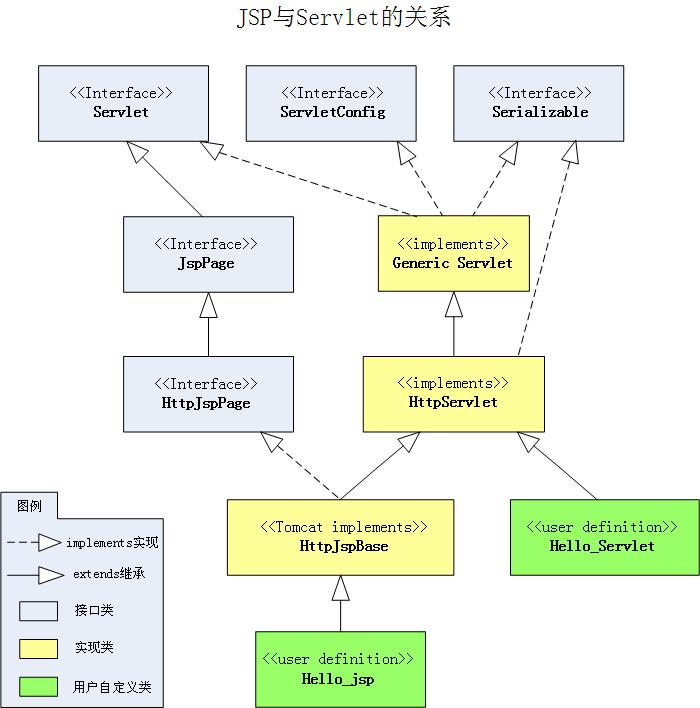
[图解JSP与Servlet的关系](http://blog.csdn.net/kaixinbingju/article/details/9409927)

Servlet是[**Java**](http://lib.csdn.net/base/javase)提供的用于开发Web服务器应用程序的一个组件，运行在服务器端，由Servlet容器所管理，用于生成动态的内容。Servlet是平台独立的Java类，编写一个Servlet，实际上就是按照Servlet规范编写一个Java类。



如图所示，Java提供一系列接口类（所谓接口类就是类中**所有**方法只提供方法声明，不提供任何的方法实现，这些类的实现就留给后继者去做。）：Servlet、ServletConfig、Serializable，然后通过多重继承产生一个最通用的Servlet实现类（图中Gerneric Servlet类），接下来，通过一个多重继承与实现，产生一个新的实现类HttpServlet，用户在开发Servlet程序时只需继承这个类，从而产生一个自己的类（图中Hello\_Servlet类），然后根据实际开发功能与信息处理需要，去实现该类中的相关方法即可。这就是前面提到的按照Servlet规范编写一个Java类，从而编写一个Servlet。

至于JSP（JavaServlet Page）从图中可以看出，实际上它也是从Servlet继承而来。只不过它在Servlet当中又添加/修改了一些方法，作了新的封装。具体到Tomcat Web应用服务器中，它通过一个多重继承，分别从Java的HttpJspPage和HttpServlet两个类那里继承和实现一些方法，然后封装一个叫做HttpJspBase的类从而实现了一个通用化的JSP类，用户在开发自己的JSP时，只需要从HttpJspBase继承一个自己的类（如图中Hello\_jsp类），然后根据需要去实现相应的方法即可。

因此这也是为什么JSP的代码中总是闪现Servlet代码框架影子的原因，其实它们只是为实现同样的功能而进行了不同封装的组件而已，血脉里留着的是一样的血。

“既生瑜何生亮？”呵呵，因为JSP确实比Servlet要更胜一筹，所谓“青出于蓝胜于蓝”，既然Sun公司要在Servlet基础上推出JSP技术，那肯定是因为JSP有它更高明的地方。

使用Servlet产生动态网页，需要在代码中打印输出很多HTML的标签，此外，在Servlet中，我们不得不将静态现实的内容和动态产生内容的代码混合在一起。使用Servlet开发动态网页，程序员和网页编辑人员将无法一起工作，因为网页编辑人员不了解Java语言，无法修改Servlet代码，而Java程序员可能也不是很了解网页编辑人员的意图，以至于无法修改和实现网页功能。为了解决这些问题，Sun公司就推出了JSP技术。

**JSP是Servlet的扩展**，在没有JSP之前，就已经出现了Servlet技术。Servlet是利用输出流动态生成HTML页面，包括每一个HTML标签和每个在HTML页面中出现的内容。

JSP通过在标准的HTML页面中插入Java代码，其静态的部分无须Java程序控制，只有那些需要从[**数据库**](http://lib.csdn.net/base/mysql)读取并根据程序动态生成信息时，才使用Java脚本控制。

事实上，JSP是Servlet的一种特殊形式，每个JSP页面就是一个Servlet实例——JSP页面由系统编译成Servlet，Servlet再负责响应用户请求。JSP其实也是Servlet的一种简化，使用JSP时，其实还是使用Servlet，因为Web应用中的每个JSP页面都会由Servlet容器生成对应的Servlet。对于Tomcat而言，JSP页面生成的Servlet放在work路径对应的Web应用下。

以apache-tomcat-7.0.37\webapps\myapp\index.jsp为例，

**[html]** [view plain](http://blog.csdn.net/kaixinbingju/article/details/9409927) [copy](http://blog.csdn.net/kaixinbingju/article/details/9409927)

1. **<html>**
2. **<body>**
3. **<center>**
4. Now time is: **<**%=new java.util.Date()%**>**
5. **</center>**
6. **</body>**
7. **</html>**

当启动Tomcat之后，可以在Tomcat的apache-tomcat-7.0.37\work\Catalina\localhost\myapp\org\apache\jsp目录下找到如下文件：indexd.java和index.class。这两个文件都是Tomcat生成的，Tomcat根据JSP页面生成对应Servlet的Java文件及class文件。

index.java

**[html]** [view plain](http://blog.csdn.net/kaixinbingju/article/details/9409927) [copy](http://blog.csdn.net/kaixinbingju/article/details/9409927)

1. //JSP页面经过Tomcat编译后默认的包
2. package org.apache.jsp;
4. import javax.servlet.\*;
5. import javax.servlet.http.\*;
6. import javax.servlet.jsp.\*;
8. //继承HttpJspBase类，该类其实是个Servlet的子类
9. public final class index\_jsp extends org.apache.jasper.runtime.HttpJspBase
10. implements org.apache.jasper.runtime.JspSourceDependent {
12. private static final javax.servlet.jsp.JspFactory \_jspxFactory =
13. javax.servlet.jsp.JspFactory.getDefaultFactory();
15. private static java.util.Map**<java.lang.String**,java.lang.Long**>** \_jspx\_dependants;
17. private javax.el.ExpressionFactory \_el\_expressionfactory;
18. private org.apache.tomcat.InstanceManager \_jsp\_instancemanager;
20. public java.util.Map**<java.lang.String**,java.lang.Long**>** getDependants() {
21. return \_jspx\_dependants;
22. }
24. public void \_jspInit() {
25. \_el\_expressionfactory = \_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
26. \_jsp\_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
27. }
29. public void \_jspDestroy() {
30. }
32. //用于响应用户的方法
33. public void \_jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse response)
34. throws java.io.IOException, javax.servlet.ServletException {
36. final javax.servlet.jsp.PageContext pageContext;
37. javax.servlet.http.HttpSession session = null;
38. final javax.servlet.ServletContext application;
39. final javax.servlet.ServletConfig config;
40. //获得页面输出流
41. javax.servlet.jsp.JspWriter out = null;
42. final java.lang.Object page = this;
43. javax.servlet.jsp.JspWriter \_jspx\_out = null;
44. javax.servlet.jsp.PageContext \_jspx\_page\_context = null;

47. //开始生成响应
48. try {
49. //设置输出的页面格式
50. response.setContentType("text/html");
51. pageContext = \_jspxFactory.getPageContext(this, request, response,
52. null, true, 8192, true);
53. \_jspx\_page\_context = pageContext;
54. application = pageContext.getServletContext();
55. config = pageContext.getServletConfig();
56. session = pageContext.getSession();
57. //页面输出流
58. out = pageContext.getOut();
59. \_jspx\_out = out;
61. //输出流，开始输出页面文档
62. out.write("**<html>**\r\n");
63. out.write("**<body>**\r\n");
64. out.write("**<center>** \r\n");
65. out.write("Now time is: ");
66. out.print(new java.util.Date());
67. out.write(" \r\n");
68. out.write("**</center>**\r\n");
69. out.write("**</body>**\r\n");
70. out.write("**</html>**");
71. } catch (java.lang.Throwable t) {
72. if (!(t instanceof javax.servlet.jsp.SkipPageException)){
73. out = \_jspx\_out;
74. if (out != null && out.getBufferSize() != 0)
75. try { out.clearBuffer(); } catch (java.io.IOException e) {}
76. if (\_jspx\_page\_context != null) \_jspx\_page\_context.handlePageException(t);
77. else throw new ServletException(t);
78. }
79. } finally {
80. \_jspxFactory.releasePageContext(\_jspx\_page\_context);
81. }
82. }
83. }

**JSP页面中内置了几个对象，如pageContext、application、config、page、session、out等\_jspService()方法，这几个内置对象就是在这里定义的。**

根据上面的JSP页面工作原理图，可以得到如下结论： JSP文件必须在JSP服务器内运行。JSP文件必须生成Servlet才能执行。每个JSP页面的第一个访问者速度很慢，因为必须等待JSP编译成Servlet。JSP页面的访问者无须安装任何客户端，甚至不需要可以运行Java的运行环境，因为JSP页面输送到客户端的是标准HTML页面。index.jsp页面中的每个字符都由index.java文件的输出流生成.  
servlet是在web服务器上的java程序，它提供服务，由它来传递给你html的格式。Servlet API为Servlet提供了统一的编程接口  
Servlet必须部署在Servlet容器，才能响应客户端的请求  对外提供服务。要对外统一接口，由容器来调用。  
jsp侧重显示；servlet侧重控制逻辑。

MVC模式：Jsp + Servlet +　JavaBean。M-JavaBean V-Jsp C-Servlet

小应用程序（Applet）是指采用Java创建的基于HTML的程序。浏览器将其暂时下载到用户的硬盘上，并在Web页打开时在本地运行。们可以直接嵌入到网页或者其他特定的容器中，并能够产生特殊的效果。

所有基于Java的服务器端编程都是构建在Servlet之上的。在J2EE中Servlet已经是一个标准的组件。

目前，Servlet引擎一般是第三方的插件，它通过一定的方法连接到Web服务器，Servlet引擎把它识别为Servlet请求的那些HTTP请求截获下来处理，而其他的HTTP请求由Web服务器按照通常的方式来处理，Servlet引擎会装载合适的Servlet到内存中，如果Servlet还没有运行的话，会分配一个可以使用的线程来处理请求，再把Servlet的输出返回到发出请求的Web客户机。

Java Servlet和[**Java**](http://lib.csdn.net/base/java)Applet正好是相对应的两种程序类型，Applet运行在客户端，在浏览器内执行，而Servlet在服务器内部运行，通过客户端提交的请求启动运行。

同样的Servlet完全可以在Apache，IIS等不同Web服务器上执行，不管底层的[**操作系统**](http://lib.csdn.net/base/operatingsystem)是Windows，Solaris，Mac，[**Linux**](http://lib.csdn.net/base/linux)。

Java Servlet有着十分广泛的应用。使用Servlet还可以实现大量的服务器端的管理维护功能，以及各种特殊的任务，比如，并发处理多个请求，转送请求，代理等。

典型的Servlet运行环境有JSWDK，Tomcat，Resin等，。它们都自带一个简单的HTTP Server，只需简单配置即可投入使用，你也可以把它们绑定到常用的Web服务器上，如Apache，IIS等，提供小规模的Web服务。还有一些商业的大中型的支持Servlet和JSP的Web服务器，如JRun，Web Sphere，Web Logic等等，配置比较复杂，并不适合初学者。但是功能较为强大，有条件的读者可以一试。

**[html]** [view plain](http://blog.csdn.net/kaixinbingju/article/details/9409927) [copy](http://blog.csdn.net/kaixinbingju/article/details/9409927)

1. import java.io.\*;
2. import java.util.\*;
3. import javax.servlet.http.\*;
4. import javax.servlet.\*;
5. //导入必要的包
6. public class HelloServlet extends HttpServlet {
7. //所有Servlet必须从HttpServlet派生
8. public void doGet (HttpServletRequest req, HttpServletResponse res)
9. throws ServletException, IOException
10. //doGet()是这个Servlet的核心，真正处理请求的地方
11. {
12. res.setContentType("text/html");
13. //设置相应的类型为text/html
14. PrintWriter pw = res.getWriter();
15. //从HttpServletResponse得到输出流
16. pw.println("<!DOCTYPE HTML PUBLIC ""-//W3C//DTD HTML 4.0 Transitional//EN""**>**");
17. pw.println("**<head>**");
18. pw.println("**<meta** http-equiv=""Content-Type"" content=""text/html; charset=ISO-8859-1""**>**");
19. pw.println("<!-- The Servlet expression tags interpolate script variables into the HTML -->");
20. pw.println("**<title>**Hello, world!**</title>**");
21. pw.println("**</head>**");
22. pw.println("**<body** bgcolor=#cc99dd**>**");
23. pw.println("**<h1>**Hello, world!**</h1>**");
24. pw.println("**</body>**");
25. //上面的语句都是向客户端打印HTML文本
26. pw.close();
27. //关闭HttpServletResponse，使Web服务器知道相应结束
28. }
29. public HelloServlet() {} //构造函数，可以不要