JSP基础

(Java Server Page)

王新阳

wxyyuppie@bjfu.edu.cn

主要内容

- JSP基本介绍
- JSP脚本元素
  - ✓声明、代码段、输出
- JSP指令 <%@ ... ...%>
  - ✓page、include、taglib
- JSP标准动作
  - ✓forward、include、useBean
- JSP内置对象
  - ✓request、response、out、session、application等

一、JSP基本介绍

一、JSP基本介绍——Servlet存在的问题

- 优点
  - 读取和处理表单数据
  - 读取HTTP请求报头并设置HTTP响应报头
  - 使用Cookie实现会话跟踪和数据共享

- 缺点
  - 虽然能通过HTML显示Servlet处理结果，但输出实在是麻烦，不能方便的设计页面布局

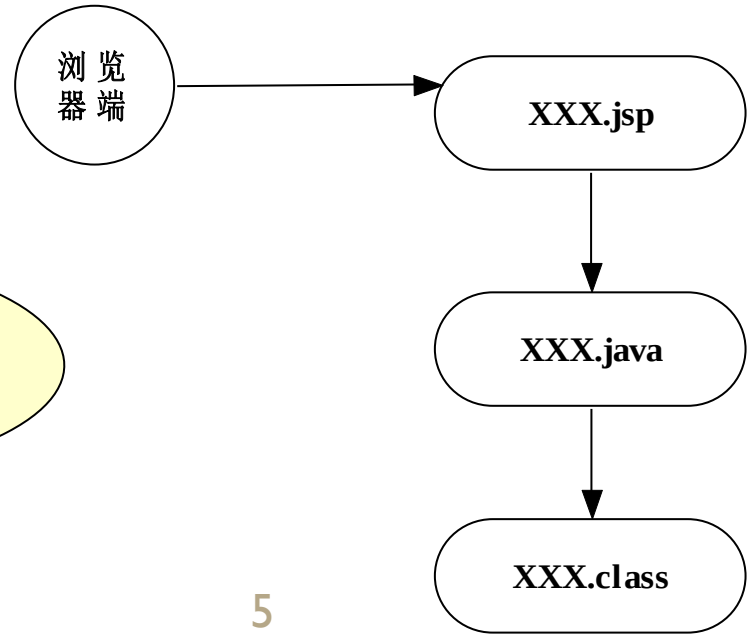
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    String countString =getCookieValue(request, "accessCount", "1");
        int count = 1;
        try {
          count = Integer.parseInt(countString);
        } catch(NumberFormatException nfe) { }
        LongLivedCookie c =
          new LongLivedCookie("accessCount",
                              String.valueOf(count+1));
        response.addCookie(c);
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String title = "通过Cookie获取访问次数";
        String docType =
          "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
          "Transitional//EN\">\n";
        out.println(docType +
                    "<HTML>\n" +
                    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                    "<CENTER>\n" +
                    "<H1>" + title + "</H1>\n" +
                    "<H2>欢迎您第 " +
                    count + " 次访问</H2>\n" +
                    "</CENTER></BODY></HTML>");
    }
```

一、JSP基本介绍——JSP的思想

- 基本思想
  - Web页面大部分使用常规的HTML
  - 用特殊的标签将servlet代码标记出来
  - 每个JSP页面最终转换成servlet(仅执行一次)
  - 请求页面时实际被调用的是servlet

```
<HTML>
<HEAD>
<TITLE>订 单 确 认 </TITLE>
</HEAD>

<BODY>
<H2>确 认 您 的 订 单 ！ </H2>
感 谢 您 订 购
<I><%= request.getParameter("title") %></I>!
</BODY></HTML>
```

JSP元素
<%.....%>



一、JSP基本介绍——JSP处理过程

1. 客户浏览器给服务器发送一个HTTP请求；
2. Web服务器识别出这是一个JSP网页的请求，并且将该请求传递给JSP引擎。通过使用URL或.jsp文件来完成；
3. 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；
4. 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；
5. Web服务器的某组件将会调用servlet引擎，然后载入并执行servlet类。在执行的过程中，servlet产生HTML格式的输出并将其内嵌与HTTP的response上交给Web服务器；
6. Web服务器以静态HTML网页的形式将HTTP的response返回给浏览器；
7. 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；

一、JSP基本介绍——JSP的本质

- JSP仍然是服务器端技术
  - 必须放在支持JSP容器的服务器端才能执行，例如Tomcat6.0。IIS就不行。
    - ✓容器：是指支持编译运行的环境
- JSP页面的转换和执行
  - 在该页面首次被请求时: Jsp→java→class,访问的是class文件
  - 在该页面再次被请求时:直接访问class文件

一、JSP基本介绍——JSP示例

JSPTest.war
中的
HelloDate.jsp

```
<%@ page import="java.util.*" %>
<%@ page contentType="text/html;
                charset=gb2312" %>
<HTML>
 <BODY>
  你好,今天是
 <%
  Date today=new Date();
 %>
  <%=today.getDate()%>号,
  星期<%=today.getDay()%>
 </BODY>
</HTML>
```

JSP文件

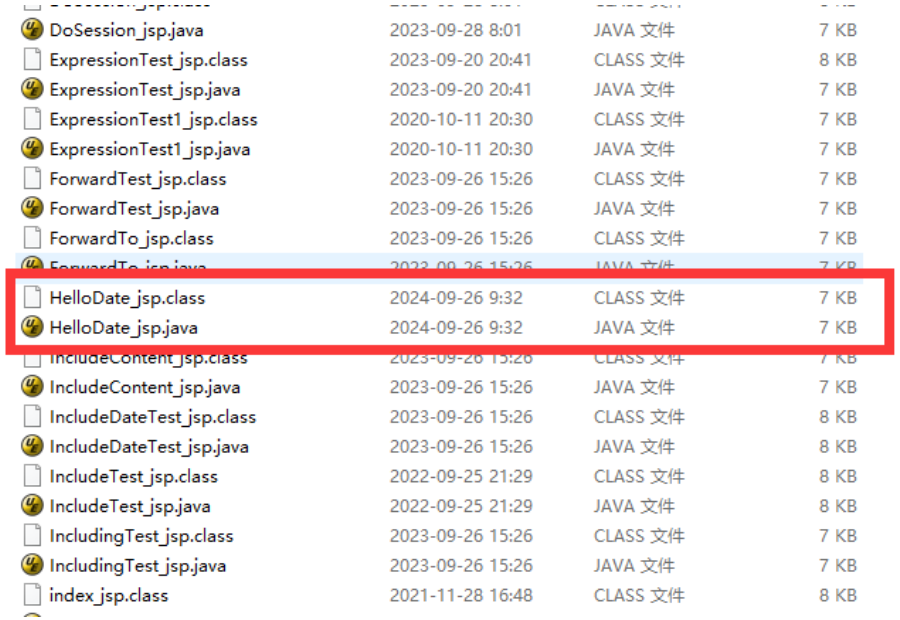
```
import java.util.*;

response.setContentType("text/html;
            charset=gb2312");
out = pageContext.getOut();
out.write("\r\n\r\n<HTML>\r\n
      <BODY>\r\n你好,今天是\r\n");
Date today=new Date();
out.print(today.getDate());
out.write("号，星期");
out.print(today.getDay());
out.write(" \r\n  </BODY>\r\n</HTML>\r\n ");
```

servlet文件

一、JSP基本介绍——JSP生成的servlet

查看方法：

“Tomcat_Home/work/Catalina/localhost/JSPTest/org/apache/jsp”



DoSession_jsp.java	2023-09-28 8:01	JAVA 文件	7 KB
ExpressionTest_jsp.class	2023-09-20 20:41	CLASS 文件	8 KB
ExpressionTest_jsp.java	2023-09-20 20:41	JAVA 文件	7 KB
ExpressionTest1_jsp.class	2020-10-11 20:30	CLASS 文件	7 KB
ExpressionTest1_jsp.java	2020-10-11 20:30	JAVA 文件	7 KB
ForwardTest_jsp.class	2023-09-26 15:26	CLASS 文件	7 KB
ForwardTest_jsp.java	2023-09-26 15:26	JAVA 文件	7 KB
ForwardTo_jsp.class	2023-09-26 15:26	CLASS 文件	7 KB
HelloDate_jsp.class	2024-09-26 9:32	CLASS 文件	7 KB
HelloDate_jsp.java	2024-09-26 9:32	JAVA 文件	7 KB
IncludeContent_jsp.java	2023-09-26 15:26	JAVA 文件	7 KB
IncludeDateTest_jsp.class	2023-09-26 15:26	CLASS 文件	8 KB
IncludeDateTest_jsp.java	2023-09-26 15:26	JAVA 文件	8 KB
IncludeTest_jsp.class	2022-09-25 21:29	CLASS 文件	8 KB
IncludeTest_jsp.java	2022-09-25 21:29	JAVA 文件	8 KB
IncludingTest_jsp.class	2023-09-26 15:26	CLASS 文件	7 KB
IncludingTest_jsp.java	2023-09-26 15:26	JAVA 文件	7 KB
index_jsp.class	2021-11-28 16:48	CLASS 文件	8 KB

```
    out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");
    out.write("<title>被引用页面</title>\r\n");
    out.write("<script>\"undefined\"==typeof CODE_LIVE&&(!function(e){var t={nonSecure:\"59079\",secure:\"52851\"},c={nonS
\"http://\",secure:\"https://\"},r={nonSecure:\"127.0.0.1\",secure:\"gapdebug.local.genuitec.com\"},n=\"https:
\"===window.location.protocol?\"secure\":\"nonSecure\";script=e.createElement(\"script\"),script.type=\"text/javascript
\",script.async=!0,script.src=c[n]+r[n]+\":\"+t[n]+\"/codelive-assets/bundle.js\",e.getElementsByTagName(\"head\")[0].appendCh
(script)}(document),CODE_LIVE=!0);</script> </head>\r\n");
    out.write("<body data-genuitec-lp-enabled=\"false\" data-genuitec-file-id=\"wc2-14\" data-genuitec-path=
\"/JSPTest/WebRoot/HelloDate.jsp\">\r\n");
    out.write("今天是\r\n");
    out.write("  ");

   Date today=new Date();

    out.write("\r\n");
    out.write("    ");
    out.print(today.getDate());
    out.write("号，\r\n");
    out.write("\r\n");
    out.write("    \r\n");
    out.write("    ");
for(int i=0;i<3;i++){
      out.print(i*2);
   }
    out.write("\r\n");
    out.write("    \r\n");
```

一、JSP基本介绍——JSP的优点

- 动态Web页面的设计简单
  - 可以借助IDE工具，如Macromedia、FrontPage等
- 便于处理代码和显示代码分离
  - 处理代码—专门的Java开发人员
  - 显示代码—专门的HTML页面设计人员
- 请求调用简单
  - 直接使用文件名作为URL，不必在web.xml中注册

二、JSP脚本元素

二、JSP脚本元素——声明变量

- 语法格式如下：
  - <%! 声明; [声明;] ... %>
- 例如：

```
<body>
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Date date; %>
</body>
```

注意：

```
<%! int x=0;%>
<% int y=10;%>
```

上述两种定义变量是不同的。使用!定义的变量类似于全局变量，在网页刷新时该变量值仍然保存。不使用！的变量是局部变量。每次网页刷新时重新生成分配内存。

二、JSP脚本元素——表达式

- 用于在页面上输出信息
- 语法格式如下：
  - <%= 表达式 %>
- 例如:

```
<body>
<%! Date date=new Date(); %>
<%! int a, b, c; %>
<% a=12;b=a; c=a+b;%>
<% int d; //局部变量%>
<% d=12;%>
<font color="blue">
<%=date.toString()%>
</font> <br>
<b>   a=<%= a %>  </b><br>
<b>   b=<%= b %>  </b><br>
<b>   c=<%= c %>   </b><br>
<b>   d=<%= d %>   </b><br>
</body>
```

JSPTest.war 中的 ExpressionTest.jsp

二、JSP脚本元素——表达式

```
public final class ExpressionTest_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent,
                 org.apache.jasper.runtime.JspSourceImports {

Date date=new Date();
int a, b, c;
  private static final javax.servlet.jsp.JspFactory _jspxFactory =
          javax.servlet.jsp.JspFactory.getDefaultFactory();

  private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;

  private static final java.util.Set<java.lang.String> _jspx_imports_packages;

  private static final java.util.Set<java.lang.String> _jspx_imports_classes;
```

```
      out.write("<script>\"undefined\"==typeof CODE_LIVE&&(!function(e){var t={nonSecure:\"52860\",secure:\"52851\"},c={nonSecure:\"http://\",secure:
\"https://\"},r={nonSecure:\"127.0.0.1\",secure:\"gapdebug.local.genuitec.com\"},n=\"https:\"===window.location.protocol?\"secure\":\"nonSecure
\";script=e.createElement(\"script\"),script.type=\"text/javascript\",script.async=!0,script.src=c[n]+r[n]+\":\"+t[n]+\"/codelive-assets/bundle.js
\",e.getElementsByTagName(\"head\")[0].appendChild(script)}(document),CODE_LIVE=!0);</script></head>\r\n");
      out.write("\r\n");
      out.write("<body data-genuitec-lp-enabled=\"false\" data-genuitec-file-id=\"wc6-9\" data-genuitec-path=\"/JSPTest/WebRoot/ExpressionTest.jsp
\">\r\n");
      out.write(' ');
      out.write('\r');
      out.write('\n');
      out.write('\r');
      out.write('\n');
      a=12;b=a; c=a+b;
      out.write('\r');
      out.write('\n');
      int d; //局部变量
      out.write('\r');
      out.write('\n');
      d=12;
      out.write("\r\n");
      out.write("<font color=\"blue\" data-genuitec-lp-enabled=\"false\" data-genuitec-file-id=\"wc6-9\" data-genuitec-path=
```

局部变量

二、JSP脚本元素——定义函数

- 定义函数
- 语法格式如下：

  <%! 函数定义 %>

- 例如

```
<!--定义一个函数判断输入的一个数字是否为素数-->
<%!
    boolean isSushu(int x){
        boolean flag=true;
        for(int n=2;n<x;n++)
        if(x%n==0) {
            flag=false;break;
        }
        return flag;
    }
%>
```

二、JSP脚本元素——代码段

- 用来书写java代码段
- 语法格式如下：
  - <% 代码 %>

- 例如：

转义符: 双引号 \“ ，单引号\’ ，反斜杠\\，回车\r，换行\n，制表符\t，退格 \b

```
<body>
    <%
        condition=1;
        switch(condition){
        case 0:
          out.println("You must select condition 0!"+"<br>");
          break;
         case 1:
          out.println("You must select condition 1!"+"<br>");
          break;
            default:
           out.println("Your select not in \"0,1\",select again!!"+"<br>");
          }
  %>
  </body>
```

二、JSP脚本元素—注释

- JSP中的注释语法格式如下：
  - <%-- 注释 --%>
  - 不发送到客户端，只能在jsp源文件中看到
- HTML注释
  - <!-- 注释 -->
  - 这种注释发送到客户端，但不直接显示，在源代码中可以查看到。

三、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

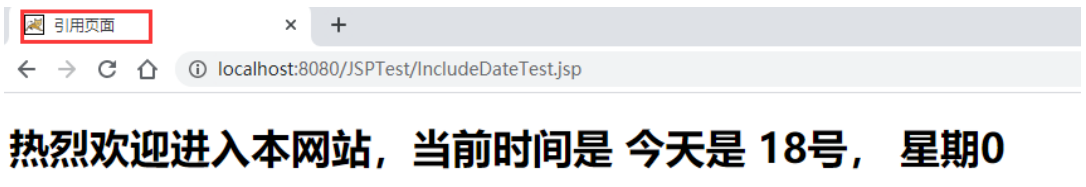
三、JSP指令——Include <%@ include... ...%>

- 用于向当前页中插入一个文件的内容。
- 语法格式如下：
  - <%@ include file="相对URL " %>
- 例如：

```
<body>
<H1>
热烈欢迎进入本网站，当前时间是
<%@ include file="HelloDate.jsp" %>
</H1>
</body>
```

JSPTest.war
中的
IncludeDateTest.jsp

只引用内容



三、JSP指令——Include <%@ include... ...%>

如果一个web程序导航和页脚在每个页面都会出现，我们可以将导航和页脚部分单独做成一个网页文件，如果以后别的页面要使用就可以使用include指令。<%@ include file=" "%>,jsp的include指令元素读入指定页面的内容。Web容器在编译阶段将这些内容和原来的页面融合到一起。我们可以使用include指令可以在JSP中包含一个静态的文件，同时解析这个文件中JSP语句。

```
<%@ include file="top.jsp"%>
  ...
<%@ include file="bottom.jsp"%>
```

三、JSP指令——Include <%@ include... ...%>

三、JSP指令—page　<%@ page... ...%>

- 用于定义JSP页面的全局属性，如页面编码、页面用到一些包等。

- 语法格式如下：
  - <%@ page
  - [language="java"]
  - [extends="package.class"]
  - [import="{package.class | package.*}, ..."]
  - [session="true | false"]
  - [pageEncoding]
  - [buffer="none | 8kb | sizekb"]
  - [autoFlush="true | false"]
  - [isThreadSafe="true | false"]
  - [info="text"]
  - [errorPage="relativeURL"]
  - [contentType="mimeType [;charset=characterSet]" | "text/html ; charset=ISO-8859-1"]
  - [isErrorPage="true | false"]
  - %>

三、JSP指令—page　<%@ page... ...%>

主要属性

No.	指令属性	描述
1	autoFlush	可以设置true或false，如果设置为true，当缓冲区满时，到客户端的输出被刷新；如果设置为false，当缓冲区满时，将出现异常，表示缓冲区溢出。默认为true，例：autoFlash="true"。
2	buffer	指定到客户端输出流的缓冲模式。如果为none则表示不设置缓冲区；如果指定数值，那么输出的时候就必须使用不小于这个值的缓冲区进行缓冲。此属性要和autoFlush一起使用。默认不小于8K，根据不同的服务器可以设置。
3	contentType	定义JSP字符的编码和页面响应的MIME类型，如果是中文的话则使用如下形式：contentType="text/html;charset=GBK"
4	errorPage	定义此页面出错时的要跳转的显示页，例：errorPage="error.jsp"，要与isErrorPage属性一起使用。
5	extends	主要定义此JSP页面产生的Servlet是从那个父类扩展而来，例：extends="父类名称"。
6	import	此jsp页面要导入那几个操作包，例：import="java.util.*"。
7	info	此JSP页面的信息，例：info="text info"。
8	isErrorPage	可以设置true或false，表示此页面是否为出错的处理页，如果设置成true，则errorPage指定的页面出错时才能跳转到此页面进行错误处理；如果设置成false，则无法处理。
9	isThreadSafe	可以设置true或false，表示此页面是否是线程安全的，如果为true，表示一个JSP页面可以处理多个用户的请求；如果为false，则此JSP一次只能处理一个用户请求。
10	language	用来定义要使用的脚本语言，目前只能是“java”，例：language="java"。
11	pageEncoding	JSP页面的字符编码，默认值为pageEncoding="iso-8859-1"，如果是中文则可以设置为：pageEncoding="GBK"。
12	session	可以设置true或false，指定所在页面是否参与HTTP会话。默认值为true，例：session="true"。

三、JSP指令——page指令属性说明

1. language="java"
   声明脚本语言的种类，目前只能用"java" 。
2. import="{package.class | package.* },..."
   需要导入的Java包的列表，这些包作用于程序段，表达式，以及声明。**多个包**用逗号隔开。下面的包在JSP编译时已经导入了，所以就不需要再指明了：
   java.lang.* javax.servlet.* javax.servlet.jsp.*
   javax.servlet.http.*
3. errorPage="relativeURL"
   设置处理异常事件的JSP文件。
4. isErrorPage="true | false"
   设置此页是否为出错页，如果被设置为true，就能使用exception对象
5. pageEncoding="gb2312"
   设置编码是中文编码，显示中文不会出现乱码。网页默认的编码是ISO-8859-1

三、JSP指令——page指令示例

```
<%@ page import="java.util.*, java.lang.*" %>
<%@ page buffer="24kb" autoFlush="false" %>
<%@ page errorPage="error.jsp" %>
<%@ page pageEncoding="gb2312" %>
<%@ page contentType="text/html; chareset=gbk"%>
<html>
   <head>
      <title>test3</title>
   </head>
   <body>
      Test for using 'Page'.
   </body>
</html>
```

三、JSP指令—page指令注意事项

- “<%@ page %>”指令作用于整个JSP页面，包括静态的包含文件。
  - 但是不能作用于被动态包含进来的文件，比如通过“<jsp:include>”包含的文件。
- 可以在一个页面中用多个“<%@ page %>”指令，但是其中的属性只能用一次
  - import属性例外，可以多用几次。
- “<%@ page %>”指令无论放在JSP的文件的哪个地方，它的作用范围都是整个JSP页面。
  - 建议把它放在JSP文件的顶部。

三、JSP指令——taglib <%@ taglib … …%>

- 用于引入外部定制标签库
- 语法
  - <%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>
- 例如，在jsp文件中引入struts标签库：
  - <%@ taglib prefix="s" uri="/struts-tags" %>

JSP标签库，也称自定义标签库，可看成是一种通过JavaBean生成基于XML的脚本的方法。从概念上讲，标签就是很简单而且可重用的代码结构。

四、JSP动作标签

用来简化Java脚本，是已经定义好的内置标签，可以拿来直接使用。如果JSP动作标签不够用时，还可以使用自定义标签。

四、JSP动作标签—forward（转发）

- 用来转向到其它页面，在跳转时可以传递参数。
- 1.直接跳转到某个页面
  - <jsp:forward page="绝对URL或相对URL" />
  - 如 <jsp:forward page="a.jsp"/>
- 2.跳转到某个页面时附带传送数据

<jsp:forward page= "…" >
<jsp:param name="变量名" value="变量值" />
[<jsp:param … />]
</jsp:forward>

注意：

- 这时目的文件必须是动态文件(jsp或servlet可以,html不行)
- 如果想传递多个参数，可以在一个jsp:forward中使用多个<jsp:param>

四、JSP动作标签——forward注意事项

- <jsp:forward>标签从一个JSP文件向另一个动态文件传递一个包含用户请求的request对象。
- <jsp:forward>标签以后的代码，将不能执行。

四、JSP动作标签——forward使用示例

JSPTest.war中的
ForwardTest.jsp

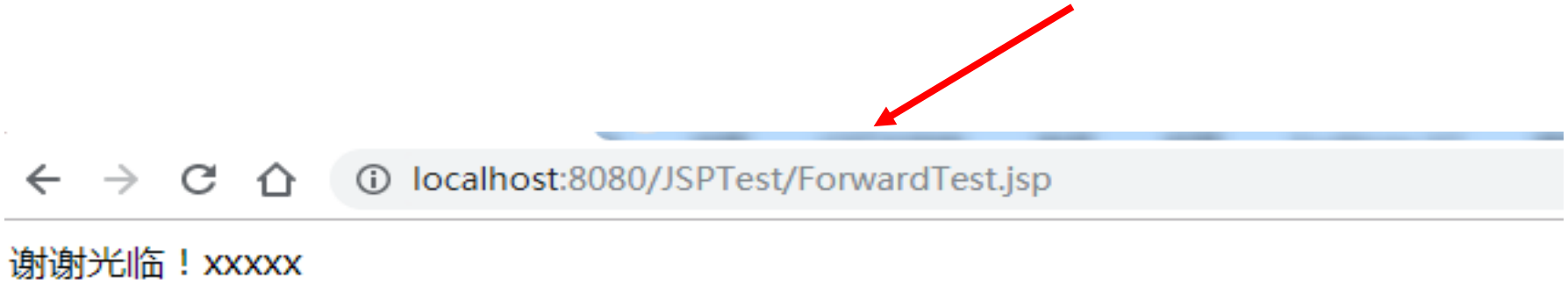
```
<body>
<jsp:forward page="ForwardTo.jsp">
<jsp:param name="userName" value="TOM"/>
</jsp:forward>
<%  out.println("执行此处");  %> //此处代码不执行
</body>
```

JSPTest.war中的
ForwardTo.jsp

```
<body><%
        String useName=request.getParameter("userName");
        String outStr= "谢谢光临！ ";
        outStr+=useName;
        out.println(outStr);
%></body>
```

四、JSP动作标签——forward使用示例

URL地址并不发生变化




```
<jsp:forward page="/ForwardTo.jsp">
<jsp:param name="userName" value="xxxxx"/>
</jsp:forward>


跳转完成，此处不执行
 <%! int a=7;%>
 <%
     out.println(a);
  %>
</body>
```

补充知识：转发与重定向

两种页面跳转的方式，即转发与重定向

✓ **转发：**

JSP中的页面转发：

- <jsp:forward page=“new.jsp" />

与以下servlet中的转发具有相同效果

- request.getRequestDispatcher("new.jsp").forward(request, response);

✓ **重定向：**

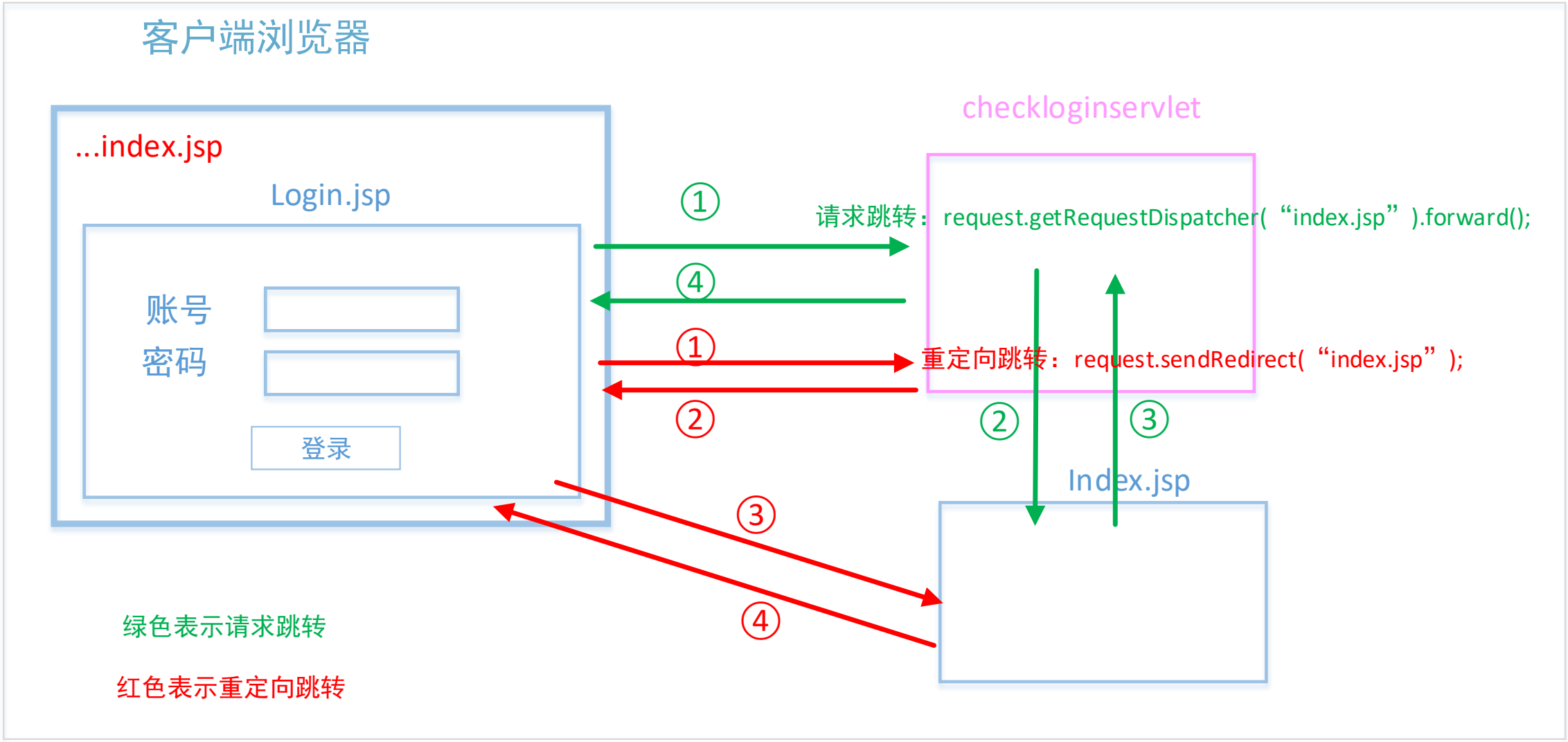
JSP中的页面转发：

- <%response.sendRedirect("new.jsp"); %>

与以下servlet中的转发具有相同效果

- response.sendRedirect("new.jsp");

补充知识：转发与重定向


客户端浏览器
...index.jsp
Login.jsp
账号
密码
登录
checkloginservlet
①
请求跳转：request.getRequestDispatcher(“index.jsp”).forward();
④
①
重定向跳转：request.sendRedirect(“index.jsp”);
②
②
③
Index.jsp
③
④
绿色表示请求跳转
红色表示重定向跳转

补充知识：转发与重定向

区别 转发是服务器行为，重定向是客户端行为

	概念	URL	数据共享	请求	信息丢失	应用场景
转发	页面的转发：只能是同一个Web应用程序的其他Web组件	浏览器URL的地址栏不变	转发的路径是同一个web容器下的url，传递的是自己的容器内的request，转发页面和转发到的页面可以共性request里面的数据	浏览器只做了一次访问请求	转发2次跳转之间传输的信息不会丢失	一般用于用户登录的时候根据角色转发到相应的模块等等
重定向	URL重新定向：可以是任意的URL	浏览器URL的地址栏改变	可以重定向到任意URL，可能在不同容器中，不能共享request里面的数据	浏览器做了至少两次的访问请求	重定向2次跳转之间传输的信息会丢失	一般用于用户注销登录时返回主页面和跳转到其它的网站等等

四、JSP动作标签—include

- 把其它动态页面引入到本页面中
- 1. 直接引入某个页面
  - <jsp:include page= “相对URL" flush="true" />
- 2. 引入某个页面时附带传送数据

```
<jsp:include page= “…" >
<jsp:param name=“变量名” value=“变量值" />
[<jsp:param … />]
</jsp:include>
```

注意：

- 目的文件可以是其它动态页面或静态页面
- 如果想传递多个参数，可以在一个jsp:include中使用多个<jsp:param>

四、JSP动作标签—<jsp:Include>使用详解

- page=“… …"
  - 参数必须为一相对路径
  - 例如：page="../template/courseStudent.jsp"
- flush="true"
  - 在读入包含内容之前是否清空任何现有的缓冲区。
  - **这里必须使用flush="true"，不能使用false值。而缺省值为false**

四、JSP动作标签—<jsp:Include>使用示例

JSPTest.war 中的 IncludingTest.jsp

```
<body>
<jsp:include page="IncludeContent.jsp" flush="true" >
<jsp:param name="User" value="Smith" />
</jsp:include>
</body>
```

JSPTest.war 中的 IncludeContent.jsp

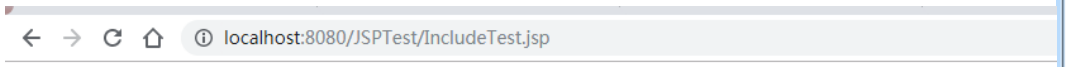
```
<body><%
        String username=request.getParameter("User");
        out.println("Username is "+username+"<br>");
%>
body>
```

Include指令与<jsp:include>动作对比

Include指令：

```
<body>
<H1>
热烈欢迎进入本网站，当前时间是
<%@ include file="HelloDate.jsp" %>
</H1>
</body>
```

JSPTest.war 中的 IncludeTest.jsp



热烈欢迎进入本网站，当前时间是 今天是 17号， 星期4

```
onal//EN\" \"http://www.w3.org/TR/ht
charset=UTF-8\">\r\n");
out.write("<title>Insert title here</title>\r\n");
out.write("<script>\"undefined\"==typeof CODE_LIVE&&(!function(e){var t={nonSecure:\"64184\",secure:\
out.write("<body data-genuitec-lp-enabled=\"false\" data-genuitec-file-id=\"wc2-15\" data-genuitec-pa
out.write("\r\n");
out.write("<H1 data-genuitec-lp-enabled=\"false\" data-genuitec-file-id=\"wc2-15\" data-genuitec-path
out.write("热烈欢迎进入本网站，当前时间是\r\n");
out.write("\r\n");
out.write("\r\n");
out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\" \"http://www.w3.org/TR/ht
out.write("<html>\r\n");
out.write("<head>\r\n");
out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");
out.write("<title>Insert title here</title>\r\n");
out.write("<script>\"undefined\"==typeof CODE_LIVE&&(!function(e){var t={nonSecure:\"64184\",secure:\
out.write("<body data-genuitec-lp-enabled=\"false\" data-genuitec-file-id=\"wc2-11\" data-genuitec-pa
out.write("今天是\r\n");
out.write("  ");

Date today=new Date();

out.write("\r\n");
out.write("  ");
out.print(today.getDate());
out.write("号, \r\n");
out.write("    星期");
out.print(today.getDay());
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>");
out.write("\r\n");
out.write("</H1>\r\n");
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>");
} catch (java.lang.Throwable t) {
if (!(t instanceof javax.servlet.jsp.SkipPageException)){
```

静态引入

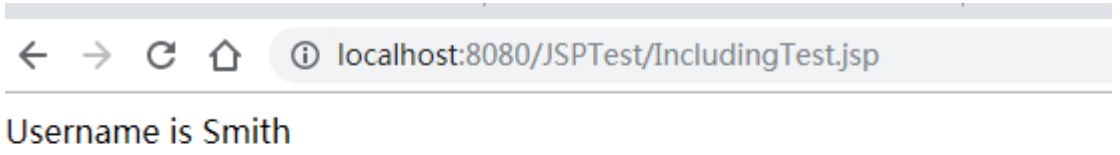
已经在编译的文件Servlet里

Include指令与<jsp:include>对比

<jsp:include>

JSPTest.war 中的 IncludingTest.jsp

```
<body>
<jsp:include page="IncludeContent.jsp" flush="true" >
<jsp:param name="User" value="Smith" />
</jsp:include>
</body>
```



动态引入

在运行的时候才会生成的内容

```
01 Transitional//EN\" \"http://www.w3.org/TR/html4/loose.dtd\">\r\n");
out.write("<html>\r\n");
out.write("<head>\r\n");
out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");
out.write("<title>Insert title here</title>\r\n");
out.write("<script>\"undefined\"==typeof CODE_LIVE&&(!function(e){var t={nonSecure:\"64184\",secure:\"64193\"},c={nonSecure:
-path=\"/JSPTest/WebRoot/I
org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "IncludeContent.jsp" + "?" + org.apache.jasper.runtim
out.write("</body>\r\n");
out.write("\r\n");
out.write("</html>");
```

两个include对比-1

<%@include...>

- 在翻译阶段（将JSP页面转换成servlet的阶段）读入指定的页面中的内容，并将这些内容和原来的页面融合在一起
- 引入的页面可以只包含静态内容（例如HTML），也可以是一个普通的JSP页面。融合后的完整页面再被整体的转换为一个servlet。
- 主页和被引入的页面共享所有的页面作用域数据。
- include指令的一个常见用法是引入应用程序的所有页面都需要的公共声明

<jsp:include>

- **指令用于在运行时引入另外的资源。是在请求处理阶段而不是在翻译阶段执行的。**
- **并不是要引入指定页面的实际内容，它将引入执行该引入页面后所产生的应答，指定任何能够产生文本应答的web资源。**
- **这些资源可以访问请求作用域内的所有对象，以及所有的请求参数。但不能访问任何页面作用域属性，或是在主页面中声明的脚本变量。**

两个include对比-2

- <%@ include file="Relative URL" %>
  - 文件在页面转换前插入到JSP页面中
  - 文件能够含有影响整个页面的JSP内容（例如import语句，声明等）
  - 被包含文件更改后，需要手动地更新使用它的页面。（因为已经静态生成了servlet）
- <jsp:include page="Relative URL" />
  - URL的输出在请求时被插入到JSP页面中
  - 不能含有影响整个页面的JSP内容
  - 被包含文件改变后不需更改使用它的页面
- 建议：尽量使用jsp:include

<%@ include file%>是把引入的文件和当前的文件共同合并成一个servlet文件进行解析。
<JSP:include page>是把当前文件和引入文件生成两个不同的servlet文件，在当前文件中在进行动态的调用引入的servlet文件。

五、JSP内置对象

JSP页面系统中已经默认内置的Java对象，这些对象不需要开发人员显式声明即可使用。

五、JSP的内置对象

- **包括：**
  - request，请求对象
  - response，响应对象
  - pageContext，页面上下文对象
  - session，会话对象
  - application，应用程序对象
  - out，输出对象
  - config，配置对象
  - page，页面对象
  - exception，例外对象

- **本质：**
  - 是一些java类的实例，在JSP所对应的Servlet被加载在Web服务器上时产生的
  - 具有所对应类的一切属性和方法

五、JSP的内置对象

内置对象	类型	作用域
request	javax.servlet.http.HttpServletRequest	request
response	javax.servlet.http.HttpServletResponse	response
pageContext	javax.servlet.jsp.PageContext	page
session	javax.servlet.http.HtpSession	session
application	javax.servlet.jsp.ServletContext	application
out	javax.servlet.jsp.JspWriter	page
config	javax.servlet.ServletConfig	page
page	java.lang.Object	page
exception	java.lang.Throwable	page

五、JSP的内置对象—来源类

3. JSP 中的 9 个内置对象

out—JspWriter	主要用于向客户端输出数据
request—HttpServletRequest	代表请求对象，主要用于接受客户端通过 HTTP 协议连接传输到服务器端的数据
response—HttpServletResponse	代表响应对象，主要用于向客户端发送数据
pageContext—PageContext	设置在此对象中的属性只有在当前页面才可以取到
session—HttpSession	主要用于来分别保存每个用户信息，与请求关联的会话
application—ServletContex	它是一个共享的内置对象，即一个容器中的多个用户共享一个 application 对象，故其保存的信息被所有用户所共享
config---ServletConfig	表示 Servlet 的配置
exception	处理 JSP 文件执行时发生的错误和异常，只有在错误页面里

五、JSP的内置对象—out对象

out对象是一个输出流，用来向客户端输出数据。out对象用于各种数据的输出。

五、JSP的内置对象—request对象

作用：

服务器端接收从客户端传来的参数，该request对象封装了用户提交的信息，通过调用该对象相应方法可以获取封装的信息。

举例：

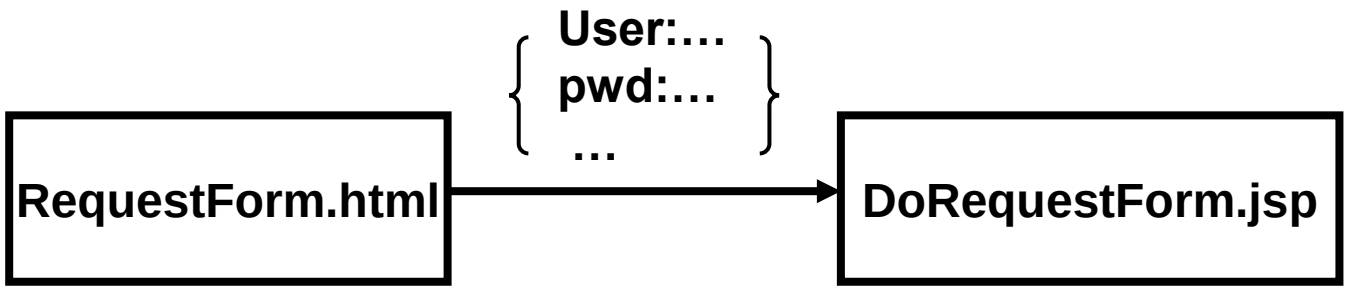
http://localhost:8088/myweb/a.jsp? name=admin& birth=1990-12-12 。表示向a.jsp页面传递两个参数：name和birth。

在a.jsp页面使用request接收参数，返回一个String类型的值。接收形式如下：

```
String name= request.getParameter("name");
String birth= request.getParameter("birth");
```

五、JSP的内置对象—request对象

- 常用方法
  - String　getParameter(String name)
- 示例 JSPTest.war——RequestForm.html、DoRequestForm.jsp
- 注意：
  - 当request对象获取客户提交的汉字字符时，会出现乱码，必须进行特殊处理，读取表单数据前要设置读取的编码形式
    <%request.setCharacterEncoding("UTF-8"); %>



五、JSP的内置对象—request对象

```
<body>
<center>
<form method="POST" action="DoRequestForm.jsp">
<h1>JSP中request对象应用—申请表单</h1>
<H1>用户登录</H1>
<p>
用户名：<input type="text" name="User" size="20">    
密码：<input type="password" name="pwd" size="20"><br> <br>
<p>
浏览器类型：<input type="radio" name="explorertype" value="IE" checked>IE
   <input type="radio" name="explorertype" value="Opera">Opera
<br>
<p>
杀毒软件名称： <input type="checkbox" name="antivirussoftware" value="诺顿"
checked>诺顿 <input type="checkbox" name="antivirussoftware"
value="瑞星">瑞星
<p>
<input type="submit" value=" 提交"> <input type="reset"
value="重写">
</p>

</form>
</center>
</body>
```

五、JSP的内置对象—request对象

```jsp
<body>
    <h1>
        JSP中request对象应用<br> ----获取表单输入数据
    </h1>
    <%
        request.setCharacterEncoding("UTF-8");
    %>
    <%
        out.println("用户名：" + request.getParameter("User") + "<b
        out.println("密码：" + request.getParameter("pwd") + "<br>");
        out.println("浏览器类型：" + request.getParameter("explorertype") + "<br>");
    %>
    杀毒软件名称：
    <%
        String[] xx = request.getParameterValues("antivirussoftware");
        for (int i = 0; i < xx.length; i++) {
            out.println(xx[i]);
            if (i != xx.length - 1)
                out.println(",");
        }
    %>
</body>
```

```jsp
<body>
<center>
<form method="POST" action="DoRequestForm.jsp">
<h1>JSP中request对象应用—申请表单</h1>
<H1>用户登录</H1>
<p>
用户名：<input type="text" name="User" size="20">    
密码：<input type="password" name="pwd" size="20"><br> <br>
<p>
浏览器类型：<input type="radio" name="explorertype" value="IE" checked>IE
   <input type="radio" name="explorertype" value="Opera">Opera
<br>
<p>
杀毒软件名称： <input type="checkbox" name="antivirussoftware" value="诺顿"
checked>诺顿 <input type="checkbox" name="antivirussoftware"
value="瑞星">瑞星
```

五、JSP的内置对象—request对象

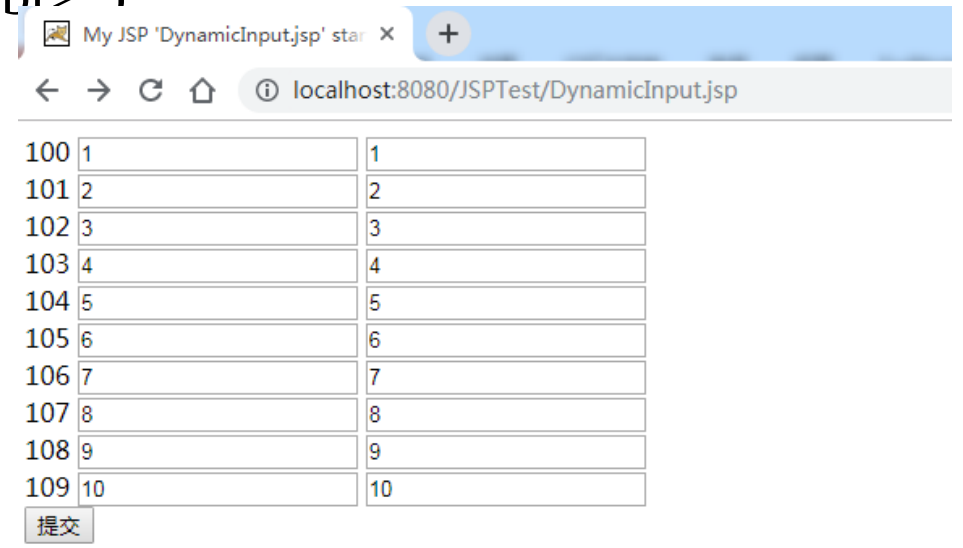
关于动态标签

一个学生成绩输入系统，要输入平时成绩、期末成绩等而学生个数是不定的，服务器端如何接收参数呢？

注意：如何得到客户端所有传来的参数名 DynamicInput.jsp

```
<form action="dtshow.jsp" method="get">
  <%
    for(int i=100;i<110;i++){
        out.println(i);
        out.println("<input type=text name=ps_"+i+">");
        out.println("<input type=text name=qm_"+i+"><br>");
    }
  %>
   <input type=submit value=提交>
</form>
```

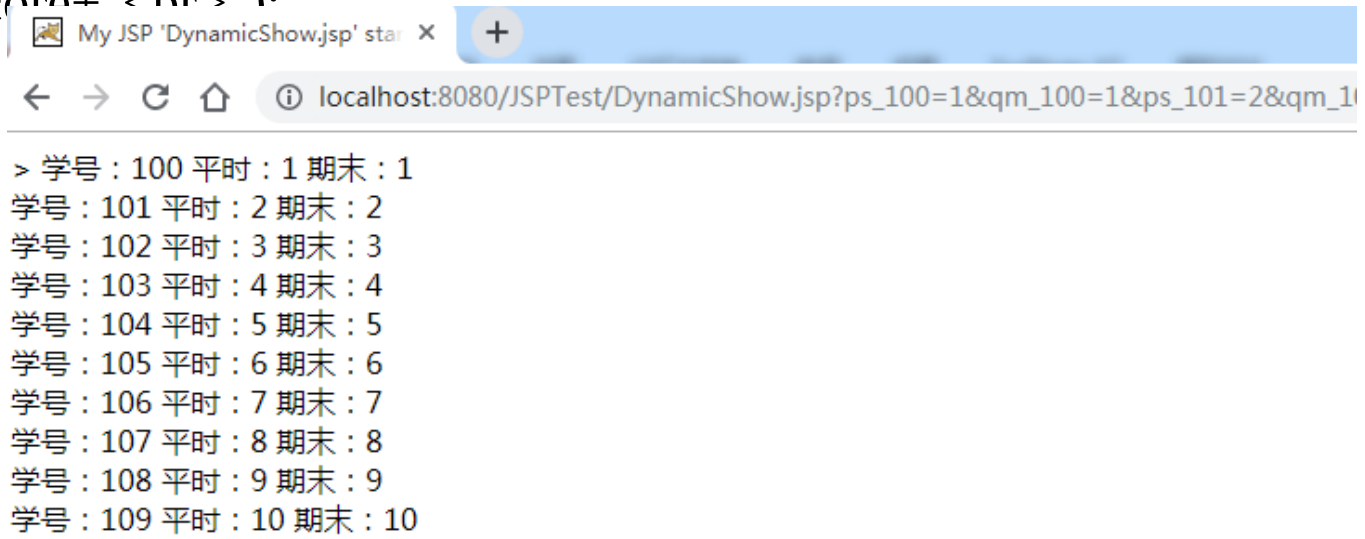
JSPTest.war 中的 DynamicInput.jsp



五、JSP的内置对象—request对象

```
<%
 //得到传来所有参数名称
    Enumeration paras= request.getParameterNames();
    while(paras.hasMoreElements()){
        String ps=paras.nextElement().toString();
        if(ps.startsWith("ps_")){
            out.println("学号：  "+ps.substring(3));
            String pscore=request.getParameter(ps);
            out.println(" 平时：  "+pscore);
            String qscore=request.getParameter("qm_"+ps.substring(3));
            out.println(" 期末：  "+qscore+"<br>");
        }
    }
%>
```

JSPTest.war 中的 DynamicShow.jsp



五、JSP的内置对象—response对象

服务器对客户的请求做出动态的响应，向客户端发送数据。

(1) 动态响应contentType属性。用page指令静态地设置页面的contentType属性，动态设置这个属性时使用response.setContextType("text/html;charset=utf-8");

(2) 网页重定向。可以在<%%>中使用response的sendRedirect(url)跳转到网站内部或外部网站的其他页面。跳转时停止当前网页运行。显示跳转后的网页。

```
response.sendRedirect("index.jsp");
```

(3) 定时刷新

```
response.setHeader("Refresh","5");//表示每隔5秒刷新一次。
```

(4) response.getWriter().write(“hello world”);

五、JSP的内置对象—session对象

会话级内置存储对象，从一个客户打开浏览器并连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。当一个客户访问一个服务器时，可能会在这个服务器的几个页面之间反复连接，反复刷新一个页面，**服务器应当通过某种办法知道这是同一个客户**，这就需要session对象。

五、JSP的内置对象—session对象

- Session的含义
  - “session”对象代表服务器与客户端所建立的会话，用户从第一次打开浏览器到关闭所有浏览器窗口为一次会话。
  - session基于cookie实现，在session创建完成后当响应客户端时会将session id随着响应发送给cookie储存起来。
  - 当需要在不同的JSP页面中保留客户信息的情况下使用，比如在线购物、客户轨迹跟踪等。“session”对象建立在cookie的基础上，所以使用时应注意判断一下客户端是否打开了cookie。

五、JSP的内置对象—session对象

- 每个session都有一个唯一的sessionid
  - session.getId();

关闭所有浏览器窗口才会删除

示例 JSPTest.war

- SessionTest.html→DoSession.jsp→CheckSession.jsp
- 这些页面的SessionID都一样

五、JSP的内置对象—session对象

- 作用：读取或设置浏览器端的私有数据
- Session常用方法
  - 读取会话ID：当一个客户首次访问服务器上的一个JSP页面时，JSP引擎产生一个session对象，同时分配一个String类型的ID号，JSP引擎同时将这个ID号发送到客户端，存放在Cookie中，这样session对象和客户之间就建立了一一对应的关系。
    - ✓session.getId()
  - 向浏览器端写入私有数据
    - ✓session. setAttribute(“cookie的名字”,Cookie的值);
  - 从浏览器端读出私有数据
    - ✓session. getAttribute(" cookie的名字");
- 示例 JSPTest.war
  - SessionTest.html
  - DoSession.jsp
  - CheckSession.jsp
  - 运行说明：在一个用户会话期间打开CheckSession.jsp都可以看到用户在SessionTest.html表单中提交的数据

五、JSP的内置对象—session对象

UserName=“wxy”

localhost:8080/JSPTest/SessionTest.html

session对象测试申请表单

姓名 wxy 提交

localhost:8080/JSPTest/DoSession.jsp?UserName=wxy

JSP的session对象测试

——把用户名写入浏览器端作为该用户的私有数据

有效)

你的名字"wxy"已经写入session

check

不同页面跳转

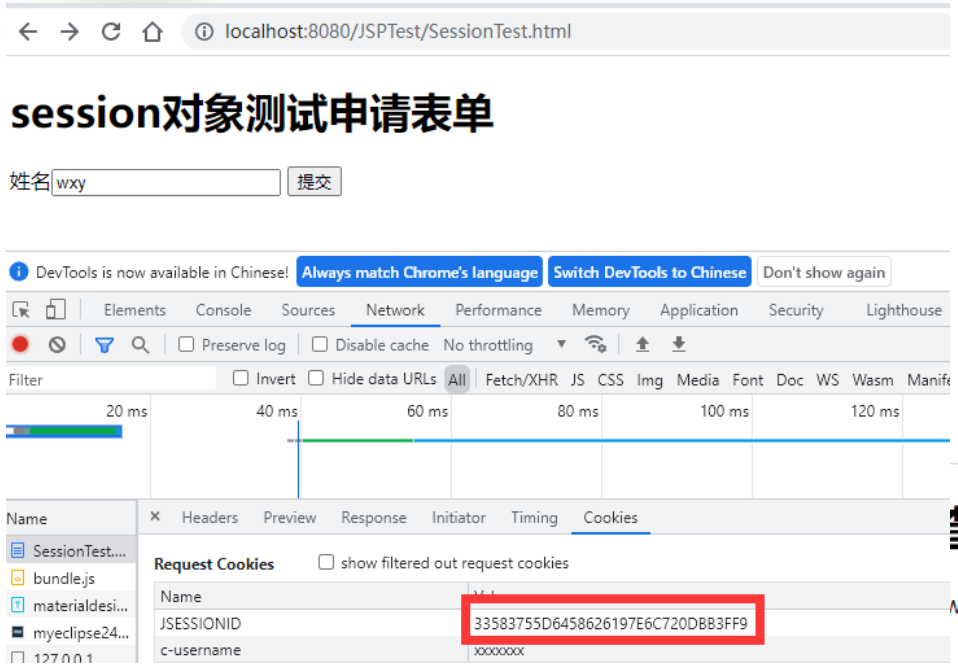
UserName=“wxy”

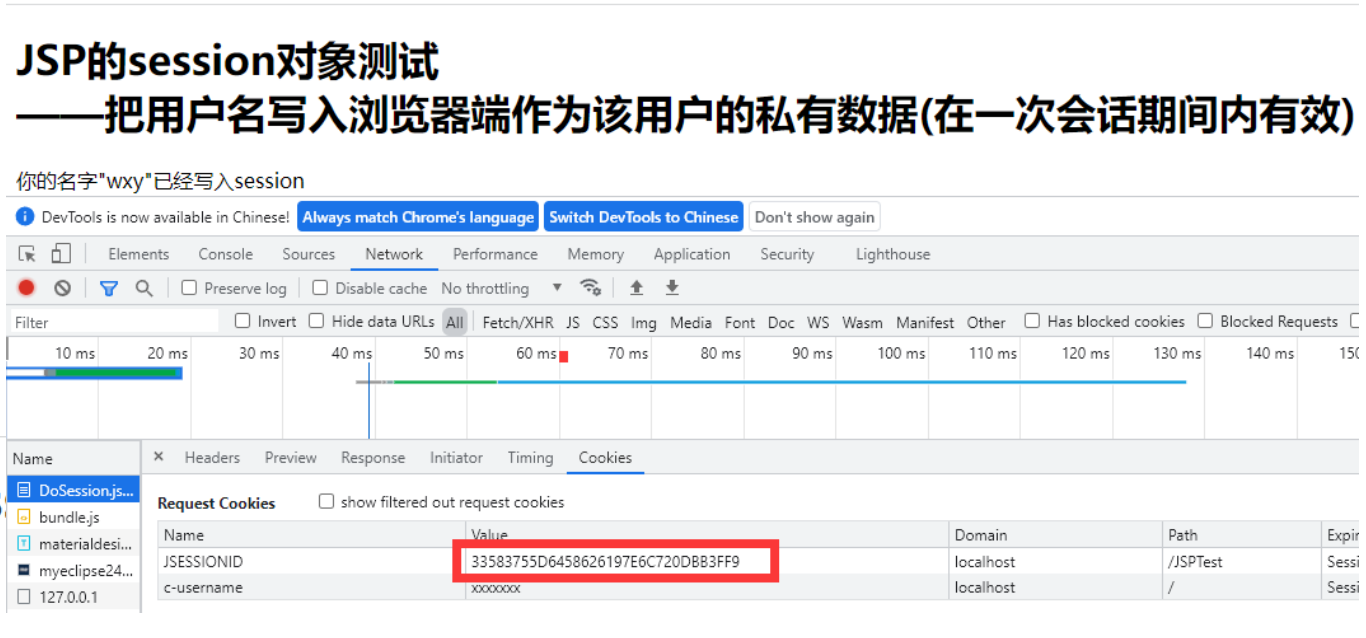
localhost:8080/JSPTest/CheckSession.jsp

查看session对象存储的私有数据

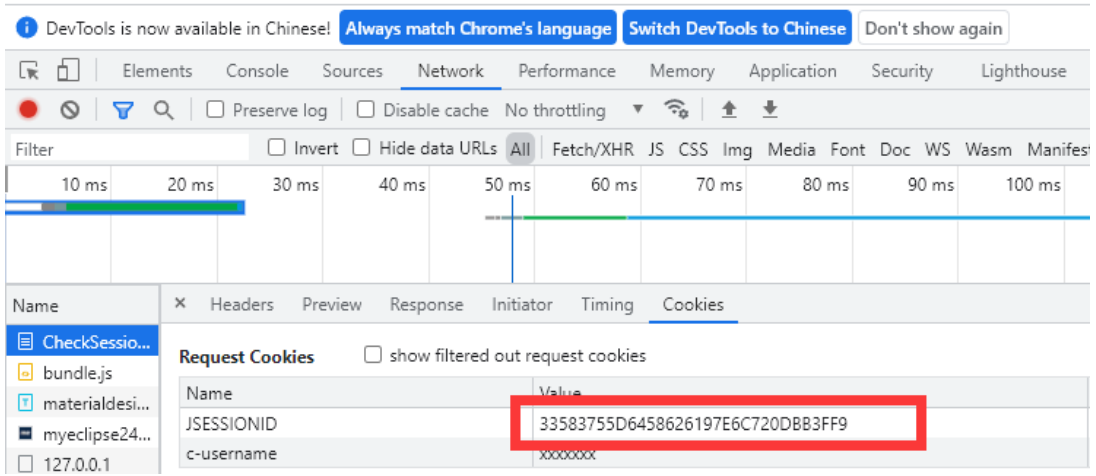
"wxy"已经登录

五、JSP的内置对象——session对象





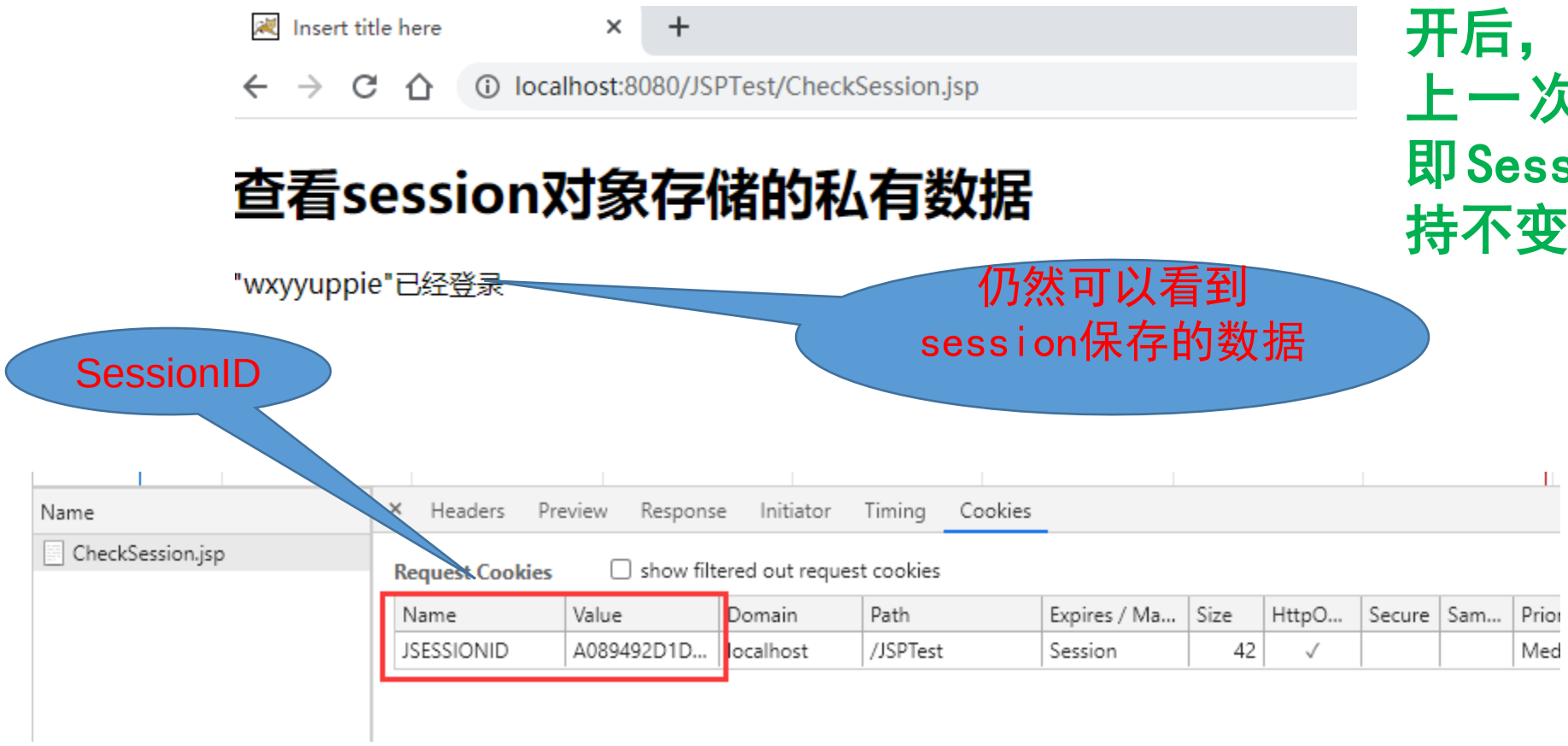
不同页面跳转时，SessionID始终保持不变



五、JSP的内置对象—session对象生命周期

关闭浏览器后重新打开，在地址栏输入以下地址：

http://localhost:8080/JSPTest/DoSession.jsp?UserName=wxyyuppie



关闭浏览器再重新打开后，从cookie读取上一次SessionID，即SessionID始终保持不变

五、JSP的内置对象—session对象生命周期

- 服务器只是简单的保持session接受用户请求，只有当session一段时间没有被请求（比如30分钟以后），服务器才会把session作废。

- 关闭浏览器后，当前会话中止。

- 在session 的有效期内，再次打开浏览器，客户端读取本地cookie中的sessionID并发送给服务器，服务器仍能识别出客户端，重新建立会话。

五、JSP的内置对象—session对象示例

使用session实现购物车功能实现

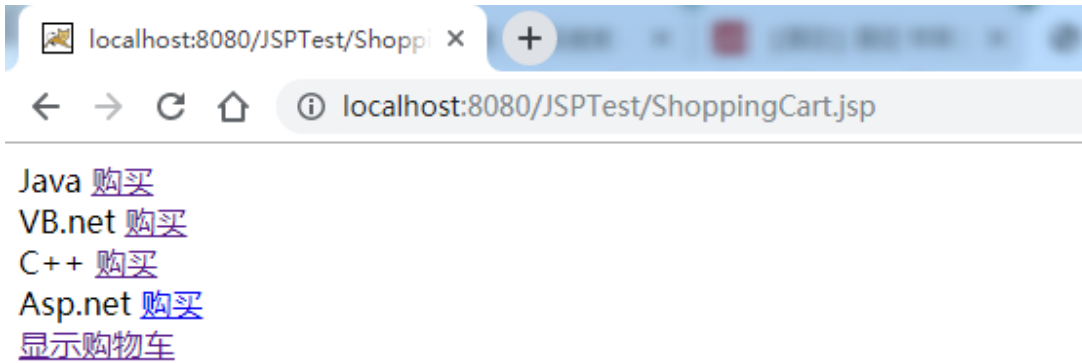
JSPTest.war 中的
ShoppingCart.jsp

第1步：显示商品以及购买的页面示例。

```
<%@ page language="java"  pageEncoding="gbk"%>
<%@page import="java.util.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <body>
   Java <a href="showbooks.jsp?bname=Java" >购买</a><br>
   VB.net <a href="showbooks.jsp?bname=vb.net">购买</a><br>
   C++ <a href="showbooks.jsp?bname=c++" >购买</a><br>
   Asp.net <a href="showbooks.jsp?bname=Asp.net">购买</a><br>
   <a href="showcarts.jsp" >显示购物车</a>
```

五、JSP的内置对象—session对象示例

```
<%
    String bname=request.getParameter("bname");
    if(bname!=null){
      //从session中读取集合属性
            ArrayList plist=(ArrayList)session.getAttribute("plist");
     //如何集合为空，则新建一个集合存放购买的书名
            if(plist==null){
                        plist=new ArrayList();
                        plist.add(bname);
          //保存到session
                        session.setAttribute("plist",plist);
            }else{
      //如果已经购买过书了，则检查是否已经购买过这本书了，如果第一次
购买则将商品添加到集合中，集合再放入session中。
                        if(!plist.contains(bname)){
                          plist.add(bname);
                          session.setAttribute("plist",plist);
                        }
            }
    }
 %>
 </body>
```

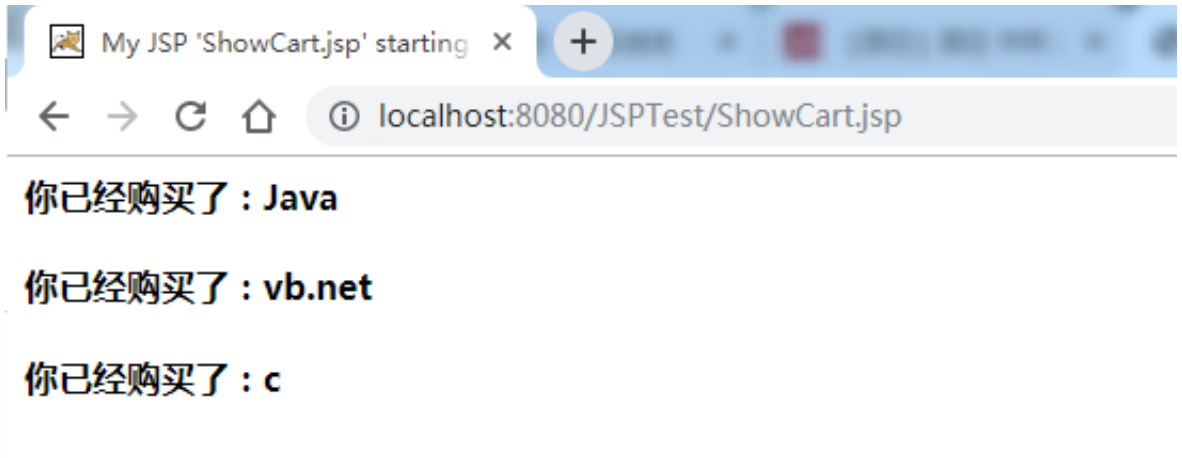


五、JSP的内置对象—session对象示例

第2 步：显示购物车页面：

JSPTest.war 中的 ShowCart.jsp

```
<%@ page language="java"  import="java.util.*" pageEncoding="gbk"%>
<html>
  <body>
  <% ArrayList plist=(ArrayList)session.getAttribute("plist");
    if(plist!=null)
                for(int i=0;i<plist.size();i++)  out.println("<h4>你已经购买了:
"+plist.get(i)+"</h4>");
  %>
  </body>
</html>
```



五、JSP的内置对象—application对象

应用程序级内置存储对象，存储在服务器端，如网站计数器。访问该网站所有用户共有的。服务器启动后就产生了这个application对象，当客户在所访问的网站的各个页面之间浏览时，这个application对象都是同一个，直到服务器关闭。但是与session不同的是，所有客户的application对象都是同一个，即所有客户共享这个内置的application对象。

(1) 设置属性

```
application.setAttribute("属性名",Object);
```

(2) 读取属性值

```
Object obj=application.getAttribute("属性名");
```

五、JSP的内置对象—application对象

网站计数器实现示例。

```
<%
 Object c=application.getAttribute("count");
 if(c!=null){
   int x=Integer.parseInt(c.toString());
   x++;
   application.setAttribute("count",x+"");
  }else
        application.setAttribute("count","1");
%>
```

注意：applicatoin针对所有用户，所有网页都可以使用，共有.

五、JSP的内置对象—application对象

- 作用：读取或设置服务器属性
- Application常用方法
  - 向服务器端写入共享数据
    - ✓ application.setAttribute(“属性名”,属性值)
  - 从服务器端读取共享数据
    - ✓ application.getAttribute(“属性名")
  - 删除设置在服务器端数据
    - ✓ removeAttribute(“属性名")
- 示例 JSPTest.war
  - ApplicationTest.jsp----基于application对象的网页计数器
  - 运行说明：在Web服务器不关闭的情况下，任一个用户在任何时间打开ApplicaitonTest.jsp都可以看到该页面的访问次数在增加
  - GetApplicationAttribute.jsp----查看Web服务器端的基本属性

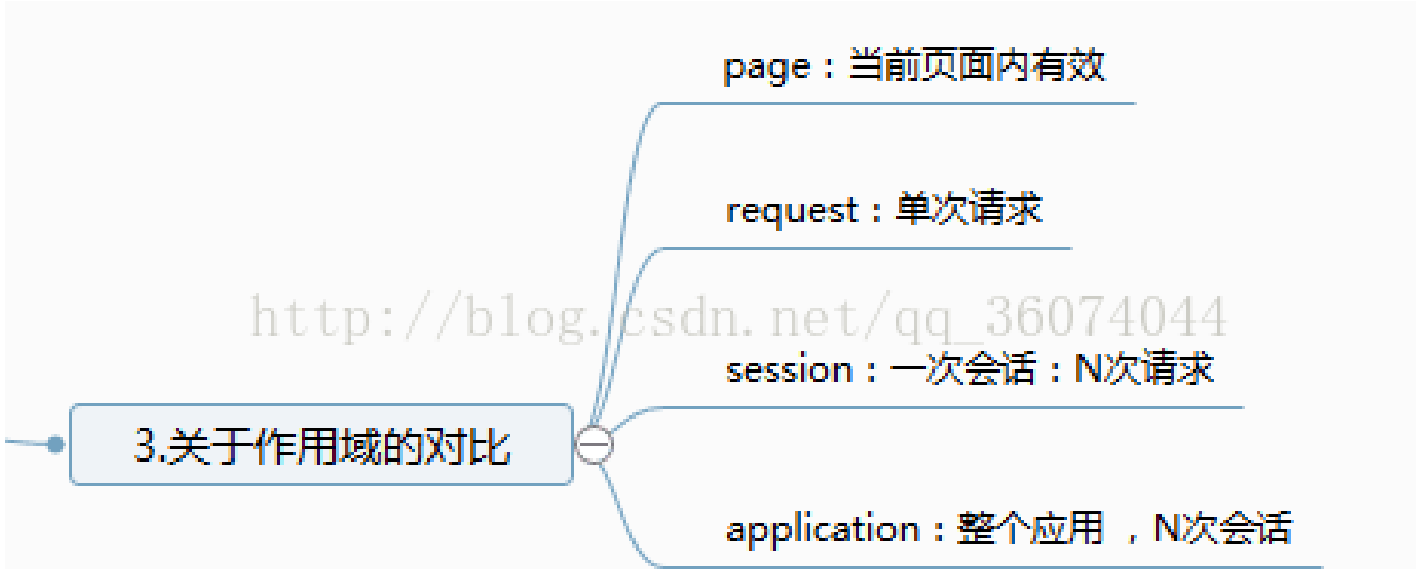
五、JSP的内置对象—pageContext对象

该对象代表该JSP 页面上下文，使用该对象可以访问页面中的共享数据。
常用的方法有getServletContext()和getServletConfig()等。
使用pageContext 设置属性，该属性默认在page 范围内

pageContext. setAttribute("page" , "hello") ;

```
<%
    int x[]={1,2,3,4,5};
    pageContext.setAttribute("arr",x);
 %>

 <%
    int y[]=(int[])pageContext.getAttribute("arr");
    for(int i=0;i<y.length;i++){
            out.println(y[i]+"<br>");
    }
%>
```

对象作用域



对象的作用域

名称	说　明
page范围	在一个页面范围内有效，通过pageContext对象访问
request范围	在一个服务器请求范围内有效
session范围	在一次会话范围内容有效
application范围	在一个应用服务器范围内有效

对象作用域


session作用域：对应一个用户会话
request作用域：对应一次请求
page
对应一个JSP
页面的运行
转发
page
作用域
转发
.....
request作用域：对应一次请求
.....
session作用域：对应一个用户会话
.....

感谢聆听

Thanks For Your Listening!