

第六章 Expression Language

6-1 EL 简介

6-1 EL 简介

EL 全名为 Expression Language，它原本是 JSTL 1.0 为方便存取数据所自定义的语言。当时 EL 只能在 JSTL 标签中使用，如下：

```
<c:out value="${ 3 + 7}">
```

程序执行结果为 10。但是你却不能直接在 JSP 网页中使用：

```
<p>Hi ! ${ username }</p>
```

到了 JSP 2.0 之后，EL 已经正式纳入成为标准规范之一。因此，只要是支持 Servlet 2.4 / JSP 2.0 的 Container，就都可以在 JSP 网页中直接使用 EL 了。

除了 JSP 2.0 建议使用 EL 之外，JavaServer Faces(JSR-127) 也考虑将 EL 纳入规范，由此可知，EL 如今已经是一项成熟、标准的技术。

注意

假若您所用的 Container 只支持 Servlet 2.3/JSP 1.2，如：Tomcat 4.1.29，您就不能在 JSP 网页中直接使用 EL，必须安装支持 Servlet 2.4 / JSP 2.0 的 Container。

6-2 EL 语法

EL 语法很简单，它最大的特点就是使用上很方便。接下来介绍 EL 主要的语法结构：

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

所有 EL 都是以 \${ 为起始、以 } 为结尾的。上述 EL 范例的意思是：从 Session 的范围中，取得用户的性别。假若依照之前 JSP Scriptlet 的写法如下：

```
User user = (User)session.getAttribute("user");  
String sex = user.getSex( );
```

两者相比较之下，可以发现 EL 的语法比传统 JSP Scriptlet 更为方便、简洁。

6-2-1 . 与 [] 运算符

EL 提供 . 和 [] 两种运算符来存取数据。下列两者所代表的意思是一样的：

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

等于

```
${sessionScope.user["sex"]}
```

. 和 [] 也可以同时混合使用，如下：

```
${sessionScope.shoppingCart[0].price}
```

回传结果为 shoppingCart 中第一项物品的价格。

不过，以下两种情况，两者会有差异：

(1) 当要存取的属性名称中包含一些特殊字符，如 . 或 - 等并非字母或数字的符号，就一定要使用 []，例如：

```
${user.My-Name }
```

上述是不正确的方式，应当改为：

```
${user["My-Name" ] }
```

(2) 我们来考虑下列情况：

```
${sessionScope.user[data]}
```

此时，data 是一个变量，假若 data 的值为"sex"时，那上述的例子等于\${sessionScope.user.sex}；假若 data 的值为"name"时，它就等于\${sessionScope.user.name}。因此，如果要动态取值时，就可以用上述的方法来做，但 . 无法做到动态取值。

接下来，我们更详细地来讨论一些情况，首先假设有一个 EL：

```
${expr-a[expr-b]}
```

(1) 当 `expr-a` 的值为 `null` 时，它会回传 `null`。

(2) 当 `expr-b` 的值为 `null` 时，它会回传 `null`。

(3) 当 `expr-a` 的值为一个 `Map` 类型时：

- 假若 `!value-a.containsKey(value-b)` 为真，则回传 `null`。

- 否则回传 `value-a.get(value-b)`。

(4) 当 `expr-a` 的值为 `List` 或 `array` 类型时：

- 将 `value-b` 的值强制转型为 `int`，假若不能转型为 `int` 时，会产生 `error`。

- 然后，假若 `value-a.get(value-b)` 或 `Array.get(value-a, value-b)` 产生 `ArrayIndexOutOfBoundsException` 或 `IndexOutOfBoundsException` 时，则回传 `null`。

- 假若 `value-a.get(value-b)` 或 `Array.get(value-a, value-b)` 产生其他的异常时，则会产生 `error`。

- 最后都没有任何异常产生时，回传 `value-a.get(value-b)` 或 `Array.get(value-a, value-b)`。

(5) 当 `expr-a` 的值为 `JavaBean` 对象时：

- 将 `value-b` 的值强制转型为 `String`。

- 假若 `getter` 产生异常时，则会产生 `error`。若没有异常产生时，则回传 `getter` 的结果。

6-2-2 EL 变量

EL 存取变量数据的方法很简单，例如：`${username}`。它的意思是取出某一范围中名称为 `username` 的变量。因为我们并没有指定哪一个范围的 `username`，所以它的默认值会先从 `Page` 范围找，假如找不到，再依序到 `Request`、`Session`、`Application` 范围。假如途中找到 `username`，就直接回传，不再继续找下去，但是假如全部的范围都没有找到时，就回传 `null`（见表 6-1）：

表 6-1

属性范围	在 EL 中的名称
Page	PageScope
Request	RequestScope
Session	SessionScope
Application	ApplicationScope

自动搜索顺序

我们也可以指定要取出哪一个范围的变量（见表 6-2）：

表 6-2

范 例	说 明
<code>\${pageScope.username}</code>	取出 Page 范围的 username 变量
<code>\${requestScope.username}</code>	取出 Request 范围的 username 变量
<code>\${sessionScope.username}</code>	取出 Session 范围的 username 变量
<code>\${applicationScope.username}</code>	取出 Application 范围的 username 变量

其中，`pageScope`、`requestScope`、`sessionScope` 和 `applicationScope` 都是 EL 的隐含对象，由它们的名称可以很容易猜出它们所代表的意思，例如：`${sessionScope.username}` 是取出 Session 范围的 username 变量。这种写法是不是比之前 JSP 的写法：

```
String username = (String) session.getAttribute("username");
```

容易、简洁许多。有关 EL 隐含对象在 6-3 节中有更详细的介绍。

6-2-3 自动转变类型

EL 除了提供方便存取变量的语法之外，它另外一个方便的功能就是：**自动转变类型**，我们来看下面这个范例：

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

假若窗体传来 `count` 的值为 10 时，那么上面的结果为 30。之前没接触过 JSP 的读者可能会认为上面的例子是理所当然的，但是在 JSP 1.2 之中不能这样做，原因是从窗体所传来的值，它们的类型一律是 `String`，所以当你接收之后，必须再将它转为其他类型，如：`int`、`float` 等等，然后才能执行一些数学运算，下面是之前的做法：

```
String str_count = request.getParameter("count");
```

```
int count = Integer.parseInt(str_count);  
  
count = count + 20;
```

接下来再详细说明 EL 类型转换的规则：

(1) 将 A 转为 String 类型

- 假若 A 为 String 时：回传 A
- 否则，当 A 为 null 时：回传 ""
- 否则，当 A.toString() 产生异常时：错误！
- 否则，回传 A.toString()

(2) 将 A 转为 Number 类型的 N

- 假若 A 为 null 或 "" 时：回传 0
- 假若 A 为 Character 时：将 A 转为 new Short((short)a.charValue())
- 假若 A 为 Boolean 时：错误！
- 假若 A 为 Number 类型和 N 一样时：回传 A
- 假若 A 为 Number 时：
 - 假若 N 是 BigInteger 时：
 - 假若 A 为 BigDecimal 时：回传 A.toBigInteger()
 - 否则，回传 BigInteger.valueOf(A.longValue())
 - 假若 N 是 BigDecimal 时：
 - 假若 A 为 BigInteger 时：回传 A.toBigDecimal()
 - 否则，回传 BigDecimal.valueOf(A.doubleValue())
 - 假若 N 为 Byte 时：回传 new Byte(A.byteValue())
 - 假若 N 为 Short 时：回传 new Short(A.shortValue())
 - 假若 N 为 Integer 时：回传 new Integer(A.intValue())

- 假若 N 为 Long 时: 回传 `new Long(A.longValue())`
- 假若 N 为 Float 时: 回传 `new Float(A.floatValue())`
- 假若 N 为 Double 时: 回传 `new Double(A.doubleValue())`
- 否则, 错误!

● 假若 A 为 String 时:

- 假若 N 是 BigDecimal 时:
 - 假若 `new BigDecimal(A)` 产生异常时: 错误!
 - 否则, 回传 `new BigDecimal(A)`
- 假若 N 是 BigInteger 时:
 - 假若 `new BigInteger(A)` 产生异常时: 错误!
 - 否则, 回传 `new BigInteger(A)`
- 假若 `N.valueOf(A)` 产生异常时: 错误!
- 否则, 回传 `N.valueOf(A)`

● 否则, 错误!

(3) 将 A 转为 Character 类型

- 假若 A 为 null 或 "" 时: 回传 `(char)0`
- 假若 A 为 Character 时: 回传 A
- 假若 A 为 Boolean 时: 错误!
- 假若 A 为 Number 时: 转换为 Short 后, 然后回传 Character
- 假若 A 为 String 时: 回传 `A.charAt(0)`
- 否则, 错误!

(4) 将 A 转为 Boolean 类型

- 假若 A 为 null 或 "" 时：回传 false
- 否则，假若 A 为 Boolean 时：回传 A
- 否则，假若 A 为 String，且 Boolean.valueOf(A) 没有产生异常时：回传 Boolean.valueOf(A)
- 否则，错误！

6-2-4 EL 保留字

EL 的保留字如表 6-3:

表 6-3

And	eq	gt	true
Or	ne	le	false
No	lt	ge	null
instanceof	empty	div	mod

所谓保留字的意思是指变量在命名时，应该避开上述的名字，以免程序编译时发生错误。

6-3 EL 隐含对象

笔者在“第五章：隐含对象（Implicit Object）”中，曾经介绍过 9 个 JSP 隐含对象，而 EL 本身也有自己的隐含对象。EL 隐含对象总共有 11 个（见表 6-4）:

表 6-4

隐含对象	类 型	说 明
PageContext	javax.servlet.ServletContext	表示此 JSP 的 PageContext
PageScope	java.util.Map	取得 Page 范围的属性名称所对应的值
RequestScope	java.util.Map	取得 Request 范围的属性名称所对应的值
sessionScope	java.util.Map	取得 Session 范围的属性名称所对应的值
applicationScope	java.util.Map	取得 Application 范围的属性名称所对应的值
param	java.util.Map	如同 ServletRequest.getParameter(String name)。回传 String 类型的值

续表

隐含对象	类 型	说 明
paramValues	java.util.Map	如同

隐含对象	类 型	说 明
		<code>ServletRequest.getParameterValues(String name)</code> 。回传 <code>String []</code> 类型的值
<code>header</code>	<code>java.util.Map</code>	如同 <code>ServletRequest.getHeader(String name)</code> 。回传 <code>String</code> 类型的值
<code>headerValues</code>	<code>java.util.Map</code>	如同 <code>ServletRequest.getHeaders(String name)</code> 。回传 <code>String []</code> 类型的值
<code>cookie</code>	<code>java.util.Map</code>	如同 <code>HttpServletRequest.getCookies()</code>
<code>initParam</code>	<code>java.util.Map</code>	如同 <code>ServletContext.getInitParameter(String name)</code> 。回传 <code>String</code> 类型的值

这 11 个隐含对象(Implicit Object)，笔者将它分成三类：

1. 与范围有关的隐含对象

`applicationScope`

`sessionScope`

`requestScope`

`pageScope`

2. 与输入有关的隐含对象

`param`

`paramValues`

3. 其他隐含对象

`cookie`

`header`

`headerValues`

`initParam`

`pageContext`

接下来笔者会依照上面的分类顺序，为读者介绍这些隐含对象。

6-3-1 属性(Attribute)与范围(Scope)

与范围有关的 EL 隐含对象包含以下四个：`pageScope`、`requestScope`、`sessionScope` 和 `applicationScope`，它们基本上就和 JSP 的 `pageContext`、`request`、`session` 和 `application` 一样，所以笔者在这里只稍略说明。不过必须注意的是，这四个隐含对象只能用来取得范围属性值，即 JSP 中的 `getAttribute(String name)`，却不能取得其他相关信息，例如：JSP 中的 `request` 对象除可以存取属性之外，还可以取得用户的请求参数或表头信息等等。但是在 EL 中，它就只能单纯用来取得对应范围的属性值，例如：我们要在 `session` 中储存一个属性，它的名称为 `username`，在 JSP 中使用 `session.getAttribute("username")` 来取得 `username` 的值，但是在 EL 中，则是使用 `${sessionScope.username}` 来取得其值的。接下来分别对这四个隐含对象做简短的说明：

● `pageScope`

范围和 JSP 的 `Page` 相同，也就是单单一页 JSP `Page` 的范围 (Scope)。

● `requestScope`

范围和 JSP 的 `Request` 相同，`requestScope` 的范围是指从一个 JSP 网页请求到另一个 JSP 网页请求之间，随后此属性就会失效。

● `sessionScope`

范围和 JSP `Scope` 中的 `session` 相同，它的属性范围就是用户持续在服务器连接的时间。

● `applicationScope`

范围和 JSP `Scope` 中的 `application` 相同，它的属性范围是从服务器一开始执行服务，到服务器关闭为止。

6-3-2 与输入有关的隐含对象

与输入有关的隐含对象有两个：`param` 和 `paramValues`，它们是 EL 中比较特别的隐含对象。一般而言，我们在取得用户的请求参数时，可以利用下列方法：

```
request.getParameter(String name)
request.getParameterValues(String name)
```

在 EL 中则可以使用 `param` 和 `paramValues` 两者来取得数据。

```
${param.name}

${paramValues.name}
```

这里 `param` 的功能和 `request.getParameter(String name)` 相同，而 `paramValues` 和 `request.getParameterValues(String name)` 相同。如果用户填了一个表格，表格名称为 `username`，我们就可以使用 `${param.username}` 来取得用户填入的值。

为了让读者更加了解 `param` 和 `paramValues` 隐含对象的使用，再来看下面这个范例。此范例共有两个文件，分别为给用户输入值用的 `Param.html` 和显示出用户所传之值的 `Param.jsp`。

Param.html

```
<html>

<head>

    <title>CH6 - Param.html</title>

</head>

<body>

<h2>EL 隐含对象 param、paramValues</h2>

<form method = "post" action = "Param.jsp">

<p>姓名: <input type="text" name="username" size="15" /></p>

<p>密码: <input type="password" name="password" size="15" /></p>

<p>性别: <input type="radio" name="sex" value="Male" checked/> 男

        <input type="radio" name="sex" value="Female" /> 女</p>

<p>年龄:
```

```
<select name="old">

<option value="10">10 - 20</option>

<option value="20" selected>20 - 30</option>

<option value="30">30 - 40</option>

<option value="40">40 - 50</option>

</select>

</p>

<p>兴趣:

<input type="checkbox" name="habit" value="Reading"/>看书

<input type="checkbox" name="habit" value="Game"/>玩游戏

<input type="checkbox" name="habit" value="Travel"/>旅游

<input type="checkbox" name="habit" value="Music"/>听音乐

<input type="checkbox" name="habit" value="Tv"/>看电视

</p>

<p>

<input type="submit" value="传送"/>

<input type="reset" value="清除"/>

</p>

</form>

</body>

</html>
```

Param.html 的执行结果如图 6-1 所示。当我们把窗体填好后按下传送钮，它将会把信息传送到 Param.jsp 做处理。



图 6-1 Param.html 的执行结果，并填入信息

接下来，Param.jsp 接收由 Param.html 传来的信息，并且将它显示出来：

Param.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>

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

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

<html>

<head>

    <title>CH6 - Param.jsp</title>

</head>

<body>

    <h2>EL 隐含对象 param、paramValues</h2>
```

```
<fmt:requestEncoding value="GB2312" />

姓名:  ${param.username}</br>

密码:  ${param.password}</br>

性别:  ${param.sex}</br>

年龄:  ${param.old}</br>

兴趣:  ${paramValues.habit[0]}

      ${paramValues.habit[1]}

</body>

</html>
```

由 Param.html 窗体传过来的值，我们必须指定编码方式，才能够确保 Param.jsp 能够顺利接收中文，传统的做法为：

```
<%

    request.setCharacterEncoding("GB2312");

%>
```

假若是使用 JSTL 写法时，必须使用 I18N 格式处理的标签库，如下：

```
<fmt:requestEncoding value="GB2312" />
```

Param.jsp 主要使用 EL 的隐含对象 param 来接收数据。但是必须注意：假若要取得多重选择的复选框的值时，必须使用 paramValues，例如：使用 paramValues 来取得“兴趣”的值，不过这里笔者最多只显示两笔“兴趣”的值：

```
${param.username}

.....

${paramValues.habit[0]}

${paramValues.habit[1]}
```

有关 JSTL 的使用，第七章有更加详细的说明。图 6-2 是 Param.jsp 的执行结果：

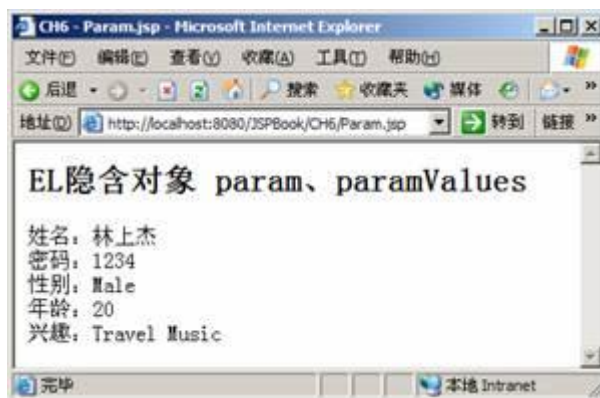


图 6-2 Param.jsp 的执行结果

6-3-3 其他隐含对象

介绍完上面六个隐含对象后，接下来将介绍最后五个隐含对象。

● cookie

所谓的 cookie 是一个小小的文本文件，它是以 key、value 的方式将 Session Tracking 的内容记录在这个文本文件内，这个文本文件通常存在于浏览器的暂存区内。JSTL 并没有提供设定 cookie 的动作，因为这个动作通常都是后端开发者必须去做的事情，而不是交给前端的开发者。假若我们在 cookie 中设定一个名称为 userCountry 的值，那么可以使用 `${cookie.userCountry}` 来取得它。

● header 和 headerValues

header 储存用户浏览器和服务端用来沟通的数据，当用户要求服务端的网页时，会送出一个记载要求信息的标头文件，例如：用户浏览器的版本、用户计算机所设定的区域等其他相关数据。假若要取得用户浏览器的版本，即 `${header["User-Agent"]}`。另外在鲜少机会下，有可能同一标头名称拥有不同的值，此时必须改为使用 headerValues 来取得这些值。

注意

因为 User-Agent 中包含“-”这个特殊字符，所以必须使用“[]”，而不能写成 `$(header.User-Agent)`。

● initParam

就像其他属性一样，我们可以自行设定 web 站台的环境参数(Context)，当我们想取得这些参数

时，可以使用 `initParam` 隐含对象去取得它，例如：当我们在 `web.xml` 中设定如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"

  version="2.4">

:

  <context-param>

    <param-name>userid</param-name>

    <param-value>mike</param-value>

  </context-param>

:

</web-app>
```

那么我们就可以直接使用 `${initParam.userid}` 来取得名称为 `userid`，其值为 `mike` 的参数。下面是之前的做法：

```
String userid = (String)application.getInitParameter("userid");
```

● `pageContext`

我们可以使用 `${pageContext}` 来取得其他有关用户要求或页面的详细信息。表 6-5 列出了几个比较常用的部分。

表 6-5

Expression	说 明
<code>\${pageContext.request.queryString}</code>	取得请求的参数字符串
<code>\${pageContext.request.requestURL}</code>	取得请求的 URL，但不包括请求之参数字符串
<code>\${pageContext.request.contextPath}</code>	服务的 web application 的名称

<code>\${pageContext.request.method}</code>	取得 HTTP 的方法(GET、POST)
<code>\${pageContext.request.protocol}</code>	取得使用的协议(HTTP/1.1、HTTP/1.0)
<code>\${pageContext.request.remoteUser}</code>	取得用户名称
<code>\${pageContext.request.remoteAddr}</code>	取得用户的 IP 地址
<code>\${pageContext.session.new}</code>	判断 session 是否为新的, 所谓新的 session, 表示刚由 server 产生而 client 尚未使用
<code>\${pageContext.session.id}</code>	取得 session 的 ID
<code>\${pageContext.servletContext.serverInfo}</code>	取得主机端的服务信息

我们来看下面这个范例: `pageContext.jsp`, 相信对读者来说能更加了解 `pageContext` 的用法。

`pageContext.jsp`

```
<%@ page contentType="text/html; charset=GB2312" %>

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

<html>

<head>

    <title>CH6 - pageContext.jsp</title>

</head>

<body>

<h2>EL 隐含对象 pageContext</h2>

\${pageContext.request.queryString}:${pageContext.request.queryString}</br>

\${pageContext.request.requestURL}:${pageContext.request.requestURL}</br>

\${pageContext.request.contextPath}:${pageContext.request.contextPath}</br>

\${pageContext.request.method}:${pageContext.request.method}</br>

\${pageContext.request.protocol}:${pageContext.request.protocol}</br>
```



```
\${pageContext.request.remoteUser}:${pageContext.request.remoteUser}</br>

\${pageContext.request.remoteAddr }:${pageContext.request.remoteAddr}</br>

\${pageContext.session.new}:${pageContext.session.new}</br>

\${pageContext.session.id}:${pageContext.session.id}</br>

</body>

</html>
```

pageContext.jsp 的执行结果如图 6-3，执行时必须在 pageContext.jsp 之后加上?test=1234，即 PageContext.jsp?test=1234，这样`${pageContext.request.queryString}`才会显示 test=1234。



图 6-3 pageContext.jsp 的执行结果

注意

因为 `${}` 在 JSP 2.0 中是特殊字符，JSP 容器会自动将它当做 EL 来执行，因此，假若要显示 `${}` 时，必须在 `$` 前加上 `\`，如：`\${ XXXXX }`

6-4 EL 算术运算符

EL 算术运算符主要有以下五个（见表 6-6）：

表 6-6

算术运算符	说 明	范 例	结 果
+	加	<code>\${ 17 + 5 }</code>	22
-	减	<code>\${ 17 - 5 }</code>	12

*	乘	<code>\${ 17 * 5 }</code>	85
/ 或 div	除	<code>\${ 17 / 5 }</code> 或 <code>\${ 17 div 5 }</code>	3
% 或 mod	余数	<code>\${ 17 % 5 }</code> 或 <code>\${ 17 mod 5 }</code>	2

接下来，我们依照下列几种情况，详细说明 EL 算术运算符的规则：

(1) `A {+ , - , *}` B

- 假若 A 和 B 为 `null`：回传 `(Long)0`
- 假若 A 或 B 为 `BigDecimal` 时，将另一个也转为 `BigDecimal`，则：
 - 假若运算符为 `+` 时：回传 `A.add(B)`
 - 假若运算符为 `-` 时：回传 `A.subtract(B)`
 - 假若运算符为 `*` 时：回传 `A.multiply(B)`
- 假若 A 或 B 为 `Float`、`Double` 或包含 `e / E` 的字符串时：
 - 假若 A 或 B 为 `BigInteger` 时，将另一个转为 `BigDecimal`，然后依照运算符执行运算
 - 否则，将两者皆转为 `Double`，然后依照运算符执行运算
- 假若 A 或 B 为 `BigInteger` 时，将另一个也转为 `BigInteger`，则：
 - 假若运算符为 `+` 时：回传 `A.add(B)`
 - 假若运算符为 `-` 时：回传 `A.subtract(B)`
 - 假若运算符为 `*` 时：回传 `A.multiply(B)`
- 否则，将 A 和 B 皆转为 `Long`，然后依照运算符执行运算
- 假若运算结果产生异常时，则错误！

(2) `A {/ , div}` B

- 假若 A 和 B 为 `null`：回传 `(Long)0`
- 假若 A 或 B 为 `BigDecimal` 或 `BigInteger` 时，皆转为 `BigDecimal`，然后回传 `A.divide(B, BigDecimal.ROUND_HALF_UP)`

- 否则，将 A 和 B 皆转为 Double，然后依照运算符执行运算
- 假若运算结果产生异常时，则错误！

(3) A {% , mod} B

- 假若 A 和 B 为 null：回传 (Long)0
- 假若 A 或 B 为 BigDecimal、Float、Double 或包含 e / E 的字符串时，皆转为 Double，然后依照运算符执行运算
- 假若 A 或 B 为 BigInteger 时，将另一个转为 BigInteger，则回传 A.reminder(B)
- 否则，将 A 和 B 皆转为 Long，然后依照运算符执行运算
- 假若运算结果产生异常时，则错误！

(4) -A

- 假若 A 为 null：回传 (Long)0
- 假若 A 为 BigDecimal 或 BigInteger 时，回传 A.negate()
- 假若 A 为 String 时：
 - 假若 A 包含 e / E 时，将转为 Double，然后依照运算符执行运算
 - 否则，转为 Long，然后依照运算符执行运算
 - 假若运算结果产生异常时，则错误！
- 假若 A 为 Byte、Short、Integer、Long、Float 或 Double
 - 直接依原本类型执行运算
 - 假若运算结果产生异常时，则 错误！
- 否则，错误！

Tomcat 上的 jsp-examples 中，有一个 EL 算术运算符的范例 basic-arithmetic.jsp。它的程序很简单，所以不在这里多做说明，它的执行结果如图 6-4 所示。

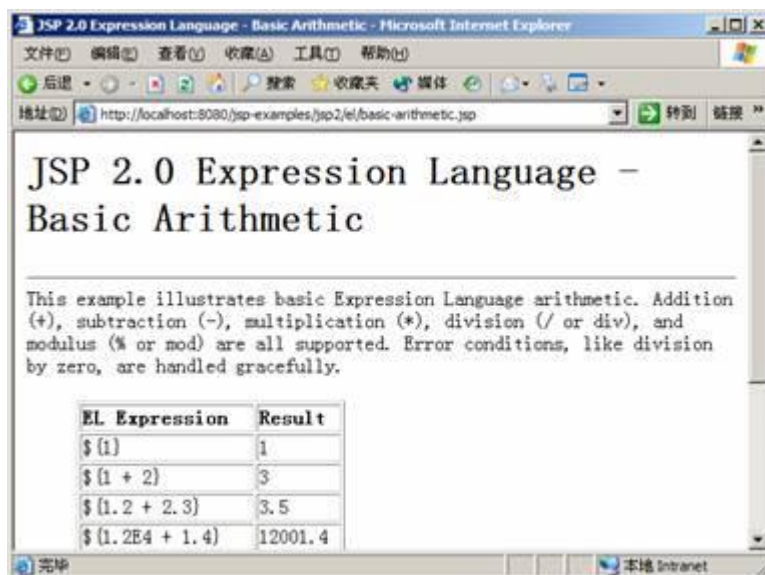


图 6-4 basic-arithmetic.jsp 的执行结果

6-5 EL 关系运算符

EL 关系运算符有以下六个运算符（见表 6-7）：

表 6-7

关系运算符	说 明	范 例	结 果
<code>=</code> 或 <code>eq</code>	等于	<code>\${ 5 == 5 }</code> 或 <code>\${ 5 eq 5 }</code>	true
<code>!=</code> 或 <code>ne</code>	不等于	<code>\${ 5 != 5 }</code> 或 <code>\${ 5 ne 5 }</code>	false
<code><</code> 或 <code>lt</code>	小于	<code>\${ 3 < 5 }</code> 或 <code>\${ 3 lt 5 }</code>	true
<code>></code> 或 <code>gt</code>	大于	<code>\${ 3 > 5 }</code> 或 <code>\${ 3 gt 5 }</code>	false
<code><=</code> 或 <code>le</code>	小于等于	<code>\${ 3 <= 5 }</code> 或 <code>\${ 3 le 5 }</code>	true
<code>>=</code> 或 <code>ge</code>	大于等于	<code>\${ 3 >= 5 }</code> 或 <code>\${ 3 ge 5 }</code>	false

注意

在使用 EL 关系运算符时，不能够写成：

```
${param.password1} == ${param.password2}
```

或者

```
${ ${param.password1} == ${ param.password2 } }
```

而应写成

```
${ param.password1 == param.password2 }
```

接下来，我们依照下列几种情况，详细说明 EL 关系运算符的规则：

(1) A {<, >, <=, >=, lt, gt, le, ge} B

- 假若 A = B，运算符为 <=, le, >=, ge 时，回传 true，否则回传 false
- 假若 A 为 null 或 B 为 null 时，回传 false
- 假若 A 或 B 为 BigDecimal 时，将另一个转为 BigDecimal，然后回传 A.compareTo(B) 的值
- 假若 A 或 B 为 Float、Double 时，皆转为 Double 类型，然后依其运算符运算
- 假若 A 或 B 为 BigInteger 时，将另一个转为 BigInteger，然后回传 A.compareTo(B) 的值
- 假若 A 或 B 为 Byte、Short、Character、Integer 或 Long 时，皆转为 Long 类型，然后依其运算符运算
- 假若 A 或 B 为 String 时，将另一个也转为 String，然后做词汇上的比较
- 假若 A 为 Comparable 时，则：
 - 假若 A.compareTo(B) 产生异常时，则错误！
- 否则，采用 A.compareTo(B) 的比较结果
- 假若 B 为 Comparable 时，则：
 - 假若 B.compareTo(A) 产生异常时，则错误！
- 否则，采用 A.compareTo(B) 的比较结果
- 否则，错误！

(2) A {=, !=, eq, ne} B

- 假若 A = B，依其运算符运算
- 假若 A 为 null 或 B 为 null 时：= / eq 则回传 false，!= / ne 则回传 true
- 假若 A 或 B 为 BigDecimal 时，将另一个转为 BigDecimal，则：
 - 假若运算符为 = / eq，则 回传 A.equals(B)
 - 假若运算符为 != / ne，则 回传 !A.equals(B)

- 假若 A 或 B 为 Float、Double 时，皆转为 Double 类型，然后依其运算符运算
- 假若 A 或 B 为 BigInteger 时，将另一个转为 BigInteger，则：
 - 假若运算符为 `=` / `eq`，则 回传 `A.equals(B)`
 - 假若运算符为 `!=` / `ne`，则 回传 `!A.equals(B)`
- 假若 A 或 B 为 Byte、Short、Character、Integer 或 Long 时，皆转为 Long 类型，然后依其运算符运算
- 假若 A 或 B 为 Boolean 时，将另一个也转为 Boolean，然后依其运算符运算
- 假若 A 或 B 为 String 时，将另一个也转为 String，然后做词汇上的比较
- 否则，假若 `A.equals(B)` 产生异常时，则 错误！
- 否则，然后依其运算符运算，回传 `A.equals(B)`

Tomcat 上的 `jsp-examples` 中，有一个 EL 关系运算符的范例 `basic-comparisons.jsp`。它的程序很简单，所以不在这里多做说明，大家直接看它的执行结果（如图 6-5 所示）：

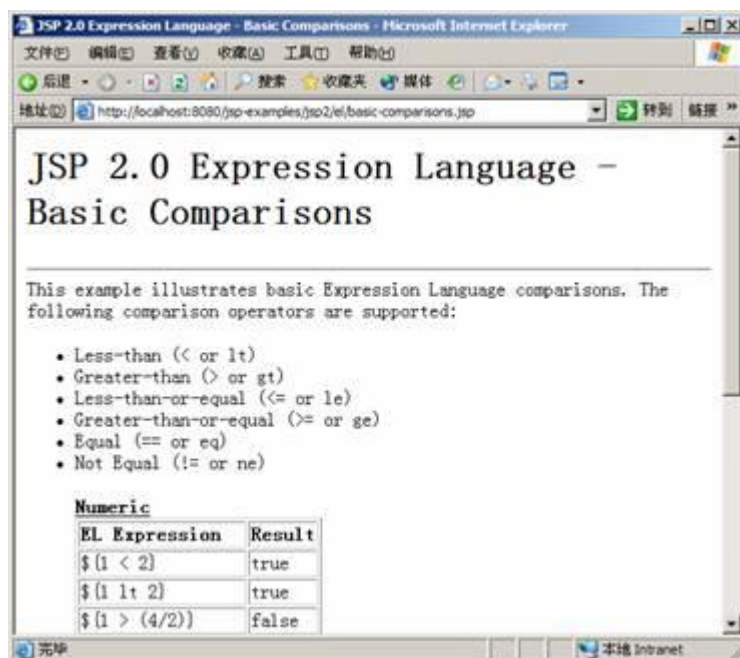


图 6-5 `basic-comparisons.jsp` 的执行结果

6-6 EL 逻辑运算符

EL 逻辑运算符只有三个（见表 6-8）：

表 6-8

逻辑运算符	说 明	范 例	结 果
&& 或 and	交集	<code>\${ A && B }</code> 或 <code>\${ A and B }</code>	true / false
或 or	并集	<code>\${ A B }</code> 或 <code>\${ A or B }</code>	true / false
! 或 not	非	<code>\${ !A }</code> 或 <code>\${ not A }</code>	true / false

下面举几个例子：

```
${ param.month == 7 and param.day == 14 }  
  
${ param.month == 7 || param.day == 14 }  
  
${ not param.choice }
```

EL 逻辑运算符的规则很简单：

(1) A {&&, and, || 或 or } B

- 将 A 和 B 转为 Boolean，然后依其运算符运算

(2) {!, not}A

- 将 A 转为 Boolean，然后依其运算符运算

6-7 EL 其他运算符

EL 除了上述三大类的运算符之外，还有下列几个重要的运算符：

(1) Empty 运算符

(2) 条件运算符

(3) () 括号运算符

6-7-1 Empty 运算符

Empty 运算符主要用来判断值是否为 null 或空的，例如：

```
${ empty param.name }
```

接下来说明 Empty 运算符的规则:

(1) {empty} A

- 假若 A 为 null 时, 回传 true
- 否则, 假若 A 为空 String 时, 回传 true
- 否则, 假若 A 为空 Array 时, 回传 true
- 否则, 假若 A 为空 Map 时, 回传 true
- 否则, 假若 A 为空 Collection 时, 回传 true
- 否则, 回传 false

6-7-2 条件运算符

所谓条件运算符如下:

```
${ A ? B : C }
```

意思是说, 当 A 为 true 时, 执行 B; 而 A 为 false 时, 则执行 C。

6-7-3 括号运算符

括号运算符主要用来改变执行优先权, 例如: `${ A * (B+C) }`

至于运算符的优先权, 如下所示(由高至低, 由左至右):

- [], .
- ()
- - (负)、not、!、empty
- *, /、div、%、mod
- +、- (减)
- <、>、<=、>=、lt、gt、le、ge
- ==、!=、eq、ne
- &&、and

- ||、or
- \${ A ? B : C }

最后笔者写一个 `ELOperator.jsp` 范例，将所有运算符实际操作一遍。

`ELOperator.jsp`

```
<%@ page contentType="text/html; charset=GB2312" %>

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

<html>

<head>

<title>CH6 - ELOperator.jsp</title>

</head>

<body>

<h2>EL 的运算符</h2>

<c:set value="mike" var="username" scope="request" />

<table border="1" width="50%" align="left">

<TR>

<TR>

<TH>运算式</TH>

<TH>结果</TH>

</TR>

<TR><TD>14 + 3</TD><TD>${ 14 + 3}</TD></TR>
```

```
<TR><TD>14 - 3</TD><TD>${14 - 3}</TD></TR>

<TR><TD>14 * 3</TD><TD>${14 * 3}</TD></TR>

<TR><TD>14 / 3</TD><TD>${14 / 3}</TD></TR>

<TR><TD>14 % 3</TD><TD>${14 % 3}</TD></TR>

<TR><TD>14 == 3</TD><TD>${14 == 3}</TD></TR>

<TR><TD>14 != 3</TD><TD>${14 != 3}</TD></TR>

<TR><TD>14 < 3</TD><TD>${14 < 3}</TD></TR>

<TR><TD>14 > 3</TD><TD>${14 > 3}</TD></TR>

<TR><TD>14 <= 3</TD><TD>${14 <= 3}</TD></TR>

<TR><TD>14 >= 3</TD><TD>${14 >= 3}</TD></TR>

<TR><TD>true && false</TD><TD>${true && false}</TD></TR>

<TR><TD>true || false</TD><TD>${true || false}</TD></TR>

<TR><TD>! false</TD><TD>${! false}</TD></TR>

<TR><TD>empty username</TD><TD>${empty username}</TD></TR>

<TR><TD>empty password</TD><TD>${empty password}</TD></TR>

</table>

</body>

</html>
```

EL 的数学运算符、相等运算符、关系运算符和逻辑运算符就跟其他程序语言一样，并没有特别的地方。但是它的 `empty` 运算符就比较特别，为了测试它，笔者写了这样一程序代码：

```
<c:set value="mike" var="username" scope="request" />
```

这样 Request 属性范围里就存在一个名称为 `username`、值为 `mike` 的属性。执行此程序时，读者将会发现 `${empty username}` 为 `false`； `${empty password}` 为 `true`，其代表的意义就是：它可以在四种属性范围中找到 `username` 这个属性，但是找不到 `password` 这个属性。ELOperator.jsp 的执行

结果如图 6-6:



运算式	结果
14 + 3	17
14 - 3	11
14 * 3	42
14 / 3	4.666666666666667
14 % 3	2
14 == 3	false
14 != 3	true
14 < 3	false
14 > 3	true
14 <= 3	false
14 >= 3	true
true && false	false
true false	true
! false	true
empty username	false
empty password	true

图 6-6 ELOperator.jsp 的执行结果

6-8 EL Functions

前面几节主要介绍 EL 语法的使用和规则, 本节笔者将介绍如何自定义 EL 的函数(functions)。

EL 函数的语法如下:

```
ns:function( arg1, arg2, arg3 ... argN)
```

其中 ns 为前置名称(prefix), 它必须和 taglib 指令的前置名称一样。如下范例:

```
<% @ taglib prefix="my"
    uri="http://jakarta.apache.org/tomcat/jsp2-example-taglib" %>
.....
${my:function(param.name)}
```

前置名称都为 my, 至于 function 为 EL 函数的名称, 而 arg1、arg2 等等, 都是 function 的传入值。在 Tomcat 5.0.16 中有一个简单的 EL 函数范例, 名称为 functions.jsp, 笔者接下来将依此范例来说明如何自定义 EL 函数。

6-8-1 Tomcat EL 函数范例

Tomcat 提供的 EL 函数范例中，自定义两个 EL 函数：reverse 和 countVowels，其中：

reverse 函数：将传入的字符串以反向顺序输出。

countVowels 函数：计算传入的字符串中，和 aeiouAEIOU 吻合的字符个数。

图 6-7 是 functions.jsp 程序的执行结果：

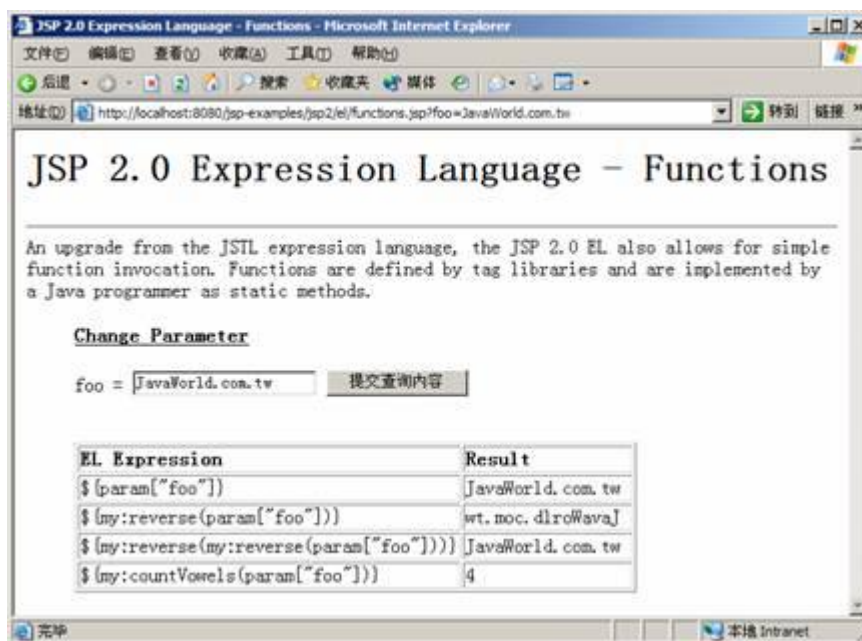


图 6-7 functions.jsp 的执行结果

输入 JavaWorld.com.tw 字符串至 reverse 函数后，回传 wt.moc.dIroWavaJ 的结果；若传入 countVowels 函数后，因为有两个 a 和 o，总共四个字符吻合，所以回传 4。

Tomcat 的 EL 函数范例，主要分为四个部分（见表 6-9）：

表 6-9

web.xml	设定 taglib 的 TLD 文件位置
functions.jsp	使用 EL 函数的范例程序
jsp2-example-taglib.tld	EL 函数、标签库的设定文件
jsp2.examples.el.Functions.java	EL 函数主要程序逻辑处理部分

这四个部分环环相扣，都互有关系，笔者依 functions.jsp 为中心，然后再慢慢说明其他部分。首先我们直接来看 functions.jsp 程序：

6-8-2 functions.jsp

functions.jsp

```
<%@ taglib prefix="my"
uri="http://jakarta.apache.org/tomcat/jsp2-example-taglib"%>

<html>

  <head>

    <title>JSP 2.0 Expression Language - Functions</title>

  </head>

  <body>

    <h1>JSP 2.0 Expression Language - Functions</h1>

    ... 略

    <blockquote>

      <u><b>Change Parameter</b></u>

      <form action="functions.jsp" method="GET">

        foo = <input type="text" name="foo" value="${param['foo']}">

          <input type="submit">

      </form>

      <br>

      <code>

        <table border="1">

          <thead>

            <td><b>EL Expression</b></td>

            <td><b>Result</b></td>
```

```
</thead>

<tr>

    <td>\${param["foo"]}</td>

    <td>${param["foo"]}&nbsp;</td>

</tr>

<tr>

    <td>\${my:reverse(param["foo"])}</td>

    <td>${my:reverse(param["foo"])}&nbsp;</td>

</tr>

<tr>

    <td>\${my:reverse(my:reverse(param["foo"]))}</td>

    <td>${my:reverse(my:reverse(param["foo"]))}&nbsp;</td>

</tr>

<tr>

    <td>\${my:countVowels(param["foo"])}</td>

    <td>${my:countVowels(param["foo"])}&nbsp;</td>

</tr>

</table>

</code>

</blockquote>

</body>

</html>
```

functions.jsp 程序中,一开始定义 taglib,它的前置名称为 my;uri 为 <http://jakarta.apache.org/>

tomcat/jsp2-example-taglib, 如下所示:

```
<%@ taglib prefix="my"
uri="http://jakarta.apache.org/tomcat/jsp2-example-taglib"%>
```

当 Container 执行这段程序时, 它会根据 uri 的值, 到 web.xml 中找相对应的 TLD (Tag Library Descriptor) 文件。至于 web.xml 如何设定两者之间的对应关系, 我们在 6-8-3 小节再说明。

functions.jsp 中包含一个窗体(form), 当用户在文本 [\[王玉 1 \]](#) 输入框(text input)中输入字符串, 按下按钮时, 底下会显示字符串经过 EL 函数处理后的结果。functions.jsp 程序最重要的部分是调用 EL 函数:

```
${my:reverse(param["foo"])} 
```

上述的意思是接收 foo 参数, 然后传入 reverse 函数。调用 EL 函数的方式很简单, 只要前置名称: 其中 EL 函数名称是被定义在 TLD 文件中, 这会在 6-8-4 小节详细说明。至于 reverse 函数的逻辑运算, 则是被定义在 jsp2.examples.el.Functions.java 程序中, 这部分会在 6-8-5 小节中说明。

注意

TLD 文件主要为标签的设定文件, 其中包含标签的名称、参数等等。在 JSP 2.0 之后, 相关 EL 函数的设定, 也可以在 TLD 文件中定义。

6-8-3 web.xml

web.xml 是每个 web 站台最主要的设定文件, 在这个设定文件中, 可以设定许多东西, 如: Servlet、Resource、Filter 等等。不过现在关心的是如何在 web.xml 中设定 taglib 的 uri 是对应到哪个 TLD 文件。笔者从范例的 web.xml 中节录出设定的片段程序如下:

web.xml

```
<jsp-config>

  <taglib>

    <taglib-uri>

      http://jakarta.apache.org/tomcat/jsp2-example-taglib
```

```
</taglib-uri>

<taglib-location>

    /WEB-INF/jsp2/jsp2-example-taglib.tld

</taglib-location>

</taglib>

</jsp-config>
```

在 web.xml 中，<taglib>用来设定标签的 TLD 文件位置。<taglib-uri>用来指定 taglib 的 uri 位置，用户可以自行给定一个 uri，例如：

```
<taglib-uri>http://www.javaworld.com.tw/jute</taglib-uri>

<taglib-uri>tw.com.javaworld</taglib-uri>
```

<taglib-location>用来指定 TLD 文件的位置。依照范例，它是指定在 WEB-INF/jsp2/目录下的 jsp2-example-taglib.tld。

因此，笔者所节录下来的 web.xml，它所代表的意思是：taglib 的 uri 为 <http://jakarta.apache.org/tomcat/jsp2-example-taglib>，它的 TLD 文件是在 WEB-INF/jsp2/目录下的 jsp2-example-taglib.tld。

6-8-4 jsp2-example-taglib.tld

在 jsp2-example-taglib.tld 中定义许多标签，其中笔者节录一段定义 EL 函数：

jsp2-example-taglib.tld

```
<function>

    <description>Reverses the characters in the given String</description>

    <name>reverse</name>

    <function-class>jsp2.examples.el.Functions</function-class>

    <function-signature>
```



```
        java.lang.String reverse( java.lang.String )

    </function-signature>

</function>

<function>

    <description>Counts the number of vowels (a,e,i,o,u) in the given
        String</description>

    <name>countVowels</name>

    <function-class>jsp2.examples.el.Functions</function-class>

    <function-signature>

        java.lang.String numVowels( java.lang.String )

    </function-signature>

</function>
```

上述定义两个 EL 函数，用<name>来设定 EL 函数名称，它们分别为 reverse 和 countVowels；用<function-class>设定 EL 函数的 Java 类，本范例的 EL 函数都是定义在 jsp2.examples.el.Functions；最后用<function-signature>来设定 EL 函数的传入值和回传值，例如：

```
<function-signature>java.lang.String
reverse( java.lang.String )</function-signature>
```

表示 reverse 函数有一 String 类型的传入值，然后回传 String 类型的值。最后我们再来看 reverse 和 countVowels 的程序。

6-8-5 Functions.java

Functions.java 主要定义三个**公开静态**的方法，分别为：reverse、numVowels 和 caps（见表 6-10）。下面是 Functions.java 完整的程序代码：

Functions.java

```
package jsp2.examples.el;

import java.util.*;

/**
 * Defines the functions for the jsp2 example tag library.
 * <p>Each function is defined as a static method.</p>
 */
public class Functions {

    public static String reverse( String text ) {

        return new StringBuffer( text ).reverse().toString();

    }

    public static int numVowels( String text ) {

        String vowels = "aeiouAEIOU";

        int result = 0;

        for( int i = 0; i < text.length(); i++ ) {

            if( vowels.indexOf( text.charAt( i ) ) != -1 ) {

                result++;

            }

        }

        return result;

    }

}
```

```
}

public static String caps( String text ) {

    return text.toUpperCase();

}

}
```

表 6-10

String reverse(String text)	将 text 字符串的顺序反向处理，然后回传反向后的字符串
int numVowels(String text)	将 text 字符串比对 aeiouAEIOU 等字符，然后回传比对中的次数
String caps(String text)	将 text 字符串都转为大写，然后回传此字符串

注意

在定义 EL 函数时，都必须为公开静态(public static)

第七章 JSTL 1.1

JSTL 全名为 **JavaServer Pages Standard Tag Library**，目前最新的版本为 1.1。JSTL 是由 JCP（Java Community Process）所指定的标准规格，它主要提供给 Java Web 开发人员一个标准通用的标签函数库。

Web 程序开发人员能够利用 JSTL 和 EL 来开发 Web 程序，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。

本章中，笔者将详细介绍如何使用 JSTL 中各种不同的标签，将依序介绍条件、循环、URL、U18N、XML、SQL 等标签的用法，让读者对 JSTL 有更深层的了解，并且能够学会如何使用 JSTL。

7-1 JSTL 1.1 简介

JavaServer Pages Standard Tag Library (1.1)，它的中文名称为 **JSP 标准标签函数库**。JSTL 是一个标准的已制定好的标签库，可以应用于各种领域，如：基本输入输出、流程控制、循环、XML 文件剖析、数据库查询及国际化和文字格式标准化的应用等。从表 7-1 可以知道，JSTL 所提供的标签函数库主要分为五大类：

- (1) 核心标签库 (Core tag library)
- (2) I18N 格式标签库 (I18N-capable formatting tag library)
- (3) SQL 标签库 (SQL tag library)
- (4) XML 标签库 (XML tag library)
- (5) 函数标签库 (Functions tag library)

表 7-1

JSTL	前置名称	URI	范例
核心标签库	c	http://java.sun.com/jsp/jstl/core	<c:out>
I18N 格式标签库	fmt	http://java.sun.com/jsp/jstl/xml	<fmt:formatDate>
SQL 标签库	sql	http://java.sun.com/jsp/jstl/sql	<sql:query>
XML 标签库	xml	http://java.sun.com/jsp/jstl/fmt	<x:forEach>
函数标签库	fn	http://java.sun.com/jsp/jstl/functions	<fn:split>

另外，JSTL 也支持 EL (Expression Language) 语法，例如：在一个标准的 JSP 页面中可能会使用到如下的写法：

```
<%= userList.getUser().getPhoneNumber() %>
```

使用 JSTL 搭配传统写法会变成这样：

```
<c_rt:out value="<%= userList.getUser( ).getPhoneNumber( ) %>" />
```

使用 JSTL 搭配 EL，则可以改写成如下的形式：

```
<c:out value="${userList.user.phoneNumber}" />
```

虽然对网页设计者来说，假如没有学过 Java Script 或者是第一次看到这种写法时，可能会搞不太懂，但是与 Java 语法相比，这应该更加容易学习。

7-1-1 安装使用 JSTL 1.1 JSTL

1.1 必须在支持 Servlet 2.4 且 JSP 2.0 以上版本的 Container 才可使用。JSTL 主要由 Apache 组织的 Jakarta Project 所实现，因此读者可以到 <http://jakarta.apache.org/builds/jakarta-taglibs/releases/standard/> 下载实现好的 JSTL 1.1，或者直接使用本书光盘中 JSTL 1.1，软件名称为：jakarta-taglibs-standard-current.zip。

下载完后解压缩，可以发现文件夹中所包含的内容如图 7-1 所示：

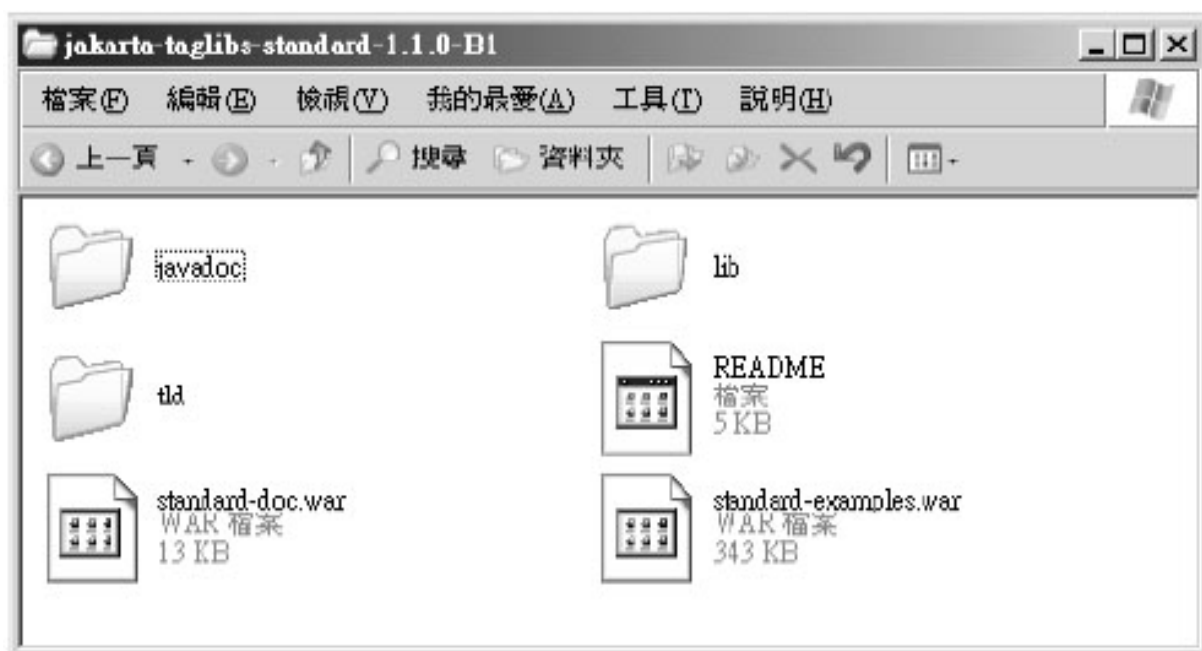


图 7-1 jakarta-taglibs-standard-1.1.0-B1 的目录结构

将 lib 中的 jstl.jar、standard.jar 复制到 Tomcat 的 WEB-INF\lib 中，然后就可以在 JSP 网页中使用 JSTL 了。除了复制 .jar 文件外，最好也把 tld 文件的目录也复制到 WEB-INF 中，以便日后使用。

注意

lib 目录下，除了 jstl.jar 和 standard.jar 之外，还有 old-dependencies 目录，这目录里面的东西是让之前 JSTL 1.0 的程序也能够在此 JSTL 1.1 环境下使用。tld 目录下有许多 TLD 文件，其中大部分都是 JSTL 1.0 的 TLD 文件，例如：c-1_0.tld 和 c-1_0-rt.tld。

下面写一个测试用的范例程序 HelloJSTL.jsp，程序主要是显示浏览器的版本和欢迎的字符串。

■ HelloJSTL.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
```

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

<html>
<head>
<title>测试你的第一个使用到 JSTL 的网页</title>
</head>
<body>
<c:out value="欢迎测试你的第一个使用到 JSTL 的网页" />
<br>你使用的浏览器是: </br>
<c:out value="${header['User-Agent']}" />
<c:set var="a" value="David O'Davies" />
<c:out value="David O'Davies" escapeXml="true" />
</body>
</html>
```

在 HelloJSTL.jsp 的范例里, 笔者用到核心标签库(Core)中的标准输出功能和 EL 的 header 隐含对象。若要在 JSP 网页中使用 JSTL 时, 一定要先做下面这行声明:

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

这段声明表示我将使用 JSTL 的核心标签库。一般而言, 核心标签库的前置名称(prefix)都为 c, 当然你也可以自行设定。不过 uri 此时就必须为 <http://java.sun.com/jsp/jstl/core>。

注意

JSTL 1.0 中, 核心标签库的 uri 默认为 <http://java.sun.com/jstl/core>, 比 JSTL 1.1 少一个 jsp/ 的路径。因为 JSTL 1.1 同时支持 JSTL 1.0 和 1.1, 所以假若核心标签库的 uri 为 <http://java.sun.com/jstl/core>, 则将会使用到 JSTL 1.0 的核心标签库。

接下来使用核心标签库中的 out 标签, 显示 value 的值。\${header['User-Agent']} 表示取得表头里的 User-Agent 的值, 即有关用户浏览器的种类。

```
<c:out value="欢迎测试你的第一个使用到 JSTL 的网页" />
<c:out value="${header['User-Agent']}" />
```

HelloJSTL.jsp 的执行结果如图 7-2 所示。

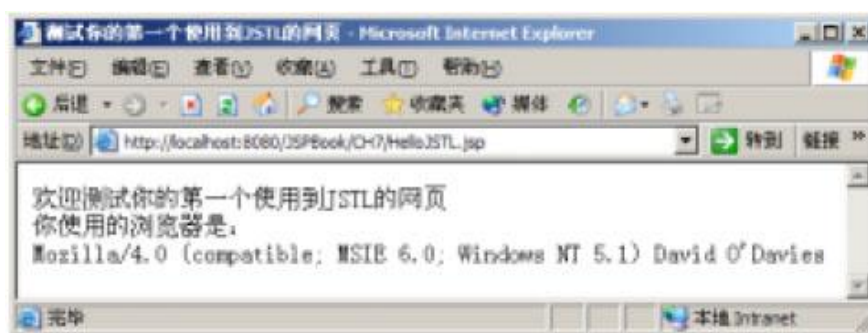


图 7-2 HelloJSTL.jsp 的执行结果

假若读者想要自定义 `taglib` 的 `uri` 时,那就必须在 `web.xml` 中加入设定值。例如:假若 `uri` 想要改为 `http://www.javaworld.com.tw/jstl/core` 时, `web.xml` 就必须加入如下设定:

```
<web-app>
:
  <jsp-config>
    <taglib>
      <taglib-uri>http://www.javaworld.com.tw/jstl/core</taglib-uri>
      <taglib-location>/WEB-INF/tld/c.tld</taglib-location>
    </taglib>
  </jsp-config>
:
</web-app>
```

在上面的设定中, `<taglib-uri>` 主要是设定标签库的 URL; 而 `<taglib-location>` 则是用来设定标签对应的 TLD 文件。因此, 使用 `<%@ taglib %>` 指令时, 可以直接写成如下语句:

```
<%@ taglib prefix="c" uri="http://www.javaworld.com.tw/jsp/jstl/core" %>
```

7-1-2 JSTL 1.1 VS. JSTL 1.0

JSTL 1.0 更新至 JSTL 1.1 时, 有以下几点不同:

(1) EL 原本是定义在 JSTL 1.0 的, 现在 EL 已经正式纳入 JSP 2.0 标准规范中, 所以在 JSTL 1.1 规范中, 已经没有 EL 的部分, 但是 JSTL 依旧能使用 EL。

(2) JSTL 1.0 中, 又分 EL 和 RT 两种函数库, 到了 JSTL 1.1 之后, 已经不再分这两种了。以下说明 EL 和 RT 的差别:

EL

- 完全使用 Expression Language
- 简单
- 建议使用

RT

- 使用 Scriptlet
- Java 语法
- 供不想转换且习惯旧表示法的开发者使用

笔者在此强烈建议大家使用 EL 来做, 简单又方便。

(3) JSTL 1.1 新增函数(functions)标签库, 主要提供一些好用的字符串处理函数, 例如: `fn:contains`、`fn:containsIgnoreCase`、`fn:endsWith`、`fn:indexOf`、`fn:join`、`fn:length`、`fn:replace`、`fn:split`、`fn:startsWith` 和 `fn:substring` 等等。

除了上述三项比较大的改变之外, 还包括许多小改变, 在此不多加说明, 有兴趣的读者可以去

看 JSTL 1.1 附录 B “Changes” 部分，那里有更详尽的说明。

7-1-3 安装 standard-examples

当解压缩 jakarta-taglibs-standard-current.zip 后，文件夹内 (见图 7-1) 有一个 standard-examples.war 的文件，将它移至 Tomcat 的 webapps 后，重新启动 Tomcat 会发现，在 webapps 目录下多了一个 standard-examples 的目录。接下来我们打开 IE，在 URL 位置上输入 <http://localhost:8080/standard-examples/>，你将会看到图 7-3 所示的画面。

这个站台有很多 JSTL 的范例，它包括以下几部分：

- General Purpose Tags
- Conditional Tags
- Iterator Tags
- Import Tags
- I18N & Formatting Tags
- XML Tags
- SQL Tags
- Functions
- Tag Library Validators
- Miscellaneous



图 7-3 standard-examples 站台

这些范例程序几乎涵盖了所有的 JSTL 标签函数库，假若读者对哪一个标签的使用有问题，可以先来找一找这里的范例程序，应该或多或少会有所帮助。

7-2 核心标签库 (Core tag library)

首先介绍的核心标签库(Core)主要有：基本输入输出、流程控制、迭代操作和 URL 操作。详细的分类如表 7-2 所示，接下来笔者将为读者一一介绍每个标签的功能。

表 7-2

分类	功能分类	标签名称
Core	表达式操作	out set remove catch
	流程控制	if choose when otherwise
	迭代操作	forEach forEachTokens
	URL 操作	import param url param redirect param

在 JSP 中要使用 JSTL 中的核心标签库时，必须使用 `<%@ taglib %>` 指令，并且设定 `prefix` 和 `uri` 的值，通常设定如下：

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

上述的功用在于声明将使用 JSTL 的核心标签库。

注意

假若没有上述声明指令，将无法使用 JSTL 的核心功能，这是读者在使用 JSTL 时必须要注意的地方。

7-2-1 表达式操作

表达式操作分类中包含四个标签：`<c:out>`、`<c:set>`、`<c:remove>`和`<c:catch>`。接下来将依序介绍这四个标签的用法。

<c:out>

<c:out>主要用来显示数据的内容，就像是 <%= scripting-language %> 一样，例如：

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

语法

语法 1：没有本体(body)内容

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

语法 2：有本体内容

```
<c:out value="value" [escapeXml="{true|false}"]>
```

default value

```
</c:out>
```

属性

名称	说明	EL	类型	必须	默认值
value	需要显示出来的值	YObject	Object	是	无
default	如果 value 的值为 null，则显示 default 的值	YObject	Object	否	无
escapeXml	是否转换特殊字符，如：<转换成<	Yboolean	boolean	否	true

注意

表格中的 EL 字段，表示此属性的值是否可以为 EL 表达式，例如：Y 表示 attribute = "\${表达式}" 为符合语法的，N 则反之。

Null 和 错误处理

- 假若 value 为 null，会显示 default 的值；假若没有设定 default 的值，则会显示一个空的字符串。

说明

一般来说，<c:out>默认会将 <、>、'、" 和 & 转换为 <、>、'、" 和 &。假若不想转换时，只需要设定<c:out>的 escapeXml 属性为 false 就可以了（见表 7-3）

表 7-3

字符	Entity
<	<
>	>
'	'
”	"
&	&

范例

```
<c:out value="Hello JSP 2.0 !! " />

<c:out value="${ 3 + 5 }" />

<c:out value="${ param.data }" default="No Data" />

<c:out value="<p>有特殊字符</p>" />

<c:out value="<p>有特殊字符</p>" escapeXml="false" />
```

1. 在网页上显示 Hello JSP 2.0 !! ；
2. 在网页上显示 8；
3. 在网页上显示由窗体传送过来的 data 参数之值，假若没有 data 参数，或 data 参数的值为 null 时，则网页上会显示 No Data；
4. 在网页上显示 “<p>有特殊字符</p>”；
5. 在网页上显示 “有特殊字符”。

<c:set>

<c:set>主要用来将变量储存至 JSP 范围中或是 JavaBean 的属性中。

语法

语法 1：将 value 的值储存至范围为 scope 的 varName 变量之中

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

语法 2：将本体内容的数据储存至范围为 scope 的 varName 变量之中

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

... 本体内容

</c:set>
```

语法 3：

将 `value` 的值储存至 `target` 对象的属性中

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

语法 4:

将本体内容的数据储存至 `target` 对象的属性中

```
<c:set target="target" property="propertyName">
```

... 本体内容

```
</c:set>
```

属性

名称	说明	EL	类型	必须	默认值
<code>value</code>	要被储存的值	Y	Object	否	无
<code>var</code>	欲存入的变量名称	N	String	否	无
<code>scope</code>	<code>var</code> 变量的 JSP 范围	N	String	否	page
<code>target</code>	为一 <code>JavaBean</code> 或 <code>java.util.Map</code> 对象	Y	Object	否	无
<code>property</code>	指定 <code>target</code> 对象的属性	Y	String	否	无

Null 和 错误处理

语法 3 和语法 4 会产生异常错误，有以下两种情况：

☆ `target` 为 null

☆ `target` 不是 `java.util.Map` 或 `JavaBean` 对象

假若 `value` 为 null 时：将由储存变量改为移除变量

☆ 语法 1：由 `var` 和 `scope` 所定义的变量，将被移除

□ 若 `scope` 已指定时，则 `PageContext.removeAttribute(varName, scope)`

□ 若 `scope` 未指定时，则 `PageContext.removeAttribute(varName)`

☆ 语法 3:

□ 假若 `target` 为 `Map` 时，则 `Map.remove(property)`

□ 假若 `target` 为 `JavaBean` 时，`property` 指定的属性为 null

说明

使用`<c:set>`时, `var` 主要用来存放表达式的结果; `scope` 则是用来设定储存的范围, 例如: 假若 `scope="session"`, 则将会把数据储存在 `session` 中。如果`<c:set>`中没有指定 `scope` 时, 则它会默认存在 `Page` 范围里。

注意

`var` 和 `scope` 这两个属性不能使用表达式来表示, 例如: 我们不能写成 `scope="${ourScope}"` 或者是 `var="${username}"`。

我们考虑下列的写法:

```
<c:set var="number" scope="session" value="${1 + 1}" />
```

把 `1+1` 的结果 `2` 储存到 `number` 变量中。如果`<c:set>`没有 `value` 属性, 此时 `value` 之值在`<c:set>`和`</c:set>`之间, 本体内内容看下面的范例:

```
<c:set var="number" scope="session">
<c:out value="${1+1}" />
</c:set>
```

上面的 `<c:out value="${1+1}" />` 部分可以改写成 `2` 或是 `<%=1+1%>`, 结果都会一样, 也就是说, `<c:set>`是把本体(body)运算后的结果来当做 `value` 的值。另外, `<c:set>`会把 `body` 中最开头和结尾的空白部分去掉。如:

```
<c:set var="number" scope="session">
    1 + 1
</c:set>
```

则 `number` 中储存的值为 `1 + 1` 而不是 `1 + 1。`

范例

```
<c:set var="number" scope="request" value="${1 + 1}" />

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

${3 + 5}

</c:set>

<c:set var="number" scope="request" value="${ param.number }" />

<c:set target="User" property="name" value="${ param.Username}" />
```

1. 将 `2` 存入 `Request` 范围的 `number` 变量中;

2. 将 8 存入 Session 范围的 number 变量中;
3. 假若 `${param.number}` 为 null 时, 则移除 Request 范围的 number 变量; 若 `${param.number}` 不为 null 时, 则将 `${param.number}` 的值存入 Request 范围的 number 变量中;
4. 假若 `${param.Username}` 为 null 时, 则设定 User(JavaBean) 的 name 属性为 null; 若不为 null 时, 则将 `${param.Username}` 的值存入 User(JavaBean) 的 name 属性(setter 机制)。

注意

上述范例的 3. 中, 假若 `${param.number}` 为 null 时, 则表示移除 Request 范围的 number 变量。

● `<c:remove>`

`<c:remove>` 主要用来移除变量。

语法

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

属性

名称	说明	EL	类型	必须	默认值
var	欲移除的变量名称	N	String	是	无
scope	var 变量的 JSP 范围	N	String	否	page

说明

`<c:remove>` 必须要有 var 属性, 即要被移除的属性名称, scope 则可有可无, 例如:

```
<c:remove var="number" scope="session" />
```

将 number 变量从 Session 范围中移除。若我们不设定 scope, 则 `<c:remove>` 将会从 Page、Request、Session 及 Application 中顺序寻找是否存在名称为 number 的数据, 若能找到时, 则将它移除掉, 反之则不会做任何的事情。

范例

笔者在这里写一个使用到 `<c:set>` 和 `<c:remove>` 的范例, 能让读者可以更快地了解如何使用它们, 此范例的名称为 Core_set_remove.jsp。

■ Core_set_remove.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
```

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

<html>

<head>

  <title>CH7 - Core_set_remove.jsp</title>

</head>

<body>

<h2><c:out value="<c:set>和<c:remove> 的用法" /></h2>

<c:set scope="page" var="number">

<c:out value="\${1+1}"/>

</c:set>

<c:set scope="request" var="number">

<%= 3 %>

</c:set>

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

4

</c:set>

初始设置

<table border="1" width="30%">
```

```
<tr>

    <th>pageScope.number</th>

    <td><c:out value="\${pageScope.number}" default="No Data" /></td>

</tr>

<tr>

    <th>requestScope.number</th>

    <td><c:out value="\${requestScope.number}" default="No Data" /></td>

</tr>

<tr>

    <th>sessionScope.number</th>

    <td><c:out value="\${sessionScope.number}" default="No Data" /></td>

</tr>

</table></br>

<c:out value='<c:remove var="number" scope="page" />之后' />

<c:remove var="number" scope="page" />

<table border="1" width="30%">

<tr>

    <th>pageScope.number</th>

    <td><c:out value="\${pageScope.number}" default="No Data" /></td>

</tr>

<tr>

    <th>requestScope.number</th>

    <td><c:out value="\${requestScope.number}" default="No Data" /></td>
```



```
</tr>

<tr>

    <th>sessionScope.number</th>

    <td><c:out value="${sessionScope.number}" default="No Data" /></td>

</tr>

</table></br>

<c:out value='<c:remove var="number" />之后' />

<c:remove var="number" />

<table border="1" width="30%">

<tr>

    <th>pageScope.number</th>

    <td><c:out value="${pageScope.number}" default="No Data" /></td>

</tr>

<tr>

    <th>requestScope.number</th>

    <td><c:out value="${requestScope.number}" default="No Data" /></td>

</tr>

<tr>

    <th>sessionScope.number</th>

    <td><c:out value="${sessionScope.number}" default="No Data" /></td>

</tr>

</table>

</body>
```

```
</html>
```

笔者一开始各在 Page、Request 和 Session 三个属性范围中储存名称为 number 的变量。然后先使用 `<c:remove var="number" scope="page" />` 把 Page 中的 number 变量移除，最后再使用 `<c:remove var="number" />` 把所有属性范围中 number 的变量移除。Core_set_remove.jsp 的执行结果如图 7-4 所示：



图 7-4 Core_set_remove.jsp 的执行结果

● <c:catch>

<c:catch>主要用来处理产生错误的异常状况，并且将错误信息储存起来。

语法

```
<c:catch [var="varName"] >
```

... 欲抓取错误的部分

```
</c:catch>
```

属性

名称	说明	EL	类型	必须	默认值
var	用来储存错误信息的变量	N	String	否	无

说明

`<c:catch>`主要将可能发生错误的部分放在`<c:catch>`和`</c:catch>`之间。如果真的发生错误，可以将错误信息储存至 `varName` 变量中，例如：

```
<c:catch var="message">

: //可能发生错误的部分

</c:catch>
```

另外，当错误发生在`<c:catch>`和`</c:catch>`之间时，则只有`<c:catch>`和`</c:catch>`之间的程序会被中止忽略，但整个网页不会被中止。

范例

笔者写一个简单的范例，文件名为 `Core_catch.jsp`，来让大家看一下`<c:catch>`的使用方式。

■ Core_catch.jsp

```
<%@ page contentType="text/html; charset=GB2312 " %>

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

<html>

<head>

    <title>CH7 - Core_catch.jsp</title>

</head>

<body>

<h2><c:out value="<c:catch> 的用法" /></h2>

<c:catch var="error_Message">

<%

    String eFormat = "not number";

    int i = Integer.parseInt(eFormat);

%>

</c:catch>

${error_Message}
```

```
</body>
```

```
</html>
```

笔者将一个字符串转成数字，如果字符串可以转为整数，则不会发生错误。但是这里笔者故意传入一个不能转成数字的字符串，让<c:catch>之间产生错误。当错误发生时，它会自动将错误存到error_Message 变量之中，最后再用<c:out>把错误信息显示出来，执行结果如图 7-5 所示。



图 7-5 Core_catch.jsp 的执行结果

可以发现到网页确实显示格式错误的信息。如果我们不使用<c:catch>，而把范例中的<c:catch>和</c:catch>拿掉，结果如图 7-6 所示。



图 7-6 Core_catch.jsp 没有<c:catch>和</c:catch>的执行结果

7-2-2 流程控制

流程控制分类中包含四个标签：`<c:if>`、`<c:choose>`、`<c:when>`和`<c:otherwise>`，笔者依此顺序依次说明这四个标签的使用。

● `<c:if>`

`<c:if>`的用途就和我们一般在程序中用的 `if` 一样。

语法

语法 1：没有本体内容(body)

```
<c:if test="testCondition" var="varName"
      [scope="{page|request|session|application}"]/>
```

语法 2：有本体内容

```
<c:if test="testCondition" [var="varName"]
      [scope="{page|request|session|application}"]>
    具体内容
</c:if>
```

属性

名称	说明	EL	类型	必须	默认值
<code>test</code>	如果表达式的结果为 <code>true</code> ，则执行本体内容， <code>false</code> 则相反	Y	boolean	是	无
<code>var</code>	用来储存 <code>test</code> 运算后的结果，即 <code>true</code> 或 <code>false</code>	N	String	否	无
<code>scope</code>	<code>var</code> 变量的 JSP 范围	N	String	否	page

说明

`<c:if>` 标签必须要有 `test` 属性，当 `test` 中的表达式结果为 `true` 时，则会执行本体内容；如果为 `false`，则不会执行。例如：`${param.username == 'admin'}`，如果 `param.username` 等于 `admin` 时，结果为 `true`；若它的内容不等于 `admin` 时，则为 `false`。

接下来看下列的范例：

```
<c:if test="${param.username == 'admin' }">
ADMIN 您好!! //body 部分
</c:if>
```

如果名称等于 `admin`，则会显示"ADMIN 您好!! "的动作，如果相反，则不会执行`<c:if>`的 `body`

部分，所以不会显示"ADMIN 您好!! //body 部分"。另外<c:if>的本体内容除了能放纯文字，还可以放任何 JSP 程序代码(Scriptlet)、JSP 标签或者 HTML 码。

除了 test 属性之外，<c:if>还有另外两个属性 var 和 scope。当我们执行<c:if>的时候，可以将这次判断后的结果存放到属性 var 里；scope 则是设定 var 的属性范围。哪些情况才会用到 var 和 scope 这两个属性呢？例如：当表达式过长时，我们会希望拆开处理，或是之后还须使用此结果时，也可以用它先将结果暂时保留，以便日后使用。

范例

笔者写了一个简单的范例，名称为 Core_if.jsp。

■ Core_if.jsp

```
<%@ page contentType="text/html; charset=GB2312 " %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
<head>
  <title>CH7 - Core_if.jsp</title>
</head>
<body>

<h2><c:out value="<c:if> 的用法" /></h2>

<c:if test="${param.username == 'Admin'}" var="condition" scope="page">
您好 Admin 先生
</c:if></br>

执行结果为: ${condition}

</body>
</html>
```

笔者在判断用户送来的参数时，如果 username 的值等于 Admin 时，则会将 condition 设为 true 并存放于 pageScope 中，否则存放于 condition 中，最后再显示结果。因为 JSTL 会自动找寻 condition 所存在的属性范围，因此只须使用 \${condition}，而不用 \${pageScope.condition}。Core_if.jsp

的执行结果如图 7-7。

注意

执行本范例时，请在 Core_if.jsp 后加上?username=Admin。

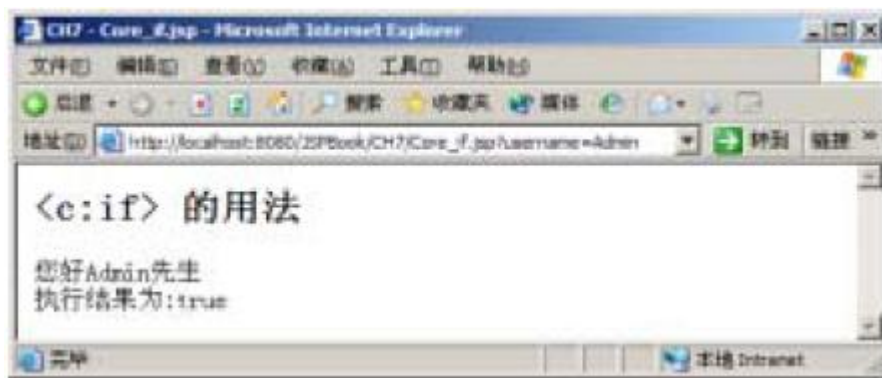


图 7-7 Core_if.jsp 的执行结果

● <c:choose>

<c:choose>本身只当做 <c:when> 和 <c:otherwise> 的父标签。

语法

```
<c:choose>
```

 本体内内容(<when> 和 <otherwise>)

```
</c:choose>
```

属性

无

限制

<c:choose>的本体内内容只能有：

- 空白
- 1 或多个 <c:when>
- 0 或多个 <c:otherwise>

说明

若使用<c:when>和<c:otherwise>来做流程控制时，两者都必须为<c:choose>的子标签，即：

```
<c:choose>
```

```
  :
```

```
  <c:when>
```

```
 </c:when>
```

```
:  
<c:otherwise>  
</c:otherwise>  
:  
</c:choose>
```

● <c:when>

<c:when> 的用途就和我们一般在程序中用的 when 一样。

语法

```
<c:when test="testCondition" >  
  本体内容  
</c:when>
```

属性

名称	说明	EL	类型	必须	默认值
test	如果表达式的结果为 true, 则执行本体内容, false 则相反	Y	boolean	是	无

限制

☆ <c:when> 必须在 <c:choose> 和 </c:choose> 之间

☆ 在同一个 <c:choose> 中时, <c:when> 必须在 <c:otherwise> 之前

说明

<c:when> 必须有 test 属性, 当 test 中的表达式结果为 true 时, 则会执行本体内容; 如果为 false 时, 则不会执行。

● <c:otherwise>

在同一个 <c:choose> 中, 当所有 <c:when> 的条件都没有成立时, 则执行 <c:otherwise> 的本体内容。

语法

```
<c:otherwise>  
  本体内容  
</c:otherwise>
```

属性

无

限制

- `<c:otherwise>` 必须在 `<c:choose>` 和 `</c:choose>` 之间
- 在同一个 `<c:choose>` 中时, `<c:otherwise>` 必须为最后一个标签

说明

在同一个 `<c:choose>` 中, 假若所有 `<c:when>` 的 `test` 属性都不为 `true` 时, 则执行 `<c:otherwise>` 的本体内容。

范例

笔者举一个典型的 `<c:choose>`、`<c:when>`和`<c:otherwise>`范例:

```
<c:choose>

  <c:when test="${condition1}">

    condition1 为 true

  </c:when>

  <c:when test="${ condition2}">

    condition2 为 true

  </c:when>

  <c:otherwise>

    condition1 和 condition2 都为 false

  </c:otherwise>

</c:choose>
```

范例说明: 当 `condition1` 为 `true` 时, 会显示 “`condition1 为 true`”; 当 `condition1` 为 `false` 且 `condition2` 为 `true` 时, 会显示 “`condition2 为 true`”, 如果两者都为 `false`, 则会显示 “`condition1 和 condition2 都为 false`”。

注意

假若 `condition1` 和 `condition2` 两者都为 `true` 时, 此时只会显示 “`condition1 为 true`”, 这是因为在同一个 `<c:choose>` 下, 当有好几个 `<c:when>` 都符合条件时, 只能有一个 `<c:when>` 成立。

7-2-3 迭代操作

迭代(Iterate)操作主要包含两个标签: <c:forEach>和<c:forEachTokens>, 笔者依此顺序依次说明这两个标签的使用。

● <c:forEach>

<c:forEach> 为循环控制, 它可以将集合(Collection)中的成员循序浏览一遍。运作方式为当条件符合时, 就会持续重复执行<c:forEach>的本体内容。

语法

语法 1: 迭代一集合对象之所有成员

```
<c:forEach [var="varName"] items="collection" [varStatus="varStatusName"] [begin="begin"]
[end="end"] [step="step"]>
  本体内容
</c:forEach>
```

语法 2: 迭代指定的次数

```
<c:forEach [var="varName"] [varStatus="varStatusName"] begin="begin" end="end"
[step="step"]>
  本体内容
</c:forEach>
```

属性

名称	说明	EL	类型	必须	默认值
var	用来存放现在指到的成员	N	String	否	无
items	被迭代的集合对象	Y	Arrays Collection Iterator Enumeration Map String	否	无
varStatus	用来存放现在指到的相关成员信息	N	String	否	无
begin	开始的位置	Y	int	否	0
end	结束的位置	Y	int	否	最后一个成员
step	每次迭代的间隔数	Y	int	否	1

限制

- 假若有 `begin` 属性时, `begin` 必须大于等于 0
- 假若有 `end` 属性时, 必须大于 `begin`
- 假若有 `step` 属性时, `step` 必须大于等于 0

Null 和 错误处理

- 假若 `items` 为 `null` 时, 则表示为一空的集合对象
- 假若 `begin` 大于或等于 `items` 时, 则迭代不运算

说明

如果要循序浏览一个集合对象, 并将它的内容显示出来, 就必须有 `items` 属性。

范例

下面的范例 `Core_forEach.jsp` 是将数组中的成员一个个显示出来的:

■ `Core_forEach.jsp`

```
<%@ page contentType="text/html; charset=GB2312 " %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>CH7 - Core_forEach.jsp</title>
</head>
<body>
<h2><c:out value="<c:forEach> 的用法" /></h2>
<%
  String atts[] = new String [5];
  atts[0]="hello";
  atts[1]="this";
  atts[2]="is";
  atts[3]="a";
  atts[4]="pen";
  request.setAttribute("atts", atts);
%>
<c:forEach items="${atts}" var="item" >
```

```
${item}</br>
</c:forEach>
</body>
</html>
```

在上述范例中,笔者先产生一个字符串数组,然后将此数组 `atts` 储存至 `Request` 的属性范围中,再用 `<c:forEach>` 将它循序浏览一遍。这里 `items` 表示被浏览的集合对象, `var` 用来存放指定的集合对象中成员,最后使用 `<c:out>` 将 `item` 的内容显示出来,执行结果如图 7-8 所示。



图 7-8 Core_forEach.jsp 的执行结果

注意

`varName` 的范围只存在 `<c:forEach>` 的本体中,如果超出了本体,则不能再取得 `varName` 的值。上个例子中,若 `${item}` 是在 `</c:forEach>` 之后执行时,如:

```
<c:forEach items="${atts}" var="item" >
</c:forEach>
${item}</br>
```

`${item}` 则不会显示 `item` 的内容。

`<c:forEach>` 除了支持数组之外,还有标准 J2SE 的集合类型,例如: `ArrayList`、`List`、`LinkedList`、`Vector`、`Stack` 和 `Set` 等等;另外还包括 `java.util.Map` 类的对象,例如: `HashMap`、`Hashtable`、`Properties`、`Provider` 和 `Attributes`。

`<c:forEach>` 还有 `begin`、`end` 和 `step` 这三种属性: `begin` 主要用来设定在集合对象中开始的位置(注意:第一个位置为 0); `end` 用来设定结束的位置;而 `step` 则是用来设定现在指到的成员和下一个将被指到成员之间的间隔。我们将之前的范例改成如下:

■ Core_forEach1.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>CH7 - Core_forEach1.jsp</title>
</head>
<body>
<h2><c:out value="<c:forEach> begin、end 和 step 的用法" /></h2>
<%
    String atts[] = new String [5];
    atts[0]="hello";
    atts[1]="this";
    atts[2]="is";
    atts[3]="a";
    atts[4]="pen";
    request.setAttribute("atts", atts);
%>
<c:forEach items="${atts}" var="item" begin="1" end="4" step="2" >
  ${item}</br>
</c:forEach>
</body>
</html>
```

`<c:forEach>`中指定的集合对象 `atts` 将会从第 2 个成员开始到第 5 个成员，并且每执行一次循环都会间隔一个成员浏览。因此结果是只显示 `atts[1]`和 `atts[3]`的内容，如图 7-9 所示。

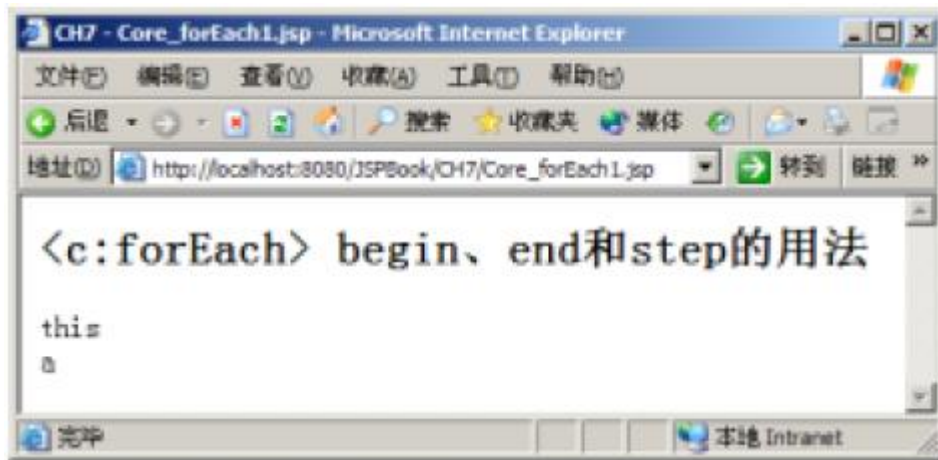


图 7-9 Core_forEach1.jsp 的执行结果

为了方便详细介绍 **begin**、**end** 和 **step** 的不同设定下所产生的结果，笔者将上面的范例改成如下：

```
<%  
    int atts[] = {1,2,3,4,5,6,7,8,9,10};  
    request.setAttribute("atts", atts);  
%>  
<c:forEach items="${atts}" var="item" begin="0" end="9" step="1" >  
    ${item}</br>  
</c:forEach>
```

这里笔者改变 **begin**、**end** 和 **step** 的值时，在网页上输出结果的变化如表 7-4。

表 7-4

begin	end	step	结果
-	-	-	1 2 3 4 5 6 7 8 9 10
5	-	-	6 7 8 9 10
-	5	-	1 2 3 4 5 6
-	-	5	1 6
5	5	-	6
5	5	5	6
0	8	2	1 3 5 7 9
0	8	3	1 4 7

0	8	4	1 5 9
15	20	-	无
20	8	-	空白结果
0	20	-	1 2 3 4 5 6 7 8 9 10

从表 7-4 中可以发现：

- (1) 当 `begin` 超过 `end` 时将会产生空的结果；
- (2) 当 `begin` 虽然小于 `end` 的值，但是当两者都大过容器的大小时，将不会输出任何东西；
- (3) 最后如果只有 `end` 的值超过集合对象的大小，则输出就和没有设定 `end` 的情况相同；
- (4) `<c:forEach>` 并不只是用来浏览集合对象而已，读者可以从表 7-4 中发现，`items` 并不是一定要有的属性，但是当没有使用 `items` 属性时，就一定要使用 `begin` 和 `end` 这两个属性。下面为一个简单的范例：

■ Core_forEach2.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>CH7 - Core_forEach2.jsp</title>
</head>
<body>
<h2><c:out value="<c:forEach> 循环" /></h2>
<c:forEach begin="1" end="10" var="item" >
${item}<br>
</c:forEach>
</body>
</html>
```

上述范例中，我们并没有执行浏览集合对象，只是设定 `begin` 和 `end` 属性的值，这样它就变成一个普通的循环。此范例是将循环设定为：从 1 开始跑到 10，总共会重复循环 10 次，并且将数字放到 `item` 的属性当中。`Core_forEach2.jsp` 的执行结果如图 7-10 所示。



图 7-10 Core_forEach2.jsp 的执行结果

当然它也可以搭配 `step` 使用，如果将 `step` 设定为 2，结果如图 7-11 所示。



图 7-11 当 `step` 设定为 2 时的结果

另外，`<c:forEach>` 还提供 `varStatus` 属性，主要用来存放现在指到之成员的相关信息。例如：我们写成 `varStatus="s"`，那么将会把信息存放在名称为 `s` 的属性当中。`varStatus` 属性还有另外四个属性：`index`、`count`、`first` 和 `last`，它们各自代表的意义如表 7-5：

表 7-5

属性	类型	意义
<code>index</code>	<code>number</code>	现在指到成员的索引
<code>count</code>	<code>number</code>	总共指到成员的总数
<code>first</code>	<code>boolean</code>	现在指到的成员是否为第一个成员
<code>last</code>	<code>boolean</code>	现在指到的成员是否为最后一个成员

我们可以使用 `varStatus` 属性取得循环正在浏览之成员的信息，下面为一个简单的范例：

■ Core_forEach3.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>CH7 - Core_forEach3.jsp</title>
</head>
<body>
<h2><c:out value="<c:forEach> varStatus 的四种属性" /></h2>
<%
    String atts[] = new String [5];
    atts[0]="hello";
    atts[1]="this";
    atts[2]="is";
    atts[3]="a";
    atts[4]="pen";
    request.setAttribute("atts", atts);
%>
<c:forEach items="${atts}" var="item"varStatus="s">
<h2><c:out value="${item}" />的四种属性: </h2>
index: ${s.index}</br>
count: ${s.count}</br>
first: ${s.first}</br>
last: ${s.last}</br>
</c:forEach>
</body>
</html>
```

执行结果如图 7-12 所示。

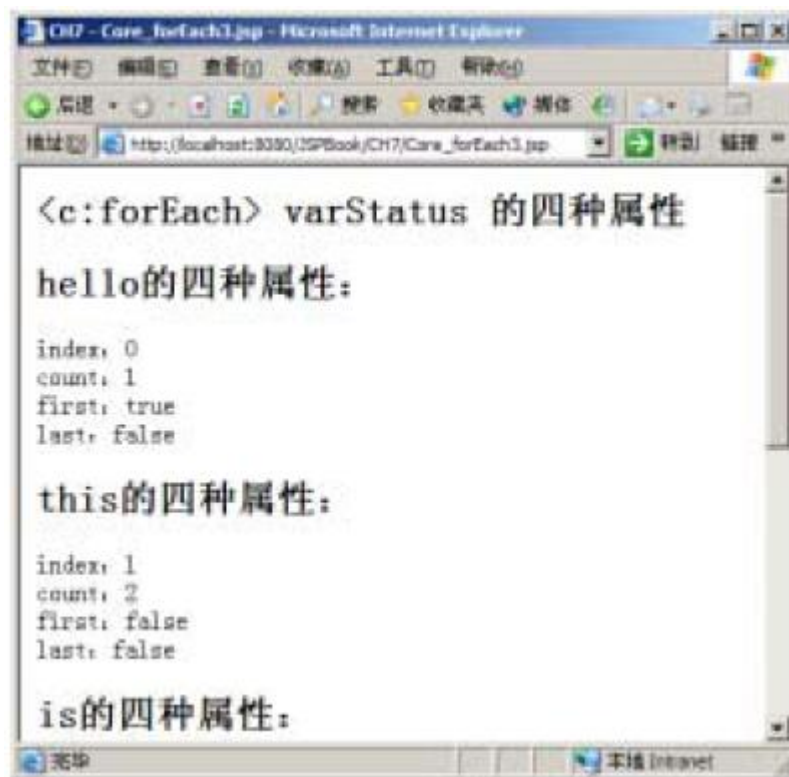


图 7-12 Core_forEach3.jsp 的执行结果

● <c:forTokens>

<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(delimiters)所分隔的。

语法

```
<c:forTokens items="stringOfTokens" delims="delimiters" [var="varName"]  
    [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>  
    本体内内容  
</c:forTokens>
```

属性

名称	说明	EL	类型	必须	默认值
var	用来存放现在指到的成员	N	String	否	无
items	被迭代的字符串	Y	String	是	无
delims	定义用来分割字符串的字符	N	String	是	无
varStatus	用来存放现在指到的相关成员信息	N	String	否	无
begin	开始的位置	Y	int	否	0
end	结束的位置	Y	int	否	最后一个成员

step	每次迭代的间隔数	Y	int	否	1
------	----------	---	-----	---	---

限制

- 假若有 begin 属性时，begin 必须大于等于 0
- 假若有 end 属性时，必须大于 begin
- 假若有 step 属性时，step 必须大于等于 0

Null 和 错误处理

- 假若 items 为 null 时，则表示为一空的集合对象
- 假若 begin 大于或等于 items 的大小时，则迭代不运算

说明

<c:forTokens>的 begin、end、step、var 和 varStatus 用法都和<c:forEach>一样，因此，笔者在这里就只介绍 items 和 delims 两个属性：items 的内容必须为字符串；而 delims 是用来分割 items 中定义的字符串之字符。

范例

下面为一个典型的<c:forTokens>的范例：

```
<c:forTokens items="A,B,C,D,E" delims="," var="item" >
  ${item}
</c:forTokens>
```

上面范例执行后，将会在网页中输出 ABCDE。它会把符号 “,” 当做分割的标记，拆成 5 个部分，也就是执行循环 5 次，但是并没有将 A,B,C,D,E 中的 “,” 显示出来。items 也可以放入 EL 的表达式，如下：

```
<%
    String phoneNumber = "123-456-7899";
    request.setAttribute("userPhone", phoneNumber);
%>
<c:forTokens items="${userPhone}" delims="-" var="item" >
  ${item}
</c:forTokens>
```

这个范例将会在网页上打印 1234567899，也就是把 123-456-7899 以 “-” 当做分割标记，将字符串拆为 3 份，每执行一次循环就将浏览的部分放到名称为 item 的属性当中。delims 不只指定一

种字符来分割字符串，它还可以一次设定多个分割字符串用的字符。如下面这个范例：

```
<c:forTokens items="A,B;C-D,E" delims=";- " var="item" >
  ${item}
</c:forTokens>
```

此范例会在网页输出 ABCDE，也就是说，`delims` 可以一次设定所有想当做分割字符串用的字符。其实用 `<c:forEach>` 也能做到分割字符串，写法如下：

```
<c:forEach items="A,B,C,D,E" var="item" >
  ${item}
</c:forEach>
```

上述范例同样也会在网页输出 ABCDE。`<c:forEach>` 并没有 `delims` 这个属性，因此 `<c:forEach>` 无法设定分割字符串用的字符，而 `<c:forEach>` 分割字符串用的字符只有 “，”，这和使用 `<c:forTokens>`，`delims` 属性设为 “，” 的结果相同。所以如果使用 `<c:forTokens>` 来分割字符串，功能和弹性上会比使用 `<c:forEach>` 来得较大。

7-2-4 URL 操作

JSTL 包含三个与 URL 操作有关的标签，它们分别为：`<c:import>`、`<c:redirect>` 和 `<c:url>`。它们主要的功能是：用来将其他文件的内容包含起来、网页的导向，还有 url 的产生。笔者将依序介绍这三个标签。

● `<c:import>`

`<c:import>` 可以把其他静态或动态文件包含至本身 JSP 网页。它和 JSP Action 的 `<jsp:include>` 最大的差别在于：`<jsp:include>` 只能包含和自己同一个 web application 下的文件；而 `<c:import>` 除了能包含和自己同一个 web application 的文件外，亦可以包含不同 web application 或者是其他网站的文件。

语法

语法 1：

```
<c:import url="url" [context="context"] [var="varName"]
          [scope="{page|request|session|application}"] [charEncoding="charEncoding"]>
  本体内容
</c:import>
```

语法 2:

```
<c:import url="url" [context="context"]  
        varReader="varReaderName" [charEncoding="charEncoding"]>
```

本体内容

```
</c:import>
```

属性

名称	说明	EL	类型	必须	默认值
url	一文件被包含的地址	Y	String	是	无
context	相同 Container 下, 其他 web 站台必须以 “/” 开头	Y	String	否	无
var	储存被包含的文件的内容(以 String 类型存入)	N	String	否	无
scope	var 变量的 JSP 范围	N	String	否	Page
charEncoding	被包含文件之内容的编码格式	Y	String	否	无
varReader	储存被包含的文件的内容(以 Reader 类型存入)	N	String	否	无

Null 和 错误处理

- 假若 url 为 null 或空时, 会产生 JspException

说明

首先<c:import>中必须要有 url 属性, 它是用来设定被包含网页的地址。它可以为绝对地址或是相对地址, 使用绝对地址的写法如下:

```
<c:import url="http://java.sun.com" />
```

<c:import>就会把 <http://java.sun.com> 的内容加到网页中。

另外<c:import>也支持 FTP 协议, 假设现在有一个 FTP 站台, 地址为 <ftp.javaworld.com.tw>, 它里面有一个文件 data.txt, 那么可以写成如下方式将其内容显示出来:

```
<c:import url="ftp://ftp.cse.yzu.edu.tw/data.txt" />
```

如果是使用相对地址, 假设存在一个文件名为 Hello.jsp, 它和使用<c:import>的网页存在于同一个 webapps 的文件夹时, <c:import>的写法如下:

```
<c:import url="Hello.jsp" />
```

如果以 “/” 开头, 那么就表示跳到 web 站台的根目录下, 以 Tomcat 为例, 即 webapps 目录。假设一个文件为 hello.txt, 存在于 webapps/examples/images 里, 而 context 为 examples, 可以

写成以下方式将 `hello.txt` 文件包含进我们的 JSP 页面之中：

```
<c:import url="images/hello.txt" />
```

接下来如果要包含在同一个服务器上，但并非同一个 web 站台的文件时，就必须加上 `context` 属性。假设此服务器上另外还有一个 web 站台，名为 `others`，`others` 站台底下有一个文件夹为 `jsp`，且里面有 `index.html` 这个文件，那么就可以写成如下方式将此文件包含进来：

```
<c:import url="/jsp/index.html" context="/others" />
```

注意

被包含文件的 web 站台必须在 `server.xml` 中定义过，且 `<Context>` 的 `crossContext` 属性值必须为 `true`，这样一来，`others` 目录下的文件才可以被其他 web 站台调用。

`server.xml` 的设定范例如下：

```
: <Context path="/others" docBase="others" debug="0"
                                reloadable="true" crossContext="true" />
:
```

除此之外，`<c:import>` 也提供 `var` 和 `scope` 属性。当 `var` 属性存在时，虽然同样会把其他文件的内容包含进来，但是它并不会输出至网页上，而是以 `String` 的类型储存至 `varName` 中。`scope` 则是设定 `varName` 的范围。储存之后的数据，我们在需要用时，可以将它取出来，代码如下：

```
<c:import url="/images/hello.txt" var="s" scope="session" />
```

我们可以把常重复使用的商标、欢迎语句或者是版权声明，用此方法储存起来，想输出在网页上时，再把它导入进来。假若想要改变文件内容时，可以只改变被包含的文件，不用修改其他网页。

另外，可以在 `<c:import>` 的本体内容中使用 `<c:param>`，它的功用主要是：可以将参数传递给被包含的文件，它有两个属性 `name` 和 `value`，如表 7-6 所示：

表 7-6

名称	说明	EL	类型	必须	默认值
<code>name</code>	参数名称	Y	<code>String</code>	是	无
<code>value</code>	参数的值	Y	<code>String</code>	否	本体内容

这两个属性都可以使用 EL，所以我们写成如下形式：

```
<c:import url="http://java.sun.com" >
<c:param name="test" value="1234" />
</c:import>
```

这样的做法等于是包含一个文件，并且所指定的网址会变成如下：

```
http://java.sun.com?test=1234
```

范例

下面为一用到 `<c:import>`、`<c:param>` 及属性范围的范例，`Core_import.jsp` 和 `Core_imported.jsp`：

■ Core_import.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>CH7 - Core_import.jsp</title>
</head>
<body>
<h2><c:out value="<c:import> 的用法" /></h2>
<c:set var="input1" value="使用属性范围传到 Core_imported.jsp 中" scope="request"/>包含
core_imported.jsp 中<hr/>
<c:import url="Core_imported.jsp" charEncoding="GB2312" >
<c:param name="input2" value="使用<c:param>传到 Core_imported.jsp 中"/>
</c:import><hr/>
${output1}
</body>
</html>
```

程序中，笔者分别使用 `<c:set>` 和 `<c:param>` 来传递参数。

■ Core_imported.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
<head>
```

```
<title>CH7 - Core_imported.jsp</title>
</head>
<body>
<fmt:requestEncoding value="GB2312" />
<c:set var="output1" value="使用属性范围传到 Core_import.jsp 中" scope="request"/>
${input1}</br>
<c:out value="${param.input2}" escapeXml="true" />
</body>
</html>
```

Core_imported.jsp 是被包含的文件，它会把从 Core_import.jsp 传来的参数分别输出到页面上，必须注意的是 input1 参数是使用属性范围来传递的，因此可以直接用`${input1}`来得到参数，而 input2 则必须使用`${param.input2}`来得到参数。

此外，笔者还使用`<c:set>`来传递值给 Core_import.jsp，这就是`<c:param>`无法做到的动作，`<c:param>`只能从包含端抛给被包含端，但是在属性范围中，可以让包含端也能得到被包含端传来的数据。Core_import.jsp 的执行结果如图 7-13 所示：



图 7-13 Core_import.jsp 的执行结果

● <c:url>

<c:url>主要用来产生一个 URL。

语法

语法 1: 没有本体内容

```
<c:url value="value" [context="context"] [var="varName"]  
      [scope="{page|request|session|application}"] />
```

语法 2: 本体内容代表查询字符串(Query String)参数

```
<c:url value="value" [context="context"] [var="varName"]  
      [scope="{page|request|session|application}"] >
```

<c:param> 标签

</c:url>

属性

名称	说明	EL	类型	必须	默认值
value	执行的 URL	Y	String	是	无
context	相同 Container 下, 其他 web 站台必须以 “/” 开头	Y	String	否	无
var	储存被包含文件的内容(以 String 类型存入)	N	String	否	无
scope	var 变量的 JSP 范围	N	String	否	Page

说明

在这里笔者直接使用例子来说明。

```
<c:url value="http://www.javafan.net " >  
<c:param name="param" value="value"/>  
</c:url>
```

读者可以发现<c:url>也可以搭配<c:param>使用, 上面执行结果将会产生一个网址为 <http://www.javafan.net?param=value>, 我们更可以搭配 HTML 的<a>使用, 如下:

```
<a href="  
  <c:url value="http://www.javafan.net " >  
  <c:param name="param" value="value"/>  
</c:url">>Java 爱好者</a>
```

另外<c:url>还有三个属性, 分别为 context、var 和 scope。context 属性和之前的<c:import>相同, 可以用来产生一个其他 web 站台的网址。如果<c:url>有 var 属性时, 则网址会被存到 varName 中, 而不会直接输出网址。

哪些状况下才会去使用<c:url>? 例如: 当我们须动态产生网址时, 有可能传递的参数不固定,

或者是需要一个网址能连至同服务器的其他 web 站台之文件，而且`<c:url>`更可以将产生的网址储存起来重复使用。另外，在以前我们必须使用相对地址或是绝对地址去取得需要的图文件或文件，现在我们可以直接利用`<c:url>`从 web 站台的角度来设定需要的图文件或文件的地址，如下：

```
" />
```

如此就会自动产生连到 image 文件夹里的 code.gif 的地址，不再须耗费精神计算相对地址，并且当网域名称改变时，也不用再改变绝对地址。

● `<c:redirect>`

`<c:redirect>`可以将客户端的请求从一个 JSP 网页导向到其他文件。

语法

语法 1：没有本体内容

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

语法 2：本体内容代表查询字符串(Query String)参数

```
<c:redirect url="url" [context="context"] > <c:param> </c:redirect >
```

属性

名称	说明	EL	类型	必须	默认值
url	导向的目标地址	Y	String	是	无
context	相同 Container 下，其他 web 站台必须以“/”开头	Y	String	否	无

说明

url 就是设定要被导向到的目标地址，它可以是相对或绝对地址。例如：我们写成如下：

```
<c:redirect url="http://www.javafan.net" />
```

那么网页将会自动导向到 <http://www.javafan.net>。另外，我们也可以加上 context 这个属性，用来导向至其他 web 站台上的文件，例如：导向到 /others 下的 /jsp/index.html 时，写法如下：

```
<c:redirect url="/jsp/index.html" context="/others" />
```

`<c:redirect>` 的功能不止可以导向网页，同样它还可以传递参数给目标文件。在这里我们同样使用`<c:param>`来设定参数名称和内容。

范例

■ Core_redirect.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
<html>

<head>

  <title>CH7 - Core_redirect.jsp</title>

</head>

<body>

<h2><c:out value="<c:redirect> 的用法" /></h2>

<c:redirect url="http://java.sun.com">

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

</c:redirect>

<c:out value="不会执行喔!!!" />

</body>

</html>
```

执行结果如图 7-14，可以发现网址部分为 <http://java.sun.com/?param=value>:



图 7-14 Core_redirect.jsp 的执行结果

注：暂时收录至此，待续部分请关注 <http://www.javafan.net>