

# 09-jsp

## 今日任务

1. 什么是 Jsp?
2. 为什么要学习 jsp 技术
3. 如何创建一个 jsp 动态页面。
4. 如何修改 jsp 页面的默认编码?
5. jsp 的本质是什么。
6. jsp 的三种语法
  - a) jsp 头部的 page 指令
    - i. language 属性
    - ii. contentType 属性
    - iii. pageEncoding 属性
    - iv. import 属性
    - v. autoFlush 属性
    - vi. buffer 属性
    - vii. errorPage 属性
    - viii. isErrorPage 属性
    - ix. session 属性
    - x. extends 属性
  - b) jsp 中的三种脚本
    - i. 声明脚本
      1. 我们可以定义全局变量。
      2. 定义 static 静态代码块
      3. 定义方法
      4. 定义内部类
    - ii. 表达式脚本
      1. 输出整型
      2. 输出浮点型
      3. 输出字符串
      4. 输出对象
    - iii. 代码脚本
      1. 代码脚本----if 语句
      2. 代码脚本----for 循环语句
      3. 翻译后 java 文件中\_jspService 方法内的代码都可以写
  - c) jsp 中的三种注释
    - i. html 注释
    - ii. java 注释

## iii. jsp 注释

7. jsp 九大内置对象
8. jsp 四大域对象
9. jsp 中的 out 输出和 response.getWriter 输出的区别
10. jsp 的常用标签
  - a) jsp 静态包含
  - b) jsp 标签-动态包含
  - c) jsp 标签-转发
11. 静态包含和动态包含的区别
12. jsp 练习题--jsp 输出一个表格，里面有 20 个学生信息。
13. 什么是 Listener 监听器？
- 13.1、ServletContextListener 监听器

## 今日课程

# 1、为什么要学习 jsp 技术

## 1.1、什么是 jsp？

JSP(全称 Java Server Pages)是由 Sun 公司专门为了解决动态生成 HTML 文档的技术。

## 1.2、Servlet 程序输出 html 页面。

在学习 jsp 技术之前，如果我们要往客户端输出一个页面。我们可以使用 Servlet 程序来实现。具体的代码如下：

### 1) Servlet 输入 html 页面的程序代码：

```
package com.atguigu.servlet;

import java.io.IOException;
import java.io.Writer;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HtmlServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // 设置返回的数据内容的数据类型和编码
        response.setContentType("text/html; charset=utf-8");
        // 获取字符输出流
        Writer writer = response.getWriter();
        //输出页面内容!
        writer.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"
\\\"http://www.w3.org/TR/html4/loose.dtd\\\">");
        writer.write("<html>");
        writer.write("<head>");
        writer.write("<meta http-equiv=\\\"Content-Type\\\" content=\\\"text/html;
charset=UTF-8\\\">");
        writer.write("<title>Insert title here</title>");
        writer.write("</head>");
        writer.write("<body>");
        writer.write("这是由 Servlet 程序输出的 html 页面内容! ");
        writer.write("</body></html>");
    }

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

2) 在浏览器中输入访问 Servlet 的程序地址得到以下结果:

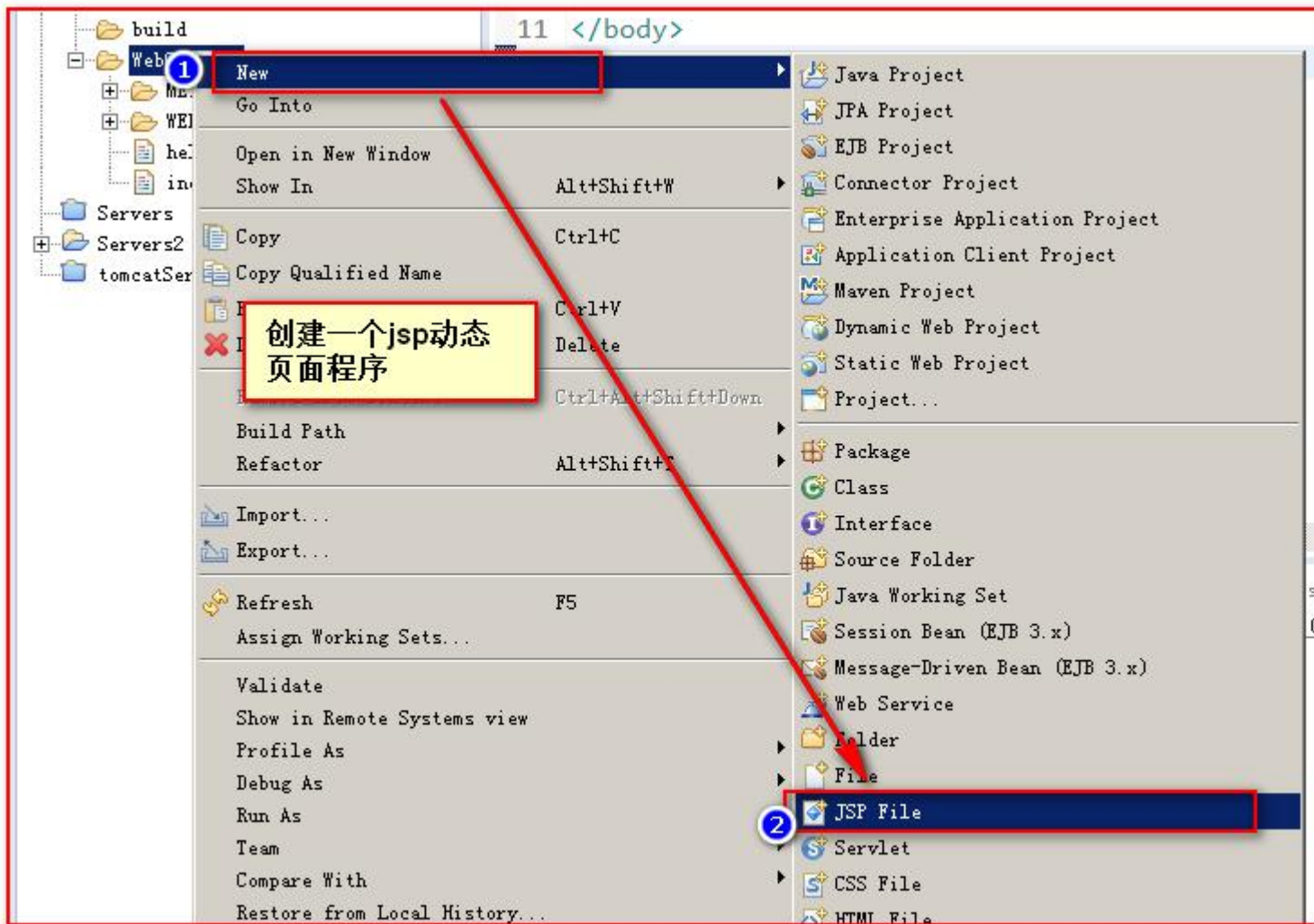


上面的代码我们不难发现。通过 Servlet 输出简单的 html 页面信息都非常不方便。那我们要输出一个复杂页面的时候，就更加的困难，而且不利于页面的维护和调试。所以 sun 公司推出一种叫做 jsp 的动态页面技术帮助我们实现对页面的输出繁琐工作。

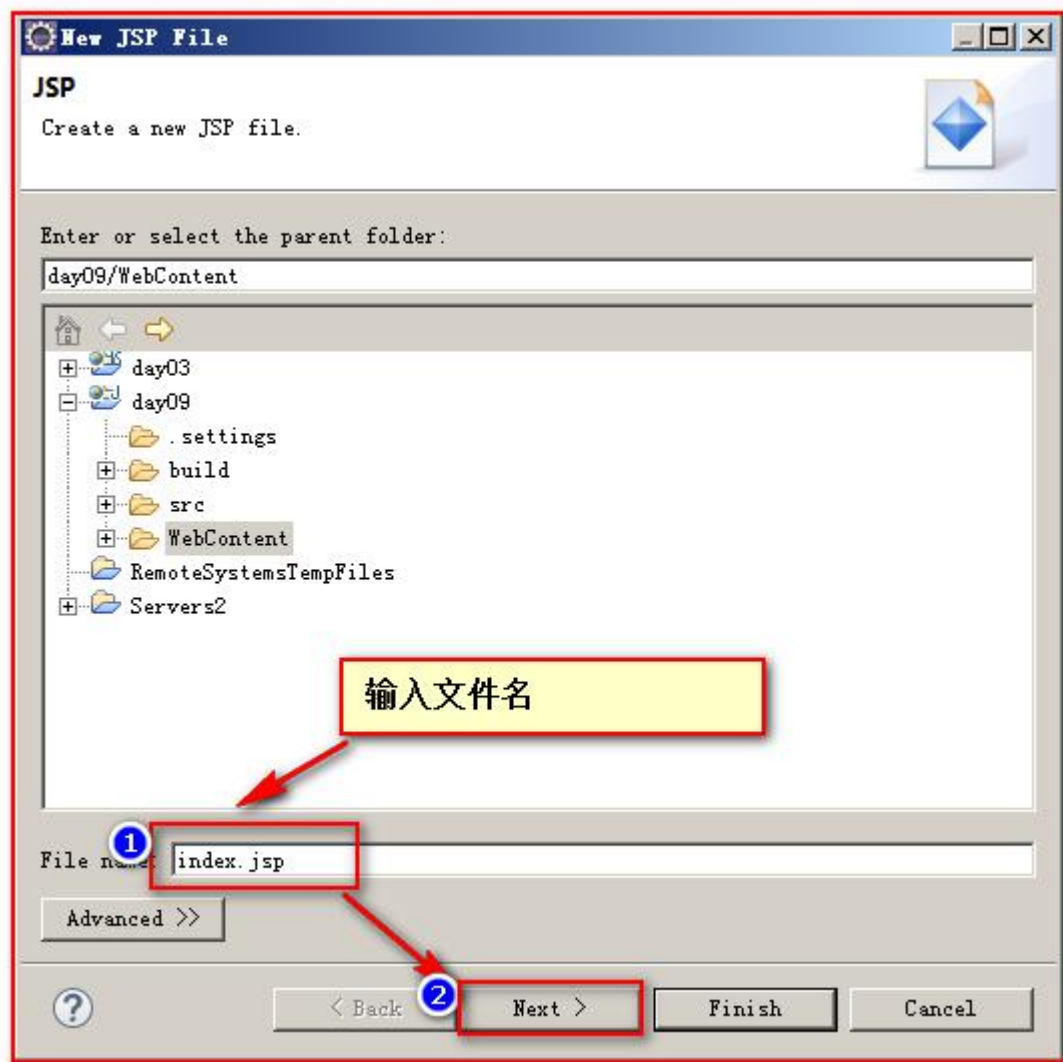
jsp 页面的访问千万不能像 HTML 页面一样。托到浏览器中。只能通过浏览器访问 Tomcat 服务器再访问 jsp 页面。

## 1.3、如何创建一个 jsp 动态页面程序

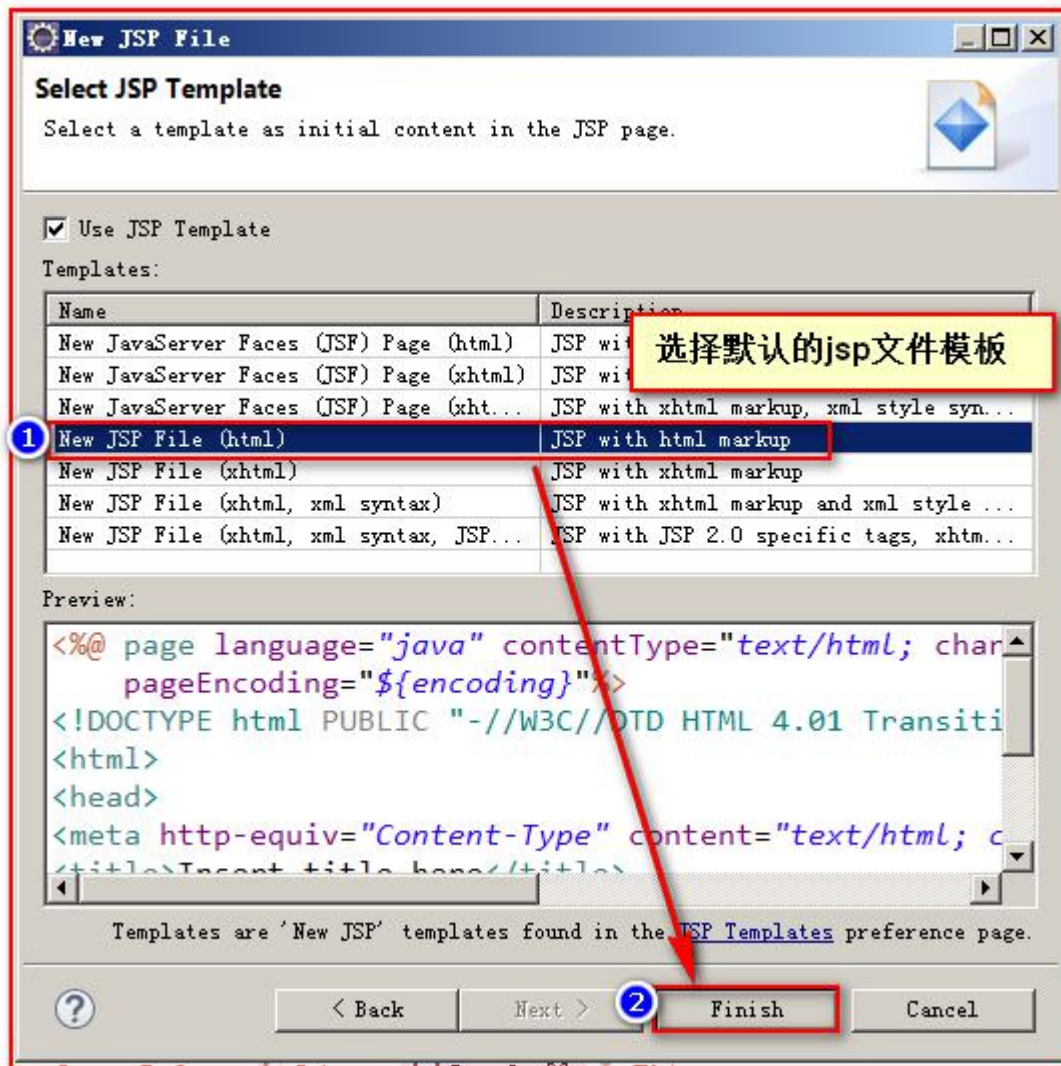
- 1) 选中 WebContent 目录，右键创建一个 jsp 文件



## 2) 修改 jsp 页面的文件名

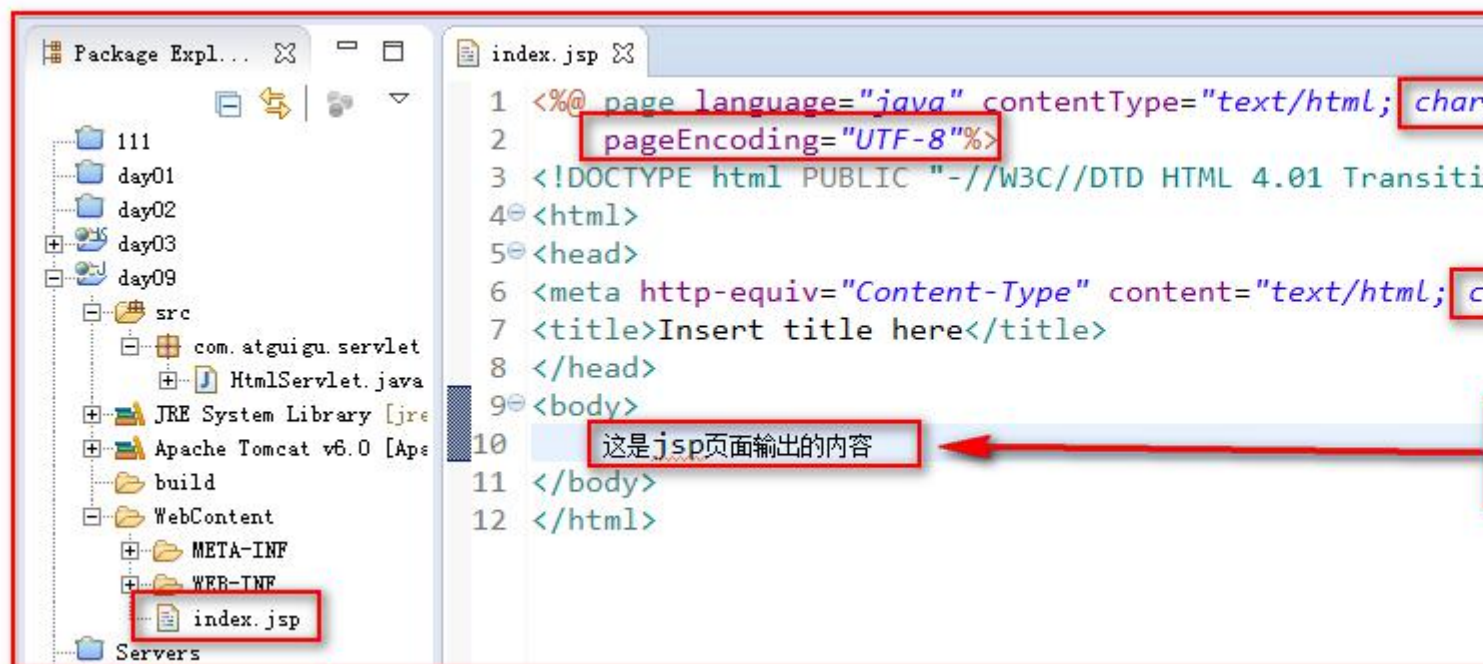


3) 选择生成 jsp 文件的模板,我们选择默认的 New JSP File(html)



4) 在 **body** 标签中添加你想要显示的文本内容





5) 然后在浏览器中输入 jsp 页面的访问地址。

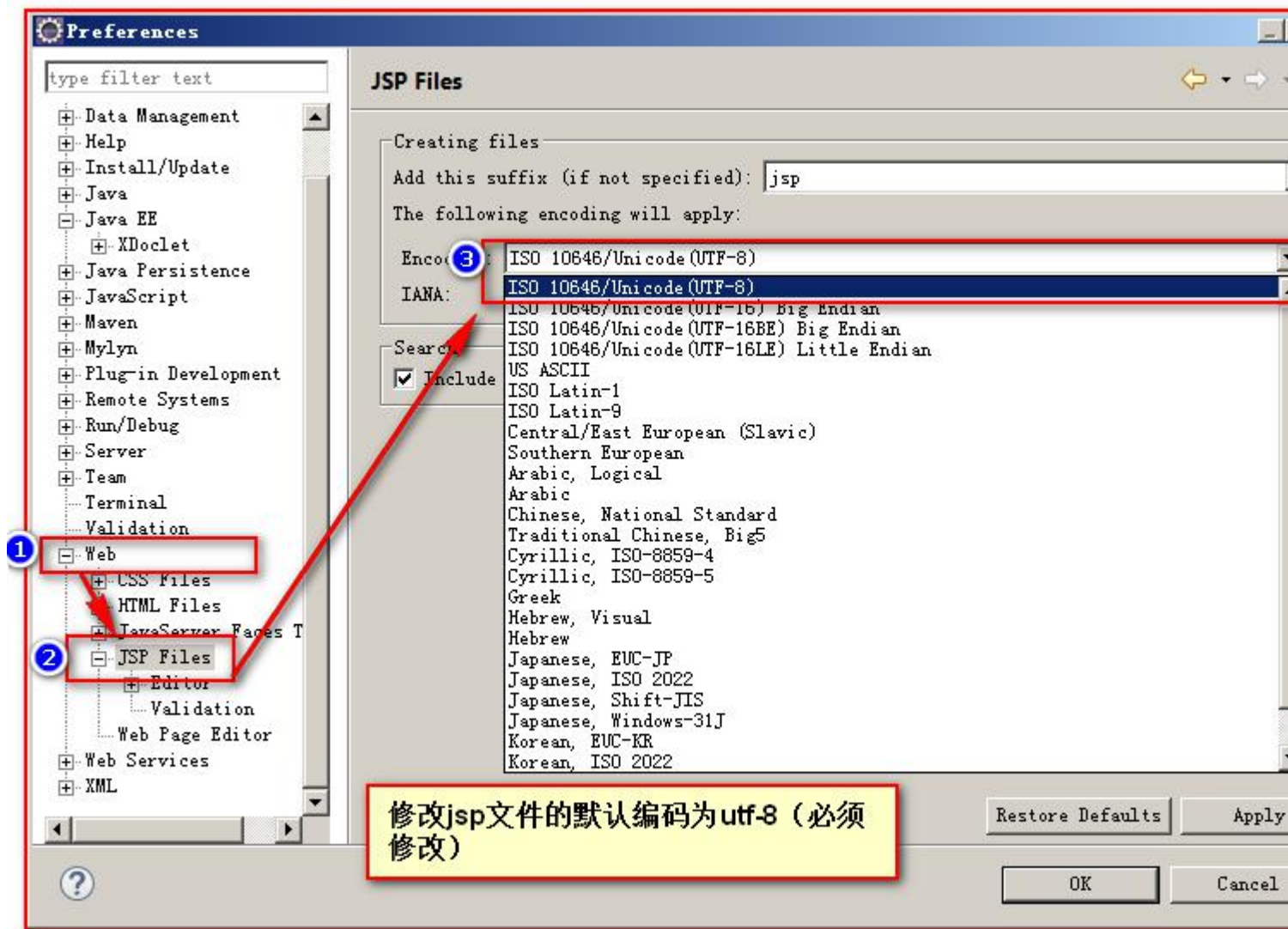
jsp 页面的访问地址和 html 页面的访问路径一样 `http://ip:端口号/工程名/文件名`

也就是 `http://127.0.0.1:8080/day08/index.jsp`



## 1.4、如何修改 jsp 文件的默认编码。





## 注意事项:

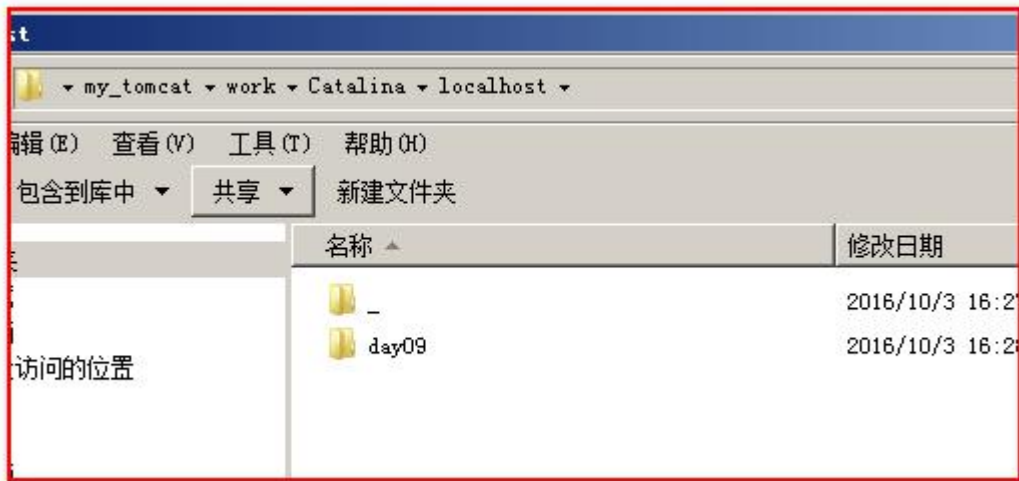
- 1、jsp 页面是一个类似于 html 的一个页面。jsp 直接存放到 WebContent 目录下，和 html 一样访问 jsp 的时候，也和访问 html 一样
- 2、jsp 的默认编码集是 iso-8859-1 修改 jsp 的默认编码为 UTF-8

## 2、jsp 的运行原理（要求知道）

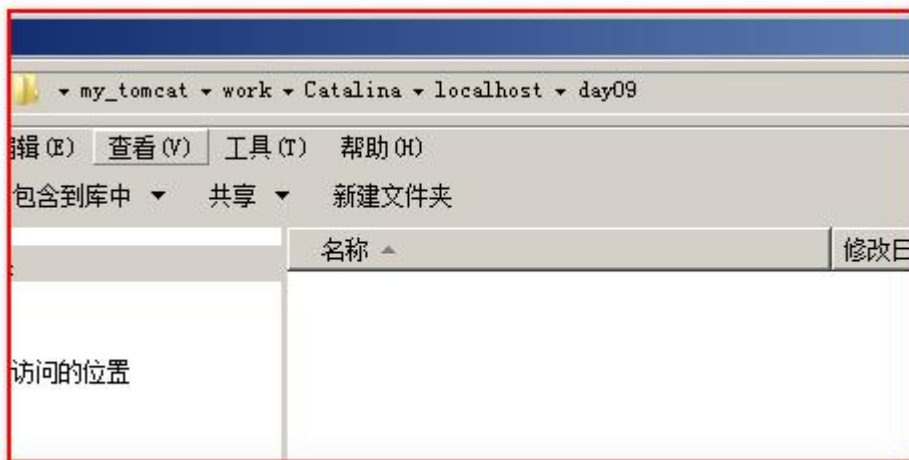
jsp 的本质，其实是一个 Servlet 程序。

首先我们去找到我们 Tomcat 的目录下的 `work\Catalina\localhost` 目录。当我们发布 day09 工程。并启动 Tomcat 服务器后。我们发现

在 `work\Catalina\localhost` 目录下多出来一个 `day09` 目录。



一开始 `day09` 目录还是空目录。

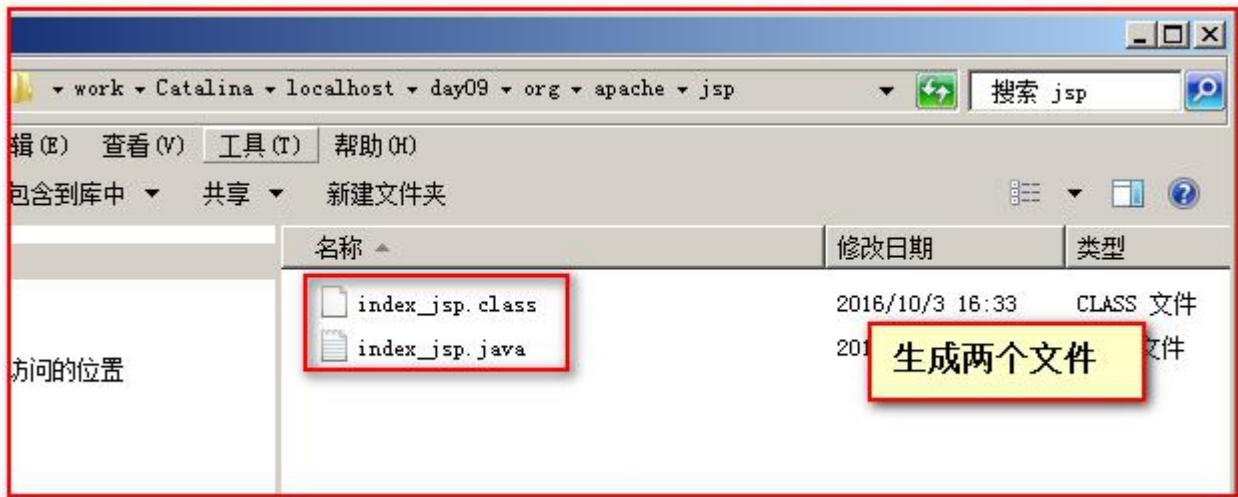


然后，我们在浏览器输入一个 `jsp` 文件的访问路径访问。

比如 `http://127.0.0.1:8080/day09/index.jsp` 访问 `index.jsp` 文件

`day09` 目录马上会生成 `org\apache\jsp` 目录。

并且在会中有两个文件。



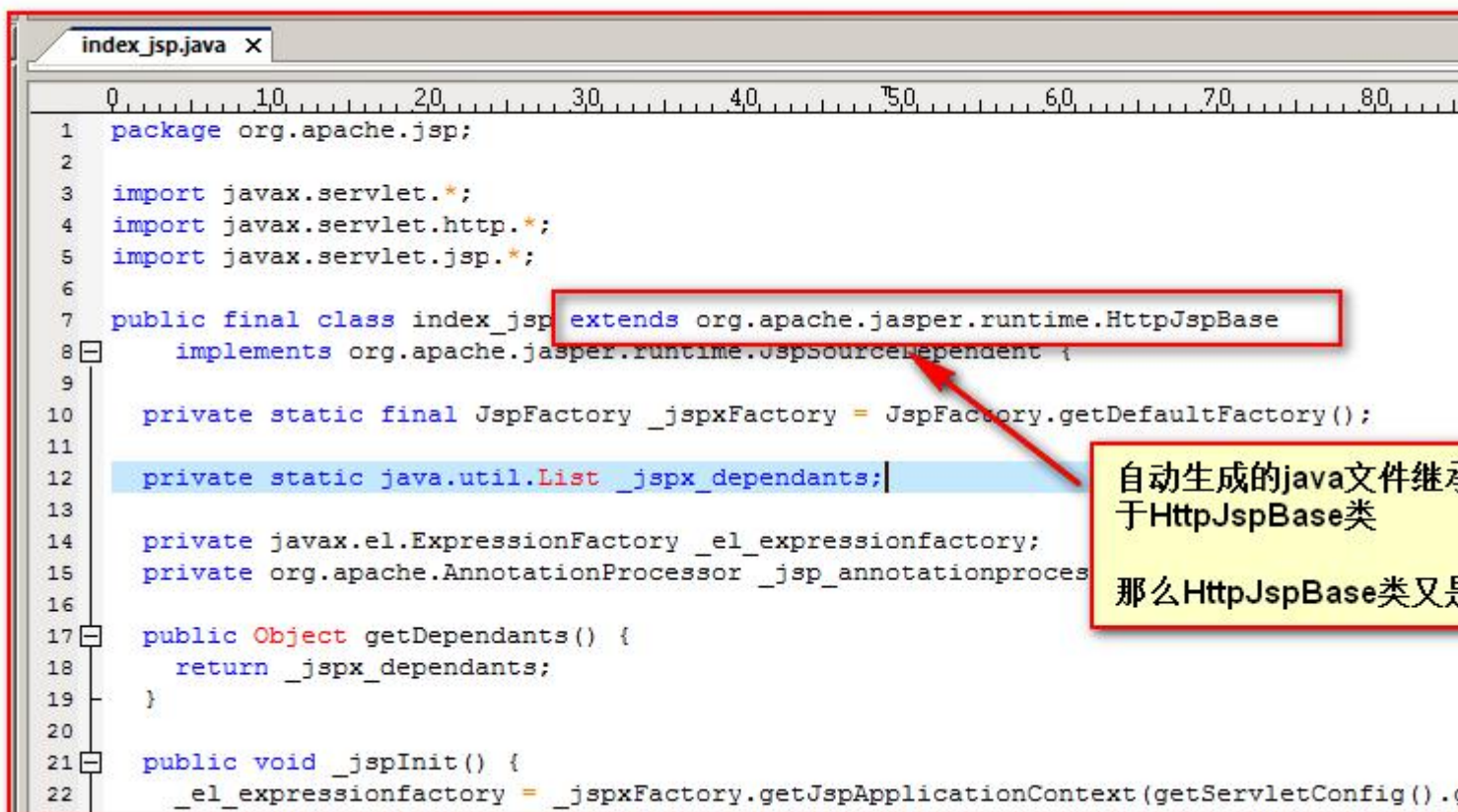
index\_jsp.class 文件很明显是 index\_jsp.java 源文件编译后的字节码文件。

那么 index\_jsp.java 是个什么内容呢？

生成的 java 文件名，是以原来的文件名加上\_jsp 得到。xxxx\_jsp.java 文件的名字

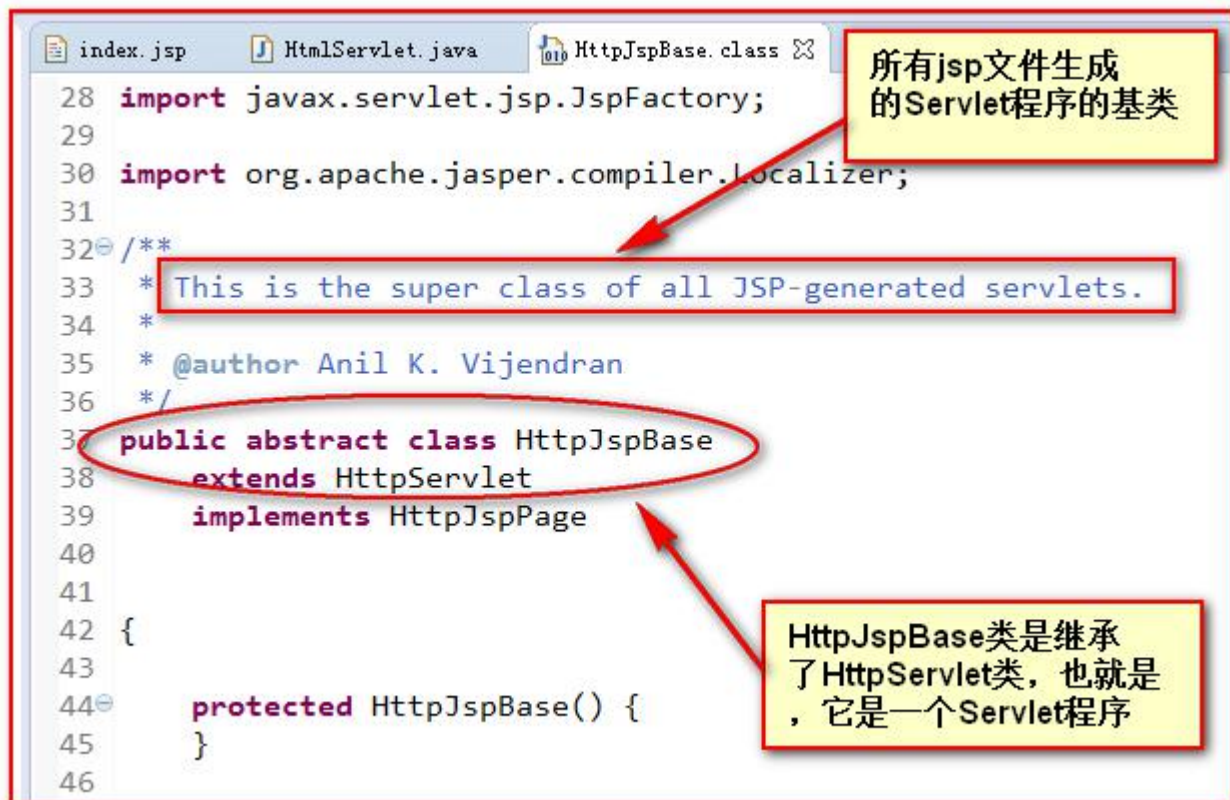
我们打开 index\_jsp.java 文件查看里面的内容：

发现，生成的类继承于 HttpJspBase 类。这是一个 jsp 文件生成 Servlet 程序要继承的基类！！



于是，我们关联源代码。去查看一下 HttpJspBase 类的内容。从源码的类注释说明中，我们发现。HttpJspBase 这个类就是所有 jsp 文件生成 Servlet 程序

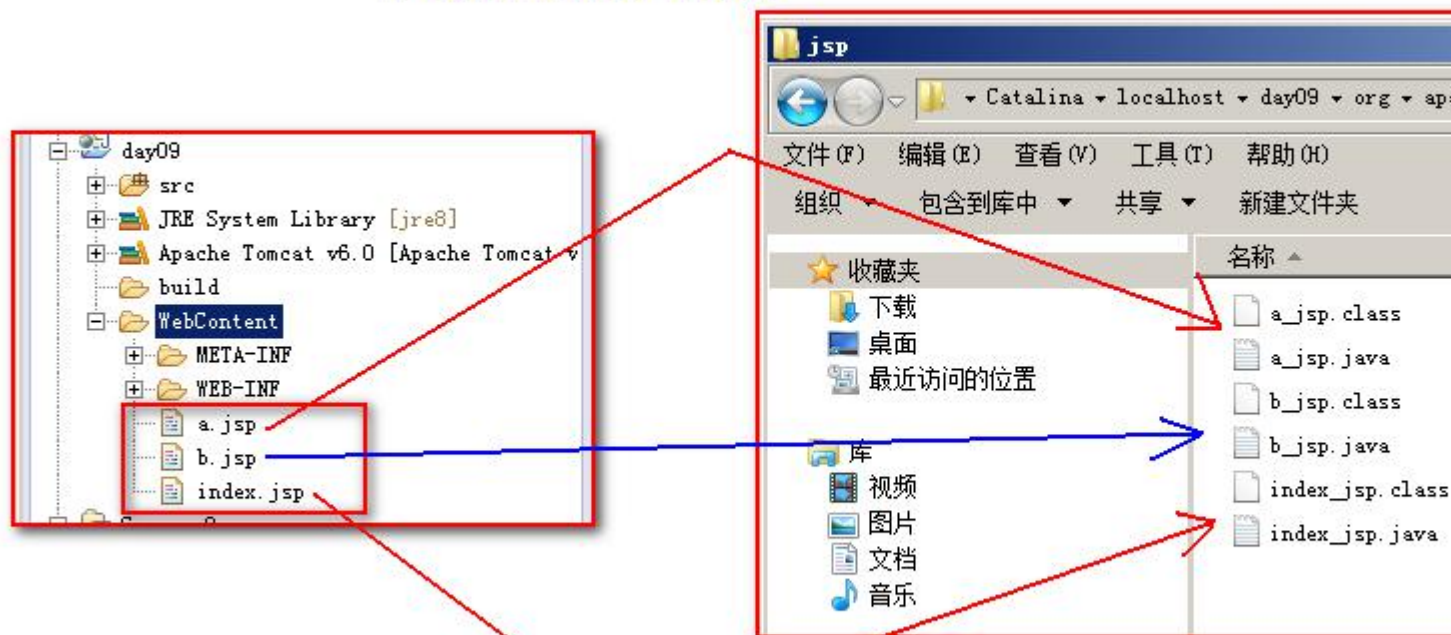
需要去继承的基类。并且这个 `HttpJspBase` 类继承于 `HttpServlet` 类。所以 `jsp` 也是一个 `Servlet` 小程序。



我们分别在工程的 `WebContent` 目录下创建多个 `jsp` 文件。然后依次访问。它们都被翻译为 `.java` 文件并编译成为 `.class` 字节码文件



每个被访问的jsp页面都翻译成了.java程序和编译成.class字节码文件



我们打开 index\_jsp.java 文件查看里面的内容不难发现。jsp 中的 html 页面内容都被翻译到 Servlet 中的 service 方法中直接输出。

不难发现。jsp页面中的html内容被翻译到Servlet程序的service方法中原样输出，这就是我们说jsp是专门用来输出html页面的Servlet程序

```
index.jsp
1 <%@ page language="java" contentType="text/html; charset="
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; char
7 <title>Insert title here</title>
8 </head>
9 <body>
10     这是jsp页面输出的内容
11 </body>
12 </html>
```

```
index_jsp.java
10
46 _jspx_page
47 applicatio
48 config = pa
49 session = p
50 out = pageC
51 _jspx_out =
52
53 out.write("
54 out.write("
55 out.write("
56 out.write("
57 out.write("
58 out.write("
59 out.write("
60 out.write("
61 out.write("
62 out.write("
63 out.write("
64 } catch (Thro
65 if (!t ins
```

## 小结:

从生成的文件我们不难发现一个规则。

a.jsp 翻译成 java 文件后的全名是 a\_jsp.java 文件  
b.jsp 翻译成 java 文件后的全名是 b\_jsp.java 文件

那么 当我们访问 一个 xxx.jsp 文件后 翻译成 java 文件的全名是 xxx\_jsp.java 文件  
xxx\_jsp.java 文件是一个 Servlet 程序。原来 jsp 中的 html 内容都被翻译到 Servlet 类的 service 方法中原样输出。

## 4、jsp 的语法（重点掌握）

## 4.1、jsp 文件头部声明介绍（page 指令介绍）

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

这是 jsp 文件的头声明。表示这是 jsp 页面。

language 属性	值只能是 java。表示翻译的得到的是 java 语言的
contentType 属性	设置响应头 contentType 的内容
pageEncoding 属性	设置当前 jsp 页面的编码
import 属性	给当前 jsp 页面导入需要使用的类包
autoFlush 属性	设置是否自动刷新 out 的缓冲区，默认为 true
buffer 属性	设置 out 的缓冲区大小。默认为 8KB
errorPage 属性	设置当前 jsp 发生错误后，需要跳转到哪个页面去显示错误信息
isErrorPage 属性	设置当前 jsp 页面是否是错误页面。是的话，就可以使用 exception 异常对象
session 属性	设置当前 jsp 页面是否获取 session 对象，默认为 true
extends 属性	给服务器厂商预留的 jsp 默认翻译的 servlet 继承于什么类

## 4.2、jsp 中的三种脚本介绍

### 1) 第一种，声明脚本：

声明脚本格式如下：

```
<%!  
    java 代码  
%>
```

在声明脚本块中，我们可以干 4 件事情

1. 我们可以定义全局变量。
2. 定义 static 静态代码块
3. 定义方法
4. 定义内部类

几乎可以写在类的内部写的代码，都可以通过声明脚本来实现



声明脚本中的代码都被——翻译成java中的代码

```

10 <body>
11 <!-- 这是jsp中的 声明脚本 --%>
12 <%!
13 // 1.我们可以定义全局变量。
14 public int a = 12;
15 private String str = "strValue";
16 private static HashMap<String,String> temp;
17
18 // 2.定义static静态代码块
19 static {
20     temp = new HashMap<String,String>();
21     temp.put("abc", "abcValue");
22     System.out.println("这是静态代码块");
23 }
24 // 3.定义方法
25 public void abc() {
26     System.out.println("这是定义的abc方法");
27 }
28 // 4.定义内部类
29 class AA{
30     public int a = 12;
31     private String str = "strValue";
32     public AA(){
33
34 }

```

```

8 public final class a_jsp extends org.apache.jasper.runtime
9     implements org.apache.jasper.runtime.JspRuntimeLibrary
10
11 // 1.我们可以定义全局变量。
12 public int a = 12;
13 private String str = "strValue";
14 private static HashMap<String,String> temp;
15
16 // 2.定义static静态代码块
17 static {
18     temp = new HashMap<String,String>();
19     temp.put("abc", "abcValue");
20     System.out.println("这是静态代码块");
21 }
22 // 3.定义方法
23 public void abc() {
24     System.out.println("这是定义的abc方法");
25 }
26 // 4.定义内部类
27 class AA{
28     public int a = 12;
29     private String str = "strValue";
30     public AA(){
31
32 }

```

## 2) 第二种，表达式脚本（\*\*\*重点，使用很多）：

表达式脚本格式如下：

<%=表达式 %>

表达式脚本 用于向页面输出内容。

表达式脚本 翻译到 Servlet 程序的 service 方法中 以 out.print() 打印输出

out 是 jsp 的一个内置对象，用于生成 html 的源代码

注意：表达式不要以分号结尾，否则会报错

表达式脚本可以输出任意类型。

比如：

- 1.输出整型
- 2.输出浮点型
- 3.输出字符串
- 4.输出对象

表达式脚本可以输出任意类型

```
35 <!-- 表达式脚本 -->
36 <%=12 %> <!--输出整型 -->
37 <%=12.12312 %> <!--输出浮点型 -->
38 <%="1234123" %> <!--输出字符串 -->
39 <%=temp %> <!--输出对象 -->
40 </body>
```

```
a_jsp.java x
76
77 out.write("\r\n");
78 out.write("\r\n");
79 out.write("<!DOCTYPE h");
80 out.write("<html>\r\n");
81 out.write("<head>\r\n");
82 out.write("<meta http-");
83 out.write("<title>Inse");
84 out.write("</head>\r\n");
85 out.write("<body>\r\n");
86 out.write('\r');
87 out.write('\n');
88 out.write('\r');
89 out.write('\n');
90 out.write('\r');
91 out.write('\n');
92 out.print(12);
93 out.write(' ');
94 out.write('\r');
95 out.write('\n');
96 out.print(12.12312);
97 out.write(' ');
98 out.write('\r');
99 out.write('\n');
100 out.print("1234123");
101 out.write(' ');
102 out.write('\r');
103 out.write('\n');
104 out.print(temp);
105 out.write(' ');
106 out.write("\r\n");
107 out.write("</body>\r\n");
108 out.write("</html>");
```

### 3) 第三种，代码脚本（\*\*\*\*\*重点，使用最多）：

代码脚本如下：

```
<% java 代码 %>
```

代码脚本里可以书写任意的 java 语句。

代码脚本的内容都会被翻译到 service 方法中。

所以 service 方法中可以写的 java 代码，都可以书写到代码脚本中

每段代码脚本都得到了对应的翻译成为java

```

41 <!-- 代码脚本1 -->
42 <%
43     int a = 0;
44     if (a == 0) {
45         System.out.println("a == 0");
46     } else {
47         System.out.println("a != 0");
48     }
49 %>

```

```

110
111     int a = 0;
112     if (a == 0) {
113         System.out.println("a == 0");
114     } else {
115         System.out.println("a != 0");
116     }
117

```

```

50 <!-- 代码脚本2 -->
51 <%
52     int b = 0;
53     if (b == 0) {
54 %><%
55         System.out.println("b == 0");
56     } else {
57 %><%
58         System.out.println("b != 0");
59     }
60 %>

```

```

122
123     int b = 0;
124     if (b == 0) {
125
126
127         System.out.println("b == 0");
128     } else {
129
130
131         System.out.println("b != 0");
132     }

```

```

61 <!--代码脚本3 -->
62 <%
63     for (int i = 0; i < 10; i++) {
64 %><%= "for循环: i=" + i %><br/>
65 <%
66     }
67 %>
68

```

```

138
139     for (int i = 0; i < 10; i++) {
140
141         out.print("for循环: i=" + i);
142         out.write("<br/>");
143     }
144
145

```

## 4.3、jsp 中的注释：

// 单行 java 注释

/\*

多行 java 代码注释

\*/

单行注释和多行注释能在翻译后的 java 源代码中看见。

<!-- jsp 注释 -->

jsp 注释在翻译的时候会直接被忽略掉

<!-- html 注释 -->

## 5、jsp 九大内置对象

我们打开翻译后的 java 文件。查看\_jspService 方法。

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

    PageContext pageContext = null;
    HttpSession session = null;
    Throwable exception = org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);
    if (exception != null) {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    }
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
```

那么 jsp 中九大内置对象分别是：

- |                |  |
|----------------|--|
| request 对象     | 请求对象，可以获取请求信息  |
| response 对象    | 响应对象。可以设置响应信息  |
| pageContext 对象 | 当前页面上下文对象。可以在当前上下文保存属性信息   |
| session 对象     | 会话对象。可以获取会话信息。   |
| exception 对象   | 异常对象只有在 jsp 页面的 page 指令中设置 <code>isErrorPage="true"</code> 的时候才会存在 |
| application 对象 | ServletContext 对象实例，可以获取整个工程的一些信息。                                 |
| config 对象      | ServletConfig 对象实例，可以获取 Servlet 的配置信息                              |
| out 对象         | 输出流。   |
| page 对象        | 表示当前 Servlet 对象实例（无用，用它不如使用 this 对象）。                              |

九大内置对象，都是我们可以在【代码脚本】中或【表达式脚本】中直接使用的对象。



## 6、jsp 四大域对象

四大域对象经常用来保存数据信息。

pageContext	可以保存数据在同一个 jsp 页面中使用
request	可以保存数据在同一个 request 对象中使用。经常用于在转发的时候传递数据
session	可以保存在一个会话中使用
application(ServletContext)	就是 ServletContext 对象

四个作用域的测试代码：

新建两个 jsp 页面。分别取名叫：context1.jsp，context2.jsp

1) context1.jsp 的页面代码如下：

```
<%@ 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>
    这是 context1 页面<br/>
    <%
        //设置 page 域的数据
        pageContext.setAttribute("key", "pageContext-value");
        //设置 request 域的数据
        request.setAttribute("key", "request-value");
        //设置 session 域的数据
        session.setAttribute("key", "session-value");
        //设置 application 域的数据
        application.setAttribute("key", "application-value");
    %>
    <!-- 测试当前页面作用域 -->
    <%=pageContext.getAttribute("key") %><br/>
    <%=request.getAttribute("key") %><br/>
    <%=session.getAttribute("key") %><br/>
    <%=application.getAttribute("key") %><br/>
    <%
        // 测试 request 作用域
    %>
    // request.getRequestDispatcher("/context2.jsp").forward(request, response);
```

```
%>
</body>
</html>
```

2) context2.jsp 的页面代码如下:

```
<%@ 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>
    这是 context2 页面 <br/>
    <%=pageContext.getAttribute("key") %><br/>
    <%=request.getAttribute("key") %><br/>
    <%=session.getAttribute("key") %><br/>
    <%=application.getAttribute("key") %><br/>
</body>
</html>
```

**测试 pageContext 作用域步骤:**

直接访问 context1.jsp 文件

**测试 request 作用域步骤:**

1. 在 context1.jsp 文件中添加转发到 context2.jsp (有数据)
2. 直接访问 context2.jsp 文件 (没有数据)

**测试 session 作用域步骤:**

1. 访问完 context1.jsp 文件
2. 关闭浏览器。但是要保持服务器一直开着
3. 打开浏览器，直接访问 context2.jsp 文件

**测试 application 作用域步骤:**

1. 访问完 context1.jsp 文件，然后关闭浏览器
2. 停止服务器。再启动服务器。
3. 打开浏览器访问 context2.jsp 文件

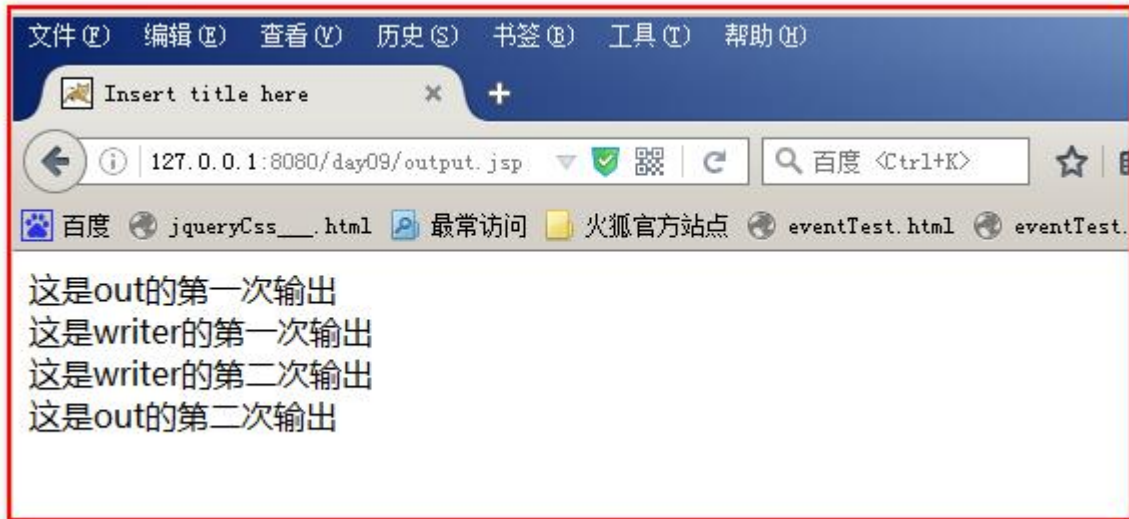
## 7、jsp 中 out 输出流 和 response.getWriter() 输出流

### 1) jsp 中 out 和 response 的 writer 的区别演示

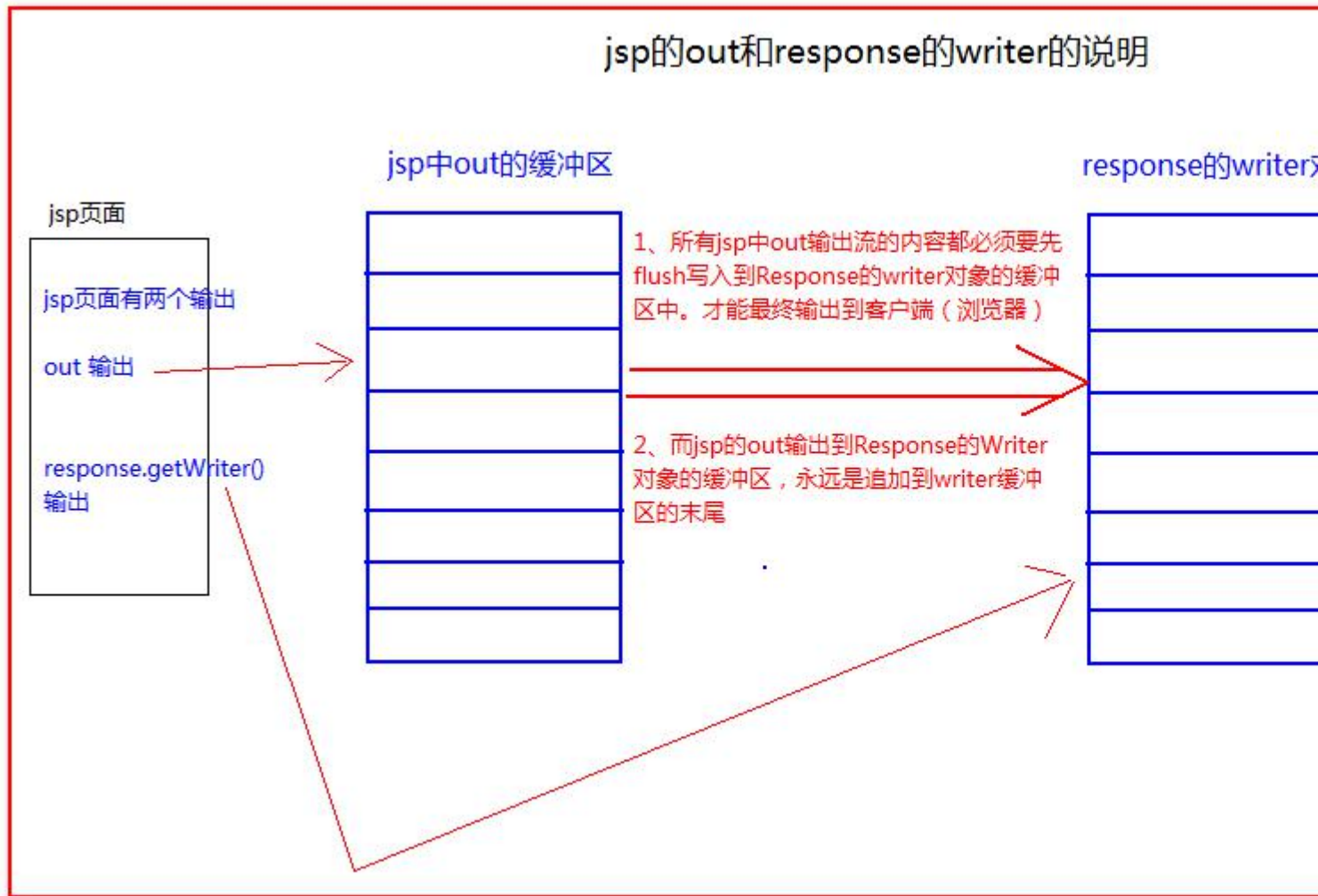
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%
        // out 输出
        out.write("这是 out 的第一次输出<br/>");
        // out flush 之后。会把输出的内容写入 writer 的缓冲区中
        out.flush();
        // 最后一次的输出，由于没有手动 flush，会在整个页面输出到客户端的时候，自动写入到 writer
        缓冲区
        out.write("这是 out 的第二次输出<br/>");
        // writer 的输出
        response.getWriter().write("这是 writer 的第一次输出<br/>");
        response.getWriter().write("这是 writer 的第二次输出<br/>");
    %>
</body>
</html>
```

在浏览器里输入 <http://127.0.0.1:8080/day09/output.jsp> 运行查看的结果：





## 2) 图解 out 流和 writer 流的两个缓冲区如何工作



## 8、jsp 的常用标签（重点\*\*\*\*）

```
<%-- 静态包含 --%>
<%-- 动态包含 --%>
<%-- 转发 --%>
```

## 1) 静态包含--很常用

```
<%@ include file="" %>
```

静态包含是把包含的页面内容原封装不动的输出到包含的位置。

## 2) 动态包含--很少用

```
<jsp:include page=""></jsp:include>
```

动态包含会把包含的 jsp 页面单独翻译成 servlet 文件，然后在执行到时候再调用翻译的 servlet 程序。并把计算的结果返回。

动态包含是在执行的时候，才会加载。所以叫动态包含。

## 3) 页面转发--常用

```
<jsp:forward page=""></jsp:forward>
```

<jsp:forward 转发功能相当于

`request.getRequestDispatcher("/xxxx.jsp").forward(request, response);` 的功能。

## 静态包含和动态包含的区别：

	静态包含	动态包含
是否生成 java 文件	不生成	生成
service 方法中的区别	把包含的内容原封拷贝到 service 中	JspRuntimeLibrary.include 方法
是否可以传递参数	不能	可以
编译次数	1	包含的文件 + 1
适用范围	适用包含纯静态内容(CSS,HTML,JS), 或没有非常耗时操作。或大量 java 代码的 jsp	包含需要传递参数。含有大量 java 代码的 jsp

在这里需要补充说明一点：我们在工作中，几乎都是使用静态包含。理由很简单。因为 jsp 页面虽然可以写 java 代

码，做其他的功能操作。但是由于 jsp 在开发过程中被定位为专门用来展示页面的技术。也就是说。jsp 页面中，基本上只有 html，css，js。还有一些简单的 EL，表达式脚本等输出语句。所以我们都使用静态包含。

## 练习题：

**练习 1：**输出一表格，里有 **20** 个学生的信息，  
表格包含学号，姓名，年龄，电话，操作（删除，修改）  
的信息。

表格样式：

```
<style type="text/css">
    table{
        width: 500px;
        border: 1px solid red;
        border-collapse: collapse;
    }
    th , td{
        border: 1px solid red;
    }
</style>
```

# 1、什么是 Listener 监听器

什么是监听器？监听器就是实时监视一些事物状态的程序，我们称为监听器。  
就好像朝阳群众？朝阳区只要有哪个明星有什么不好的事，他们都会知道，然后举报。  
那么朝阳群众就是监听器，明星就是被监视的事物，举报就是响应的内容。

又或者说是，电动车的报警器。当报警器锁上的时候。我们去碰电动车，电动车就会报警。  
报警器，就是监听器，电动车就是被监视的对象。报警就是响应的内容。

# 2、ServletContextListener 监听器

javax.servlet.ServletContextListener      ServletContext 监听器

监听器的使用步骤。

第一步：我们需要定义一个类。然后去继承生命周期的监听器接口。

第二步：然后在 Web.xml 文件中配置。

ServletContextListener 监听器，一定要在 web.xml 文件中配置之后才会生效

```
<listener>
  <listener-class>全类名</listener-class>
</listener>
```

生命周期监听器两个方法：

**public void** contextInitialized(ServletContextEvent sce) 是 ServletContext 对象的创建回调

**public void** contextDestroyed(ServletContextEvent sce) 是 ServletContext 对象的销毁回调

我们以 ServletContext 的监听器为例演示如下：

1) 创建一个 ServletContextListenerImpl 类实现 ServletContextListener 接口。

```
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
```

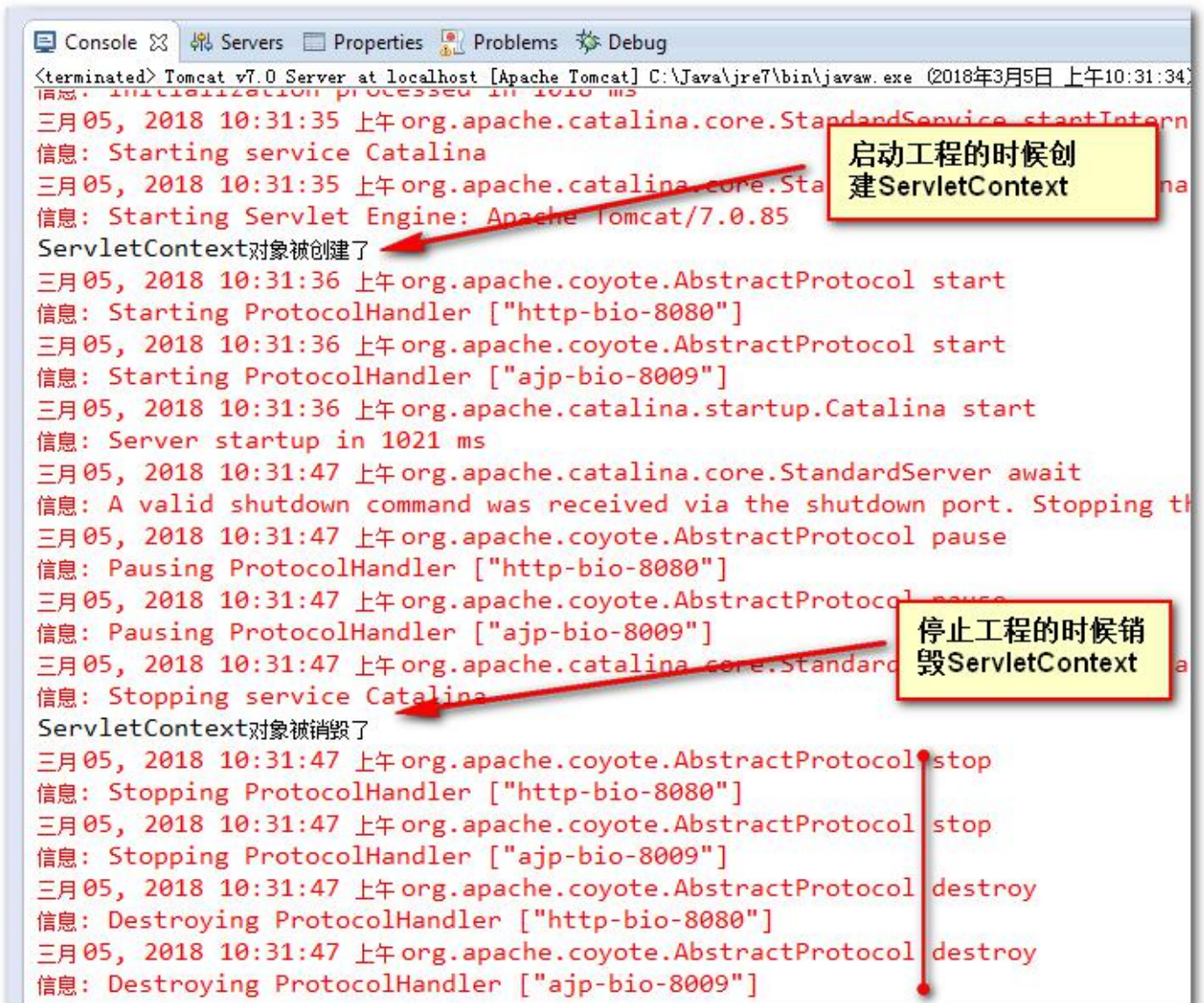
```
public class ServletContextListenerImpl implements ServletContextListener {  
  
    @Override  
    public void contextInitialized(ServletContextEvent sce) {  
        System.out.println("ServletContext 对象被创建了");  
    }  
  
    @Override  
    public void contextDestroyed(ServletContextEvent sce) {  
        System.out.println("ServletContext 对象被销毁了");  
    }  
  
}
```

2) 在 **web.xml** 文件中的配置如下:

```
<listener>  
    <listener-class>com.atguigu.listener.RequestListener</listener-class>  
</listener>
```

3) 这个时候, 启动 **web** 工程和正常停止 **web** 工程, 后台都会如下打印:





```
<terminated> Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Java\jre7\bin\javaw.exe (2018年3月5日 上午10:31:34)
信息: Initialization processed in 1010 ms
三月 05, 2018 10:31:35 上午 org.apache.catalina.core.StandardService.startInternal
信息: Starting service Catalina
三月 05, 2018 10:31:35 上午 org.apache.catalina.core.StandardEngine.start
信息: Starting Servlet Engine: Apache Tomcat/7.0.85
ServletContext对象被创建了
三月 05, 2018 10:31:36 上午 org.apache.coyote.AbstractProtocol.start
信息: Starting ProtocolHandler ["http-bio-8080"]
三月 05, 2018 10:31:36 上午 org.apache.coyote.AbstractProtocol.start
信息: Starting ProtocolHandler ["ajp-bio-8009"]
三月 05, 2018 10:31:36 上午 org.apache.catalina.startup.Catalina.start
信息: Server startup in 1021 ms
三月 05, 2018 10:31:47 上午 org.apache.catalina.core.StandardServer.await
信息: A valid shutdown command was received via the shutdown port. Stopping the
三月 05, 2018 10:31:47 上午 org.apache.coyote.AbstractProtocol.pause
信息: Pausing ProtocolHandler ["http-bio-8080"]
三月 05, 2018 10:31:47 上午 org.apache.coyote.AbstractProtocol.pause
信息: Pausing ProtocolHandler ["ajp-bio-8009"]
三月 05, 2018 10:31:47 上午 org.apache.catalina.core.StandardService.stop
信息: Stopping service Catalina
ServletContext对象被销毁了
三月 05, 2018 10:31:47 上午 org.apache.coyote.AbstractProtocol.stop
信息: Stopping ProtocolHandler ["http-bio-8080"]
三月 05, 2018 10:31:47 上午 org.apache.coyote.AbstractProtocol.stop
信息: Stopping ProtocolHandler ["ajp-bio-8009"]
三月 05, 2018 10:31:47 上午 org.apache.coyote.AbstractProtocol.destroy
信息: Destroying ProtocolHandler ["http-bio-8080"]
三月 05, 2018 10:31:47 上午 org.apache.coyote.AbstractProtocol.destroy
信息: Destroying ProtocolHandler ["ajp-bio-8009"]
```



