JSTL 概述

概述

在 JSP 页面中，使用标签库代替传统的 Java 片段语言来实现页面的显示逻辑已经不是新技术了，然而，由自定义标签很容易造成重复定义和非标准的实现。鉴于此，出现了 JSTL （ JSP Standard Tag Library ）。大多数 JSP 页面逻辑提供了实现的 JSTL 技术，该技术本身就是一个标签库。

Sun 公司 Java 规范标准的 JSTL 由 apache jakarta 组织负责维护。作为开源的标准技术，它一直在不断地完善。 JSTL 的发布包有两个版本： Standard-1.0 Taglib 、 Standard-1.1 Taglib ，它们在使用时是不同的。

- Standard-1.0 Taglib （ JSTL1.0 ）支持 Servlet2.3 和 JSP1.2 规范， Web 应用服务器 Tomcat4 支持这些规范，而它的发布也在 Tomcat 4.1.24 测试通过了。
- Standard-1.1 Taglib （ JSTL1.1 ）支持 Servlet2.4 和 JSP2.0 规范， Web 应用服务器 Tomcat5 支持这些规范，它的发布在 Tomcat 5.0.3 测试通过了。

在本章的介绍中，将以由 Sun 发布的 Standard-1.1 Taglib 标签库为主，而 apache jakarta 组织发布的开源标签库，可以从 http://jakarta.apache.org/taglibs/ 找到所需要的帮助。 Sun 发布的标准 JSTL1.1 标签库有以下几个标签：

- 核心标签库：包含 Web 应用的常见工作，比如：循环、表达式赋值、基本输入输出等。
- 国际化标签库：用来格式化显示数据的工作，比如：对不同区域的日期格式化等。
- 数据库标签库：可以做访问数据库的工作。
- XML 标签库：用来访问 XML 文件的工作，这是 JSTL 标签库的一个特点。
- 函数标签库：用来读取已经定义的某个函数。

此外， JSTL 还提供了 EL 表达式语言（ Expression Language ）来进行辅助的工作。

JSTL 标签 库由标签库和 EL 表达式语言两个部分组成。 EL 在 JSTL 1.0 规范中被引入，当时用来作为 Java 表达式来工作而该表达式必须配合 JSTL 的标签库才能得到需要的结果。

说明：在 JSTL 1.1 规范中， JSP2.0 容器已经能够独立的理解任何 EL 表达式。 EL 可以独立出现在 JSP 页面的任何角落。本文随后的内容将以 JSTL 1.1 规范作为介绍的重点。

9.2.3 EL 表达式的操作符

EL 表达式中还有许多操作符可以帮助完成各种所需的操作，之前的示例中“ . ”、“ [] ”就是其中的两个，下面将用表 9.1 来展示所有操作符及它们各自的功能。

表 9.1 EL 表达式的操作符

操作符	功能和作用
.	访问一个 bean 属性或者 Map entry
[]	访问一个数组或者链表元素
()	对子表达式分组，用来改变赋值顺序
? :	条件语句，比如：条件 ?ifTrue:ifFalse 如果条件为真，表达式值为前者，反之为后者

+	数学运算符，加操作
-	数学运算符，减操作或者对一个值取反
*	数学运算符，乘操作
/ 或 div	数学运算符，除操作
% 或 mod	数学运算符，模操作（ 取余 ）
== 或 eq	逻辑运算符，判断符号左右两端是否相等，如果相等返回 true ，否则返回 false
!= 或 ne	逻辑运算符，判断符号左右两端是否不相等，如果不相等返回 true ，否则返回 false
< 或 lt	逻辑运算符，判断符号左边是否小于右边，如果小于返回 true ，否则返回 false
> 或 gt	逻辑运算符，判断符号左边是否大于右边，如果大于返回 true ，否则返回 false
<= 或 le	逻辑运算符，判断符号左边是否小于或者等于右边，如果小于或者等于返回 true ， 否则返回 false
>= 或 ge	逻辑运算符，判断符号左边是否大于或者等于右边，如果大于或者等于返回 true ， 否则返回 false
&& 或 and	逻辑运算符，与操作赋。如果左右两边同为 true 返回 true ， 否则返回 false
‖ 或 or	逻辑运算符，或操作赋。如果左右两边有任何一边为 true 返回 true ， 否则返回 false
! 或 not	逻辑运算符，非操作赋。如果对 true 取运算返回 false ， 否则返回 true
empty	用来对一个空变量值进行判断 ：null 、一个空 String 、空数组、空 Map 、没有条目的 Collection 集合
func(args)	调用方法 ，func 是方法名， args 是参数，可以没有，或者有一个、多个参数 ．参数间用逗号隔开

这些操作符都是极其有用的，下面通过几个示例来演示它们的使用方法：

例 9.4 ： 几组操作符的示例

```
${pageScope.sampleValue + 12} <br>              // 显示 12
${(pageScope.sampleValue + 12)/3} <br>       // 显示 4.0
${(pageScope.sampleValue + 12) /3==4} <br>          // 显示 true
```

```
${(pageScope.sample1Value + 12) /3>=5} <br>          // 显示 false
<input type="text" name="sample1" value="${pageScope.sample1Value + 10}"> // 显示值为 10 的 Text 控件
```

可以看到，对于这些示例，程序设计者完全无需管理它们的类型转换，在表达式内部都已经处理了。有了 EL 表达式，在 JSP 页面的编程变得更灵活，也更容易。

9.2.3 JSTL 标签库介绍

在 JSTL1.1 中有以下这些标签库是被支持的： Core 标签库、 XML processing 标签库、 I18N formatting 标签库、 Database access 标签库、 Functions 标签库。对应的标识符见表 9.2 所示：

表 9.2 标签库的标识符

标签库	URI	前缀
Core	http://java.sun.com/jsp/jstl/core	c
XML processing	http://java.sun.com/jsp/jstl/xml	x
I18N formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Database access	http://java.sun.com/jsp/jstl/sql	sql
Functions	http://java.sun.com/jsp/jstl/functions	fn

下面看例 9.5 ，简单使用标签库的示例。

例 9.5 ： 简单 JSTL 标签库示例

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
    <c:forEach var="i" begin="1" end="10" step="1">
      ${i}
      <br />
    </c:forEach>
  </body>
</html>
```

在该示例的 JSP 页面中声明了将使用 Core 标签库，它的 URI 为“ http://java.sun.com/jsp/jstl/core ”，前缀为“ c ”。之后，页面中 <c:forEach> 标签就是使用了 JSTL 的标签进行了工作。对于该标签的功能，这里暂时不作具体讲解，只是让读

者能够有个简单的概念，了解怎样定义和使用标签库。

9.3 JSTL Core 标签库

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：

- 多用途核心标签：<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> 。

以下是各个标签的用途和属性以及简单示例。

9.3.1 用于显示的 <c:out> 标签

<c:out> 标签是一个最常用的标签，用于在 JSP 中显示数据。它的属性和描述如表 9.3 所示：

表 9.3 <c:out> 标签属性和说明

属性	描述
value	输出到页面的数据，可以是 EL 表达式或常量（必须）
default	当 value 为 null 时显示的数据（可选）
escapeXml	当设置为 true 时会主动更换特殊字符，比如“ <,>,& ”（可选，默认为 true ）

在 JSTL1.0 的时候，在页面显示数据必须使用 <c:out> 来进行。然而，在 JSTL1.1 中，由于 JSP2.0 规范已经默认支持了 EL 表达式 ，因此可以直接在 JSP 页面使用表达式。下面看一个示例。

```
<c:out value="${sessionScope.anyValue}" default="no value" escapeXml="false"/>
```

该示例将从 Session 查找名为“ anyValue ”的参数，并显示在页面，若没有找到则显示“ no value ”。

9.3.2 用于赋值的 <c:set> 标签

<c:set> 标签用于为变量或 JavaBean 中的变量属性赋值的工作。它的属性和描述如表 9.4 所示：

表 9.4 <c:set> 标签属性和说明

属性	描述
value	值的信息，可以是 EL 表达式或常量
target	被赋值的 JavaBean 实例的名称，若存在该属性则必须存在 property 属性（可选）
proper	JavaBean 实例的变量属性名称（可选）

ty	
var	被赋值的变量名（可选）
scope	变量的作用范围，若没有指定，默认为 page （可选）

当不存在 value 的属性时，将以包含在标签内的实体数据作为赋值的内容。下面看一个示例：

```
<c:set value="this is andy" var="oneString"/>
${oneString} <br>
```

该示例将为名为“ oneString ”的变量赋值为“ this is andy ”，其作用范围为 page 。

9.3.3 用于删除的 <c:remove> 标签

<c:remove> 标签用于删除存在于 scope 中的变量。它的属性和描述如表 9.5 所示：

表 9.5 <c:remove> 标签属性和说明

属性	描述
var	需要被删除的变量名
scope	变量的作用范围，若没有指定，默认为全部查找（可选）

下面看一个示例：

```
<c:remove var="sampleValue" scope="session"/>
${sessionScope.sampleValue} <br>
```

该示例将存在于 Session 中名为“ sampleValue ”的变量删除。下一句 EL 表达式显示该变量时，该变量已经不存在了。

9.3.4 用于异常捕获的 <c:catch> 标签

<c:catch> 标签允许在 JSP 页面中捕捉异常。它包含一个 var 属性，是一个描述异常的变量，改变量可选。若没有 var 属性的定义，那么仅仅捕捉异常而不做任何事情，若定义了 var 属性，则可以利用 var 所定义的异常变量进行判断转发到其他页面或提示报错信息。看一个示例。

```
<c:catch var="err">
        ${param.sampleSingleValue[9] == 3}
</c:catch>
${err}
```

当“ ${param.sampleSingleValue[9] == 3} ”表达式有异常时，可以从 var 属性“ err ”得到异常的内容，通常判断“ err ”是否为 null 来决定错误信息的提示。

9.3.5 用于判断的 <c:if> 标签

<c:if> 标签用于简单的条件语句。它的属性和描述如表 9.6 所示：

表 9.6 <c:if> 标签属性和说明

属性	描述
test	需要判断的条件
var	保存判断结果 true 或 false 的变量名，该变量可供之后的工作使用（可选）
scope	变量的作用范围，若没有指定，默认为保存于 page 范围中的变量（可选）

下面看一个示例:

```
<c:if test="${paramValues.sampleValue[2] == 12}" var="visits">
      It is 12
</c:if><br>
${visits} <br>
```

该示例将判断 request 请求提交的传入控件数组参数中，下标为“ 2 ”的控件内容是否为“ 12 ”，若为 12 则显示“ It is 12 ”。判断结果被保存在 page 范围中的“ visits ”变量中。

9.3.6 用于复杂判断的 <c:choose> 、 <c:when> 、 <c:otherwise> 标签

这三个标签用于实现复杂条件判断语句，类似“ if,elseif ”的条件语句。

- <c:choose> 标签没有属性，可以被认为是父标签， <c:when> 、 <c:otherwise> 将作为其子标签来使用。
- <c:when> 标签等价于“ if ”语句，它包含一个 test 属性，该属性表示需要判断的条件。
- <c:otherwise> 标签没有属性，它等价于“ else ”语句。

下面看一个复杂条件语句的示例。

```
<c:choose>
            <c:when test="${paramValues.sampleValue[2] == 11}">
                        not 12 not 13,it is 11
            </c:when>
            <c:when test="${paramValues.sampleValue[2] == 12}">
                        not 11 not 13,it is 12
            </c:when>
            <c:when test="${paramValues.sampleValue[2] == 13}">
                        not 11 not 12,it is 13
            </c:when>
            <c:otherwise>
                        not 11 、 12 、 13
            </c:otherwise>
</c:choose>
```

该示例将判断 request 请求提交的传入控件数组参数中，下标为“ 2 ”控件内容是否为“ 11 ”或“ 12 ”或“ 13 ”，并根据判断结果显示各自的语句，若都不是则显示“ not 11 、 12 、 13 ”。

9.3.7 用于循环的 <c:forEach> 标签

<c:forEach> 为循环控制标签。它的属性和描述如表 9.7 所示:

表 9.7　<c:forEach> 标签属性和说明

属性	描述
items	进行循环的集合（可选）
begin	开始条件（可选）
end	结束条件（可选）
step	循环的步长，默认为 1 （可选）
var	做循环的对象变量名，若存在 items 属性，则表示循环集合中对象的变量名（可选）
varStatus	显示循环状态的变量（可选）

下面看一个集合循环的示例。

```
<%ArrayList arrayList = new ArrayList();
                        arrayList.add("aa");
                        arrayList.add("bb");
                        arrayList.add("cc");
%>
<%request.getSession().setAttribute("arrayList", arrayList);%>
<c:forEach items="${sessionScope.arrayList}" var="arrayListI">
            ${arrayListI}
</c:forEach>
```

该示例将保存在 Session 中的名为 “ arrayList ” 的 ArrayList 类型集合参数中的对象依次读取出来， items 属性指向了 ArrayList 类型集合参数， var 属性定义了一个新的变量来接收集合中的对象。最后直接通过 EL 表达式显示在页面上。下面看一个简单循环的示例。

```
<c:forEach var="i" begin="1" end="10" step="1">
        ${i}<br />
</c:forEach>
```

该示例从 “ 1 ” 循环到 “ 10 ” ，并将循环中变量 “ i ” 显示在页面上。

9.3.8　用于分隔字符的 <c:forTokens> 标签

<c:forTokens> 标签可以根据某个分隔符分隔指定字符串，相当于 java.util.StringTokenizer 类。它的属性和描述如表 9.8 所示。

表 9.8　<c:forTokens> 标签 属性和说明

属性	描述
items	进行分隔的 EL 表达式或常量
delims	分隔符
begin	开始条件（可选）

end	结束条件（可选）
step	循环的步长，默认为 1（可选）
var	做循环的对象变量名（可选）
varStatus	显示循环状态的变量（可选）

下面看一个示例。

```
<c:forTokens items="aa,bb,cc,dd" begin="0" end="2" step="2" delims="," var="aValue">
            ${aValue}
</c:forTokens>
```

需要分隔的字符串为“ aa,bb,cc,dd ”，分隔符为“ , ”。 begin 属性 指定从第一个“ , ”开始分隔， end 属性指定分隔到第三个“ , ”，并将做循环的变量名指定为“ aValue ”。由于步长为“ 2 ”，使用 EL 表达式 ${aValue} 只能显示“ aa

```
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('AjaxHolder$scriptmanager1',
document.getElementById('Form1'));
Sys.WebForms.PageRequestManager.getInstance()._updateControls(['tAjaxHolder$UpdatePanel1'], [],
[], 90);
//]]>function pageLoad()
{
    Sys.WebForms.PageRequestManager.getInstance().add_initializeRequest(handleInitializeRequ
est);
    //Sys.WebForms.PageRequestManager.getInstance().add_endRequest(handleEndRequest);
}

function handleInitializeRequest(sender, args)
{
    var prm = Sys.WebForms.PageRequestManager.getInstance();
    var eid = args.get_postBackElement().id;
    if (eid.indexOf("DeleteLink")>0)
    {
        args.get_postBackElement().innerHTML = "<font color='red'>正在删除...</font>";

    }
    else if (eid.indexOf("btnSubmit")>0)
    {
        document.getElementById("AjaxHolder_PostComment_ltSubmitMsg").innerHTML="正在提交...";

        document.getElementById("AjaxHolder_PostComment_btnSubmit").disabled = true;
```

```
    }
    else if(eid.indexOf("refreshList")>0)
    {
        document.getElementById("AjaxHolder_PostComment_refreshList").innerHTML="<font
color='red'>正在刷新...</font>";
    }

}

function TempSave(ElementID)
{
      try
      {

CommentsPersistDiv.setAttribute("CommentContent",document.getElementById(ElementID).value);
      CommentsPersistDiv.save("CommentXMLStore");
      }
      catch(ex)
      {
      }

}
function Restore(ElementID)
{
      CommentsPersistDiv.load("CommentXMLStore");

document.getElementById(ElementID).value=CommentsPersistDiv.getAttribute("CommentContent");
}
```

9.3.9 用于包含页面的 <c:import>

<c:import> 标签允许包含另一个 JSP 页面到本页面来。它的属性和描述如表 9.9 所示:

表 9.9 <c:import> 标签属性和说明

属性	描述
url	需要导入页面的 URL
context	Web Context 该属性用于在不同的 Context 下导入页面，当出现 context 属性时，必须以“/”开头，此时也需要 url 属性以“/”开头

	（可选）
charEncoding	导入页面的字符集（可选）
var	可以定义导入文本的变量名（可选）
scope	导入文本的变量名作用范围（可选）
varReader	接受文本的 java.io.Reader 类变量名（可选）

下面看一个示例。

```
<c:import url="/MyHtml.html" var="thisPage" />
<c:import url="/MyHtml.html" context=" /sample2"  var="thisPage"/>
<c:import url="www.sample.com/MyHtml.html" var="thisPage"/>
```

该示例演示了三种不同的导入方法，第一种是在同一 Context 下的导入，第二种是在不同的 Context 下导入，第三种是导入任意一个 URL 。

9.3.10 用于得到 URL 地址的 <c:url> 标签

<c:url> 标签用于得到一个 URL 地址。它的属性和描述如表 9.10 所示：

表 9.10 <c:url> 标签属性和说明

属性	描述
value	页面的 URL 地址
context	Web Context 该属性用于得到不同 Context 下的 URL 地址，当出现 context 属性时，必须以“ / ”开头，此时也需要 url 属性以“ / ”开头（可选）
charEncoding	URL 的字符集（可选）
var	存储 URL 的变量名（可选）
scope	变量名作用范围（可选）

下面看一个示例：

```
<c:url value="/MyHtml.html" var="urlPage" />
<a href="${urlPage}">link</a>
```

得到了一个 URL 后，以 EL 表达式放入 <a> 标签的 href 属性，达到链接的目的。

9.3.11 用于页面重定向的 <c:redirect> 标签

<c:redirect> 用于页面的重定向，该标签的作用相当于 response.setRedirect 方法的工作。它包含 url 和 context 两个属性，

属性含义和 <C:url> 标签相同。下面看一个示例。

```
<c:redirect url="/MyHtml.html"/>
```

该示例若出现在 JSP 中，则将重定向到当前 Web Context 下的“ MyHtml.html ”页面，一般会与 <c:if> 等标签一起使用。

9.3.12 用于包含传递参数的 <c:param> 标签

<c:param> 用来为包含或重定向的页面传递参数。它的属性和描述如表 9.11 所示：

表 9.11 <c:param> 标签属性和说明

属性	描述
name	传递的参数名
value	传递的参数值（可选）

下面是一个示例：

```
<c:redirect url="/MyHtml.jsp">
<c:param name="userName" value=" RW"  />
</c:redirect>
```

该示例将为重定向的“ MyHtml.jsp ”传递指定参数“ userName=’ RW’ ”。

9.4 JSTL XML processing 标签库

在企业级应用越来越依赖 XML 的今天， XML 格式的数据被作为信息交换的优先选择。 XML processing 标签库为程序设计者提供了基本的对 XML 格式文件的操作。在该标签库中的标签一共有 10 个，被分为了三类，分别是：

- XML 核心标签： <x:parse> 、 <x:out> 、 <x:set> 。
- XML 流控制标签： <x:if> 、 <x:choose> 、 <x:when> 、 <x:otherwise> 、 <x:forEach> 。
- XML 转换标签： <x:transform> 、 <x:param> 。

由于该组标签库专注于对某一特定领域的实现，因此本书将只选择其中常见的一些标签和属性进行介绍。

9.4.1 用于解析 XML 文件的 <x:parse> 标签

<x:parse> 标签是该组标签库的核心，从其标签名就可以知道，它是作为解析 XML 文件而存在的。它的属性和描述如表 9.12 所示：

表 9.12 <x:parse> 标签属性和说明

属性	描述
doc	源 XML 的内容，该属性的内容应该为 String 类型或者 java.io.Reader 的实例，可以用 xml 属性来替代，但是不被推荐
var	将解析后的 XML 保存在该属性所指定的变量中，之后 XML processing 标签库中的其他标签若要取 XML 中的内容就可以

	从该变量中得到（可选）
scope	变量的作用范围（可选）
varDom	指定保存的变量为 org.w3c.dom.Document 接口类型（可选）
scopeDom	org.w3c.dom.Document 的接口类型变量作用范围（可选）
systemId	定义一个 URI ，该 URI 将被使用到 XML 文件中以接入其他资源文件（可选）
filter	该属性必须为 org.xml.sax.XMLFilter 类的一个实例，可以使用 EL 表达式传入，将对 XML 文件做过滤得到自身需要的部分（可选）

其中， var 、 scope 和 varDom 、 scopeDom 不应该同时出现，而应该被视为两个版本来使用，二者的变量都可以被 XML processing 标签库的其他标签来使用。

<x:parse> 标签单独使用的情况很少，一般会结合 XML processing 标签库中的其他标签来一起工作。下面看一个示例。

首先给出一个简单的 XML 文件，将对该 XML 文件做解析，该 XML 文件名为 SampleXml.xml 。

```
<?xml version="1.0" encoding="UTF-8"?>
<xml-body>
            <name>RW</name>
            <passWord>123456</passWord>
            <age>28</age>
            <books>
                        <book>book1</book>
                        <book>book2</book>
                        <book>book3</book>
            </books>
</xml-body>
```

标签库的工作：

```
<c:import var="xmlFile" url="http://localhost:8080/booksample.jstl/SampleXml.xml"/>
<x:parse var="xmlFileValue" doc="${xmlFile}"/>
```

```
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('ajaxHolder$scriptmanager1',
document.getElementById('form1'));
Sys.WebForms.PageRequestManager.getInstance()._updateControls(['tajaxHolder$UpdatePanel1'], [],
[], 90);
//]]>
```

9.5 I18N formatting 标签库

看到 I18N 就应该想到知识“国际化”， I18N formatting 标签库就是用于在 JSP 页面中做国际化的动作。在该标签库中的标签一共有 12 个，被分为了两类，分别是：

q 国际化核心标签： <fmt:setLocale> 、 <fmt:bundle> 、 <fmt:setBundle> 、 <fmt:message> 、 <fmt:param> 、 <fmt:requestEncoding> 。

q 格式化标签：<fmt:timeZone>、<fmt:setTimeZone>、<fmt:formatNumber>、<fmt:parseNumber>、<fmt:formatDate>、<fmt:parseDate>。

下面只选择其中常见的一些标签和属性进行介绍。

9.5.1 用于设置本地化环境的 <fmt:setLocale> 标签

<fmt:setLocale> 标签用于设置 Locale 环境。它的属性和描述如表 9.17 所示：

表 9.17 <fmt:setLocale> 标签属性和说明

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）

下面看一个示例：

```
<fmt:setLocale value="zh_TW"/>
```

表示设置本地环境为繁体中文。

9.5.2 用于资源文件绑定的 <fmt:bundle>、<fmt:setBundle> 标签

这两组标签用于资源配置文件的绑定，唯一不同的是 <fmt:bundle> 标签将资源配置文件绑定于它标签体中的显示，<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。

根据 Locale 环境的不同将查找不同后缀的资源配置文件，这点在国际化的任何技术上都是一致的，通常来说，这两种标签单独使用是没有意义的，它们都会与 I18N formatting 标签库中的其他标签配合使用。它们的属性和描述如表 9.18 所示：

表 9.18 <fmt:bundle>、<fmt:setBundle> 标签属性和说明

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，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

下面看一个示例

```
<fmt:setLocale value="zh_CN"/>
<fmt:setBundle basename="applicationMessage" var="applicationBundle"/>
```

该示例将会查找一个名为 applicationMessage_zh_CN.properties 的资源配置文件，来作为显示的 Resource 绑定。

9.5.3 用于显示资源配置文件信息的 <fmt:message> 标签

用于信息显示的标签，将显示资源配置文件中定义的信息。它的属性和描述如表 9.19 所示：

表 9.19 <fmt:message> 标签属性和说明

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

下面看一个示例：

```
<fmt:setBundle basename="applicationMessage" var="applicationBundle"/>
<fmt:bundle basename="applicationAllMessage">
        <fmt:message key="userName" />
        <p>
        <fmt:message key="passWord" bundle="${applicationBundle}" />
</fmt:bundle>
```

该示例使用了两种资源配置文件的绑定的做法，“ applicationMessage ”资源配置文件利用 <fmt:setBundle> 标签被赋于了变量“ applicationBundle ”，而作为 <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。

q 第一个 <fmt:message> 标签将使用“ applicationAllMessage ”资源配置文件中“键”为“ userName ”的信息显示。

q 第二个 <fmt:message> 标签虽然被定义在 <fmt:bundle> 标签体内，但是它使用了 bundle 属性，因此将指定之前由 <fmt:setBundle> 标签保存的“ applicationMessage ”资源配置

文件，该“键”为“ passWord ”的信息显示。

9.5.4 用于参数传递的 <fmt:param> 标签

<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 。

<fmt:param> 标签有两种使用版本，一种是直接将参数值写在 value 属性中，另一种是将参数值写在标签体内。

9.5.6 用于为请求设置字符编码的 <fmt:requestEncoding> 标签

<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。

9.5.7 用于设定时区的 <fmt:timeZone> 、 <fmt:setTimeZone> 标签

这两组标签都用于设定一个时区。唯一不同的是 <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:setBundle> 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。它们的属性和描述如表 9.20 所示：

表 9.20 <fmt:timeZone> 、 <fmt:setTimeZone> 标签 属性和说明

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

9.5.8 用于格式化数字的 <fmt:formatNumber> 标签

<fmt: formatNumber > 标 签用于格式化数字。它的属性和描述如表 9.21 所示：

表 9.21 <fmt:formatNumber> 标签属性和说明

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

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

<fmt:formatNumber> 标签实际是对应 java.util.NumberFormat 类， type 属性的可能值包括 currency （货币）、 number （数字）和 percent （百分比）。

下面看一个示例。

```
<fmt:formatNumber value="1000.888" type="currency" var="money"/>
```

该结果将被保存在“ money ”变量中，将根据 Locale 环境显示当地的货币格式。

9.5.9 用于解析数字的 <fmt:parseNumber> 标签

<fmt:parseNumber> 标签用于解析一个数字，并将结果作为 java.lang.Number 类的实例返回。<fmt:parseNumber> 标签看起来和 <fmt:formatNumber> 标签的作用正好相反。它的属性和描述如表 9.22 所示：

表 9.22 <fmt:parseNumber> 标签属性和说明

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

下面看一个示例。

```
<fmt:parseNumber value="15%" type="percent" var="num"/>
```

解析之后的结果为“ 0.15 ”。

9.5.10 用于格式化日期的 <fmt:formatDate> 标签

<fmt:formatDate> 标签用于格式化日期。它的属性和描述如表 9.23 所示：

表 9.23 <fmt:formatDate> 标签属性和说明

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

<fmt:formatDate> 标签与 <fmt:timeZone> 、 <fmt:setTimeZone> 两组标签的关系密切。若没有指定 timeZone 属性， 也可以通过 <fmt:timeZone> 、 <fmt:setTimeZone> 两组标签设定的时区来格式化最后的结果。

9.5.11 用于解析日期的 <fmt:parseDate> 标签

<fmt:parseDate> 标签用于解析一个日期，并将结果作为 java.lang.Date 类型的实例返回。
<fmt:parseDate> 标签看起来和 <fmt:formatDate> 标签的作用正好相反。它的属性和描述如表 9.24 所示：

表 9.24　<fmt:parseDate> 标签属性和说明

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

<fmt:parseNumber> 和 <fmt:parseDate> 两组标签都实现解析字符串为一个具体对象实例的工作，因此，这两组解析标签对 var 属性的字符串参数要求非常严格。就 JSP 页面的表示层前段来说，处理这种解析本不属于份内之事，因此 <fmt:parseNumber> 和 <fmt:parseDate> 两组标签应该尽量少用，替代工作的地方应该在服务器端表示层的后段，比如在 Servlet 中。

9.6　Database access 标签库

Database access 标签库中的标签用来提供在 JSP 页面中可以与数据库进行交互的功能，虽然它的存在对于早期纯 JSP 开发的应用以及小型的开发有着意义重大的贡献，但是对于 MVC 模型来说，它却是违反规范的。因为与数据库交互的工作本身就属于业务逻辑层的工作，所以不应该在 JSP 页面中出现，而是应该在模型层中进行。

对于 Database access 标签库本书不作重点介绍，只给出几个简单示例让读者略微了解它们的功能。

Database access 标签库有以下 6 组标签来进行工作：<sql:setDataSource>、<sql:query>、<sql:update>、<sql:transaction>、<sql:setDataSource>、<sql:param>、<sql:dateParam>。

9.6.1　用于设置数据源的 <sql:setDataSource> 标签

<sql:setDataSource> 标签用于设置数据源，下面看一个示例：

<sql:setDataSource

```
        var="dataSrc"
        url="jdbc:postgresql://localhost:5432/myDB"
        driver="org.postgresql.Driver"
        user="admin"
        password="llll"/>
```

该示例定义一个数据源并保存在“ dataSrc ”变量内。

9.6.2 用于查询的 <sql:query> 标签

<sql:query> 标签用于查询数据库，它标签体内可以是一句查询 SQL 。下面看一个示例：

```
<sql:query var="queryResults" dataSource="${dataSrc}">
    select * from table1
</sql:query>
```

该示例将返回查询的结果到变量“ queryResults ”中，保存的结果是 javax.servlet.jsp.jstl.sql.Result 类型的实例。要取得结果集中的数据可以使用 <c:forEach> 循环来进行。下面看一个示例。

```
<c:forEach var="row" items="${queryResults.rows}">
    <tr>
        <td>${row.userName}</td>
        <td>${row.passWord}</td>
    </tr>
</c:forEach>
```

“ rows ”是 javax.servlet.jsp.jstl.sql.Result 实例的变量属性之一，用来表示数据库表中的“列”集合，循环时，通过“ ${row.XXX} ”表达式可以取得每一列的数据，“ XXX ”是表中的列名。

9.6.3 用于更新的 <sql:update> 标签

<sql:update> 标签用于更新数据库，它的标签体内可以是一句更新的 SQL 语句。其使用和 <sql:query> 标签没有什么不同。

9.6.4 用于事务处理的 <sql:transaction> 标签

<sql:transaction> 标签用于数据库的事务处理，在该标签体内可以使用 <sql:update> 标签和 <sql:query> 标签，而 <sql:transaction> 标签的事务管理将作用于它们之上。

<sql:transaction> 标签对于事务处理定义了 read_committed 、 read_uncommitted 、

repeatable_read 、 serializable4 个隔离级别。

9.6.5 用于事务处理的 <sql:par am> 、 <sql:d atePa ra m> 标签

这两个标签用于向 SQL 语句提供参数，就好像程序中预处理 SQL 的“ ？ ”一样。 <sql:param> 标签传递除 java.util.Date 类型以外的所有相融参数， <sql:dateParam> 标签则指定必须传递 java.util.Date 类型的参数。

9.7 Functions 标签库

称呼 Functions 标签库为标签库，倒不如称呼其为函数库来得更容易理解些。因为 Functions 标签库并没有提供传统的标签来为 JSP 页面的工作服务，而是被用于 EL 表达式语句中。在 JSP2.0 规范下出现的 Functions 标签库为 EL 表达式语句提供了许多更为有用的功能。 Functions 标签库分为两大类，共 16 个函数。

q 长度函数： fn:length

q 字符串处理函数： fn:contains 、 fn:containsIgnoreCase 、 fn:endsWith 、 fn:escapeXml 、 fn:indexOf 、 fn:join 、 fn:replace 、 fn:split 、 fn:startsWith 、 fn:substring 、 fn:substringAfter 、 fn:substringBefore 、 fn:toLowerCase 、 fn:toUpperCase 、 fn:trim

以下是各个函数的用途和属性以及简单示例。

9.7.1 长度函数 fn:l eng th 函数

长度函数 fn:length 的出现有重要的意义。在 JSTL1.0 中，有一个功能被忽略了，那就是对集合的长度取值。虽然 java.util.Collection 接口定义了 size 方法，但是该方法不是一个标准的 JavaBean 属性方法（没有 get,set 方法），因此，无法通过 EL 表达式“ ${collection.size} ”来轻松取得。

fn:length 函数正是为了解决这个问题而被设计出来的。它的参数为 input ，将计算通过该属性传入的对象长度。该对象应该为集合类型或 String 类型。其返回结果是一个 int 类型的值。下面看一个示例。

```
<%ArrayList arrayList1 = new ArrayList();
                        arrayList1.add("aa");
                        arrayList1.add("bb");
                        arrayList1.add("cc");
%>
<%request.getSession().setAttribute("arrayList1", arrayList1);%>
${fn:length(sessionScope.arrayList1)}
```

假设一个 ArrayList 类型的实例“ arrayListl ”，并为其添加三个字符串对象，使用 fn:length 函数后就可以取得返回结果为“ 3 ”。

9.7.2 判断函数 fn:contains 函数

fn:contains 函数用来判断源字符串是否包含子字符串。它包括 string 和 substring 两个参数，它们都是 String 类型，分布表示源字符串和子字符串。其返回结果为一个 boolean 类型的值。下面看一个示例。

```
${fn:contains("ABC", "a")}<br>
${fn:contains("ABC", "A")}<br>
```

前者返回“ false ”，后者返回“ true ”。

9.7.3 fn:containsIgnoreCase 函数

fn:containsIgnoreCase 函数与 fn:contains 函数的功能差不多，唯一的区别是 fn:containsIgnoreCase 函数对于子字符串的包含比较将忽略大小写。它与 fn:contains 函数相同，包括 string 和 substring 两个参数，并返回一个 boolean 类型的值。下面看一个示例。

```
${fn:containsIgnoreCase("ABC", "a")}<br>
${fn:containsIgnoreCase("ABC", "A")}<br>
```

前者和后者都会返回“ true ”。

9.7.4 词头判断函数 fn:startsWith 函数

fn:startsWith 函数用来判断源字符串是否符合一连串的特定词头。它除了包含一个 string 参数外，还包含一个 subffx 参数，表示词头字符串，同样是 String 类型。该函数返回一个 boolean 类型的值。下面看一个示例。

```
${fn:startsWith ("ABC", "ab")}<br>
${fn:startsWith ("ABC", "AB")}<br>
```

前者返回“ false ”，后者返回“ true ”。

9.7.5 词尾判断函数 fn:endsWith 函数

fn:endsWith 函数用来判断源字符串是否符合一连串的特定词尾。它与 fn:startsWith 函数相同，包括 string 和 subffx 两个参数，并返回一个 boolean 类型的值。下面看一个示例。

```
${fn:endsWith("ABC", "bc")}<br>
${fn:endsWith("ABC", "BC")}<br>
```

前者返回“ false ”，后者返回“ true ”。

9.7.6 字符实体转换函数 fn:escapeXml 函数

fn:escapeXml 函数用于将所有特殊字符转化为字符实体码。它只包含一个 string 参数，返回一个 String 类型的值。

9.7.8 字符匹配函数 fn:indexOf 函数

fn:indexOf 函数用于取得子字符串与源字符串匹配的开始位置，若子字符串与源字符串中的内容没有匹配成功将返回“ -1 ”。它包括 string 和 substring 两个参数，返回结果为 int 类型。下面看一个示例。

```
${fn:indexOf("ABCD","aBC")}<br>
${fn:indexOf("ABCD","BC")}<br>
```

前者由于没有匹配成功，所以返回 -1 ，后者匹配成功将返回位置的下标，为 1 。

9.7.9 分隔符函数 fn:join 函数

fn:join 函数允许为一个字符串数组中的每一个字符串加上分隔符，并连接起来。它的参数、返回结果和描述如表 9.25 所示：

表 9.25 fn:join 函数

参数	描述
array	字符串数组。其类型必须为 String[] 类型
separator	分隔符。其类型必须为 String 类型
返回结果	返回一个 String 类型的值

下面看一个示例。

```
<% String[] stringArray = {"a","b","c"}; %>
<%request.getSession().setAttribute("stringArray", stringArray);%>
${fn:join(sessionScope.stringArray,";")}<br>
```

定义数组并放置到 Session 中，然后通过 Session 得到该字符串数组，使用 fn:join 函数并传入分隔符“ ; ”，得到的结果为“ a;b;c ”。

9.7.10 替换函数 fn:replace 函数

fn:replace 函数允许为源字符串做替换的工作。它的参数、返回结果和描述如表 9.26 所示：

表 9.26　fn:replace 函数

参数	描述
inputString	源字符串。其类型必须为 String 类型
beforeSubstring	指定被替换字符串。其类型必须为 String 类型
afterSubstring	指定替换字符串。其类型必须为 String 类型
返回结果	返回一个 String 类型的值

下面看一个示例。

```
${fn:replace("ABC","A","B")}<br>
```

将“ ABC ”字符串替换为“ BBC ”，在“ ABC ”字符串中用“ B ”替换了“ A ”。

9.7.11　分隔符转换数组函数　fn:spli t 函数

fn:split 函数用于将一组由分隔符分隔的字符串转换成字符串数组。它的参数、返回结果和描述如表 9.27 所示：

表 9.27　fn:split 函数

参数	描述
string	源字符串。其类型必须为 String 类型
delimiters	指定分隔符。其类型必须为 String 类型
返回结果	返回一个 String[] 类型的值

下面看一个示例。

```
${fn:split("A,B,C",",")}<br>
```

将“ A,B,C ”字符串转换为数组 {A,B,C} 。

9.7.12 字符串截取函数 fn:substring 函数

fn:substring 函数用于截取字符串。它的参数、返回结果和描述如表 9.28 所示:

表 9.28 fn:substring 函数

参数	描述
string	源字符串。其类型必须为 String 类型
beginIndex	指定起始下标（值从 0 开始）。其类型必须为 int 类型
endIndex	指定结束下标（值从 0 开始）。其类型必须为 int 类型
返回结果	返回一个 String 类型的值

下面看一个示例。

```
${fn:substring("ABC","1","2")}<br>
```

截取结果为“ B ”。

9.7.14 起始到定位截取字符串函数 fn:substringBefore 函数

fn:substringBefore 函数允许截取源字符从开始到某个字符串。它的参数和 fn:substringAfter 函数相同，不同的是 substring 表示的是结束字符串。下面看一个示例。

```
${fn:substringBefore("ABCD","BC")}<br>
```

截取的结果为“ A ”。

9.7.15 小写转换函数 fn:toLowerCase 函数

fn:toLowerCase 函数允许将源字符串中的字符全部转换成小写字符。它只有一个表示源字符串的参数 string ，函数返回一个 String 类型的值。下面看一个示例。

```
${fn:toLowerCase("ABCD")}<br>
```

转换的结果为“ abcd ”。

9.7.16 大写转换函数 fn:toUpperCase 函数

fn:toUpperCase 函数允许将源字符串中的字符全部转换成大写字符。它与 fn:toLowerCase 函数相同

也只有一个 String 参数，并返回一个 String 类型的值。下面看一个示例。

```
${fn:toUpperCase("abcd")}<br>
```

转换的结果为“ ABCD ”。

9.7.17 空格删除函数 fn:trim 函数

fn:trim 函数将删除源字符串中结尾部分的“空格”以产生一个新的字符串。它与 fn:toLowerCase 函数相同，只有一个 String 参数，并返回一个 String 类型的值。下面看一个示例。

```
${fn:trim("AB C ")}D<br>
```

转换的结果为“ AB CD ”，注意，它将只删除词尾的空格而不是全部，因此“ B ”和“ C ”之间仍然留有一个空格。

9.8 Struts 与 JSTL

9.8.1 JSTL 与 Struts 协同工作

作为服务器端表示层 MVC 经典框架的 Struts ，其突出表现就是在表示层页面流转方面。虽然在显示的视图层， Struts 框架提供了一组功能强大的标签库来帮助运用。但是这组标签库还是比较复杂，例如要取得一个 Session 中的 JavaBean ，需要做两个步骤的动作。

（ 1 ）使用 <bean:define> 标签来定义一个目标 JavaBean 的标识，并从 Session 中取得源 JavaBean 赋给目标 JavaBean 。若该 JavaBean 本身是 String 类型，则只需要设置它的 name 属性，否则还需要设置 property 属性。

（ 2 ）使用 <bean:write> 标签将该 JavaBean 的变量属性显示出来。若该 JavaBean 本身是 String 类型，则只需要设置它的 name 属性，否则还需要设置 property 属性。

下面看一个示例，假设 Session 中有一个参数为“ TEST ”，其值为 String 类型的字符串“ hello ”。那么使用 Struts 框架的 <bean> 标签库的代码就应该是这样：

```
<bean:define id="test" name="TEST" scope="session"/>

<bean:write name="test"/>
```

定义一个目标 JavaBean 的标识“ test ”，然后将从 Session 中的参数“ TEST ”所取得的源 JavaBean 的实例赋给目标 JavaBean 。 <bean:write> 标签会根据 <bean:define> 标签的 id 属性设置自身的 name 属性，来获取目标 JavaBean 并显示出来。由于它们操作的是 String 类型的字符串，因此编码还算比较简单。可是，如果它们操作的是一个非 String 类型的 JavaBean ，那么编码就比较麻烦了。

如果使用的是 JSTL ，这部分的操作就十分简单了，仅仅通过 EL 表达式语言就可以完成了，转换成 EL 表达式的操作编码如下：

```
${sessionScope.TEST}
```

转换成 JSTL ，只要一句表达式就已经完成了 <bean> 标签库需要用两个标签和许多属性才能完成的工作。即使使用的是 JavaBean 中的属性， JSTL 表达式也只需要再加个“ . ”操作符而已。

使用 JSTL 中的 EL 表达式和 JSTL 标签库中的标签，可以简化 Struts 标签库中许多标签的操作。下面就根据具体的对比来进行介绍。

9.8.2 JSTL VS Struts Bean 标签库 ：

Struts 的 Bean 标签库在 EL 表达式没有出现前是十分常用的，无论从 Session 、 request 、 page 或是其他作用范围（ Scope ）中取得参数、或者从标准 JavaBean 中读取变量属性都处理得得心应手。然而，在 EL 表达式出现之后， Struts Bean 标签库的标签在操作的时候就显示出了烦琐的缺点。因此用 EL 表达式来替代 Struts Bean 标签库中的标签是一种较好的做法。

1. <bean:def ine> 标签和 <bean:write> 标签处理 显示被 EL 表达式替换

q 原形： <bean:define> 标签的作用是定义一个 JavaBean 类型的变量，从 Scope 源位置得到该 JavaBean 的实例。 <bean:write> 标签可以通过 JavaBean 变量来做显示的工作。

q 替换方案：利用 EL 表达式来替换。

q 示例比较

<bean:define> 标签和 <bean:write> 标签的动作：

```
<bean:define id="javaBeanName"
name="javaBeanParameter"
property="javaBeanProperty"
scope="request"/>
<bean:write name="javaBeanName"/>
```

EL 表达式的动作：

```
${requestScope.javaBeanParameter.javaBeanProperty}
```

或

```
${requestScope.javaBeanParameter['javaBeanProperty' ]}
```

处理相同的一个动作，使用 define 标签，通常需要记住各种属性的功能，并有选择地根据实际情况来挑选是否需要 property 属性，还要指定其 scope 属性。而 EL 表达式就方便多了，直接使用默认变量 pageScope 、 requestScope 、 sessionScope 、 applicationScope 指定源 JavaBean 作用范围，利用“ . ”操作符来指定 JavaBean 的名称以及利用“ [] ”或“ . ”来指定 JavaBean 中的变量属性。

q 比较结果：无论是可读性还是程序的简洁性方面， EL 表达式无疑要胜过许多，唯一的缺点是 EL 表达式必须使用 Serv1et2.4 以上的规范。

2. <bean:cookie> 、 <bean :header> 、 <bean:para meter> 标签和 <bean:write> 标签

处理显示 被 EL 表达式替换

q 原形： <bean:cookie> 、 <bean:header> 、 <bean:parameter> 标签的作用是，定义一个 JavaBean 类型的变量，从 cookie 、 request header 、 request parameter 中得到该 JavaBean 实例。 <bean:write> 标签可以通过 JavaBean 变量来做显示的工作。

q 替换方案：利用 EL 表达式来替换。

q 示例比较： <bean:parameter> 标签的动作：

<bean:parameter id="requestString" name="requestParameterString" />

<bean:write name="requestString"/>

EL 表达式的动作：

${param.requestParameterString}

q 比较结果： EL 表达式默认的 5 个变量： cookie 、 header 、 headerValues 、 paramValues 、 param 完全可以提供更方便简洁的操作。

3. <bean:i nclude> 标签被 <c:impor t> 标签替换

q 原形： <bean:include> 标签的作用是定义一个 String 类型的变量，可以包含一个页面、一个响应或一个链接。

q 替换方案：利用 <c:import> 标签来替换。

q 示例比较

<bean:include> 标签的动作：

<bean:include page="/MyHtml.html" id="thisurlPage" />

<c:import> 标签的动作：

<c:import url="/MyHtml.html" var="thisurlPage" />

<bean:include> 标签的 page 属性所起的作用可以由 <c:import> 标签来替换，二者的操作结果是一样的。

q 比较结果：这一对标签的比较没有明显区别，而 <bean:include> 标签有更多属性提供更多功能，因此替换并不是十分必要。

尤其是当要用到配置在 struts-config.xml 中的 <global-forwards> 元素进行全局转发页面时，必须使用 <bean:include> 标签的 forward 元素来实现。

4. <bean :mes sage > 标签处理资源配 置文件被 <fmt:bundle > 、 <fmt:setB undle > 、 <fmt:mes sage > 标签合作替换

q 原形： <bean:message> 标签是专门用来处理资源配置文件显示的，而它的资源配置文件被配置在 struts-config.xml 的 <message-resources> 元素中。

q 替换方案：利用 <fmt:bundle> 、 <fmt:setBundle> 、 <fmt:message> 标签合作来替换，

由 <fmt:bundle> 、 <fmt:setBundle> 设置资源配置文件的实体名称，再由 <fmt:message> 标签负责读取显示。

q 示例 比较

<bean:message> 标签的动作：

```
<bean:message key="message.attacksolution"/>
```

<fmt:bundle> 、 <fmt:message> 标签的动作：

```
<fmt:bundle basename="resources.application">
        <fmt:message key="message.attacksolution" />
</fmt:bundle>
```

或 <fmt:setBundle> 、 <fmt:message> 标签的动作：

```
<fmt:setBundle basename="resources.application" var="resourceaApplication"/>
<fmt:message key="message.attacksolution" bundle="${resourceaApplication}"/>
```

q 比较结果：这一对标签对于国际化的支持都相当好，唯一最大的区别在于利用 <bean:message> 标签所操作的资源配置文件是配置在 struts-config.xml 中的，而 <fmt:message> 标签所操作的资源配置文件则是根据 <fmt:bundle> 、 <fmt:setBundle> 两组标签来得到的。看起来，后者的灵活性不错，但就笔者的眼光来看，前者更为规范，对于用户协作的要求也更高。试想，维护一到两个资源配置文件与维护一大堆资源配置文件哪个更方便呢？自然是前者了，因此除非是不依赖 Struts 框架的应用，否则最好使用 <bean:message> 标签。

9.8.3 JSTL VS Struts Logic 标签库

Struts Logic 标签库中的标签在页面显示时是时常被用到的，但是常用的却不一定是最好用的，有了 JSTL 标签库和 EL 表达式后，许多 Struts Logic 标签库的标签可以被简单替换。

1. 所有判断标签被 EL 表达式和 <c:if> 标签替换

q 原形：判断标签有一个特点，就是需要取得一个实例的变量，因此通过 <bean:define> 标签来取得实例的变量是必须的，随后就通过各种判断标签来完成判断的工作。常用的判断标签如表 9.30 所示：

表 9.30 常用判断标签

标签名	描述
empty	判断变量是否为空
notEmpty	与 empty 标签正好相反
equal	判断变量是否与指定的相同

notEqual	与 equal 标签正好相反
lessThan	判断变量是否比指定的小
greaterThan	判断变量是否比指定的大
lessEqual	判断变量是否小于等于指定的值
greaterEqual	判断变量是否大于等于指定的值
present	检查 header 、 request parameter 、 cookie 、 JavaBean 或 JavaBean propertie 不存在或等于 null 的时候，判断成功
notPresent	与 present 标签正好相反
match	比较 String 类型字符串是否与指定的相同
notMatch	与 match 标签正好相反

q 替换方案：利用 EL 表达式和 <c:if> 标签来替换。

q 示例比较：判断标签的动作：

```
<bean:define id="javaBeanName"
name="javaBeanParameter"
property="attack_event_code"
scope="request"/>
<logic:notEmpty name="javaBeanParameter">
        javaBeanParameter not empty
</logic:notEmpty>
```

EL 表达式和 <c:if> 标签的动作：

```
<c:if test="${requestScope.javaBeanParameter.attack_event_code != null
&& requestScope.javaBeanParameter.attack_event_code != "" }>
        javaBeanParameter not empty
</c:if>
```

EL 表达式利用操作符来完成判断动作，然后通过 <c:if> 标签来根据判断结果处理对应工作。

q 比较结果： EL 表达式的操作符对判断的贡献很大， EL 表达式的灵活性是 Struts 判

断标签无法比拟的，任何判断标签都可以通过表达式来实现。 <c:if> 标签还可以将判断的结果保存为一个变量，随时为之后的页面处理服务。

反观 Struts 框架的判断标签，在工作之前必须先定义被判断的变量，而判断后又无法保存判断结果，这样的程序设计远不如 EL 表达式和 <c:if> 标签的协作来得强大。因此使用 EL 表达式和 <c:if> 标签来替换判断标签是更好的选择。

2. <lo gic:iterate> 标签被 <c:forEa ch> 标签和 EL 表达式替 换

q 原形： <logic:iterate> 标签用来对集合对象的迭代，可以依次从该集合中取得所需要的对象。

q 替换方案：利用 <c:forEach> 标签和 EL 表达式的协作替换 <logic:iterate> 标签。

q 示例比较

<logic:iterate> 标签的动作：

```
<logic:iterate name="allAttackSolution"
     id="attackSolution"
     type="struts.sample.capl.sample3.entity.AttackSolution">
          <bean:write property="attack_event_code" name="attackSolution"/>
          <bean:write property="attack_mean" name="attackSolution"/>
          <bean:write property="attack_action" name="attackSolution"/>
</logic:iterate>
```

<c:forEach> 标签 EL 表达式协作的动作：

```
<c:forEach items="${requestScope.allAttackSolution}" var="attackSolution">
          ${attackSolution.attack_event_code}
          ${attackSolution.attack_mean}
          ${attackSolution.attack_action}
</c:forEach>
```

两个动作都做的是同一件事，从 request 中得到保存的“ allAttackSolution ”参数，该参数为一个集合，集合中的对象为 struts.sample.capl.sample3.entity.AttackSolution 类型的实例。

<logic:iterate> 标签本身可以接收集合，保存为一个变量，利用迭代子模式，使 <logic:iterate> 标签体中的 <bean:write> 标签将集合中的每个 JavaBean 显示出来。

提示：在本例中由于要显示 JavaBean 中的变量属性，因此 <bean:write> 标签还需要设置 property 属性。

替换工作的 <c:forEach> 标签则相对要方便些， items 属性使用 EL 表达式取得集合，然后设置 var 属性作为集合中对象的变量，最后使用 EL 表达式来显示数据。

q 比较结果:

值得注意的一个地方是， <logic:iterate> 标签必须为集合中的对象指定类型，因为标签库处理时会将集合中的对象作为 Object 类型得到，然后需要读取 type 属性定义的 Java 类为它强制转型。

而 <c:forEach> 标签则完全不用，只要符合标准 JavaBean （为变量属性提供 get 、 set 方法）的对象都可以通过 EL 表达式来从 var 属性定义的变量中取得该 JavaBean 的变量属性。

因此 <c:forEach> 标签和 EL 表达式的方式更加简单，也更加灵活。

当然，熟悉 <logic:iterate> 标的程序设计者也可以将 <bean:write> 标签替换为 EL 表达式而仍然使用 <logic:iterate> 标签。代码可以是这样:

```
<logic:iterate name="allAttackSolution"
id="attackSolution"
type="struts.sample.cap1.sample3.entity.AttackSolution">
        ${attackSolution.attack_event_code}
        ${attackSolution.attack_mean}
        ${attackSolution.attack_action}
</logic:iterate>
```

结果一样，但这种方式比 <bean:write> 标签显示方式灵活多了。

3. <lo gic:redirect> 标签被 <c:redirect> 和 <c:param> 标签替换

q 原形： <logic:redirect> 标签用来转发到一个页面，并可以为转发传递参数。

q 替换方案：利用 <c:redirect> 和 <c:param> 标签的协作替换 <logic:redirect> 标签。

q 示例比较： <logic:iterate> 标签的动作:

```
<%
        HashMap paramMap = new HashMap();
        paramMap.put("userName", "RW");
        paramMap.put("passWord", "123456");
%>
 <logic:redirect page="/MyHtml.jsp" name="paramMap" scope="request" />
```

<c:redirect> 和 <c:param> 标签协作的动作:

```
<c:redirect url="/MyHtml.jsp">
        <c:param name="userName" value="RW"/>
        <c:param name="passWord" value="123456"/>
```

</c:redirect>

两个动作都做的是同一件事，都将转发到当前 Web Context 下的“ MyHtml.jsp ”去，而且都将为它提供两个参数。最后的转发链接看起来应该如下所示：

http://localhost:8080/test/ MyHtml.jsp? userName=RW&password=123456

q 比较结果

一眼就可以看出， <logic:redirect> 标签的可读性不强，它的 name 属性表示的是一个 Map 类型的变量。如果还有 property 属性，则 name 属性指的是一个标准 JavaBean 。 property 属性指的是 JavaBean 中的一个 Map 类型的变量属性，通过 Map 的“名值对”来为转发页面传递参数。如果转发参数是来自于一个 Map 或 JavaBean 中的 Map 类型变量属性，那还好，因为可以在 Java 类中处理。可是如果纯粹是从页面上取得某些值作为转发参数，那就困难了，必须像本示例所给出的那样，自行定义一个 Map 实例。这种情况下，页面就会看到 Java 语言的片段，既麻烦又不符合标准。

而使用 <c:redirect> 和 <c:param> 标签协作，由于包含在 <c:redirect> 标签体内的 <c:param> 标签可以有多个，因此显式地提供 <c:param> 标签就完成了给出转发参数的工作，即使用到 JavaBean ，也可以使用 EL 表达式来实现。

综上所述，利用 <c:redirect> 和 <c:param> 标签来代替 <logic:redirect> 标签是有必要的。

9.8.4 总结

Struts 框架和 JSTL 并不是互相冲突的两种技术，虽然 Struts 框架提供了功能不错的标签库，但是使用 JSTL 可以简化 Struts 框架标签库复杂的地方，这对于服务器端表示层框架的 Struts 来说帮助很大。 Struts 的 HTML 标签库无法使用 JSTL 来替换，但是，使用 EL 表达式作为一些 value 属性，来做赋值的工作仍然不失为一种好的选择。因此，在 JSTL 已经比较成熟的今天，使用 Struts 框架和 JSTL 整合来作 JSP 页面将使程序设计更为轻松。

9.9 完整示 例

在这一小节中，将修改在第三章中曾经给出的 Struts 框架示例，以 Struts 框架和 JSTL 的协同工作来实现。

对于第三章的示例，要将 JSTL 整合进去，需要做以下几步工作。

（ 1 ）下载 JSTL 并配置。

（ 1 ）修改原先的 web.xml 使其作为 Servlet2.4 的实现。

（ 3 ）修改 JSP 显示页面，整合 JSTL 和 Struts 标签库一起工作。

9.9.1 下载 JSTL 并配置

可以从 http://.java.sun.com/products/.jsp/.jstl 网址中下载 JSTL1.1 的最新版本。要使用这些标签库需要做 3 个步骤的工作。

（1）将下载的 jstl.jar 放置到 Web 应用的 WEB-INF 的 lib 目录下。

（2）将下载的 TLD 文件放置到 Web 应用的 WEB-INF 目录下。

（3）在需要使用的 JSP 页面中声明该标签库。

9.9.2 修改 web.xml 使其作为 Servlet2.4 的实现

在第三章的示例中所给出的 web.xml 是 Servlet2.3 规范的，因此无法很好的支持 JSTL1.1 ，要修改为符合 Servlet2.4 规范的代码。使 web.xml 成为 Servlet2.4 规范是十分容易的，需要修改的是其头部 DTD 声明。

在 Servlet2.3 之前，校验和规范 web.xml 都是使用 DTD ，因此其头部声明如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
…
</web-app>
```

而到了 Servlet2.4 规范，首此使用了 xmlns 来声明 web.xml ，因此其头部声明为：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
        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
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd ">
…
</web-app>
```

所以，为了支持 Servlet2.4 规范，应该将第三章示例的 web.xml 改成如例 9.6 的样子。

例 9.6 ：修改后的 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
        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
```

```
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<filter>
     <filter-name>Set Character Encoding</filter-name>
     <filter-class>struts.sample.cap1.sample3.util.SetCharacterEncodingFilter</filter-class>
     <init-param>
               <param-name>encoding</param-name>
               <param-value>UTF-8</param-value>
     </init-param>
</filter>
<filter-mapping>
     <filter-name>Set Character Encoding</filter-name>
     <url-pattern>*.do</url-pattern>
</filter-mapping>
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
   <init-param>
     <param-name>config</param-name>
     <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   <init-param>
     <param-name>debug</param-name>
     <param-value>2</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

```
<welcome-file-list>
   <welcome-file>setSolution.jsp</welcome-file>
</welcome-file-list>
<taglib>
   <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
<taglib>
   <taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
```

</taglib>

</web-app>

修改的地方不多，仅仅是头部的声明。其他地方完全不必改动，这样的 web.xml 已经支持了 Servlet2.4 规范了。

9.9.3 JSP 部分： <logic:notEmpty> 和 <c:if> 标签

对于 JSP 显示页面的修改是整合的核心部分，在第三章示例的 showAttackSolution.jsp 中出现了这样的语句：

```
<logic:notEmpty name="allAttackSolution">
…
</logic:notEmpty>
```

将类型为 ArrayList 的变量“ allAttackSolution ”从作用范围中取出，利用 <logic:notEmpty> 标签判断该 ArrayList 是否为空。

根据之前讨论的“所有判断标签被 EL 表达式和 <c:if> 标签替换”，可以利用 <c:if> 标签和 EL 表达式来修改该段 JSP 代码。

修改后的结果如下：

```
<c:if test="${(requestScope.allAttackSolution != null)
&& fn:length(requestScope.allAttackSolution) != 0}">
…
</c:if>
```

<logic:notEmpty> 标签其本身具有多种功能：

q 一是判断是否为 null 。

q 二是当它为 String 类型的变量时，判断字符串长度是否为 0 。

q 三是当它为集合类型的变量时，利用集合类的 isEmpty 方法可以判断是否是一个空的集合。

本示例既然要在替换后与替换前的工作一致，就应该对集合做两个判断：

q 一是该集合不为 null 。

q 二是该集合中的对象数量不为 0 。

“ !=null ”的 EL 表达式实现了集合实例不为 null 的判断； fn:length() 函数实现了集合内对象数量不为 0 的判断，两个判断用“ && ”连接起来就实现了 <logic:notEmpty> 标签对于集合判断的工作。

在这里应该利用“ <logic:notEmpty> 标签”，还是利用“ EL 表达式和 <c:if> 标签”呢？

<logic:notEmpty> 标签相对来说可读性更强些， EL 表达式作为判断条件则可读性稍差些。然而，这些仅是就本示例的改动而言的，其他情况下，利用 EL 表达式和 <c:if> 标签还是有其优势的。

9.9.4 JSP 部分： <logic:iterate> 和 <c:forEach> 标签

在第三章示例的 showAttackSolution.jsp 中出现了这样的使用：

```
<logic:iterate name="allAttackSolution"
id="attackSolution"
type="struts.sample.cap1.sample3.entity.AttackSolution">
        <tr>
                <td style="word-break: break-all;">
                        <bean:write property="attack_event_code" name="attackSolution"
/>
                </td>
                <td style="word-break: break-all;">
                        <bean:write property="attack_mean" name="attackSolution" />
                </td>
                <td style="word-break: break-all;">
                        <bean:write property="attack_action" name="attackSolution" />
                </td>
                <td style="word-break: break-all;">
                        <input type="button"
                 onclick="del('<%=attackSolution.getAttack_event_code()%>');"
value="<bean:message key="message.delete"/>">
                </td>
        </tr>
</logic:iterate>
```

由于在 Action 中将显示的内容作为 ArrayList 类型的实例保存在 request 中，因此这段 JSP 页面标签的工作是：

（ 1 ）利用 <logic:iterate> 标签对保存在 ArrayList 实例中的所有对象进行循环取得。

（ 2 ） ArrayList 类型实例中的对象为 struts.sample.cap1.sample3.entity.AttackSolution 类型，

AttackSolution Java 类中的变量属性都有 get 、 set 方法，因此可以被认为是一个标准的 JavaBean 。利用 <bean:write> 标签将 AttackSolution 实例的变量属性读取出来，并显示。

根据之前讨论的“ <logic:iterate> 标签被 <c:forEach> 标签和 EL 表达式替换”，可以利用 <c:forEach> 标签和 EL 表达式来修改该段 JSP 代码。修改的方式有两种:

q 完全使用 <c:forEach> 标签和 EL 表达式来替换全部。

q 仅使用 EL 表达式来替换 <bean:write> 标签。

1. <c:f orEach> 标签和 EL 表达式

<c:forEach> 标签和 EL 表达式:

```
<c:forEach items="${requestScope.allAttackSolution}"
var="attackSolution">
        <tr>
                <td style="word-break: break-all;" >
                        ${attackSolution.attack_event_code}
                </td>
                <td style="word-break: break-all;" >
                        ${attackSolution.attack_mean}
                </td>
                <td style="word-break: break-all;" >
                        ${attackSolution.attack_action}
                </td>
                <td style="word-break: break-all;" >
<input type="button"
onclick="del('${attackSolution.attack_event_code}');"
value="<bean:message key="message.delete"/>">
</td>
        </tr>
</c:forEach>
```

这种修改方式将屏弃 Struts 框架的 <logic:iterate> 标签，而以 <c:forEach> 标签来作为循环迭代的工作。它的最大优点是无需关注集合中的对象类型，只要保证该对象是一个标准的 JavaBean 就可以了。

2. 使用 EL 表达式来 替换 <bean :write> 标签

```
<logic:iterate name="allAttackSolution"
id="attackSolution"
type="struts.sample.capl.sample3.entity.AttackSolution">
        <tr>
                <td style="word-break: break-all;" >
                    ${attackSolution.attack_event_code}
                </td>
                <td style="word-break: break-all;" >
                    ${attackSolution.attack_mean}
                </td>
                <td style="word-break: break-all;" >
                    ${attackSolution.attack_action}
                </td>
                <td style="word-break: break-all;" >
<input type="button"
onclick="del('${attackSolution.attack_event_code}');"
value="<bean:message key="message.delete"/>">
</td>
        </tr>
</logic:iterate>
```

这种方式对原来的代码没有做多大的改动，依然会使用 <logic:iterate> 标签来作为循环标签。不过对于原来使用 <bean:write> 标签做显示功能的地方，摒弃了 <bean:write> 标签而直接使用 EL 表达式。灵活的 EL 表达式对页面显示逻辑有很大帮助，这种方式比较适合熟悉 <logic:iterate> 标签的程序设计者。

9.9.5 完整的 JSP

下面看一个完整的修改后 JSP 页面的代码，注释掉的是被替换之前的代码，读者可以比较一下两种实现方法。请见例 9.7 。

例 9.7 ： 修改后 showAttackSolution.jsp 。

```
<%@ page contentType="text/html; charset=utf-8"%>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
```

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>
<html>
 <head>
    <!-- 略过 JavaScript 部分 -->
…
 </head>
 <body>
   <em><bean:message key="message.attacksolutionDB"/></em><p>
   <table>
        <html:errors/>
    </table>
<bean:message key="message.attackcode"/>:
<input name="attack_event_codeC" value="" type="text"> 
<bean:message key="message.attackdesc"/>:
<TEXTAREA style="height:100" name=attack_meanC></TEXTAREA> 
<bean:message key="message.attacksolution"/>:
<TEXTAREA style="height:100" name=attack_actionC></TEXTAREA> 
    <p/>
    <html:form action="AddAttackSolutionAction.do">
        <html:hidden property="attack_event_code"/>
        <html:hidden property="attack_mean"/>
        <html:hidden property="attack_action"/>
        <input type="button" onclick="add();" value="<bean:message key="message.add"/>">
        <input type="button"
onclick="search();"
value="<bean:message key="message.search"/>">
    </html:form>
```

```
    <table border=1 cellspacing=1 cellpadding=2>
        <tr>
            <td style="background-color: #808080;font-size: 12pt;color: #ffffff;font-weight:
 bold;line-height: 15pt;border: 1px solid #808080;">
                <bean:message key="message.attackcode"/>
            </td>
            <td style="background-color: #808080;font-size: 12pt;color: #ffffff;font-weight:
 bold;line-height: 15pt;border: 1px solid #808080;">
                <bean:message key="message.attackdesc"/>
            </td>
            <td style="background-color: #808080;font-size: 12pt;color: #ffffff;font-weight:
 bold;line-height: 15pt;border: 1px solid #808080;">
                <bean:message key="message.attacksolution"/>
            </td>
            <td style="background-color: #808080;font-size: 12pt;color: #ffffff;font-weight:
 bold;line-height: 15pt;border: 1px solid #808080;">
                <bean:message key="message.delete"/>
            </td>
        </tr>
        <!-- 没有替换前的代码 -->
        <!--
            <logic:notEmpty name="allAttackSolution">
                <logic:iterate name="allAttackSolution"
id="attackSolution"
type="struts.sample.capl.sample3.entity.AttackSolution">
                    <tr>
                        <td style="word-break: break-all;" >
                            <bean:write property="attack_event_code"
name="attackSolution"/>
                        </td>
```

```
                        <td style="word-break: break-all;" >
                            <bean:write property="attack_mean" name="attackSolution"/>
                        </td>
                        <td style="word-break: break-all;" >
                            <bean:write property="attack_action" name="attackSolution"/>
                        </td>
                        <td style="word-break: break-all;" >
                            <input type="button"
onclick="del('
<bean:write
property="attack_event_code"
name="attackSolution"/>');"
value="<bean:message key="message.delete"/>">
                        </td>
                    </tr>
                </logic:iterate>
            </logic:notEmpty>
        -->

        <!-- 仅替换 <bean:write> 标签的代码 -->
        <!--
            <logic:notEmpty name="allAttackSolution">
                <logic:iterate name="allAttackSolution"
id="attackSolution"
type="struts.sample.capl.sample3.entity.AttackSolution">
                    <tr>
                        <td style="word-break: break-all;" >
                            ${attackSolution.attack_event_code}
                        </td>
                        <td style="word-break: break-all;" >
```

```
                                ${attackSolution.attack_mean}
                            </td>
                            <td style="word-break: break-all;" >
                                ${attackSolution.attack_action}
                            </td>
                            <td style="word-break: break-all;" >
                                <input type="button"
onclick="del('${attackSolution.attack_event_code}');"
value="<bean:message key="message.delete"/>">
                            </td>
                        </tr>
                    </logic:iterate>
                </logic:notEmpty>
            -->

            <!-- 替换后的实现代码 -->
            <c:if test="${(requestScope.allAttackSolution != null)
&& fn:length(requestScope.allAttackSolution) != 0}">
                <c:forEach items="${requestScope.allAttackSolution}" var="attackSolution">
                    <tr>
                        <td style="word-break: break-all;" >
                            ${attackSolution.attack_event_code}
                        </td>
                        <td style="word-break: break-all;" >
                            ${attackSolution.attack_mean}
                        </td>
                        <td style="word-break: break-all;" >
                            ${attackSolution.attack_action}
                        </td>
                        <td style="word-break: break-all;" >
```

```
                        <input type="button"
                               onclick="del('${attackSolution.attack_event_code}');"
value="<bean:message key="message.delete"/>">
                    </td>
                </tr>
            </c:forEach>
        </c:if>
    </table>
 </body>
</html>
```

可以看到，在这个被修改的 JSP 页面代码中，利用了 Struts 框架提供的标签来实现提交部分的工作以及国际化资源配置文件读取显示的工作，也利用 JSTL 的标签库和 EL 表达式来实现页面逻辑部分的工作。

在 JSP 页面使用 JSTL 是一种规范，也是一件令人兴奋的事情，因为它使 JSP 部分的程序设计变得更加有效合理。