta.apache.org/taglibs/> 下载的压缩文件中只包含jstl.jar**和standard.jar两个 jar 包，而 xalan.jar包要从** <http://xml.apache.org/xalan-j/downloads.html> 下载。

**2．安装JSTL**

在Tomcat中安装JSTL很简单，只要将 [JSTL](http://www.kissjava.com/zhuanti/struts/2005-06-05/)的jar包复制到<WEB应用程序>\WEB-INF\lib目录下即可。本章中所有的例子程序都在一个名为JSTL的WEB应用程序中运行，所以，将[JSTL](http://www.kissjava.com/zhuanti/struts/2005-06-05/)**的三个jar包复制到 <TOMCAT\_HOME>\webapps\JSTL\WEB-INF\lib目录下**即完成了JSTL的安装**。**

**提示：Tomcat 5.5自带的JSP例子应用程序中提供了使用JSTL标签的例子，在<TOMCAT\_HOME>\webapps\jsp-examples\WEB-INF\lib目录中有JSTL的jstl.jar包和standard.jar包，所以，对于读者来说，如果不使用JSTL的**XML**标签库中的标签，可以直接将这两个包复制到<WEB应用程序的主目录>\WEB-INF\lib目录中，而不用事先从Apache的网站下载JSTL的jar包。**

**3．测试JSTL**

安装完JSTL后，接下来就可以准备测试[JSTL](http://www.kissjava.com/zhuanti/struts/2005-06-05/)安装得是否成功了。

Apache小组在提供JSTL时，同时也提供了JSTL标签的标签描述文件（即TLD文件），它们都封装在standard.jar包中，位于该jar文件的META-INF目录中。用WinRAR 打开standard.jar包，进入META-INF目录，可以看到JSTL标签的标签库描述文件。查看这些TLD源文件，可以看到文件中定义的 **<uri> 元素的内容与表8.1中列出的标签库的URI相同。**因为standard.jar 包中封装了JSTL标签的标签库描述文件，**Tomcat在启动时会自动加载standard.jar包中的META-INF目录下的所有TLD文件，所以使用JSTL时不必在web.xml文件中配置JSTL标签库的URI信息，就可以在JSP文件中直接使用JSTL标签。**

要在JSP文件中使用JSTL，首先就需要**使用taglib指令导入所需要使用的JSTL标签库，**taglib指令的uri属性必须为相应标签库的TLD文件中的**<uri>元素的值，**taglib指令的**prefix属性可以自己随意指定，但最好是采用表8.1中的建议前缀。**在JSTL应用程序的主目录中按例程8-2编写一个包含JSTL标签的简单JSP文件test.jsp。

例程8-2 test.jsp

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

<c:out value="hello world!" />

**上面的**test.jsp页面中的<c:out>标签用于向浏览器输出文本内容，它属于**JSTL的核心标签库中的标签**。**由于在test.jsp文件中只使用了JSTL的核心标签库中的标签，所以，只需要使用一条taglib指令导入JSTL的核心标签库，而不必使用多条taglib指令导入JSTL的所有标签库。**

启动[Tomcat](http://www.kissjava.com/zhuanti/struts/2005-06-05/)后，在浏览器地址栏中输入test.jsp页面的地址进行访问，如果浏览器中显示出了“hello world!”，说明JSTL安装成功。

## 8.3 核心标签库

JSTL核心标签库包含了一组用于实现WEB应用中的通用操作的标签，JSP规范为核心标签库建议的前缀名为c。

### 8.3.1 <c:out>标签

<c:out> 标签用于输出一段文本内容到pageContext对象当前保存的“out”对象中，在通常情况下，pageContext对象当前保存的“out”对象的数据是输出到客户端浏览器，所以，<c:out> 标签通常用于输出一段文本内容到客户端浏览器。如果<c:out> 标签输出的文本内容中包含了需要进行转义的HTML特殊字符，例如，<、>、'、"、&等，<c:out> 标签默认对它们按表8.2进行HTML编码转换后再进行输出，这样就可以在浏览器中显示出这些字符。

表8.2 特殊字符转换

****

<c:out>标签标签有两种语法格式：

**语法1，没有标签体的情况：**

<c:out value="*value*"

[escapeXml="{true|false}"]

[default="*defaultValue*"] />

**语法2，有标签体的情况，在标签体中指定输出的默认值：**

<c:out value="*value*"

[escapeXml="{true|false}"] >

*default value*

</c:out>

<c:out>标签的属性说明如表8.3所示。

表8.3 <c:out>标签的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 是否支持EL | 属性类型 | 属 性 描 述 |
| value | true | Object | 指定要输出的内容 |
| escapeXml | true | Boolean | 指定是否将>、<、&、'、" 等特殊字符进行HTML编码转换后再进行输出。默认值为true |
| default | true | Object | 指定如果value属性的值为null时所输出的默认值 |

当且仅当value属性的值为null时，<c:out> 标签输出默认值；如果没有指定默认值，默认为空字符串。<c:out> 标签的标签体的内容用于指定输出的默认值，如果value属性的值不为null，即使标签体部分不为空，标签体的内容也不会被输出。如果value属性不是指向一个java.io.Reader对象，<c:out> 标签将调用这个对象的toString方法，然后输出获得的字符串内容。如果value属性指向一个java.io.Reader对象，<c:out> 标签将从这个Reader对象中读取数据后进行输出，当有大量数据要被输出时，将这些数据以Reader对象的形式提供给<c:out>标签进行输出，将会极大提高系统性能。当采用escapeXml属性的默认设置值true时，<c:out>标签将对要输出的特殊字符按表8.2进行转换；如果将escapeXml属性设置为false，<c:out>标签将不对要输出的特殊字符进行转换，而是直接输出这些特殊字符。

例程8-3是<c:out> 标签的一个演示例子程序。

例程8-3 c\_out1.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<%

session.setAttribute("test\_session", "testValue\_session");

%>

直接输出一个字符串：

<c:out value="test" default="123456" /><br /><hr />

输出Web域中不存在的属性的值：<br />

<c:out value="${test\_request}" default="这个属性不存在"/><br /><hr />

输出Web域中的test\_session属性的值：<br />

<c:out value="${test\_session}" />

例程8-3的运行结果如图8.1所示。

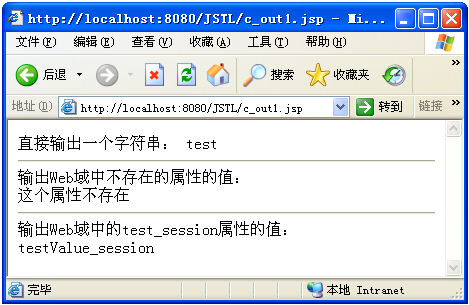
****

图8.1

例程8-4和例程8-5是两个用于演示<c:out>标签的escapeXml属性的例子程序，例程8-4没有设置escapeXml属性，例程8-5将escapeXml属性设置为了false。

例程8-4 c\_out2.jsp

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

<c:out value="${null}" >

<meta http-equiv="refresh" content="0;url=http://www.it315.org" />

</c:out>

例程8-5 c\_out3.jsp

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

<c:out value="${null}" escapeXml="false">

<meta http-equiv="refresh" content="0;url=http://www.it315.org" />

</c:out>

例程8-4和例程8-5的运行结果分别如图8.2和图8.3所示。

****

图8.2

****

图8.3

### 8.3.2 <c:set>标签

<c:set>标签用于设置各种Web域中的属性，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。<c:set>标签有四种语法格式：

语法1，使用value属性设置指定域中的某个属性的值：

<c:set value="*value*"

var="varName"

[scope="{page|request|session|application}"] />

语法2，在标签体中设置指定域中的某个属性的值：

<c:set var="varName"

[scope="{page|request|session|application}"]>

*body content*

</c:set>

语法3，使用value属性设置Web域中的一个属性对象的某个属性：

<c:set value="*value*"

target="target"

property="propertyName" />

语法4，在标签体中设置Web域中的一个属性对象的某个属性性：

<c:set target="target"

property="propertyName">

*body content*

</c:set>

<c:set>标签的属性说明如表8.4所示。

表8.4 <c:set>标签的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 是否支持EL | 属性类型 | 属 性 描 述 |
| value | true | Object | 用于指定属性值 |
| var | false | String | 用于指定要设置的Web域属性的名称 |
| scope | false | String | 用于指定属性所在的Web域 |
| target | true | Object | 用于指定要设置属性的对象，这个对象必须是JavaBean对象或java.util.Map对象 |
| property | true | string | 用于指定当前要为对象设置的属性名称 |

如果使用第1种语法格式时的value属性值为null，或者使用第2种语法格式时的标签体内容为空，<c:set>标签将从scope属性指定的域范围中删除var属性指定的属性。

在第3种语法格式和第4语法格式中，如果target属性的值是java.util.Map对象，property属性表示该Map对象的关键字，如果Map对象没有指定的关键字，就给Map对象增加指定的关键字；如果target属性的值是JavaBean对象，property属性表示JavaBean对象的属性，如果value的类型与JavaBean属性的类型不匹配时，会根据EL的转换规则自动进行转换。当使用第3种语法格式或第4种语法格式时，如果target属性的值为null(即target属性指定的对象不存在)，或者target属性的值是一个JavaBean对象，但该JavaBean中不存在property属性指定的属性，<c:set>标签将抛出异常。如果使用第3种语法格式时value属性的值为null，或者使用第4种语法格式时标签体的内容为空，如果target属性的值是一个java.util.Map对象，就从Map对象中删除property属性指定的关键字对应的项；如果target属性的值是一个JavaBean对象，就将JavaBean的相应属性的值设置为null。

例程8-6是使用<c:set>标签设置某个Web域中的属性的一个演示例子程序。

例程8-6 c\_set1.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<c:set var="userName" scope="session">

zxx

</c:set>

输出session 作用域中的userNamer属性的值:

<c:out value="${userName}" /><br /><hr />

输出session 作用域中的bookname属性的值:

<c:set var="bookname" scope="session" />

<c:out value="${bookname}" />

例程8-6的运行结果如图8.4所示。

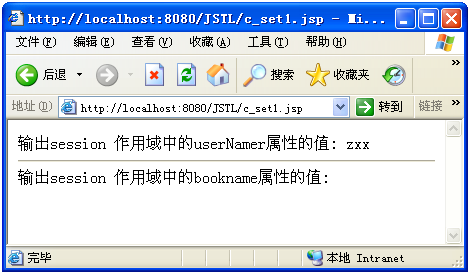


图8.4

例程8-7是使用<c:set>标签设置UserBean对象和Map对象的属性的一个演示例子程序。

例程8-7 c\_set2.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<%@ page import="java.util.HashMap" %>

<jsp:useBean id="user" class="org.it315.UserBean" />

<%

HashMap preferences = new HashMap();

session.setAttribute("preferences",preferences);

%>

设置和输出UserBean对象的userName属性值:

<c:set value="zxx" target="${user}" property="userName" />

<c:out value="${user.userName}" /><br /><hr />

设置和输出UserBean对象的password属性值:

<c:set target="${user}" property="password" />

<c:out value="${user.password}" /><br /><hr />

设置和输出Map对象的color关键字的值:

<c:set target="${preferences}" property="color" value="${param.color}" />

<c:out value="${preferences.color}" />

在浏览器地址栏输入如下地址访问例程8-7：

http://localhost:8080/JSTL/c\_set2.jsp?color=green

例程8-7的运行结果如图8.5所示。

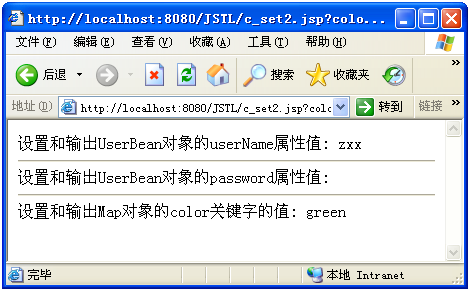


图8.5

### 8.3.3 <c:remove>标签

<c:remove>标签用于删除各种Web域中的属性，其语法格式如下：

<c:remove var="varName"

[scope="{page|request|session|application}"] />

var属性用于指定要删除的属性的名称，scope属性用于指定要删除的属性所属的Web域，它们的值都不能接受动态值。如果没有指定scope属性，<c:remove>标签就调用PageContext.removeAttribute(*varName*)方法，否则就调用PageContext.removeAttribute(*varName*, *scope*) 方法。<c:remove>与<c:set>标签第一种语法格式的value属性值为null时的作用相同。

例程8-8是使用<c:remove>标签的一个演示例子程序。

例程8-8 c\_remove.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<c:set value="org.it315" var="company" scope="request" />

<c:set var="passWord" scope="session">

a123456a

</c:set>

在没有使用 &lt;c:remove&gt; 标签之前，变量的值为：<br />

company:<c:out value="${company}" /><br />

passWord:<c:out value="${passWord}" /><br />

<c:remove var="company" scope="request" />

<!--c:set var="company" scope="request" 此行与黑体字的一行作用是一样的/-->

<c:remove var="passWord" scope="session" /><hr />

在使用 &lt;c:remove&gt; 标签之后，变量的值为：<br />

company:<c:out value="${company}" /><br />

passWord:<c:out value="${passWord}" />

例程8-8的运行结果如图8.6所示。

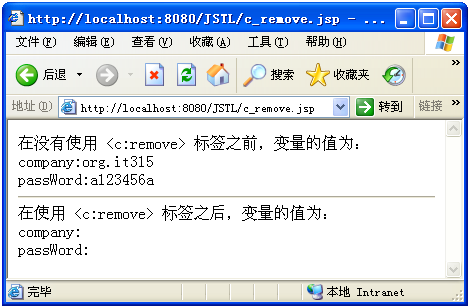
****

图8.6

### 8.3.4 <c:catch>标签

<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="varName"]>

*nested actions*

</c:catch>

var属性用于标识<c:catch>标签捕获的异常对象，其值是一个静态的字符串，不支持动态属性值。<c:catch>标签将捕获的异常对象以var指定的名称保存到page这个Web域中，如果没有指定var属性，则<c:catch>标签仅捕获异常，不在page域保存异常对象。如果<c:catch>标签体中的内容没有抛出异常，<c:catch>标签将从page域中删除var属性指定的属性。

<c:catch>标签可以捕获任何标签抛出的异常，并且可以同时处理多个标签抛出的异常，这样，可以对JSP页面的异常进行统一处理，显示给用户一个更友好的页面。JSP处理异常的通用机制是出现重要异常后跳转到错误处理页面，建议尽量不要用<c:catch>标签来代替JSP的错误处理机制，只对一些次要异常才使用<c:catch>标签进行捕获处理。

例程8-9是使用<c:catch>标签进行异常捕获处理的一个演示例子程序。

例程8-9 c\_catch.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<c:catch var="myex">

<%

int i = 0;

int j = 10;

out.println(j + "/" + i + "=" + j/i);

%>

</c:catch>

异常：<c:out value="${myex}" /><br />

异常 myex.getMessage：<c:out value="${myex.message}" /><br />

异常 myex.getCause：<c:out value="${myex.cause}" /><br />

异常 myex.getStackTrace：<c:out value="${myex.stackTrace}" />

在例程8-9中，<c:catch>标签内嵌套的脚本元素抛出了异常，<c:catch var="myex">将捕获到这个异常，调用<c:out value="${myex.message}" />，相当于调用<%=myex.getMessage()%>。例程8-9的运行结果如图8.7所示。

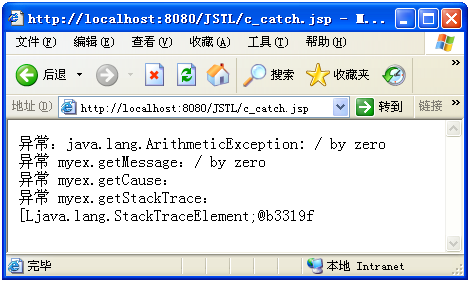
****

图8.7

### 8.3.5 <c:if>标签

JSP页面的显示逻辑中也经常需要进行条件判断，<c:if>标签可以构造简单的“if-then”结构的条件表达式，如果条件表达式的结果为真就执行标签体部分的内容。<c:if>标签有两种语法格式：

语法1，没有标签体的情况：

<c:if test="*testCondition*" var="*varName*"

[scope="{**page**|request|session|application}"] />

语法2，有标签体的情况，在标签体中指定要执行的内容：

<c:if test="*testCondition*" [var="*varName*"]

[scope="{**page**|request|session|application}"]>

*body content*

</c:if>

<c:if>标签的属性说明如表8.5所示。

表8.5 <c:if>标签的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 是否支持EL | 属性类型 | 属 性 描 述 |
| test | true | boolean | 决定是否处理标签体中的内容的条件表达式 |
| var | false | String | 用于指定将test属性的执行结果保存到某个Web域中的某个属性的名称 |
| scope | false | String | 指定将test属性的执行结果保存到哪个Web域中 |

对于语法2，如果指定了<c:if>标签的scope属性，则必须指定var属性。

例程8-10是使用<c:if>标签的一个演示例子程序。

例程8-10 c\_if.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<jsp:useBean id="user" class="org.it315.UserBean" />

<c:set value="${param.count}" target="${user}" property="visitCount" />

<c:if test="${user.visitCount == 1}">

这是您第一次访问本网站，欢迎您！

</c:if>

在浏览器地址栏输入如下地址访问例程8-10：

http://localhost:8080/JSTL/c\_if.jsp?count=1

例程8-10的运行结果如图8.8所示。

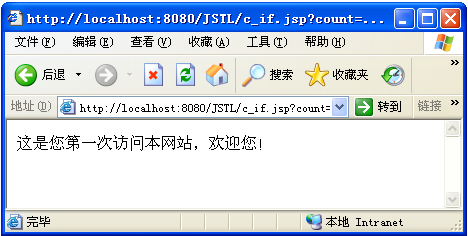


图8.8

### 8.3.6 <c:choose>标签

<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>标签没有属性，在它的标签体内只能嵌套一个或多个<c:when>标签和0个或一个<c:otherwise>标签，并且同一个<c:choose>标签中的所有<c:when>子标签必须出现在<c:otherwise>子标签之前。如果<c:choose>标签内嵌套一个<c:when>标签和<c:otherwise>标签，就相当于“if-else”的条件判断结构；如果<c:choose>标签内嵌套多个<c:when>标签和一个<c:otherwise>标签，就相当于“if-else if-else”标签。

<c:when>标签只有一个test属性，该属性的值为布尔类型。test属性支持动态值，其值可以是一个条件表达式，如果条件表达式的值为true，就执行这个<c:when>标签体的内容。<c:when>标签体的内容可以是任意的JSP代码。<c:otherwise>标签没有属性，它必须作为<c:choose>标签的最后分支出现。

当JSP页面中使用<c:choose>标签时，嵌套在<c:choose>标签内的test条件成立的第一个<c:when>标签的标签体内容将被执行和输出。当且仅当所有的<c:when>标签的test条件都不成立时，才执行和输出<c:otherwise>标签的标签体内容。如果所有的<c:when>标签的test条件都不成立，并且<c:choose>标签内没有嵌套<c:otherwise>标签，则不执行任何操作。

例程8-11是使用<c:choose>、<c:when>、<c:otherwise>标签的一个演示例子程序。

例程8-11 c\_choose.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

<c:set value="${param.count}" var="count" />

<c:choose>

<c:when test="${count == 0}">

对不起，没有符合您要求的记录。

</c:when>

<c:otherwise>

符合您要求的记录共有${count}条.

</c:otherwise>

</c:choose>

在浏览器地址栏输入如下地址访问例程8-11：

http://localhost:8080/JSTL/c\_choose.jsp?count=0

例程8-11的运行结果如图8.9所示。如果将参数count的值修改为10，则运行结果如图8.10所示。

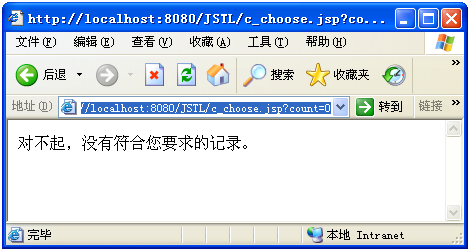


图8.9

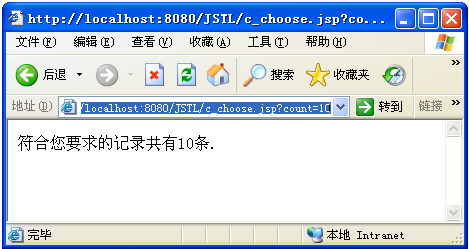


图8.10

例程8-12是一个综合使用<c:if>标签和<c:choose>等标签的例子程序，在这个例子程序中，首先使用<c:if>标签判断表单提交的方式是否是POST，如果是，就再使用<c:choose>等标签根据表单提交的内容进行不同的处理。

例程8-12 c\_customLogic.jsp

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

<c:if test="${pageContext.request.method=='POST'}">

<c:choose>

<c:when test="${param.favor == 'computer'}">

Your favourite course is <b>computer</b>.

</c:when>

<c:otherwise>

Your favourite course is <i>other</i>.

</c:otherwise>

</c:choose>

</c:if>

<form method="POST">What is your favourite course?<br/>

<input type="text" name="favor" />

<input type="submit" value="submit" />

</form>

在浏览器地址栏中直接访问c\_customLogic.jsp的运行结果如图8.11所示，在文本框中输入“computer”，单击其中的submit按钮后的运行结果如图8.12所示。

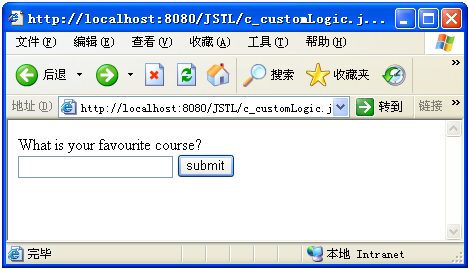


图8.11

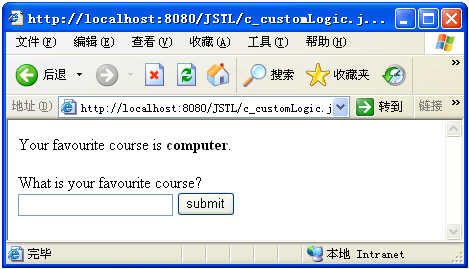


图8.12

### 8.3.7 <c:forEach>标签

JSP页面的显示逻辑中也经常需要对集合对象进行循环迭代操作，**<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。<c:forEach>标签有两种语法格式：**

**语法1，在集合对象中迭代：**

**<c:forEach [var="*varName*"]**

**items="*collection*"**

**[varStatus="*varStatusName*"]**

**[begin="*begin*"] [end="*end*"] [step="*step*"]>**

***body content***

**</c:forEach>**

**语法2，迭代固定的次数：**

**<c:forEach [var="*varName*"]**

**[varStatus="*varStatusName*"]**

**begin="*begin*" end="*end*" [step="*step*"]>**

***body content***

**</c:forEach>**

**<c:forEach>标签的属性说明如表8.6所示。**

**表8.6 <c:forEach>标签的属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| var | false | String | 指定将当前迭代到的元素保存到page这个Web域中的属性名称 |
| items | true | 任何支持的类型 | 将要迭代的集合对象 |
| varStatus | false | String | 指定将代表当前迭代状态信息的对象保存到page这个Web域中的属性名称 |
| begin | true | int | 如果指定items属性，就从集合中的第begin个元素开始进行迭代，begin的索引值从0开始编号；如果没有指定items属性，就从begin指定的值开始迭代，直到end值时结束迭代 |
| end | true | int | 参看begin属性的描述 |
| step | true | int | 指定迭代的步长，即迭代因子的迭代增量 |

**在使用<c:forEach>标签时，需要注意如下几点说明：**

* **如果指定begin属性，其值必须大于或等于零；**
* **如果指定步长（step属性），其值必须大于或等于1；**
* **如果items属性的值为null，则要处理的集合对象为空，这时不执行迭代操作；**
* **如果指定的begin属性的值大于或等于集合对象的长度，不执行迭代操作；**
* **如果指定的end属性的值小于begin属性的值，不执行迭代操作；**

**<c:forEach>标签的items属性的值支持下面的数据类型：**

* 任意类型的数组
* java.util.Collection
* java.util.Iterator
* java.util.Enumeration
* java.util.Map
* String

**items属性还支持与数据库有关的数据类型java.sql.ResultSet(包括javax.sql.RowSet)，这些数据类型将在8.5 节的数据库标签中进行介绍。对字符串的迭代操作通常使用<c:forTokens>标签或JSTL函数，例如fn:split和fn:jion，JSTL函数将在8.7节进行介绍。**

**1．迭代Collection类型的集合对象**

**例程8-13是使用<c:forEach>标签迭代Collection类型的集合对象的一个应用举例。**

**例程8-13 c\_forEach\_collection.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<%@ page import="java.util.\*,org.it315.UserBean" %>**

**<%**

**Collection users = new ArrayList();**

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

**{**

**UserBean user = new UserBean();**

**user.setUserName("user" + i);**

**user.setPassword("guess" + i);**

**users.add(user);**

**}**

**session.setAttribute("users", users);**

**%>**

**<div style="text-align:center">User List**

**<table border="1">**

**<tr><td>用户名</td><td>密码</td></tr>**

**<c:forEach var="user" items="${users}">**

**<tr>**

**<td>${user.userName}</td><td>${user.password}</td>**

**</tr>**

**</c:forEach>**

**</table></div>**

**例程8-13的运行结果如图8.13所示。**



**图8.13**

**2．迭代Map对象**

**使用<c:forEach>标签迭代Map类型的集合对象时，迭代出的每个元素的类型为[Map.Entry](mk:@MSITStore:D:\\toshiba_bak\\JAVA%20CD\\jdk\\jdk150.chm::/jdk150/api/java/util/Map.Entry.html" \o "interface in java.util)，Map.Entry代表Map集合中的一个条目项，其中的getKey()方法可获得条目项的关键字，getValue()方法可获得条目项的值。**

**EL中的requestScope隐含对象**代表request作用域中的所有属性的Map对象**，所以我们可以使用<c:forEach>标签迭代输出EL中的requestScope隐含对象中的所有元素，如例程8-14所示。**

**例程8-14 c\_forEach\_map.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<%**

**request.setAttribute("attr1","value1");**

**request.setAttribute("attr2","value2");**

**%>**

**<div style="text-align:center">Properties(Map)**

**<table border="1">**

**<tr><td>Map的关键字</td><td>Map的对应关键字的值</td></tr>**

**<c:forEach var="entry" items="${requestScope}">**

**<tr><td>${entry.key}</td><td>${entry.value}</td></tr>**

**</c:forEach>**

**</table></div>**

**例程8-14的运行结果如图8.14所示。**



**图8.14**

**3．迭代指定的次数**

**<c:forEach>标签可以按指定的次数重复迭代执行标签体中的内容，使用这种方式迭代时，可以指定迭代的步长。例程8-15中分别演示了指定迭代步长和没有指定迭代步长的情况。**

**例程8-15 c\_forEach\_count.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**从11到16迭代，指定迭代步长为2：**

**<c:forEach var="i" begin="11" end="16" step="2">**

**${i}**

**</c:forEach><br /><hr />**

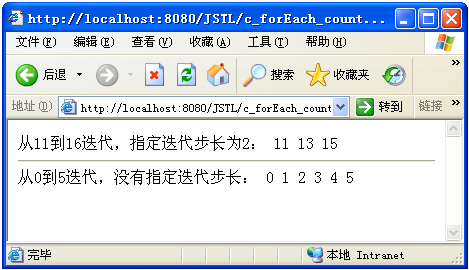
**从0到5迭代，没有指定迭代步长：**

**<c:forEach var="i" begin="0" end="5">**

**${i}**

**</c:forEach>**

**例程8-15的运行结果如图8.15所示。**



**图8.15**

**4．指定迭代集合对象的范围和步长**

**<c:forEach>标签迭代集合类对象时，也可以指定迭代的范围和步长，如例程8-16所示。**

**例程8-16 c\_forEach\_col.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<%@ page import="java.util.\*,org.it315.UserBean" %>**

**<%**

**Collection users = new ArrayList();**

**for(int i=0; i<6; i++)**

**{**

**UserBean user = new UserBean();**

**user.setUserName("user" + i);**

**user.setPassword("guest" + i);**

**users.add(user);**

**}**

**session.setAttribute("users", users);**

**%>**

**<div style="text-align:center">User List(指定迭代范围和步长)**

**<table border="1">**

**<tr><td>用户名</td><td>密码</td></tr>**

**<c:forEach var="user" items="${users}" end="5" step="2">**

**<tr>**

**<td>${user.userName}</td><td>${user.password}</td>**

**</tr>**

**</c:forEach>**

**</table></div>**

**例程8-16的运行结果如图8.16所示。**

********

**图8.16**

**5．获取迭代的状态信息**

**不管是迭代集合对象，还是迭代指定的次数，在迭代时都可以获得**当前的迭代状态信息**。<c:forEach>标签可以**将代表当前迭代状态信息的对象保存到page域**中，varStatus属性**指定了这个对象保存在page域中的属性名称。代表当前迭代状态信息的对象的类型为javax.servlet.jsp.jstl.core.LoopTagStatus，从JSTL规范中可以查看到这个类的详细信息，其中定义了如下一些方法：

* **public java.lang.Integer getBegin()  
  返回为标签设置的begin属性的值，如果没有设置begin属性则返回null**
* **public int getCount()  
  返回当前已循环迭代的次数**
* **public java.lang.Object getCurrent()  
  返回**当前迭代到的元素对象
* **public java.lang.Integer getEnd()  
  返回为标签设置的end属性的值，如果没有设置end属性则返回null**
* **public int getIndex()  
  返回**当前迭代的索引号
* **public java.lang.Integer getStep()  
  返回为标签设置的step属性的值，如果没有设置step属性则返回null**
* **public boolean isFirst()  
  返回**当前是否是第一次迭代操作
* **public boolean isLast()  
  返回**当前是否是最后一次迭代操作

**例程8-17是一个获取迭代状态信息的例子程序。**

**例程8-17 c\_forEach\_col2.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<%@ page import="java.util.\*,org.it315.UserBean" %>**

**<%**

**Collection users = new ArrayList();**

**for(int i=0; i<6; i++)**

**{**

**UserBean user = new UserBean();**

**user.setUserName("user" + i);**

**user.setPassword("guest" + i);**

**users.add(user);**

**}**

**session.setAttribute("users", users);**

**%>**

**<div style="text-align:center">User List**

**<table border="1">**

**<tr><td>用户名</td><td>密码</td><td>index</td>**

**<td>count</td><td>first?</td><td>last?</td></tr>**

**<c:forEach var="user" items="${users}" varStatus="sta" begin="1" step="2">**

**<tr>**

**<td>${user.userName}</td><td>${user.password}</td>**

**<td>${sta.index}</td><td>${sta.count}</td>**

**<td>${sta.first}</td><td>${sta.last}</td>**

**</tr>**

**</c:forEach>**

**</table></div><hr>**

**<div style="text-align:center">迭代固定的次数**

**<table border="1">**

**<tr><td>数值</td><td>index</td><td>count</td>**

**<td>first?</td><td>last?</td></tr>**

**<c:forEach var="i" varStatus="sta1" begin="101" end="103">**

**<tr>**

**<td>${i}</td><td>${sta1.index}</td><td>${sta1.count}</td>**

**<td>${sta1.first}</td><td>${sta1.last}</td>**

**</tr>**

**</c:forEach>**

**</table></div>**

**例程8-17的运行结果如图8-17所示。**

********

**图8.17**

**6．与条件标签结合使用**

**迭代标签可以与条件标签结合使用，对数据进行有条件的迭代，应用举例如例程8-18所示。**

**例程8-18 c\_forEach\_com.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<div style="text-align:center">**

**<table border="1"><tr><td>number</td><td>count</td><td>even|odd</td></tr>**

**<c:forEach var="i" begin="11" end="13" varStatus="status">**

**<tr><td>${i}</td><td>${status.count}</td><td>**

**<c:choose>**

**<c:when test="${status.count % 2 == 0}">**

**偶数行**

**</c:when>**

**<c:otherwise>**

**奇数行**

**</c:otherwise>**

**</c:choose></td></tr>**

**</c:forEach>**

**<table></div>**

**例程8-18的运行结果如图8.18所示。**

********

**图8.18**

### 8.3.8 <c:forTokens>标签

<c:forTokens>标签专门用于实现类似java.util.StringTokenizer类的迭代功能，但它是以单个字符作为分隔符，同时可以指定多个字符作为多个并行的分隔符。<c:forTokens>标签的语法格式如下：

<c:forTokens items="*stringOfTokens*" delims="*delimiters*"

[var="*varName*"]

[varStatus="*varStatusName*"]

[begin="*begin*"] [end="*end*"] [step="*step*"]>

*body content*

</c:forTokens>

<c:forTokens>标签的属性说明如表8.7所示。

表8.7 c:forTokens

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | **是否支持EL** | **属性类型** | **属 性 描 述** |
| var | false | String | 指定将当前迭代出的子字符串保存到page这个Web域中的属性名称 |
| items | true | String | 将要迭代的字符串 |
| delims | true | String | 指定一个或多个分隔符 |
| varStatus | false | String | 指定将代表当前迭代状态信息的对象保存到page这个Web域中的属性名称，代表当前迭代的状态信息的对象的类型为javax.servlet.jsp.jstl.core.LoopTagStatus，从JSTL规范中可以查看这个类的详细信息 |
| begin | true | int | 指定从第begin个子字符串开始进行迭代，begin的索引值从0开始编号 |
| end | true | int | 指定迭代到第begin个子字符串，begin的索引值从0开始编号 |
| step | true | int | 指定迭代的步长，即每次迭代后的迭代因子增量 |

在使用<c:forTokens>标签时，**需要注意如下几点说明**：

* **如果指定begin属性，其值必须大于或等于零。**
* **如果指定步长（step属性），其值必须大于或等于1。**
* **如果指定的end属性的值小于begin属性的值，不执行迭代操作。**

例程8-19是一个使用<c:forTokens>标签的例子程序。

例程8-19 c\_forTokens.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

使用"|"作为分隔符<br />

<c:forTokens var="token" items="spring,summer|autumn,winter" delims="|">

${token}&copy;

</c:forTokens><br />

使用"|"和","作为分隔符<br />

<c:forTokens var="token" items="spring,summer|autumn,winter" delims="|," end="3">

${token}&copy;

</c:forTokens><br />

使用"-"作为分隔符<br />

<c:forTokens var="token" items="year--season--month-week" delims="-">

${token}&copy;

</c:forTokens>

例程8-19**的**运行结果如图8.19 所示。

****

图8.19

### 8.3.9 URL相关的标签概述

**JSTL核心标签库中提供了如下一些与URL操作相关的标签：**

* **<c:import>**
* <c:url>
* <c:redirect>
* <c:param>

**在举例讲解上面的某些标签时，要引用另外一个JSP文件，这里先创建好这个JSP文件，如例程8-20所示。**

**例程8-20 register.jsp**

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<%**

**String name = request.getParameter("name");**

**name = new String(name.getBytes("iso-8859-1"),"gb2312");**

**session.setAttribute("name",name);**

**String country = request.getParameter("country");**

**country = new String(country.getBytes("iso-8859-1"),"gb2312");**

**session.setAttribute("country",country);**

**%>**

**name=${name};**

**country=${country}<br />**

### 8.3.10 <c:param>标签

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:param>标签在为一个URL地址附加参数时，将自动对参数值进行URL编码，例如，如果传递的参数值为“中国”，则将其转换为“%d6%d0%b9%fa”后再附加到URL地址后面，这也就是使用<c:param>标签的最大好处。<c:param>标签有两种语法格式：

**语法1，使用value属性指定参数的值：**

<c:param name="*name*" value="*value*" />

**语法2，在标签体中指定参数的值：**

<c:param name="*name*">

*parameter value*

</c:param>

<c:param>标签的属性说明如表8.8所示。

表8.8 <c:param>标签的属性



### 8.3.11 <c:url>标签

**<c:url>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，详细细节请参看笔者编著的《深入体验java Web开发内幕——核心基础》一书中的第7.4.8的讲解。<c:url>标签有两种语法格式：**

**语法1，没有标签体的情况：**

**<c:url value="*value*"**

**[context="*context*"]**

**[var="*varName*"]**

**[scope="{page|request|session|application}"] />**

**语法2，有标签体的情况，在标签体中指定构造的URL的参数：**

**<c:url value="*value*"**

**[context="*context*"]**

**[var="*varName*"]**

**[scope="{page|request|session|application}"]>**

***<c:param>标签***

**</c:url>**

**<c:url>标签的属性说明如表8.9所示。**

**表8.9 <c:url>标签的属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| value | true | String | 指定要构造的URL |
| context | true | String | 当要使用相对路径导入同一个服务器下的其他WEB应用程序中的URL地址时，context属性指定其他WEB应用程序的名称 |
| var | false | String | 指定将构造出的URL结果保存到Web域中的属性名称 |
| scope | false | String | 指定将构造出的URL结果保存到哪个Web域中 |

**value属性所指定的URL可以是相对路径和绝对路径，其具体细节与8.3.12节的<c:import>标签的url属性的细节相同。**使用<c:url>标签构造URL时，可以通过嵌套的<c:param>标签指定参数，或在value属性中直接指定参数。

例程8-21是一个使用**<c:url>标签**的例子程序。

例程8-21 c\_url.jsp

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

<%@ page contentType="text/html;charset=gb2312" %>

使用绝对路径构造URL:

<c:url value="http://localhost:8080/JSTL/URL/register.jsp" var="myUrl1">

<c:param name="name" value="张三" />

<c:param name="country" value="${param.country}" />

</c:url>

<a href="${myUrl1}">Register1</a><hr />

使用相对当前JSP页面的路径构造URL:

<c:url value="register.jsp?name=wangwu&country=France" var="myUrl2" />

<a href="${myUrl2}">Register2</a><hr />

使用相对当前WEB应用的路径构造URL:

<c:url value="/URL/register.jsp?name=zhaoliu&country=England" var="myUrl3" />

<a href="${myUrl3}">Register3</a>

在浏览器地址栏输入如下地址访问例程8-21：

http://localhost:8080/JSTL/c\_url.jsp?country=China

查看例程8-21的运行结果的源文件，内容如例程8-22所示。

例程8-22 c\_url.jsp运行结果的源文件

使用绝对路径构造URL:

<a href="http://localhost:8080/JSTL/URL/register.jsp?name=%d5%c5%c8%fd&country=China">Register1</a><hr />

使用相对当前JSP页面的路径构造URL:

<a href="register.jsp?name=wangwu&country=France">Register2</a><hr />

使用相对当前WEB应用的路径构造URL:

<a href="/JSTL/URL/register.jsp?name=zhaoliu&country=England">Register3</a>

### 8.3.12 <c:import>标签

<c:import>标签用于在JSP页面中导入一个URL地址指向的资源内容，其作用有点类似<jsp:include>这个JSP标准标签，但功能更强大。<c:import>标签标签有两种语法格式：

语法1，将URL地址指向的资源内容以字符串形式输出或以字符串形式保存到一个变量中：

<c:import url="*url*"

[context="*context*"]

[var="*varName*"]

[scope="{page|request|session|application}"]

[charEncoding="*charEncoding*"]>

*optional body content for <c:param> subtags*

</c:import>

语法2，将URL地址指向的资源内容保存到一个Reader对象中：

<c:import url="*url*"

[context="*context*"]

varReader="*varReaderName*"

[charEncoding="*charEncoding*"]>

*body content where varReader is consumed by another action*

</c:import>

<c:import>标签的属性说明如表8.10所示。

表8.10 <c:import>标签的属性

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| url | true | String | 指定要导入的资源的URL地址 |
| context | true | String | 当要使用相对路径导入同一个服务器下的其他WEB应用程序中的资源时，context属性指定其他WEB应用程序的名称 |
| var | false | String | 指定将导入的资源内容保存到Web域中的属性名称 |
| scope | false | String | 指定将导入的资源内容保存到哪个Web域中 |
| charEncoding | true | String | 将导入的资源内容转换成字符串时所使用的字符集编码 |
| varReader | false | String | 指定将导入的资源内容保存到page域中的一个java.io.Reader对象中，varReader属性指定了该Reader对象在page这个Web域中的属性名称。 |

使用<c:import>标签导入其他资源文件时，如果被导入的资源文件中含有非ASCII码字符，必须注意指定charEncoding属性，否则可以不设置这个属性。

当使用第1种语法格式时，如果指定了var属性，导入的资源内容以字符串形式保存到一个变量中，var属性指定了这个变量保存在Scope属性指定的Web域中的名称；如果没有指定var属性，导入的资源内容将以字符串形式直接输出。第1种语法格式的标签体内中可以嵌套<c:param>标签来给导入的资源传递参数。

使用第2种语法格式时，导入的资源内容保存到page域中的一个java.io.Reader对象中，varReader属性指定了该Reader对象在page这个Web域中的属性名称。因为<c:import>标签在标签结束时将关闭Reader对象对应的输入流，所以varReader属性指定的变量只在标签内有效。第2种语法格式的标签体内中应该且只能嵌套调用varReader属性指定的reader对象的其他标签。当使用第2种语法格式时，<c:import>标签体内不能嵌套<c:param>标签，如果要给导入的资源传递参数，则必须在url属性值中设置好这些参数，这时可以使用<c:url>标签来创建一个带参数的URL地址，此时如果有必要，<c:import>标签将删除导入资源中的所有URL重写后产生的session id信息。

使用<c:import>标签导入外部资源文件时，url属性可以设置为被导入资源的相对URL或绝对URL。

**1．使用相对URL导入其他资源文件**

**（1）当被导入的外部资源文件与当前JSP文件属于同一个WEB应用程序时，JSP容器对资源路径的处理与<jsp:include>标签相同，这时路径可以以“/”字符开始，也可以以文件名或目录名开始。如果路径以“/”字符开始，则被导入资源文件的路径是相对于JSP页面所在的WEB应用的根路径；如果路径以文件名或目录名开始，则被导入的资源文件的路径就是相对于当前的JSP页面的路径。**

**（2）当被导入的外部资源文件与JSP文件属于同一个服务器下的不同的WEB应用时，如果要使用相对路径，路径必须以“/”开始。此时，<c:import>标签的context属性必须被设置为被导入的资源文件所在的WEB应用程序的名称。注意：使用相对路径导入其它WEB应用程序中的资源时，必须在<TOMCAT\_HOME>\conf\server.xml配置文件中将使用<c:import>标签的WEB应用程序的<Context>元素的crossContext属性设置为true，如下所示：**

**<Context path="/JSTL" docBase="JSTL" crossContext="true"/>**

**2．使用绝对URL导入其他资源**

**使用绝对路径导入外部资源文件时，即使被导入的文件与当前JSP文件属于同一个WEB应用程序，被导入的资源文件也不会共享当前JSP页面的request和session对象。因为在这种情况下，当前JSP文件所属的Web服务器充当了一个Http客户端程序，去访问绝对路径所指定的外部资源文件，然后将响应结果的实体内容加入到当前JSP页面中，此时，访问当前JSP页面的客户端是浏览器，而访问绝对路径所指定的外部资源文件的客户端是当前JSP文件所属的Web服务器，这显然属于两个不同的客户端，所以，被导入的资源文件和当前JSP页面不可能共享request和session对象。**

**例程8-23是使用<c:import>标签的一个应用举例。**

例程8-23 c\_import.jsp

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**使用相对路径导入同一个WEB应用中的资源:<br />**

**<c:import url="/register.jsp?name=zhangsan">**

**<c:param name="name" value="zxx" />**

**</c:import><hr />**

**使用相对路径导入同一个服务器下的不同WEB应用中的资源:<br />**

**<c:import url="/hello.jsp" context="/EL" /><hr />**

**使用绝对路径导入资源示例1:**

**<c:import url="http://localhost:8080/EL/hello.jsp" /><hr />**

**使用绝对路径导入资源示例2:**

**<c:import url="http://localhost:8080/JSTL/register.jsp" var="myImport"**

**charEncoding="gb2312">**

**<c:param name="name" value="${param.name}" />**

**<c:param name="country" value="中国" />**

**</c:import>**

**${myImport}**

**在浏览器地址栏输入如下地址访问例程8-23，其运行结果如图8.20 所示：**

**http://localhost:8080/JSTL/c\_import.jsp?name=zxx&country=China**

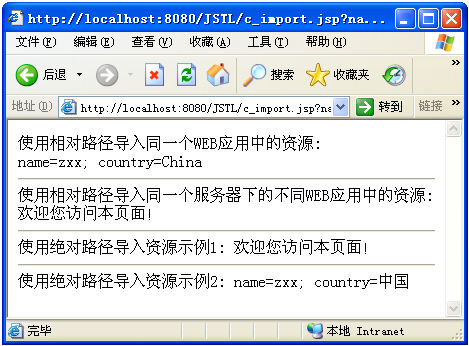
****

图8.20

在<c:import>标签中使用相对路径导入其他资源时，其工作原理与<jsp:include>标签相同，所以，在被导入的资源文件中可以获得传递给当前JSP页面的请求参数，例如，例程8-23中的第一个<c:import>标签并没有在目标url后增加country参数，但是，在register.jsp页面中获得了country参数。

### 8.3.13 <c:redirect>标签

**<c:redirect>标签用于将当前的访问请求转发或重定向到其他资源，它可以根据url属性所指定的地址，执行类似**<jsp:forward>这个JSP标准标签的功能，**将访问请求转发到其他资源**；或执行response.sendRedirect()方法的功能，**将访问请求重定向到其他资源。<c:redirect>标签有两种语法格式：**

**语法1，没有标签体的情况：**

**<c:redirect url="*value*" [context="*context*"] />**

**语法2，有标签体的情况，在标签体中指定重定向时的参数：**

**<c:redirect url="*value*" [context="*context*"]>**

***<c:param>subtags***

**</c:redirect>**

**<c:redirect>标签的属性说明如表8.11所示。**

**表8.11 <c:redirect>标签的属性**

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| url | true | String | 指定要转发或重定向到的目标资源的URL地址 |
| context | true | String | 当要使用相对路径重定向到同一个服务器下的其他WEB应用程序中的资源时，context属性指定其他WEB应用程序的名称 |

url属性指定将要重定向的资源的URL时，可以使用相对路径和绝对路径，**其具体细节与**<c:import>标签的url属性相同。例程8-24是一个使用<c:redir**ect>标签的**应用例子。

例程8-24 c\_redirect.jsp

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

**<%@ page contentType="text/html;charset=gb2312" %>**

**<c:url value="http://localhost:8080/JSTL/URL/register.jsp" var="myUrl">**

**<c:param name="name" value="张三" />**

**<c:param name="country" value="中国" />**

**</c:url>**

**<c:redirect url="${myUrl}" />**

例程8-24的运行结果如图8.21 所示。

****