# Day07

## JSP入门

### 1　JSP概述

#### 1.1　什么是JSP

Tomcat服务器上有两种动态资源：

Servlet：调用service层的方法，实现用户请求的处理，

JSP：进行响应

JSP当做一个Servlet，处理客户端请求，JSP负责页面显示和数据提交。

Jsp实质还是class，jvm编译成class字节码，jvm执行class字节码

JSP页面组成：html+javascript+css+JAVA代码

JSP（Java Server Pages）是JavaWeb服务器端的**动态资源**。它与html页面的作用是相同的，**显示数据和获取数据**。

#### 1.2　JSP的组成

<% java代码（业务逻辑） %>====>jsp引入java的流程控制语句的符号

<%= 输出%>===>jsp变量值的输出

<%-- --%>===>jsp注释 浏览器上看不到jsp注释

HTML注释有什么区别？？HTML注释浏览器上看源代码

JSP = DHtml + Java脚本（代码片段） + JSP动态标签

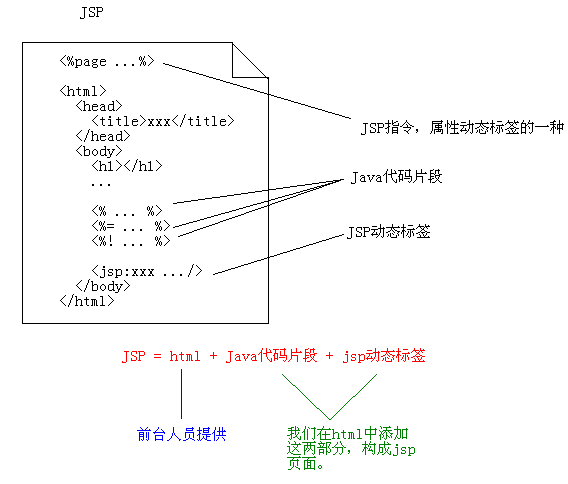
DHtml四种技术的统称：html css ESMAScript dom解析

Java脚本（代码片段）:指servlet编写java程序代码

JSP动态标签：学习

结论：

JSP=DHTML+JSTL+EL+JSP动作



### 2　JSP语法

#### 2.1　JSP脚本

JSP脚本就是Java代码片段，它分为三种：

* <%...%>：Java语句；
  + <%

按照java语法书写

程序=数据+逻辑

* + %>
* <%=…%>：Java表达式；
  + <%= 变量或者常量的值或表达式%>
* <%!...%>：Java定义类成员；
  + 成员变量和成员方法
  + <%!

Public void fun(int num1){

}

* + %>

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>JSP演示</title>  </head>    <body>  <h1>JSP演示</h1>  <%  // Java语句  String s1 = "hello jsp";  // 不会输出到客户端，而是在服务器端的控制台打印  System.out.println(s1);  %>  <!-- 输出到客户端浏览器上 -->  输出变量：<%=s1 %><br/>  输出int类型常量：<%=100 %><br/>  输出String类型常量：<%="你好" %><br/>  <br/>  使用表达式输出常量是很傻的一件事，因为可以直接使用html即可，下面是输出上面的常量：<br/>  100<br/>  你好  </body>  </html> |

2.2　内置对象out

out对象在JSP页面中无需创建就可以使用，它的作用是用来向客户端输出。

|  |
| --- |
| <body>  <h1>out.jsp</h1>  <%  //向客户端输出  out.print("你好！");  %>  </body> |

其中<%=…%>与out.print()功能是相同的！它们都是向客户端输出，例如：

<%=s1%>等同于<% out.print(s1); %>

<%=”hello”%>等同于<% out.print(“hello”); %>，也等同于直接在页面中写hello一样。

#### 2.3　多个<%...%>可以通用

在一个JSP中多个<%...%>是相通的。例如：

|  |
| --- |
| <body>  <h1>out.jsp</h1>  <%  String s = "hello";  %>  <%  out.print(s);  %>  </body> |

循环打印表格：

|  |
| --- |
| <body>  <h1>表格</h1>    <table border=*"1"* width=*"50%"*>  <tr>  <th>序号</th>  <th>用户名</th>  <th>密码</th>  </tr>  <%  **for**(**int** i = 0; i < 10; i++) {  %>  <tr>  <td><%=i+1 %></td>  <td>user<%=i %></td>  <td><%=100 + 1 %></td>  </tr>  <%  }  %>  </table>  </body> |

### 3　JSP的原理

#### 3.1　JSP是特殊的Servlet

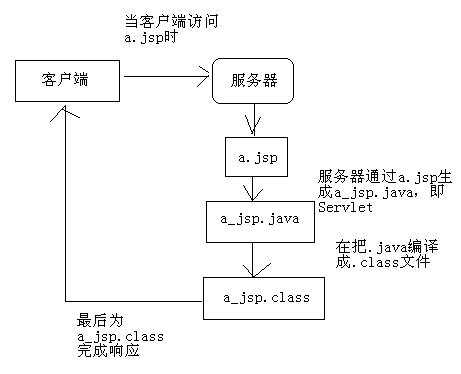
JSP是一种特殊的Servlet，当JSP页面首次被访问时，容器（Tomcat）会先把JSP编译成Servlet，然后再去执行Servlet。所以JSP其实就是一个Servlet！

请求Jsp页面，服务器做哪些事情？？

第一次请求：init() service()

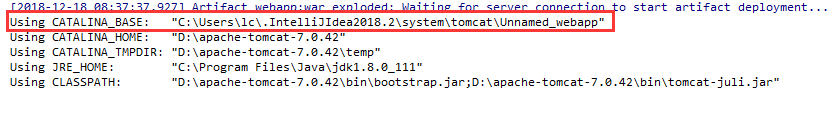
第二次请求：service

destroy()



#### 3.2　JSP真身存放目录

IDEA查看方式：



JSP生成的Servlet存放在${CATALANA}/work目录下，我经常开玩笑的说，它是JSP的“真身”。我们打开看看其中的内容，了解一下JSP的“真身”。

你会发现，在JSP中的静态信息（例如<html>等）在“真身”中都是使用out.write()完成打印！这些静态信息都是作为字符串输出给了客户端。

JSP的整篇内容都会放到名为\_jspService的方法中！

a\_jsp.java的\_jspService()方法：

|  |
| --- |
| public void \_jspService(final javax.servlet.http.HttpServletRequest request,  final javax.servlet.http.HttpServletResponse response)  throws java.io.IOException, javax.servlet.ServletException {  final javax.servlet.jsp.PageContext pageContext;  javax.servlet.http.HttpSession session = null;  final javax.servlet.ServletContext application;  final javax.servlet.ServletConfig config;  javax.servlet.jsp.JspWriter out = null;  final java.lang.Object page = this;  javax.servlet.jsp.JspWriter \_jspx\_out = null;  javax.servlet.jsp.PageContext \_jspx\_page\_context = null;  try {  response.setContentType("text/html;charset=UTF-8");  pageContext = \_jspxFactory.getPageContext(this, request, response,  null, true, 8192, true);  \_jspx\_page\_context = pageContext;  application = pageContext.getServletContext();  config = pageContext.getServletConfig();  session = pageContext.getSession();  out = pageContext.getOut();  \_jspx\_out = out;  …  } |

### 4　再论JSP脚本

JSP脚本一共三种形式：

* <%...%>：内容会直接放到“真身”中；
* <%=…%>：内容会放到out.print()中，作为out.print()的参数；
* <%!…%>：内容会放到\_jspService()方法之外，被类直接包含；

前面已经讲解了<%...%>和<%=…%>，但还没有讲解<%!...%>的作用！

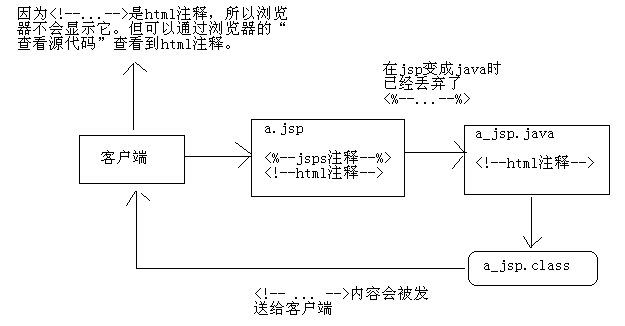
现在我们已经知道了，JSP其实就是一个类，一个Servlet类。<%!...%>的作用是在类中添加方法或成员的，所以<%!...%>中的内容不会出现在\_jspService()中。

|  |
| --- |
| <%!  **private** String name;  **public** String hello() {  **return** "hello JSP!";  }  %> |

### 5　JSP注释

我们现在已经知道JSP是需要先编译成.java，再编译成.class的。其中<%-- ... --%>中的内容在JSP编译成.java时会被忽略的，即JSP注释。

也可以在JSP页面中使用html注释：<!-- … -->，但这个注释在JSP编译成的.java中是存在的，它不会被忽略，而且会被发送到客户端浏览器。但是在浏览器显示服务器发送过来的html时，因为<!-- … -->是html的注释，所以浏览器是不会显示它的。



## JSP指令

JSP=dhtml+java脚本+动态标签+JSP指令

### JSP指令概述

JSP指令的格式：<%@指令名 attr1=”” attr2=”” %>，一般都会把JSP指令放到JSP文件的最上方，但这不是必须的。

JSP中有三大指令：page、include、taglib，最为常用，也最为复杂的就是page指令了。

taglib使用场景：导入标签库，一般用在自定义标签的开发技术。<c:page/>

include使用场景：jsp包含

<jsp:include/>

### 2　page指令 ========= 理解

page指令是最为常用的指定，也是属性最多的属性！

page指令没有必须属性，都是可选属性。例如<%@page %>，没有给出任何属性也是可以的！

在JSP页面中，任何指令都可以重复出现！至少得有一个

<%@ page language=”java”%>

<%@ page import=”java.util.\*”%>

<%@ page pageEncoding=”utf-8”%>

这也是可以的！

#### 2.1　page指令的pageEncoding和contentType（重点）

pageEncoding指定当前JSP页面的编码！这个编码是给服务器看的，服务器需要知道当前JSP使用的编码，不然服务器无法正确把JSP编译成java文件。所以这个编码只需要与真实的页面编码一致即可！在Eclipse中，在JSP文件上点击右键，选择属性就可以看到当前JSP页面的编码了。

contentType属性与response.setContentType()方法的作用相同！它会完成两项工作，一是设置响应字符流的编码，二是设置content-type响应头。例如：<%@ contentType=”text/html;charset=utf-8”%>，它会使“真身”中出现response.setContentType(“text/html;charset=utf-8”)。

无论是page指令的pageEncoding还是contentType，它们的默认值都是ISO-8859-1，我们知道ISO-8859-1是无法显示中文的，所以JSP页面中存在中文的话，一定要设置这两个属性。

其实pageEncoding和contentType这两个属性的关系很“暧昧”：

* 当设置了pageEncoding，而没设置contentType时： contentType的默认值为pageEncoding；
* 当设置了contentType，而没设置pageEncoding时： pageEncoding的默认值与contentType；

**也就是说，当pageEncoding和contentType只出现一个时，那么另一个的值与出现的值相同。如果两个都不出现，那么两个属性的值都是ISO-8859-1。所以通过我们至少设置它们两个其中一个！**

#### 2.2　 page指令的import属性

import是page指令中一个很特别的属性！

import属性值对应“真身”中的import语句。

import属性值可以使逗号：<%@page import=”java.net.\*,java.util.\*,java.sql.\*”%>

import属性是唯一可以重复出现的属性：

<%@page import=”java.util.\*” import=”java.net.\*” import=”java.sql.\*”%>

但是，我们一般会使用多个page指令来导入多个包：增强代码阅读性

<%@ page import=”java.util.\*”%>

<%@ page import=”java.net.\*”%>

<%@ page import=”java.text.\*”%>

#### 2.3　page指令的errorPage和isErrorPage

我们知道，在一个JSP页面出错后，Tomcat会响应给用户错误信息（500页面）！如果你不希望Tomcat给用户输出错误信息，那么可以使用page指令的errorPage来指定错误页！也就是自定义错误页面，例如：<%@page errorPage=”xxx.jsp”%>。这时，在当前JSP页面出现错误时，会请求转发到xxx.jsp页面。

a.jsp

|  |
| --- |
| <%@ page import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ page errorPage=*"b.jsp"* %>  <%  **if**(**true**)  **throw** **new** Exception("哈哈~");  %> |

b.jsp

|  |
| --- |
| <%@ page pageEncoding=*"UTF-8"*%>  <html>  <body>  <h1>出错啦！</h1>  </body>  </html> |

在上面代码中，a.jsp抛出异常后，会请求转发到b.jsp。在浏览器的地址栏中还是a.jsp，因为是请求转发！

而且客户端浏览器收到的响应码为200，表示请求成功！如果希望客户端得到500，那么需要指定b.jsp为错误页面。

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ page isErrorPage=*"true"* %>  <html>  <body>  <h1>出错啦！</h1>  <%=exception.getMessage() %>  </body>  </html> |

　　注意，当isErrorPage为true时，说明当前JSP为错误页面，即专门处理错误的页面。那么这个页面中就可以使用一个内置对象exception了。其他页面是不能使用这个内置对象的！

温馨提示：*IE会在状态码为500时，并且响应正文的长度小于等于512B时不给予显示！而是显示“网站无法显示该页面”字样。这时你只需要添加一些响应内容即可，例如上例中的b.jsp中我给出一些内容，IE就可以正常显示了！*

**2.3.1　web.xml中配置错误页面**

　　不只可以通过JSP的page指令来配置错误页面，还可以在web.xml文件中指定错误页面。这种方式其实与page指令无关，但想来想去还是在这个位置来讲解比较合适！

web.xml

|  |
| --- |
| <error-page>  <error-code>404</error-code>  <location>/error404.jsp</location>  </error-page>  <error-page>  <error-code>500</error-code>  <location>/error500.jsp</location>  </error-page>  <error-page>  <exception-type>java.lang.RuntimeException</exception-type>  <location>/error.jsp</location>  </error-page> |

<error-page>有两种使用方式：

* <error-code>和<location>子元素；
* <exception-type>和<location>子元素；

　　其中<error-code>是指定响应码；<location>指定转发的页面；<exception-type>是指定抛出的异常类型。

在上例中：

* 当出现404时，会跳转到error404.jsp页面；
* 当出现RuntimeException异常时，会跳转到error.jsp页面；
* 当出现非RuntimeException的异常时，会跳转到error500.jsp页面。

　　这种方式会在控制台看到异常信息！而使用page指令时不会在控制台打印异常信息。

#### 2.4　page指令的autoFlush和buffer

buffer表示当前JSP的输出流（out隐藏对象）的缓冲区大小，**默认为8kb**。

autoFlush表示在out对象的缓冲区满时如果处理！当autoFlush为true时，表示缓冲区满时把缓冲区数据输出到客户端；当autoFlush为false时，表示缓冲区满时，抛出异常。**AutoFlush的默认值为true**。

　　这两个属性一般我们也不会去特意设置，都是保留默认值！

#### 2.5　page指令的isELIgnored

忽略：不支持EL表达式

　　后面我们会讲解EL表达式语言，page指令的isElIgnored属性表示当前JSP页面是否忽略EL表达式，默认值为false，表示不忽略（即支持）。

#### 2.6　page指令的其他属性================了解

* language：**只能是Java**，这个属性可以看出JSP最初设计时的野心！希望JSP可以转换成其他语言！但是，到现在JSP也只能转换成Java代码；
* info：JSP说明性信息；备注
* isThreadSafe：默认为false，为true时，JSP生成的Servlet会去实现一个过时的标记接口SingleThreadModel，这时JSP就只能处理单线程的访问；

线程安全特点：性能差 1.jsp和Servlet在tomcat里面有且仅有一个对象（单例模式）

* session：默认为true，表示当前JSP页面可以使用session对象，如果为false表示当前JSP页面不能使用session对象；
* extends：指定当前JSP页面生成的Servlet的父类；

#### 2.7　<jsp-config>（了解）

在web.xml页面中配置<jsp-config>也可以完成很多page指定的功能！

|  |
| --- |
| <jsp-config>  <jsp-property-group>  <url-pattern>\*.jsp</url-pattern>  <el-ignored>true</el-ignored>  <page-encoding>UTF-8</page-encoding>  <scripting-invalid>true</scripting-invalid>  </jsp-property-group>  </jsp-config> |

### include指令

Include:包含。作用合并jsp页面

前端：frameset框架页

实现方式：

1. request.getRequestDispacher().include(request,response);//Servlet包含组合多个响应协议
2. <%@ include %> jsp页面上使用指令
3. <jsp:include/> jsp页面使用标准动作

静态包含：先把所有的要合并的jsp合并起来形成一个jsp，再把这一个jsp编译对应java，翻译class，最后执行 include指令的包含方式是静态包含

动态包含：多个jsp分别翻译成java文件，最后运行时候把多个jsp翻译的java文件合并显示

学习原因：合并界面。减少重复内容的二次开发

比如：淘宝 页面的头部+页面底部

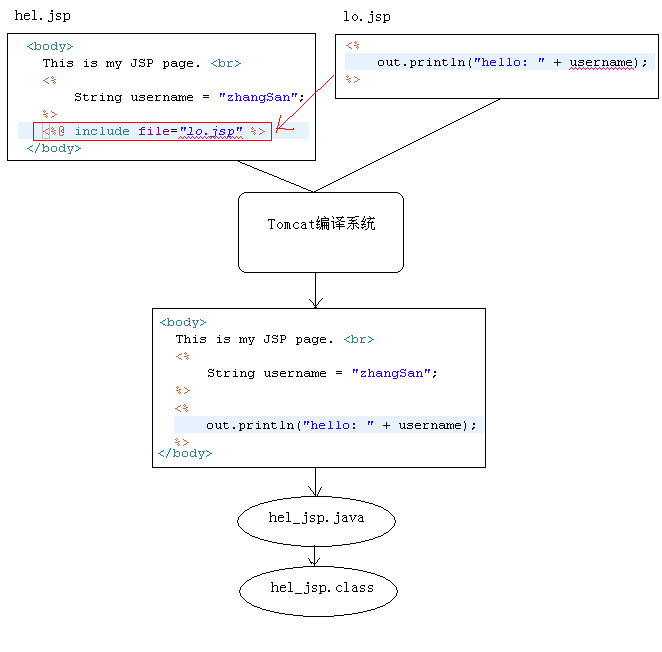
静态包含：先合并jsp，再生产一个java类，然后再编译成一个class这种包含方式，

Request.getRequestDispatcher().include()===请求包含

include指令表示静态包含！即目的是把多个JSP合并成一个JSP文件！

include指令只有一个属性：file，指定要包含的页面，例如：<%@include file=”b.jsp”%>。

静态包含：当hel.jsp页面包含了lo.jsp页面后，在编译hel.jsp页面时，需要把hel.jsp和lo.jsp页面合并成一个文件，然后再编译成Servlet（Java文件）。



很明显，在ol.jsp中在使用username变量，而这个变量在hel.jsp中定义的，所以只有这两个JSP文件合并后才能使用。通过include指令完成对它们的合并！

### 4　taglib指令\*\*\*\*\*\*\*\*

　　这个指令需要在学习了自定义标签后才会使用，现在只能做了了解而已！

在JSP页面中使用第三方的标签库时，需要使用taglib指令来“导包”。例如：

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

Prefix:标签的前缀。理解别名。

Uri:标签库实际的访问链接

其中prefix表示标签的前缀，这个名称可以随便起。uri是由第三方标签库定义的，所以你需要知道第三方定义的uri。

小结：jsp指令一共是三个：

Page：jsp必须的指令，jsp页面属性。

include：实现页面的合并，属于静态包含

taglib：引入第三方的标签库==========================常用的指令

## JSP九大内置对象

### 什么是JSP九大内置对象

内置对象：tomcat创建，不是程序员创建的对象。

在JSP中无需创建就可以使用的9个对象，它们是：

* out（JspWriter）：等同与response.getWriter()，用来向客户端发送文本数据；
* config（ServletConfig）：对应“真身”中的ServletConfig；
* page（当前JSP的真身类型）：当前JSP页面的“this”，即当前对象；
* pageContext（PageContext）：页面上下文对象，它是最后一个没讲的域对象；
  + setAttribute getAttribute removeAttribute
* exception（Throwable）：只有在错误页面中可以使用这个对象；
* request（HttpServletRequest）：即HttpServletRequest类的对象；
* response（HttpServletResponse）：即HttpServletResponse类的对象；
* application（ServletContext）：即ServletContext类的对象；
* session（HttpSession）：即HttpSession类的对象，不是每个JSP页面中都可以使用，如果在某个JSP页面中设置<%@page session=”false”%>，说明这个页面不能使用session。

在这9个对象中有很多是极少会被使用的，例如：config、page、exception基本不会使用。

在这9个对象中有两个对象不是每个JSP页面都可以使用的：exception、session。

在这9个对象中有很多前面已经学过的对象：out、request、response、application、session、config。

### 2　通过“真身”来对照JSP

　　我们知道JSP页面的内容出现在“真身”的\_jspService()方法中，而在\_jspService()方法开头部分已经创建了9大内置对象。

|  |
| --- |
| public void \_jspService(HttpServletRequest request, HttpServletResponse response)  throws java.io.IOException, ServletException {  PageContext pageContext = null;  HttpSession session = null;  ServletContext application = null;  ServletConfig config = null;  JspWriter out = null;  Object page = this;  JspWriter \_jspx\_out = null;  PageContext \_jspx\_page\_context = null;  try {  response.setContentType("text/html;charset=UTF-8");  pageContext = \_jspxFactory.getPageContext(this, request, response,  null, true, 8192, true);  \_jspx\_page\_context = pageContext;  application = pageContext.getServletContext();  config = pageContext.getServletConfig();  session = pageContext.getSession();  out = pageContext.getOut();  \_jspx\_out = out;  **从这里开始，才是JSP页面的内容**  }… |

### pageContext对象===理解

保存数据，共享数据范围：当前jsp页面

使用场景：自定义标签

　　在JavaWeb中一共四个域对象，其中Servlet中可以使用的是request、session、application三个对象，而在JSP中可以使用pageContext---当前页面、request、session、application（即ServletContext）四个域对象。

pageContext 对象是PageContext类型，它的主要功能有：

* 域对象功能；
* 代理其它域对象功能；
* 获取其他内置对象；

Public void fun(pageContext pc){

//使用四大域对象和九大内置对象

}

#### 3.1　域对象功能

域对象作用范围从小到大依次是：

pageContext request session application （指ServletContext）

　　pageContext也是域对象，它的范围是当前页面。它的范围也是四个域对象中最小的！

* void setAttribute(String name, Object value)；
* Object getAttrbiute(String name, Object value)；
* void removeAttribute(String name, Object value)；
* findAttribute(name,域范围):根据name找值=====就是EL的底层实现
  + 翻译：${name}===>xx.findAttribute(name)
  + ${}底层findAttribute()查找key对应的值过程，依照特定顺序查找：

PageContext request session application

${name}等价功能${requestScope.name}

* + 举例：pageContext.findAttribute(key,applictionScope)🡺${}

#### 3.2　代理其它域对象功能

PageContext以一顶四个对象

还可以使用pageContext来代理其它3个域对象的功能，也就是说可以使用pageContext向request、session、application对象中存取数据，例如：

|  |
| --- |
| pageContext.setAttribute("x", "X");  pageContext.setAttribute("x", "XX", PageContext.REQUEST\_SCOPE);  pageContext.setAttribute("x", "XXX", PageContext.SESSION\_SCOPE);  pageContext.setAttribute("x", "XXXX", PageContext.APPLICATION\_SCOPE); |

* void setAttribute(String name, Object value, int scope)：在指定范围中添加数据；
* Object getAttribute(String name, int scope)：获取指定范围的数据；
* void removeAttribute(String name, int scope)：移除指定范围的数据；
* Object findAttribute(String name)：依次在page、request、session、application范围查找名称为name的数据，如果找到就停止查找。这说明在这个范围内有相同名称的数据，那么page范围的优先级最高！

#### 3.3　获取其他内置对象

举例：public void fun(pageContext context){

//使用四大域对象

//使用9大内置对象

}

一个pageContext对象等于所有内置对象，即1个当9个。这是因为可以使用pageContext对象获取其它8个内置对象：

* JspWriter getOut()：获取out内置对象；
* ServletConfig getServletConfig()：获取config内置对象；
* Object getPage()：获取page内置对象；
* ServletRequest getRequest()：获取request内置对象；
* ServletResponse getResponse()：获取response内置对象；
* HttpSession getSession()：获取session内置对象；
* ServletContext getServletContext()：获取application内置对象；
* Exception getException()：获取exception内置对象；

## JSP动作标签

### JSP动作标签概述

Jsp:html+css+js+jq+java

动作标签的作用是用来简化Java脚本的！

JSP动作标签是JavaWeb内置的动作标签，它们是已经定义好的动作标签，我们可以拿来直接使用。

如果JSP动作标签不够用时，还可以使用自定义标签。JavaWeb一共提供了20个JSP动作标签，但有很多基本没有用，这里只介绍一些有用的动作标签。

JSP动作标签的格式：<jsp:标签名 …>

### <jsp:include> 动态包含

就是用作jsp代码合并

<jsp:include>标签的作用是用来包含其它JSP页面的！你可能会说，前面已经学习了include指令了，它们是否相同呢？虽然它们都是用来包含其它JSP页面的，但它们的实现的级别是不同的

include指令是在编译级别完成的包含，即把当前JSP和被包含的JSP合并成一个JSP，然后再编译成一个Servlet。

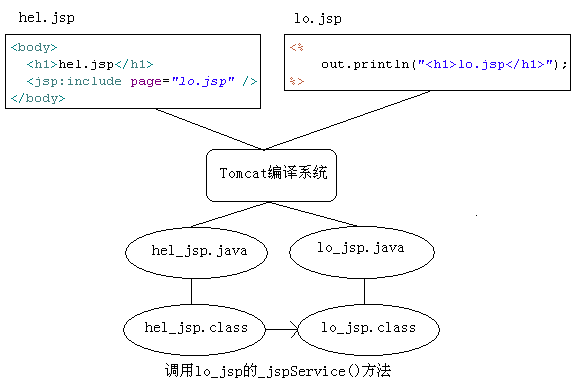
include动作标签是在运行级别完成的包含，即当前JSP和被包含的JSP都会各自生成Servlet，然后在执行当前JSP的Servlet时完成包含另一个JSP的Servlet。它与RequestDispatcher的include()方法是相同的！

hel.jsp

|  |
| --- |
| <body>  <h1>hel.jsp</h1>  <jsp:include page=*"lo.jsp"* />  </body> |

lo.jsp

|  |
| --- |
| <%  out.println("<h1>lo.jsp</h1>");  %> |



　　其实<jsp:include>在“真身”中不过是一句方法调用，即调用另一个Servlet而已。

### 3　<jsp:forward>jsp动作标签

forward标签的作用是请求转发！forward标签的作用与RequestDispatcher#forward()方法相同。

hel.jsp

|  |
| --- |
|  |

lo.jsp

|  |
| --- |
| <%  out.println("<h1>lo.jsp</h1>");  %> |

注意，最后客户端只能看到lo.jsp的输出，而看不到hel.jsp的内容。也就是说在hel.jsp中的<h1>hel.jsp</h1>是不会发送到客户端的。<jsp:forward>的作用是“别在显示我，去显示它吧！”。

### <jsp:param>

不能独立使用，一般就是用在<jsp:include>和<jsp:forward>内部，实现参数传递

还可以在<jsp:include>和<jsp:forward>标签中使用<jsp:param>子标签，它是用来传递参数的。下面用<jsp:include>来举例说明<jsp:param>的使用。

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>a.jsp</title>  </head>    <body>  <h1>a.jsp</h1>  <hr/>  <jsp:include page=*"/b.jsp"*>  <jsp:param value=*"zhangSan"* name=*"username"*/>  </jsp:include>  </body>  </html> |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>b.jsp</title>  </head>    <body>  <h1>b.jsp</h1>  <hr/>  <%  String username = request.getParameter("username");  out.print("你好：" + username);  %>  </body>  </html> |