[**JSTL-核心标签库**](http://www.cnblogs.com/Fskjb/archive/2009/07/05/1517164.html)

JSTL 全名为Java Server Pages Standard Tag Library（JSP Standard Tag Library）,它的中文名称为JSP 标准标签函数库。

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。

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。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)

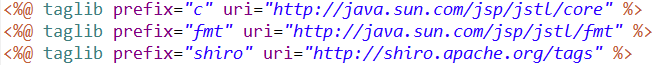
（3）SQL 标签库 (SQL tag library)

（4）XML 标签库 (XML tag library)

（5）函数标签库 (Functions tag library)

|  |  |  |
| --- | --- | --- |
| JSTL | 前缀 | URI |
| 核心标签库 | c | http://java.sun.com/jsp/jstl/core |
| I18N格式标签库 | fmt | http://java.sun.com/jsp/jstl/xml |
| SQL标签库 | sql | http://java.sun.com/jsp/jstl/sql |
| XML标签库 | xml | http://java.sun.com/jsp/jstl/fmt |
| 函数标签库 | fn | http://java.sun.com/jsp/jstl/functions |

若要在JSP 网页中使用JSTL ，一定要先做下面这行声明：





核心标签库 (Core tag library)

Core 标签库，又被称为核心标签库，该标签库的工作是对于 JSP 页面一般处理的封装。在该标签库中的标签一共有 14 个，被分为了四类，分别是：

         通用核心标签： <c:out> 、 <c:set> 、 <c:remove> 、 <c:catch> 。

         条件控制标签： <c:if> 、 <c:choose> 、 <c:when> 、 <c:otherwise> 。

         循环控制标签： <c:forEach> 、 <c:forTokens> 。

         URL 相关标签： <c:import> 、 <c:url> 、 <c:redirect> 、 <c:param> 。

在使用jstl的函数标签库之前需要在页面中引入：

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

<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 | Y | Object | 是 | 无 |
| default | 如果value的值为null则显示default的值 | Y | Object | 否 | 无 |
| escapeXml | 是否转换特殊字符，如：<转换成&lt; | Y | 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 &lt;、&gt;、&#039;、&#034; 和 &amp;。

假若不想转换时，只需要设定<c:out>的escapeXml 属性为fasle 就可以了。

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 |
| value | 要被储存的值 | Y | Object | 否 | 无 |
| var | 欲存入的变量名称 | N | String | 否 | 无 |
| scopevar | 变量的JSP范围 | N | String | 否 | page |
| target | 为一JavaBean或java.util.Map对象 | Y | Object | 否 | 无 |
| property | 指定target对象的属性 | Y | String | 否 | 无 |

**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；<c:set>是把本体(body)运算后的结果来当做value的值。

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。

注意：当不指定scope时，回remove掉从page到application的var指定的变量

<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。

<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 |
| 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 |
| var | 用来储存test运算后的结果，即true或false | N | String | 否 | 无 |
| scope | var变量的JSP范围 | N | String | 否 | page |

<c:choose>

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

**语法**

<c:choose>

本体内容( <when> 和 <otherwise> )

</c:choose>

**限制**

<c:choose>的本体内容只能有：

·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.**

**<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>

<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>

**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

<%@ 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>

<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.  
  
<c:forEach>并不只是用来浏览集合对象而已，读者可以从语法2中发现，items并不存在，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。

<%@ 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.

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.

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： 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：

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 意义 |
| index | number | 现在指到成员的索引 |
| count | number | 总共已经指到成员的总数 |
| first | boolean | 现在指到的成员是否为第一个成员(true/false) |
| last | boolean | 现在指到的成员是否为最后一个成员(true/false) |

 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：

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

还有一个例子：

我要获取：http://localhost:8080/t/index.jsp?a=123&b=256&com=297中的参数列表按如下:

键com：297  
键b：256  
键a：123

可以结合EL表单式实现：

<c:forEach items="${param}" var="b">  
        键${b.key}：${b.value }<br/>  
</c:forEach>

<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"]>  
本体内容  
</c:forTokens>**

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

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：

[复制代码](javascript:void(0);)

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

[复制代码](javascript:void(0);)

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。

[fn标签](http://tianhei.iteye.com/blog/1021625)

在使用jstl的函数标签库之前需要在页面中引入：

<%@ taglib prefix="fn" uri="[http://java.sun.com/jsp/jstl/functions"%](http://java.sun.com/jsp/jstl/functions%22%25)>

1、fn:contains(string, substring)   
如果参数string中包含参数substring，返回true。

2、fn:containsIgnoreCase(string, substring)   
如果参数string中包含参数substring（忽略大小写），返回true

3、fn:endsWith(string, suffix)   
如果参数 string 以参数suffix结尾，返回true。

4、fn:escapeXml(string)   
将有特殊意义的XML (和HTML)转换为对应的XML character entity code，并返回。

5、fn:indexOf(string, substring)   
返回参数substring在参数string中第一次出现的位置。

6、fn:join(array, separator)   
将一个给定的数组array用给定的间隔符separator串在一起，组成一个新的字符串并返回。

7、fn:length(item)   
返回参数item中包含元素的数量。参数Item类型是数组、collection或者String。如果是String类型,返回值是String中的字符数。

**Html代码  [收藏代码](javascript:void())**

1. **<body>**
2. **<**%
3. String a[] = {"aa","bb","cc","dd"};
4. request.setAttribute("array",a);
5. request.setAttribute("store","guomei8899");
6. %**>**
7. **<c:if** test="${fn:contains('guomeiddd','guoMei')}"**>**ok**</c:if><br>**
8. **<c:if** test="${fn:containsIgnoreCase(store,'guoMei')}"**>**ok ok**</c:if><br>**
9. **<c:if** test="${fn:endsWith(store,'99')}"**>**end**</c:if><br>**
10. **<c:out** value="${fn:escapeXml('<>')}"**/><br>**
11. **<c:out** value="${fn:indexOf(store,'om')}"**/><br>**
12. **<c:out** value="${fn:join(array,'|')}"**/><br>**
13. **<c:out** value="${fn:length(array)}"**/><br>**
14. **</body>**

8、fn:replace(string, before, after)   
返回一个String对象。用参数after字符串替换参数string中所有出现参数before字符串的地方，并返回替换后的结果。

9、fn:startsWith(string, prefix)   
如果参数string以参数prefix开头，返回true。

10、fn:substring(string, begin, end)   
返回参数string部分字符串, 从参数begin开始到参数end位置。

11、fn:substringAfter(string, substring)   
返回参数substring在参数string中后面的那一部分字符串。

12、fn:substringBefore(string, substring)   
返回参数substring在参数string中前面的那一部分字符串。

13、fn:toLowerCase(string)   
将参数string所有的字符变为小写，并将其返回。

14、fn:toUpperCase(string)   
将参数string所有的字符变为大写，并将其返回。

15、fn:trim(string)   
去除参数string 首尾的空格，并将其返回。

**Html代码  [收藏代码](javascript:void())**

1. **<body>**
2. **<**%
3. String a[] = {"aa","bb","cc","dd"};
4. request.setAttribute("array",a);
5. request.setAttribute("store","guomei8899");
6. request.setAttribute("user","u1,u2,u3,u4,u5");
7. request.setAttribute("test","aBcDeF   ");
8. %**>**
9. **<c:out** value="${fn:replace(store,'8','9')}"**/><br>**
10. **<c:out** value="${fn:split(user,',')}"**/><br>**
11. **<c:out** value="${fn:startsWith(store,'g')}"**/><br>**
12. **<c:out** value="${fn:substring(store,2,5)}"**/><br>**
13. **<c:out** value="${fn:substringAfter(store,'mei')}"**/><br>**
14. **<c:out** value="${fn:substringBefore(store,'mei')}"**/><br>**
15. **<c:out** value="${fn:toLowerCase(test)}"**/><br>**
16. **<c:out** value="${fn:toUpperCase(test)}"**/><br>**
17. **<c:out** value="${test}hoho"**/><br>**
18. **<c:out** value="${fn:trim(test)}hoho"**/><br>**
19. **</body>**

fmt标签

formatting标签库:就是用于在 JSP 页面中做国际化格式化的动作分为了两类，分别是:

国际化核心标签:<fmt:setLocale>、<fmt:bundle>、<fmt:setBundle>、<fmt:message>、<fmt:param>、<fmt:requestEncoding>  
格式化标签:<fmt:timeZone>、<fmt:setTimeZone>、<fmt:formatNumber>、<fmt:parseNumber>、<fmt:formatDate>、<fmt:parseDate>  
  
**1.<fmt:setLocale>标签:用于设置本地化环境**属性描述   
value:Locale 环境的指定，可以是 java.util.Locale 或 String 类型的实例   
scope:Locale 环境变量的作用范围（可选）   
如:  
    设置本地环境为繁体中文  
    <fmt:setLocale value="zh\_TW"/>  
    设置本地环境为简体中文  
    <fmt:setLocale value="zh\_CN"/>  
  
**2.<fmt:requestEncoding>标签:用于为请求设置字符编码**它只有一个属性 value ，在该属性中可以定义字符编码。   
如：  
    <fmt:requestEncoding value="GB2312"/>  
  
**3.<fmt:bundle> 、 <fmt:setBundle> 标签:用于资源配置文件的数据来源**3.1<fmt:bundle> 标签将资源配置文件绑定于它标签体中的显示  
属性描述  
basename:资源配置文件的指定，只需要指定文件名而无须扩展名  
prefix:前置关键字  
如:  
资源文件中配置的数据为:  
label.backcolor=#FFF  
label.fontcolor=#000  
则,可以用如下方法取得label的backcolor和fontcolor值:  
    <fmt:bundle basename="MyResourse" prefix="label.">   
        <fmt:message key="backcolor" />  
        <fmt:message key="fontcolor" />  
    </fmt:bundle>  
  
3.2<fmt:setBundle> 标签则允许将资源配置文件保存为一个变量，在之后的工作可以根据该变量来进行  
属性描述 ，二组标签共有的属性   
var:<fmt:setBundle> 独有的属性，用于保存资源配置文件为一个变量   
scope:变量的作用范围   
如:  
    查找一个名为 applicationMessage\_zh\_CN.properties 的资源配置文件，来作为显示的 Resource 绑定  
    <fmt:setBundle basename="applicationMessage" var="applicationBundle"/>   
  
  
**4.<fmt:message> 标签:用于显示资源配置文件信息(该资源文件必须遵循如下格式:1.扩展名必须为properties,2.文件的内容必须依照key = value的格式;3.文件要放到WEB-INF/classes目录下)**属性描述   
key:资源配置文件的“键”指定   
bundle:若使用 <fmt:setBundle> 保存了资源配置文件，该属性就可以从保存的资源配置文件中进行查找   
var:将显示信息保存为一个变量   
scope:变量的作用范围   
如:  
1)用<fmt:setBundle>标签将"applicationMessage"资源配置文件被赋于了变量"applicationBundle"  
    用<fmt:message>标签显示由<fmt:setBundle>标签保存的资源配置文件中"键"为"passWord"的信息  
      
        <fmt:setBundle basename="applicationMessage" var="applicationBundle"/>   
        <fmt:message key="passWord" bundle="${applicationBundle}" />  
  
2)用<fmt:bundle>标签定义的"applicationAllMessage"资源配置文件作用于其标签体内的显示  
    用<fmt:message>标签显示"applicationAllMessage"资源配置文件中"键"为"userName"的信息  
  
        <fmt:bundle basename="applicationAllMessage">   
            <fmt:message key="userName" />  
        </fmt:bundle>  
  
**5.<fmt:param 标签:用于参数传递**<fmt:param>标签应该位于 <fmt:message> 标签内，将为该消息标签提供参数值。它只有一个属性value   
如：在MyResourse.properties文件中,有一个索引值如下(其中,{0}代表占位符):  
Str2=Hi,{0}   
则,使用<fmt:param>标签传入值如下:  
    <fmt:bundle basename="MyResourse">   
        <fmt:message key="Str2">  
            <fmt:param value="张三" />  
        </fmt:message>  
    </fmt:bundle>  
也可以在资源文件中指定参数的类型:  
如:在MyResourse.properties文件中,有一个索引值如下:  
Str3={0,date}  
则,使用<fmt:param>标签传入值如下:  
    <% request.setAttribute("now",new Date()); %>  
    <fmt:bundle basename="MyResourse">   
        <fmt:message key="Str3">  
            <fmt:param value="${now}" />  
        </fmt:message>  
    </fmt:bundle>  
  
  
**6.<fmt:timeZone>、<fmt:setTimeZone>标签:用于设定时区**<fmt:timeZone> 标签将使得在其标签体内的工作可以使用该时区设置  
<fmt:setTimeZone> 标签则允许将时区设置保存为一个变量，在之后的工作可以根据该变量来进行  
属性描述   
value:时区的设置   
var:<fmt:setTimeZone> 独有的属性，用于保存时区为一个变量   
scope:变量的作用范围   
  
  
**7.<fmt:formatNumber>标签:用于格式化数字**属性描述   
value:格式化的数字,该数值可以是 String 类型或 java.lang.Number 类型的实例   
type:格式化的类型,可能值包括:currency（货币）、number（数字）和percent（百分比）  
pattern:格式化模式   
var:结果保存变量   
scope:变量的作用范围   
maxIntegerDigits:指定格式化结果的最大值   
minIntegerDigits:指定格式化结果的最小值   
maxFractionDigits:指定格式化结果的最大值，带小数   
minFractionDigits:指定格式化结果的最小值，带小数   
  
如:  
    结果将被保存在“ money ”变量中，将根据 Locale 环境显示当地的货币格式  
        <fmt:formatNumber value="1000.888" type="currency" var="money"/>  
  
  
**8.<fmt:parseNumber> 标签:用于解析数字**属性描述   
value:将被解析的字符串   
type:解析格式化的类型   
pattern:解析格式化模式   
var:结果保存变量，类型为 java.lang.Number   
scope:变量的作用范围   
parseLocale:以本地化的形式来解析字符串，该属性的内容应为 String 或 java.util.Locale 类型的实例   
  
如:  
    将"15%"转换为数字  
        <fmt:parseNumber value="15%" type="percent" var="num"/>   
  
  
**9.<fmt:formatDate>标签:用于格式化日期**属性描述  
value:格式化的日期，该属性的内容应该是 java.util.Date 类型的实例  
type:格式化的类型  
pattern:格式化模式  
var:结果保存变量  
scope:变量的作用范围  
timeZone:指定格式化日期的时区  
  
  
**10.<fmt:parseDate>标签:用于解析日期**属性描述   
value:将被解析的字符串   
type:解析格式化的类型   
pattern:解析格式化模式   
var:结果保存变量，类型为 java.lang.Date   
scope:变量的作用范围   
parseLocale:以本地化的形式来解析字符串，该属性的内容为 String 或 java.util.Locale 类型的实例   
timeZone:指定解析格式化日期的时区