# 自定义标签

## 简单的Demo

简单的Demo使用自定义标签打印客户机的IP地址

### 自定义标签类:ViewIpTag

1. **package** com.weijia.traditionaltag;
3. **import** java.io.IOException;
5. **import** javax.servlet.http.HttpServletRequest;
6. **import** javax.servlet.jsp.JspException;
7. **import** javax.servlet.jsp.JspWriter;
8. **import** javax.servlet.jsp.tagext.TagSupport;
10. /\*\*
11. \* 自定义标签，然后将这个标签映射到这个类:mytag:viewIP
12. \* 记得将自定义的标签绑定到一个url上面，这个url一般是公司的网址
13. \*
14. \*/
15. **public** **class** ViewIpTag **extends** TagSupport{
17. **private** **static** **final** **long** serialVersionUID = 1L;
19. @Override
20. **public** **int** doStartTag() **throws** JspException {
21. //内置一个pageContext对象，我们之前说到pageContext对象，它里面是封装了9个隐式对象
22. HttpServletRequest request = (HttpServletRequest)**this**.pageContext.getRequest();
23. JspWriter out = **this**.pageContext.getOut();
24. String ip = request.getRemoteAddr();
25. **try** {
26. out.print(ip);
27. } **catch** (IOException e) {
28. **throw** **new** RuntimeException(e);
29. }
30. **return** **super**.doStartTag();
31. }
33. }

### 自定义tld文件:mytag.tld

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
3. **<taglib** xmlns="http://java.sun.com/xml/ns/j2ee"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-jsptaglibrary\_2\_0.xsd"
6. version="2.0"**>**
8. **<description>**JSTL 1.1 core library**</description>**
9. **<display-name>**JSTL core**</display-name>**
10. **<tlib-version>**1.1**</tlib-version>**
11. **<short-name>**weijia**</short-name>**
12. **<uri>**http://www.weijia.cn/mytag**</uri>**
14. <!-- 显示IP地址 -->
15. **<tag>**
16. **<description>**
17. Catches any Throwable that occurs in its body and optionally
18. exposes it.
19. **</description>**
20. **<name>**viewIP**</name>**
21. **<tag-class>**com.weijia.traditionaltag.ViewIpTag**</tag-class>**
22. **<body-content>**empty**</body-content>**
23. **</tag>**
24. **</taglib>**

注:

<short-name>这个标签是指定我们定义标签的简称,这个作用不大

<uri>这个标签是给这个标签文件指定一个访问路径,这个路径我们在jsp页面中引入这个标签的时候需要用到

<tag-class>这个标签就是指定我们自定义的标签类的全称

<body-content>这个标签表明自定义标签是否有标签体内容(empty:没有,JSP:有)

### 自定义标签使用:Jsp页面指令

1. **<**%@ taglib uri="http://www.weijia.cn/mytag" prefix="mytag" %**>**
2. 客户机的IP地址是:**<mytag:viewIP/>**

## 2.自定义标签执行原理

JSP引擎将遇到的自定义标签是,首先创建标签处理器类的实例对象,然后按照JSP规范定义的通信规则依次调用它的方法.

1. public void setPageContext(PageContext pc),JSP引擎实例化标签处理器后,将调用setPageContext方法将JSP页面的pageContext对象传递给标签处理器,标签处理器以后可以通过这个pageContext对象与JSP页面进行通信.
2. public void setParent(Tag t),setPageContext方法执行完后,WEB容器接着调用的setParent方法将当前标签的父标签传递给当前标签处理器,如果当前标签没有父标签,则传递给setParent方法的参数值为null.
3. Public int doStartTag(),调用了setPageContext方法和setParent方法之后,WEB容器执行到自定义标签的开始标记时,就会调用标签处理器的doStartTag方法.
4. Public int doEndTag(),WEB容器执行完自定义标签的标签体后,就会接着去执行自定义标签的结束标记,此时,WEB容器会去调用标签处理器的doEndTag方法.
5. Public void release(),通常WEB容器执行完自定义标签后,标签处理器会驻留在内存中,为其它请求服务器,直至停止web应用时,web容器才会调用release方法.

## 自定义标签实现特定的功能

### 3.1 不执行标签体内容

自定义标签类:

只要doStartTag方法返回TagSupport.EVAL\_BODY\_INCLUDE常量,就会执行标签体内容,如果返回的是TagSupport.SKIP\_BODY常量,就不会执行标签体内容

1. **package** com.weijia.traditionaltag;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.tagext.TagSupport;
6. /\*\*
7. \* 是否输出标签体内容
8. \* @author weijiang204321
9. \*
10. \*/
11. **public** **class** TagDemo1 **extends** TagSupport{
13. @Override
14. **public** **int** doStartTag() **throws** JspException {
15. **return** TagSupport.EVAL\_BODY\_INCLUDE;//输出标签体内容
16. //return TagSupport.SKIP\_BODY;//不输出标签体内容
17. }
19. }

自定义tld文件:

1. <!-- 是否显示标签体 -->
2. **<tag>**
3. **<description>**
4. Catches any Throwable that occurs in its body and optionally
5. exposes it.
6. **</description>**
7. **<name>**demo1**</name>**
8. **<tag-class>**com.weijia.traditionaltag.TagDemo1**</tag-class>**
9. **<body-content>**JSP**</body-content>**
10. **</tag>**

在JSP页面中使用:

1. <!-- 不执行标签体 -->
2. **<simpletag:demo1>**
3. aaaa
4. **</simpletag:demo1>**

### 3.2 控制JSP余下页面的内容不执行

自定义标签类:

当doEndTag方法返回的是TagSupprot.EVAL\_PAGE常量的话就执行jsp余下的内容,如果返回的是TagSupport.SKIP\_PAGE常量的话就不执行jsp余下的内容

1. **package** com.weijia.traditionaltag;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.tagext.TagSupport;
6. /\*\*
7. \* 控制整个JSP是否输出
8. \* @author weijiang204321
9. \*
10. \*/
11. **public** **class** TagDemo2 **extends** TagSupport{
13. @Override
14. **public** **int** doStartTag() **throws** JspException {
15. **return** **super**.doStartTag();
16. }
18. @Override
19. **public** **int** doEndTag() **throws** JspException {
20. **return** TagSupport.EVAL\_PAGE;
21. //return TagSupport.SKIP\_PAGE;不执行余下的jsp内容
22. }
24. }

自定义tld文件:

1. <!-- 控制是否显示jsp页面 -->
2. **<tag>**
3. **<description>**
4. Catches any Throwable that occurs in its body and optionally
5. exposes it.
6. **</description>**
7. **<name>**demo2**</name>**
8. **<tag-class>**com.weijia.traditionaltag.TagDemo2**</tag-class>**
9. **<body-content>**empty**</body-content>**
10. **</tag>**

在jsp页面中使用:

1. <!-- 不执行余下的页面内容 -->
2. **<simpletag:demo2/>**

这个标签使用之后,标签之后的内容就不会执行了,如果我们将这个标签放在页面的第一行,那么这个页面就是一片空白,在浏览器中查看页面的源代码,也是一片空白,因为out对象没有进行print了

### 3.3 重复执行标签体内容

自定义标签体类:

这里我们需要在doAfterBody方法中操作了,因为这个方法的返回值为TagSupport.EVAL\_BODY\_AGAIN常量的话,这个方法还会被调用,直到这个方法返回TagSupport.SKIP\_BODY,所以我们这里控制标签体内容执行5次,我们定义一个变量就可以了,然后控制doAfterBody方法的返回值,这里还要注意的是,在doStartTag方法中返回值是TagSupport.EVAL\_BODY\_INCLUDE常量,因为要执行标签体内容.

1. **package** com.weijia.traditionaltag;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.tagext.TagSupport;
6. /\*\*
7. \* 控制标签体重复执行
8. \* @author weijiang204321
9. \*
10. \*/
11. **public** **class** TagDemo3 **extends** TagSupport{
13. **private** **int** count = 5;
15. @Override
16. **public** **int** doStartTag() **throws** JspException {
17. **return** TagSupport.EVAL\_BODY\_INCLUDE;
18. }
20. @Override
21. **public** **int** doAfterBody() **throws** JspException {
22. count--;
23. **if**(count > 0){
24. **return** TagSupport.EVAL\_BODY\_AGAIN;//执行完之后接着执行doAfterBody()方法
25. }**else**{
26. **return** TagSupport.SKIP\_BODY;
27. }
28. }
30. @Override
31. **public** **int** doEndTag() **throws** JspException {
32. **return** TagSupport.SKIP\_BODY;
33. }
35. }

自定义tld文件:

1. <!-- 控制标签体重复输出 -->
2. **<tag>**
3. **<description>**
4. Catches any Throwable that occurs in its body and optionally
5. exposes it.
6. **</description>**
7. **<name>**demo3**</name>**
8. **<tag-class>**com.weijia.traditionaltag.TagDemo3**</tag-class>**
9. **<body-content>**JSP**</body-content>**
10. **</tag>**

在jsp页面中使用:

1. <!-- 重复执行标签体内容 -->
2. **<simpletag:demo3>**
3. aaaa
4. **</simpletag:demo3>**

### 3.4 修改标签体内容

自定义标签类:

这里我们要注意的是,我们继承的是BodyTagSupport类了,要在doStartTag方法中返回BodyTagSupport.EVAL\_BODY\_BUFFERED常量,才可以取出标签体内容缓存,然后在doEndTag方法中取出标签体内容然后进行操作之后再写到浏览器中

1. **package** com.weijia.traditionaltag;
3. **import** java.io.IOException;
5. **import** javax.servlet.jsp.JspException;
6. **import** javax.servlet.jsp.tagext.BodyContent;
7. **import** javax.servlet.jsp.tagext.BodyTagSupport;
9. /\*\*
10. \* 修改标签体内容
11. \* @author weijiang204321
12. \*
13. \*/
14. **public** **class** TagDemo4 **extends** BodyTagSupport{
16. @Override
17. **public** **int** doEndTag() **throws** JspException {
18. BodyContent bc = **this**.getBodyContent();//获取标签体内容对象
19. String content = bc.getString();
20. content = content.toUpperCase();//将标签体内容转成大写
21. **try** {
22. **this**.pageContext.getOut().write(content);//在将转化之后的内容输出到浏览器中
23. } **catch** (IOException e) {
24. **throw** **new** RuntimeException(e);
25. }
26. **return** BodyTagSupport.EVAL\_BODY\_INCLUDE;
27. }
29. @Override
30. **public** **int** doStartTag() **throws** JspException {
31. **return** BodyTagSupport.EVAL\_BODY\_BUFFERED;//这里返回缓存标签体内容常量
32. }
34. }

自定义tld文件:

1. <!-- 修改标签体内容 -->
2. **<tag>**
3. **<description>**
4. Catches any Throwable that occurs in its body and optionally
5. exposes it.
6. **</description>**
7. **<name>**demo4**</name>**
8. **<tag-class>**com.weijia.traditionaltag.TagDemo4**</tag-class>**
9. **<body-content>**JSP**</body-content>**
10. **</tag>**

在jsp页面中使用:

1. <!-- 修改标签体内容 -->
2. **<simpletag:demo4>**
3. bbbb
4. **</simpletag:demo4>**

这时候在浏览器中输出的是：BBBB

# SimpleTagSupport

SimpleTag接口定义方法:

setJSPContext方法------用于把jsp页面的pageContext对象传递给标签处理器对象

setParent方法------用于把父标签处理器对象传递给当前标签处理器对象

getParent方法------用于获得当前标签的父标签处理器对象

setJSPBody方法-------用于把代表标签体的JspFragment对象传递给标签处理器对象

doTag方法-------用于完成所有的标签逻辑,包括输出,迭代,修改标签体内容等.

在doTag方法中可以抛出javax.servlet.jsp.SkipPageException异常,用于通知WEB容器不再执行JSP页面中位于结束标记后面的内容,这等效于在传统标签的doEndTag方法中返回Tag.SKIP\_PAGE常量的情况.