**JSP深入浅出**

拓薪教育

刘洋

1. JSP概述(了解)
   1. JavaWeb发展历程

* JavaWeb早期：applet + servlet
* JavaWeb早期：servlet，在servlet中需要写大量的

response.getWriter().println(“<html>”);

* JavaWeb后期：JSP，只需要在html中直接添加动态信息，即java代码片段就可以了。服务器会把所有的<html>语句转换成response.getWriter().println(“<html>”);

<html> -- response.getWriter().println(“<html>”);

<body></body> -- response.getWriter().println(“<body></body>”);

</html> -- response.getWriter().println(“</html>”);

* 1. 什么是JSP

html + java代码 + JSP动态标签 = jsp

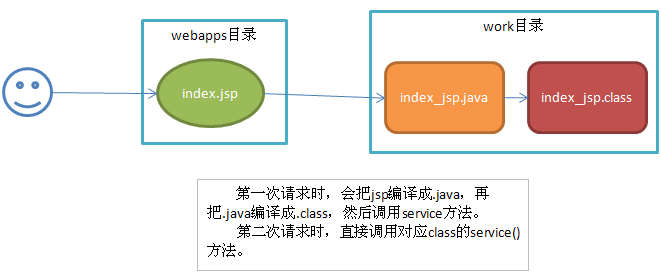
Java Server page

在静态页面上添加动态信息就可以了，如果是Servlet还需要一行一行的输出。

通常在前台开发人员给出静态页面后，后台开发人员只需在静态页面中添加动态信息即可，而不用再使用Servlet一行一行的去输出了。

* 1. JSP真身:Servlet

当客户请求的是JSP页面时，服务器会查看JSP对应的Servlet是否存在，如果存在，那么直接调用Servlet的service()方法来处理请求。如果不存在，那么服务器会先把JSP编译成.java，再把.java编译成.class，然后调用service()方法。当这个JSP页面，第二处理请求时，就直接调用直身的service()方法了。



JSP是Servlet，JSP与我们编写的Servlet有什么不同呢？通常我们编写的Servlet就不需要输出HTML代码了，而是转发到JSP页面。

即Servlet不再负责显示页面，而由JSP来完成！而Servlet专注与控制层内容！

1. JSP的scriptlet(熟练)

从单词上分析script是脚本：let在JAVA中表示的小程序，scriptlet表示脚本小程序。

JSP中的Java代码块就是最常见的动态信息。它分为三种：

* **<%...%>**：java代码片段

代码片段会原模原样的出现在真身中，不会有变动。正常的Java代码都可以在代码片段中出现；

* **<%=…%>**：表达式在页面上打印数据

表达式会在“真身”中输出，例如：<%=a%>，对应真身中的out.print(a)。只能出现常量、变量，以及有返回值的方法调用，而且不能出现分号！即在out.println()的合法参数都可以！

<%=a%>对应out.println(a)；

<%=”a”%>对应out.println(“a”)；

<%=fun()%>对应out.println(fun())；

<%=a;%>对应out.println(a;)，所以出错；

<%=System.out.println(“hello”)%>对应out.println(System.out.println(“hello”))，所以出错。

* **<%!...%>**：声明

声明对应“真身”中的属性和方法！

1. JSP注释(熟练)

语法：<%-- … --%>

其中JSP只有一种注释：<%-- … --%>，注释中的内容会被JSP编译系统忽略！

|  |
| --- |
| *<%--<% String name = "txjava";%>--%>* |

* java片段中的代码注释依然遵守java的注释//，/\*...\*/， /\*\*.../
* html片段中的代码依然遵守html的注释<!-- ... -->，注意的是此注释不能注释带有jsp的scriptlet的脚本，虽然页面上不展示注释掉的代码，但是查看网页源代码可以发现jsp脚本已经生效只不过产出的html代码被注释掉了。（在jsp中不建议使用这种注释）

Html注释和jsp注释二者的区别！

* **<!-- -->**：Tomcat不把它当注释，和其他东西一样，会发送到客户端。但浏览器不会去显示它，因为浏览器知道它是注释；
* **<%-- --%>**：Tomcat在生成“真身”时，就会忽略它，因为Tomcat认识它，知道它是注释。所以在“真身”中就没有它，也不会发现到客户端。也就是说，客户端永远看不到它！

1. JSP内置对象(精通)
   1. 什么是内置对象（面试常问）

内置对象是在JSP页面中无需创建就可以直接使用的变量。在JSP中一共有9个这样的对象！它们分别是：

* out（JspWriter）；
* config（ServletConfig）；
* page（当前JSP的真身类型）；
* **pageContext**（PageContext）；
* exception（Throwable）；
* **request**（HttpServletRequest）；
* **response**（HttpServletResponse）；
* **application**（ServletContext）；
* **Session** （HttpSession）。
  1. 内置对象概述
* out：最为常用的方法是print()，向页面输出！它与response.getWriter()基本相同！
* config：在页面中基本没有什么用，但如果JSP在web.xml中存在配置，而且存在初始化参数，那么可以使用config来获取；config对象的功能是：getServletContext()、getServletName()、getInitParameter()，这三个方法在JSP中都没什么用。所以config就没什么用了。JSP也可以在web.xml中配置，但我们没有配置它的必要！
* page：基本没用！表示当前JSP页面的当前实例！在页面中使用this和使用page是相同的；也没什么用。
* **request**：与Servlet中的request一样，没有区别；
* **response**：与Servlet中的response一样，没有区别；
* **application**：就是ServletContext对象；
* **session**：就是HttpSession对象；
* exception：虽然名字叫exception，但类型为Throwable。它只能在错误页中可以使用！后面讲
* **pageContext**：很有用的东西，下面会讲解。你以后可能不会经常直接使用它，但一定间接使用它
  1. 对照JSP真身查看内置对象

我们发现，在JSP中的内容会出现在“真身”的\_jspService()方法中，而且在\_jspService()方法上方还有一些其实代码：

|  |
| --- |
| ***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  **}**… |

JSP中的内容都出现在try块中，而且在try块的正文！上方是对隐藏对象的初始化！！！

上面代码只是给出普通页面的“真身”，而错误页面的“真身”你会看到exception对象的出现。

* 1. JSP四个域对象
     1. 域的范围和属性管理
* pageContext范围； 当前页面之内有效
* request范围；当前的请求内有效
* session范围；当前的会话内有效
* application范围；当前这次服务器生命周期内有效

域对象的共同特点是都管理域中的属性,他们有着相同的方法:

* **void** **setAttribute**(String name, Object value)；
* **Object** **getAttrbiute**(String name, Object value)；
* **void** **removeAttribute**(String name, Object value)；
  + 1. pageContext详细分析

pageContext 对象是PageContext类型，它不只是域对象，而且还可以操作所有域对象，还可以获取其他隐藏对象。

* 本身是域对象：pageContext是JSP中的域对象，而在Servlet中不能使用它！它表示的当前页面中可以使用，是最小的范围！
* **void** **setAttribute**(String name, Object value)；
* **Object** **getAttrbiute**(String name, Object value)；
* **void** **removeAttribute**(String name, Object value)；
* 操作所有域（四个域）：可以使用pageContext对象操作所有域对象，在getAttribute()、setAttribute()、removeAttribute()三个方法中多添加一个参数，int scope来指定范围。在PageContext类中包含四个int类型的常量表示四个范围：
* **PAGE\_SCOPE**：pageContext范围；
* **REQUEST\_SCOPE**：request范围；
* **SESSION\_SCOPE**：session范围；
* **APPLICATION\_SCOPE**：application范围；
* void setAttribute(String name, Object value, int scope)：设置数据到指定的范围中，例如：pageContext.setAttribute(“hello”, “hello world”, PageContext.REQUEST)，这个方法调用等同与：request.setAttribute(“hello”, “hello world”)；
* 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)：在所有范围内查找数据，依次为page、request、session、application。如果在前一个范围内查找到数据，那么就返回，而不会在到下一个范围中去查找！
* 获取其他隐藏对象：可以使用pageContext获取其他隐藏对象。
* JspWriter getOut()：获取out隐藏对象；
* ServletConfig getServletConfig()：获取config隐藏对象；
* Object getPage()：获取page隐藏对象；
* HttpServletRequest getRequest()：获取request隐藏对象；
* HttpServletResponse getResponse：获取response隐藏对象；
* **HttpSession getSession()**：获取session隐藏对象；
* **ServletContext getServletContext()**：获取application隐藏对象；
* JspException getException()：获取exception隐藏对象转换后的JspException对象。

1. JSP指令(熟练)
   1. 什么是JSP指令

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。

指令在“真身”中不存在，生成“真身”时需要使用指令！

Tomcat编译系统会根据JSP的指令信息来编译JSP，生成Java文件。

在生成的Java文件中，不存在指令信息！

* 1. page指令的pageEncoding和contentType

pageEncoding指定的是当前JSP页面的编码！Tomcat编译系统会使用这个编码把JSP编译成Java文件。所以这个编译只需要与真实的页面编译一致即可！

contentType指定的是响应给我客户端时使用的编码，即对应response.setConteType()方法的参数值！ Tomcat都会把响应正文转换成UTF-8编译，然后发送给我客户端，并且会在响应头中设置Content-Type头信息为text/html;charset=utf-8，这样浏览器就知道使用服务器发送过来的正文使用了什么编码。

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：表示contentType的取值为pageEncoding的值；
* 当设置了contentType，而没设置pageEncoding时：表示pageEncoding的取值与contentType的值；

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。

**处理乱码的方案：**

* 把JSP页面的编码设置为UTF-8；
* 把contentType和pageEncoding都设置为UTF-8。
  1. 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：

但是，我们一般会使用多个page指令来导入多个包：

|  |
| --- |
| <%@ ***page*** import="**java.util.\***"%> <%@ ***page*** import="**java.net.\***"%> <%@ ***page*** import="**java.text.\***"%> |

* 1. page指令的errorPage和isErrorPage
     1. errorPage

我们知道，在一个JSP页面出错后，Tomcat会响应给我用户错误信息！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！例如：

|  |
| --- |
| <%@ ***page*** errorPage="**error.jsp**"%> |

这时，在当前JSP页面出现错误时，会转发到error.jsp页面。

pageError的路径，由于是服务器端的跳转，所以绝对路径不需要加项目名，直接指定项目名称后面的路径即可

* + 1. isErrorPage

如果希望在error.jsp页面中获得异常对象，那么就需要把error.jsp标记为错误页！这需要设置page指令的isErrorPage属性为true。

|  |
| --- |
| <%@ ***page*** isErrorPage="**true**" %> **<*html*> <*head*>  <*title*>**错误页面**</*title*> </*head*> <*body*>** 出错了!!!  ***<%=exception*.getMessage() *%>* </*body*> </*html*>** |

* + 1. error-page

还可以在web.xml文件中配置错误页面，在出现某种错误时跳转到对应的页面处理。

|  |
| --- |
| **<*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*>** |

当出现404时，会跳转到error404.jsp页面；

当出现500时，会跳转到error500.jsp页面。

还可以配置更多，更细致的异常类型。

* 1. page指令的isELIgnored

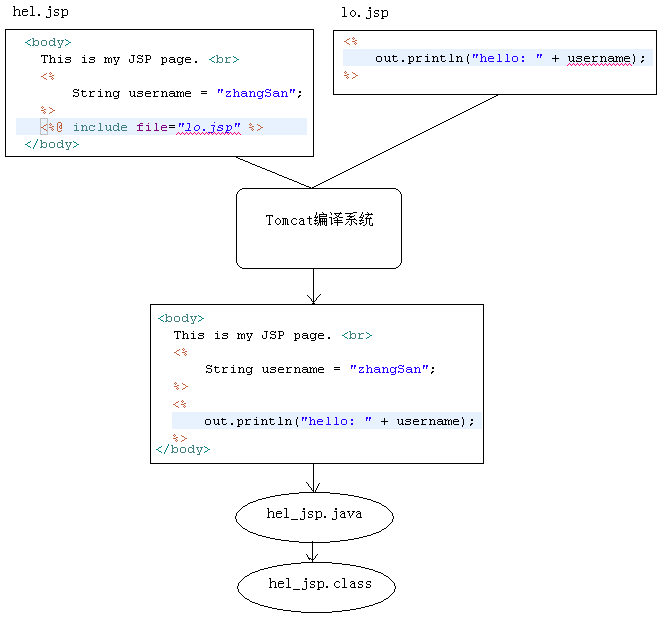
后面我们会讲解EL表达式语言，page指令的isElIgnored属性表示当前JSP页面是否忽略EL表达式，默认值为false，表示不忽略（即支持）。

* 1. include指令(\*\*\*\*\*)

include指令表示静态包含！

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）。



在上面代码中，lo.jsp中没有定义username变量，所以lo.jsp不能处理请求，只有hel.jsp才能处理请求！当访问hel.jsp时，hel.jsp会包含lo.jsp，所以两个会合并成一个文件后再编译成Java文件。在合并之后，因为在hel.jsp中定义了username变量，所以lo.jsp中也就不会出现错误了。

* 1. taglib指令

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" %>

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。

在下面我们讲解JSTL标签库时会说明taglib指令的使用方式。

1. JSP动态标签(了解)

JSP自己的标签，不用导包！

JSP动态标签是JSP自己的标签，不是由第三方提供的，所以使用JSP动态标签时无需使用taglib指令“导包”。

JSP动态标签的格式为：<jsp:xxx …>

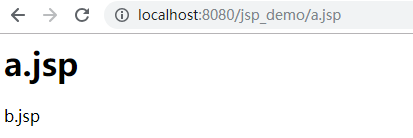
无论是自定义的JSP标签，还是JSP自己的动态标签，还有第三方的标签，最终都会对应一组方法的调用！！！

* 1. include标签

include标签是动态包含，与include指令不同，include标签与RequestDispatcher.include()方法的功能是相同的。

|  |
| --- |
| <%@ ***page*** contentType="**text/html;charset=UTF-8**" language="**java**" %> **<*html*> <*head*>  <*title*>**a.jsp**</*title*> </*head*> <*body*>**  **<*h1*>**a.jsp**</*h1*>** <***jsp:include*** page**="b.jsp"**/> **</*body*> </*html*>** |

|  |
| --- |
| <%@ ***page*** contentType="**text/html;charset=UTF-8**" language="**java**" %> **<*html*> <*head*>  <*title*>**b.jsp**</*title*> </*head*> <*body*>** b.jsp **</*body*> </*html*>** |



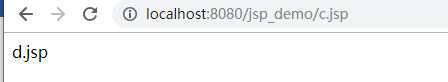
动态包含是会为两个JSP页面都生成“真身”，然后a.jsp的“真身”中会调用b.jsp的“真身”的\_jspService()方法而已。

* 1. forward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。

|  |
| --- |
| <%@ ***page*** contentType="**text/html;charset=UTF-8**" language="**java**" %> **<*html*> <*head*>  <*title*>**c.jsp**</*title*> </*head*> <*body*> <*h1*>**c.jsp**</*h1*>** <***jsp:forward*** page**="d.jsp"**/> **</*body*> </*html*>** |

|  |
| --- |
| <%@ ***page*** contentType="**text/html;charset=UTF-8**" language="**java**" %> **<*html*> <*head*>  <*title*>**d.jsp**</*title*> </*head*> <*body*>** d.jsp **</*body*> </*html*>** |



注意，显示结果中没有<h1>c.jsp</h1>，即c.jsp中的所有输出都会被清除！

1. 学习目标
   * 掌握jsp的本质,并熟练使用JSP相关语法,能够使用JAVA代码和html拼装成完整页面;
   * 熟练掌握JSP常用内置对象;
   * JSP四个域对象倒背如流,并知晓每个对象的作用范围及应用场景;
   * 熟练掌握JSP指令,能解决JSP页面中文乱码问题;
   * 了解JSP动态标签,知晓其含义并能够使用;
2. 作业练习

使用JSP做一套对person表的增删改查