

## 08-jsp

讲师：王振国

### 今日任务

## 1.什么是jsp，它有什么用？

jsp 的全换是 java server pages。Java 的服务器页面。

jsp 的主要作用是代替 Servlet 程序回传 html 页面的数据。

servlet回传页面需要一行行敲

因为 Servlet 程序回传 html 页面数据是一件非常繁琐的事情。开发成本和维护成本都极高。

Servlet 回传 html 页面数据的代码：

```
public class PringHtml extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        // 通过响应的回传流回传 html 页面数据

        resp.setContentType("text/html; charset=UTF-8");

        PrintWriter writer = resp.getWriter();

        writer.write("<!DOCTYPE html>\r\n");
        writer.write(" <html lang=\"en\">\r\n");
        writer.write(" <head>\r\n");
        writer.write("     <meta charset=\"UTF-8\">\r\n");
        writer.write("     <title>Title</title>\r\n");
        writer.write(" </head>\r\n");
        writer.write(" <body>\r\n");
        writer.write("     这是 html 页面数据 \r\n");
        writer.write(" </body>\r\n");
        writer.write("</html>\r\n");
        writer.write("\r\n");

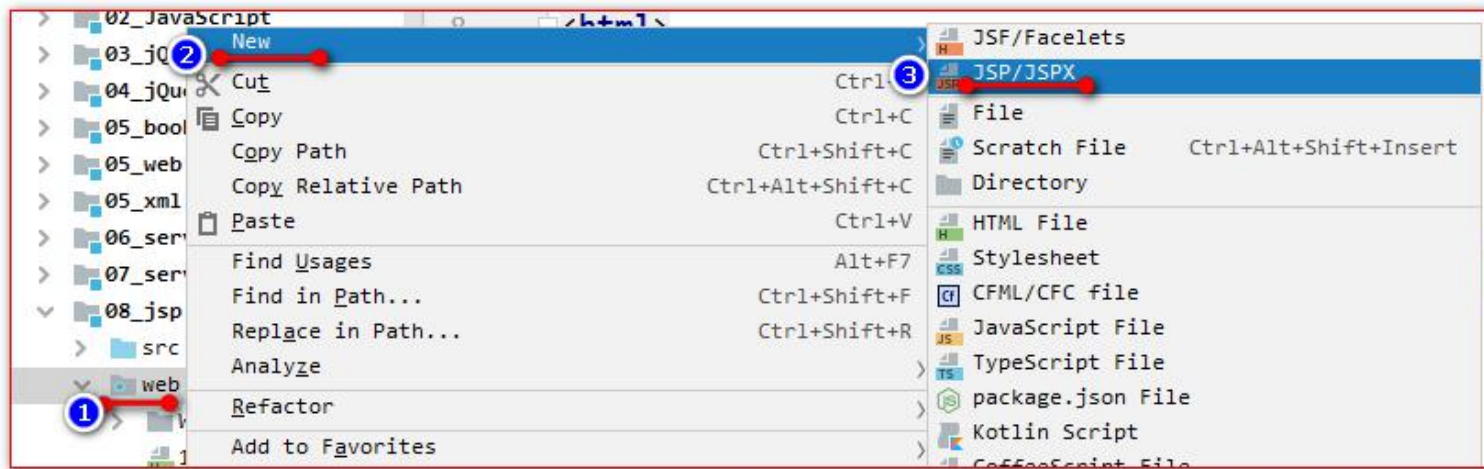
    }
}
```

jsp 回传一个简单 html 页面的代码：

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    这是 html 页面数据
</body>
</html>
```

jsp 的小结：

1、如何创建 jsp 的页面？



输入文件名敲回车即可！！



2、jsp 如何访问：

jsp 页面和 html 页面一样，都是存放在 web 目录下。访问也跟访问 html 页面一样。

比如：

在 web 目录下有如下的文件：

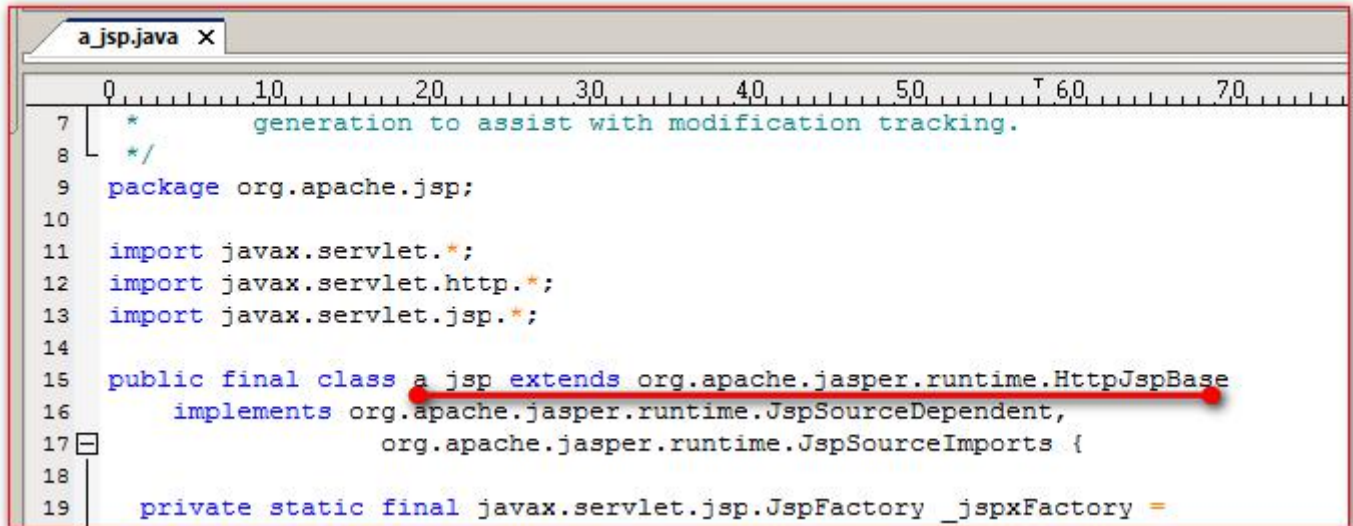
web 目录

a.html 页面	访问地址是 =====>>>>>>>>>>	http://ip:port/工程路径/a.html
b.jsp 页面	访问地址是 =====>>>>>>>>>>	http://ip:port/工程路径/b.jsp

## 2.jsp 的本质是什么。

jsp 页面本质上是一个 Servlet 程序。

当我们第一次访问 jsp 页面的时候。Tomcat 服务器会帮我们把 jsp 页面翻译成为一个 java 源文件。并且对它进行编译成为.class 字节码程序。我们打开 java 源文件不难发现其里面的内容是：



```

1  0
2  10
3  20
4  30
5  40
6  50
7  60
8  70
9  7
10 8
11 9 package org.apache.jsp;
12 10
13 11 import javax.servlet.*;
14 12 import javax.servlet.http.*;
15 13 import javax.servlet.jsp.*;
16 14
17 15 public final class a_jsp extends org.apache.jasper.runtime.HttpJspBase
18 16     implements org.apache.jasper.runtime.JspSourceDependent,
19 17         org.apache.jasper.runtime.JspSourceImports {
20 18
21 19     private static final javax.servlet.jsp.JspFactory _jspxFactory =

```

我们跟踪原代码发现，HttpJspBase 类。它直接地继承了 HttpServlet 类。也就是说。jsp 翻译出来的 java 类，它间接地继承了 HttpServlet 类。也就是说，翻译出来的是一个 Servlet 程序



```

30
31 /**
32  * This is the super class of all JSP-generated servlets.
33  *
34  * @author Anil K. Vijendran
35  */
36 public abstract class HttpJspBase extends HttpServlet implements HttpJspPage {

```

总结：通过翻译的 java 源代码我们就可以得到结果：jsp 就是 Servlet 程序。

大家也可以去观察翻译出来的 Servlet 程序的源代码，不难发现。其底层实现，也是通过输出流。把 html 页面数据回传给客户端。

```

public void _jspService(final javax.servlet.http.HttpServletRequest request, final
javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException, javax.servlet.ServletException {

    final java.lang.String _jspx_method = request.getMethod();
    if (!"GET".equals(_jspx_method) && !"POST".equals(_jspx_method) && !"HEAD".equals(_jspx_method)
    && !javax.servlet.DispatcherType.ERROR.equals(request.getDispatcherType())) {

```

```
response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, "JSPs only permit GET POST or HEAD");
return;
}

final javax.servlet.jsp.PageContext pageContext;
javax.servlet.http.HttpSession session = null;
final javax.servlet.ServletContext application;
final javax.servlet.ServletConfig config;
javax.servlet.jsp.JspWriter out = null;
final java.lang.Object page = this;
javax.servlet.jsp.JspWriter _jspx_out = null;
javax.servlet.jsp.PageContext _jspx_page_context = null;

try {
    response.setContentType("text/html;charset=UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response,
        null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("\r\n");
    out.write("\r\n");
    out.write("<html>\r\n");
    out.write("<head>\r\n");
    out.write("    <title>Title</title>\r\n");
    out.write("</head>\r\n");
    out.write("<body>\r\n");
    out.write("    a.jsp 页面\r\n");
    out.write("</body>\r\n");
    out.write("</html>\r\n");
} catch (java.lang.Throwable t) {
    if (!(t instanceof javax.servlet.jsp.SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            try {
                if (response.isCommitted()) {
                    out.flush();
                } else {
                    out.clearBuffer();
                }
            } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
        else throw new ServletException(t);
    }
}
```

```

    }
} finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
}
}
}

```

## 3.jsp 的三种语法

### a)jsp 头部的 page 指令

jsp 的 page 指令可以修改 jsp 页面中一些重要的属性，或者行为。

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

- i. **language** 属性                      表示 jsp 翻译后是什么语言文件。暂时只支持 java。
- ii. **contentType** 属性                表示 jsp 返回的数据类型是什么。也是源码中 response.setContentType() 参数值
- iii. **pageEncoding** 属性            表示当前 jsp 页面文件本身的字符集。
- iv. **import** 属性                      跟 java 源代码中一样。用于导包，导类。

=====两个属性是给 out 输出流使用=====

- v. **autoFlush** 属性                    设置当 out 输出流缓冲区满了之后，是否自动刷新缓冲区。默认值是 true。
- vi. **buffer** 属性                      设置 out 缓冲区的大小。默认是 8kb

缓冲区溢出错误：

```

org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:396)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:340)
javax.servlet.http.HttpServlet.service(HttpServlet.java:648)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)

```

**out缓冲区满了之后不能自动刷新，就会报错**

**root cause**

```

java.io.IOException: Error: JSP Buffer overflow
    org.apache.jasper.runtime.JspWriterImpl.bufferOverflow(JspWriterImpl.java:158)
    org.apache.jasper.runtime.JspWriterImpl.write(JspWriterImpl.java:328)
    java.io.Writer.write(Writer.java:157)
    org.apache.jsp.b_jsp._jspService(b_jsp.java:159)
    org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:729)

```

=====两个属性是给 out 输出流使用=====

- vii. **errorPage** 属性                设置当 jsp 页面运行时出错，自动跳转去的错误页面路径。

```
<!--
```

**errorPage** 表示错误后自动跳转去的路径 <br/>

这个路径一般都是以斜杠打头，它表示请求地址为 http://ip:port/工程路径/



- |                                   |   |
|-----------------------------------|---|
| viii. <code>isErrorPage</code> 属性 | 设置当前 <code>jsp</code> 页面是否是错误信息页面。默认是 <code>false</code> 。如果是 <code>true</code> 可以获取异常信息。 |
| ix. <code>session</code> 属性       | 设置访问当前 <code>jsp</code> 页面，是否会创建 <code>HttpSession</code> 对象。默认是 <code>true</code> 。      |
| x. <code>extends</code> 属性        | 设置 <code>jsp</code> 翻译出来的 <code>java</code> 类默认继承谁。                                       |

## b)jsp 中的常用脚本

### i. 声明脚本(极少使用)

声明脚本的格式是: `<%! 声明 java 代码 %>`

作用: 可以给 `jsp` 翻译出来的 `java` 类定义属性和方法甚至是静态代码块。内部类等。

练习:

- 1、声明类属性
- 2、声明 `static` 静态代码块
- 3、声明类方法
- 4、声明内部类

代码示例:

```
<!--1、声明类属性-->
<%!
    private Integer id;
    private String name;
    private static Map<String,Object> map;
%>

<!--2、声明 static 静态代码块-->
<%!
    static {
        map = new HashMap<String,Object>();
        map.put("key1", "value1");
        map.put("key2", "value2");
        map.put("key3", "value3");
    }
%>

<!--3、声明类方法-->
<%!
    public int abc(){
        return 12;
    }
%>
```

<!--4, 声明内部类-->

```
<%!
    public static class A {
        private Integer id = 12;
        private String abc = "abc";
    }
%>
```

声明脚本代码翻译对照:

<!--1, 声明类属性-->

```
<%!
    private Integer id;
    private String name;
    private static Map<String,Object> map;
%>
```

```
private Integer id;
private String name;
private static Map<String,Object> map;
```

<!--2, 声明static静态代码块-->

```
<%!
    static {
        map = new HashMap<String,Object>();
        map.put("key1", "value1");
        map.put("key2", "value2");
        map.put("key3", "value3");
    }
%>
```

```
27     static {
28         map = new HashMap<String,Object>();
29         map.put("key1", "value1");
30         map.put("key2", "value2");
31         map.put("key3", "value3");
32     }
```

<!--3, 声明类方法-->

```
<%!
    public int abc(){
        return 12;
    }
%>
```

```
5     public int abc(){
6         return 12;
7     }
```

## ii. 表达式脚本 (常用)

表达式脚本的格式是: `<%=表达式%>`

表达式脚本的作用是: 的 jsp 页面上输出数据。

表达式脚本的特点:

- 1、所有的表达式脚本都会被翻译到 `_jspService()` 方法中
- 2、表达式脚本都会被翻译成为 `out.print()` 输出到页面上
- 3、由于表达式脚本翻译的内容都在 `_jspService()` 方法中,所以 `_jspService()` 方法中的对象都可以直接使用。
- 4、表达式脚本中的表达式不能以分号结束。

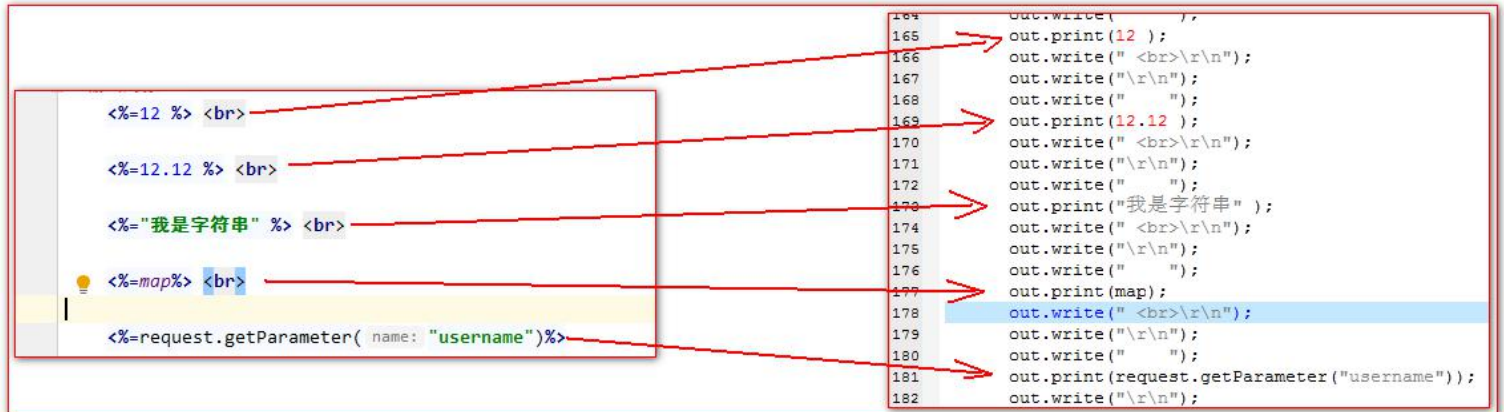
练习:

1. 输出整型
2. 输出浮点型
3. 输出字符串
4. 输出对象

示例代码：

```
<%=12 %> <br>
<%=12.12 %> <br>
<%= "我是字符串" %> <br>
<%=map%> <br>
<%=request.getParameter("username")%>
```

翻译对照：



JSP 表达式	对应的 Java 代码
<code>&lt;%=12 %&gt; &lt;br&gt;</code>	<code>out.print(12);</code>
<code>&lt;%=12.12 %&gt; &lt;br&gt;</code>	<code>out.print(12.12);</code>
<code>&lt;%= "我是字符串" %&gt; &lt;br&gt;</code>	<code>out.print("我是字符串");</code>
<code>&lt;%=map%&gt; &lt;br&gt;</code>	<code>out.print(map);</code>
<code>&lt;%=request.getParameter("username")%&gt;</code>	<code>out.print(request.getParameter("username"));</code>

### iii. 代码脚本

代码脚本的格式是：

`<%`

java 语句

`%>`

代码脚本的作用是：可以在 jsp 页面中，编写我们自己需要的功能（写的是 java 语句）。

代码脚本的特点是：

- 1、代码脚本翻译之后都在 `_jspService` 方法中
- 2、代码脚本由于翻译到 `_jspService()` 方法中，所以在 `_jspService()` 方法中的现有对象都可以直接使用。
- 3、还可以由多个代码脚本块组合完成一个完整的 java 语句。
- 4、代码脚本还可以和表达式脚本一起组合使用，在 jsp 页面上输出数据

练习：

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

示例代码：

```
<!-- 练习: --%>
<!-- 1. 代码脚本 ---- if 语句 --%>
<%
```

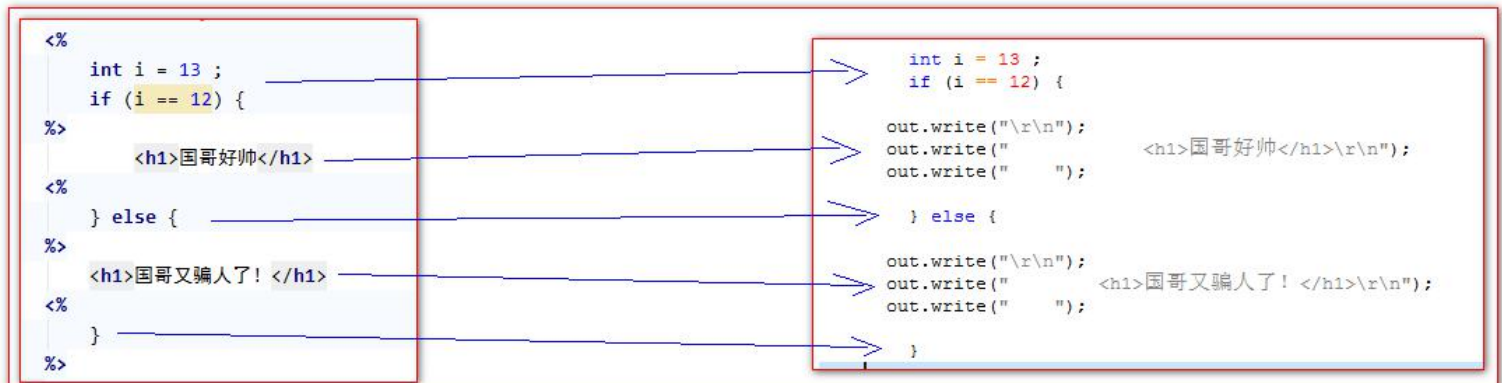


```

    int i = 13 ;
    if (i == 12) {
%>
        <h1>国哥好帅</h1>
    <%
    } else {
%>
        <h1>国哥又骗人了! </h1>
    <%
    }
%>
<br>
<!--2. 代码脚本----for 循环语句-->
<table border="1" cellspacing="0">
<%
    for (int j = 0; j < 10; j++) {
%>
        <tr>
            <td>第 <%=j + 1%>行</td>
        </tr>
    <%
    }
%>
</table>
<!--3. 翻译后 java 文件中_jspService 方法内的代码都可以写-->
<%
    String username = request.getParameter("username");
    System.out.println("用户名的请求参数值是: " + username);
%>

```

翻译之后的对比:



## c)jsp 中的三种注释

### i. html 注释

`<!-- 这是html 注释 -->`

html 注释会被翻译到 java 源代码中。在 `_jspService` 方法里，以 `out.writer` 输出到客户端。

## ii. java 注释

```
<%
// 单行 java 注释
/* 多行 java 注释 */
%>
```

java 注释会被翻译到 java 源代码中。

## iii. jsp 注释

```
<%-- 这是 jsp 注释 --%>
```

jsp 注释可以注掉，jsp 页面中所有代码。

# 4.jsp 九大内置对象

jsp 中的内置对象，是指 Tomcat 在翻译 jsp 页面成为 Servlet 源代码后，内部提供的九大对象，叫内置对象。

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

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

## jsp的九大内置对象

<code>request</code>	请求对象
<code>response</code>	响应对象
<code>pageContext</code>	jsp的上下文对象
<code>session</code>	会话对象
<code>application</code>	ServletContext对象
<code>config</code>	ServletConfig对象
<code>out</code>	jsp输出流对象
<code>page</code>	指向当前jsp的对象
<code>exception</code>	异常对象

## 5.jsp 四大域对象

四个域对象分别是：

pageContext	(PageContextImpl 类)	当前 jsp 页面范围内有效
request	(HttpServletRequest 类)、	一次请求内有效
session	(HttpSession 类)、	一个会话范围内有效（打开浏览器访问服务器，直到关闭浏览器）
application	(ServletContext 类)	整个 web 工程范围内都有效（只要 web 工程不停止，数据都在）

域对象是可以像 Map 一样存取数据的对象。四个域对象功能一样。不同的是它们对数据的存取范围。

虽然四个域对象都可以存取数据。在使用上它们是有优先顺序的。

四个域在使用的时候，优先顺序分别是，他们从小到大的范围的顺序。

pageContext >>> request >>> session >>> application

scope.jsp 页面

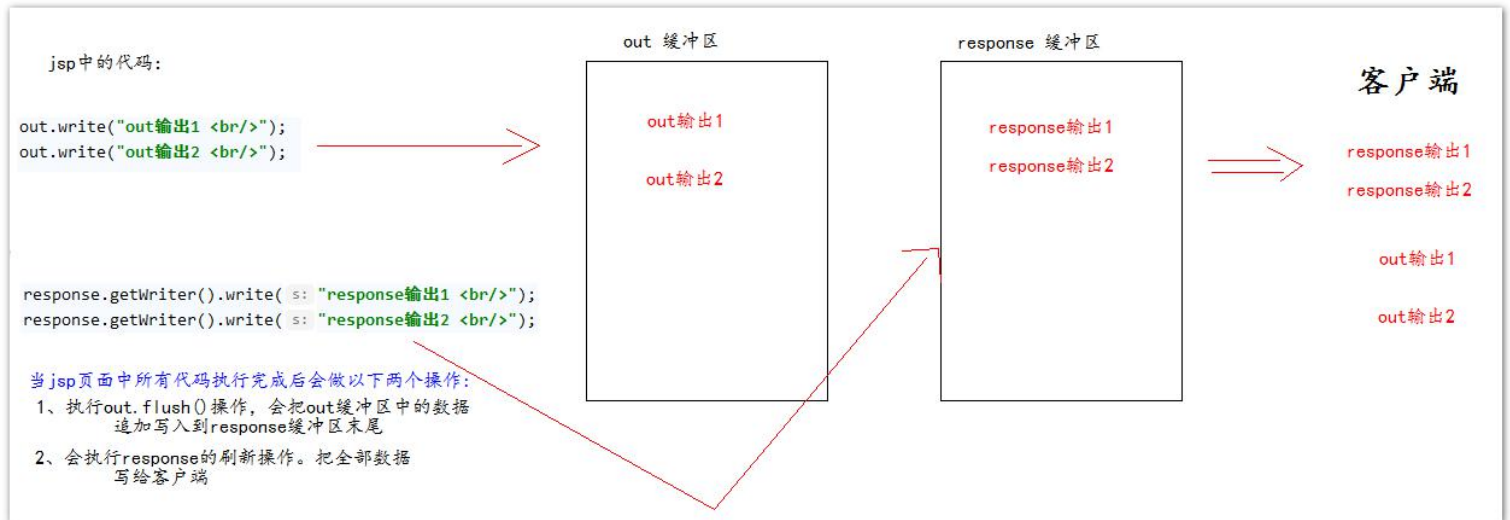
```
<body>
  <h1>scope.jsp 页面</h1>
  <%
    // 往四个域中都分别保存了数据
    pageContext.setAttribute("key", "pageContext");
    request.setAttribute("key", "request");
    session.setAttribute("key", "session");
    application.setAttribute("key", "application");
  %>
  pageContext 域是否有值：<%=pageContext.getAttribute("key")%> <br>
  request 域是否有值：<%=request.getAttribute("key")%> <br>
  session 域是否有值：<%=session.getAttribute("key")%> <br>
  application 域是否有值：<%=application.getAttribute("key")%> <br>
  <%
    request.getRequestDispatcher("/scope2.jsp").forward(request,response);
  %>
</body>
```

scope2.jsp 页面

```
<body>
  <h1>scope2.jsp 页面</h1>
  pageContext 域是否有值：<%=pageContext.getAttribute("key")%> <br>
  request 域是否有值：<%=request.getAttribute("key")%> <br>
  session 域是否有值：<%=session.getAttribute("key")%> <br>
  application 域是否有值：<%=application.getAttribute("key")%> <br>
</body>
```

# 6.jsp 中的 out 输出和 response.getWriter 输出的区别

response 中表示响应，我们经常用于设置返回给客户端的内容（输出）  
out 也是给用户做输出使用的。



由于 jsp 翻译之后，底层源代码都是使用 out 来进行输出，所以一般情况下。我们在 jsp 页面中统一使用 out 来进行输出。避免打乱页面输出内容的顺序。

out.write() 输出字符串没有问题  
out.print() 输出任意数据都没有问题（都转换成为字符串后调用的 write 输出）

深入源码，浅出结论：在 jsp 页面中，可以统一使用 out.print() 来进行输出

## 7.jsp 的常用标签

### a)jsp 静态包含

示例说明：

```
<%--
<%@ include file=""%> 就是静态包含
file 属性指定你要包含的 jsp 页面的路径
地址中第一个斜杠 / 表示为 http://ip:port/工程路径/ 映射到代码的 web 目录
```

静态包含的特点：

- 1、静态包含不会翻译被包含的 jsp 页面。
- 2、静态包含其实是把被包含的 jsp 页面的代码拷贝到包含的位置执行输出。

```
--%>
```

```
<%@ include file="/include/footer.jsp"%>
```

## b)jsp 动态包含

示例说明：

```
<%--
```

```
<jsp:include page=""></jsp:include>    这是动态包含
page 属性是指定你要包含的 jsp 页面的路径
动态包含也可以像静态包含一样。把被包含的内容执行输出到包含位置
```

动态包含的特点：

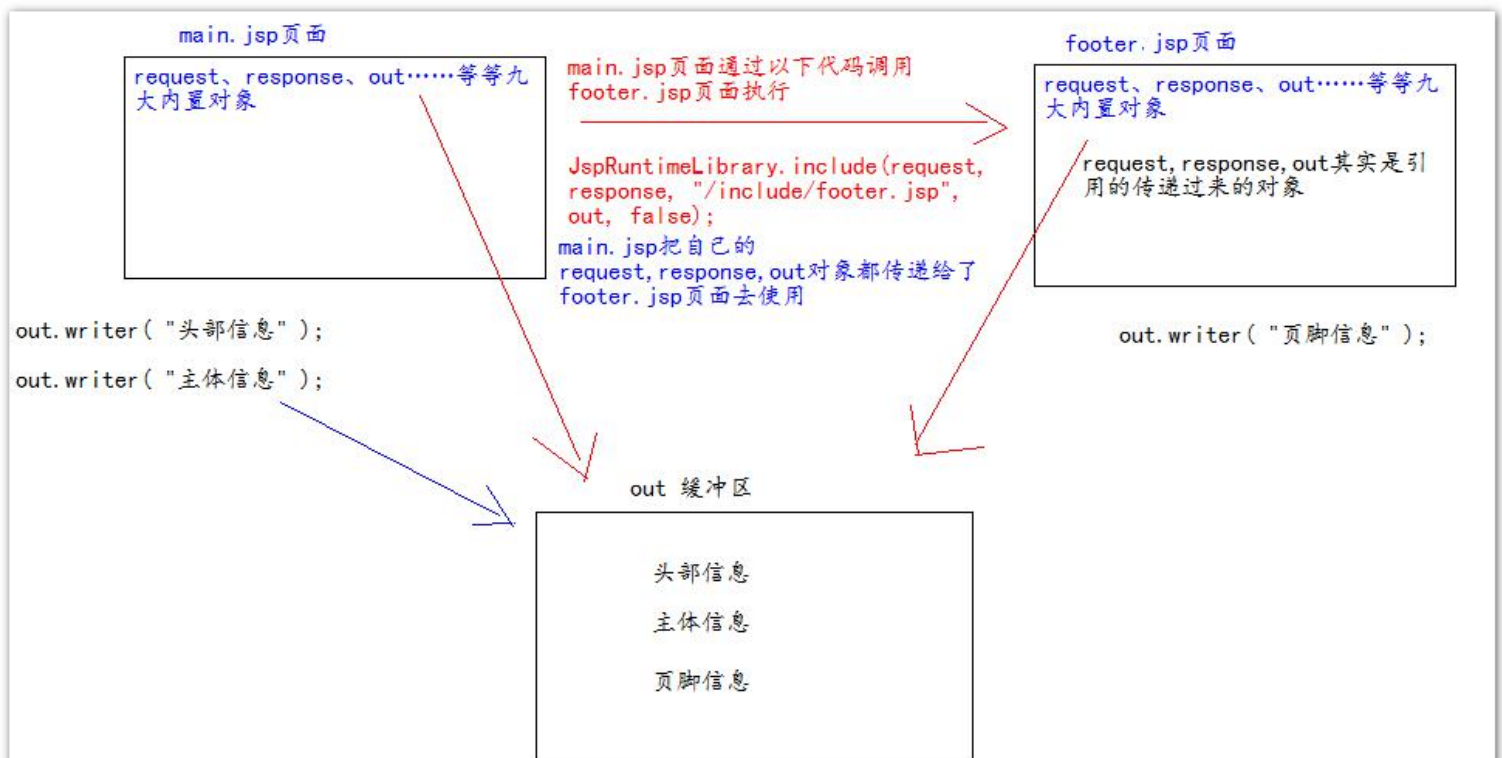
- 1、动态包含会把包含的 jsp 页面也翻译成为 java 代码
- 2、动态包含底层代码使用如下代码去调用被包含的 jsp 页面执行输出。  

```
JspRuntimeLibrary.include(request, response, "/include/footer.jsp", out, false);
```
- 3、动态包含，还可以传递参数

```
--%>
```

```
<jsp:include page="/include/footer.jsp">
  <jsp:param name="username" value="bbj"/>
  <jsp:param name="password" value="root"/>
</jsp:include>
```

动态包含的底层原理：





## c)jsp 标签-转发

示例说明：

```
<%--  
    <jsp:forward page=""></jsp:forward> 是请求转发标签，它的功能就是请求转发  
    page 属性设置请求转发的路径  
--%>  
<jsp:forward page="/scope2.jsp"></jsp:forward>
```

## 8、jsp 的练习题

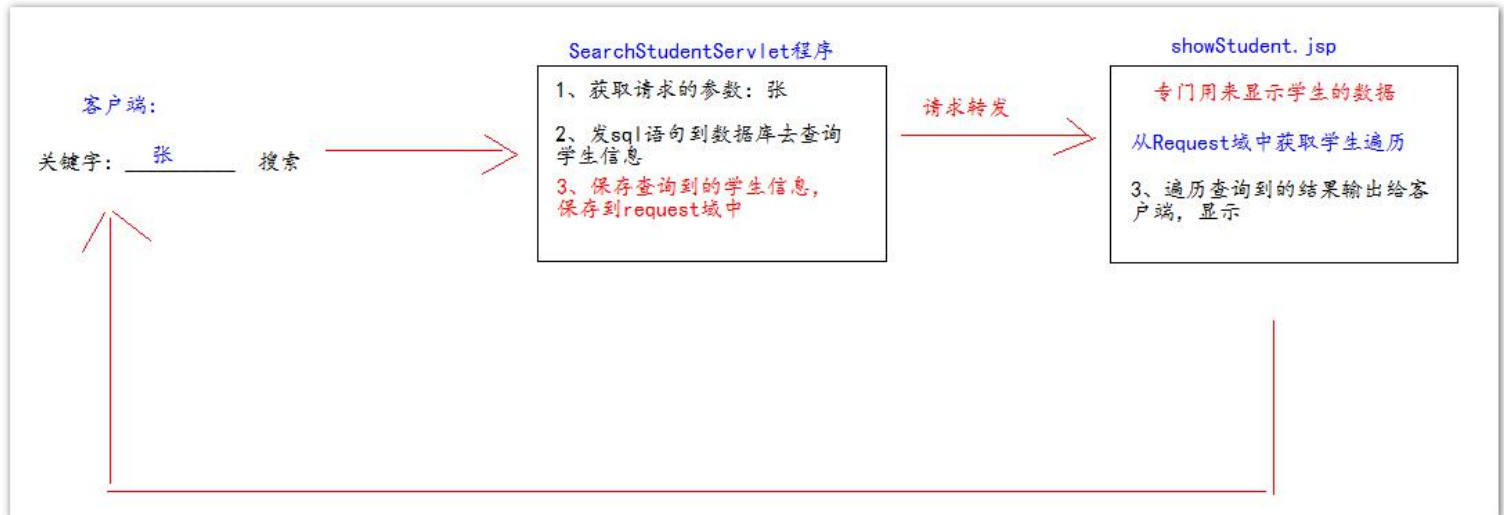
### 练习一：在 jsp 页面中输出九九乘法口诀表

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
    <style type="text/css">  
  
        table{  
            width: 650px;  
        }  
  
    </style>  
</head>  
<body>  
    <%-- 练习一：在 jsp 页面中输出九九乘法口诀表 --%>  
    <h1 align="center">九九乘法口诀表</h1>  
    <table align="center">  
    <%-- 外层循环遍历行 --%>  
    <% for (int i = 1; i <= 9; i++) { %>  
        <tr>  
            <%-- 内层循环遍历单元格 --%>  
            <% for (int j = 1; j <= i ; j++) { %>  
                <td><%=j + "x" + i + "=" + (i*j)%></td>  
            <% } %>  
        </tr>  
    <% } %>  
</table>
```

```
</body>
```

```
</html>
```

## 练习二：jsp 输出一个表格，里面有 10 个学生信息。



Student 类:

```
public class Student {
    private Integer id;
    private String name;
    private Integer age;
    private String phone;
}
```

SearchStudentServlet 程序:

```
public class SearchStudentServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // 获取请求的参数
        // 发 sql 语句查询学生的信息
        // 使用 for 循环生成查询到的数据做模拟
        List<Student> studentList = new ArrayList<Student>();
        for (int i = 0; i < 10; i++) {
            int t = i + 1;
            studentList.add(new Student(t, "name"+t, 18+t, "phone"+t));
        }
        // 保存查询到的结果 (学生信息) 到 request 域中
        req.setAttribute("stuList", studentList);
        // 请求转发到 showStudent.jsp 页面
    }
}
```

```
req.getRequestDispatcher("/test/showStudent.jsp").forward(req, resp);
}
}
```

showStudent.jsp 页面

```
<%@ page import="java.util.List" %>
<%@ page import="com.atguigu.pojo.Student" %>
<%@ page import="java.util.ArrayList" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <style>
        table{
            border: 1px blue solid;
            width: 600px;
            border-collapse: collapse;
        }
        td,th{
            border: 1px blue solid;
        }
    </style>
</head>
<body>
<!-- 练习二: jsp 输出一个表格, 里面有10 个学生信息. -->
<%
    List<Student> studentList = (List<Student>) request.getAttribute("stuList");
%>
<table>
    <tr>
        <td>编号</td>
        <td>姓名</td>
        <td>年龄</td>
        <td>电话</td>
        <td>操作</td>
    </tr>
    <% for (Student student : studentList) { %>
        <tr>
            <td><%=student.getId()%></td>
            <td><%=student.getName()%></td>
            <td><%=student.getAge()%></td>
            <td><%=student.getPhone()%></td>
            <td>删除、修改</td>
        </tr>
    <% } %>
</table>
```

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

## 9、什么是 Listener 监听器？

- 1、Listener 监听器它是 **JavaWeb 的三大组件之一**。JavaWeb 的三大组件分别是：Servlet 程序、Filter 过滤器、Listener 监听器。
- 2、Listener 它是 **JavaEE 的规范，就是接口**
- 3、监听器的作用是，**监听某种事物的变化。然后通过回调函数，反馈给客户（程序）去做一些相应的处理。**

### 9.1、ServletContextListener 监听器

ServletContextListener 它可以**监听 ServletContext 对象的创建和销毁**。

ServletContext 对象在 web 工程启动的时候创建，在 web 工程停止的时候销毁。

监听到创建和销毁之后都会**分别调用 ServletContextListener 监听器的方法反馈**。

两个方法分别是：

```
public interface ServletContextListener extends EventListener {
    /**
     * 在 ServletContext 对象创建之后马上调用，做初始化
     */
    public void contextInitialized(ServletContextEvent sce);
    /**
     * 在 ServletContext 对象销毁之后调用
     */
    public void contextDestroyed(ServletContextEvent sce);
}
```

**如何使用** ServletContextListener 监听器监听 ServletContext 对象。

使用步骤如下：

- 1、编写一个类去实现 ServletContextListener
- 2、实现其两个回调方法
- 3、**到 web.xml 中去配置监听器**

监听器实现类:

```
public class MyServletContextListenerImpl 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");
    }
}
```

web.xml 中的配置:

```
<!-- 配置监听器 -->
<listener>
    <listener-class>com.atguigu.listener.MyServletContextListenerImpl</listener-class>
</listener>
```