JSP知识

1. **jsp概念**

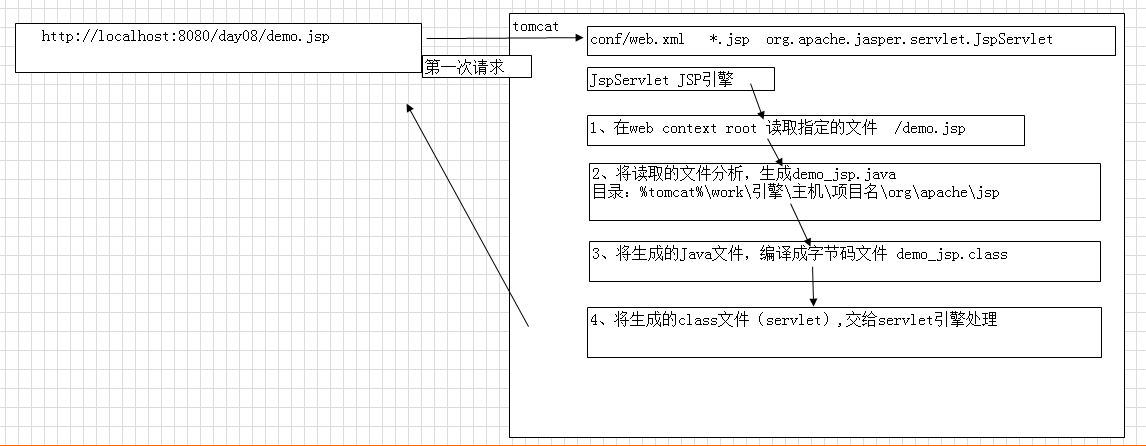
概念：是以Java语言为基础的动态网页开发技术，是在服务器端执行的，jsp本质上就是一个servlet。HTMl代码中嵌套java代码。

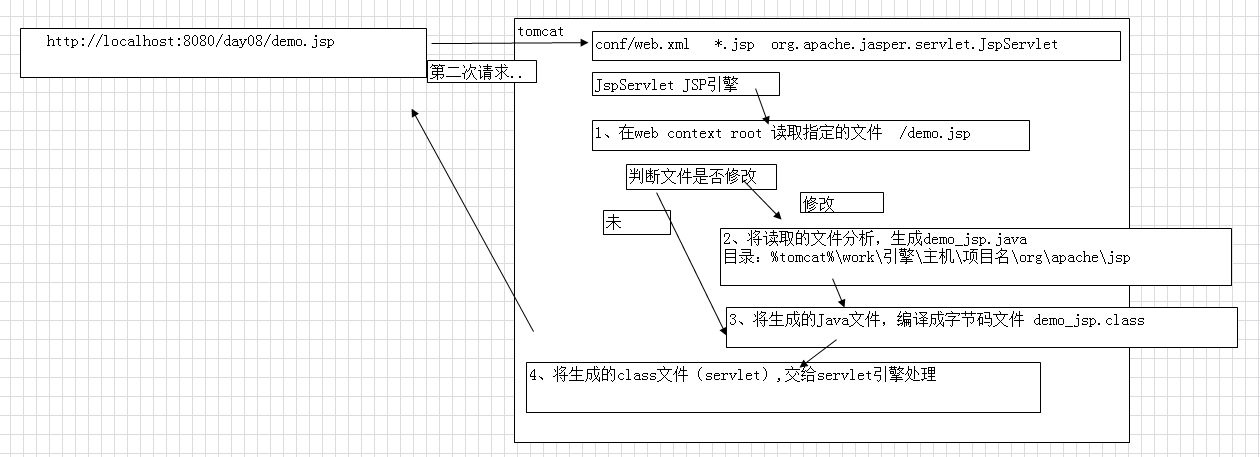
继承关系：demo\_jsp可以在tomcat服务器下的work目录下可以找到

\* public final class demo\_jsp extends org.apache.jasper.runtime.HttpJspBase

\* public abstract class HttpJspBase extends HttpServlet implements HttpJspPage

\* 总结：demo\_jsp --> HttpJspBase --> HttpServlet





1. **page命令**

page指令用来定义JSP文件的全局属性。

**格式：**<%@ page ...%>

**属性配置：**

\* pageEncoding:配置当前页面保存信息的编码

\* contentType:配置当前页面生成的servlet文件响应浏览器的编码

\* response.setContentType("text/html;charset=UTF-8");

\* 比较：

\* 仅有pageEncoding

\* 配置当前页面的编码

\* 配置生成servlet文件响应浏览器的编码

\* 仅有contentType

\* 配置当前页面的编码

\* 配置生成servlet文件响应浏览器的编码

\* 两个都有

\* pageEncoding仅设置当前页面的编码

\* contentType仅设置servlet文件响应浏览器的编码

\* import：引入当前JSP页面需要使用的Java类或接口的包路径

\* 可以重复引用

\* 多个之间可以使用逗号分隔

\* 导包最后一条可以输入多个分号

\* language:确定当前jsp页面使用的语言

\* 默认：java

\* info:确定当前页面的输出信息

\* session：确定当前页面是否可以使用session

\* 默认：ture

\* extends,确定当前jsp页面在生成servlet文件时，继承的父类

\* 扩展

\* 例如：page extends="java.lang.String"

\* isErrorPage:是否是错误页面

\* 默认值：false

\* 当设置成true，在当前页面将可以使用一个变量 exception

\* errorPage：确定错误页面

\* 当页面出现异常时，跳转的页面

\* / 指 context root

\* isThreadSafe:是否是线程安全

\* 默认值：true

\* 如果设置成false，在生成的java源码中添加implementsSingleThreadModel

\* isELIgnored：是否忽略EL表达式

\* 默认值：false

\* 如果设置true，将忽略EL表达式

\* buffer:当前servlet文件的缓存区大小

\* 默认值是8kb

\* autoFlush:是否自动刷新,当buffer缓存区被填写满后，会自动的刷新内容到浏览器

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

<%@ page pageEncoding="UTF-8" %>

<%@ page import="java.util.ArrayList" import="java.util.List"%>

<%@ page import="java.util.\*"%>

<%@ page import="java.util.ArrayList,java.util.List;"%>

<%@ page language="java"%>

<%@ page info="Hello World" %>

<%@ page session="true" %>

1. **include命令**
2. **页面出错跳转到错误页面**

发生错误的页面：

<%@ page contentType=*"text/html; charset=UTF-8"*%>

<%@ page pageEncoding=*"UTF-8"* %>

<%@ page import=*"java.util.ArrayList"* import=*"java.util.List"*%>

<%@ page import=*"java.util.\*"*%>

<%@ page import=*"java.util.ArrayList,java.util.List;"*%>

<%@ page language=*"java"*%>

<%@ page info=*"Hello World"* %>

<%@ page session=*"true"* %>

<!--配置错误跳转页面 -->

<%@ page isErrorPage=*"true"* errorPage=*"/error.html"*%>

这个是第一个jsp页面

<%

List list = **new** ArrayList();

session.isNew();

**int** i = 10 / 0 ;

%>

错误跳转的页面：

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>error.html</title>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"this is my page"*>

<meta http-equiv=*"content-type"* content=*"text/html; charset=UTF-8"*>

<!--<link rel="stylesheet" type="text/css" href="./styles.css">-->

</head>

<body>

This is my HTML page. error <br>

</body>

</html>

1. **Jsp脚本命令**

①<%! Java方法或字段 %>

这种方式嵌入的java代码块相当于在类中定义的成员变量或者方法。

②<%= java表达式 %>

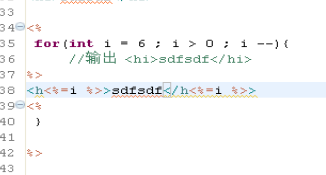
这种方式只能写表达式，而不能出现java语句。这种值会直接输出到页面。

比如<%=name%> 其中name=gh，那么在页面上就会输出gh。

③<% java代码片段 %>

此种方式的java代码块只会出现在服务器编辑过后的service方法中，也就是我们请求服务所调用的方法。因此这种方式的java代码块只能出现java 方法中能出现的语句，而不能出现方法体等。

**综合应用：**





1. **Jsp内置对象**

Jsp内置对象是在JSP页面中隐藏的Java对象。一共9个内置对象

①Object page ： 当前类的引用。作用范围：当前页面

②HttpServletRequest request 作用范围：一次请求（可能涉及多个JSP或Servlet，转发）。

③HttpServletResponse response

④HttpSession session 作用范围：一个会话

⑤ServletContext application 作用范围：整个web项目

⑥ServletConfig config 当前servlet的配置。可以单独配置，把JSP文件当成servlet使用。

\* 配置文件的位置：WebRoot/web.xml

\* 内容

<!-- 注册jsp -->

<servlet>

<servlet-name>jspfile</servlet-name>

<jsp-file>/obj.jsp</jsp-file>

<init-param>

<param-name>user</param-name>

<param-value>lt</param-value>

</init-param>

</servlet>

<!-- 映射 -->

<servlet-mapping>

<servlet-name>jspfile</servlet-name>

<url-pattern>/jsp</url-pattern>

</servlet-mapping>

\* 注意：

1、使用tomcat，servlet引擎

http://localhost:8080/day08/obj.jsp

2、自定义访问路径

<http://localhost:8080/day08/jsp>

⑦Throwable exception 错误页面中获得异常信息

⑧JspWriter out

⑨PageContext pageContext 当前JSP页面的上下文对象，管理当前JSP页面的。

具有其他隐式对象的引用。

设置属性的值

\* 在不同的作用域设置属性值

pageContext.setAttribute("user","liang\_page",PageContext.PAGE\_SCOPE);

pageContext.setAttribute("user","liang\_request",PageContext.REQUEST\_SCOPE);

pageContext.setAttribute("user","liang\_session",PageContext.SESSION\_SCOPE);

pageContext.setAttribute("user","liang\_application",PageContext.APPLICATION\_SCOPE);

\* 从不同的作用域获得属性值

<%=pageContext.getAttribute("user",PageContext.PAGE\_SCOPE) %><br/>

<%=pageContext.getAttribute("user",PageContext.REQUEST\_SCOPE) %><br/>

<%=pageContext.getAttribute("user",PageContext.SESSION\_SCOPE) %><br/>

<%=pageContext.getAttribute("user",PageContext.APPLICATION\_SCOPE) %><br/>

\* 从不同的作用域，依次查询需要的属性值

\* <%=pageContext.findAttribute("user") %>

\* page、request、session、application