

第 17 章 标准标签库JSTL

第 16 章介绍了自定义标签的开发，本章所介绍的内容和前一章有关联。标准标签库 JSTL 是由自定义标签产生的，它抽取出了最为常用的标签操作。可以说标准标签库实现了对通用操作的封装。有效的利用标准标签库以及自定义标签可使 JSP 页面风格统一，给 Web 开发人员带来开发和维护上的方便。

本章要点包括以下内容：

- ☐ JSTL 简介
- ☐ JSTL 标签库的安装
- ☐ 核心标签库的使用方法
- ☐ 国际化标签库的使用方法
- ☐ XML 标签库的使用方法
- ☐ 函数标签库的使用方法
- ☐ SQL 标签库的使用方法

17.1 JSTL简介

标准标签库 JSTL 的全名为 Java Server Pages Standard Tag Library。它是由 JCP（Java Community Process）所指定的标准规范，它主要是给 Java Web 开发者提供了一个标准的通用标签库。通过 JSTL，可以部分地取代传统 JSP 程序中嵌入 Java 代码的做法，可以使得 JSP 页面程序的风格趋于统一，并且容易维护。

从 JSP1.1 规范开始就开发支持在 JSP 文件中使用自定义标签了，就是因为自定义标签的广泛使用，从而使得大量的同功能标签不断地重复定义，为了减少对解决类似通用问题的独立标记库的需求，在 Java Community Process（JSR 52）的赞助下创建了 JSTL（JavaServer Pages Standard Tag Library，JSTL）标准标记库，为解决这些通用功能提供一个单一的标准解决方案。

JSTL 是一个开放源代码的 JSP 标签库，并且还在不断地完善过程中。JSTL 特别为条件处理、迭代、国际化、数据库访问和可扩展标记语言（XML）处理提供支持。JSTL 还引入了 expression language（EL，表达式语言），极大地简化了对 JSP 中应用数据的访问和操作。

JSTL 至今主要为开发者提供了如下五大类的标签库：

（1）核心标签库：为日常任务提供通用支持，如显示和设置变量、重复使用一组项目、测试条件以及其他操作（如导入和重定向 Web 页面等）。

（2）国际化（I18N）标签库：支持多国语种的应用程序。

（3）SQL 标签库：对访问和修改数据库提供标准化支持。

（4）XML 标签库：对 XML 文件处理和操作的支持，包括 XML 节点的解析、迭代、基于 XML 数据的条件评估以及可扩展样式语言转换（Extensible Style Language Transformations，XSLT）的执行。

（5）函数标签库：通过在 EL 表达式中调用函数标签库中的函数来实现特定的操作。例如 `${fn:contains(string, substring)}`，判断 string 字符串中是否包含 substring 字符串。

表 17.1 展示了各标签库的使用方法。

表 17.1 JSTL标准标签库

标签库	URI	前缀	使用模式
核心标签库	http://java.sun.com/jstl/core	c	<c:tagname...>
国际化 (I18N) 标签库	http://java.sun.com/jstl/fmt	fmt	<fmt:tagname...>
SQL标签库	http://java.sun.com/jstl/sql	sql	<sql:tagname...>
XML标签库	http://java.sun.com/jstl/xml	x	<x:tagname...>
函数标签库	http://java.sun.com/jstl/functions	fn	fn:functionName()

17.1.1 使用JSTL的优点

JSTL（标准标签库）的使用对 JSP/servlet 开发者来说是一个重大的进展，因为它存在很多的优点，并且很有可能在不久的将来成为实现动态、基于 Java 站点的一个主要方法。

概括起来，JSTL 主要的优点如下：

（1）在所有的应用服务器之间提供了一致的接口程序，这样可以最大程度地提供 Web 应用程序在各种应用服务器之间的可移植性。

（2）简化了 JSP 的 Web 应用系统的开发，并且使得 JSP 页面的编程风格统一、易于维护。

（3）大量使用 JSTL 提供的逻辑操作（例如迭代、判断，甚至数据库访问），可以大大减少 JSP 中脚本代码（Scriptlets）的数量，甚至可以没有任何脚本代码就可以实现其大部分动态效果。

（4）运行 JSP 设计工具与 Web 应用程序开发的进一步集成，虽然现在支持 JSTL 开发的 IDE 开发工具还没有出现（或者正是由于这个原因，使得 JSTL 还不这么流行），但是可以相信不久就会有支持 JSTL 的开发工具出现。

JSTL 已经封装了 JSP 中很多常用的功能，比如，可以使用 JSTL 中的标签来进行迭代输出某个 List 或者 Set 集合类。

注意：由于从 JSP1.1 规范才开始支持 JSP 中的自定义标签，所以 JSTL 需要运行在支持 JSP1.2 和 Servlet2.3 规范的容器上，比如 Tomcat4.x 和 Tomcat5.x 服务器。

17.2 JSTL的安装

JSTL 标准标签库的安装非常简单，下面具体介绍 JSTL 的下载和安装配置过程。

17.2.1 JSTL标签库的下载

在 jakarta.apache.org 网站上下载 JSTL 安装包。单击主页面中的“Taglibs”链接，然后选择“Standard 1.1 Taglib”下载。下载完的文件全名为：jakarta-taglibs-standard-1.1.2.tar.gz。

17.2.2 安装配置JSTL标签库

具体的安装配置过程如下：

（1）把下载的文件 jakarta-taglibs-standard-1.1.2.tar.gz 解压，得到如图 17.1 所示的文件夹和文件。这里有一个名为 standard-examples.war 的文件，它是示范 JSTL 用法的例子程序，读者可以把它部署到相应服务器上，以便能够快速学习 JSTL。部署非常简单，只要把这个 war 文件复制到 Tomcat 的 webapps

目录下，然后直接调用即可。

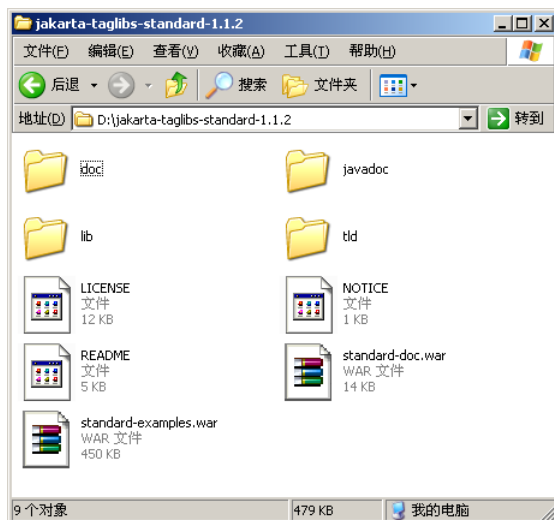


图 17.1 JSTL 目录结构

(2) 在 Web 模块下的 WEB-INF 目录下创建一个 tlds 文件夹。

(3) 把 jakarta-taglibs-standard-1.1.2\tld 目录下的所有 tld 文件复制到 Web 模块下的 WEB-INF\tlds 目录下。

(4) 将 jakarta-taglibs-standard-1.1.2\lib 目录下的所有 jar 文件复制到 Web 模块下的 WEB-INF\lib 目录下。

(5) 接下来需要在 WEB-INF/web.xml 文件中进行相应的部署描述，在文件的</web-app>前面部分添加如下描述代码：

```
<taglib>
<taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
<taglib-location>/WEB-INF/tlds/fmt.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
<taglib-location>/WEB-INF/tlds/c.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
<taglib-location>/WEB-INF/tlds/sql.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>http://java.sun.com/jstl/x</taglib-uri>
<taglib-location>/WEB-INF/tlds/x.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>http://java.sun.com/jsp/jstl/functions</taglib-uri>
<taglib-location>/WEB-INF/tlds/fn.tld</taglib-location>
</taglib>
```

代码说明：在 web.xml 文件描述需要使用到的标签库，这里定义了之上介绍到的五个标准标签库。<taglib-uri>元素定义该标签的唯一表示，在 JSP 文件中就是通过该值来引用相应标签的。<taglib-location>

定义标签库所在的位置，本书已将所有标准标签库放置在 WEB-INF\tlds 目录下。

至此，就已经成功地把 JSTL 标准标签支持安装和部署了。

17.2.3 实例示范

通过 Eclipse+Lomboz 创建一个名为 MyJSTL 项目，Web 模块名为 JSTL，然后按照上面的配置方法来配置 JSTL 标签库。下面建立一个简单的例子来查看是否可以正确使用 JSTL 标准标签。在 Web 模块中创建一个 helloJSTL.jsp 文件。该文件的详细源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>JSTL Example--hello JSTL</title>
</head>
<body bgcolor="#FFFFFF">
<h4><c:out value="第一个 JSTL 的 JSP 程序，使用 JSTL 的标签迭代输出信息"/></h4>
<%
String items[] = new String[3];
items[0] = "核心标签库";
items[1] = "国际化标签库";
items[2] = "SQL 标签库";
request.setAttribute("items",items);           //把数组变量保存在 request 中
%>
<table border="1">
  <c:forEach var="item" items="${items}">
    <tr>
      <td><c:out value="${item}"/></td>
    </tr>
  </c:forEach>
</table>
</body>
</html>
```

程序说明：

(1) 当在一个页面中使用 JSTL 标签库时，需要在<%@ taglib %>指令中包含这个标签库，同时要指定标签的前缀。例如，此处的设置为<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>。

(1) 在 page 指令中指定 isELIgnored="false"，使得该页面支持 EL 表达式的输出。

(2) 该程序使用了核心标签库中的 out 和 forEach 两个标签，第一标签用来输出数据，第二个标签用来进行迭代处理，这将在后面重点讲解。

(3) 运行这个页面，如果出现如图 17.2 所示，则表示 JSTL 标签库的安装和设置正确，可以正常执行。

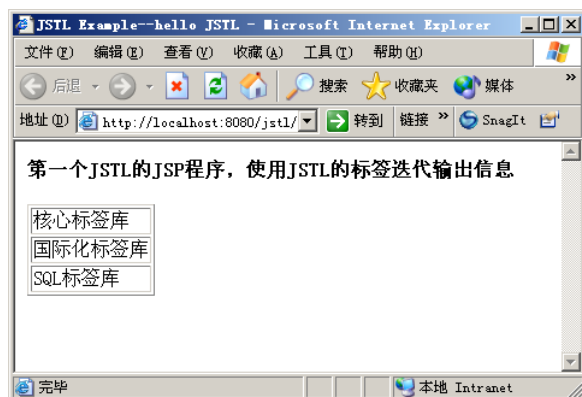


图 17.2 helloJSTL.jsp 页面效果

17.3 核心标签库

这一小节将重点向读者介绍 JSTL 中一组专门负责表达式操作以及流程控制的标签，这包括：out、set、remove、catch、if、choose、when 和 URL 等。在下面的讲解过程可能会经常使用到 EL 表达式，例如之上的例子就是使用到了 EL 表达式。关于 EL 表达式的介绍将在下面章节重点介绍。

17.3.1 表达式操作标签

表达式操作标签包括 out、set、remove 和 catch。

17.3.1.1 out 标签

核心标签库中最为基本的标签就是 `<c:out>`。它可以在页面中显示一个字符串或者一个 EL 表达式的值。它的功能与 JSP 传统的 `<%=表达式%>` 相类似。

out 标签的使用格式如下：

```
<c:out value="object" [escapeXml="true|false"] />
```

这个标签还可以有一个体，如下：

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

这表示当 value 属性指定的 object 值为 null 时，就会显示体中指定的值（也就是 default value），体中也可以是 JSP 代码。

`<c:out>` 中的各个属性：

- ❑ value：可以指定一个字符串作为输出内容，也可以为 EL 表达式，例如 `${3+5}`。
- ❑ escapeXml：类型为 boolean，确定 `<`、`>`、`&`、`'`、`"` 等这些字符在结构的字符串中是否被转换成字符串实体代码，默认的是 true。
- ❑ default value：可以是字符串，还是一个表达式（EL 表达式或者 `<%=表达式%>` 的 JSP 表达式）。

如果属性指定的表达式或者对象值为 null 时，那么将输出这个 default value 部分的值。

下面通过一个 `<c:out>` 的简单实例，来示范以上各属性设置的效果，创建的 c_out.jsp 页面代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<h4><c:out value="核心标签库中的 out 标签的简单实例"/></h4>
<c:out value="${3+7}"/><br>
```

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

程序说明：

(1) `<c:out value="核心标签库中的 out 标签的简单实例"/>`直接输出 value 指定的字符串值。

(2) `<c:out value="{3+7}"/>`中使用了 EL 表达式，输出 EL 表达式的计算结果 10。

(3) `<c:out value="<p>有特殊字符</p>" />`中因为默认 escapeXml 值为 true，即把 `<p>` 转换成字符实体代码，这段代码输出的 HTML 为：

```
&lt;p&gt;有特殊字符&lt;/p&gt;
```

(4) `<c:out value="<p>有特殊字符</p>" escapeXml="false" />`中设置了 escapeXml="false"，所以输出的 HTML 为：

```
<p>有特殊字符</p>
```

该 JSP 文件运行的效果如图 17.3 所示。

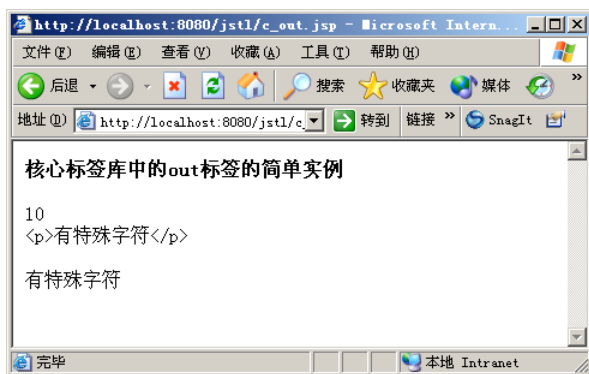


图 17.3 c_out.jsp 页面效果

17.3.1.2 set 标签

set 标签是使用来在某个范围（request、session 或者 application）内设值，或者设置某个对象的属性值。

set 标签的使用格式如下：

(1) 使用 value 属性在一个特定的范围内指定某个变量的值，其使用格式如下：

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

(2) 当使用 value 属性在一个特定范围内指定某个变量的值时，也可以包括一个体，它的作用和 out 标签体的作用一样，即 value 指定的值为 null 时，默认使用体中指定的值，使用格式如下：

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

(3) 设置某一个特定对象属性的使用格式如下：

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

(4) 在设置某个特定对象的一个属性时，也可以有一个体，也指定默认值，使用格式如下：

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

该标签中的各属性说明如下：

- ❑ value: 该属性指定变量或者对象中某个属性的值，并且可以是一个表达式。
- ❑ var: 变量名称，value 属性指定的值就保存在这个变量中。

- ❑ scope: 设定变量的有效范围, 如果不设置, 则默认为 page。
- ❑ target: 设置属性的一个对象, 它必须是 JavaBean 或者 java.util.Map 对象。
- ❑ property: 设置对象中的一个属性。

接下来给出一个<c:set>标签的应用实例, 创建的 c_set.jsp 页面代码如下:

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>JSTL: -- Set 标签实例</title>
</head>
<body>
<h3>set 标签实例</h3>
<hr>
<!-- 通过 set 标签赋值 -->
<c:set var="num1" scope="request" value="{3+5}"/>
<c:set var="num2" scope="session" >
${3+6}
</c:set>
<c:set var="num3" scope="session" >
10
</c:set>
<br>
<!-- 通过 out 标签输出变量值 -->
num1 变量值为: <c:out value="{num1}" /><br>
num2 变量值为: <c:out value="{num2}" /><br>
num3 变量值为: <c:out value="{num3}" />
</body>
</html>
```

程序说明: 在这个页面程序中使用三种赋值方法给三个变量 num1、num2 和 num3 赋值。然后使用 out 标签把变量值输出。运行效果如图 17.4 所示。

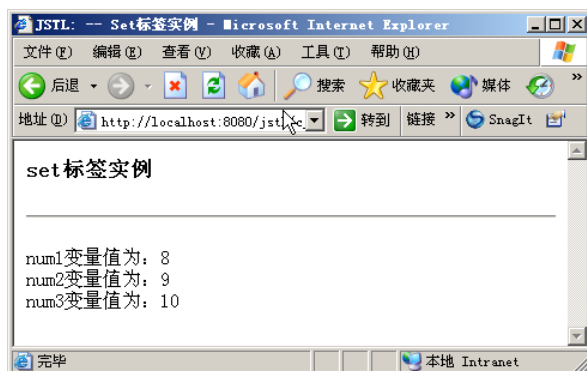


图 17.4 c_set.jsp 页面效果

17.3.1.3 remove 标签

remove 标签一般和 set 标签配套使用, 两者是相对应的, remove 标签用于删除某个变量或者属性。使用格式如下:

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

remove 标签中使用的各属性说明如下：

- scope: 设定这个需要删除的变量所在范围。
- var: 需要删除的变量或者对象属性的名称。

如果没有设置 scope 的属性,即采用默认值,那么就相当于调用 `PageContext.removeAttribute(varName)` 方法;如果指定了这个变量所在的范围,那么系统会调用 `PageContext.removeAttribute(varName,scope)` 方法。

接下来还是通过一个实例来了解 remove 标签的使用,创建的 `c_remove.jsp` 源代码如下:

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>JSTL: -- Set 标签实例</title>
</head>
<body>
<h3>set 标签实例</h3>
<hr>
<!-- 通过 set 标签赋值 -->
<c:set var="num" scope="page" value="${3+5}"/> <!-- 此定义的变量只能在本页范围内有效 -->
<c:set var="num" scope="request" >
  ${3+6}
</c:set>
<c:set var="num" scope="session" >
<c:out value="10"/>
</c:set>
<!-- 变量显示 -->
<table border="1" width="50%">
  <tr>
    <td>pageScope.num</td><td><c:out value="${pageScope.num}" default="NULL" /></td>
  </tr>
  <tr>
    <td>requestScope.num</td><td><c:out value="${requestScope.num}" default="NULL" /></td>
  </tr>
  <tr>
    <td>sessionScope.num</td><td><c:out value="${sessionScope.num}" default="NULL" /></td>
  </tr>
</table>
<!-- 使用 remove 标签删除 page 范围中的 num 之后, 变量值的显示 -->
<c:out value='<c:remove var="num" scope="page" />之后, 各变量的值: '/>
<c:remove var="num" scope="page" />
<table border="1" width="50%">
  <tr>
    <td>pageScope.num</td><td><c:out value="${pageScope.num}" default="NULL" /></td>
  </tr>
  <tr>
    <td>requestScope.num</td><td><c:out value="${requestScope.num}" default="NULL" /></td>
  </tr>
  <tr>
    <td>sessionScope.num</td><td><c:out value="${sessionScope.num}" default="NULL" /></td>
```



```

</tr>
</table>
<!-- 使用 remove 标签删除 num 之后(scope 没有指定, 即把所有范围内的变量删除, 变量值的显示 -->
<c:out value='<c:remove var="num" />之后, 各变量的值: '/>
<c:remove var="num" />
<table border="1" width="50%">
  <tr>
    <td>pageScope.num</td><td><c:out value="${pageScope.num}" default="NULL" /></td>
  </tr>
  <tr>
    <td>requestScope.num</td><td><c:out value="${requestScope.num}" default="NULL" /></td>
  </tr>
  <tr>
    <td>sessionScope.num</td><td><c:out value="${sessionScope.num}" default="NULL" /></td>
  </tr>
</table>
</body>
</html>

```

程序说明:

- (1) 此程序中, 首先在三种不同范围内指定了 num 变量的值。
- (2) <c:remove var="num" scope="page" />表示把在 page 范围内设定的 num 变量删除掉, 而其他范围内的 num 变量则可以正常显示原有值, 效果请看图 9.4 所示。
- (3) <c:remove var="num" />表示把所有范围内的 num 变量全部删除。

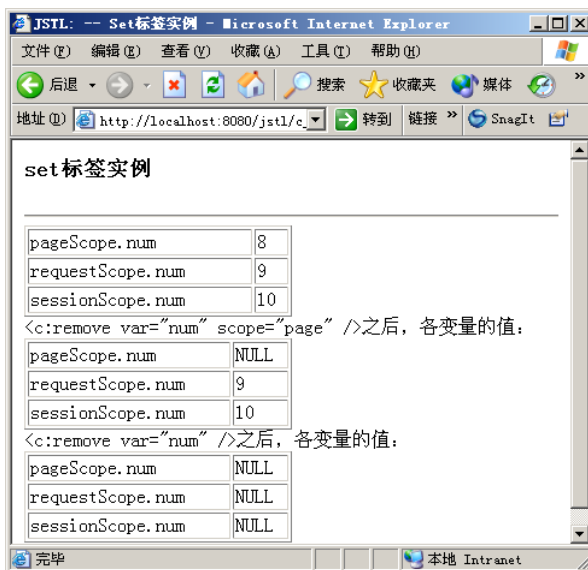


图 17.5 c_remove.jsp 页面效果

17.3.1.4 catch 标签

catch 标签的功能和 Java 程序中 try...catch{} 语句功能很类似, 它用于捕获嵌入到它中间语句抛出的异常。这个标签的使用格式如下:

```

<c:catch var="varName" >
  相关操作语句
</c:catch>

```

这个标签中的 `var` 属性用于保存异常错误的一个变量名。下面还是通过一个实例来具体了解 `catch` 标签的使用，创建的 `c_catch.jsp` 页面代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>JSTL: -- catch 标签实例</title>
</head>
<body>
<h4>catch 标签实例</h4>
<hr>
<c:catch var="errors">
<%
    String num = request.getParameter("num");
    int i = Integer.parseInt(num);
    out.println("the number is "+i);
%>
</c:catch>
<c:out value="${errors}"/>
</body>
</html>
```

在 IE 浏览器中输入地址：http://localhost:8080/jstltest/c_catch.jsp?num=3。所显示的页面如图 17.5 所示，这时 `<c:catch>` 体中的代码执行正确，没有抛出异常。

而当输入的地址为 http://localhost:8080/jstltest/c_catch.jsp?num=ok。这里给 `num` 值赋予了一个字符串 `ok`，而在以上程序中调用 `Integer.parseInt(num)` 把这个字符串硬转换成数值时，这时会抛出异常。这时显示的页面为图 17.6 所示。

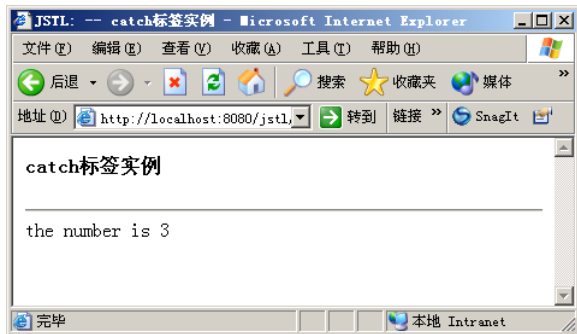


图 17.6 无异常的页面显示

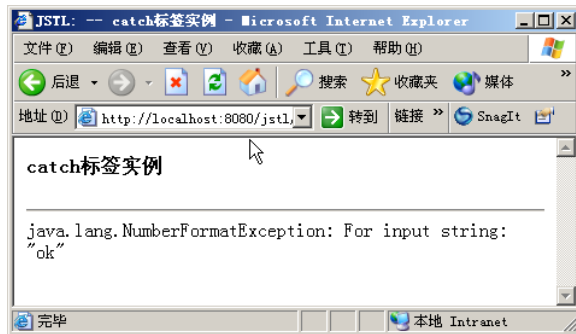


图 17.7 有异常抛出的页面显示

`catch` 标签会把出现的异常保存在指定的 `errors` 变量中。然后在页面中使用 EL 表达式把这个页面显示出来。

17.3.2 流程控制标签

下面将要讲的用于流程控制的标签，其中包括：`if`、`choose`、`when` 与 `otherwise` 等。接下来对这些标签逐一讲解。

17.3.2.1 if 标签

这个标签的作用和 Java 程序中的 if 语句作用相同，用于判断条件语句，主要使用的格式如下：

(1) 在<c:if>体中不包括体的使用格式如下：

```
<c:if test="checkCondition" var="varName" scope="page|request|session|application" />
```

(2) 当<c:if>中包含体时，使用的格式如下：

```
<c:if test="checkCondition" var="VarName" scope="page|request|session|application" >
body content
</c:if>
```

标签中所使用的属性说明如下：

- ☐ test：判断条件的表达式，返回类型为 true 或者 false。
- ☐ var：这个变量用于保存 test 条件表达式判断所返回的 true 或者 false 值。
- ☐ scope：指定 var 变量所在的有效范围。

如果这个标签包含体，那么在 test 判断条件返回 true 时，将执行体中的代码或者表达式。

创建一个 c_if.jsp 页面，其源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>JSTL: -- if 标签实例</title>
</head>
<body>
<h4>if 标签实例</h4>
<hr>
<c:set var="num" scope="page" value="admin" />
<c:if test="${num == 'admin'}" var="condition" scope="page">
    你好 admin!
</c:if>
<c:out value="这里判断结果：${condition}"/>
<br>
<c:if test="${num == 'guest'}" var="condition" scope="page">
    你好 admin!
</c:if>
<c:out value="这里判断结果：${condition}"/>
</body>
</html>
```

在 IE 浏览器中运行这个页面，其执行效果如图 17.8 所示。

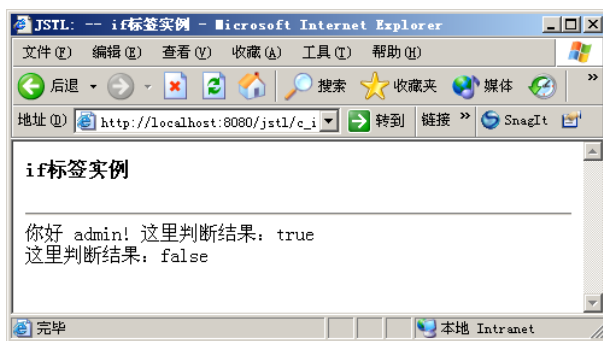


图 17.8 c_if.jsp 页面执行效果

17.3.2.2 choose、when 和 otherwise 一组标签

这些标签一般是组合起来一起使用的，就相当于 Java 程序中的 switch 条件语句。

在<c:choose>标签体中包括<c:when>和<c:otherwise>子标签。<c:when>子标签代表<c:choose>的一个条件分支。

下面通过<c:choose>的实例来了解这些标签的具体使用方法，创建一个 c_choose.jsp 文件代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <title>JSTL: -- choose 标签实例</title>
</head>
<body>
<h4>choose 标签实例</h4>
<hr>
<c:set var="num" scope="page" value="admin" />
<c:choose>
  <c:when test="${num == 'guest'}">
    <c:out value="guest" />
  </c:when>
  <c:when test="${num == 'admin'}">
    <c:out value="admin" />
  </c:when>
  <c:otherwise>
    <c:out value="other" />
  </c:otherwise>
</c:choose>
</body>
</html>
```

程序说明：<c:otherwise>子标签必须使用在<c:choose>标签的体内，而且必须是在最后的分支上。在本实例中，由于 num 值等于“admin”，所在页面中会输出字符串“admin”。运行效果如图 17.9 所示。



图 17.9 c_if.jsp 页面执行效果

17.3.2.3 forEach 迭代标签

该标签使用来对一个 Collection 集合中的一系列对象进行迭代输出，并且可以指定迭代次数。一般使用格式如下：

```
<c:forEach items="collection" var="varName" [varstatus="varStatusName"]
[begin="begin"] [end="end"] [step="step"]>
body content
</c:forEach>
```

这个标签中所使用的属性描述如下：

- ❑ var: 也就是保存在 Collection 集合类中的对象名称。
- ❑ items: 将要迭代的集合类名。
- ❑ varStatus: 存储迭代的状态信息，可以访问到迭代自身的信息。
- ❑ begin: 如果指定了 begin 值，就表示从 items[begin]开始迭代，如果没有指定 begin 值，则从集合的第一个值开始迭代。
- ❑ end: 表示迭代到集合的 end 位时结束，如果没有指定 end 值，则表示一直迭代中集合的最后一位。
- ❑ step: 指定迭代的步长。

下面还时通过一个具体的实例来了解这个标签的使用方法，创建一个 c_forEach.jsp 文件代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>JSTL: -- forEach 标签实例</title>
</head>
<body>
<h4><c:out value="forEach 实例"/></h4>
<hr>
<%
String items[] = new String[5];
items[0] = "核心标签库";
items[1] = "国际化标签库";
items[2] = "SQL 标签库";
items[3] = "XML 标签库";
items[4] = "函数标签库";
request.setAttribute("items",items);
%>
<B><c:out value="不指定 begin 和 end 的迭代: " /></B><br>
<c:forEach var="item" items="${items}">
    &nbsp;<c:out value="${item}"/><br>
</c:forEach>
<B><c:out value="指定 begin 和 end 的迭代: " /></B><br>
<c:forEach var="item" items="${items}" begin="1" end="3" step="1">
    &nbsp;<c:out value="${item}" /><br>
</c:forEach>
<B><c:out value="输出整个迭代的信息: " /></B><br>
<c:forEach var="item" items="${items}" begin="3" end="4" step="1" varStatus="s">
    &nbsp;<c:out value="${item}" />的四种属性: <br>
    &nbsp;&nbsp;&nbsp;所在位置，即索引: <c:out value="${s.index}" /><br>
    &nbsp;&nbsp;&nbsp;总共已迭代的次数: <c:out value="${s.count}" /><br>
    &nbsp;&nbsp;&nbsp;是否为第一个位置: <c:out value="${s.first}" /><br>
    &nbsp;&nbsp;&nbsp;是否为最后一个位置: <c:out value="${s.last}" /><br>
```

```
</c:forEach>
</body>
</html>
```

程序说明：

(1) 如以上程序，如果对 `forEach` 标签不指定 `begin` 和 `end` 属性，则表示从集合的第一位一直跌到到集合的最后一位。

(2) 可以使用 `varStatus` 属性来存放循环到当前元素的相关信息。

(3) 在 IE 浏览器上运行该页面，效果如图 17.10 所示。

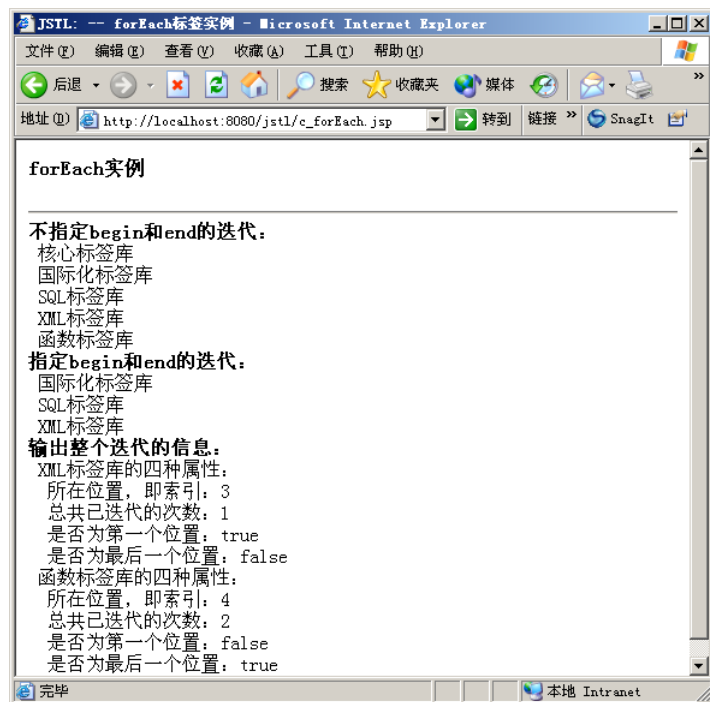


图 17.10 c_forEach.jsp 页面效果

17.3.2.4 forTokens 标签

这个标签的作用和 Java 中的 `StringTokenizer` 类的作用非常相似，它通过 `items` 属性来指定一个特定的字符串，然后通过 `delims` 属性指定一种分隔符（可以同时指定多个），通过指定的分隔符把 `items` 属性指定的字符串进行分组。和 `forEach` 标签一样，`forTokens` 标签也可以指定 `begin` 和 `end` 以及 `step` 属性值。

这个标签的使用格式如下：

```
<c:forTokens items="stringOfTokens" delims="delimiters" var="varName" [varStatus="varStatusName"]
  [begin="begin"] [end="end"] [step="step"]>
  body content
</c:forTokens>
```

标签中的各个属性描述如下：

- ❑ `var`：进行迭代的参数名称。
- ❑ `items`：指定的进行标签化的字符串。
- ❑ `varStatus`：每次迭代的状态信息。
- ❑ `delims`：使用这个属性指定的分隔符来分割 `items` 指定的字符串。
- ❑ `begin`：开始迭代的位置。

- 虽然 `forEach` 也是可以迭代 `TokenString` 的，例如下面一段代码：

但是 `forTokens` 标签在处理 `TokenString` 时，功能更加的强大。

看下面的例子，创建一个 `c_forTokens.jsp`，详细代码如下：

程序说明:

- (1) 通过 `delims` 属性指定的分隔符把 `items` 设定的字符串分割，并存储在一个数组当中。
- (2) 可以指定多个字符来作为分隔符，例如 `delims="|,"`，即 `|` 和 `,` 同时作为分隔符进行使用。
- (3) 在 IE 浏览器中运行这个页面，看到的页面效果如图 17.11 所示。



图 17.11 c_forTokens.jsp 页面效果

17.3.3 URL 相关的标签

与 URL 相关的标签主要是用来将其他文件包含进来，或者提供页面之间的重定位以及 URL 地址的生成、参数的输出等等。一般包括如下几个标签

- ❑ `<c:import>` 标签：与传统 JSP 页面中的 `<jsp:include>` 标记相类似。
- ❑ `<c:redirect>` 标签：进行页面的重定位。
- ❑ `<c:url>`：主要是用来重写 URL 地址。
- ❑ `<c:param>`：一般 `param` 标签会和别的标签一起配套使用，用于参数值的传递。

17.3.3.1 `<c:import>` 标签

该标签用于把其他静态文件包含到该文件当中来。它和传统的 JSP 标记 `<jsp:include>` 相类似，但是有所不同：`<jsp:include>` 标签只能使用来包括该应用中的其他文件，而 `<c:import>` 则还可以包含外部站点中的静态文件，所以它的功能更加的强大。这个标签的使用格式如下：

```
<c:import url="url" [context="context"] [var="varName"] [scope="page|request|session|application"]  
    [varReader="varReader"] [charEncoding="charEncoding"]>  
body content  
</c:import>
```

标签中的属性描述如下：

- ❑ `url`：待引用静态文件的 URL 地址。
- ❑ `context`：当时用相对路径访问一个外部静态文件时，这里的 `context` 指定这个外部文件的名称。
- ❑ `var`：当使用字符串输出时，把输出的内容存储在这个 `var` 指定的变量中。
- ❑ `scope`：指定 `var` 参数变量的有效范围。
- ❑ `charEncoding`：引入文件所采用的字符编码。
- ❑ `varReader`：这个属性指定的参数变量类型是 `Reader`，可以用于读取文件内容。

创建一个 `c_import.jsp` 文件，它的源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>  
<%@ page isELIgnored="false" %>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<html>  
<head>  
    <title>JSTL: -- import 标签实例</title>  
</head>  
<body>  
<h4><c:out value="import 实例"/></h4>  
<hr>  
<B><c:out value="绝对路径引用的实例" /></B>  
<blockquote>  
<ex:escapeHtml>  
    <c:import url="http://jakarta.apache.org"/>  
</ex:escapeHtml>  
</blockquote>  
<B><c:out value="相对路径引用的实例，引用本应用中的文件" /></B>  
<blockquote>  
<ex:escapeHtml>  
    <c:import url="helloJSTL.jsp"/>  
</ex:escapeHtml>
```



```

</blockquote>
<B><c:out value="使用字符串输出、绝对路径引用的实例" /></B>
<c:import var="myurl" url="http://jakarta.apache.org"/>
<blockquote>
<pre>
    <c:out value="{myurl}" />
</pre>
</blockquote>
<B><c:out value="使用字符串输出、相对路径引用的实例" /></B>
<c:import var="myurl" url="helloJSTL.jsp"/>
<blockquote>
<pre>
    <c:out value="{myurl}" />
</pre>
</blockquote>
</body>
</html>

```

程序说明：

(1) import 标签既可以相对路径或者绝对路径来应用本地站点中的任何静态文件，还使用绝对路径引用外部站点中的静态文件，甚至可以是 ftp 站点中的文件。例如：<c:import var="myurl" url="http://jakarta.apache.org"/>，就是引用了 http://jakarta.apache.org 站点的主页面。

(2) 可以把引用文件中的内容赋值给 var 属性指定的变量，然后进行读取，也可以赋值给 varReader 指定的 Reader 类型的变量，再进行相应的读取。

(3) 有时引入的文件在本地显示会出现乱码，这时可以通过 charEncoding 来指定这些内容所采用的字符集编码。

17.3.3.2 <c:redirect>和<c:param>标签

redirect 标签使用来进行页面之间的重定向，它和传统 JSP 程序中的<jsp:redirect>标记功能相类似。

param 标签是和 redirect 一起使用的，它用来进行参数值的传递。redirect 标签使用的格式如下：

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

在 redirect 标签体中指定 param 参数的使用格式如下：

```

<c:redirect url="url" [context="context"] >
<c:param name="paramName" value="value" />
</c:redirect>

```

下面创建 c_redirect1.jsp 和 c_redirect2.jsp 两个文件进行示范。

其中 c_redirect1.jsp 的源代码如下：

```

<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>JSTL: -- redirect 标签实例</title>
</head>
<body>
<!-- 这个页面的内容不会在显示出来，因为直接重定向到 c_redirect2.jsp 页面上 -->
<c:redirect url="c_redirect2.jsp">
<c:param name="userName" value="admin" />
</c:redirect>

```

```
</body>
</html>
```

c_redirect2.jsp 文件的源代码为:

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<h4>重定向的页面</h4>
<hr>
userName=<c:out value="${param.userName}" />
```

程序说明:

- (1) 一般 param 标签会和别的标签一起配套使用, 用于参数值的传递。
- (2) c_redirect2.jsp 程序中的 \${param.userName} 表达式获取到上一个页面传递过来的参数。
- (3) 打开浏览器, 输入地址: http://localhost:8080/jstl/c_redirect1.jsp, 运行效果如图 17.12 所示。

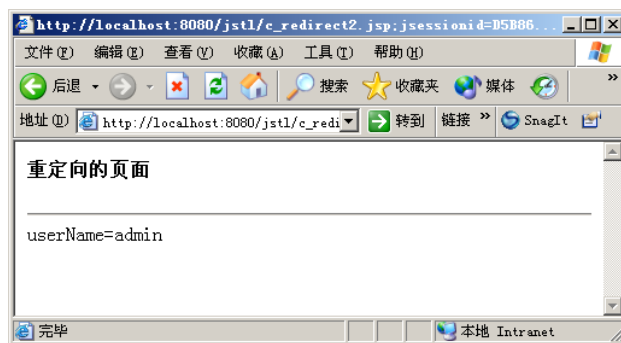


图 17.12 <c:redirect>标签使用页面效果

17.3.3.3 <c:url>标签

这个标签主要是用来重写 URL 地址。它的使用格式如下所示:

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

当进行参数传递时, 所使用的格式如下:

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

标签中使用的属性描述如下:

- ❑ value: 将要处理的 URL 地址。
- ❑ context: 当使用的是相对路径方法指定外部文件时, 这个属性所指定即为外在文件的名称。
- ❑ var: 给这个 URL 地址起一个标识。
- ❑ scope: 规定这个 var 属性所指定变量的有效范围。

创建一个 c_url.jsp 文件, 其源代码如下:

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url var="url" value="c_url2.jsp" scope="session">
  <c:param name="userName" value="admin"/>
  <c:param name="password" value="123456"/>
</c:url>
<c:out value="${url}" />
<a href='<c:out value="${url}" />'>Link to other page </a>
```

程序说明：url 标签可以指定一个变量（通过 var 属性指定）来把 URL 地址保存起来，并且可以指定这个变量的有效范围为 session，所以当需要使用到这个 URL 地址时，就可以使用 session.getAttribute(varName)方法来获取到这个地址。所以一般需要重复使用同一个 URL 地址时，才会使用<c:url>标签的功能。

17.4 国际化（I18N）标签库

在第十五章中，已经向读者介绍了国际化以及本地化的概念和内容，并针对问题详细讲解了如何解决 JSP 开发的国际化和本地化问题

这里的 JSTL 标准标签库也对国际化问题提供了很好的支持，其中用于支持国际化和本地化开发的标签包括：

- ❑ <fmt:setLocale>：用于设置当前本地化环境，其实为对应的 Java 类 javax.servlet.jsp.jstl.fmt.locale 配置参数值，这个指定的参数由 JSP 运行时维护，用于确定各个 JSTL 标记使用的本地化环境。
- ❑ <fmt:bundle>：指定消息资源使用的文件。
- ❑ <fmt:message>：显示消息资源文件中指定 key 的消息，支持带参数消息。
- ❑ <fmt:param>：给带参数的消息设置参数值。
- ❑ <fmt:setBundle>：设置消息资源文件。

下面对各个标签逐一介绍，然后通过一个综合实例来具体讲解这些标签的使用方法。

17.4.1 设置本地化环境标签<fmt:setLocale>

HTML 请求到达服务器时，浏览器提供的 HTTP 首部可以指出用户的首选本地化环境（可能是多个本地化环境的列表）。这个列表放在 Accept-Language HTTP 首部中。JSP 容器会访问这个首部信息，如果没有使用标签<fmt:setLocale>明确地指定应用本地化环境，JSTL 标记就会使用这个列表中的首选本地化环境。

遗憾的是，用这种方法来确定用户想用的本地化环境极不可靠，原因如下：

- ❑ 不同的浏览器会以不同的方式配置首选本地化环境列表。
- ❑ 大多数用户从来都不配置浏览器的首选本地化环境。
- ❑ 有些用户可能想使用另外某个本地化环境访问页面，而不是所配置的首选本地化环境。
- ❑ 有些用户可能会打开多个浏览器窗口（当前所有浏览器都支持这一点），并希望每个浏览器窗口使用不同的本地化环境。

由于存在这些不可控的因素，建议还是要求用户明确地指定想用什么本地化环境。

该<fmt:setLocale>标签就是专门用于设置当前本地化环境，它基本使用格式如下：

```
<fmt:setLocale value="...locale value..." [variant="... variant value..."]  
[scope="page|request|session|application"] />
```

<fmt:setLocale>标签的属性描述如下：

- ❑ value：这个属性是必须要设置的，它用来设置本地化环境名，例如 en_US 或者 zh_HK。
- ❑ variant：这个属性设置是可选的，而且很少使用，
- ❑ scope：即指定 value 设置的本地化环境名所在的有效范围，默认值为 page，即本页面。如果把 scope 设置为 session，则表示同一个用户发出的所有请求都有相同的本地化环境。

17.4.2 执行信息资源标签<fmt:bundle>

一旦已经设置了 Web 应用的本地化环境之后, 就可以使用<fmt:bundle>标记了, 其体中可以包括一些调用本地化文本的<fmt:message>标签。<fmt:bundle>标签的使用格式一般如下:

```
<fmt:bundle basename="...the bundle's base name"
[prefix="...prefix name..."] >
<fmt:message key="...key name..." />
</fmt:bundle>
```

这个标签的属性描述如下:

- ❑ **Basename:** 资源文件 (这里使用文件, 而不是类, 在第八章介绍了编写资源类来实现国际化的方法) 的基名, 例如资源文件 `Res_zh_CN.property`, 则基名为 `Res`。
- ❑ **Prefix:** 这一设置是可选的, 如果指定这个属性, 就会为标签体中的嵌套的<fmt:message>标签附加了一个前缀。

当<fmt:bundle>标签体中嵌套<fmt:message>标签时, 这时<fmt:message>标签默认就是使用<fmt:bundle>标签中 `basename` 所指定的资源文件。

17.4.3 获取资源属性值标签<fmt:message>

该标签用于显示本地化的文本, 它通过 `key` 属性来取得资源文件中相应的消息。<fmt:message>标签的一般使用格式如下:

```
<fmt:message key="...name of property..." [bundle="...resourceBundle..."] [var="...variable name..."]
[scope="...scope of var..."] />
```

这个标签从资源文件中获取到一个消息, 生成相应的一个本地化文本串。该标签中的属性描述如下:

- ❑ **key:** 用于查找资源文件中相应的关键字名, 它对应着一条特定的消息。
- ❑ **bundle:** 如果设置了这个属性, 就会使用这个属性指定的资源文件。否则若嵌套在<fmt:bundle>标签中, 就会直接使用<fmt:bundle>标签中 `basename` 属性指定的资源文件。
- ❑ **var:** 如果指定这个属性, 则把取出的消息字符串存储在这个变量当中。
- ❑ **scope:** 这个属性指定了 `var` 设置变量所在的有效范围, 默认为 `page`, 有可以设置为 `session`。

17.4.4 设置资源文件标签<fmt:setBundle>

该标签用于设置一个资源文件, 并给定一个标记, 以便可以通过在<fmt:message>标签中指定 `bundle` 属性值来取得这个资源文件中的消息。其使用的格式一般如下:

```
<fmt:setBundle basename="...the bundle's base name..." var="...var name..."
[scope="page|request|session|application"] />
```

标签中的属性描述如下:

- ❑ **basename:** 该属性与<set:bundle>标签中的 `basename` 属性类似。
- ❑ **var:** 给指定的资源文件取一个变量名, 以便<fmt:message>标签可以通过这个变量名来读取该资源文件中消息。
- ❑ **scope:** 设置 `var` 属性所指定变量的有效范围。

该标签与<fmt:bundle>标签功能相类似, 只是<fmt:bundle>标签一般在体中嵌套使用<fmt:message>, 而<fmt:setBundle>标签用来先定义一个资源文件, 并给它指定一个变量, 以便在后面通过这个变量来单独使用<fmt:message>标签获取资源文件中的消息。

17.4.5 获取参数值标签<fmt:param>

该标签一般和<fmt:message>标签配套使用，用来给获取的消息中插入一个值。例如，资源文件中的一条消息如下：

密码错误 = “{0}的密码错误”

<fmt:message>标签首先通过 key=“密码错误”这个关键字找到以上这条消息，然后在<fmt:message>标签体中使用<fmt:param>标签赋一个值来替代{0}部分。

<fmt:param>标签的使用格式如下：

```
<fmt:message ...>
<fmt:param value=" value" />
</fmt:message>
```

其中 value 属性指定的值即为要替代{0}部分的值。

17.4.6 综合实例

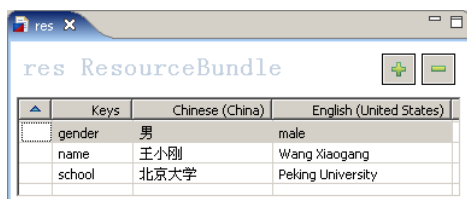
下面通过实例来具体演示以上介绍的各标签的使用方法。开发步骤如下：

17.4.6.1 创建资源文件

资源文件需要放置在创建 Web 模块 JSTL 的 WEB-INF\classes 目录或者子目录下，这里把资源文件创建在 MyJSTL 项目 j2src 源文件目录的 cn.jstl 包下，这样 Eclipse 会自动把这些资源文件输出到 WEB-INF\classes 目录下。

- ❑ res_zh_CN.properties 资源文件：中文的资源文件，基名为 res。
- ❑ res_en_US.properties 资源文件：对应的英文资源文件。
- ❑ app_zh_CN.properties 资源文件：中文的资源文件。
- ❑ app_en_US.properties 资源文件：对应的英文资源文件，基名为 app。

这些资源文件都创建在 cn.jstl 包下。图 17.13 和图 17.14 显示了这两种资源文件中的消息。



Keys	Chinese (China)	English (United States)
gender	男	male
name	王小刚	Wang Xiaogang
school	北京大学	Peking University

图 17.13 基名为 res 的资源文件



Keys	Chinese (China)	English (United States)
avocation	篮球、阅读	playing basketball and reading
courseMark	语文{0}、数学{1}、英语{2}	chinese{0}, math{1}, english{2}
currentTime	现在时间为{0}	The current time is {0}

图 17.14 基名为 app 的资源文件

这里对资源文件的编辑使用了一个名为 Jinto 的插件，下载地址为：<http://www.guh-software.de/>。在 Eclipse 中的安装方法和其他插件一样，可以参考第九章的内容。

17.4.6.2 创建 selectLocale.jsp 语言选择页面

首先创建一个语言选择页面 selectLocale.jsp，详细代码如下：

```
<%@ page contentType="text/html; charset=GBK"%>
<html>
<head><title>选择语言</title></head>
<body>
<h4>选择所要显示的语言</h4>
<form action="fmt.jsp" method="get">
  <select name="locale">
```

```

        <option value="zh_CN">中文</option>
        <option value="en_US">English</option>
    </select>
    <input type="submit" value="提交">
</form>
</body>
</html>

```

程序说明：此页面用于选择需要显示的语言，这里只提供了中文和英文的显示。这个页面的显示如图 17.15 所示。



图 17.15 语言选择页面

17.4.6.3 创建 fmt.jsp 国际化显示页面

根据上一页面选择的语言来设置本地化环境，然后读取相应的资源文件中的消息，从而实现程序代码的国际化和本地化。该页面的详细代码如下：

```

<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ 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>JSTL fmt</title></head>
<body>
<c:set var="loc" value="${param.locale}" />
<!-- 设置本地化环境 -->
<fmt:setLocale value="${loc}" />
<!--bundle 体中取得资源消息 --><br>
<fmt:bundle basename="cn.com.zzb.res">
    &nbsp;&nbsp;&nbsp;<fmt:message key="name"/><br>
    &nbsp;&nbsp;&nbsp;<fmt:message key="gender"/><br>
    &nbsp;&nbsp;&nbsp;<fmt:message key="school"/><br>
</fmt:bundle>

<!--setBundle 定义资源，再由 message 获得消息--><br>
<fmt:setBundle basename="cn.com.zzb.app" var="appBundle" />
&nbsp;&nbsp;&nbsp;<fmt:message key="avocation" bundle="${appBundle}"/><br>

<!--message 标签体中使用 param 标签--><br>
<jsp:useBean id="now" class="java.util.Date" />
<fmt:formatDate value="${now}" type="both" var="currentDateString"/>
<fmt:parseDate value="${currentDateString}" type="both" var="currentDate"/>

```

```
<fmt:message key="currentTime" bundle="${appBundle}" >
    <fmt:param value="${currentTime}"/>
</fmt:message><br>
<fmt:message key="currentTime" bundle="${appBundle}" >
    <fmt:param value="${currentDateString}"/>
</fmt:message><br>

<!--message 标签体中使用多个 param 标签--><br>
<fmt:message key="courseMark" bundle="${appBundle}" >
    <fmt:param value="${85}"/>
    <fmt:param value="${90}"/>
    <fmt:param value="${75}"/>
</fmt:message>

</body>
</html>
```

程序说明：

(1) 首先使用<c:set var="loc" value="\${param.locale}" />来获取到上一页面中传递过来的 locale 参数值，并把它保存在 loc 变量中。

(2) <fmt:setLocale value="\${loc}" />设置了本地化的语言环境。这里的 loc 变量为 zh_CN 或者 en_US。

(3) 该程序首先由<fmt:bundle>和<fmt:message>标签组合获取基名为 res 的资源文件中的消息。

(4) 然后通过<fmt:setBundle>标签首先定一个基名为 app 的资源文件，并给这个资源文件一个 appBundle 的标记。然后<fmt:message>标签通过这个标记来获取相应资源文件中的消息。

(5) <fmt:formatDate value="\${now}" type="both" var="currentDateString"/>语句把时间 now 转化为本地化的格式显示，并把它保存在 currentDateString 变量中。

(6) <fmt:parseDate value="\${currentDateString}" type="both" var="currentDate"/>是把本地化的时间显示格式换另一种显示方式。

(7) 在<fmt:message>标签中可以嵌套多个<fmt:param>标签来把相应的多个参数插入到要获取的消息中。

如果在上一个页面中选择是中文显示则会出现如图 17.16 所示的页面效果，如果选择为 english 显示，则出现如图 17.17 所示的页面。



图 17.16 中文显示页面

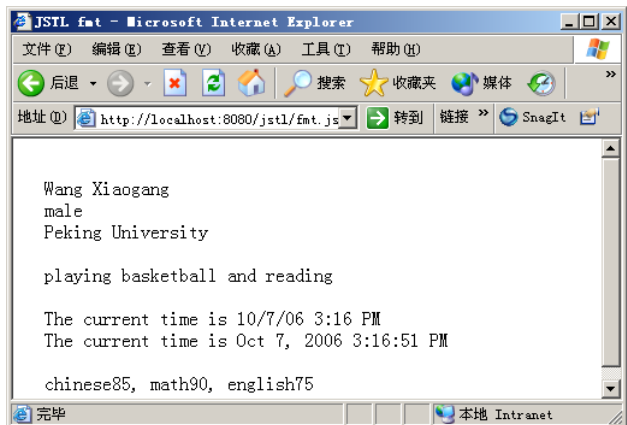


图 17.17 英文显示页面

17.4.7 格式化日期、货币的实例

接下来再具体介绍一下日期和货币的本地化方法。当然国际化和本地化的内容还不仅仅如此，至于其他方面就不一一介绍，读者有兴趣可以参考相关书籍。

17.4.7.1 货币的国际化

创建 `fmt_currency.jsp` 文件，其源代码如下：

```
<%@ page contentType="text/html; charset=GBK"%>
<%@ 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>Currency Formatting</title></head>
<body>
  <h3>Currency Formatting and locale:</h3>
  <hr>
  <!-- 中文，中国显示格式-->
  <h5>Chinese, China</h5>
  <fmt:setLocale value="zh_CN" />
  <fmt:formatNumber type="currency" value="80000" /><br/>
  <!-- 英语，英国显示格式-->
  <h5>English, Great Britain</h5>
  <fmt:setLocale value="en_GB" />
  <fmt:formatNumber type="currency" value="80000" /><br/>
  <fmt:formatNumber type="currency" value="80000" currencyCode="EUR"/><br/>
  <!-- 英语，美国显示格式-->
  <h5>English, USA</h5>
  <fmt:setLocale value="en_US" />
  <fmt:formatNumber type="currency" value="80000" /><br/>
</body>
</html>
```

程序说明：

(1) 本页面根据设置的本地化环境显示了三种不同的货币输出格式。

(2) `currencyCode` 属性的值必须是 ISO-4217 货币代码。货币代码都是三字母代码（均大写），可以在 unece.org/cefact/rec/cocucod.htm 找到完整的货币代码表。不过，并非所有货币代码在所有本地化环境中都得到支持，是否支持很大程度上取决于所用的 Java 2 平台实现。

该页面在浏览器中的显示效果如图 17.18 所示。

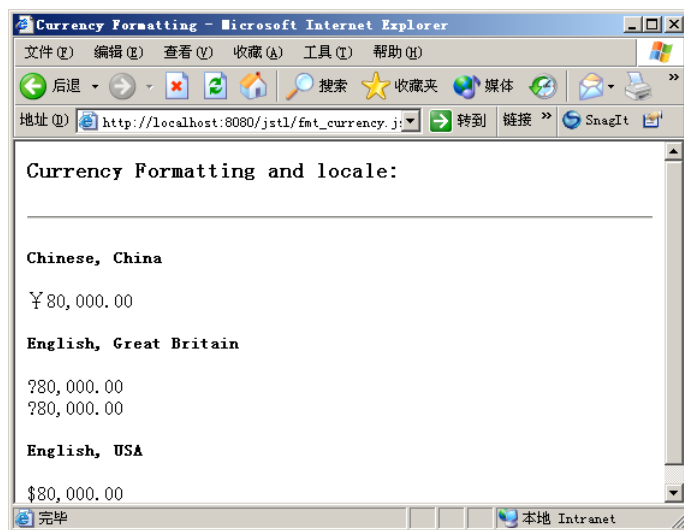


图 17.18 fmt_currency.jsp 页面效果

17.4.7.2 日期的本地化

下面再通过一个例子展示本地化环境对日期和时间信息的格式有什么影响。在创建实例之前，先了解一下<fmt:formatDate>这个标签。

该标签的使用格式一般如下：

```
<fmt:formatDate type="time|date|both" value="...value..." [dateStyle="short|medium|long|full"]
[timeStyle="short|medium|long|full"] />
```

属性说明如下：

- ❑ type: 可以是 time、date 或者 both（前两者一起）类型。控制输出的是否只是时间、只是日期、还是日期和时间组合。both 指定的是日期和时间的组合。
- ❑ value: 这是一个 java.util.Date 类型的一个值，用于生成日期和时间。
- ❑ dateStyle: 可以是 short、medium 或者 full 类型，full 为默认设置。控制打印日期使用的具体格式。
- ❑ timeStyle: 可以是 short、medium 或者 full 类型，full 为默认设置。控制打印时间使用的具体格式。

接下来创建一个实例文件 fmt_date.jsp，详细源代码如下：

```
<%@ page contentType="text/html; charset=GBK"%>
<%@ 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>Date Formatting</title></head>
<body>
  <h3>Date Formatting and locale:</h3>
  <hr>
  <fmt:timeZone value="EST">
    <jsp:useBean id="currentTime" class="java.util.Date" />
    <!-- 中文，中国显示格式-->
    <h5>Chinese, China</h5>
    <fmt:setLocale value="zh_CN" />
    <fmt:formatDate type="both" dateStyle="full" timeStyle="full" value="${currentTime}" /><br>
    <!-- 英语，英国显示格式-->
```

```
<h5>English, Great Britain</h5>
<fmt:setLocale value="en_GB" />
<fmt:formatDate type="both" dateStyle="full" timeStyle="full" value="${currentTime}" /><br>
<!-- 英语, 美国显示格式 -->
<h5>English, USA</h5>
<fmt:setLocale value="en_US" />
<fmt:formatDate type="both" dateStyle="full" timeStyle="full" value="${currentTime}" /><br>
</fmt:timeZone>
</body>
</html>
```

程序说明:

(1) 在本程序中除了设置一个本地化环境外, 在输出时间之前还有一点很重要, 就是要为用户设置一个时区。如果应用可能在国际范围内访问, 那么只采用服务器本身的时区来生成时间通常是没有意义的。理想情况下, 应当也由用户来选择时区, 具体方式与选择本地化环境类似。在这个例子中, 时区设置为东部标准时间。就是使用 JSTL 的<fmt:timeZone>标签完成的, 其中 value 属性指定一个地区 ID。

(2) 代码中使用标准<jsp:useBean>动作创建了 java.util.Date 类的一个 JavaBean 实例, 并用这个实例访问当前时间(服务器上的当前时间)。注意这个实例赋给了一个名为 currentTime 的 page 作用域变量。

17.5 XML 标签库

XML 是 Extensible Markup Language 的缩写, 即可扩展标记语言是一种您可以用来创建自己的标记的标记语言。它由万维网协会(W3C)创建, 用来克服 HTML (即超文本标记语言—Hypertext Markup Language, 它是所有网页的基础) 的局限。和 HTML 一样, XML 也基于标准通用标记语言(Standard Generalized Markup Language, SGML)。尽管 SGML 已在出版业使用了数十年, 但其理解方面的复杂性使许多本打算使用它的人望而却步。XML 是为 Web 设计的。

XML 提供了非常灵活的方式来表示结构化的数据, 可以创建自己的标记来识别数据, 不仅仅人可以看懂, 而且计算机也可以读懂。并且 XML 可以在不兼容的系统之间进行数据的交换, 所以 XML 必定将称为最普遍的数据操作和数据传输的工具。这也注定它能成为 Web 应用程序中极具引力的集成技术之一。

JSTL 标准标签库已经提供了一些关于 XML 操作的标签, 使得 JSP 开发者可以在不深入了解 SAX (Simple API for XML) 和 DOM (Document Object Model, 即文件对象模式) 等技术条件就可以处理 XML 文件。

JSTL 主要提供了如下多种类型的标签来处理 XML 文件:

- ☐ XML 核心动作
- ☐ XML 流程控制动作
- ☐ XML 转换动作

在 XML 操作过程, 一般先实现<c:import>标签把要处理的 XML 文件检索到, 然后再使用<x:parse>标签来解析这个文件, <x:parse>标签支持标准的 XML 解析技术, 例如文件对象模式(DOM)和简单的 XML API (SAX) 技术。还有标签<x:transform>标签可以使用来转换 XML 文件, 它使用的是扩展样式语言(Extensible Stylesheet Language, 简称 XSL) 技术。关于 XML 的标签库还提供了很多标签来处理解析后的 XML 数据, 其中就使用到了一种标准——XML 路径语言(XML Path Language, XPath), 用

来引用和读取解析后的 XML 文件中的数据。

17.5.1 XPath介绍

XPath 标准是用来对 XML 文档各部分数据进行定位的语言。它除了提供了一套定位语法之外，还包括一些函数，可以提供基本的数字运算、布尔运算以及字符串处理等功能。

XPath 使用了一种紧凑和非 XML 的语法来方便地实现 XPath 在 XML 属性值中使用，它基于 XML 文档的逻辑结构，在该结构中进行导航。除了用于定位，XPath 自身还包含了一个子集用于进行匹配，还可以用来验证一个节点是否匹配某个模式。XPath 是把一个 XML 文档看成一个树和节点的模式。节点的类型可以有多种，其中包括元素节点、属性节点和文本节点。

XPath 的基本语法由表达式构成。在计算机表达式的值后面生成一个对象，这个对象有以下四种基本类型：

- ☐ 节点集 (Node Set)
- ☐ 布尔数值
- ☐ 数字型
- ☐ 字符串类型

XPath 基本上和文件系统中查找文件相类似，如果路径是以 “/” 开头的，则表示该路径是一个决定路径，这和 UNIX 操作系统中关于文件路径的定义是一致的。

当然了 XPath 它本身有自己的一套完整的语法说明来进行 XML 文件的定位。这里不列出所有严格的定义表达式，而是使用一些实例来说明如何使用 XPath 语言。

下面是一个 XML 文件：

```
<authors>
  <author>
    <name>Zou Hai</name>
    <nationality>China</nationality>
  </author>
  <author period="classical">
    <name>Johnson</name>
    <nationality>French</nationality>
  </author>
  <author>
    <name>Kitten</name>
    <nationality>German</nationality>
  </author>

  <super-author>
    <name>Chen Yong</name>
    <nationality>China</nationality>
  </super-author>
</authors>
```

17.5.1.1 关于 “/” 和 “//” 的使用

“/” 是表示当前文档的节点，类似 DOS 目录分割符，“//” 则表示当前文档所有的节点。类似查看整个目录。

(1) /authors/author: 表示选择根目录下、父节点为 authors 的元素 “author”。

(2) `/authors/author/name`: 表示查找到所有名称为“name”的元素,但是它的父节点为“author”,而“author”的父节点又为“authors”。

(3) `//name`: 表示查找 XML 文件中的所有“name”元素,而无论这个元素在什么层次。

17.5.1.2 关于“*”的使用

“*”标记表示某个层次上的所有元素。

(1) `/authors/author/*`: 表示在 author 元素(它的父节点为 authors)下的所有子元素(本例为 name 和 nationality 两个元素)。

(2) `/authors/*/name`: 表示查找所有名为 name 的元素,而不管它的父节点是谁,或者是 author,也或者是 super-author 元素,但是再上一个父节点必须是 authors 元素。

(3) `//*`: 表示查找到所有元素。

17.5.1.3 路径分支

方括号表示路径分支。

(1) `/authors/author[nationality]/name`: 表示只查找那些在 authors 元素下包含 nationality 子元素的 author 元素的 name 节点。

(2) `authors/author[nationality='Russian']/name`: 表示查找那些 nationality 子元素值为 German 的 author 元素下的 name 节点,而且 author 元素的父节点为 authors。

10.5.2 XML核心动作标签

包括的核心动作标签有如下几个:

- ❑ `<x:parse>`: 用于解析 XML 文件。
- ❑ `<x.out>`: 通过 XPath 来读取 XML 文件中的某元素。
- ❑ `<x:set>`: 该标签用来计算 XPath 表达式,并且把结果保存在指定的变量当中。

17.5.2.1 <x:parse>标签

该标签的使用格式如下:

```
<x:parse doc="XMLDocument"
[var="varName"] [scope="scope"] | [varDom="varName"] [scopeDom="scope"]
[systemId="systemId"] [filter="filter"] />
```

标签中的各属性描述如下:

- ❑ doc: 指定要解析的 XML 文件。一般使用`<c:import>`检索到相应的 XML 文件。
- ❑ var: 把解析之后的 XML 文件存储在 var 属性指定的变量中。
- ❑ scope: 设置 var 属性指定的变量有效范围。
- ❑ varDom: 存储解析后的 XML 文件。
- ❑ scopeDom: 设置 varDom 属性指定变量的有效范围。
- ❑ systemId: 指定 XML 文件的 URI。
- ❑ filter: 对应的是 org.xml.sax.XMLFilter 类。

下面还是通过一个`<x:parse>`标签使用的实例,创建一个 x_parse.jsp 文件,源代码如下:

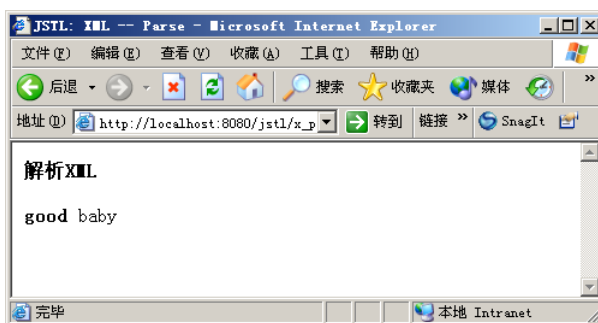
```
<%@ page contentType="text/html;charset=GBK" %>
<html>
<head>
  <title>JSTL: XML -- Parse</title>
</head>
```

```
<body>
<h4>解析 XML</h4>
<c:set var="xmlText">
  <a>
    <b>
      <c>good</c>
    </b>
    <d>
      <c>baby</c>
    </d>
  </a>
</c:set>
<x:parse var="xmlDoc" doc="{xmlText}" />
<x:out select="$xmlDoc/c"/><br>
<x:out select="$xmlDoc/a/d"/><br>
<x:out select="$xmlDoc/a/*/"/><br>
</body>
</html>
```

程序说明:

(1) 该程序中, 首先使用<c:set>定义一个 xml 文件内容, 并保存在 xmlText 变量中, 然后通过<x:parse>标签对这个 XML 文件内容进行解析。

(2) 最后使用<x:out>标签和指定的 XPath 表达式来查找 XML 文件的元素。该页面运行结果如图 17.19 所示。



17.19 x_parse.jsp 页面运行效果

(3) 还存在另外一个解析 XML 文档的方法, 直接把 XML 部分定义在<x:parse>解析标签的体中。修改代码如下所示:

```
<x:parse var="xmlDoc">
  <a>
    <b>
      <c>good</c>
    </b>
    <d>
      <c>baby</c>
    </d>
  </a>
</x:parse>
```

这样就可以直接对体中的 XML 文档进行了解析, 然后调用<x:out>标签对指定的元素进行输出。

17.5.2.2 <x:out>标签

该标签是用于计算 XPath 表达式，然后把查找到的元素进行输出。

其一般的使用格式如下：

```
<x:out select="XPathExpression" [escapeXml="true|false"] />
```

属性说明如下：

- select：将要计算的 XPath 表达式。
- escapeXml：确定<、>、&、'、"这些字符是否被转换成字符实体代码，默认为 true。

接下来还是通过一个实例进行讲解，首先需要创建一个 XML 文件，这个文件名为 games.xml，代码如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<games>
  <country id="Luxembourg">
    <athlete>
      <name>Lux 1</name>
      <sport>swimming</sport>
      <age>23</age>
      <gender>M</gender>
    </athlete>
    <athlete>
      <name>Lux 2</name>
      <sport>wrestling</sport>
      <age>31</age>
      <gender>M</gender>
    </athlete>
  </country>
  <country id="Denmark">
    <athlete>
      <name>Den 1</name>
      <sport>cycling</sport>
      <age>18</age>
      <gender>F</gender>
    </athlete>
    <athlete>
      <name>Den 2</name>
      <sport>sailing</sport>
      <age>27</age>
      <gender>M</gender>
    </athlete>
  </country>
</games>
```

然后创建一个处理 XML 文件的 JSP 文件，在这个 JSP 文件中使用<c:import>标签检索到上面的 XML 文件，这个 JSP 文件取名为 x_out.jsp，其源代码如下：

```
<%@ page contentType="text/html; charset=GBK"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
  <title>JSTL: XML -- Parse / Out</title>
```

```

</head>
<body>
<h3>parse 和 out 标签的实例</h3>
<c:import var="docString" url="games.xml"/><!--引入 games.xml 文件-->
<x:parse var="doc" doc="{docString}"/>
<table border=1>
  <tr>
    <td valign="top"><pre><c:out value="{docString}"/></pre></td>
    <td valign="top">
      <table border=1>
        <tr>
          <th>XPath 表达式</th>
          <th>查找到的元素</th>
        </tr>
        <tr>
          <td>${doc//sport}</td>
          <td><pre><x:out select="${doc//sport}"/></pre></td>
        </tr>
        <tr>
          <td>${doc/games/country/*}</td>
          <td><pre><x:out select="${doc/games/country/*}"/></pre></td>
        </tr>
        <tr>
          <td>${doc//}*</td>
          <td><pre><x:out select="${doc//}*"/></pre></td>
        </tr>
        <tr>
          <td>${doc/games/country}</td>
          <td><pre><x:out select="${doc/games/country}"/></pre></td>
        </tr>
        <tr>
          <td>${doc/games/country[last()]}</td>
          <td><pre><x:out select="${doc/games/country[last()]}"/></pre></td>
        </tr>
        <tr>
          <td>${doc//@id}</td>
          <td><pre><x:out select="${doc//@id}"/></pre></td>
        </tr>
        <tr>
          <td>${doc//country[@id='Denmark']}</td>
          <td><pre><x:out select="${doc//country[@id='Denmark']}"/></pre></td>
        </tr>
      </table>
    </td>
  </tr>
</table>
</body>
</html>

```

程序说明：以上程序中基本包含 XPath 查找元素的所有方法。

可以看出,有了 XML 标签库之后,对 XML 文件的处理方便多了,不再需要手工使用 JAXP(Java API's

for XML Processing-Java XML)配合其他的一些解析器来除了 XML 文档。

17.5.2.3 <x:set>标签

该标签首先计算 XPath 表达式，然后把计算的结果保存在指定的变量中，而不是输出。

<x:set>标签的一般使用格式如下：

```
<x:set select="XPathExpression"
      var="varName" [scope="page|request|session|application"] />
```

标签中各属性描述如下：

- ❑ select: 将要被计算的 XPath 表达式。
- ❑ var: 把 XPath 表达式计算之后的值保存在这个属性所指定的变量中，以后可以通过指定的变量来引用这个值。
- ❑ scope: 设置 var 属性所指定变量的有效范围。

下面还是通过一个简单的实例来了解这个标签的具体使用方法，创建一个 x_set.jsp 文件，其源代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
  <title>JSTL: XML -- Parse</title>
</head>
<body>
<h4>JSTL -- set 标签</h4>
<c:set var="xmlText">
<a>
  <d>
    <c>baby</c>
    <k>good</k>
  </d>
</a>
</c:set>
<x:set var="doc" select="$xmlText//d"/>
<x:out select="$doc/c"/>
</body>
</html>
```

程序说明：该程序首先通过<x:set>标签把\$xmlText//d 表达式计算得到的结果保存在 doc 变量中。其实可以使用<x:out select="\$xmlText//d/c" />替代下面两行代码：

```
<x:set var="doc" select="$xmlText//d"/>
<x:out select="$doc/c"/>
```

使用<x:set>标签只是把要重复使用的元素部分保存起来，方便以后的调用。

10.5.3 XML的流程控制动作

用于 XML 流程控制的标签包括：

- ❑ <x:if>
- ❑ <x:choose>

- ☐ `<x:when>`
- ☐ `<x:otherwise>`
- ☐ `<x:forEach>`

这些标签和核心标签库中的`<c:if>`、`<x:choose>`、`<x:otherwise>`以及`<x:forEach>`非常相似，不同的只是 XML 流程控制标签使用的是 XPath 表达式。

17.5.3.1 `<x:if>` 标签

通过计算 `select` 属性指定的 XPath 表达式是否为正，即如果 XPath 表达式指定的元素存在则为正，否则为假，然后根据判断结果决定是否输出`<x:if>`体中的代码程序或者表达式。`<x:if>`标签使用格式如下：

```
<x:if select="XPathExpression" var="varName" [scope="page|request|session|application"] />
```

如果这个标签有一个体，使用格式如下：

```
<x:if select="XPathExpression" [var="varName"] [scope="page|request|session|application"] >
Body content
</x:if>
```

各属性描述如下：

- ☐ `select`：指定的 XPath 表达式，如果这个表达式指定的元素存在返回 `true`，否则返回 `false`。
- ☐ `var`：保存判断的结果，`true` 或者 `false`。
- ☐ `scope`：设置 `var` 属性指定变量的范围。

通过下面的小例子具体认识`<x:if>`标签的使用，创建一个 `x_if.jsp` 文件，源代码如下：

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head><title>JSTL: XML -- If</title></head>
<body>
<h3>If 标签</h3>
<x:parse var="xmlText">
  <a>
    <b>
      <c>foo</c>
    </b>
    <d>bar</d>
  </a>
</x:parse>
<x:if select="$xmlText//c"><!--XPath 表达式指定的元素存在，返回 true-->
  $a//c exists
</x:if><br>
<x:if select="$xmlText/a/d"><!--XPath 表达式指定的元素存在，返回 true-->
  $a/a/d exists
</x:if><br>
<x:if select="$xmlText/w/o/l"><!--XPath 表达式指定的元素不存在，返回 false-->
  $a/w/o/l exists
</x:if><hr>
</body>
</html>
```

程序说明：XML 文档内容直接嵌套在`<x:parse>`标签体中进行解析，`<x:if>`标签根据 XPath 表达式判断的结果决定是否输出体中的代码。

17.5.3.2. `<x:choose>`、`<x:when>`和`<x:otherwise>`标签

这三个标签一般配套使用，与核心标签库中的流程控制标签一样，`<x:when>`和`<x:otherwise>`作为子标签嵌套在`<x:choose>`标签体中。这三个标签的一般组合使用格式如下：

```
<x:choose>
  <x:when select="XPathExpression">
    Body content1
  </x:when>
  ...
  <x:otherwise>
    Conditional block
  </x:otherwise>
</x:choose>
```

下面还是看一个实例，创建 `x_choose.jsp` 文件，其源代码如下：

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head><title>JSTL: XML -- choose When otherwise</title></head>
<body>
<h3>choose - when - otherwise</h3>
<x:parse var="a">
  <a>
    <b>
      <c foo="bar">foo</c>
    </b>
    <d>bar</d>
  </a>
</x:parse>
<x:choose>
  <x:when select='$a//c[@foo="bar"]'>
    @foo = bar
  </x:when>
  <x:when select='$a//c[@foo="foo"]'>
    @foo = foo
  </x:when>
  <x:otherwise>
    @foo not recognized
  </x:otherwise>
</x:choose>
</body>
</html>
```

程序说明：

(1) 在节点 `c` 处指定了一个 `foo="bar"` 的属性。而 XPath 表达式 “`//c[@foo="bar"]`” 表示查找所有属性 `foo` 为 “bar” 的 `c` 节点。

(2) `<x:when>` 标签判断 `select` 属性指定的 XPath 表达式，如果这个表达式查找的元素存在则返回 `true`，否则返回 `false`。`true` 值表示输出 `<x:when>` 体中的代码，`<x:otherwise>` 标签则表示之上所有 `<x:when>` 标签都返回 `false` 时，执行这个体中的代码。

(3) `<x:otherwise>` 只能包含在 `<x:choose>` 体中，而且是放在 `<x:when>` 标签的最后面。

17.5.3.3 迭代标签 `<x:forEach>`

`<x:forEach>` 标签首先通过计算 `select` 属性所指定的 XPath 表达式，然后把得到的元素集合作迭代处

理。基本使用格式如下：

```
<x:forEach [var="varName"] select="XPathExpression" >
body content
</x:forEach>
```

下面还是通过一个实例来加深对这个标签使用的理解，创建一个 x_forEach.jsp 页面，源代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head><title>JSTL: XML -- forEach</title></head>
<body>
<h3>forEach 实例</h3>
<x:parse var="xmlText">
  <books>
    <book>
      <name>JSP2.0</name>
      <publisher>清华大学出版社</publisher>
      <price>48</price>
      <authors>
        <author id="1">
          <name>Johnson</name>
          <address>Johnson@hotmail.com</address>
        </author>
        <author id="2">
          <name>Tom</name>
          <address>Tom@hotmail.com</address>
        </author>
      </authors>
    </book>
  </books>
</x:parse>

<x:forEach select="$xmlText/books">
  &nbsp;&nbsp;&nbsp;<x:out select="."/><br>
</x:forEach>
<x:forEach select="$xmlText//book">
  &nbsp;&nbsp;&nbsp;<x:if select="//author">
    <x:out select="//author/name" />
  </x:if><br>
</x:forEach>
<x:forEach select="$xmlText//book">
  &nbsp;&nbsp;&nbsp;<x:choose>
    <x:when select="$xmlText//author[@id='2']">
      <x:out select="$xmlText//author[@id='2']"/>
    </x:when>
    <x:otherwise>
      no author's id=2
    </x:otherwise>
  </x:choose>
</x:forEach>
</body>
```

```
</html>
```

程序说明：在程序中的`<x:out select="."/>`表示将无条件的选择所有存在的元素进行输出。可以查看图 17.20 所示的输出结果。

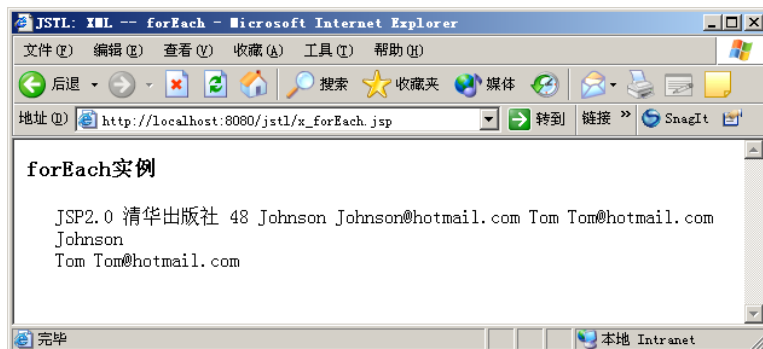


图 17.20 x_forEach.jsp 页面效果

17.5.4 XML 的转换动作

XML 标签库提供的 XML 转换动作标签使得 XML 文件以 XSLT 可扩展样式表(它是一种用于 XML 文件转换的语言，作为 Web 应用的一种表示层技术已经非常流行)转换成为了可能。

转换动作主要使用到了两个标签：

- ❑ `<x:transform>`：用于把 XML 文件转换成 XSLT 格式。
- ❑ `<x:param>`：和`<x:transform>`配套使用，用于传递参数。

17.5.4.1 `<x:transform>` 标签

与`<x:parse>`标签情况一样，`<x:transform>`标签支持多种不同的格式。`<x:transform>`最基本的格式的语法是：

```
<x:transform xml="expression" xslt="expression"
    var="name" scope="scope"
    xmlSystemId="expression" xsltSystemId="expression">
  <x:param name="expression" value="expression"/>
  ...
</x:transform>
```

格式说明：

- ❑ 此处，`xml` 属性规定要被转换的文件，`xslt` 属性规定定义这次转换的样式表(文件后缀为“.xsl”)。这两种属性是必要的，其他属性为可选。
- ❑ 与`<x:parse>`的 `xml` 属性一样，`<x:transform>`的 `xml` 属性值可以是包含 XML 文件的字符串，或者是接入这类文件的 `Reader`。此外，它还可以采用 `org.w3c.dom.Document` 类或 `javax.xml.transform.Source` 类的实例格式。最后，它还可以是使用`<x:parse>`操作的 `var` 或 `varDom` 属性分配的变量值。

另外，还可以根据`<x:transform>`操作的主体内容来包含要被转换的 XML 文件。在这种情况下，`<x:transform>`的语法是：

```
<x:transform xslt="expression"
    var="name" scope="scope"
    xmlSystemId="expression" xsltSystemId="expression">
  body content (XML document)
```

```
<x:param name="expression" value="expression"/>
...
</x:transform>
```

格式说明：

- ❑ 在这两种情况下，规定 XSL 样式表的 xslt 属性应是字符串、Reader 或 javax.xml.transform.Source 实例。
- ❑ 如果 var 属性存在，转换后的 XML 文件将分配给相应的 var 属性指定的变量，作为 org.w3c.dom.Document 类的一个实例。通常，scope 属性规定这类变量分配的范围。

<x:transform> 标记还支持将转换结果存储到 javax.xml.transform.Result 类的一个实例中，而不是作为 org.w3c.dom.Document 的一个实例。如果 var 和 scope 属性被省略，result 对象规定作为 result 属性的值，<x:transform> 标记将使用该对象来保存应用该样式表的结果。下面介绍了使用 <x:transform> 的 result 属性的这两种语法的变化：

```
<x:transform xml="expression" xslt="expression"
result="expression"
xmlSystemId="expression" xsltSystemId="expression">
<x:param name="expression" value="expression"/>
...
</x:transform>

<x:transform xslt="expression"
result="expression"
xmlSystemId="expression" xsltSystemId="expression">
body content (XML document)
<x:param name="expression" value="expression"/>
...
</x:transform>
```

格式说明：无论您采用这两种 <x:transform> 格式中的那一种，您都必须从定制标记单独创建 javax.xml.transform.Result 对象。该对象自身作为 result 属性的值提供。

如果既不存在 var 属性，也不存在 result 属性，转换的结果将简单地插入到 JSP 页面，作为处理 <x:transform> 操作的结果。当样式表用于将数据从 XML 转换成 HTML 时尤其有用，如下所示：

```
<c:import var="rssFeed" url="http://slashdot.org/slashdot.rdf"/>
<c:import var="rssToHtml" url="/WEB-INF/xslt/rss2html.xsl"/>
<x:transform xml="{rssFeed}" xslt="{rssToHtml}"/>
```

在本例中，使用 <c:import> 标记来读取 RSS 反馈和适当的样式表。样式表的输出结果是 HTML，通过忽略 <x:transform> 的 var 和 result 属性来直接显示。

与 <x:parse> 的 systemId 属性一样，<x:transform> 的 xmlSystemId 和 xsltSystemId 属性用于解析 XML 文件中相关的路径。在这种情况下，xmlSystemId 属性应用于根据标记的 xml 属性值提供的文件，而 xsltSystemId 属性用于解析根。

17.5.4.2 <x:param> 标签

该标签是用来给 <x:transform> 标签传递参数，并且是嵌套在 <x:transform> 标签体中。

它基本的格式语法有如下两种：

- (1) 传递的参数变量值在 “value” 属性中设置：

```
<x:param name="varname" value="value" />
```

- (2) 传递的参数值在标签体中指定：

```
<x:param name="varname" >
```

Parameter value

</x:param>

其中 “name” 属性用来指定转换参数的变量名, “value” 属性设定参数的值。

17.5.4.3 实例

下面通过一个简单的实例来演示这两个标签的具体使用方法, 创建的文件 x_transform.jsp 代码如下:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<%@ page contentType="text/html; charset=GBK"%>
<%@ page isELIgnored="false" %>
<html>
<head>
  <title>JSTL: XML -- Transform, Param</title>
</head>
<body bgcolor="#FFFFFF">
<h3>Transform and Param</h3>
<hr>
<!-- 设定 XML 文档 -->
<c:set var="xml">
  <a>
    <b>header!</b>
    <c>the two header!</c>
  </a>
</c:set>
<!-- 设定一个 XSL 文件 -->
<c:set var="xsl">
  <?xml version="1.0"?>
  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="text()">
      <h4><xsl:value-of select="."/></h4>
    </xsl:template>
  </xsl:stylesheet>
</c:set>
<!-- 基本的直接显示转换内容 -->
Prints "header" as a header:<br>
<x:transform doc="{xml}" xslt="{xsl}" />
<hr>
<!-- 把转换内容保存在 doc1 变量中, 再调用<x:out>输出 -->
Prints "header" in normal size:<br>
<x:transform doc="{xml}" xslt="{xsl}" var="doc1" />
<x:out select="$doc1//h4" />
<hr>
<!-- 先解析 xml 文档, 再把它 b 元素下的内容保存再 miniDoc 变量中, 再转换显示--> <h3>Transformations using
output from XPath expressions</h3>
<x:parse var="xml" doc="{xml}" />
<x:set var="miniDoc" select="$xml//b" />
<x:transform xslt="{xsl}" doc="{miniDoc}" />
<hr>
<!-- 转换结果直接插入到 jsp 页面中 -->
```

```
<h3>Inline transformations:</h3>
<x:transform xslt="{xsl}">
  <a>
    <b>
      <c>Paragraph one!</c>
      <d>Paragraph foo!</d>
    </b>
  </a>
</x:transform>
</body>
</html>
```

程序说明：该实例演示了<x:transform>标签的不同使用格式。

17.6 函数（function）标签库

JSTL 中包含了很多常用的功能函数的标签，使得 Web 开发者能够更加方便和快捷地开发出应用系统，而不需要为那些基本的功能再编写 JavaBean 类，并使得代码得到了很好的可重用性。

函数标签库中标签基本分成两种：

- ☐ 长度度量的函数，例如 fn:length()。
- ☐ 字符串操作函数

如下详细列出了各类函数标签的功能：

- ☐ fn:contains(string, substring)：如果参数 string 中包含参数 substring，返回 true。
- ☐ fn:containsIgnoreCase(string, substring)：如果参数 string 中包含参数 substring（忽略大小写），返回 true。
- ☐ fn:endsWith(string, suffix)：如果参数 string 以参数 suffix 结尾，返回 true。
- ☐ 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。
- ☐ fn:indexOf(string, substring)：返回参数 substring 在参数 string 中第一次出现的位置。
- ☐ 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。
- ☐ 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。
- ☐ 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。
- ☐ fn:split(string, separator)：返回一个数组，以参数 separator 为分割符分割参数 string，分割后的每一部分就是数组的一个元素。
- ☐ fn:startsWith(string, prefix)：如果参数 string 以参数 prefix 开头，返回 true。
- ☐ 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，包括 end 位置的字符。
- ☐ fn:substringAfter(string, substring)：返回参数 substring 在参数 string 中后面的那一部分字符串。
- ☐ fn:substringBefore(string, substring)：返回参数 substring 在参数 string 中前面的那一部分字符串。
- ☐ fn:toLowerCase(string)：将参数 string 所有的字符变为小写，并将其返回。

- ❑ `fn:toUpperCase(string)`: 将参数 `string` 所有的字符变为大写，并将其返回。
- ❑ `fn:trim(string)`: 去除参数 `string` 首尾的空格，并将其返回。

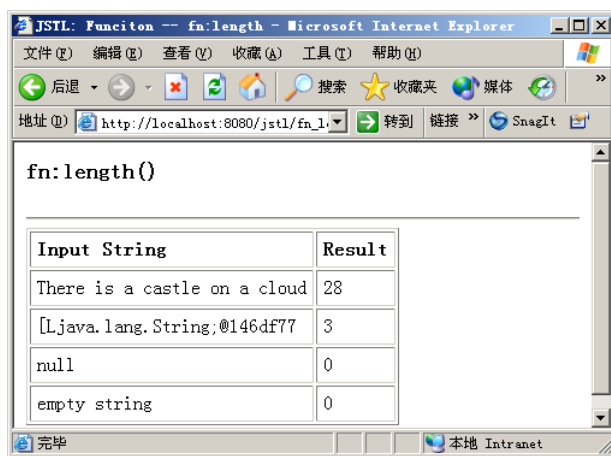
17.6.1 综合实例

下面以一个简单的实例作示范，其他函数的使用方法相类似。创建一个 `fn_length.jsp` 文件，其源代码如下：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ page contentType="text/html;charset=GBK"%>
<%@ page isELIgnored="false" %>
<html>
<head>
  <title>JSTL: Function -- fn:length</title>
</head>
<body bgcolor="#FFFFFF">
<h3>fn:length()</h3>
<hr>
<c:set var="s1" value="There is a castle on a cloud"/>
<%
  String[] customers = {"tom","johnson","jake"};
  session.setAttribute("customers",customers);
%>
<table cellpadding="5" border="1">
  <tr>
    <th align="left">Input String</th>
    <th>Result</th>
  </tr>
  <tr>
    <td>${s1}</td>
    <td>${fn:length(s1)}</td>
  </tr>
  <tr>
    <td>${customers}</td>
    <td>${fn:length(customers)}</td>
  </tr>
  <tr>
    <td>null</td>
    <td>${fn:length(undefined)}</td>
  </tr>
  <tr>
    <td>empty string</td>
    <td>${fn:length("")}</td>
  </tr>
</table>
</body>
</html>
```

程序说明：`s1` 为字符串，所以调用 `${fn:length(s1)}` 返回字符串 `s1` 中的字符个数；`customers` 为一个

字符串数组，所以调用`{fn:length(customers)}`返回 `customers` 数组的维数；当字符串变量为 `NULL` 或者空时，但返回 0。该页面运行效果如图 17.21 所示。



17.21 fn_length.jsp 页面运行效果

其他函数调用方式类似，这就不一一列举。

17.7 SQL标签库

JSTL 标准标签库还提供了数据库操作的标准标签，使得 Web 开发者不再需要在 JSP 页面中嵌入脚本语言或者编写 `JavaBean` 类来连接数据库，使得页面风格更加统一和易于维护。

用于数据库操作的主要标签包括如下：

- ☐ `<sql:setDataSource>`：该标签用于数据库连接设置。
- ☐ `<sql:transaction>`：把其他`<sql:query>`以及`<sql:update>`操作放置在一个事务中。
- ☐ `<sql:query>`：进行数据库查询操作。
- ☐ `<sql:update>`：进行数据库更新操作。
- ☐ `<sql:param>`：进行参数的设定。
- ☐ `<sql:dateParam>`：进行日期型参数的设置。

17.7.1 综合实例

下面通过一个综合实例来演示以上各标签的具体使用方法，创建的 `sql_all.jsp` 代码如下：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<%@ page contentType="text/html; charset=GBK"%>
<%@ page isELIgnored="false" %>
<html>
<head>
  <title>JSTL: SQL</title>
</head>
<body bgcolor="#FFFFFF">
  <h3>SQL 标签库</h3>
  <hr>
```

```

<%
request.setAttribute("newName", new String("Jake")); //设定 newName 变量，并保存在 request 范围内
//设数据库连接驱动到 session 范围的 myDbDriver 变量中
session.setAttribute("myDbDriver", "com.mysql.jdbc.Driver");
//设定数据库连接到 session 范围的 myDbUrl 变量中
session.setAttribute("myDbUrl", "jdbc:mysql://localhost/mydb");
session.setAttribute("myDbUserName", "root"); //设定数据库用户名
session.setAttribute("myDbPassword", "123456"); //设定数据库用户密码
%>
<!-- 首先设定数据库连接 -->
<sql:setDataSource
  var="example"
  driver="${sessionScope.myDbDriver}"
  url="${sessionScope.myDbUrl}"
  user="${sessionScope.myDbUserName}"
  password="${sessionScope.myDbPassword}"
/>
<!-- 把数据库查询和更新操作放在一个事务中 -->
<sql:transaction dataSource="${example}">
  <!-- 使用<sql:update>标签创建一个数据库表 mytable -->
  <sql:update var="newTable">
    create table mytable (
      nameid int primary key,
      name varchar(80)
    )
  </sql:update>
  <!-- 向 mytable 表中插入数据 -->
  <sql:update var="updateCount">
    INSERT INTO mytable VALUES (1,'Tom')
  </sql:update>
  <!-- 向 mytable 表中插入数据,并使用<sql:param>标签插入参数值 -->
  <sql:update var="updateCount">
    INSERT INTO mytable VALUES (?, 'Johnson')
    <sql:param value="2"/>
  </sql:update>
  <!-- 向 mytable 表中插入数据 -->
  <sql:update var="updateCount">
    INSERT INTO mytable VALUES (?,?)
    <sql:param value="3"/>
    <sql:param value="${newName}"/>
  </sql:update>
  <!-- 查询数据库表 mytable 中所有内容 -->
  <sql:query var="deejay">
    SELECT * FROM mytable
  </sql:query>
</sql:transaction>
<!-- 对返回的 Result 结果中的每行的列进行迭代处理 ,取出每一列-->
<table border="1">
  <c:forEach var="row" items="${deejay.rowsByIndex}">
    <tr>

```

```
<c:forEach var="column" items="${row}">
    <td><c:out value="${column}" /></td>
</c:forEach>
</tr>
</c:forEach>
</table>
<!-- 没有指定每行的列索引,注意与上面的不同, 这里可以指定取出哪一列-->
<table border="1">
    <c:forEach var="row" items="${deejay.rows}">
        <tr>
            <td><c:out value="${row.nameid}" /></td>
            <td><c:out value="${row.name}" /></td>
        </tr>
    </c:forEach>
</table>
<!-- 把 mytable 数据库表删除 -->
<sql:update var="newTable" dataSource="${example}">
    drop table mytable
</sql:update>
</body>
</html>
```

程序说明:

(1) 首先进行了数据库连接的各参数的设定, 并把它们存储在 session 范围内的变量中, 这样有利于在整个 Web 应用中的其他页面中进行调用。

(2) 把整个的数据库表的创建、插入以及查询操作放置在<sql:transaction>标签设定的一个事务中。

(3) 在获取查询信息时, 本实例使用了两种方法, 前一种方法使用索引迭代取出行中每个列的数据; 第一种方法没有使用索引, 而是通过指定行中的特定列来取得数据。

17.8 本章小结

本章重点介绍了标准标签库 JSTL, 其中包括核心标签库, 提供了 Web 开发的日常功能; 国际化标签库, 提供了用于国际化和本地化程序开发的各个标签, 使用非常方便; XML 标签库, 对 XML 文件处理提供支持; 函数标签库, 提供了一些包含字符串和数组长度度量以及字符串操作的函数; SQL 标签库, 可以使用各类标签进行数据库的连接, 以及各表的创建、插入和删除等操作, 使得 Web 开发者不再需要插入脚本语言或者编写 JavaBean 来进行数据库操作。

在上一章节中, 介绍的自定义标签给 JSP 开发带来的好处, JSTL 标准标签库基本提供了大部分进行动态网页开发的功能, 使得 Web 开发者可以达到面向文档的开发方式, 摆脱了 JavaBean 类以及脚本语言的编程。

在这一章节中已经多次使用 EL 表达式语言, 它和 JSTL 能够得到很好的搭配使用, 这部分内容将在一下章节重点介绍。