第 16 章 自定义标签的开发

JavaBean 的使用实现了业务逻辑与视图层的分离，使得代码重用得到了提高，另外也分化了工作职责，使得专注于页面显示的工作更加容易。JSP 自带的标准动作元素和标签（见第八章内容）已经简化了 JSP 页面的开发过程。这一章将教会读者如何自定义一个 JSP 标签，并且用它来封装一些动态功能。JSP 技术允许开发者自定义标签，这些自定义的标签将成为 JSP 语言的一个扩展。

自定义标签通常是以标签库的形式出现的，它定义了一组相关的自定义标签，并包含实现这些标签的对象类，就像 JSP 中已有的<jsp:useBean>、<jsp:getProperty>和<jsp:forward>这些标签。

自定义的标签可以执行的任务包括对隐式对象的操作、处理表单、访问数据库和其他企业级服务，例如电子邮件和目录，以及执行流程控制等。

本章要点包括以下内容：

- 标签库的体系结构
- 简单格式的标签开发
- 定义带有属性的标签开发
- 自定义带有体的标签开发
- 标签体中嵌套其他子标签的开发
- 简单标签处理－JSP2.0 新特性

16.1 标签库的体系结构

自定义标签其实就是一个个相对应的处理类，当服务器在 JSP 页面上遇到自定义标签时，会调用相应的处理类，而在标签中属性设置就相当于给对应处理类中的属性赋值。当多个同类型的标签组合在一起就形成了一个标签库，这时候还需要为这个标签库中的所有标签编写一个描述性的配置文件，这样服务器才能通过页面上的标签查找到相应的处理类。

所以概括起来说，编写自定义标签就是要编写相应的处理类以及描述性的配置文件。另外，进行 JSP 标签库的开发，需要开发人员精通 Java 编程语言和对访问数据库和其他服务非常熟悉。当专门 Java 开发者开发出这些标签库后，Web 应用程序开发人员就可以把注意力放在内容和格式的显示上面，而不要费心考虑如何访问企业级服务等复杂问题。这样可以使得 Web 页面开发者和标签库开发者相对分开，使他们能够专注于自己的工作。

JavaBean 类开发的实现可以一定程度上使得精通 Java 底层开发者和 Web 页面开发者独立开来。但是 Web 页面开发者对 JavaBean 类的调用却不是很方便，而且这样使得 JSP 页面中还必须存在一定的 Java 程序，使得页面显得混乱，从而不易维护。在 JSP 应用程序中添加自定义标签的能力可以使得 JSP 页面开发者能够将工作重点放到以文档为中心的开发方式上，使得 Web 页面开发者可以很容易调用各种功能实现页面动态。

16.1.1　自定义标签库的概念

自定义标签是开发者自己定义的 JSP 语言元素，它的功能类似于 JSP 自带的<jsp:forward>等标准动作元素。当包含自定义标签的 JSP 页面被翻译成 Servlet 后，这个标签就会转换成一个名为标签处理类（tag handler）的对象操作。当 JSP 页面的 Servlet 请求执行时，Web 容器就会调用这些对象类中的操作方法。

总结起来，自定义标签的功能和好处有如下：

- ❑ 通过从调用页面传递的属性进行相应的定制操作。
- ❑ 访问 JSP 页面上可以使用的所有类对象。
- ❑ 修改由调用页面生成的相应。
- ❑ 彼此通信。可以创建并初始化 JavaBean 组件、在一个标签中创建引用该 bean 的变量，再在另外一个标签中使用这个 Bean。
- ❑ 彼此嵌套，可以在 JSP 页面中实现负责的交互操作。

而所谓的自定义标签库就是多个自定义标签的集合，即一组自定义标签。下面首先介绍有关自定义标签的相关知识。

16.1.1.1　自定义标签格式

JSP 自定义标签在页面中是通过 XML 语法格式来调用的。它们统一地有一个开始标签和一个结束标签，具体形式有如下几种：

（1）简单格式的标签

一个简单格式的标签没有体，也没有任何的属性（即不需要向处理类赋值），例如：

```
<tt:simple />
```

（2）带属性的标签

自定义标签是允许带属性的，它的作用就是给相应处理类中的属性变量赋值。属性值的设置是在开始标签中进行，语法格式为 attributeName=”value”，其中 attributeName 为定义的属性，value 为该属性对应的值。类似于参数定制方法的行为一样，属性值用于定制自定义标签的行为。在标签库描述符中已经指定了标签各类属性的类型。

属性值可以是一个常量或者运行时表达式（例如 EL 表达式）。下面以下一章将要讲到的标准标签库中的<c:set>标签为例，这个标签带两个属性：

```
<c:set var="num" value="68" />
```

value 属性也可以用一个表达式（例如 EL 表达式）来设置值，例如 value=”${23+45}”。

（3）带体的标签

在自定义的开始标签和结束标签之间的体中，可以包 JSP 文本或者脚本元素等。例如：

```
<c:set var="num" >
  hello
</c:set>
```

（4）自定义标签的嵌套使用

嵌套标签的使用是指在自定义标签的体中再嵌套一个其他值定义标签，从而进行组合使用。下面通过下一章将要介绍的标准标签库中的标签为例来具体说明：

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

前面已经提及到了，开发的自定义标签库需要有以下成员组成：

- 自定义标签的处理类。
- 自定义标签库的描述文件（TLD 文件）。

下面依次介绍自定义标签的处理类和描述文件的编写方法。

16.1.1.2　自定义标签处理类

每个开发的自定义标签都需要有一个相对应的标签处理类，自定义标签的功能就是通过这些标签处理类来实现的，因此，可以说自定义标签的开发其实就是标签处理类的开发。

开发的标签处理类必须实现 Tag 或者 BodyTag 接口类（它们的包名为 javax.servlet.jsp.tagext）。BodyTag 接口是继承了 Tag 接口的子接口，如果创建的自定义标签不带体式，可以实现 Tag 接口，但是如果创建的自定义标签包含体，则需要实现 BodyTag 接口。

Tag 接口类中所定义的方法有：

- setPageContext(PageContext pc)：设置当前页面的上下文。
- setParent(Tag t)：设置这个标签处理类的父类。
- getParent()：获得父类。
- doStartTag()：处理这个实例中的开发标签。
- doEndTag()：处理这个实例中的结束标签。
- release()：由标签处理类引起，来释放状态。

BodyTag 子接口类又重新定义了两个新方法：

- setBodyContent(BodyContent b)：为体中代码作初始化。
- doInitBody()：为标签体中的内容设置属性。

JSP 页面编译成的 Servlet 在对标签进行处理的不同阶段会调用 Tag 或者 BodyTag 接口类中所定义的相应方法。当遇到自定义标签的开始标签时，JSP 页面的 Servlet 将调用相应的方法以初始化这个自定义标签所对应的处理类，然后调用处理类从接口类继承过来的 doSartTag()方法。遇到自定义标签的结束标签时，将调用处理类继承过来的 doEndTag()方法。

为了简化自定义标签的开发过程，JSP 规范又定义了另外一些辅助类来帮助开发者。例如 TagSupport 和 BodyTagSupport 类，其中 TagSupport 类实现了对应的 Tag 接口，BodyTagSupport 实现了 BodyTag 接口类，并且继承 TagSupport 类。

在接下来的实例部分，会具体介绍标签处理类的编写方法。

16.1.1.3　标签库描述文件（TLD 文件）

标签库描述符（TLD 文件）是一个描述标签库的 XML 配置文件。在 TLD 文件中，包含有关整个库以及库中包含的每一个标签的信息，它将自定义标签和对应的标签处理类关联起来。Web 容器用 TLD 验证标签，JSP 页面开发工具也要使用到 TLD。

TLD 文件名称必须扩展名为.tld。TLD 文件存储在 Web 模块的 WEB-INF 目录下或者子目录下，并且一个标签库要对应一个标签库描述文件，而在一个描述文件中可以包含多个自定义标签的声明。

下面举一个实例，标签库名为 mytag（对应的标签库描述文件名为 mytag.tld），定义了一个标签 print，并带有属性 type 和 value。描述性文件内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib 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
    h ttp://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
```

```
    <!-- 以上为描述性文件的头部 -->
    <description>mytag 1.1 print library</description>  <!-- 对这个标签库进行描述 -->
    <display-name>mytag 标签库</display-name>      <!--  可由工具显示的选择名，例如 Dreamweaver MX
工具 -->
    <tlib-version>1.0</tlib-version>                <!-- 此标签库的版本 -->
    <jsp-version>1.2></jsp-version>                 <!-- 这个标签库需要的 JSP 规范版本 -->
    <short-name>mytag</short-name>                  <!-- JSP 页面编写工具可以用来创建助记名的可选名字
-->
    <uri>http://www.tag.com/mytag</uri>             <!—唯一标识该标签库的 URI -->

    <tag>
      <description>打印页面 taglib</description>   <!-- 此标签的描述信息 -->
      <name>print</name>  <!-- 惟一标签名 -->
      <tag-class>cn. tag.mytagPrint</tag-class>     <!-- 定义这个标签对应的处理类文件 -->
      <body-content>JSP</body-content>              <!-- 体中正文内容类型 -->
      <attribute>
        <name>value</name>                          <!-- 开发标签中包括的属性 -->
        <required>true</required>                   <!-- 是否为必选属性 -->
      </attribute>
      <attribute>
        <name>type</name>
        <required>false</required>
      </attribute>
    </tag>
  </taglib>
```

创建了自定义标签（包括标签对应的处理类以及标签库的描述性文件 TLD）之后，如果想在 JSP 页面中使用这些标签，还必须做以下三件事：

（1）在 web.xml 配置文件中对标签库的引用进行声明。以上面创建的 mytag.tld 文件为例，需要在 web.xml 文件中进行配置的一段代码如下：

```
<taglib>
<taglib-uri> http://www.tag.com/mytag </taglib-uri>
<taglib-location>/WEB-INF/mytlds/mytag.tld</taglib-location>
</taglib>
```

该配置文件说明创建的 mytag.tld 文件放置在 Web 模块的/WEB-INF/mytlds/目录下。

（2）让标签库实现对 Web 应用程序可用：可以有两种方式，一种是把实现了标签处理类（tag handler）以非打包的形式存储在 web 应用程序的 WEB-INF/classes 目录或者子目录中；另一种是以 JAR 形式发布库，就是将这些标签 handler 的类打包成一个 JAR 包，然后存放在 web 应用程序的 WEB-INF/lib 目录下。如果有多个应用程序需要使用的标签，就存储在 Tomcat 服务器（这里以 Tomcat 为例）的 common/lib 主环境目录下。

（3）在 JSP 文件的头部使用 taglib 指令声明包含了标签的标签库，并定义一个前缀。例如<%@ taglib uri="http://www.tag.com/mytag" prefix="tt" %>。

其中 uri 属性表示惟一表示标签库描述符（TLD）的 URI。prefix 属性定义了区分指定标签库所定义的标签与其他标签库提供的标签的前缀。

描述性 TLD 文件必须首先编写好，并且储存在 WAR 的 WEB-INF 目录中，或者在 WEB-INF 的子目录中。

16.1.2　自定义标签与JavaBean之间的区别

JavaBean 的实现目的也是为了让逻辑处理和控制从 JSP 页面中脱离开来，从而减少 JSP 页面中嵌入的 Java 脚本，使得页面程序易于维护和代码重用。但是某些复杂 JavaBean 的调用还是需要 Web 开发者在页面程序中编写 Java 脚本。并且 JavaBean 类的方法调用还不是那么方便，它要有 Web 页面开发者对类编写和调用有所了解。而自定义标签在这方面要比 JavaBean 更加优秀，它使得 Web 开发者可以完全从 Java 编程中脱离开来，专注于页面显示和格式上面去。

总结起来自定义标签与 JavaBean 之间的区别有：

- JavaBean 并不能操作 JSP 形式的内容，而自定义标签在体中嵌套 JSP 语句，并对它进行处理。
- 自定义标签对 Web 页面开发者来说更加容易理解和操作，它有更简洁的形式。
- 但是建立自定义标签的过程要更加的复杂，所要它花费的时间更多，这也是很多开发者放弃自定义标签开发的一个原因。但是从趋势来看，自定义标签的使用将有很大的发展前景。

16.2　自定义标签的开发实例

介绍完自定义标签的基本概念、结构和原理后，下面通过多个实例来具体讲解自定义标签的开发过程。使用 Eclipse＋Lomboz 工具构架一个名为 MyTags 的 J2EE 项目，其中 Web 模块名为 tags。Web 容器仍然使用 Tomcat。

16.2.1　简单格式的标签开发

简单格式的标签没有属性和体，它必须实现 Tag 接口中的 doStartTag()和 doEndTag()方法。当 Web 容器遇到开始标签时会自动调用 doStartTag()方法。由于简单格式的标签没有体，所以这个方法会直接返回一个 SKIP_BODY。在遇到结束标签的时候会调用 doEndTag()方法。如果还需要页面中的其他部分进行判断，则 doEndTag()方法会返回 EVAL_PAGE，否则，会返回 SKIP_PAGE。

在这一小节，将要创建的简单格式的标签为<mytag:print />，没有属性，也没有体的部分，直接打印一个“Hello”字符串。下面按照步骤来创建该标签。

16.2.1.1　创建标签处理类：MytagPrint.java

此处创建的标签处理类需要继承 TagSupport 辅助类，创建在 cn.tag 包中，详细代码如下：

```
package cn.tag;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspTagException;
import javax.servlet.jsp.tagext.TagSupport;

public class MytagPrint extends TagSupport{
    private static final long serialVersionUID = 1L;        //因为父类继承了一个序列化接口，这只是一个编号
                                                            //遇到开始标签时，容器调用这个方法
    public int doStartTag() throws JspException {
        try {
                pageContext.getOut().print("Hello");    //此标签仅仅打印出“Hello”字符串
            }catch (Exception ex) {
              throw new JspTagException("SimpleTag: " + ex.getMessage());}
```

```
            return SKIP_BODY;
        }
                                                    //遇到结束标签时，调用这个方法
        public int doEndTag() {
            return EVAL_PAGE;                       //需要对页面其他部分进行判断，否则返回 SKIP_PAGE
        }
}
```

程序说明：

（1）上面创建的类继承了 TagSupport 辅助类，这里只需要把 TagSupport 辅助类中实现的 doStartTag()和 doEndTag()方法覆盖掉，就开发出了一个标签处理类。

（2）由于本实例开发的是一个简单格式的标签，没有体的部分，所以在 doStartTag()方法中返回 SKIP_BODY 变量。在 doEndTag()中返回的是 EVAL_PAGE 常量，这说明还需要对页面其他部分进行判断，否则返回 SKIP_PAGE。

（3）在 TagSupport 继承类中已经定义了一个可以被子类使用的 pageContext 实例（在讲解 JSP 隐含对象时提及到）变量，这个实例变量是由 TagSupport 类中定义的 setPageContext()方法设置的。通过 pageContext 对象获取到 out 对象，然后再通过 out 对象向页面输出字符串信息。

16.2.1.2　创建相应的 TLD 描述性文件

关于描述性文件 TLD 的编写格式，在上面已经所有介绍，一个 TLD 文件对应着一个自定义的标签库，在这个文件中可以申明一组自定义的标签。首先在 Web 模块的 WEB-INF/mytlds 目录（需要创建 mytlds 文件夹）下创建一个 mytag.tld 描述性文件。文件内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib 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-jsptaglibrary_2_0.xsd"
    version="2.0">
     <!--上面是描述性文件的头部 -->
  <description>mytag 1.1 print library</description>
  <display-name>mytag 标签库</display-name>
  <tlib-version>1.0</tlib-version>
  <short-name>mytag</short-name>
  <uri>http://www.tag.com/mytag</uri>

  <tag>
    <description>打印 Hello</description>
    <name>print</name>
    <tag-class>cn.tag.MytagPrint</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

代码说明：由于这个简单格式的标签没有体的部分，所以在<body-content>部分设置为 empty。

16.2.1.3　在 web.xml 中对自定义标签库进行引用

在 web.xml 中使用<taglib>元素来定义将要引用的标签库的路径，其中<taglib-uri>子元素指定的 URI 将在 JSP 文件头部进行声明时使用。在 web.xml 配置文件的</web-app>前面加上如下一段代码：

```
<taglib>
<taglib-uri>http://www.tag.com/mytag</taglib-uri>
```

```
<taglib-location>/WEB-INF/mytlds/mytag.tld</taglib-location>
</taglib>
```

16.2.1.4　在 JSP 文件中声明和使用自定义标签

要在 JSP 中使用自定义的标签，首先需要使用 taglib 指令来声明引用的标签库。例如，创建一个 mytag_print.jsp 文件，源代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ taglib prefix="mytag" uri="http://www.tag.com/mytag" %>
<html>
<head><title>简单标签实例</title></head>
<body>
      <h3>调用 mytag 标签库中的 print 标签</h3>
      调用 print 标签的结果：<mytag:print />
</body></html>
```

代码说明：该 JSP 文件对标签库的声明为<%@ taglib prefix="mytag" uri="http://www.tag.com/mytag" %>，其中 uri 对应着 web.xml 配置文件中的<taglib-uri>元素指定的值，prefix 属性是定义的一个前缀。在<mytag:print />中的 mytag 就是 prefix 指定的前缀。

运行这个 JSP 文件，将在页面打印出一个“Hello”字符串信息。

16.2.2　定义带有属性的标签

如果开发的标签需要带有属性，则针对每一个标签属性，都需要在标签处理类中定义一个相对应的属性以及符合 JavaBean 结构规范的 get 和 set 方法。例如，核心标准标签库中的<c:set>标签：

```
<c:set var="num" value="${45+56}" />
```

其中${45+56}为 EL 表达式，这将在下面章节中重点介绍。

针对属性 value，就需要在这个标签相对应的处理类中定义一个属性以及 set 和 get 方法，部分代码如下：

```
protected String value=null;
public void setValue(String value){
   this.value=value;
}
public String getValue(){
   return value;
}
```

下面在 mytag 标签库中再创建一个带有两个属性的 printa 标签，value 属性指定要打印出来的内容，type 属性指定输出后是否换行，“0”表示打印后不换行（默认值），“1”表示打印后换行。

16.2.2.1　创建带有属性的标签处理类

在包 cn.tag 中创建一个 MytagPrinta.java 的标签处理类，其源代码如下：

```
package cn.tag;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspTagException;
import javax.servlet.jsp.tagext.TagSupport;
public class MytagPrinta extends TagSupport{
      private static final long serialVersionUID = 1L;          //因为父类继承了一个序列化接口，这只是一个编号
      protected String value = null;
      protected String type = "0";
```

```
    public void setValue(String value){                    //value 属性对应的 set 方法
        this.value = value;
    }
    public String getValue(){                              //value 属性对应的 get 方法
        return value;
    }
    public void setType(String type){                      //type 属性对应的 set 方法
        this.type = type;
    }
    public String getType(){                               //type 属性对应的 get 方法
        return type;
    }
                                                           //容器遇到开始标签时，调用这个方法
    public int doStartTag() throws JspException {
                                                           //try…catch 语句抓获异常
        try {
                if(type.equals("0"))                       //type 值为 0，表示打印后不换行
                   pageContext.getOut().print("Hello "+value);     //此标签仅仅打印出 value 值
                else                                               //否则，打印后换行
                    pageContext.getOut().print ("Hello "+value+"<br>");          }catch (Exception ex) {
                throw new JspTagException("PrintaTag: " + ex.getMessage());}
        return SKIP_BODY;
    }
                                                           //容器遇到结束标签时，调用的方法
    public int doEndTag() {
        return EVAL_PAGE;                                  //需要对页面其他部分进行判断，否则返回 SKIP_PAGE
    }
}
```

程序说明：

（1）在这个处理类中定义了两个相对应的属性以及 set 和 get 方法。

（2）根据 type 属性值判断在页面中打印出 value 值后是否换行，如果为“0”则换行，否则不换行。

16.2.2.2　在 mytag.tld 文件中定义 printa 标签

mytag 标签库的描述性文件已经在上一部分创建了，这里需要把创建的新标签 printa 在 mytag.tld 文件中进行定义。把下面一段代码插入到 mytag.tld 文件的<taglib>元素中：

```
  <tag>
    <description>带有 type 和 value 属性的 printa 标签</description>
    <name>printa</name>
    <tag-class>cn.tag.MytagPrinta</tag-class>
    <body-content>empty</body-content>
    <attribute>
      <name>type</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>value</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
```

```
    </attribute>
  </tag>
```

代码说明；

（1）这里的标签同样没有体，所以<body-content>元素设置为 empty。

（2）标签的属性必须在<tag>元素内的<attribute>子元素进行声明：

- <name>元素指定属性的名称，并且区分大小写；
- <required>元素的设置是来规定该属性是否必需指定的，true 表示必需，false 表示该属性的设置是可选的；
- <rtexprvalue>元素指明该属性是否能够接受像<%= expression%>的 JSP 表达式或者 EL 表达式，默认值为 false，即支持。

由于 mytag 标签库已经在 web.xml 配置文件中进行了声明，所以这里就跳过这一步骤。

16.2.2.3．创建标签调用的 JSP 文件

创建一个 mytag_printa.jsp 文件，用来调用新创建的<mytag:printa/>标签，其源代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ taglib prefix="mytag" uri="http://www.tag.com/mytag" %>
<html>
<head><title>带属性标签实例</title></head>
<body>
      <h3>调用 mytag 标签库中的 printa 标签</h3>
      打印后不换行：<mytag:printa value="print 标签" type="0" />
      打印后换行：    <mytag:printa value="printa 标签" type="1" />
</body>
</html>
```

程序说明：value 属性指定需要打印的内容，type 值确定打印后是否换行。

16.2.3　自定义带有体的标签

之前定义的各标签都是不带体的，这一部分将介绍如何创建一个带体的标签。

当 Web 容器遇到一个带体的标签时，首先会把标签体中的内容按照 JSP 代码进行解析、执行和处理，然后再把它封装到一个 BodyContent 对象中。

自定义带体的标签处理类需要继承 BodyTagSupport 辅助类（这个类实现了 BodyTag 接口类），实现其中的 doInitBody()和 doAfterBody()方法，这些方法与有 JSP 页面的 Servlet 传递给标签处理类的正文内容相互交互。

- doInitBody 方法：在已经设置体中正文内容后，但是对它进行判断之前调用 doInitBody 方法。一般用这个方法执行所有依赖于正文内容的初始化。
- doAfterBody 方法：doAfterBody 方法在判断了正文内容后调用。

像 doStartTag 方法一样，doAfterBody 必须返回指明是否继续判断体中正文内容的指示。因此，如果应该再次判断正文，就像实现枚举标签的情况，那么 doAfterBody 应该返回 EVAL_BODY_BUFFERED，否则 doAfterBody 应该返回 SKIP_BODY。

另外，在 BodyTagSupport 辅助类中定义了一个 BodyContext 类型（它是 JspWriter 的子类）的变量 BodyContext。因为标签体的内容存储在这个 bodyContext 对象中，所以可以通过这个对象访问标签体，BodyContext 类可以调用的方法有：

- GetEnclosingWriter()方法：主要用来返回一个 doStartTag 和 doEndTag 可以使用的 JspWriter 对

象。

- GetReader()方法：返回一个 Reader 类型的对象，可以使用来读取标签体中的内容。
- GetString()方法：返回整个标签体中的字符串。
- ClearBody()方法：清除掉已经存储的标签体处理结果。

下面还是通过实例来讲解如何创建一个带体的自定义标签。

16.2.3.1. 创建带标签体的标签处理类

由于此处创建的自定义标签 forEach 带有体，所以必须继承 BodyTagSupport 辅助类，这个自定义标签对应的处理类 MytagforEach.java 代码如下：

```
package cn.tag;
import java.io.IOException;
import javax.servlet.jsp.JspTagException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.BodyTagSupport;
public class MytagForEach extends BodyTagSupport{
    private static final long serialVersionUID = 1L;
    protected String value = null;          //对应 forEach 标签中的 value 属性
    protected int count = 0;                //对应 forEach 标签中的 count 属性，默认为 0
    public MytagForEach(){
        super();
    }
    public void setValue(String value){   //设置 value 属性值，容器会自动调用
        this.value = value;
    }
    public String getValue(){
        return value;
    }
    public void setCount(int count){       //设置 count 属性值，容器会自动调用
        this.count = count;
    }
    public int getCount(){
        return count;
    }
    /***** 覆盖父类实现的 doStartTag()方法 *****/
    public int doStartTag() throws JspTagException{
        if(count > 0){                                  //如果循环没有结束
            return EVAL_BODY_INCLUDE;                   //表示继续进行体的处理
        }
        else{                                           //循环结束
            return SKIP_BODY;
        }
    }
    /***** 覆盖父类实现的 doEndTag()方法 *****/
   public int doEndTag() throws JspTagException{
        return EVAL_PAGE;
   }
    /***** 覆盖父类实现的 doAfterBody()方法 *****/
    public int doAfterBody() throws JspTagException{
        if(count-- >= 1){                               //循环一次，count 减一，如果 count 还大于 1，执行
```

```
                try{
                    JspWriter out = bodyContent.getEnclosingWriter();
                    out.println(bodyContent.getString()+value);
                    out.println("<br>");
                    bodyContent.clearBody();                //清空 bodyContent 对象中的内容
                }catch(IOException e){
                    System.out.println("Error in RepeatTag: "+e);
                }
                return EVAL_BODY_INCLUDE;
            }else{
                return SKIP_BODY;
            }
        }
        /***** 覆盖父类实现的 doInitBody()方法 *****/
        public void doInitBody() throws JspTagException{
            //这里不作任何初始化
        }
}
```

程序说明：

（1）doInitBody()方法是初始化标签体中的正文内容，但是这里不需要进行任何的初始化，所以这个方法暂时空实现。

（2）容器首先遇到开始标签，从而执行 doStartTag()方法，通过判断 count 是否大于 0，决定跳出整个类的操作。如果大于 0，则调用 doInitBody()方法对标签体进行初始化处理，然后调用 doAfterBody()方法（改方法每次在处理完标签体后都会被调用），使 count 减 1。如果 count 大于等于 1，则执行相应的操作，并返回 EVAL_BODY_INCLUDE，继续下一个循环。

（2）由于标签体中的正文内容是保存在 bodyContent 对象中的，所以调用 bodyContent.getString()方法来获取到正文内容。

（3）调用 bodyContent.getEnclosingWriter()方法获取到在 JSP 页面中写的操作。

16.2.3.2．定义 TLD 文件

forEach 标签的处理类创建后，需要把该标签在 mytag.tld 描述性文件中进行定义，把下面的一段代码插入到 mytag.tld 文件中：

```
  <tag>
    <description>带有标签体和属性的 forEach 实例</description>
    <name>forEach</name>
    <tag-class>cn.tag.MytagForEach</tag-class>
    <body-content>jsp</body-content>
    <attribute>
      <name>value</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>count</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

代码说明：由于该标签包含了体，所以在<body-content>元素体中定义 jsp。

16.2.3.3　在 JSP 页面中调用自定义标签

下面创建一个 forEach.jsp 页面文件来调用以上创建的带体标签，该 JSP 文件的代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ taglib prefix="mytag" uri="http://www.tag.com/mytag" %>
<html>
<head><title>带有标签体、属性的标签实例</title></head>
<body>
    <h3>调用 mytag 标签库中的 forEach 标签</h3>
    <hr>
    <mytag:forEach value="Johnson" count="5">
        Hello
    </mytag:forEach>
</body>
</html>
```

程序说明：value 属性指定插入到体中的字符串，count 设置需要循环的次数。这里 count 设置为 5，所以在页面将看到 5 次的“Hello Johnson”字符串的输出。

16.2.4　标签体中嵌套其他子标签

在下一章将要讲到的标准标签库中，有很多标签体中可以嵌套其他一些标签，例如<c:choose>。这里也将通过一个实例，来具体讲解这样的嵌套标签是如何创建的。首先创建一个 info 父标签，嵌套的 param 子标签向父标签传递地点和时间参数。

16.2.4.1　创建嵌套标签的处理类 MytagInfo 和 MytagParam

自定义标签 info 的处理类代码如下：

```
package cn.tag;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class MytagInfo extends TagSupport{
    private static final long serialVersionUID = 1L;
    protected String var = null;
    protected String address = null;
    protected String date = null;
                                                    //下面为各属性变量对应的 set 和 get 方法
    public void setVar(String var){
        this.var = var;
    }
    public String getVar(){
        return var;
    }
    public void setAddress(String address){
        this.address = address;
    }
    public void setDate(String date){
        this.date = date;
    }
```

```
                                                    //容器遇到开始标签时，调用的方法
    public int doStartTag() throws JspException{
        this.var = "时间："+date+" 地点："+address;
        return EVAL_BODY_INCLUDE;                   //表示需要执行标签体
    }
    public void release(){
        super.release();
        var = null;
    }
}
```

需要创建的 param 子标签的处理类代码如下：

```
package cn.tag;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class MytagParam extends TagSupport{
    private static final long serialVersionUID = 1L;
    protected String value = null;
    protected String type = null;                   //指定参数为时间还是地点
    public void setValue(String value){
        this.value = value;
    }
    public String getValue(){
        return value;
    }
    public void setType(String type){
        this.type = type;
    }
    public String getType(){
        return type;
    }
                                                    //容器遇到开始标签时，调用的方法
    public int doEndTag() throws JspException{
        MytagInfo mi = (MytagInfo)this.getParent();  //获得父标签
        if(type.equals("date"))                      //判断是否为时间
            mi.setDate(value);
        else if(type.equals("address"))              //判断是否为地点
            mi.setAddress(value);
        else                                         //当时间和地点都不是时
        {
            mi.setAddress("Error");
            mi.setDate("Error");
        }
        return EVAL_PAGE;
    }
}
```

程序说明：在内置的子标签处理类中，可以调用相应的 getParent()方法获得对父（外层）标签处理类的引用。这里通过子标签处理类来给父标签中的 address 和 date 参数设值，然后组合成字符串赋给 var 变量，在页面中显示。

16.2.4.2　在 mytag.tld 描述文件中添加 info 和 param 两标签的声明

把下面对 info 和 param 标签定义的语句插入到 mytag.tld 描述性文件中：

```
  <tag>
    <description>info 标签的实例</description>
    <name>info</name>
    <tag-class>cn.tag.MytagInfo</tag-class>
    <body-content>jsp</body-content>
    <attribute>
      <name>var</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>

  <tag>
    <description>param 标签</description>
    <name>param</name>
    <tag-class>cn. tag.MytagParam</tag-class>
    <body-content>empty</body-content>
    <atribute>
      <name>value</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>type</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
```

由于这两个标签是放置在 mytag 标签库中的，所以就不需要在 web.xml 文件中再定义该标签库了。

16.2.4.3　创建调用自定义标签的 JSP 页面文件

创建一个 mytag_info.jsp 页面来调用以上创建的两个自定义标签，其代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="mytag" uri="http://www.tag.com/mytag" %>
<html>
<head><title>嵌套标签实例</title></head>
<body>
      <h3>可嵌套 param 标签的 info 标签</h3>
      <hr>
      <%
        String date = new java.util.Date().toString();
      %>
      <mytag:info var="info">
            <mytag:param value="清华大学" type="address" />
            <mytag:param value="${date}" type="date" />
      </mytag:info>
</body>
</html>
```

程序说明：在该页面程序中，通过 param 标签向 info 父标签传递两个参数，一个是 address，另一个是 date，然后在页面中打印出这些信息。

之上介绍了各种类型的自定义标签开发，读者会发现这一开发过程步骤太多而且复杂，这也是为什么 Web 开发者不选择自定义标签而仍然是 JavaBean 的一个原因，下一小节将介绍一种简单标签开发的方法。

16.3　JSP2.0 新特性－简单标签处理类的开发

上一小节所介绍的标签开发能够给 Web 开发带来很多的好处，但是自定义标签开发过程相比 JavaBean 开发要复杂，开发工作涉及标签处理类和标签库描述符（TLD）文件的开发，以及标签库在 web.xml 文件中的注册。所以在实际 Web 开发过程中自定义标签的开发是很少见的。在本小节中，将向读者介绍 JSP2.0 的一个新特征：简单标签处理类的开发，它比之前自定义标签开发过程更加简单、实现接口更少。

16.3.1　简单标签简介

JSP2.0 引入一个新的用于创建自定义标签的 API：javax.servlet.jsp.tagext.SimpleTag。该 API 定义了用来实现简单标记的接口。和 JSP1.2 中的已有接口不同的是，SimpleTag 接口不使用 doStartTag()和 doEndTag()方法，而提供了一个简单的 doTag()方法。这个方法在调用该标记时只被使用一次。而需要在一个自定义标签中实现的所有逻辑过程、循环和对标签体的评估等都在这个方法中实现。

从这个方面来讲，SimpleTag 和 IterationTag 可以达到同等的作用。但 SimpleTag 的方法和处理周期要简单得多。在 SimpleTag 中还有用来设置 JSP 内容的 setJspBody()和 getJspBody()方法。Web 容器会使用 setJspBody()方法定义一个代表 JSP 内容的 JspFragment 对象。实现 SimpleTag 标记的程序可以在 doTag 方法中根据需要多次调用 getJspBody().invoke()方法以处理 JSP 内容。

简单标签处理类的开发是通过实现 SimpleTag 接口，或者来继承 SimpleTagSupport 辅助类完成的。完整的 SimpleTag 接口类定义的方法如下：

- doTag()
- setParent(JspTag parent)
- getParent();
- setJspContext(JspContent pc)
- setJspBody(JspFragment jspBody)

在这些定义的方法中，只有一个生命周期的 doTag()方法，即与该标签有关的所有逻辑、迭代或者主题赋值都在这个方法中执行。

它还提供了一个 setJspBody()方法来把标签体中的内容赋给 JspFragment 对象进行封装，然后还可以调用这个对象中的 invoke()方法来对标签体内容赋值。

其实大部分开发的简单标签处理类都是通过继承 SimpleTagSupport 类完成，该类对 SimpleTag 进行了扩展。在 SimpleTagSupport 类中另外定义的方法有：

- getJspBody()：调用这个方法返回一个 JspFragment 对象，这个对象封装了标签体中的内容。
- findAncestorWithClass(JspTag form,java.util.Class class)：该方法用于查找给定类的类型实例。

16.3.2　简单标签实例

下面通过一个简单的实例，来演示简单标签的具体开发过程，这里创建的 iterator 简单标签的作用是把标签体中的内容循环输出 10 次。

16.3.2.1　创建标签处理类 SimpleIterator.java

iterator 标签的处理类代码如下：

```
package cn.tag;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class MytagIterator extends SimpleTagSupport{

    protected int count = 0;                    //记录循环次数
    public void setCount(int count){
        this.count = count;
    }
    public int getCount(){
        return count;
    }
                                                //下面为主体调用方法
    public void doTag() throws JspException,IOException{
        for(int i=0;i<count;i++){
            getJspContext().setAttribute("count",String.valueOf(i+1));
            getJspBody().invoke(null);
        }
    }
}
```

程序说明：

（1）在 doTag()方法中首先调用了 getJspContext()方法来获得一个 JspContext 对象，该对象存储的是 JSP 页面上下文。然后调用 setAttribute()方法在 page 范围内设置 count 变量。待会儿创建页面文件的时候，会看到在标签体中使用 EL 对这个变量的调用。

（2）设置 count 参数值后，还需要调用 getJspBody()方法来获得 JspFragment 对象，然后在调用这个对象中的 invoke()方法处理标签体。invoke()方法的参数是一个 PrintWriter 对象，如果设置为 null，则表示采用默认的输出流。

16.3.2.2．在 TLD 文件中注册该标签

上述标签处理类编写后，下一步还需要在 tld 描述性文件中定义这个 iterator 标签，把下面的一段代码插入到 mytag.tld 文件中：

```
  <tag>
    <name>iterator</name>
    <tag-class>cn. tag.MytagIterator</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
      <name>count</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
```

```
</tag>
```

16.3.2.3. 调用该简单标签的 JSP 页面

最后编写一个 mytag_simple_iterator.jsp 文件来使用之上创建的标签，这个 JSP 文件的代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="mytag" uri="http://www.tag.com/mytag" %>
<html>
<head><title>简单标签实例</title></head>
<body>
      <h3>iterator 标签</h3>
      <hr>
      <mytag:iterator count="10">
            这是第${count}次循环!<br>
  </mytag:iterator>
</body>
</html>
```

程序说明：${count}EL 表达式获取到在处理类中设置的 count 变量值。运行这个 JSP 页面，将标签体中的正文依次循环 10 次，${count}表达式的值由 1 变到 10。

16.4　标签文件

标签文件也是 JSP2.0 中的一个新特性。可知进行自定义标签的开发需要开发者具有 Java 编程的知识，但是很多 Web 开发者并不太了解 Java 编程。正是针对这个问题，JSP2.0 支持了标签文件的定义。

标签文件并不像自定义标签，它是一种资源文件，Web 开发者可以使用它抽取一段 JSP 代码，并通过定制功能来实现代码的重用。即标签文件允许 Web 开发人员使用 JSP 语法来创建可重复使用的标签库。标签文件是以“.tag”作为扩展名。

下面还是以实例来直观地教会读者怎么创建一个标签文件，首先创建一个静态的标签文件并在 JSP 页面中实现对它的调用。

16.4.1　静态标签文件

顾名思义，静态标签文件就是不带定制功能，即没有参数的传递，这里创建的 mygreeting.tag 标签文件放置在 WEB-INF\tags 目录下，具体内容如下：

```
<table border="1">
  <tr>
    <td>greet:</td>
    <td>Hello, How are you these days?</td>
  <tr>
</table>
```

此标签文件只是输出一个两行的表单，并有简单的问候语。

下面创建一个 JSP 文件 mygreeting.jsp 来调用以上的标签文件，该 JSP 文件内容如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
<html>
```

```
<head><title>静态标签文件的实例</title></head>
<body>
<h4>静态标签文件的实例</h4>
<hr>
标签文件输出的内容：<br>
<tags:mygreeting />
</body>
</html>
```

程序说明：<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>是对该标签文件的引用声明。运行该 JSP 文件，页面效果如图 16.1 所示。

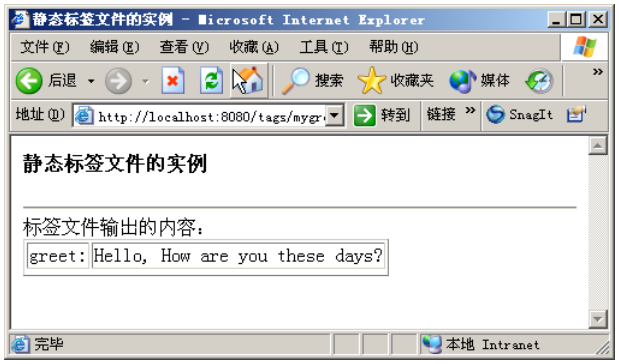

图 16.1　静态标签文件的调用

16.4.2　动态标签文件

下面给出一个动态标签文件的实例，即可以在 JSP 文件中向这个标签文件传递参数进行定制。在 Tags\WEB-INF\tags 目录下创建动态标签文件 myInfo.tag 代码如下：

```
<%@ attribute name="username" %>
<%@ attribute name="size" %>
<%@ attribute name="align" %>
<table border="1">
<tr><td align="${align}"><h4>${username}</h4></td></tr>
<tr><td aling="${align}"><font size="${size}"><jsp:doBody /> </td></tr>
</table>
```

程序说明：在这个标签文件中定义了三个属性，并使用 EL 表达式来进行引用。

下面再创建一个 myInfo.jsp 的页面文件，其源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
<html>
<head><title>动态标签文件的实例</title></head>
<body>
<h4>动态标签文件的实例</h4>
<hr>
标签文件输出的内容：<br>
<tags:myInfo username="Johnson" size="2" align="center">
      Address:BeiJing Tel:88889999
</tags:myInfo>
<tags:myInfo username="Tom" size="4" align="left">
```

```
    Address:NanJing Tel:55556666
</tags:myInfo>
</body>
</html>
```

程序说明：例如<tags:myInfo username="Johnson" size="2" align="center">，进行相应的参数设置，在标签文件中的<jsp:doBody />标记是用来读取<tags:myInfo>标签体中的内容。

运行该页面，显示的效果如图 16.2 所示。

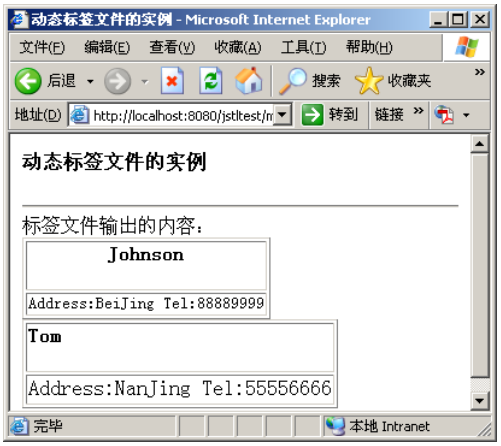

图 16.2　动态标签文件的调用

16.5　本章小结

本章向读者介绍了自定义标签的开发过程，自定义标签分为如下几种：简单格式的标签、带属性的标签、带体的标签以及嵌套标签，并一一做了介绍。自定义标签的使用可以使得标签开发人员和 Web 页面开发人员进行工作划分，页面开发人员可以专注于页面数据的显示和格式，达到面向文档的开发模式。但是自定义标签开发过程过于复杂和繁琐，所以在实际开发过程中使用很少，开发人员还是倾向于使用 JavaBean 来处理逻辑问题。为了简化自定义标签的开发过程，JSP2.0 提供了开发简单标签的接口类 SimpleTag，并且还提出了标签文件的概念，使得不懂 Java 程序的人员也可以开发出可重用的标签。

正是由于标签给 Web 开发人员带来的好处和方便，从而出现了一些标准标签库，而且已经比较成熟，下一章讲重点介绍这些标准标签库的使用。