**JSP 教程**

JSP 与 PHP、ASP、ASP.NET 等语言类似，运行在服务端的语言。

JSP（全称Java Server Pages）是由 Sun Microsystems 公司倡导和许多公司参与共同创建的一种使软件开发者可以响应（me：动态）客户端请求，而动态生成 HTML、XML 或其他格式文档的Web网页的技术标准。

JSP 技术是以 Java 语言作为脚本语言的，JSP 网页为整个服务器端的 Java 库单元提供了一个接口来服务于HTTP的应用程序。

JSP文件后缀名为 **\*.jsp** 。

JSP开发的WEB应用可以跨平台使用，既可以运行在 Linux 上也能运行在 Windows 上。

**第一个 JSP 程序**

语言学习入门的第一个程序一般都是输出 "Hello World"，JSP输出 "Hello World" 代码如下所示：

<html>

<head>

<title>第一个 JSP 程序</title>

</head>

<body>

<%

out.println("Hello World！");

%>

</body>

</html>

**开始学习 JSP**

了解了 JSP 的基本概念后，现在让我们[开始来学习 JSP](http://www.runoob.com/jsp/jsp-intro.html)吧。

# JSP 简介

## 什么是Java Server Pages?

JSP全称Java Server Pages，是一种动态网页开发技术。它使用JSP标签在HTML网页中插入Java代码。标签通常以<%开头以%>结束。

JSP是一种Java servlet（java小服务，类似C#的ASP），主要用于实现Java web应用程序的用户界面部分。网页开发者们通过结合HTML代码、XHTML代码、XML元素以及嵌入JSP操作和命令来编写JSP。

JSP通过网页表单获取用户输入数据、访问数据库及其他数据源，然后动态地创建网页。

JSP标签有多种功能，比如访问数据库、记录用户选择信息、访问JavaBeans组件等，还可以在不同的网页中传递控制信息和共享信息。

## 为什么使用JSP？

JSP程序与CGI（me：php-cgi）程序有着相似的功能，但和CGI程序相比，JSP程序有如下优势：

* 性能更加优越，因为JSP可以直接在HTML网页中动态嵌入元素而不需要单独引用CGI文件。
* 服务器调用的是已经编译好的JSP文件，而不像CGI/Perl那样必须先载入解释器和目标脚本。
* JSP 基于Java Servlet API，因此，JSP拥有各种强大的企业级Java API，包括JDBC，JNDI，EJB，JAXP等等。
* JSP页面可以与处理业务逻辑的 Servlet 一起使用，这种模式被Java servlet 模板引擎所支持。

最后，JSP是Java EE不可或缺的一部分，是一个完整的企业级应用平台。这意味着JSP可以用最简单的方式来实现最复杂的应用。

## JSP的优势

以下列出了使用JSP带来的其他好处：

* 与ASP相比：JSP有两大优势。首先，动态部分用Java编写，而不是VB或其他MS专用语言，所以更加强大与易用。第二点就是JSP易于移植到非MS平台上。
* 与纯 Servlet 相比：JSP可以很方便的编写或者修改HTML网页而不用去面对大量的println语句。
* 与SSI相比：SSI无法使用表单数据、无法进行数据库链接。
* 与JavaScript相比：虽然JavaScript可以在客户端动态生成HTML，但是很难(ajax完全可以与服务器端交互)与服务器交互，因此不能提供复杂的服务，比如访问数据库和图像处理等等。
* 与静态HTML相比：静态HTML不包含动态信息。

## 接下来呢？

我们将会带您一步一步地来搭建JSP运行环境，这需要有一定的Java基础。

如果您还未学过Java，可以先学习我们为您提供的[Java教程](http://www.runoob.com/java/java-tutorial.html)。

**JSP 开发环境搭建**

JSP开发环境是您用来开发、测试和运行JSP程序的地方。

本节将会带您搭建JSP开发环境，具体包括以下几个步骤。

如果你使用的是 Eclipse 环境，可以直接参阅：[Eclipse JSP/Servlet 环境搭建](http://www.runoob.com/jsp/eclipse-jsp.html)。

**配置Java开发工具（JDK）**

这一步涉及Java SDK的下载和PATH环境变量的配置。

您可以从Oracle公司的Java页面中下载SDK：[Java SE Downloads](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

Java SDK下载完后，请按照给定的指示来安装和配置SDK。最后，通过设置PATH和JAVA\_HOME环境变量来指明包括java和javac的文件夹路径，通常是java\_install\_dir/bin和java\_install\_dir。

假如您用的是Windows系统并且SDK的安装目录为C::\jdk1.5.0\_20，那么您就需要在 C:\autoexec.bat 文件中添加以下两行：

set PATH=C:\jdk1.5.0\_20\bin;%PATH%

set JAVA\_HOME=C:\jdk1.5.0\_20

或者，在Windows NT/2000/XP下，您可以直接右击我的电脑图标，选择属性，然后高级，然后环境变量，接下来您就可以很方便地设置PATH变量并且确定退出就行了。

在Linux/Unix系统下，如果SDK的安装目录为/usr/local/jdk1.5.0\_20并且使用的是C shell，那么您就需要在.cshrc文件中添加以下两行：

setenv PATH /usr/local/jdk1.5.0\_20/bin:$PATH

setenv JAVA\_HOME /usr/local/jdk1.5.0\_20

或者，假如您正在使用类似于Borland JBuilder、Eclipse、IntelliJ IDEA和Sun ONE Studio这样的集成开发环境，可以试着编译并运行一个简单的程序来确定IDE（集成开发环境）是否已经知道 SDK的安装目录。

本步骤你也可以参考本站[Java开发环境配置](http://www.runoob.com/java/java-environment-setup.html)章节的教程。

**设置Web服务器：Tomcat**

目前，市场上有很多支持JSP和Servlets开发的Web服务器。他们中的一些可以免费下载和使用，Tomcat就是其中之一。

Apache Tomcat是一个开源软件，可作为独立的服务器来运行JSP和Servlets，也可以集成在 Apache Web Server中。以下是Tomcat的配置方法：

* 下载最新版本的Tomcat：<http://tomcat.apache.org/>。
* 下载完安装文件后，将压缩文件解压到一个方便的地方，比如Windows下的C:\apache-tomcat-5.5.29目录或者Linux/Unix下的/usr/local/apache-tomcat-5.5.29目录，然后创建CATALINA\_HOME环境变量指向这些目录。

在Windows机器下，Tomcat可以通过执行以下命令来启动：

%CATALINA\_HOME%\bin\startup.bat

或者

C:\apache-tomcat-5.5.29\bin\startup.bat

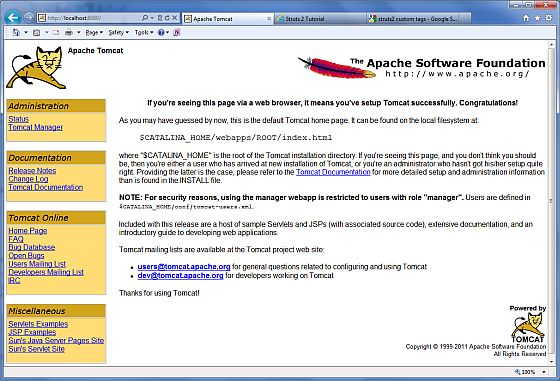
在Linux/Unix机器下，Tomcat可以通过执行以下命令来启动：

$CATALINA\_HOME/bin/startup.sh

或者

/usr/local/apache-tomcat-5.5.29/bin/startup.sh

成功启动Tomcat后，通过访问http://localhost:8080/便可以使用Tomcat自带的一些web应用了。假如一切顺利的话，您应该能够看到以下的页面：



更多关于配置和运行Tomcat的信息可以在Tomcat提供的文档中找到，或者去Tomcat官网查阅：http://tomcat.apache.org。

在Windows机器下，Tomcat可以通过执行以下命令来停止：

%CATALINA\_HOME%\bin\shutdown

或者

C:\apache-tomcat-5.5.29\bin\shutdown

在Linux/Unix机器下，Tomcat可以通过执行以下命令来停止：

$CATALINA\_HOME/bin/shutdown.sh

或者

/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh

**设置CLASSPATH环境变量**

由于servlets不是Java SE的一部分，所以您必须标示出servlet类的编译器。

假如您用的是Windows机器，您需要在C:\autoexec.bat文件中添加以下两行：

set CATALINA=C:\apache-tomcat-5.5.29

set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%

或者，在Windows NT/2000/XP下，您只要右击我的电脑，选择属性，然后点击高级，然后点击环境变量，接下来便可以设置CLASSPATH变量并且确定退出即可。

在Linux/Unix机器下，假如您使用的是C shell，那么您就需要在.cshrc文件中添加以下两行：

setenv CATALINA=/usr/local/apache-tomcat-5.5.29

setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH

注意：如果您的开发路径是C:\JSPDev (Windows)或者 /usr/JSPDev (Linux/Unix)，那么您就需要将这些路径添加进CLASSPATH变量中。

# Eclipse JSP/Servlet 环境搭建

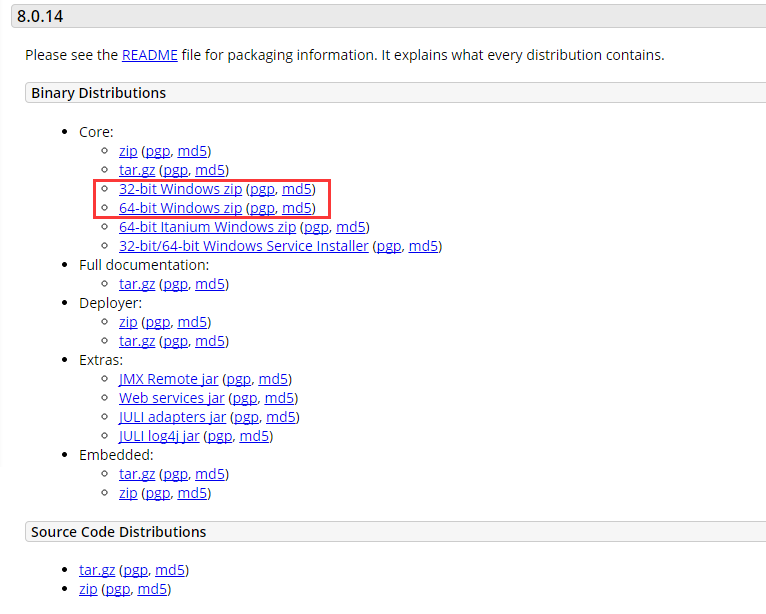
本文假定你已安装了 JDK 环境，如未安装，可参阅 [Java 开发环境配置](http://www.runoob.com/java/java-environment-setup.html)。

我们可以使用 Eclipse 来搭建 JSP 开发环境，首先我们分别下载一下软件包：

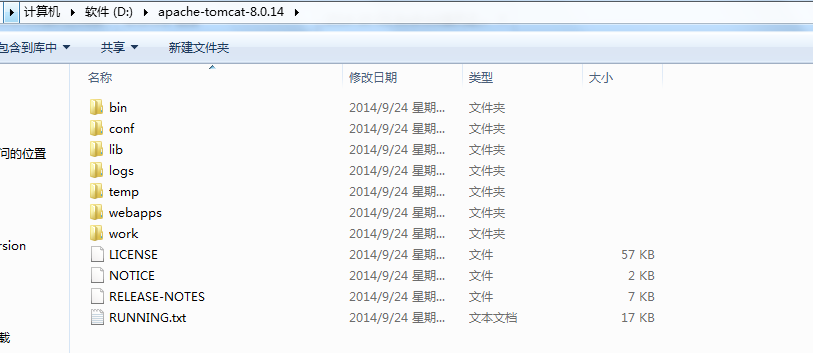
* **Eclipse J2EE：**<http://www.eclipse.org/downloads/>
* **Tomcat：**<http://tomcat.apache.org/download-70.cgi>

## Tomcat 下载安装

你可以根据你的系统下载对应的包(以下以Window系统为例)：



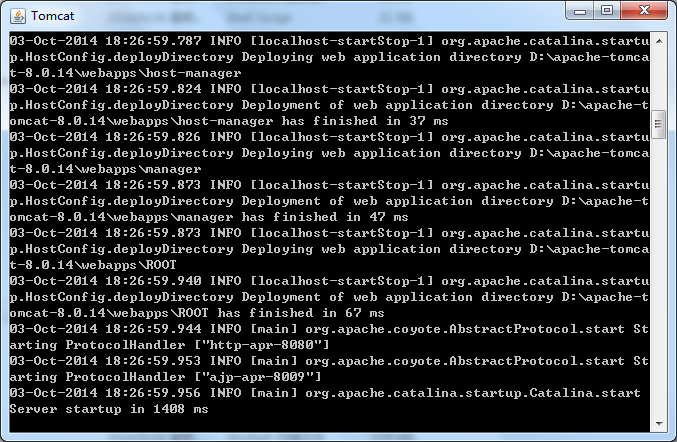
下载之后，将压缩包解压到D盘（你可以自己选择）：



注意目录名不能有中文和空格。目录介绍如下：

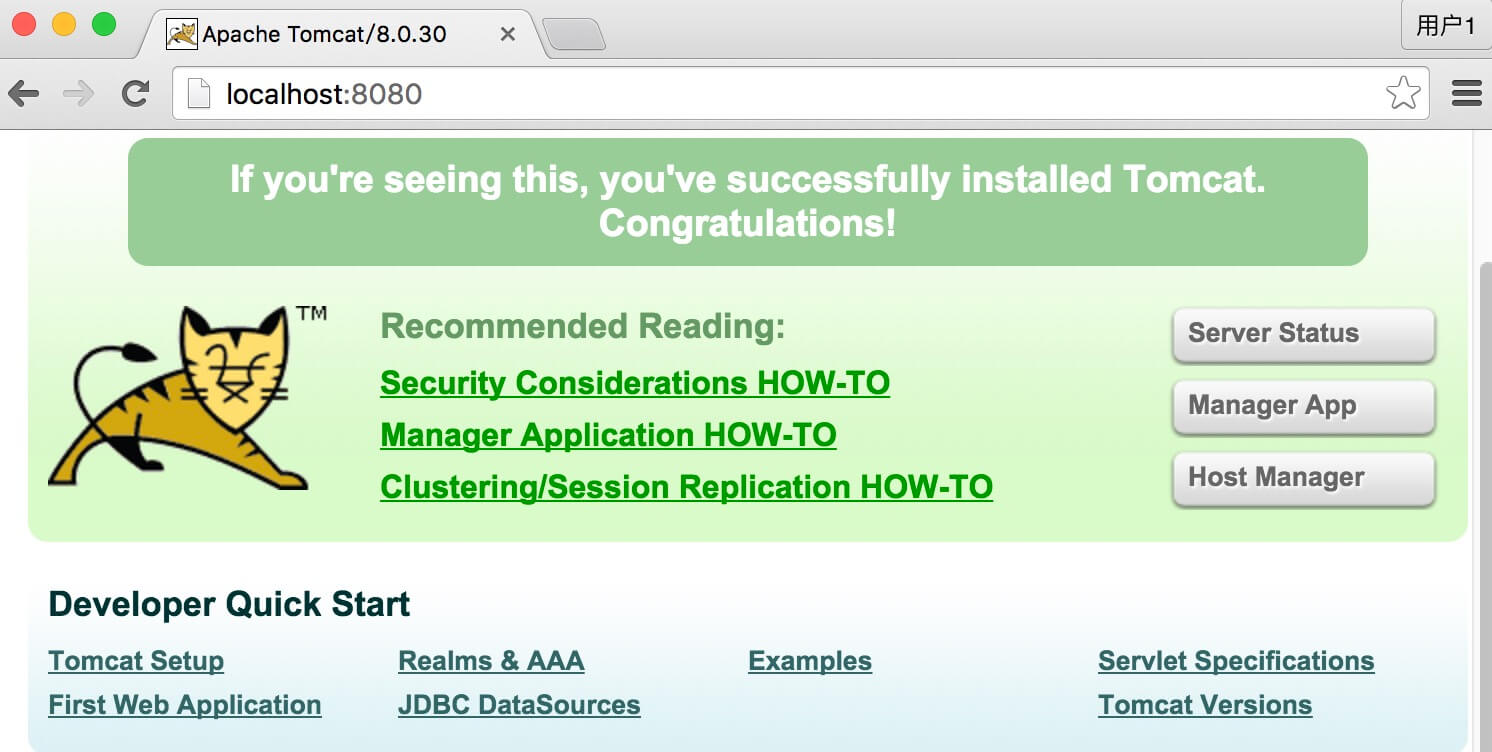
* bin：二进制执行文件。里面最常用的文件是**startup.bat**，如果是 Linux 或 Mac 系统启动文件为 **startup.sh**。
* conf:配置目录。里面最核心的文件是**server.xml**。可以在里面改端口号等。默认端口号是8080，也就是说，此端口号不能被其他应用程序占用。
* lib：库文件。tomcat运行时需要的jar包所在的目录
* logs：日志
* temp：临时产生的文件，即缓存
* webapps：web的应用程序。**web应用放置到此目录下浏览器可以直接访问**
* work：编译以后的class文件。

接着我们可以双击 startup.bat 启动 Tomcat，弹出如下界面：



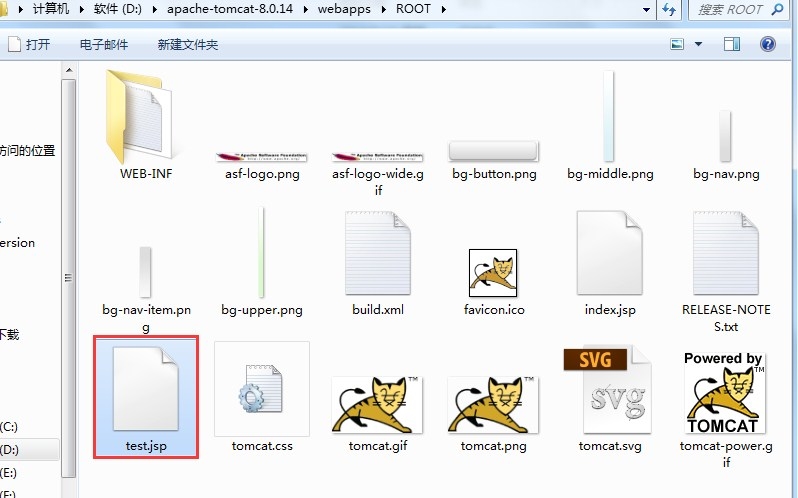
这个时候，本地的服务器就已经搭建起来了。如果想关闭服务器，可以直接关闭上面的窗口，或者在里面输入Ctrl+C禁止服务。

接着我们在浏览器中输入 **http://localhost:8080/**，如果弹出如下界面，表示tomcat安装成功并且启动起来了：



我们现在在浏览器上测试一下它吧：

首先在D:\apache-tomcat-8.0.14\webapps\ROOT目录中新建一个jsp文件：



test.jsp 文件代码如下：

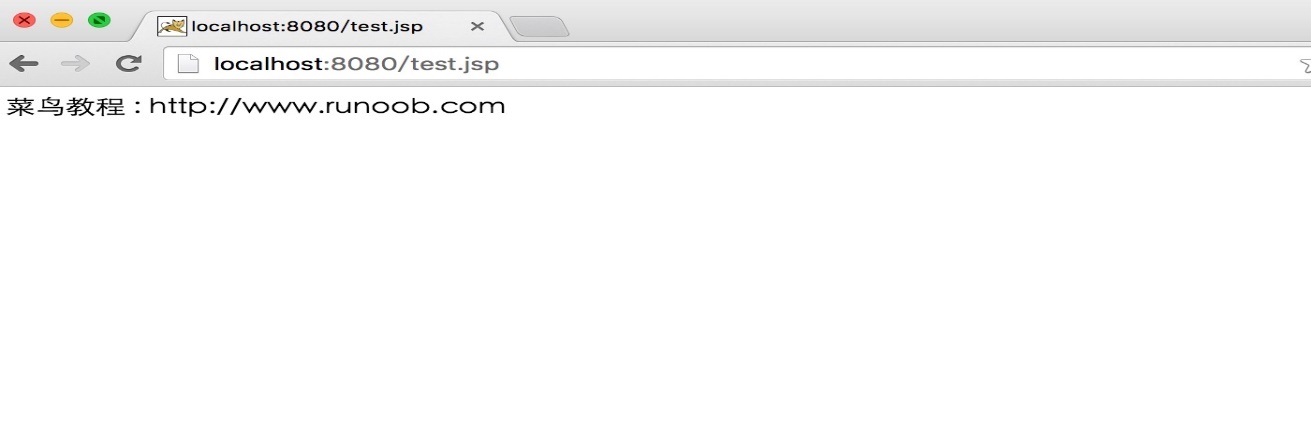
<%@ page contentType="text/html;charset=UTF-8" %>

<%

out.print("菜鸟教程 : http://www.runoob.com");

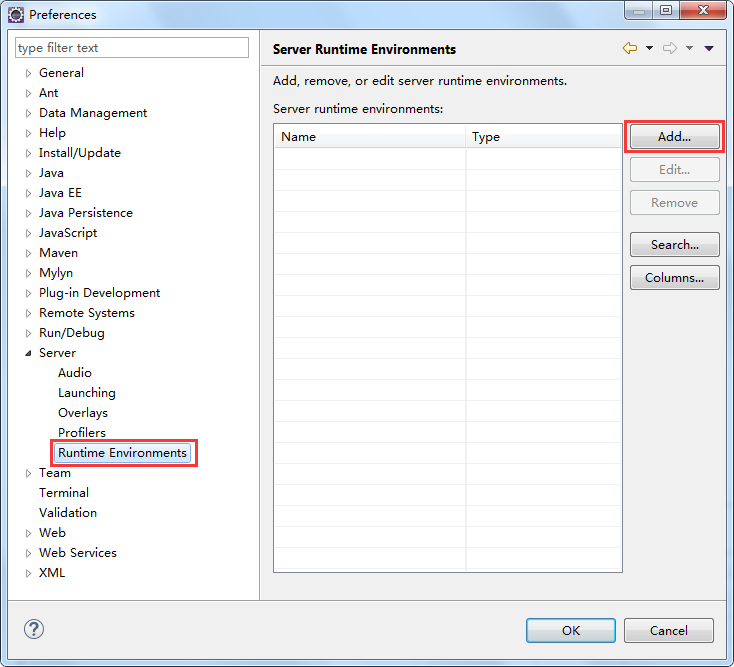
%>

接着在浏览器中访问地址 **http://localhost:8080/test.jsp**, 输出结果如下：

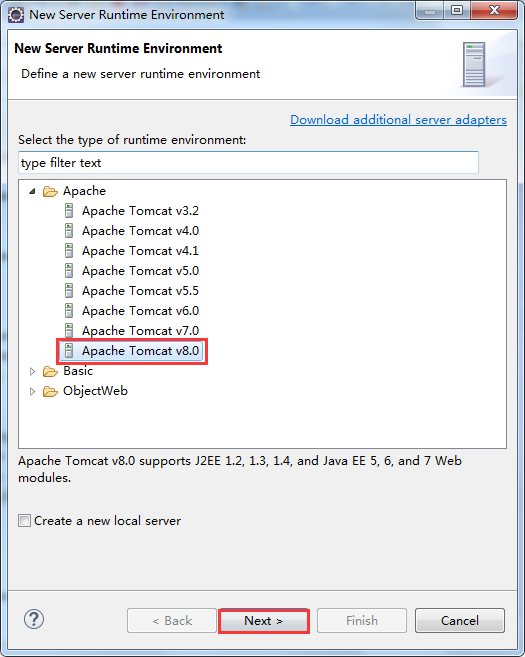


## 将 Tomcat 和 Eclipse 相关联

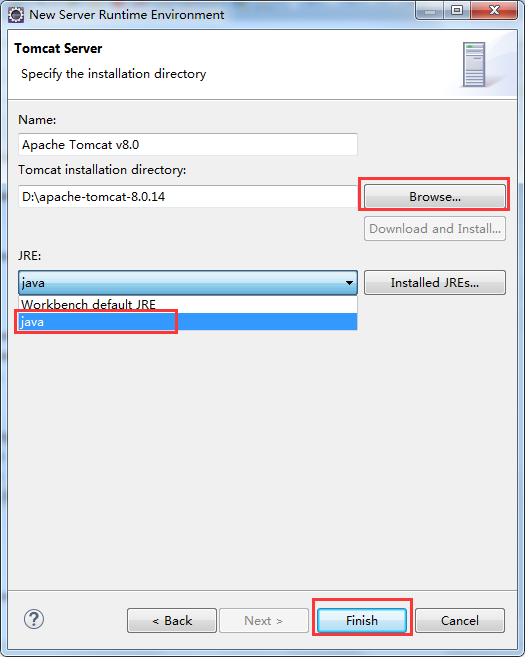
Eclipse J2EE下载后，解压即可使用，我们打开Java EE ，选择菜单栏Windows-->preferences（Mac 系统为 Eclipse-->偏好设置），弹出如下界面：



上图中，点击"add"的添加按钮，弹出如下界面：



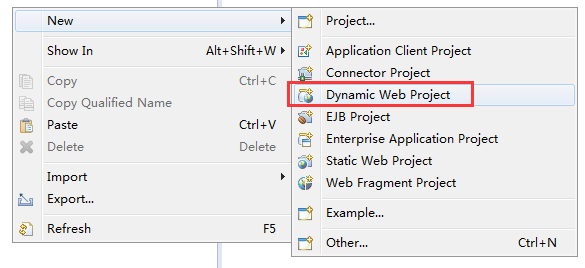
在选项中，我们选择对应的 Tomcat 版本，接着点击 "Next"，选择 Tomcat 的安装目录，并选择我们安装的 Java 环境：

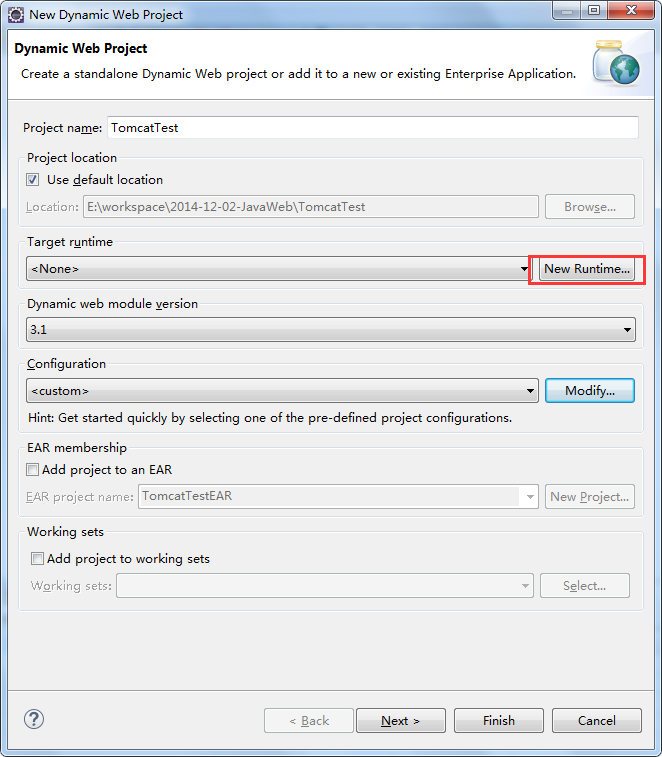


点击 "Finish"，完成配置。

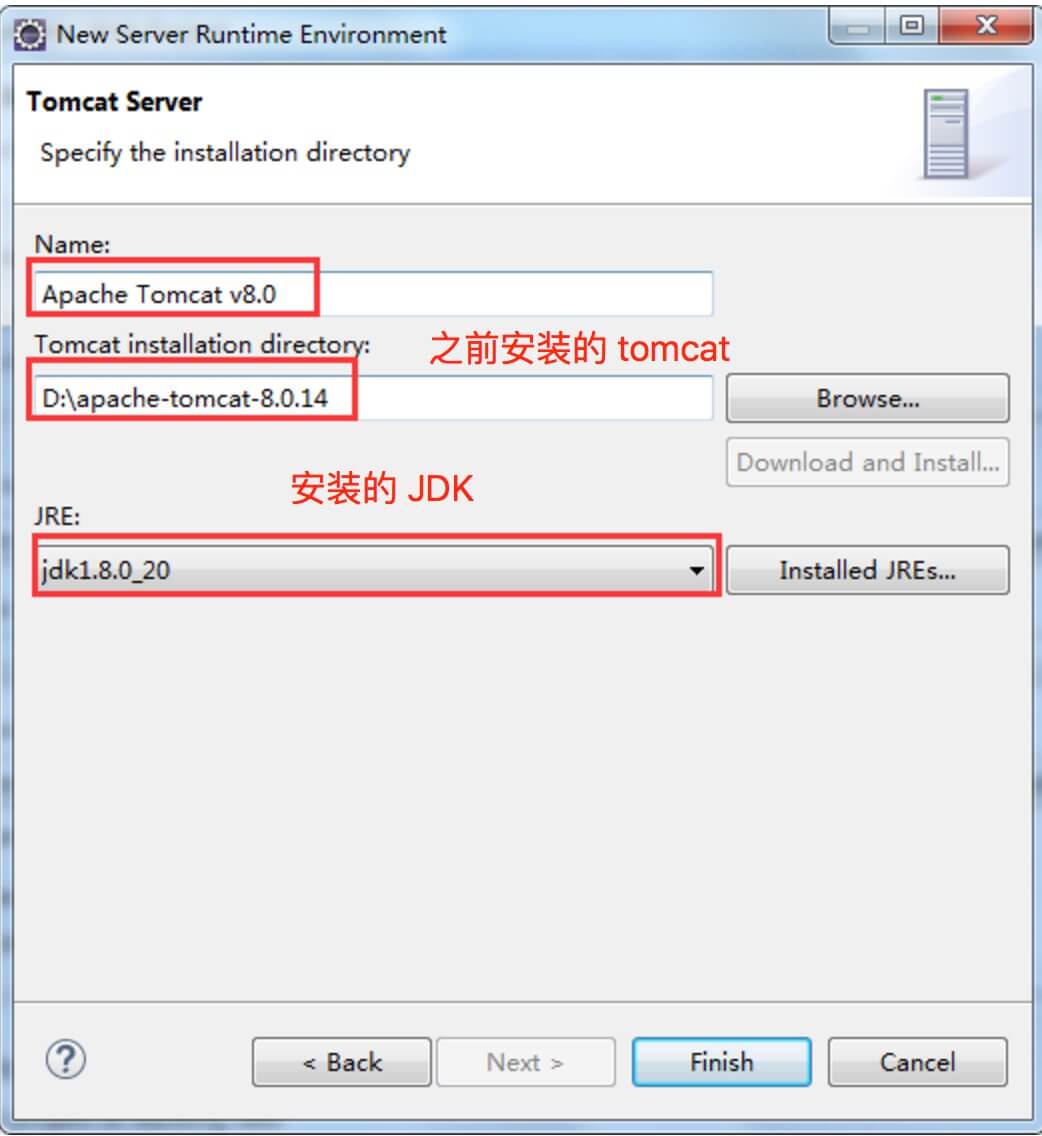
### 创建实例

选择 "File-->New-->Dynamic Web Project"，创建 TomcatTest 项目：



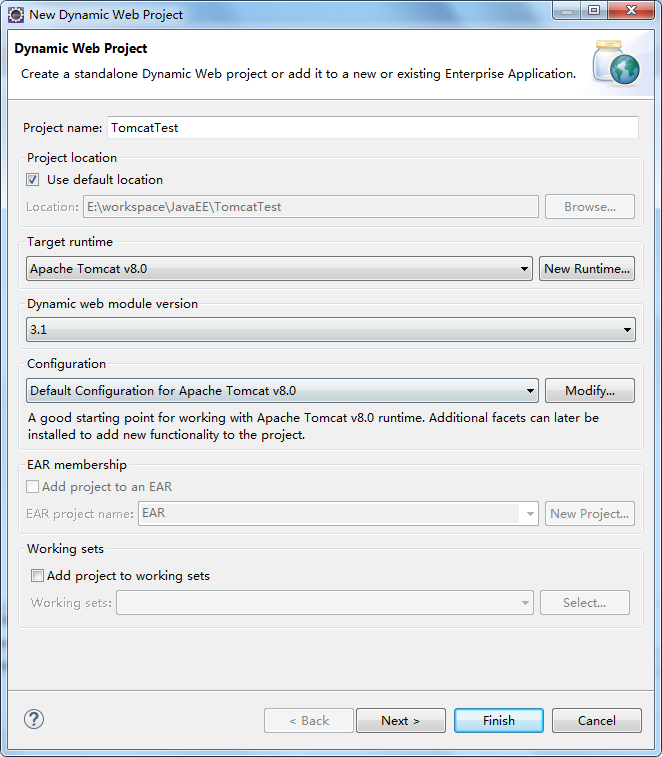


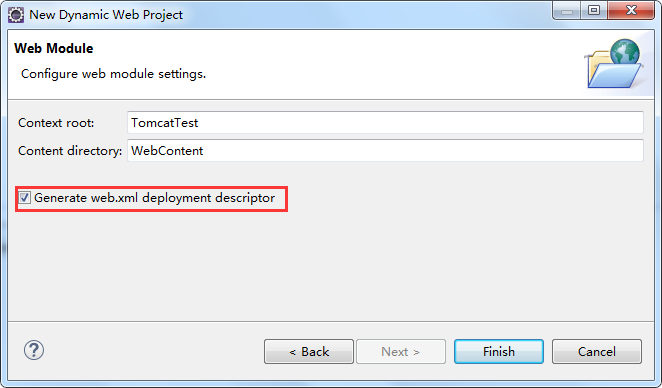
点开上图中的红框部分，弹出如下界面：



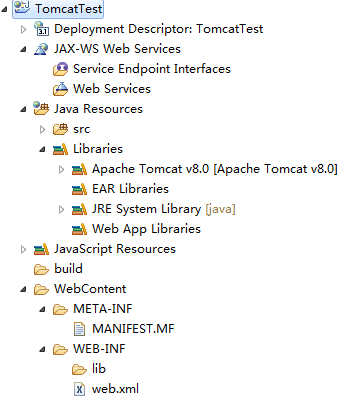
注意如果已默认选择了我们之前安装的 Tomcat 和 JDK 则可跳过此步。

然后，单击finish, 继续：





工程文件结构：



上图中各个目录解析：

* deployment descriptor：部署的描述。
* Web App Libraries：自己加的包可以放在里面。
* build：放入编译之后的文件。
* WebContent:放进写入的页面。

在WebContent文件夹下新建一个test.jsp文件。在下图中可以看到它的默认代码：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

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

<title>Insert title here</title>

</head>

<body>

</body>

</html>

接着我们修改下test.jsp文件代码如下所示：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

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

<title>菜鸟教程</title>

</head>

<body>

<%

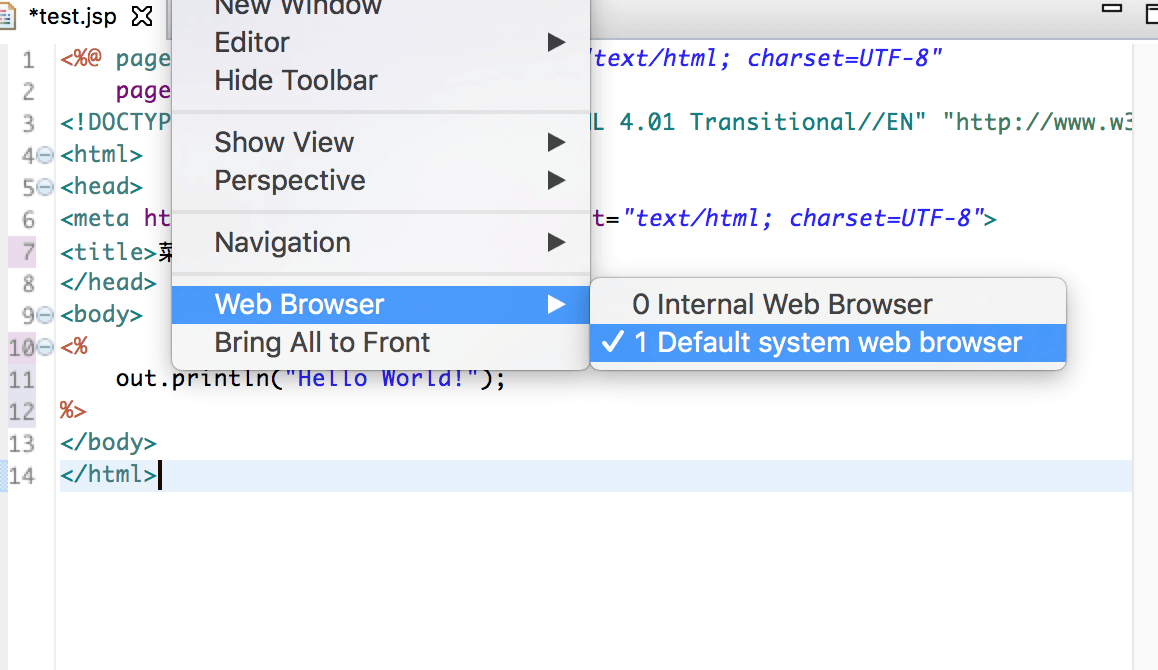
out.println("Hello World!");

%>

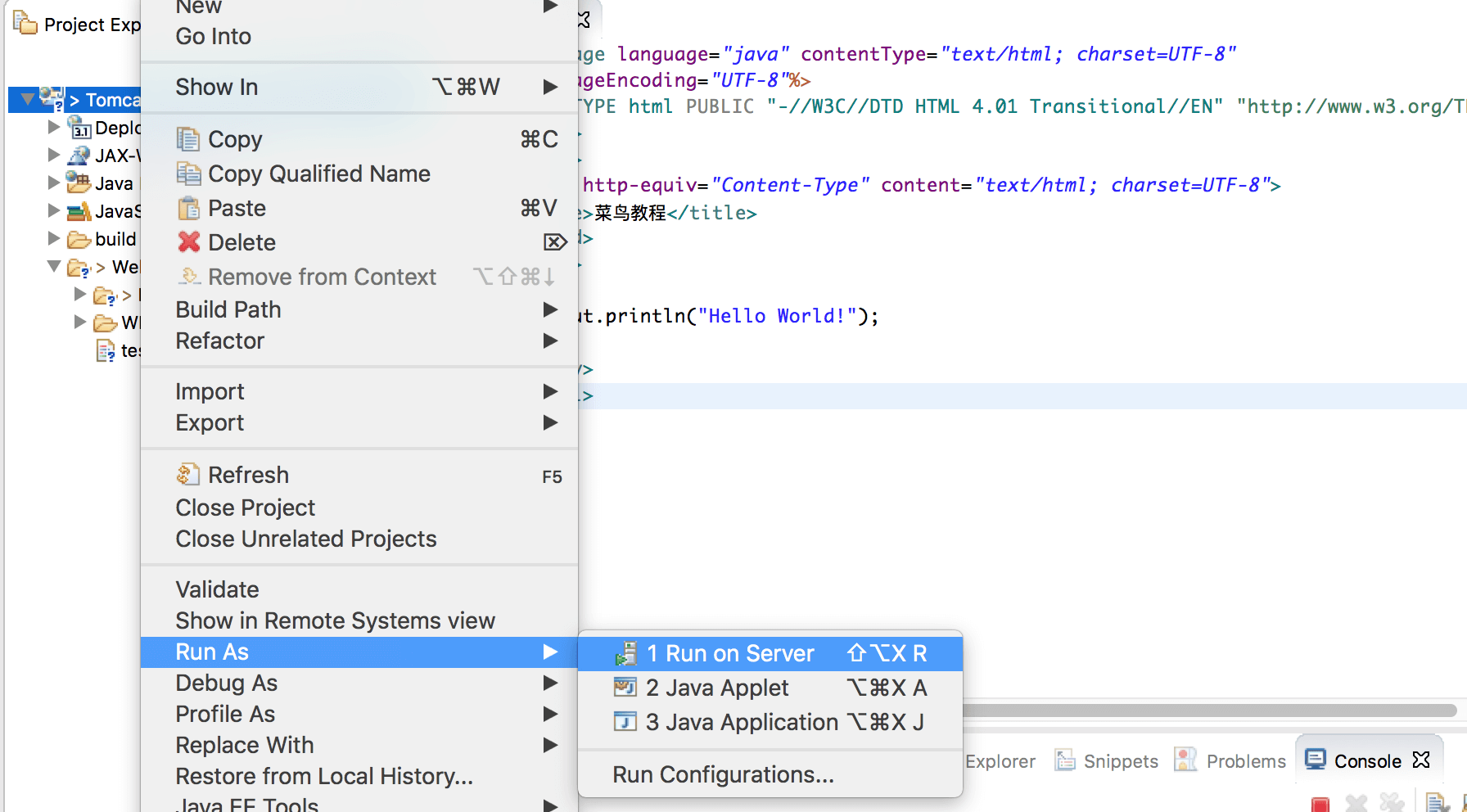
</body>

</html>

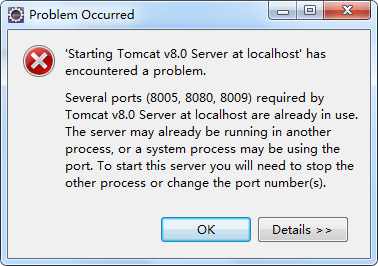
程序运行之前，我们先修改一下浏览器选项:



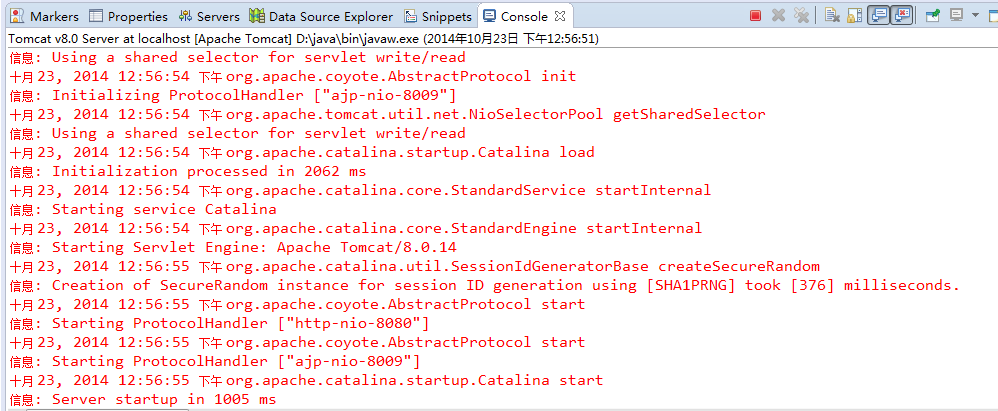
接着我们运行该项目:



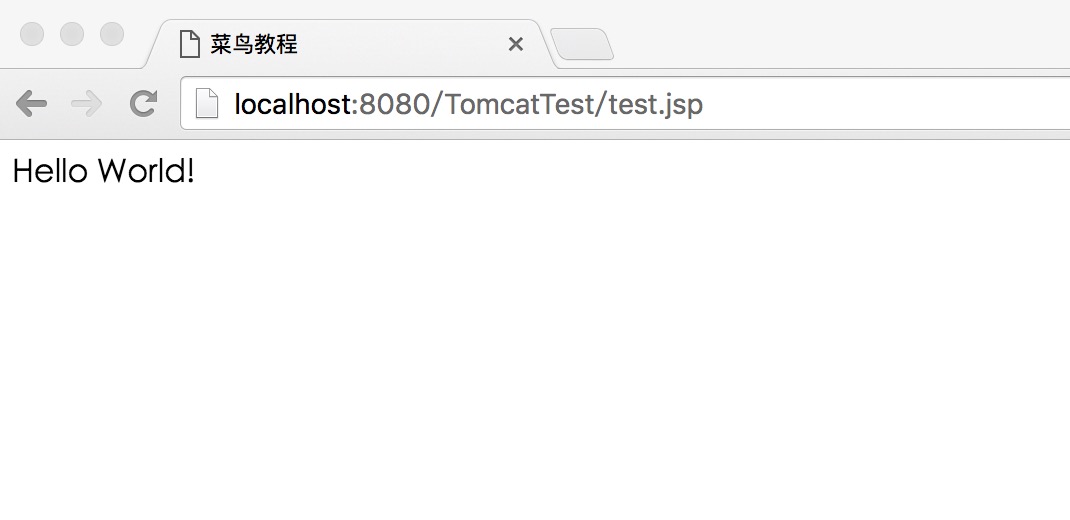
运行时，弹出如下错误：(如果没有此错误，请忽略)



原因是，我们之前点击了Tomcat安装包中的​startup.bat，这样一来就手动打开了Tomcat服务器，这明显是多余的，因为程序运行时，eclipse会自动开启Tomcat服务器。所以我们先手动关掉tomcat软件，再次运行程序，就行了。控制台信息如下：

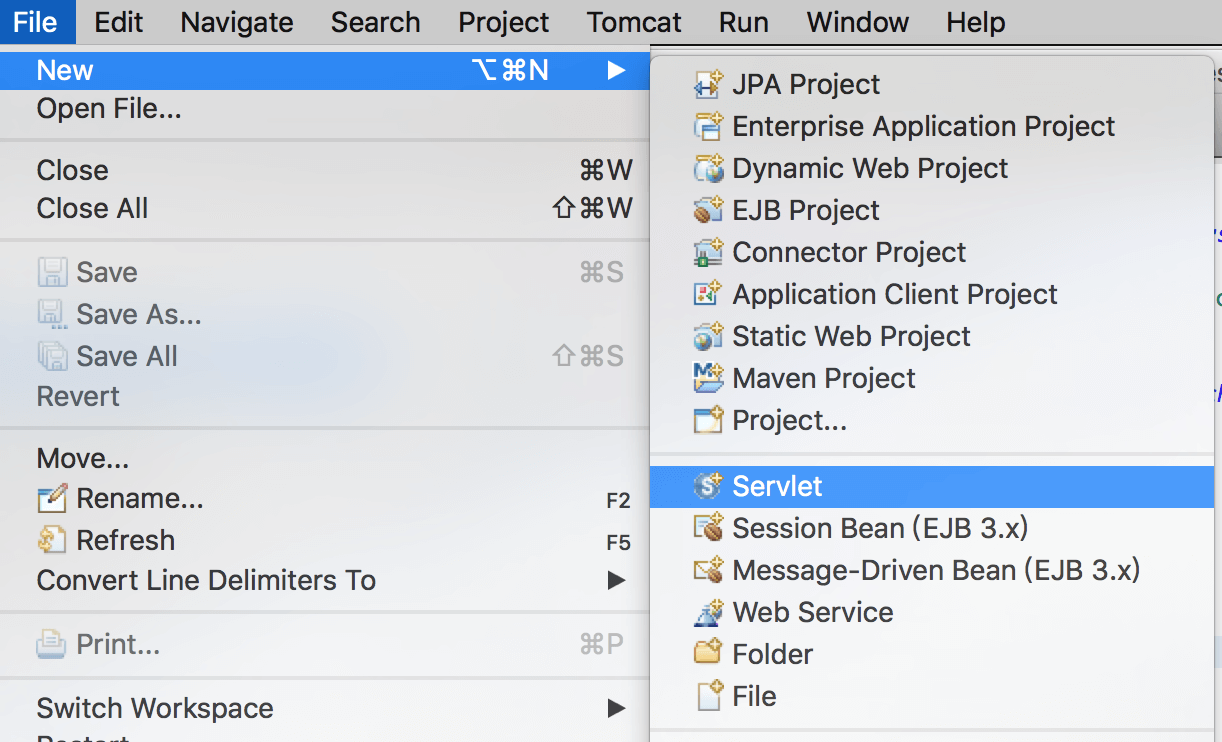


浏览器访问 **http://localhost:8080/TomcatTest/test.jsp**, 即可输出正常结果：

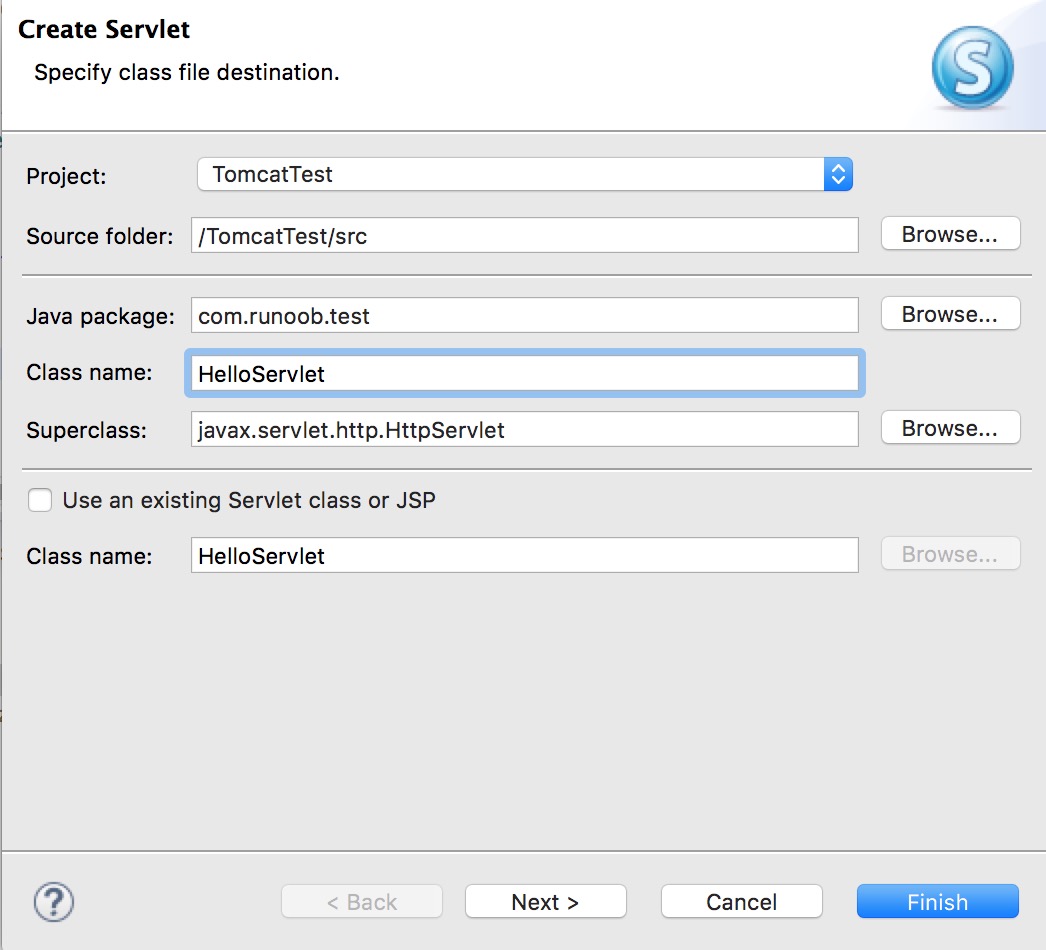


## Servlet 实例创建

我们也可以使用以上环境创建 Servlet 文件，选择 "File-->New-->Servlet":



位于 TomcatTest项目的 /TomcatTest/src 目录下创建 "HelloServlet" 类，包为 "com.runoob.test":



HelloServlet.java 代码如下所示：

package com.runoob.test;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class HelloServlet

\*/

@WebServlet("/HelloServlet")

public class HelloServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public HelloServlet() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// 使用 GBK 设置中文正常显示

response.setCharacterEncoding("GBK");

response.getWriter().write("菜鸟教程：http://www.runoob.com");

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}

创建 /TomcatTest/WebContent/WEB-INF/web.xml 文件（如果没有），代码如下所示：

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<servlet>

<!-- 类名 -->

<servlet-name>HelloServlet</servlet-name>

<!-- 所在的包 -->

<servlet-class>com.runoob.test.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

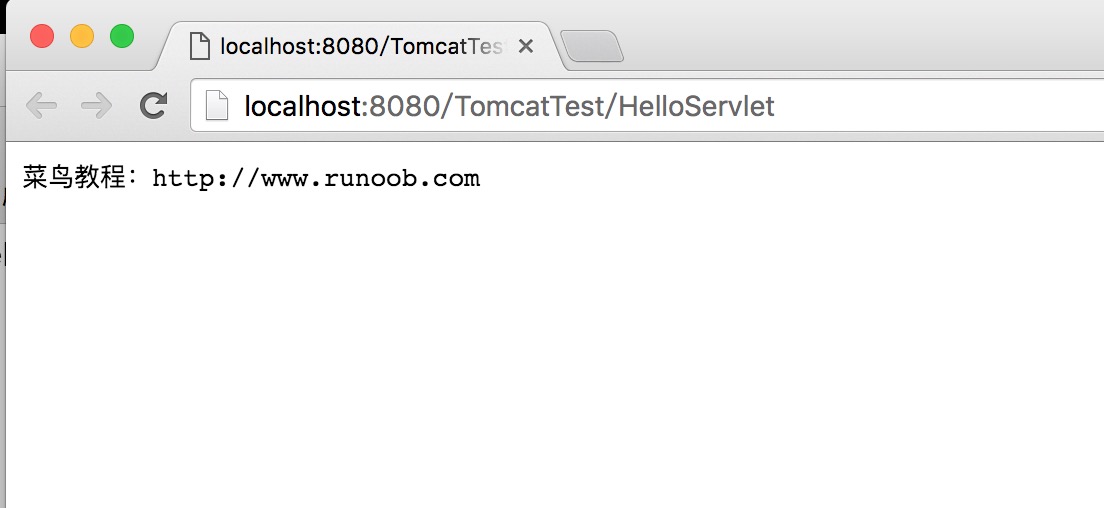
<!-- 访问的网址 -->

<url-pattern>/TomcatTest/HelloServlet</url-pattern>

</servlet-mapping>

</web-app>

接着重启 Tomcat，浏览器访问 **http://localhost:8080/TomcatTest/HelloServlet**：



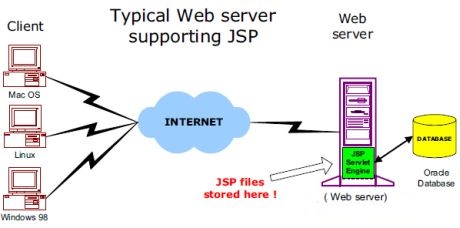
参考文章：http://www.cnblogs.com/smyhvae/p/4046862.html

# JSP 结构

网络服务器需要一个JSP引擎，也就是一个容器（me：web有web容器，spring有spring的管理容器，ejb有ejb容器等；容器就使用来封装组件的）来处理JSP页面。容器负责截获对JSP页面的请求。本教程使用内嵌JSP容器的Apache来支持JSP开发。

JSP容器与Web服务器协同合作，为JSP的正常运行提供必要的运行环境和其他服务（me：类似apache和php的组合，但解析php文件的时候会将请求转到php-cgi来处理，并将处理结果返回），并且能够正确识别专属于JSP网页的特殊元素。

下图显示了JSP容器和JSP文件在Web应用中所处的位置。

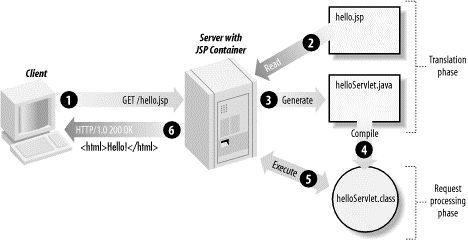


## JSP处理

以下步骤表明了Web服务器是如何使用JSP来创建网页的：

* 就像其他普通的网页一样，您的浏览器发送一个HTTP请求给服务器。
* Web服务器识别出这是一个对JSP网页的请求，并且将该请求传递给JSP引擎。通过使用URL或者.jsp文件来完成。
* JSP引擎从磁盘中载入JSP文件，然后将它们转化为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。
* JSP引擎将servlet编译成可执行类，并且将原始请求传递给servlet引擎。
* Web服务器的某组件将会调用servlet引擎，然后载入并执行servlet类。在执行过程中，servlet产生HTML格式的输出并将其内嵌于HTTP response中上交给Web服务器。
* Web服务器以静态HTML网页的形式将HTTP response返回到您的浏览器中。
* 最终，Web浏览器处理HTTP response中动态产生的HTML网页，就好像在处理静态网页一样。

以上提及到的步骤可以用下图来表示：



一般情况下，JSP引擎会检查JSP文件对应的servlet是否已经存在，并且检查JSP文件的修改日期是否早于servlet。如果JSP文件的修改日期早于对应的servlet，那么容器就可以确定JSP文件没有被修改过并且servlet有效。这使得整个流程与其他脚本语言（比如PHP）相比要高效快捷一些。

总的来说，JSP网页就是用另一种方式来编写servlet而不用成为Java编程高手。除了解释阶段外，JSP网页几乎可以被当成一个普通的servlet来对待。

[Eclipse JSP/Servlet](http://www.runoob.com/jsp/eclipse-jsp.html)

[JSP 生命周期](http://www.runoob.com/jsp/jsp-life-cycle.html)

## 笔记列表

1. 学习++

  1336705531@qq.com

Web服务器使用JSP来创建网页的步骤：

1.客户端向服务器发送一个HTTP请求。

2.服务器识别是对JSP网页（me：通过路由映射或者请求的文件路径的后缀名如.jsp）的请求后将其请求传递给Servlet引擎。

Jsp->servlet类->编译成class->执行

Jsp是先转换然后编译，而PHP是直接解释执行（其实也是先分词，然后编译）

3.Servlet引擎将磁盘中的Jsp文件转化为Servlet类，并将所有的元素转化为Java代码

4.JSP引擎将Servlet类编译成可执行的文件并将最原始的请求发送给Servlet引擎

5.服务器通过调用Servlet引擎来执行Servlet可执行文件，Servlet会产生HTML格式的输出内嵌在HTTP Response中并将其上传给服务器

6.服务器将其以HTML的格式上传到客户端的浏览器中以静态页面的形式展现给用户。

# JSP 生命周期

理解JSP底层功能的关键就是去理解它们所遵守的生命周期。

JSP生命周期就是从创建到销毁的整个过程，类似于servlet生命周期，区别在于JSP生命周期还包括将JSP文件编译成servlet。

以下是JSP生命周期中所走过的几个阶段：

* **编译阶段：**

servlet容器编译servlet源文件，生成servlet类

* **初始化阶段：**

加载与JSP对应的servlet类，创建其实例，并调用它的初始化方法

* **执行阶段：**

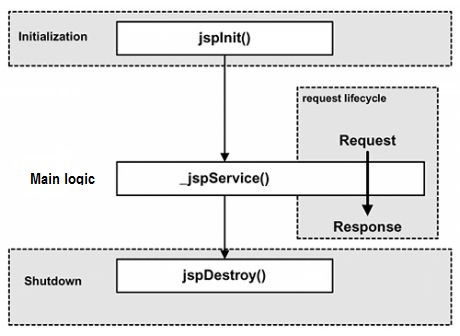
调用与JSP对应的servlet实例的服务方法

* **销毁阶段：**

调用与JSP对应的servlet实例的销毁方法，然后销毁servlet实例

编译类->加载类，构建对象->执行相应的对象方法->servlet实例销毁

很明显，JSP生命周期的四个主要阶段和servlet生命周期非常相似，下面给出图示：



## JSP编译

当浏览器请求JSP页面时，JSP引擎会首先去检查是否需要编译这个文件。如果这个文件没有被编译过，或者在上次编译后被更改过，则编译这个JSP文件。

编译的过程包括三个步骤：

* 解析JSP文件。
* 将JSP文件转为servlet。
* 编译servlet。

## JSP初始化

容器载入JSP文件后，它会在为请求提供任何服务前调用jspInit()方法。如果您需要执行自定义的JSP初始化任务，复写jspInit()方法就行了，就像下面这样：

public void jspInit(){

// 初始化代码

}

一般来讲程序只初始化一次，servlet也是如此。通常情况下您可以在jspInit()方法中初始化数据库连接、打开文件和创建查询表。

## JSP执行

这一阶段描述了JSP生命周期中一切与请求相关的交互行为，直到被销毁。

当JSP网页完成初始化后，JSP引擎将会调用\_jspService()方法。

\_jspService()方法需要一个HttpServletRequest对象和一个HttpServletResponse对象作为它的参数，就像下面这样：

void \_jspService(HttpServletRequest request,

HttpServletResponse response)

{

// 服务端处理代码

}

\_jspService()方法在每个request中被调用一次并且负责产生与之相对应的response，并且它还负责产生所有7个HTTP方法的回应，比如GET、POST、DELETE等等。

## JSP清理

JSP生命周期的销毁阶段描述了当一个JSP网页从容器中被移除时所发生的一切。

jspDestroy()方法在JSP中等价于servlet中的销毁方法。当您需要执行任何清理工作时复写jspDestroy()方法，比如释放数据库连接或者关闭文件夹等等。

jspDestroy()方法的格式如下：

public void jspDestroy()

{

// 清理代码

}

### 实例

JSP生命周期代码实例如下所示：

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

pageEncoding="UTF-8"%>

<html>

<head>

<title>life.jsp</title>

</head>

<body>

<%!

private int initVar=0;

private int serviceVar=0;

private int destroyVar=0;

%>

<%!

public void jspInit(){

initVar++;

System.out.println("jspInit(): JSP被初始化了"+initVar+"次");

}

public void jspDestroy(){

destroyVar++;

System.out.println("jspDestroy(): JSP被销毁了"+destroyVar+"次");

}

%>

<%

serviceVar++;

System.out.println("\_jspService(): JSP共响应了"+serviceVar+"次请求");

String content1="初始化次数 : "+initVar;

String content2="响应客户请求次数 : "+serviceVar;

String content3="销毁次数 : "+destroyVar;

%>

<h1>菜鸟教程 JSP 测试实例</h1>

<p><%=content1 %></p>

<p><%=content2 %></p>

<p><%=content3 %></p>

</body>

</html>

浏览器打开该页面，输出结果为：



# JSP 语法

本小节将会简单地介绍一下JSP开发中的基础语法。

## 脚本程序

（任意的java代码或者xml格式）

脚本程序可以包含任意量的Java语句、变量、方法或表达式，只要它们在脚本语言中是有效的。

脚本程序的语法格式：

<% 代码片段 %>

或者，您也可以编写与其等价的XML语句，就像下面这样：

<jsp:scriptlet>

代码片段

</jsp:scriptlet>

任何文本、HTML标签、JSP元素必须写在脚本程序的外面。

下面给出一个示例，同时也是本教程的第一个JSP示例：

<html>

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

<body>

Hello World!<br/>

<%

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

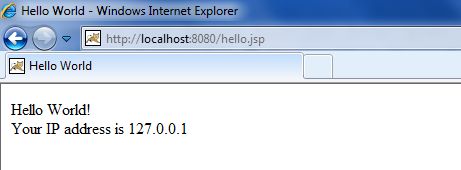
%>

</body>

</html>

**注意：**请确保Apache Tomcat已经安装在C:\apache-tomcat-7.0.2目录下并且运行环境已经正确设置。

将以上代码保存在hello.jsp中，然后将它放置在 C:\apache-tomcat-7.0.2\webapps\ROOT目录下，打开浏览器并在地址栏中输入http://localhost:8080/hello.jsp。运行后得到以下结果：



### 中文编码问题

如果我们要在页面正常显示中文，我们需要在 JSP 文件头部添加以下代码：<>

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

pageEncoding="UTF-8"%>

接下来我们将以上程序修改为：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

Hello World!<br/>

<%

out.println("你的 IP 地址 " + request.getRemoteAddr());

%>

</body>

</html>

这样中文就可以正常显示了。

## JSP声明

一个声明语句可以声明一个或多个变量、方法，供后面的Java代码使用。在JSP文件中，您必须先声明这些变量和方法然后才能使用它们。

JSP声明的语法格式：

<%! declaration; [ declaration; ]+ ... %>

或者在普通的语法中声明也可以

<% declaration; [declaration;]%>

或者，您也可以编写与其等价的XML语句，就像下面这样：

<jsp:declaration>

代码片段

</jsp:declaration>

程序示例：

<%! int i = 0; %>

<%! int a, b, c; %>

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

## JSP表达式

一个JSP表达式中包含的脚本语言表达式，先被转化成String，然后插入到表达式出现的地方。

由于表达式的值会被转化成String，所以您可以在一个文本行中使用表达式而不用去管它是否是HTML标签。

表达式元素中可以包含任何符合Java语言规范的表达式，但是不能使用分号来结束表达式。

JSP表达式的语法格式：

<%= 表达式 %> //这样就不需输出了。表达式的结果将转换为字符串并填充到该表达式出现地地方

同样，您也可以编写与之等价的XML语句：

<jsp:expression>

表达式

</jsp:expression>

程序示例：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<p>

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

</p>

</body>

</html>

运行后得到以下结果：

今天的日期是: 2016-6-25 13:40:07

## JSP注释

JSP注释主要有两个作用：为代码作注释以及将某段代码注释掉。

JSP注释的语法格式：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<%-- 该部分注释在网页中不会被显示--%>

<p>

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

</p>

</body>

</html>

运行后得到以下结果：

今天的日期是: 2016-6-25 13:41:26

不同情况下使用注释的语法规则：

|  |  |
| --- | --- |
| **语法** | **描述** |
| <%-- 注释 --%> | JSP注释，注释内容不会被发送至浏览器甚至不会被编译 |
| <!-- 注释 --> | HTML注释，通过浏览器查看网页源代码时可以看见注释内容 |
| <\% | 代表静态 <%常量 |
| %\> | 代表静态 %> 常量 |
| \' | 在属性中使用的单引号 |
| \" | 在属性中使用的双引号 |

## JSP指令

JSP指令用来设置与整个JSP页面相关的属性。

JSP指令语法格式：

<%@ directive attribute="value" %>

这里有三种指令标签：

|  |  |
| --- | --- |
| **指令** | **描述** |
| <%@ page ... %> | 定义页面的依赖属性，比如脚本语言、error页面、缓存需求等等 |
| <%@ include ... %> | 包含其他文件 |
| <%@ taglib ... %> | 引入标签库的定义，可以是自定义标签 |

## JSP行为

JSP行为标签使用XML语法结构来控制servlet引擎。它能够动态插入一个文件，重用JavaBean组件，引导用户去另一个页面，为Java插件产生相关的HTML等等。

行为标签只有一种语法格式，它严格遵守XML标准：

<jsp:action\_name attribute="value" />

行为标签基本上是一些预先就定义好的函数，下表罗列出了一些可用的JSP行为标签：：

|  |  |
| --- | --- |
| **语法** | **描述** |
| jsp:include | 用于在当前页面中包含静态或动态资源 |
| jsp:useBean | 寻找和初始化一个JavaBean组件 |
| jsp:setProperty | 设置 JavaBean组件的值 |
| jsp:getProperty | 将 JavaBean组件的值插入到 output中 |
| jsp:forward | 从一个JSP文件向另一个文件传递一个包含用户请求的request对象 |
| jsp:plugin | 用于在生成的HTML页面中包含Applet和JavaBean对象 |
| jsp:element | 动态创建一个XML元素 |
| jsp:attribute | 定义动态创建的XML元素的属性 |
| jsp:body | 定义动态创建的XML元素的主体 |
| jsp:text | 用于封装模板数据 |

## JSP隐含对象

JSP支持九个自动定义的变量（me：类似PHP中的全局常量），江湖人称隐含对象。这九个隐含对象的简介见下表：

|  |  |
| --- | --- |
| **对象** | **描述** |
| request | **HttpServletRequest**类的实例 |
| response | **HttpServletResponse**类的实例 |
| out | **PrintWriter**类的实例，用于把结果输出至网页上 |
| session | **HttpSession**类的实例 |
| application | **ServletContext**类的实例，与应用上下文有关 |
| config | **ServletConfig**类的实例 |
| pageContext | **PageContext**类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | **Exception**类的对象，代表发生错误的JSP页面中对应的异常对象 |

## 控制流语句

JSP提供对Java语言的全面支持。您可以在JSP程序中使用Java API甚至建立Java代码块，包括判断语句和循环语句等等。

## 判断语句

类似PHP

If…else块，请看下面这个例子：

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

pageEncoding="UTF-8"%>

<%! int day = 3; %>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h3>IF...ELSE 实例</h3>

<% if (day == 1 | day == 7) { %>

<p>今天是周末</p>

<% } else { %>

<p>今天不是周末</p>

<% } %>

</body>

</html>

运行后得到以下结果：

IF...ELSE 实例

今天不是周末

现在来看看switch…case块，与if…else块有很大的不同，它使用out.println()，并且整个都装在脚本程序的标签中，就像下面这样：

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

pageEncoding="UTF-8"%>

<%! int day = 3; %>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h3>SWITCH...CASE 实例</h3>

<%

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");

}

%>

</body>

</html>

浏览器访问，运行后得出以下结果：

SWITCH...CASE 实例

星期三

## 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。

让我们来看看for循环的例子，以下输出的不同字体大小的"菜鸟教程"：

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

pageEncoding="UTF-8"%>

<%! int fontSize; %>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h3>For 循环实例</h3>

<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>

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

菜鸟教程

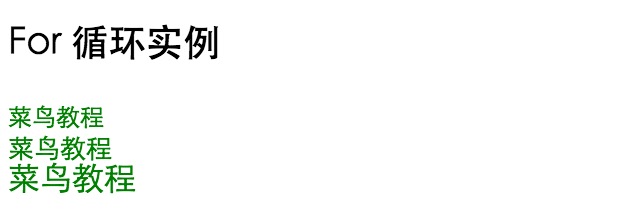
</font><br />

<%}%>

</body>

</html>

运行后得到以下结果：



将上例改用while循环来写：

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

pageEncoding="UTF-8"%>

<%! int fontSize; %>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h3>While 循环实例</h3>

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

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

菜鸟教程

</font><br />

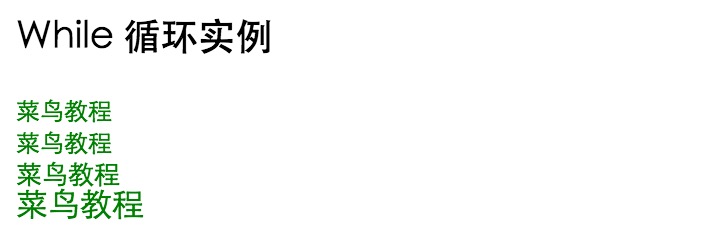
<%fontSize++;%>

<%}%>

</body>

</html>

浏览器访问，输出结果为（fontSize 初始化为0，所以多输出了一行）：



JSP运算符

JSP支持所有Java逻辑和算术运算符。

下表罗列出了JSP常见运算符，优先级从高到底：

|  |  |  |
| --- | --- | --- |
| **类别** | **操作符** | **结合性** |
| 后缀 | () [] . (点运算符) | 左到右 |
| 一元 | ++ - - ! ~ | 右到左 |
| 可乘性 | \* / % | 左到右 |
| 可加性 | + - | 左到右 |
| 移位 | >> >>> << | 左到右 |
| 关系 | > >= < <= | 左到右 |
| 相等/不等 | == != | 左到右 |
| 位与 | & | 左到右 |
| 位异或 | ^ | 左到右 |
| 位或 | | | 左到右 |
| 逻辑与 | && | 左到右 |
| 逻辑或 | || | 左到右 |
| 条件判断 | ?: | 右到左 |
| 赋值 | = += -= \*= /= %= >>= <<= &= ^= |= | 右到左 |
| 逗号 | , | 左到右 |

## JSP 字面量

JSP语言定义了以下几个字面量：

* 布尔值(boolean)：true 和 false;
* 整型(int)：与 Java 中的一样;
* 浮点型(float)：与 Java 中的一样;
* 字符串(string)：以单引号或双引号开始和结束;
* Null：null。

**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。

语法格式如下：

<%@ directive attribute="value" %>

指令可以有很多个属性，它们以键值对的形式存在，并用逗号隔开。

JSP中的三种指令标签：

|  |  |
| --- | --- |
| **指令** | **描述** |
| <%@ page ... %> | 定义网页依赖属性，比如脚本语言、error页面、缓存需求等等 |
| <%@ include ... %> | 包含其他文件 |
| <%@ taglib ... %> | 引入标签库的定义 |

**Page指令**

Page指令为容器提供当前页面的使用说明。一个JSP页面可以包含多个page指令。

Page指令的语法格式：

<%@ page attribute="value" %>

等价的XML格式：

<jsp:directive.page attribute="value" />

**属性**

下表列出与Page指令相关的属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| buffer | 指定out对象使用缓冲区的大小 |
| autoFlush | 控制out对象的 缓存区 |
| contentType | 指定当前JSP页面的MIME类型和字符编码 |
| errorPage | 指定当JSP页面发生异常时（一般在jsp初始化后，如果发生异常就会自动跳转到这个错误页面）需要转向的错误处理页面 |
| isErrorPage | 指定当前页面是否可以作为另一个JSP页面的错误处理页面 |
| extends | 指定servlet从哪一个类继承（类似C#的inherit） |
| import | 导入要使用的Java类 |
| info | 定义JSP页面的描述信息 |
| isThreadSafe | 指定对JSP页面的访问是否为线程安全 |
| language | 定义JSP页面所用的脚本语言，默认是Java |
| session | 指定JSP页面是否使用session |
| isELIgnored | 指定是否执行EL表达式 |
| isScriptingEnabled | 确定脚本元素能否被使用 |

**Include指令**

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。包含的文件就好像是该JSP文件的一部分，会被同时编译执行。

Include指令的语法格式如下：

<%@ include file="文件相对 url 地址" %>

**include** 指令中的文件名实际上是一个相对的 URL 地址。

如果您没有给文件关联一个路径，JSP编译器默认在当前路径下寻找。

等价的XML语法：

<jsp:directive.include file="文件相对 url 地址" />

**Taglib指令**

JSP API允许用户自定义标签，一个自定义标签库就是自定义标签的集合。

Taglib指令引入一个自定义标签集合的定义，包括库路径、自定义标签。

Taglib指令的语法：

<%@ taglib uri="uri" prefix="prefixOfTag" %>

uri属性确定标签库的位置，prefix属性指定标签库的前缀。

等价的XML语法：

<jsp:directive.taglib uri="uri" prefix="prefixOfTag" />