**JSP**

**1.基础语法**

1）中文编码问题

如果我们要在页面正常显示中文，我们需要在 JSP 文件头部添加以下代码：<>

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

2）表达式：<%= 表达式 %>

注释：<%-- 该部分注释在网页中不会被显示--%>

**3）JSP指令**

指令 描述

<%@ page ... %> 定义页面的依赖属性

<%@ include ... %> 包含其他文件

<%@ taglib ... %> 引入标签库的定义，可以是自定义标签

**4)jsp隐含对象**

**对象 描述**

request HttpServletRequest类的实例

response HttpServletResponse类的实例

out PrintWriter类的实例，用于把结果输出至网页上

session HttpSession类的实例

application ServletContext类的实例，与应用上下文有关

config ServletConfig类的实例

pageContext PageContext类的实例，提供对JSP页面所有对象以及命名空间的访问

page 类似于Java类中的this关键字

Exception Exception类的对象，代表发生错误的JSP页面中对应的异常对象

**JSTL标签介绍**

**1、使用taglib指令导入标签库：**

1. **<**%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %**>**

* prefix="c"：指定标签库的前缀，这个前缀可以随便给值，但大家都会在使用core标签库时指定前缀为c；
* uri="http://java.sun.com/jstl/core"：指定标签库的uri，它不一定是真实存在的网址，但它可以让JSP找到标签库的描述文件；

**4、core标签库常用标签：**

**（1）out和set标签**

|  |  |
| --- | --- |
| <c:out value=”aaa”/> | 输出aaa字符串常量 |
| <c:out value=”${aaa}”/> | 与${aaa}相同 |
| <c:out value=”${aaa}” default=”xxx”/> | 当${aaa}不存在时，输出xxx字符串 |
| <%  request.setAttribute("a","<script>alert('hello');</script>");  %>  <c:out value="${a }" default="xxx" escapeXml="false" /> | 当escapeXml为false，不会转换“<”、“>”。这可能会受到[**JavaScript**](http://lib.csdn.net/base/javascript)攻击。 |

|  |  |
| --- | --- |
| <c:set var=”a” value=”hello”/> | 在pageContext中添加name为a，value为hello的数据。 |
| <c:set var=”a” value=”hello” scope=”session”/> | 在session中添加name为a，value为hello的数据。 |

**（2）remove标签**

|  |  |
| --- | --- |
| <%                     pageContext.setAttribute("a","pageContext");                     request.setAttribute("a","session");                     session.setAttribute("a","session");                     application.setAttribute("a","application");    %>      <c: remove var="a"/>      <c: out value="${a}" default="none"/> | 删除所有域中name为a的数据！ |
| <c:remove var="a" scope=”page”/> | 删除pageContext中name为a的数据 |

**（3）url标签：该标签会在需要重写URL时添加。**

|  |  |
| --- | --- |
| <c:url value="/"/> | 输出上下文路径：/项目名/ |
| <c:url value="/" var="a" scope="request"/> | 把本该输出的结果赋给变量a。范围为request |
| <c:url value="/AServlet"/> | 输出：/项目名/AServlet |
| <c:url value="/AServlet">  <c:param name="username" value="abc"/>  <c:param name="password" value="123"/> | 输出：/项目名/AServlet?username=abc&password=123       如果参数中包含中文，那么会自动使用URL编码！ |

**（4）if标签：**

if标签的test属性必须是一个boolean类型的值，如果test的值为true，那么执行if标签的内容，否则不执行。

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<c:set** var="a" value="hello"**/>**
2. **<c:if** test="${not empty a }"**>**
3. **<c:out** value="${a }"**/>**
4. **</c:if>**

**（5）choose标签：**

choose标签对应[**Java**](http://lib.csdn.net/base/java)中的if/else if/else结构。when标签的test为true时，会执行这个when的内容。当所有when标签的test都为false时，才会执行otherwise标签的内容。

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<c:set** var="score" value="${param.score }"**/>**
2. **<c:choose>**
3. **<c:when** test="${score > 100 || score < 0}"**>**错误的分数：${score }**</c:when>**
4. **<c:when** test="${score >= 90 }"**>**A级**</c:when>**
5. **<c:when** test="${score >= 80 }"**>**B级**</c:when>**
6. **<c:when** test="${score >= 70 }"**>**C级**</c:when>**
7. **<c:when** test="${score >= 60 }"**>**D级**</c:when>**
8. **<c:otherwise>**E级**</c:otherwise>**
9. **</c:choose>**

**（6）forEach标签：**

forEach当前就是循环标签了，forEach标签有多种两种使用方式：

* 使用循环变量，指定开始和结束值，类似for(int i = 1; i <= 10; i++) {}；
* 循环遍历集合，类似for(Object o : 集合)；

循环变量：

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<c:set** var="sum" value="0" **/>**
2. **<c:forEach** var="i" begin="1" end="10"**>**
3. **<c:set** var="sum" value="${sum + i}" **/>**
4. **</c:forEach>**
5. **<c:out** value="sum = ${sum }"**/>**
6. **<c:set** var="sum" value="0" **/>**
7. **<c:forEach** var="i" begin="1" end="10" step ="2"**>**
8. **<c:set** var="sum" value="${sum + i}" **/>**
9. **</c:forEach>**
10. **<c:out** value="sum = ${sum }"**/>**

遍历集合或数组方式：

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%
2. String[] names = {"zhangSan", "liSi", "wangWu", "zhaoLiu"};
3. pageContext.setAttribute("ns", names);
4. %**>**
5. **<c:forEach** var="item" items="${ns }"**>**
6. **<c:out** value="name: ${item }"**/><br/>**
7. **</c:forEach>**

遍历List：

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%
2. List**<String>** names = new ArrayList**<String>**();
3. names.add("zhangSan");
4. names.add("liSi");
5. names.add("wangWu");
6. names.add("zhaoLiu");
7. pageContext.setAttribute("ns", names);
8. %**>**
9. **<c:forEach** var="item" items="${ns }"**>**
10. **<c:out** value="name: ${item }"**/><br/>**
11. **</c:forEach>**

遍历Map：

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%
2. Map**<String**,String**>** stu = new LinkedHashMap**<String**,String**>**();
3. stu.put("number", "N\_1001");
4. stu.put("name", "zhangSan");
5. stu.put("age", "23");
6. stu.put("sex", "male");
7. pageContext.setAttribute("stu", stu);
8. %**>**
9. **<c:forEach** var="item" items="${stu }"**>**
10. **<c:out** value="${item.key }: ${item.value }"**/><br/>**
11. **</c:forEach>**

forEach标签还有一个属性：varStatus，这个属性用来指定接收“循环状态”的变量名，例如：<forEach varStatus=”vs” …/>，这时就可以使用vs这个变量来获取循环的状态了。

* count：int类型，当前以遍历元素的个数；
* index：int类型，当前元素的下标；
* first：boolean类型，是否为第一个元素；
* last：boolean类型，是否为最后一个元素；
* current：Object类型，表示当前项目。

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<c:forEach** var="item" items="${ns }" varStatus="vs"**>**
2. **<c:if** test="${vs.first }"**>**第一行：**</c:if>**
3. **<c:if** test="${vs.last }"**>**最后一行：**</c:if>**
4. **<c:out** value="第${vs.count }行: "**/>**
5. **<c:out** value="[${vs.index }]: "**/>**
6. **<c:out** value="name: ${vs.current }"**/><br/>**
7. **</c:forEach>**

**5、fmt标签库常用标签：**

      fmt标签库是用来格式化输出的，通常需要格式化的有时间和数字。

**格式化时间：**

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %**>**
2. ......
3. **<**%
4. Date date = new Date();
5. pageContext.setAttribute("d", date);
6. %**>**
7. **<fmt:formatDate** value="${d }" pattern="yyyy-MM-dd HH:mm:ss"**/>**

**格式化数字：**

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%
2. double d1 = 3.5;
3. double d2 = 4.4;
4. pageContext.setAttribute("d1", d1);
5. pageContext.setAttribute("d2", d2);
6. %**>**
7. **<fmt:formatNumber** value="${d1 }" pattern="0.00"**/><br/>**
8. **<fmt:formatNumber** value="${d2 }" pattern="#.##"**/>**

**二、自定义标签**

**1、自定义标签**

**1.1步骤：**

其实我们在JSP页面中使用标签就等于调用某个对象的某个方法一样，例如：<c:if test=””>，这就是在调用对象的方法一样。自定义标签其实就是自定义类一样！

* 定义标签处理类：必须是Tag或SimpleTag的实现类；
* 编写标签库描述符文件（TLD）；

  SimpleTag接口是JSP2.0中新给出的接口，用来简化自定义标签，所以现在我们基本上都是使用SimpleTag。

Tag是老的，传统的自定义标签时使用的接口，现在不建议使用它了。

**1.2　SimpleTag接口介绍：**

SimpleTag接口内容如下：

* void doTag()：标签执行方法；
* JspTag getParent()：获取父标签；
* void setParent(JspTag parent)：设置父标签
* void setJspContext(JspContext context)：设置PageContext
* void setJspBody(JspFragment jspBody)：设置标签体对象；

请记住，万物皆对象！在JSP页面中的标签也是对象！你可以通过查看JSP的源码，清楚的知道，所有标签都会变成对象的方法调用。标签对应的类我们称之为“标签处理类”！

**标签的生命周期：**

**1、当容器（Tomcat）第一次执行到某个标签时，会创建标签处理类的实例；**

**2、然后调用setJspContext(JspContext)方法，把当前JSP页面的pageContext对象传递给这个方法；**

**3、如果当前标签有父标签，那么使用父标签的标签处理类对象调用setParent(JspTag)方法；**

**4、如果标签有标签体，那么把标签体转换成JspFragment对象，然后调用setJspBody()方法；**

**5、每次执行标签时，都调用doTag()方法，它是标签处理方法**。

HelloTag.java

**[java]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **public** **class** HelloTag **implements** SimpleTag {
2. **private** JspTag parent;
3. **private** PageContext pageContext;
4. **private** JspFragment jspBody;
6. **public** **void** doTag() **throws** JspException, IOException {
7. pageContext.getOut().print("Hello Tag!!!");
8. }
9. **public** **void** setParent(JspTag parent) {
10. **this**.parent = parent;
11. }
12. **public** JspTag getParent() {
13. **return** **this**.parent;
14. }
15. **public** **void** setJspContext(JspContext pc) {
16. **this**.pageContext = (PageContext) pc;
17. }
18. **public** **void** setJspBody(JspFragment jspBody) {
19. **this**.jspBody = jspBody;
20. }
21. }

**1.3　标签库描述文件（TLD）**

标签库描述文件是用来描述当前标签库中的标签的！标签库描述文件的扩展名为tld，你可以把它放到WEB-INF下，这样就不会被客户端直接访问到了。

hello.tld

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<taglib** version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"
3. xmlns:xml="http://www.w3.org/XML/1998/namespace"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6. http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd "**>**
8. **<tlib-version>**1.0**</tlib-version>**
9. **<short-name>**ywq**</short-name>**
10. **<uri>**http://www.ywq.cn/tags**</uri>**
11. **<tag>**
12. **<name>**hello**</name>**
13. **<tag-class>**cn.ywq.tag.HelloTag**</tag-class>**
14. **<body-content>**empty**</body-content>**
15. **</tag>**
16. **</taglib>**

**1.4　使用标签**

在页面中使用标签分为两步：

* 使用taglib导入标签库；
* 使用标签；

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%@ taglib prefix="it" uri="/WEB-INF/hello.tld" %**>**
2. ......
3. **<it:hello/>**

**2、自定义标签进阶**

**2.1　继承SimpleTagSupport**

　　继承SimpleTagSuppport要比实现SimpleTag接口方便太多了，现在你只需要重写doTag()方法，其他方法都已经被SimpleTagSuppport完成了。

**[java]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **public** **class** HelloTag **extends** SimpleTagSupport {
2. **public** **void** doTag() **throws** JspException, IOException {
3. **this**.getJspContext().getOut().write("<p>Hello SimpleTag!</p>");
4. }
5. }

**2.2　有标签体的标签**

我们先来看看标签体内容的可选值：

<body-content>元素的可选值有：

* empty：无标签体。
* JSP：传统标签支持它，**SimpleTag已经不再支持使用<body-content>JSP</body-content>。**标签体内容可以是任何东西：EL、JSTL、<%=%>、<%%>，以及html；
* scriptless：标签体内容不能是Java脚本，但可以是EL、JSTL等。在SimpleTag中，**如果需要有标签体，那么就使用该选项**；
* tagdependent：标签体内容不做运算，由标签处理类自行处理，无论标签体内容是EL、JSP、JSTL，都不会做运算。**这个选项几乎没有人会使用！**

自定义有标签体的标签需要：

* 获取标签体对象：JspFragment jspBody = getJspBody();；
* 把标签体内容输出到页面：jspBody.invoke(null)；
* tld中指定标签内容类型：scriptless。

**[java]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **public** **class** HelloTag **extends** SimpleTagSupport {
2. **public** **void** doTag() **throws** JspException, IOException {
3. PageContext pc = (PageContext) **this**.getJspContext();
4. HttpServletRequest req = (HttpServletRequest) pc.getRequest();
5. String s = req.getParameter("exec");
6. **if**(s != **null** && s.endsWith("true")) {
7. JspFragment body = **this**.getJspBody();
8. body.invoke(**null**);
9. }
10. }
11. }

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<tag>**
2. **<name>**hello**</name>**
3. **<tag-class>**cn.ywq.tags.HelloTag**</tag-class>**
4. **<body-content>**scriptless**</body-content>**
5. **</tag>**

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<itcast:hello>**
2. **<h1>**哈哈哈~**</h1>**
3. **</itcast:hello>**

**2.3　不执行标签下面的页面内容**

　　如果希望在执行了自定义标签后，不再执行JSP页面下面的东西，那么就需要在doTag()方法中使用SkipPageException。

**[java]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **public** **class** SkipTag **extends** SimpleTagSupport {
2. **public** **void** doTag() **throws** JspException, IOException {
3. **this**.getJspContext().getOut().print("<h1>只能看到我！</h1>");
4. **throw** **new** SkipPageException();
5. }
6. }

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<tag>**
2. **<name>**skip**</name>**
3. **<tag-class>**cn.ywq.tags.SkipTag**</tag-class>**
4. **<body-content>**empty**</body-content>**
5. **</tag>**

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<itcast:skip/>**
2. **<h1>**看不见我！**</h1>**

**2.4　带有属性的标签**

一般标签都会带有属性，例如<c:iftest=””>，其中test就是一个boolean类型的属性。完成带有属性的标签需要：

* 在处理类中给出JavaBean属性（提供get/set方法）；
* 在TLD中部属相关属性。

**[java]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **public** **class** IfTag **extends** SimpleTagSupport {
2. **private** **boolean** test;
3. **public** **boolean** isTest() {
4. **return** test;
5. }
6. **public** **void** setTest(**boolean** test) {
7. **this**.test = test;
8. }
9. @Override
10. **public** **void** doTag() **throws** JspException, IOException {
11. **if**(test) {
12. **this**.getJspBody().invoke(**null**);
13. }
14. }
15. }

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<tag>**
2. **<name>**if**</name>**
3. **<tag-class>**cn.ywq.IfTag**</tag-class>**
4. **<body-content>**scriptless**</body-content>**
5. **<attribute>**
6. **<name>**test**</name>**
7. **<required>**true**</required>**
8. **<rtexprvalue>**true**</rtexprvalue>**
9. **</attribute>**
10. **</tag>**

**[html]** [view plain](http://blog.csdn.net/qq_25827845/article/details/53311722) [copy](http://blog.csdn.net/qq_25827845/article/details/53311722)

1. **<**%
2. pageContext.setAttribute("one", true);
3. pageContext.setAttribute("two", false);
4. %**>**
5. **<it:if** test="${one }"**>**xixi**</it:if>**
6. **<it:if** test="${two }"**>**haha**</it:if>**
7. **<it:if** test="true"**>**hehe**</it:if>**