# JSP(Java Server Pages)简介

## 1.1 JSP简介

简单地说，JSP的作用就是使用JSP标签在HTML中嵌入java代码，这样做可以便于将后台获取的数据填入前端页面，也可以对后台取出的数据做简单的处理，服务器会对页面进行解析，将JSP解析成HTML页面再转发给客户端，它的功能和CGI相似，都是动态地生成页面(动态部分由嵌入的java编写，静态内容由HTML编写)，再转交给服务器，由服务器将页面发送给客户端

虽然JavaScript也可以动态地生成HTML内容，但显然，JSP更容易和服务器进行交互

## 1.2 JSP环境配置

JSP需要java SDK，只要安装好了JAVA编译环境，SDK就不需要我们操心了

JSP还需要一个web服务器Tomcat，tomcat的安装配置参见《servlet学习笔记》

JSP的运行还依赖servlet的包，因此还需要导入servlet的包，参见《servlet学习笔记》

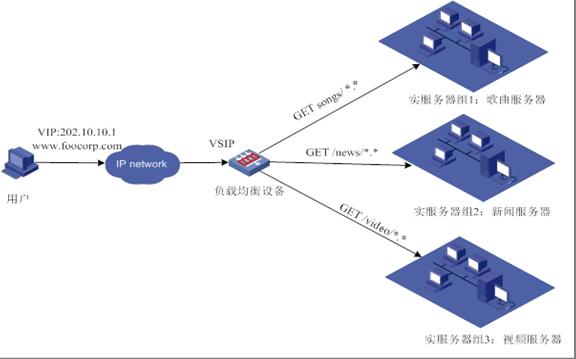
Tomcat既是servlet容器，也是web服务器，但我们也可以配置一个web服务器来替代Tomcat，推荐Nginx(不要使用apache那种老掉牙的东西)

下面简单介绍一下nginx服务器，niginx服务器是一款出现较晚的高性能服务器，它是一款HTTP和反向代理服务器，HTTP服务器我们都知道，就是支持HTTP协议的服务器，那么什么是反向代理？

在服务器的世界里，随着我们部署的项目越来越大，用户量越来越多，一台服务器已经无法满足我们的需求了，我们需要多台服务器帮助我们分担压力，这些服务器对外仍提供原来的服务，对于用户来说，感觉不到任何差别(唯一的差别就是不卡了)，但是却可以大幅提升项目的运行速度

但是，由于一般一个项目对外只暴露一个ip，因此，我们需要有一个公共的出入口，担任出入口的服务器，需要拥有将收到的HTTP请求转发出去的能力，这就是反向代理，反向代理常常用来实现负载均衡，这样一来，用户访问网站，解析网站域名得到的IP实际上是负载均衡服务器的IP，而不是所有服务器的ip，这样一来，加入/移除服务器只需要修改负载均衡服务器的服务器列表即可

下图是一个典型的分布式负载均衡架构

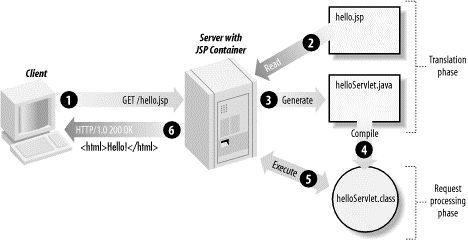


分布式意思是多台服务器做不同的事，将业务拆分为多个子业务交由多个服务器实现，而集群意为由多个服务器组合起来实现一项业务

## JSP的生命周期

了解一样新技术，首先要了解它的生命周期，接下来我们来探讨一下JSP的生命周期

1. 客户端发送HTTP请求给服务器
2. 服务器识别出这是一个对jsp网页的请求，并将该请求传递给JSP容器，如Tomcat，这一步通过访问URL或者访问jsp 文件来完成
3. JSP容器载入相应的jsp文件，并将jsp文件转化为servlet，这种转化只是简单地将所有模板文本改用 println() 语句，并且将所有的 JSP 元素转化成 Java 代码。
4. servlet引擎编译并执行得到的servlet，执行结果将产生HTML输出，servlet容器将输出嵌入到HTTP response 中上交给 Web 服务器
5. Web 服务器将动态生成的HTML 页面返回到客户端



以上是一次对JSP文件进行HTTP请求的全过程，那么JSP的生命周期又是如何的呢？

**编译阶段：**

servlet容器编译JSP源文件，生成servlet类，并将其编译为Java字节码文件(.class文件)

**初始化阶段：**

加载编译JSP生成的servlet类，创建其实例，并调用它的初始化方法

public void jspInit(){

// 初始化代码

}

如果需要自定义初始化，只需要重写jspInit()方法即可，一般在其中执行初始化数据库连接、打开文件和创建查询表等操作

**执行阶段：**

调用编译JSP所生成的servlet类的实例的服务方法

void \_jspService(HttpServletRequest request,HttpServletResponse response)

{

// 服务端处理代码

}

jspService()方法在每个request请求中只被调用一次并且负责产生与之相对应的response响应

**销毁阶段：**

调用编译JSP所生成的servlet类的实例的销毁方法，然后销毁servlet实例

public void jspDestroy()

{

// 清理代码

}

如果需要自定义销毁方法，只需要重写该方法即可，一般在其中执行如释放数据库连接或者关闭文件夹等操作

# JSP使用

## 2.1基础语法

### 2.1.1基本用法

JAVA使用**<% %>**符号来标明这是一个java脚本元素，任何文本、HTML标签、其他JSP元素必须写在脚本元素的外面。如：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"* contentType=*"text/html; charset=UTF-8"*%>

<html>

<head><title>Hello World</title></head>

<body>

Hello World!<br/>

<%out.println("Your IP address is " + request.getRemoteAddr());%>

</body>

</html>

### 2.1.2变量声明

JSP中使用的变量也需要声明

**语法：**

**<%! 声明语句 %>**

eg：

<%! **int** i = 0; %>

<%! Circle a = **new** Circle(2.0); %>

### 2.1.3表达式

一个JSP表达式中包含的JAVA语句表达式的值，先被转化成String，然后插入到表达式出现的地方，不需要我们print打印输出

**语法：**

**<%= 表达式 %>**

eg：

<p>

今天的日期是: <%= (**new** java.util.Date()).toLocaleString()%>

</p>

上述语句等价于

<p>

今天的日期是: <% out.print((**new** java.util.Date()).toLocaleString())%>

</p>

### 2.1.4 注释

一个JSP表达式中包含的JAVA语句表达式的值，先被转化成String，然后插入到表达式出现的地方，不需要我们print打印输出

**语法：**

**<%-- 该部分注释在网页中不会被显示--%>**

### 2.1.5 流程控制

JSP提供了完备的流程控制语法，使得我们可以在html页面中嵌入判断语句和循环语句

#### 2.1.5.1 if判断语句

**语法：**

**<% if (if条件1) { %>**

**<p>如果条件1成立，就显示本标签</p>**

**<% } else if(if条件2) { %>**

**<p>如果条件2成立，就显示本标签</p>**

**<% } else { %>**

**<p>否则显示本标签</p>**

**<% } %>**

其实语法部分和java没什么区别，只是要将其用**<% %>**包裹起来(因为jsp最终要转化为servlet执行)

jsp也是支持switch的，eg：

<%

**switch**(day) {

**case** 0:

out.println("星期天");

**break**;

**case** 1:

out.println("星期一");

**break**;

**case** 2:

out.println("星期二");

**break**;

**case** 3:

out.println("星期三");

**break**;

**case** 4:

out.println("星期四");

**break**;

**case** 5:

out.println("星期五");

**break**;

**default**:

out.println("星期六");

}

%>

#### 2.1.5.2 循环语句

在JSP程序中可以使用Java的三个基本循环类型：for，while，和 do…while。

**语法：**

**<% for (for条件) { %>**

**循环体**

**<% }%>**

**eg:**

<%! int fontSize; %>

//for循环

<%for ( fontSize = 1; fontSize <= 10; fontSize++){ %>

<font color="green" size="<%= fontSize %>">

我是循环体

</font><br />

<%}%>

//while循环

<%while ( fontSize <= 3){ %>

<font color="green" size="<%= fontSize %>">

菜鸟教程

</font><br />

<%fontSize++;%>

<%}%>

## 2.2 JSP指令

JSP指令用来设置整个JSP页面相关的属性，如网页的编码方式和脚本语言。它在编译阶段起作用

### 2.2.1<%@ page %>

头部的<%@ page %>定义网页本身的属性，eg：

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"utf-8"* contentType=*"text/html; charset=UTF-8"*%>

其中：

import 导入jsp中用到的包

pageEncoding 是告诉服务器当前JSP页面所使用的编码格式，方便服务器将JSP页面翻译成Servlet，而charset是服务器发送给浏览器用的文字编码

contentType 设置本页面文件的MIME类型，即本页面文件应当由哪一种类型的应用程序打开，如：application/msword代表这是一个.doc文件，即word文档；text/html意味着这是一个html页面

charset设置本页面的字符编码，浏览器将以该编码读取本文件

session 指定JSP页面是否使用session，默认值为true

isErrorPage 由于可以将某个JSP页面设置成错误处理页面来处理另一个JSP页面的错误，因此，该属性指定本页面**是否作为异常处理页面**

errorPage 设置当前页面的异常处理页面，对应的异常处理页面isErrorPage必须设置为true，如果设置该属性，那么在web.xml文件中定义的任何错误处理页面都将被忽略，优先使用该属性定义的异常处理页面。如果本页面出了异常，将全部交由errorPage指定的异常处理页面处理(该异常处理页面的isErrorPage必须为true)

### 2.2.1<%@ include %>

JSP可以通过include指令来包含其他文件。被包含的文件可以是JSP文件、HTML文件或文本文件。被包含的文件就好像是该JSP文件的一部分，会被嵌入该部位，组成一个全新的JSP，再编译成servlet执行。

**语法：**

**<%@ include file="文件相对 url 地址" %>**

## 2.3 JSP动作元素

与JSP指令不同，JSP动作元素在执行阶段起作用

所有的动作元素都有两个属性：id属性和scope属性。

**1.id属性：**

id属性是动作元素的唯一标识，可以在JSP页面中引用。动作元素创建的id值可以通过PageContext来调用。

**2.scope属性：**

该属性用于识别动作元素的生命周期。 id属性和scope属性有直接关系，scope属性定义了相关联id对象的寿命。 scope属性有四个可能的值： page, request, session, application。

page 默认值，表示该Bean只在当前页面内可用（保存在当前页面的PageContext内）。

request 表示该Bean在当前的客户请求内有效（保存在ServletRequest对象内）。

session 表示该Bean对当前HttpSession内的所有页面都有效。

application 表示该Bean对所有具有相同ServletContext的页面都有效。

### 2.3.1<jsp:include>

和前面的<%@ include %>指令不同，**<jsp:include>**是在本页面收到http请求时执行的，而前者是在编译阶段执行的，**<jsp:include>**是将被包含的jsp文件和本jsp文件分别编译成不同的servlet，再将被包含的jsp生成的servlet执行后的结果放入本jsp执行后的结果中，也就是说，如果<%@ include %>和**<jsp:include>**都包含一个jsp文件，前者只会将被包含的jsp先导入本jsp，然后编译成一个servlet执行，而后者是分别将jsp编译成两个servlet，将被包含的jsp文件执行结果放入本jsp中

这样做的话，**<jsp:include>**不止可以导入jsp文件，还可以导入servlet，只要是任何能执行并生成输出的文件都被**<jsp:include>**导入jsp中。并且，对于<%@ include %>，如果有其他程序在编译过程后修改了被包含的jsp，该修改不会生效，而**<jsp:include>**是即时修改，即时生效

**eg：**

**<jsp:include page="date.jsp" flush="true" />**

page 指定被包含的jsp的相对路径

flush 布尔属性，定义在包含资源前是否刷新缓存区。

## 2.4 JSP隐式对象

JSP隐式对象也被称为预定义变量(一共九个)。也即是可以在jsp页面任何部位使用而不用主动声明定义的变量，声明定义的过程jsp容器都帮我们做好了

|  |  |
| --- | --- |
| request | 请求作用域，就是**客户端的一次请求。** |
| response | HttpServletResponse类的实例 |
| out | JspWriter类的实例，用于**把结果输出至网页上** |
| session | **会话作用域**。生命周期：会话超时，或者服务器端强制使会话失效。 |
| application | ServletContext类的实例，**通过向application中添加属性，则所有组成本web应用的JSP文件都能访问到这些属性** |
| config | ServletConfig类的实例，不常用 |
| pageContext | 页面上下文对象，通过它可以获取JSP页面所有对象(其他8个)以及命名空间的访问 |
| page | page对象是当前页面转换后的Servlet类的实例，如果使用了动作元素include，则一个request可以包含多个page |
| Exception | Exception类的对象，代表发生错误的JSP页面中对应的异常对象 |

# JSP 标准标签库（JSTL）

JSTL实际上是把jsp常用的标签封装，简单地理解为换一种写法，既然只是换一种写法，那为什么要增加学习成本呢？

主要有以下原因：

1. jstl的标签可以无限次的重用，而jsp页面嵌入的java代码每次修改都必须重写
2. 简化页面，使页面的代码结构更清晰，增加可读性，便于前端进行页面修改
3. 更彻底地将数据与显示分离
4. 最主要的一点还是大家都在用(逃

其实用原生的jsp也是可以代替jstl的，但建议还是使用jstl，毕竟都在用

进入正题：

## 3.1 环境配置

只需要在maven项目的pom.xml中添加以下代码，然后maven install即可：

<dependency>  
 <groupId>jstl</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
</dependency>  
<dependency>  
 <groupId>taglibs</groupId>  
 <artifactId>standard</artifactId>  
 <version>1.1.2</version>  
</dependency>

这是将包导入项目，如果在页面中使用的话，用哪个标签库导入哪个即可，jstl一般有如下标签库：

<%@ **taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> <!--输出,条件,迭代等核心标签库-->  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="fmt"%> <!--数据格式化标签库-->  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="sql"%> <!--数据库相关标签库-->  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/core" prefix="fn"%> <!--常用函数标签库-->  
<%@ **page** isELIgnored="false"%> <!--支持EL表达式，不设的话，EL表达式不会解析-->

## 3.2 核心标签

在使用核心标签之前，需要在页面头部加上：

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

核心标签库中有如下标签：

**1.<c:out>**

用于在JSP中显示数据，就像<%= ... >，它只用于输出数据，value属性是要显示的值，default是默认值，escapeXml是否忽略xml特殊字符，如果为true，则不处理xml字符，为false则处理，eg：

下例会显示”你好啊”

<h1><**c:out** value="&lt你好啊" default="我是默认值" escapeXml="false"/></h1>

下例会显示myParam的变量值(要设置变量值才能显示)

<h1><**c:out** value="**${**myParam**}**" default="我是默认值" escapeXml="true"/></h1>

**2.<c:set>**

用于保存数据，这个标签之所以很有用呢，是因为它会计算表达式的值，然后使用计算结果来设置 JavaBean 对象或 java.util.Map 对象的值

var设置变量名，scope参见2.3节，value要存储的值

<**c:set** var="salary" scope="session" value="**${**2000\*2**}**"/>  
<**c:out** value="**${**salary**}**"/>

输出4000

**3.<c:remove>**

用于删除数据，这个标签不是特别有用，不过可以用来确保JSP完成清理工作。

**4.<c:catch>**

用来处理产生错误的异常状况，并且将错误信息储存到变量中

<**c:catch** var="exception">  
 ...  
</**c:catch**>

**5.<c:if>**

与我们在一般程序中用的if一样

<**c:if** test="<boolean>" var="<string>" scope="<string>">  
 ...  
</**c:if**>

**6.<c:choose>**

本身只当做<c:when>和<c:otherwise>的父标签，也就是switch语句

**<c:when>**

<c:choose>的子标签，用来判断条件是否成立

**<c:otherwise>**

<c:choose>的子标签，接在<c:when>标签后，当所有<c:when>标签判断为false时被执行

eg：

<c:choose>

<c:when test="<boolean>">

...

</c:when>

<c:when test="<boolean>">

...

</c:when>

...

...

<c:otherwise>

...

</c:otherwise>

</c:choose>

**7.<c:import>**

将一张页面的所有内容导入本页面，这一张页面可以是一个URL也可以是相对路径和绝对路径

<**c:import** var="baidu" url="http://www.baidu.com"/>  
<**c:import** var="hello" url="../../index.jsp"/>  
<**c:out** value="**${**hello**}**"/>  
<**c:out** value="**${**baidu**}**"/>

**8.<c:forEach>**

基础迭代标签，接受多种集合类型

<**c:set** var="salary" scope="session" value="**${**[1,3,5,7,9,11,13]**}**"/>  
<**c:forEach** items="**${**salary**}**" var="sa" begin="1" end="3">  
 <div><h3>**${**sa**}**</h3></div>  
</**c:forEach**>

输出：3 5 7

**9.<c:forTokens>**

根据指定的分隔符来分隔内容并迭代输出

<**c:forTokens** items="google,runoob,taobao" delims="," var="name">  
 <**c:out** value="**${**name**}**"/>  
</**c:forTokens**>

**10.<c:param>**

用来给包含或重定向的页面传递参数

**11.<c:redirect>**

重定向至一个新的URL

**12.<c:url>**

使用可选的查询参数来创造一个URL

## 3.3 核心标签

**1.<fmt:formatNumber>**

使用指定的格式或精度格式化数字

**2.<fmt:parseNumber>**

解析一个代表着数字，货币或百分比的字符串

**3.<fmt:formatDate>**

使用指定的风格或模式格式化日期和时间

**4.<fmt:parseDate>**

解析一个代表着日期或时间的字符串

**5.<fmt:bundle>**

绑定资源

**6.<fmt:setLocale>**

指定地区

**7.<fmt:setBundle>**

绑定资源

**8.<fmt:timeZone>**

指定时区

**9.<fmt:setTimeZone>**

指定时区

**10.<fmt:message>**

显示资源配置文件信息

**11.<fmt:requestEncoding>**

设置request的字符编码