# 一 JSP基础

## 1什么是JSP

Java Server Pages，它和servle技术一样，是Java中用于开发动态web资源的技术。

Jsp相当于Java + html

JSP的最大的特点在于，写jsp就像在写html。相比html而言，html只能为用户提供静态数据，而Jsp技术允许在页面中嵌套java代码，为用户提供动态数据。相比servlet而言，servlet很难对数据进行排版，而jsp除了可以用java代码产生动态数据的同时，也很容易对数据进行排版。

不管是JSP还是Servlet，虽然都可以用于开发动态web资源。但由于这2门技术各自的特点，在长期的软件实践中，人们逐渐把servlet作为web应用中的控制器组件来使用，而把JSP技术作为数据显示模板来使用。

原因在于，程序的数据通常要美化后再输出，让jsp既用java代码产生动态数据，又做美化会导致页面难以维护；让servlet既产生数据，又在里面嵌套html代码美化数据，同样也会导致程序可读性差，难以维护。

因此最好的办法就是根据这两门技术的特点，让它们各自负责各的，servlet只负责响应请求产生数据，并把数据通过转发技术带给jsp，数据的显示jsp来做。

## 2 JSP执行过程

Jsp本质上就是servlet

第一次访问jsp文件时，会在tomcat的work目录下生成对应的java文件，然后编译为字节码文件，执行jsp时，其实就是执行对应的class文件，并执行其中的相关方法

再次访问该jsp时，直接从work下找到对应的class文件，并执行

# 二 JSP语法

## 1 JSP脚本

<%

Java语句

%>

注意：JSP脚本中只能出现java代码，不能出现其它模板元素（html相关标签）， JSP引擎在翻译JSP页面中，会将JSP脚本中的Java代码将被原封不动地放到Servlet的\_jspService方法中。

JSP脚本片断中的Java代码必须严格遵循Java语法

<!-- jsp脚本，脚本中声明的变量放在\_jspService方法中 -->

<%

**int** num = 10;

String name = "zhangsan";

System.out.println(name);

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String dateStr = sdf.format(date);

System.out.println(dateStr);

%>

单个脚本中的Java语句可以是不完整的，但是，多个脚本片断组合后的结果必须是完整的Java语句，例如：

<%

for (int i=1; i<5; i++)

{

%>

<H1>hahahaha</H1>

<%

}

%>

<%

**for**(**int** i = 1; i <= 6; i++){

response.getWriter().write("<h" + i + ">标题" + i + "<h" + i + "/>");

}

%>

<%

**for**(**int** i = 1; i <= 6; i++){

%>

<h<%=i %>>标题<%=i %></h<%=i %>>

<%

}

%>

## 2 JSP表达式

<%=java的表达式 %>

当前时间:<%= new java.util.Date() %>

JSP引擎在翻译脚本表达式时，会将程序数据转成字符串，然后在相应位置用out.write(…) 将数据输给客户端。

注意：JSP脚本表达式中的变量或表达式后面不能有分号（;）。

<!-- jsp表达式 -->

<!-- 相当于out.print(name.length() ); -->

<%=name %>

<%=name.length() %>

## 3 JSP声明

<%!

定义变量、方法等

%>

Jsp声明中的java代码被翻译到\_jspService方法的外面，即成为实例变量。JSP声明可用于定义JSP页面转换成的Servlet程序的静态代码块、成员变量和方法 。

<!-- jsp声明 -->

<%!

//相当于成员变量

**private** **int** aaa = 10;

//定义了一个方法

**public** **int** add(**int** a, **int** b){

**return** a + b;

}

**int** sum = add(10, 12);

//System.out.println(sum);

%>

## 4 JSP注释

<%-- 注释的内容 --%>

# 三 JSP三大指令

JSP的指令并不直接产生任何可见输出，而只是告诉引擎如何处理JSP页面中的其余部分

JSP指令的基本语法格式：<%@ 指令 属性名="值" %>

## 1 include指令 理解

include指令用于引入其它JSP页面，如果使用include指令引入了其它JSP页面，JSP引擎将把这**两个JSP翻译成一个servlet**。所以include指令引入通常也称之为静态引入。

语法：

<%@ include file="relativeURL"%>

其中的file属性用于指定被引入文件的相对路径。file属性的设置值必须使用相对路径，如果以“/”开头，表示相对于当前WEB应用程序的根目录，否则，表示相对于当前文件。

注意：

被引入的文件必须遵循JSP语法。

被引入的文件可以使用任意的扩展名，即使其扩展名是html，JSP引擎也会按照处理jsp页面的方式处理它里面的内容。

由于使用include指令将会涉及到2个JSP页面，并会把2个JSP翻译成一个servlet，所以这2个JSP页面的指令不能冲突（pageEncoding和导包除外）。

<!-- include指令 ，相当于将多个文件合为一个文件，转换为一个java文件-->

<%@ include file=*"header.jsp"* %>

## 2 page指令 理解

用于定义JSP页面的各种属性，无论page指令出现在JSP页面中的什么地方，它作用的都是整个JSP页面，为了保持程序的可读性和遵循良好的编程习惯，page指令最好是放在整个JSP页面的起始位置。

page指令的完整语法：

<%@ page

[ language="**java**" ]

[ **import="{*package*.*class* | *package*.*\**}, ..."** ]

[ **session="true | false"** ]

[ buffer="none | **8kb** | *size*kb" ]

[ autoFlush="**true** | false" ]

[ isThreadSafe="**true** | false" ]

[ info="*text*" ]

[ **errorPage="relative\_url"** ]

[ **isErrorPage="true | false"** ]

[ **contentType="*mimeType* [ ;charset=*characterSet* ]"** | "**text/html ; charset=ISO-8859-1**" ]

[ **pageEncoding="*characterSet* | ISO-8859-1"** ]

[ **isELIgnored="true | false"** ]

<%--

page中

session值默认为true，表示会自动创建一个session对象。当不需要时，可以设置为false

pageEncoding jsp文件本身的编码方式

contentType 服务器发送给浏览器的内容的编码

--%>

<%@ page language=*"java"*

import=*"java.util.\*"*

contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*

session=*"false"*

%>

## 3 taglib指令 掌握

用于在JSP页面中导入标签库

# 四 内置对象

每个JSP 页面在第一次被访问时，WEB容器都会把请求交给JSP引擎（即一个Java程序）去处理。JSP引擎先将JSP翻译成一个\_jspServlet(实质上也是一个servlet) ，然后按照servlet的调用方式进行调用。

由于JSP第一次访问时会翻译成servlet，所以第一次访问通常会比较慢，但第二次访问，JSP引擎如果发现JSP没有变化，就不再翻译，而是直接调用，所以程序的执行效率不会受到影响。

JSP引擎在调用JSP对应的\_jspService时，会传递或创建9个与web开发相关的对象供\_jspService使用。JSP技术的设计者为便于开发人员在编写JSP页面时获得这些web对象的引用，特意定义了9个相应的变量，开发人员在JSP页面中通过这些变量就可以快速获得这9大对象的引用。

## 1 九大内置对象 了解

request HttpServletRequest

response HttpServletResponse

application ServletContext

session HttpSession

out JspWriter

pageContext PageContext

exception Throwable

config ServletConfig

page Object(this) 较少使用

## 2 pageContext

pageContext对象是JSP技术中最重要的一个对象，它代表当前JSP页面的运行环境，这个对象不仅封装了对其它8大隐式对象的引用，它自身还是一个域对象，可以用来保存数据。并且，这个对象还封装了web开发中经常涉及到的一些常用操作，例如包含和跳转其它资源、检索其它域对象中的属性等

<%

//可以获取其他内置对象

pageContext.getRequest();

pageContext.getResponse();

pageContext.getServletContext();

pageContext.getSession();

pageContext.getServletConfig();

pageContext.getException();

pageContext.getOut();

pageContext.getPage();

%>

<%

//pageContext 域对象

//pageContext.setAttribute("msg", "page\_msg");

//第三个参数，指定向哪个域中设置数据

//四个域对象

pageContext.setAttribute("msg", "page\_msg", PageContext.PAGE\_SCOPE);

pageContext.setAttribute("msg", "request\_msg", PageContext.REQUEST\_SCOPE);

pageContext.setAttribute("msg", "session\_msg", PageContext.SESSION\_SCOPE);

pageContext.setAttribute("msg", "application\_msg", PageContext.APPLICATION\_SCOPE);

%>

## 3 四个域对象 记住

request

session

application

pageContext

<%

System.out.println(pageContext.getAttribute("msg"));

System.out.println(request.getAttribute("msg"));

System.out.println(session.getAttribute("msg"));

System.out.println(application.getAttribute("msg"));

//从域中查找属性

//pageContext->reqeust->session->application

System.out.println(pageContext.findAttribute("msg"));

%>

# 五 EL表达式

EL：Expression Language

## 1 获取数据

EL表达式主要用于替换JSP页面中的脚本表达式，以从各种类型的域中检索java对象、获取数据。

使用EL表达式获取数据语法：${标识符}

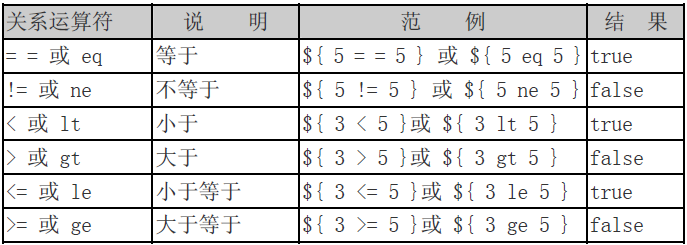
1）从指定域中获取数据

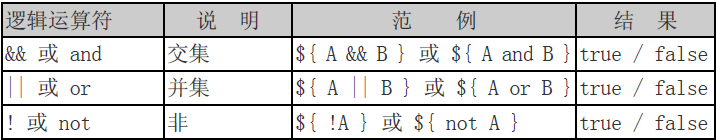
2）获取对象属性

3）获取集合中值

## 2 表达式运算

利用EL表达式可以在JSP页面中执行一些基本的关系运算、逻辑运算和算术运算，以在JSP页面中完成一些简单的逻辑运算





## 3 内置对象

EL 表达式定义了一些内置(隐式)对象，利用这些隐式对象，开发人员可以获得对web中常用对象的引用，从而获得这些对象中的数据

|  |  |
| --- | --- |
| **隐含对象名称** | **描 述** |
| pageContext | 对应于JSP页面中的pageContext对象（注意：取的是pageContext对象。） |
| pageScope | 代表page域中用于保存属性的Map对象 |
| requestScope | 代表request域中用于保存属性的Map对象 |
| sessionScope | 代表session域中用于保存属性的Map对象 |
| applicationScope | 代表application域中用于保存属性的Map对象 |
| param | 表示一个保存了所有请求参数的Map对象 |
| paramValues | 表示一个保存了所有请求参数的Map对象，它对于某个请求参数，返回的是一个string[] |
| header | 表示一个保存了所有http请求头字段的Map对象 |
| headerValues | 同上，返回string[]数组。注意：如果头里面有“-” ，例Accept-Encoding，则要headerValues[“Accept-Encoding”] |
| cookie | 表示一个保存了所有cookie的Map对象 |
| initParam | 表示一个保存了所有web应用初始化参数的map对象 |

# 六 jsp标签

## 1 内置动作标签

## 2 jstl标签

JSTL（JSP Standard Tag Library，JSP标准[标签库](https://baike.baidu.com/item/%E6%A0%87%E7%AD%BE%E5%BA%93" \t "_blank))是一个不断完善的开放源代码的JSP标签库，是由apache的jakarta小组来维护的。主要包括：

核心标签(c:)

JSTL函数(fn:)

格式化标签(fmt:)

数据库标签

XML标签

使用步骤

1. 导入jstl-1.2.jar文件
2. 在jsp中导入相关的标签库

<!-- 引入核心标签库，前缀必须使用 *"c"* -->

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