Web应用开发
之
JSP标签技术

赵小敏

浙江工业大学计算机科学与技术学院

本节内容

- 6.1 自定义标签的开发
- 6.2 理解TLD文件
- 6.3 几种类型标签的开发

JSP标签简介

- 从JSP1.1版开始就可以在JSP页面中使用标签了，使用标签不但可以实现代码重用，而且可以使JSP代码更简洁。
- 为了进一步简化JSP的开发，在JSP 2.0的标签扩展API中又增加了SimpleTag接口和其实现类SimpleTagSupport，使用它们可以开发简单标签。

6.1 自定义标签的开发

- 6.1.1 标签扩展API
- 6.1.2 自定义标签的开发步骤
- 6.1.3 SimpleTag接口及其生命周期
- 6.1.4 SimpleTagSupport类

6.1 自定义标签的开发

- 所谓自定义的标签是用Java语言开发的程序，当其在JSP页面中使用时将执行某种动作，所以有时自定义标签又叫作自定义动作（custom action）。

- 在JSP页面中可以使用两类自定义标签。一类是简单的（simple）自定义标签，一类是传统的（classic）自定义标签。传统的自定义标签是JSP 1.1中提供的，简单的自定义标签是JSP 2.0增加的。

6.1.1 标签扩展API

- 要开发自定义标签，需要使用javax.servlet.jsp.tagext包中的接口和类，这些接口和类称为标签扩展API。
- 下图给出了简单标签扩展API的层次结构。

6.1.1 标签扩展API

- 除上面的接口和类外，标签处理类还要使用到javax.servlet.jsp包中定义的两个异常类：JspException和JspTagException类。

6.1.2 自定义标签的开发步骤

- 自定义标签的开发需要下面三步：

  1、创建标签处理类。

  2、创建标签库描述文件TLD

  3、在JSP页面中使用标签

1. 创建标签处理类

- 标签处理类（tag handler）是实现某个标签接口或继承某个标签类的实现类

- 标签处理类的示例（程序HelloTag.java），它实现了SimpleTag接口，该标签的功能是向JSP页面输出一条消息。

2. 创建标签库描述文件

- 标签库描述文件(Tag Library Descriptor，TLD) 用来定义使用标签的URI和对标签的描述，它是XML格式的文件，扩展名一般为.tld。
- TLD文件（程序mytaglib.tld ）定义了一个名为hello的标签。
- TLD文件一般存放在Web应用程序的WEB-INF目录或其子目录下。

3. 在JSP页面中使用标签

- 在JSP页面使用自定义标签，需要通过taglib指令声明自定义标签的前缀和标签库的URI，格式如下：

  <%@ taglib prefix="*prefixName*" uri="*tag library uri*" %>

- prefix属性值为标签的前缀，uri属性值为标签库的URI。在JSP的taglib指令中，前缀名称不能使用JSP的保留前缀名，它们包括jsp、jspx、java、javax、servlet、sun、sunw。
- 程序helloTag.jsp

6.1.3 SimpleTag接口及其生命周期

- SimpleTag接口中定义了简单标签的生命周期方法。
- SimpleTag接口中的方法有两个目的

① 允许在Java类和JSP之间传输信息

② 由Web容器调用来初始化SimpleTag操作。该接口共定义了5个方法：

1. SimpleTag接口的方法

- public void setJspContext(JspContext pc)：该方法由容器调用，用来设置JspContext对象，使其在标签处理类中可用。
- public void setParent(JspTag parent)：该方法由容器调用，用来设置父标签对象。
- public void setJspBody(JspFragment jspBody)：若标签带标签体，容器调用该方法将标签体内容存放到JspFragment中。
- public JspTag getParent()：返回当前标签的父标签。
- public void doTag() throws JspException, IOException：该方法是简单标签的核心方法，由容器调用完成简单标签的操作。

2. 简单标签的生命周期

- 当容器在JSP页面中遇到自定义标签时，它将加载标签处理类并创建一个实例，然后调用标签类的生命周期方法。
- 标签的生命周期有下面几个主要阶段：

- 1）调用setJspContext()
- 2）调用setParent()
- 3）调用属性的修改方法
- 4）调用setJspBody()
- 5）调用doTag()

6.1.4 SimpleTagSupport类

- SimpleTagSupport类是SimpleTag接口的实现类，它除实现了SimpleTag接口中的方法外，还提供了另外三个方法。
- protected JspContext getJspContext()：返回标签中要处理的JspContext对象。
- protected JspFragment getJspBody()：返回JspFragment对象，它存放了标签体的内容。
- public static final JspTag findAncestorWithClass(JspTag from, Class klass)：根据给定的实例和类型查找最接近的实例。该方法主要用在开发协作标签中。

6.1.4 SimpleTagSupport类

- 编写简单标签处理类通常不必实现SimpleTag接口，而是继承SimpleTagSupport类，并且仅需覆盖该类的doTag()。

```
public class HelloTag extends SimpleTagSupport{
   public void doTag() throws JspException, IOException{
      JspWriter out = getJspContext().getOut();
      out.print("<font color='blue'>Hello, A simple tag.</font><br>");
     out.print("现在时间是："+ new java.util.Date());
  }
}
```

6.2 理解TLD文件

- 6.2.1 <taglib>元素
- 6.2.2 <uri>元素
- 6.2.3 <tag>元素
- 6.2.4 <attribute>元素
- 6.2.5 <body-content>元素

6.2 理解TLD文件

- 自定义标签需要在TLD文件中声明。当在JSP页面中使用自定义标签时，容器将读取TLD文件，从中获取有关自定义标签的信息，如标签名、标签处理类名、是否是空标签以及是否有属性等。
- TLD文件的第一行是声明，它的根元素是<taglib>，该元素定义了一些子元素。

6.2.1 <taglib>元素

- <taglib>元素是TLD文件的根元素，该元素带若干属性，它们指定标签库的命名空间、版本等信息等。
- < taglib>元素的定义：

<!ELEMENT taglib (description*, display-name*, icon*, **tlib-version**, **short-name***,**uri?,** validator?, listener*, **tag*,function***) >

- 只有<tlib-version>和<short-name>元素是必须的，其他元素都是可选的。

6.2.2 <uri>元素

- <uri>元素指定在JSP页面中使用taglib指令时uri属性的值。例如，若该元素的定义如下：

```
<uri>http://www.mydomain.com/sample</uri>
```

- 则在JSP页面中taglib指令应该如下所示：

```
<%@ taglib prefix="demo"
uri="http://www.mydomain.com/sample" %>
```

- < uri>元素值是一个逻辑名称，并不与任何Web资源对应，容器使用它仅完成URI与TLD文件的映射。

6.2.2 <uri>元素

- Web应用中可以使用三种类型的URI：
- 绝对URI。例如，http://www.mydomain.com/sample和http://localhost:8080/taglibs都是绝对URI。
- 根相对URI。以“/”开头且不带协议、主机名或端口号的URI。它被解释为相对于Web应用程序文档根目录。/mytaglib和/taglib1/helloLib是根相对URI。
- 非根相对URI。不以“/”开头也不带协议、主机名或端口号的URI。它被解释为相对于当前JSP页面或相对于WEB-INF目录，这要看它在哪使用的。HelloLib 和taglib2/helloLib是非根相对URI。

6.2.2 <uri>元素

- 在TLD文件中也可以不指定<uri>元素，这时容器会尝试将taglib指令中的uri属性看作TLD文件的实际路径（以“/”开头）。
- 对HelloTag标签，如果没有在TLD文件中指定<uri>元素，在JSP页面中可以这样访问标签库：

```
<%@ taglib prefix="demo" uri="/WEB-INF/mytaglib.tld" %>
```

1. 容器如何查找TLD文件

- 容器是如何找到正确的TLD文件呢？实际上，在部署一个Web应用时，容器会自动建立一个URI与TLD之间的映射。
- 只要把TLD文件放在容器会查找的位置上，容器就会找到这个TLD，并为标签库建立一个映射。
- 容器自动查找TLD文件的位置包括：
  - 在/WEB-INF目录或其子目录中查找。
  - 在/WEB-INF/lib目录下的JAR文件中的META-INF目录或其子目录中查找。

2. 在DD文件中定义URI

- 在JSP 2.0之前，开发人员必须在DD文件（web.xml）中为URI指定其TLD文件的具体位置。然后，容器会查找web.xml文件的<taglib>元素，建立URI与TLD之间的映射。
- 例如，对于上述标签库，可以将下面代码加到web.xml文件的<web-app>元素中。

```
<jsp-config>
  <taglib>
     <taglib-uri>http://www.mydomain.com/sample
     </taglib-uri>
    <taglib-location>/WEB-INF/mytaglib.tld
    </taglib-location>
 </taglib>
<jsp-config>
```

6.2.3 〈tag〉元素

- <taglib>元素可以包含一个或多个<tag>元素，每个<tag>元素都提供了关于标签的信息，如在JSP页面中使用的标签名、标签处理类及标签的属性等。<tag>元素的DTD定义如下：

<!ELEMENT **tag** (description* **,** display-name*, icon* **，** **name**, **tag-class**,

tei-class?, **body-content**?, variable*, **attribute***, example?) >

6.2.3 〈tag〉元素

- 在一个TLD中不能定义多个同名的标签，因为容器不能解析标签处理类。
- 下面代码是非法的:

```
<tag>
    <name>hello</name>
    <tag-class>com.mytag.HelloTag</tag-class >
</tag>
<tag>
    <name>hello</name>
    <tag-class>com.mytag.WelcomeTag</tag-class>
</tag>
```

6.2.3 <tag>元素

- 可以使用一个标签处理类定义多个名称不同的标签。例如：

```
<tag>
  <name>hello</name>
  <tag-class>com.mytag.HelloTag</tag-class >
</tag>
<tag>
   <name>welcome</name>
   <tag-class>com.mytag.HelloTag</tag-class>
</tag>
```

- 在JSP页面中，假设使用demo作为前缀，则<demo:hello>和<demo:welcome>两个标签都将调用com.mytag.HelloTag类。

6.2.4 <attribute>元素

- 如果自定义标签带属性，则每个属性的信息应该在<attribute>元素中指定。下面是<attribute>元素的DTD定义：

<!ELEMENT **attribute** (description*,**name**, **required**?, **rtexprvalue**?,type?) >

- 在<attribute>元素中，只有<name>元素是必须的且只能出现一次。所有其他元素都是可选的并最多只能出现一次。

6.2.5 <body-content>元素

- <tag>的子元素<body-content>指定标签体的内容类型，在简单标签中它的值是下面三者之一：
  - empty（默认值）
  - scriptless
  - tagdependent。

1. empty

- <body-content>元素值指定为empty，表示标签不带标签体。下面的例子声明了<hello>标签并指定标签体为空。

```
<tag>
  <name>hello</name>
  <tag-class>com.mytag.HelloTag</tag-class>
  <body-content>empty</body-content>
</tag>
```

1. empty

- 对空标签，如果使用时页面作者指定了标签体，容器在转换时产生错误。下面对该标签的使用是不合法的。

```
<demo:hello>john</demo:hello>
<demo:hello><%= "john" %></demo:hello>
<demo:hello> </demo:hello >
<demo:hello>
</demo:hello>
```

2. scriptless

- <body-content>元素值指定为scriptless，表示标签体中不能包含JSP脚本元素（JSP声明<%! >、表达式<%=>和小脚本<% >），但可以包含普通模板文本、HTML、EL表达式、标准动作、甚至在该标签中嵌套其他自定义标签。下面的例子声明了<if>标签，并指定标签体中不能使用脚本。

```
<tag>
  <name>if</name>
  <tag-class>com.mytag.IfTag</tag-class>
  <body-content>scriptless</body-content>
</tag>
```

2. scriptless

- 因此，下面对<if>标签的使用是合法的：

```
<demo:if condition="true">
   <demo:hello user="john" />
    2+3 = ${2+3}
</demo:if>
```

3. tagdependent

- <body-content>元素值指定为tagdependent，表示容器不会执行标签体，而是在请求时把它传递给标签处理类，由标签处理类根据需要决定处理标签体。

```
<demo:query>
    SELECT * FROM customers
</demo:query>
```

3. tagdependent

- 对该标签，<body-content>元素值必须指定为tagdependent。

```
<tag>
  <name>query</name>
  <tag-class>com.mytag.QueryTag</tag-class>
  <body-content>tagdependent<body-content>
</tag>
```

6.3 几种类型标签的开发

- 6.3.1 空标签的开发
- 6.3.2 带属性标签的开发
- 6.3.3 带标签体的标签
- 6.3.4 迭代标签
- 6.3.5 在标签中使用EL
- 6.3.6 使用动态属性
- 6.3.7 编写协作标签

6.3.1　空标签的开发

- 空标签是不含标签体的标签，它主要向JSP发送静态信息。
- 下面是一个标签处理类的实现，它是一个空标签。当它在页面中使用时打印一个红色的星号（*）字符。
- 程序 RedStarTag.java

6. 3. 1　空标签的开发

- 下面在**TLD**文件中通过**<tag>**元素描述该标签的定义。

```
<tag>
  <name>star</name>
  <tag-class>com.mytag.RedStarTag</tag-class>
  <body-content>empty</body-content>
</tag>
```

6.3.1 空标签的开发

- 在JSP页面中访问空标签有两种写法，一种是由一对开始标签和结束标签组成，中间不含任何内容，例如：

  <prefix:tagName></prefix:tagName>

- 简化格式:在开始标签末尾使用一个斜线（/）表示标签结束，例如：

  <prefix:tagName />

- 程序register.jsp

6.3.2 带属性标签的开发

- 自定义标签可以具有属性，属性可以是必选的，也可以是可选的。
- 对必选的属性，如果没有指定值，容器在JSP页面转换时将给出错误。
- 对可选的属性，如果没有指定值，标签处理类将使用默认值。默认值依赖于标签处理类的实现。

6.3.2　带属性标签的开发

- 在JSP页面中使用带属性的自定义标签的格式如下。

```
<prefix:tagName attrib1="fixedValue"
attrib2="${elVariable}"
attrib3="<%= someJSPExpression %>"  />
```

- 属性值可以是常量或EL表达式，也可以是JSP表达式。表达式是在请求时计算的，并传递给相应的标签处理类。

6.3.2 带属性标签的开发

- 当标签接受属性时，对每个属性需要做三件重要的事情。
  - 必须在标签处理类中声明一个实例变量存放属性的值。
  - 如果属性不是必须的，则必须要么提供一个默认值，要么在代码中处理相应的null实例变量。
  - 对每个属性，必须实现适当的修改方法。
- 开发一个名为welcome的标签，它接受一个名为user的属性，它在输出中打印欢迎词。
- 程序WelcomeTag.java

6.3.2 带属性标签的开发

- 下面的<tag>元素是在TLD文件中对该标签的描述。

```
<tag>
   <name>welcome</name>
  <tag-class>com.mytag.WelcomeTag</tag-class>
   <body-content>scriptless</body-content>
   <attribute>
        <name>user</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
   </attribute>
</tag>
```

6.3.2 带属性标签的开发

- 对上述定义的<welcome>标签，若使用demo前缀，则下面的使用是合法的。

```
<demo:welcome />
<demo:welcome></demo:welcome>
<demo:welcome user="john" />
<demo:welcome user='<%=
   request.getParameter("userName") %>' />
<demo:welcome
   user="${param.userName}"></demo:hello>
```

6.3.2　带属性标签的开发

- 属性值的指定也可以使用JSP的标准动作<jsp:attribute>，通过该标签的name属性指定属性名，属性值在标签体中指定。

```
<demo:welcome>
    <jsp:attribute
    name="user">${param.userName}</jsp:attribute>
</demo:welcome>
```

- 程序　welcome.jsp

6.3.3　带标签体的标签

- 在起始标签和结束标签之间包含的内容称为标签体（body content）。
- 对于SimpleTag标签，标签体可以是文本、HTML、EL表达式等，但不能包含JSP脚本（如声明、表达式和小脚本）。
- 如果需要访问标签体，应该调用简单标签类的getJspBody()，它返回一个抽象类JspFragment对象。

6.3.3 带标签体的标签

- JspFragment类只定义了两个方法。
- public JspContext getJspContext()：返回与JspFragment有关的JspContext对象。
- public void invoke(Writer out)：执行标签体中的代码并将结果发送到Writer对象。如果将结果输出到JSP页面，参数应该为null。
- 程序 BodyTagDemo.java

6.3.3　带标签体的标签

- 由于简单标签的标签体中不能包含脚本元素，所以在TLD中应将<body-content>的值指定为scriptless或tagdependent，如下所示。

```
<tag>
    <name>dobody</name>
    <tag-class>com.mytag.BodyTagDemo</tag-class>
    <body-content>scriptless</body-content>
</tag>
```

- 程序6.9　dobody.jsp

6.3.3 带标签体的标签

- 如果希望多次执行标签体，可以在doTag()中使用循环结构，多次调用JspFragment的invoke(null)即可。
- 修改SimpleTagExample类的doTag()中的代码。

```
for(int i = 0 ; i<5 ; i++){
    getJspBody().invoke(null);
}
```

6.3.3　带标签体的标签

- 如果需要对标签体进行处理，可以将标签体内容保存到StringWriter对象中，然后将修改后的输出流对象发送到JspWrier对象。
- 下面的marker标签从标签体中查找指定的字符串，然后将其使用蓝色大字输出。
- 程序 MarkerTag.java

6.3.3 带标签体的标签

- 在TLD文件中使用下面代码定义该标签。

```
<tag>
   <name>marker</name>
   <tag-class>com.mytag.MarkerTag</tag-class>
   <body-content>scriptless</body-content>
   <attribute>
        <name>search</name>
        <required>true</required>
   </attribute>
</tag>
```

- 使用marker标签jsp程序marker.jsp

6.3.4 迭代标签

- 所谓迭代标签就是能够多次访问标签体的标签，它实现了类似于编程语言的循环的功能。
- 程序 LoopTag.java：迭代标签通过一个名为count的属性指定对标签体的迭代次数。

6.3.4 迭代标签

- 下面的<tag>元素在TLD文件中描述了该循环标签。

```
<tag>
   <name>loop</name>
   <tag-class>com.mytag.LoopTag</tag-class>
   <body-content>scriptless</body-content>
   <attribute>
        <name>count</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

6.3.4 迭代标签

- 使用loop标签的JSP页面loop.jsp。
- 标签有一个名为count的属性，它接收一个整型值，指定标签主体应该执行的次数。

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="demo" uri="http://www.mydomain.com/sample" %>
<html>
<head><title>Loop Tag Example</title></head>
<body>
<demo:loop count="3" >
   这是标签体内容!
</demo:loop>
</body></html>
```

6.3.5　在标签中使用EL

- 在标签体中还可以使用EL表达式，例如：

```
<demo:dobody>
    商品名称为:${product}。
</demo:dobody>
```

- 在标签处理类中的doTag()应该如下：

```
public void doTag() throws JspException,IOException{
    getJspContext().setAttribute("product", "华为P40手机");
    getJspBody().invoke(null);
}
```

6.3.5　在标签中使用EL

- 标签体中的EL表达式可以是一个集合（数组、List或Map）对象，在标签体中可以访问它的每个元素，这只需要在doTag()中使用循环即可，例如：

```
<table border='1'>
  <demo:dobody>
    <tr><td>${product}</td></tr>
  </demo:dobody>
</table>
```

6.3.5　在标签中使用EL

- 在标签处理类的doTag()中的代码如下：

```
public void doTag() throws JspException,IOException{
    String products[]={
            "华为P40手机", "联想笔记本电脑", "文曲星电子词典"};
    for( int i = 0; i<products.length; i++){
      getJspContext().setAttribute("product", products[i]);
      getJspBody().invoke(null);
    }
}
```

6.3.5　在标签中使用EL

- 在自定义标签的属性值中还可以使用EL表达式。
- 示例：ProductServlet从文件中读取指定商品，创建一个ArrayList<Product>对象并存储在会话作用域中，将控制重定向到showProduct.jsp页面，在JSP页面中使用<showProduct>标签显示商品信息，并为其传递productList属性。
- 程序ProductServlet.java
- 程序ProductTag.java

6.3.5　在标签中使用EL

- 在TLD文件中使用下面代码定义showProduct标签。

```
<tag>
    <name>showProduct</name>
    <tag-class>com.mytag.ProductTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
        <name>productList</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

6.3.5 在标签中使用EL

- showProduct.jsp页面使用showProduct标签显示商品信息。

```
<table border='1'>
     <tr>
       <td>商品号</td><td>商品名</td><td>价格</td><td>库存量</td>
     </tr>
    <demo:showProduct productList="${prodList}">
     <tr>
       <td>${product.prod_id}</td>
       <td>${product.pname}</td>
       <td>${product.price}</td>
       <td>${product.stock}</td>
     </tr>
    </demo:showProduct>
  </table>
```

6.3.6 使用动态属性

- 在简单标签中还可以处理动态属性。所谓动态属性（dynamic attribute），就是不需要在TLD文件中指定的属性。
- 要在简单标签中使用动态属性，标签处理类应该实现DynamicAttributes接口，该接口中只定义了一个名为setDynamicAttribute()的方法，它用来处理动态属性，格式为：

6.3.6 使用动态属性

```
public void setDynamicAttribute(
        String uri, String localName,
        Object value) throws JspException
```

- 参数uri表示属性的命名空间，如果属于默认命名空间，其值为null；参数localName表示要设置的动态属性名；value表示属性值。当标签声明允许接受动态属性，而传递的属性又没有在TLD中声明时将调用该方法。

6.3.6 使用动态属性

- 程序MathTag.java定义了一个带动态属性的标签处理类。在该类中创建了一个String对象output，对每个动态属性它将被setDynamicAttribute()更新。一旦结束读取属性，它将调用doTag()，把该String对象发送给JSP显示。

6.3.6 使用动态属性

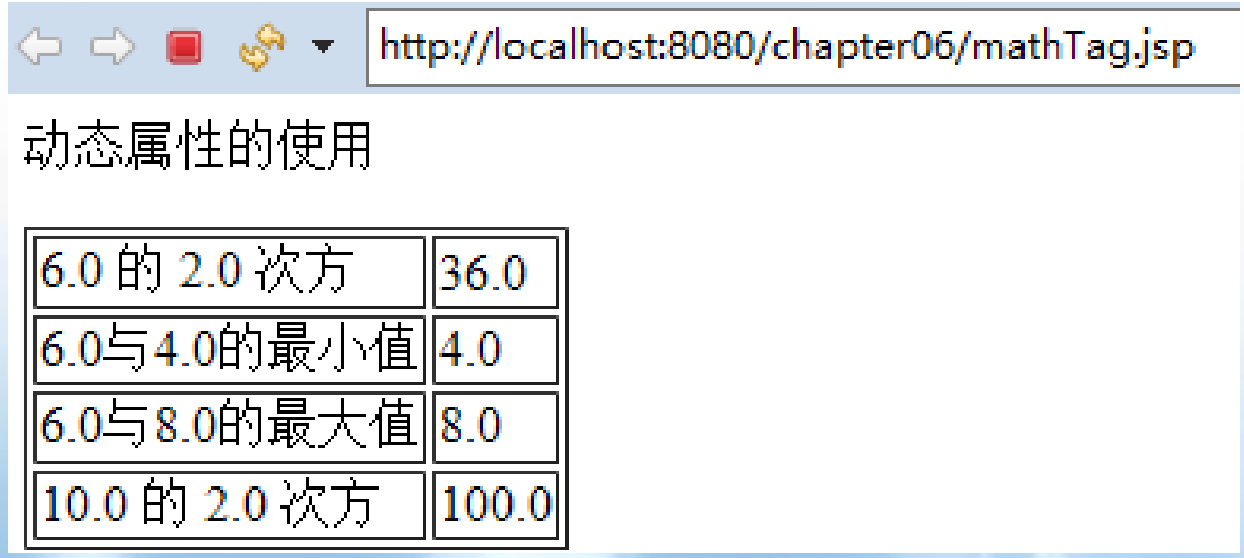
- 在TLD件的<tag>标签中，动态属性需要使用<dynamic-attributes>元素定义并将其值指定为true，如下所示。

```
<tag>
  <name>mathtag</name>
   <tag-class>com.mytag.MathTag</tag-class>
   <body-content>empty</body-content>
   <attribute>
        <name>num</name>
       <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   <dynamic-attributes>true</dynamic-attributes>
</tag>
```

6.3.6 使用动态属性

- mathTag.jsp给出了如何在**JSP**中使用该标签。

```
<p>动态属性的使用</p>
<table border="1">
  <demo:mathtag num="6" pow="2" min="4" max="8"/>
  <demo:mathtag num="${5*2}" pow="2" />
</table>
```



6.3.7 编写协作标签

- 在标签的设计和开发中，通常一组标签协同工作，这些标签称为协作标签（cooperative tags）。
- 协作标签的一个最简单的例子是实现类似于Java编程语言提供的switch-case功能
- 程序switchTag.jsp使用了下面三个标签：<switch>、<case>和<default>。

6.3.7 编写协作标签

- 程序SwitchTag.java
- 程序CaseTag.java
- 程序DefaultTag.java

小　结

- 自定义标签可以改善业务逻辑和表示逻辑的分离。自定义标签包括传统的自定义标签和简单的自定义标签。
- 自定义标签的开发首先需要创建标签处理类，该类封装了标签要实现的业务逻辑。自定义标签的类型包括：带或不带主体内容的标签、带属性的标签、迭代标签和嵌套标签等。

小　结

- JSP 2.0的一个主要目标是简化JSP的开发。为了减少标签处理类中的代码，开发了SimpleTag接口。
- 对于SimpleTag来说，在Web容器执行初始化后，它只需调用一个doTag()，它执行简单标签的所有处理功能。SimpleTagSupport类是SimpleTag接口的一个实现类。

练习

1、简单标签的TLD文件的<body-content>元素内容，下面哪个是不合法的？（　　）

A. JSP　　B. scriptless

C. tagdependent　　D. empty

2、JspContext.getOut()返回的是哪一种对象类型？（　　）

A. ServletOutputStream　　B. PrintWriter

C. BodyContent　　D. JspWriter

练习

3、考虑下面的Web应用程序部署描述文件中的<taglib> 元素：

<taglib>

<taglib-uri>/accounting</taglib-uri>

<taglib-location>/WEB-INF/tlds/SmartAccount.tld</taglib-location>

</taglib>

下面在JSP页面中哪个正确指定了上述标签库的使用?（　）

A. <%@ taglib uri="/accounting" prefix="acc"%>

B. <%@ taglib uri="/acc" prefix="/accounting"%>

C. <%@ taglib name="/accounting" prefix="acc"%>

D. <%@ taglib library="/accounting" prefix="acc"%>

练习

- 4、一个标签库有一个名为printReport的标签，该标签可以接受一个名为department的属性，它不能接受动态值。下面哪两个是该标签的正确使用?（　）

A. <mylib:printReport/>

B. <mylib:printReport department="finance"/>

C. <mylib:printReport attribute="department" value="finance"/>

D. <mylib:printReport attribute="department" attribute-value="finance"/>

E. <mylib:printReport>

<jsp:attribute name="department" value="finance" />

</mylib:printReport>

练习

- 5、下面哪个是将一个标签嵌套在另一个标签中的正确用法?（　）

A.
```
<greet:hello>
    <greet:world>
    </greet:hello>
</greet:world>
```

B.
```
<greet:hello>
    <greet:world>
    </greet:world>
</greet:hello>
```

C.
```
<greet:hello
    <greet:world/>
/>
```

D.
```
<greet:hello>
    </greet:hello>
    <greet:world>
</greet:world>
```

练习

下面有三个文件分别是JSP页面、标签处理类和TLD文件的部分代码，根据已有内容在划线中填上正确的内容，并请指出它们之间的关系。

标签处理类LoginTagHandler.java部分代码如下：

```
public class LoginTagHandler{
    public void doTag(){
        // 标签逻辑
   }
   public void setUser(String user){
        this.user = user;
   }
}
```

TLD文件的部分代码如下：

```
<taglib …>
<uri>randomthings</uri>
<tag>
        <name>advice</name>
        <tag-class>foo.LoginTagHandler</tag-class>
        <body-content>empty</body-content>
        <attribute>
<name>_____</name>
<required>true</required>
<rtexprvalue>_____ </rtexprvalue>
</attribute>
</tag>
</taglib>
```

JSP页面代码如下：

```
<html><body>
<%@ taglib prefix="____"  uri="_____"%>
Login page<br>
<mine: _____ user = "${foo}" />
</body><html>
```

作业

- 1、开发一个带属性的标签类MathTag，该标签类有一个double型属性x，标签功能是求x的平方根，用math.jsp访问该标签，输出10、100、200的平方根值。
- 2、用自定义带标签体的标签类MarkTag实现如下功能：在一个查询页面search.jsp输入关键字查询存在某文章（article.txt）中是否存在，如果存在则在查询结果result.jsp显示该文章，并对所有关键字标红显示。