



Java Server Page

JSP程序设计(中)

isszym sysu.edu.cn

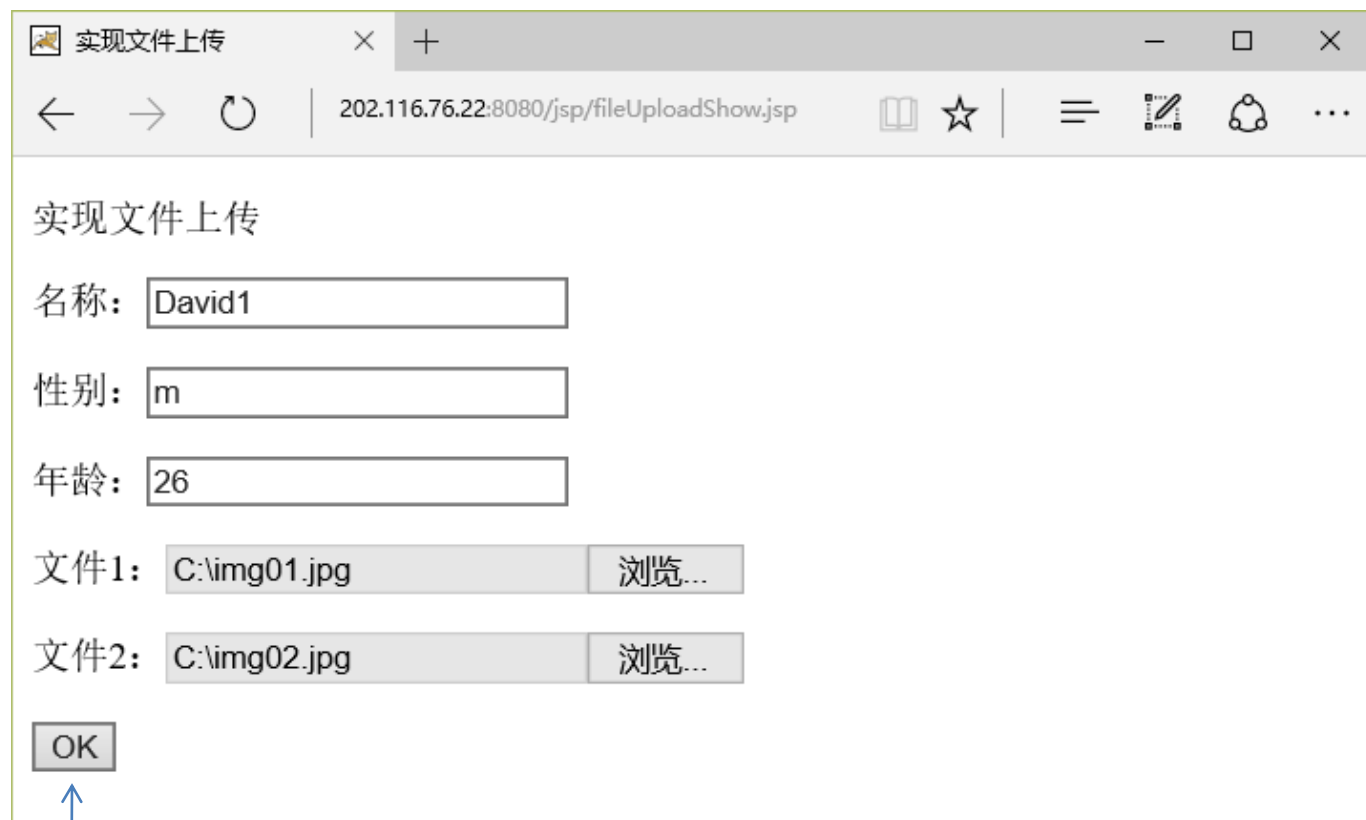
2017.11.22

目录

- 如何上传文件
- 如何在网页之间传递参数
- 使用Cookie
- 使用session
- include指令
- 动作标签<jsp:include>
- 动作标签<jsp:forward>
- 注释语句
- JSP 的基本内置对象
- JavaBean
- 附录1、利用底层功能上传文件
- 附录2、Cookie对象的方法
- 附录3、request对象的其它方法
- 附录4、JSP页面构成元素
- 附录5、JSP指令标签(Directive)
- 附录6、JavaBean的scope
- 附录7、配置虚拟主机和虚拟路径

如何上传文件

[fileUploadShow.jsp](#)



The screenshot shows a web browser window with the title "实现文件上传" (Implement File Upload). The address bar displays the URL "202.116.76.22:8080/jsp/fileUploadShow.jsp". The page content includes a form with the following fields and controls:

- 名称:
- 性别:
- 年龄:
- 文件1:
- 文件2:
-

A blue arrow points to the "OK" button.

点击

[fileUpload.jsp](#)



如果上传的文件使用汉字作为文件名，需要conf/server.xml的Connector中加入
URIEncoding="UTF-8":

```
<Connector port="8080" protocol="HTTP/1.1"  
    connectionTimeout="20000"  
    redirectPort="8443"  
    URIEncoding="UTF-8" />
```

```
<%@ page language="java" import="java.util.*" contentType="text/html;
charset=utf-8"%>
<!DOCTYPE HTML>
<html>
<head>
    <title>文件上传</title>
</head>
<body>
<p>文件上传</p>
<form name="fileupload" action="fileupload.jsp" method="POST"
        enctype="multipart/form-data">
    <p>名称: <input type="text" name="name" size=24 value="David"></p>
    <p>性别: <input type="text" name="sex" SIZE=24 value="male"></p>
    <p>年龄: <input type="text" name="age" SIZE=24 value="28"></p>
    <p>文件1: <input type="file" name="file1" size=24></p>
    <p>文件2: <input type="file" name="file2" size=24></p>
    <p><input type="submit" name="submit" value="OK"></p>
</form>
</body>
</html>
```

元素form的属性enctype的三个主要取值:

- (1) application/x-www-form-urlencoded: 用于上传表单数据, 这是默认方式
- (2) multipart/form-data: 用于上传文件, 还可以同时上传表单数据。
- (3) text/plain: 上传纯文本。

fileUpload.jsp

```
<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%>
<%@ page import="java.io.*, java.util.*,org.apache.commons.io.*"%>
<%@ page import="org.apache.commons.fileupload.*"%>
<%@ page import="org.apache.commons.fileupload.disk.*"%>
<%@ page import="org.apache.commons.fileupload.servlet.*"%>
<html><head><title>文件传输例子</title></head>
<body><%request.setCharacterEncoding("utf-8");%>
    <% boolean isMultipart
        = ServletFileUpload.isMultipartContent(request);//是否用multipart提交的?
    if (isMultipart) {
        FileItemFactory factory = new DiskFileItemFactory();
        //factory.setSizeThreshold(yourMaxMemorySize); //设置使用的内存最大值
        //factory.setRepository(yourTempDirectory); //设置文件临时目录
        ServletFileUpload upload = new ServletFileUpload(factory);
        //upload.setSizeMax(yourMaxRequestSize); //允许的最大文件尺寸
        List items = upload.parseRequest(request);
        for (int i = 0; i < items.size(); i++) {
            FileItem fi = (FileItem) items.get(i);
            if (fi.isFormField()) {//如果是表单字段
                out.print(fi.getFieldName()+":"+fi.getString("utf-8"));
            }
            else {//如果是文件
```

```
DiskFileItem dfi = (DiskFileItem) fi;
if (!dfi.getName().trim().equals("")) { //getName()返回文件名称或空串
    out.print("文件被上传到服务上的实际位置: ");
    String fileName=application.getRealPath("/file")
        + System.getProperty("file.separator")
        + FilenameUtils.getName(dfi.getName());
    out.print(new File(fileName).getAbsolutePath());
    dfi.write(new File(fileName));
} //if
} //if
} //for
} //if
%>
</body>
</html>
```

所用的jar包:

commons-io-1.4.jar

commons-fileupload-1.2.1.jar

如何在网页之间传递参数

由于HTTP协议不保存状态，因此，要有专门的方法在从一个网页中访问另一个网页时传递参数，哪怕是访问的是同一个网页：

方法1、URL参数:

把要传递给另一个网页的参数放在链接的url里。

例如： `做加法`

网页(index.jsp)可以通过`request.getParameter("x")`取到参数x的值。

方法2、利用session

把键值对保存在服务器端。属于一次会话的所有页面都可以共享这些键值对。

方法3、利用cookie

把键值对保存在客户端(浏览器)中，每次访问相同网站时浏览器都会把它放在HTTP请求中发给服务器。

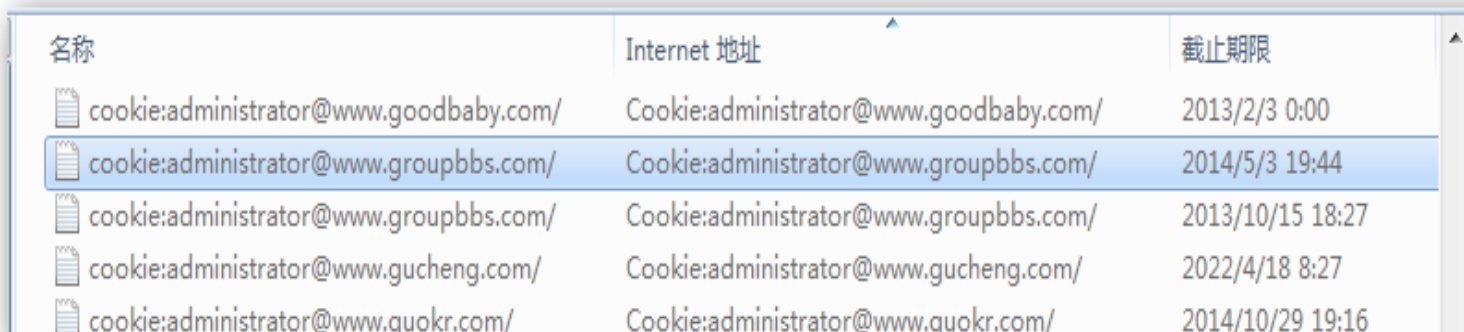
方法4、利用post

当前页面通过提交数据把参数传给另一个页面（或者自己）。较少用这种方法传递参数。

使用Cookie

• 概述

- 。虽然http协议不保留状态，但是利用Cookie可以在客户端保留一些键值对，例如：用户名和密码。当再次访问相同的网站，浏览器会自动把这些数据提交给服务器。
- 。cookie都是采用key-value方式保存信息。每个cookie包含很多项目：名(name)、值(value)、域(domain)、路径(path)、过期时间(expires)。
- 。查看IE 上的cookie： 工具/选项/常规/浏览历史记录/设置/查看文件。



名称	Internet 地址	截止时间
cookie:administrator@www.goodbaby.com/	Cookie:administrator@www.goodbaby.com/	2013/2/3 0:00
cookie:administrator@www.groupbbs.com/	Cookie:administrator@www.groupbbs.com/	2014/5/3 19:44
cookie:administrator@www.groupbbs.com/	Cookie:administrator@www.groupbbs.com/	2013/10/15 18:27
cookie:administrator@www.gucheng.com/	Cookie:administrator@www.gucheng.com/	2022/4/18 8:27
cookie:administrator@www.quokr.com/	Cookie:administrator@www.quokr.com/	2014/10/29 19:16

• cookie的内容 (IE)

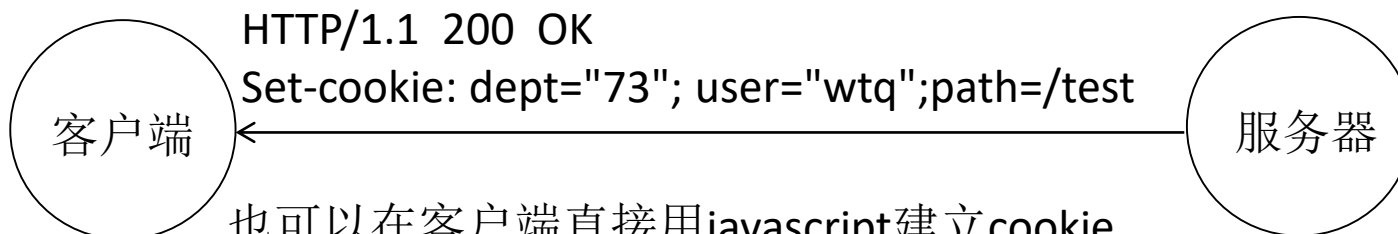
```
1 UserRnd
2 Rnd=843
3 www.groupbbs.com/
4 1536
5 138375808
6 30369477
7 2438931904
8 30296051
9 *
```

第1行: Cookie 名称
第2行: Cookie 的值 (域名+路径)
第3行: Cookie 所属站点的地址
第4行: 一个标记值(是否被加密)
第5行: 超时时间的低位(Cardinal/DWORD)
第6行: 超时时间的高位
第7行: 创建时间的低位
第8行: 创建时间的高位
第9行: 固定为 * , 表示一节的结束

```
Login
Password=123456&UserN
ame=admin
www.groupbbs.com/
1536
1342440960
30310740
883030176
30298469
*
```

• http中的cookie

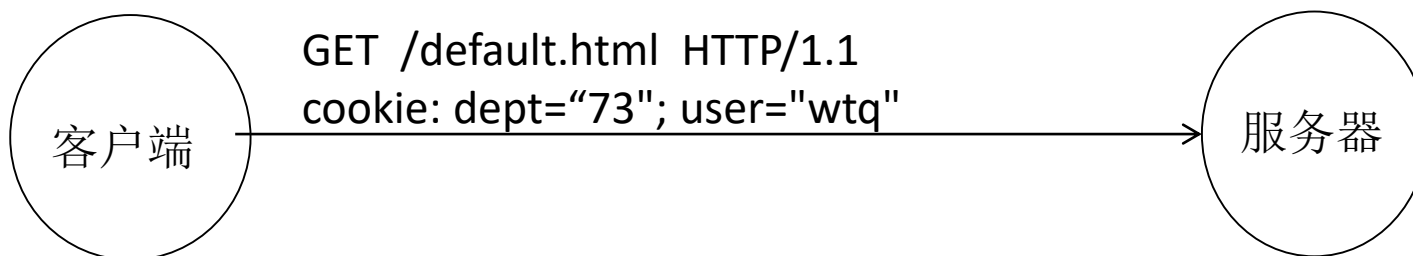
浏览器 服务器端可以利用HTTP响应建立新的cookie



也可以在客户端直接用javascript建立cookie

*未设路径使用当前路径，未设域名采用当前域名

以后每次提交的HTTP请求会带上为该页面设置的有效cookie



- 客户端每次提交网页都会附上在当前页面及其祖先路径上设置的**cookie**。
- 客户端只提交**cookie**的**name=value**，不会上传**cookie**的路径和过期时间等

Set-Cookie: <name>=<value>[; <name>=<value>]...
[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; httponly]

[参考](#)

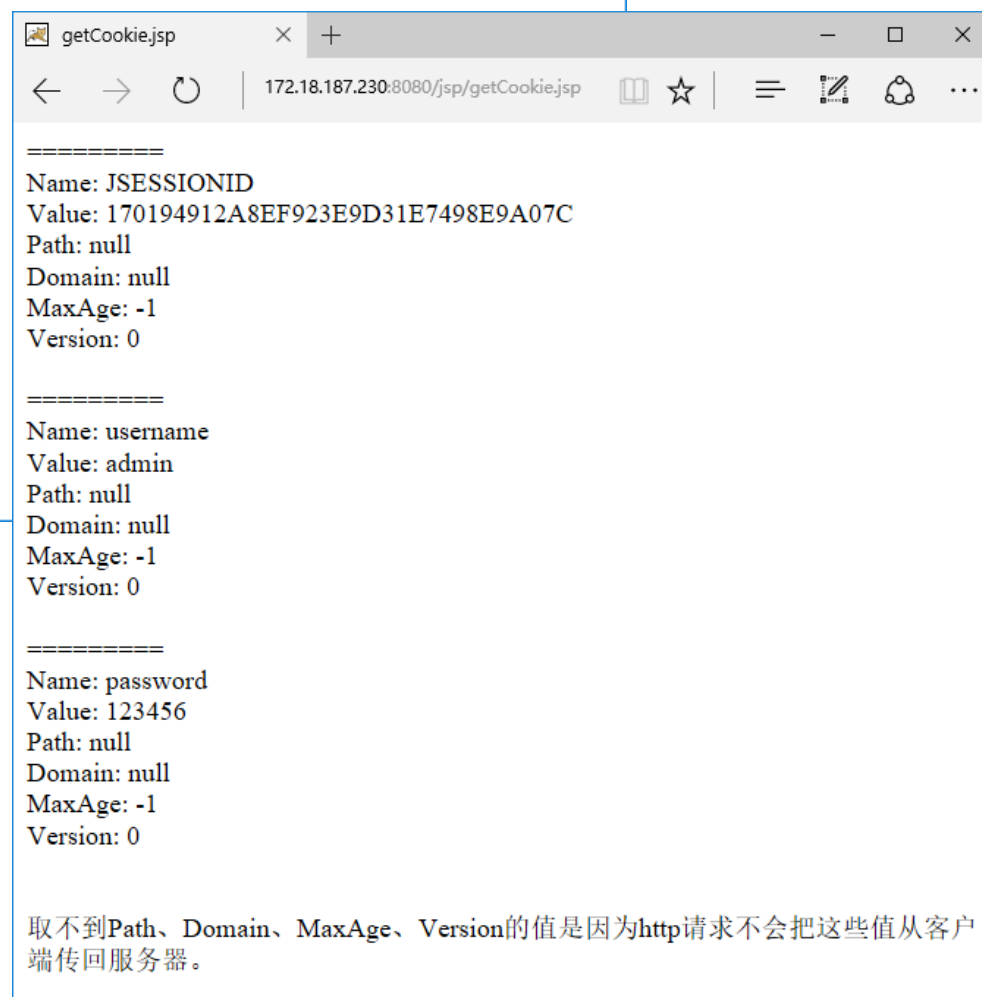
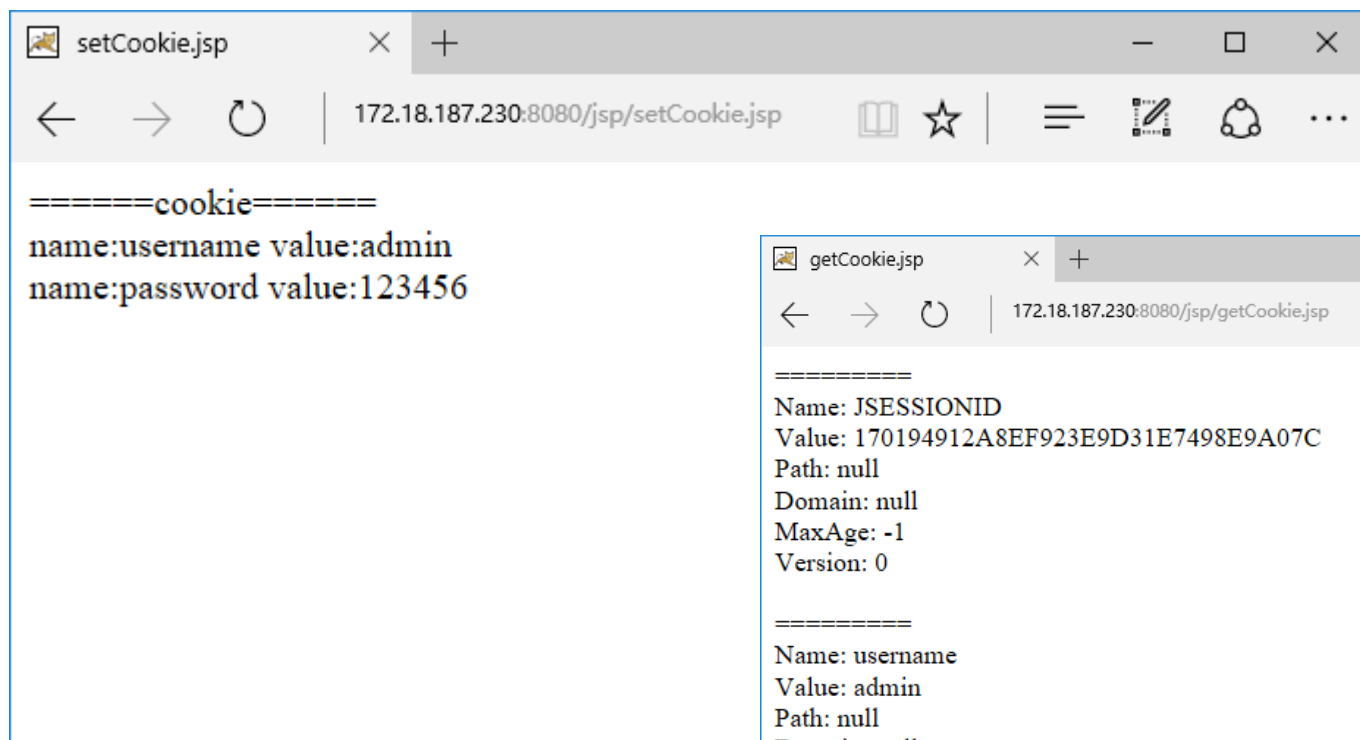
• JSP的Cookie 编程

```
<%  
Cookie cookie = new Cookie("cookieName","cookieValue");  
cookie.setMaxAge(3600); //设置保留时间 3600秒。  
                        //设置为负值表示只保存在内存(与未设置时间相同),  
                        // 关闭浏览器则消失。  
                        // 设置为0表示要删除该cookie。  
cookie.setPath("/"); //设置路径。未设置路径使用当前路径。  
response.addCookie(cookie);  
%>
```

[setCookie.jsp](#)

```
<%  
Cookie[] cookies = request.getCookies();  
for(Cookie cookie : cookies){  
    String name= cookie.getName(); // get the cookie name  
    String value=cookie.getValue(); // get the cookie value  
}  
%>
```

[getCookie.jsp](#)



使用session

• 概述

- http协议是无状态的，但是通过session可以在Web服务器上临时保存少量变量的状态。一般用来保存所有网页都可以使用到的数据，例如，保存已登录的用户名。
- 每次访问设置了session变量的JSP网页，浏览器都会与服务器建立一个会话，服务器会为这次会话生成一个会话id（session id）并用cookie发送给浏览器保存起来。
- 在会话期间在服务器上设置的所有session变量都与该会话的会话id关联，因此，在每次访问该网站时要通过cookie把该会话id发送给服务器。
- 如果该浏览器长时间不访问该网站，也就是长于超时时间，服务器会自动结束本次会话。结束会话的另一种方法是完全关闭该浏览器及其副本。Tomcat的会话超时时间默认为三十分钟。

• http协议中的session



• session对象编程

//在一个网页中给session变量user赋值

```
<% session.setAttribute("user", "LiJN"); %>
```

//在另一个网页中取出session变量user的值

```
<% String user= (String)session.getAttribute("user"); %>
```

//取到所有session变量的名和值

```
<% Enumeration enums=session.getAttributeNames();  
    while(enums.hasMoreElements()){  
        String name=(String)enums.nextElement();  
        out.println(name+": "+session.getAttribute(name)+"<br />");  
    }  
%>
```

// 取出当前浏览器的session Id

```
<% String id= session.getId(); %>
```



```
//删除session变量
session.removeAttribute("user");
// 令所有session无效
session.invalidate();
//获得变量的创建时间(毫秒)
long tm=session.getCreateTime("user");
out.print(new Date(tm))

long ac=session.getLastAccessedTime(); //获得最后的访问时间(毫秒)
session.setMaxInactiveInterval(10);    //设置最大不活动时间(秒)
long mi=session.getMaxInactiveInterval();
boolean n=session.isNew();              //是否是新建的session
```

默认的interval time为30分钟（过期后session边的无效），存放在tomcat/conf/web.xml中：

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

include指令

功能

用于在JSP转换为Servlet程序前将源文件包含进来。include为指令标签。

语法

```
<%@ include file="文件的绝对路径或相对路径" %>
```

说明

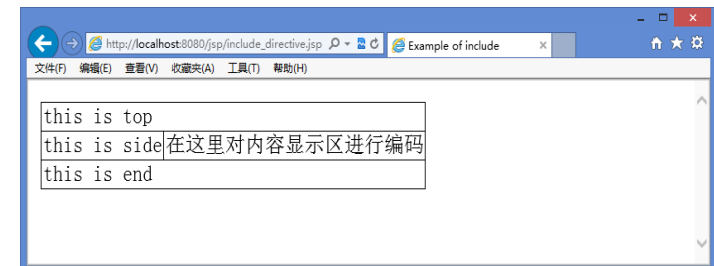
注意：不是url

该属性指定被包含的文件，不支持表达式，也不允许传递参数。被包含的文件的内容原封不动地插入到主文件中，只要被包含的文件发生改变，整个主页面文件就会重新被编译，编译后只有一个.class文件。主文件和子文件的page指令不应重复。

include_directive.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html> <head> <title>Example of include</title>
<style type="text/css">
    table {
        border-collapse: collapse;
    }
    table, td, th {
        border: 1px solid black;
    }
</style> </head>
<body>
<table style="border: 1px solid black">
    <tr><td colspan="2">
        <%@ include file="include_top.jsp"%>
    </td>
</tr>
<tr> <td>
        <%@ include file="include_side.jsp"%></td>
        <td>在这里对内容显示区进行编码</td>
</tr>
<tr> <td colspan="2">
        <%@ include file="include_end.jsp"%>
    </td>
</tr>
</table>
</body> </html>
```

include_top.jsp、include_side.jsp和include_end.jsp都只有一句话，分别是：this is top、this is side和this is end。执行include_directive.jsp的结果：



动作标签<jsp:include>

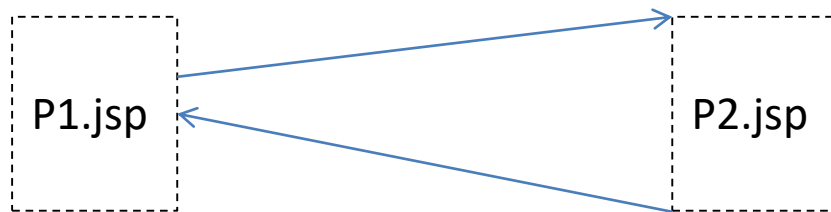
用于向当前的页面中嵌入其它的动态文件并执行它，类似调用一个外部函数。它们共享环境，例如：`request`对象、`response`对象、`session`对象、`out`对象等。

JSP编译器会分别对这两个文件进行编译。被包含的文件中不能含有某些JSP代码，例如，不能设置HTTP头。

```
<jsp:include page="relativeURL" flush="true|false"/>
```

或者

```
<jsp:include page=" relativeURL " flush="true|false">  
    <jsp:param name="参数名称" value="参数值"/>  
</jsp:include>
```



属性`flush`表示当输出缓冲区满时是否清空缓冲区，默认为`false`。

属性`page`指定被包含的页面，该属性支持JSP表达式。

例子:

[include_action.jsp](#)

```
<%@ page language="java" import="java.util.*"
      pageEncoding="ISO-8859-1"%>

<html>
<body>
    Here is three new stories:
    <ol>
        <li><jsp:include page="included1.jsp" />
        <li><jsp:include page="included2.jsp" />
        <li><jsp:include page="included3.jsp" />
    </ol>
</body>
</html>
```

[included1.jsp](#)

```
<html>
<head>
<title>new story 1</title>
</head>
<body>this is story 1!
</body>
</html>
```

Here is three new stories:

1. this is story 1!
2. this is story 2!
3. this is story 3!

* included2.jsp、included3.jsp与included1.jsp类似

动作标签<jsp:forward>

用于后台转移到另外一个JSP文件(相当于goto), 会继承原页面的环境。

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

forward.jsp

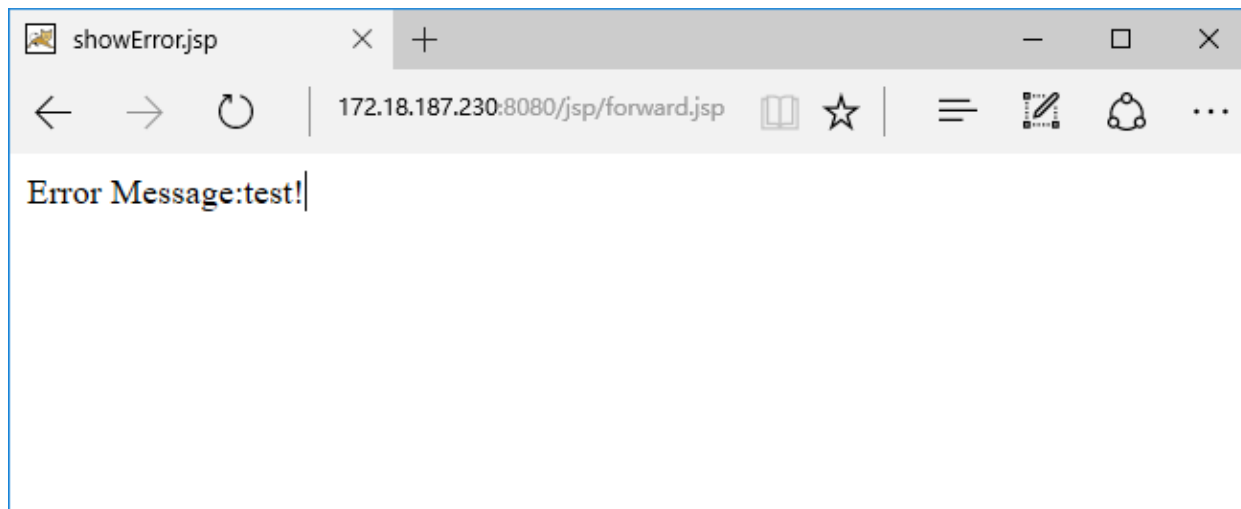
```
<%@ page language="java" import="java.util.*"
      pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>My JSP 'forward.jsp' starting page</title>
  </head>
  <body>
    <%String url = "showError.jsp?error=test!"; %>
    <jsp:forward page="<%=url%>" />
  </body>
</html>
```

* 前台转移方式: `response.sendRedirect(url)`

showError.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>showError.jsp</title>
  </head>

  <body>
    <%= "Error Message:" + request.getParameter("error") %><br>
  </body>
</html>
```



注释语句

功能：对JSP程序进行说明，不会被执行。

语法：

- (1) **HTML注释：** `<!-- 注释内容 -->` 会出现在客户端
- (2) **脚本外注释：** `<%-- 注释内容 --%>` 不会出现在客户端
- (3) **脚本内注释：**
 - `// 行注释内容`
 - `/* 多行注释 */`
 - `/**可以被javadoc读取 */`

comments.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<!-- 欢迎提示信息! --%>
<!-- 展示跨越多个小块进行注释 -->
<table>
    <tr>
        <td>欢迎访问!!!</td>
    </tr>
</table>
<%
    String state = "用户";           // 定义当前状态
    /* if(state.equals("用户")){     // 判断两个字符串是否相等
        state="版主";
    }
    将变量state赋值为"版主"。<br>
    */
%>
</body>
</html>
```

执行结果:



客户端源码:



JSP 的基本内置对象

- 概述

JSP包含可直接使用的9种基本内置对象：

request	用来获取客户端请求的信息
response	网页传回用户端的回应
pageContext	网页的属性是在这里管理
session	与请求有关的会话期
application	Servlet 正在执行的内容
out	用来传送响应的输出
config	Servlet的构架部件
page	JSP网页本身
exception	针对错误网页，未捕捉的例外

- request对象

用于获取HTTP请求的信息。

<%

```
/* 如果原页面的编码为utf-8，在取post参数时要进行如下设置 */  
request.setCharacterEncoding("utf-8");
```

```
/* 取到名为empid的控件值或URL参数值 */  
String empId=request.getParameter("empid");
```

```
/* 取到所有提交的控件名或URL参数名。*/  
Enumeration enums=request.getParameterNames();
```

```
/* 取到同名的多个控件值*/  
String[] courses=request.getParameterValues("courses");
```

```
/*获取HTTP请求的头名字的一个枚举 */  
Enumeration enums=request.getHeaderNames();  
String val=    getHeader(String s)    /* 获取名为s的header的值 */
```

%>

所有提交的内容都会自动执行url编码(urlencode)，getParameter会自动做一次（按照setCharacterEncoding设置的字符编码）进行url解码(urldecode)：

[request_p1.jsp](#)

```
<%@ page contentType="text/html;charset=gb2312" %>
<form method="post" action="request_p1a.jsp">
  学校: <input type="text" name="sname" > <br>
       <input type="submit" name="submit1" value="提交">
```

[request_p1a.jsp](#)

```
<%@ page contentType="text/html;charset=utf-8" %>
<% request.setCharacterEncoding("gb2312");
   String str=request.getParameter("sname");
   out.print(str);
%>
```

在不使用setCharacterEncoding时，也可以采用下面语句进行转换：

```
String str=request.getParameter("sname");
byte[] s1=str.getBytes("iso-8859-1");
str=new String(s1,"gb2312");
```

URL参数必须采用utf-8编码:

[request_p2.jsp](#)

```
<%@ page contentType="text/html;charset=gb2312" %>
<a href="request_p2a.jsp?url=<%=java.net.URLEncoder.encode("中大","utf-8")%>">
    点击这里</a>
```

request_p2a.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
<% String str=request.getParameter("url");
    out.print(str);
%>
```

另一种方法是修改server.xml :

```
<Connector port="8080" maxThreads="150" minSpareThreads="25"
    maxSpareThreads="75" enableLookups="false" redirectPort="8443"
    acceptCount="100" debug="0" connectionTimeout="20000"
    disableUploadTimeout="true"
    URIEncoding="gb2312"
/>
```

• response对象

response对象用来设置返回给用户的信息（http响应）。

```
<% response.setContentType("application/msword;charset=GB2312");  
    // 设置返回的媒体类型和编码。  
%>  
<% response.setContentType("image/jpeg");  
    // 设置返回的媒体类型和编码。  
%>  
<% response.addHeader("refresh",5); //增加头部。不存在就增加，否则更新。  
    // 设置每5秒刷新一次当前网页  
%>  
<% response.sendRedirect("example.jsp");//重定向到example.jsp(前台转移)  
  
%>  
<% response.setStatus(404); //设置状态码  
    // 404为错误码。  
%>
```

- out对象

把一串字符流或字节流输出到HTTP响应页面。

```
out.print(String/boolean/double/float/long var1);  
out.println(String/boolean/double/float/long var2);  
out.newLine();  
out.flush();           //把缓存的内容立即写入字节流  
out.close();  
out.write();           //写字节流
```

在全局函数或类中不能访问out对象，只能通过参数带入。

- application对象

整个应用程序（所有网页）共享的对象。

```
application.getAttribute(String key);  
application.getAttributeNames();  
application.setAttribute(String key, Object obj);  
application.removeAttribute(String key);  
application.getServletInfo();
```

有些JSP Web服务器不支持application对象，需要先用ServletContext()声明这个对象，然后用getServletContext()进行初始化。

JavaBean

通过JavaBean可以调用类，JavaBean要求该类的构造器不能带参数，并只通过方法isXXX()、getXXX()或setXXX()访问其数据域。isXXX()用于返回布尔类型。

```
package com.group.bean;
public class Add{
    private int numA;
    private int numB;
    public int getNumA(){
        return numA;
    }
    public int getNumB(){
        return numB;
    }
    public void setNumA(int val){
        numA = val;
    }
    public void setNumB(int val){
        numB = val;
    }
    public int getSum(){
        return numA+numB;
    }
}
```

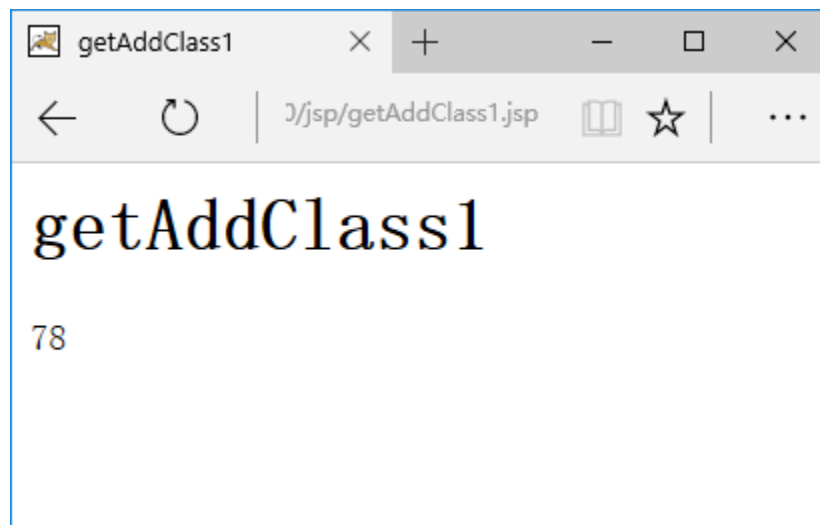
* Add.class放在classes\com\group\bean\下

[getSumClass1.jsp](#)

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>getAddClass1</title>
</head>
<body>
  <h1>getAddClass1</h1>
  <jsp:useBean id="add" scope="page" class="com.group.bean.Add" />
  <jsp:setProperty name="add" property="numA" value="33"/>
  <jsp:setProperty name="add" property="numB" value="45"/>
  <jsp:getProperty name="add" property="sum"/>
</body>
</html>
```

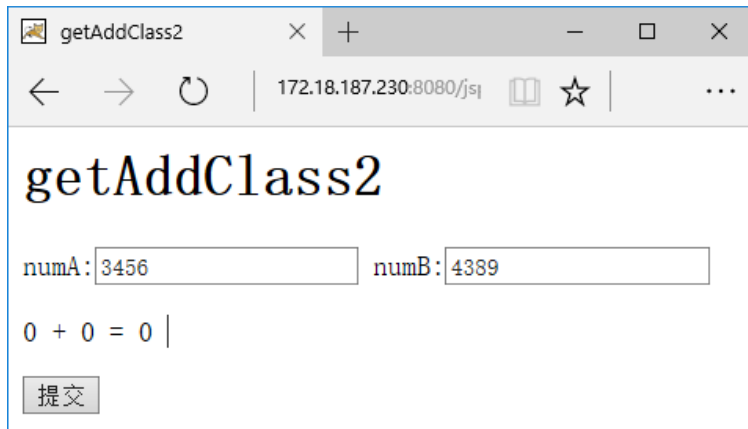
page: 当前页面有效（刷新页面后失效）
request: 当前请求有效，效果与page相同
session: 当前会话有效（sessionID不同时失效）
application: 当前应用有效（重启Web服务器时失效）
* 只有失效后才会产生新对象并重新执行。

* JavaBean必须定义包，不能使用默认包



getSumClass2.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html><head><title>getAddClass2</title></head><body>
  <h1>getAddClass2</h1>
  <jsp:useBean id="add" scope="page" class="com.group.bean.Add" />
  <form method="post" action="getAddClass2.jsp">
    <jsp:setProperty name="add" property="numA"/>
    <jsp:setProperty name="add" property="numB"/>
    <p>numA:<input type="text" name="numA"/>
      numB:<input type="text" name="numB"/></p>
    <p><jsp:getProperty name="add" property="numA"/>
      + <jsp:getProperty name="add" property="numB"/>
      = <jsp:getProperty name="add" property="sum"/></p>
    <input type="submit" name="submit" value="提交"/>
  </form>
</body></html>
```

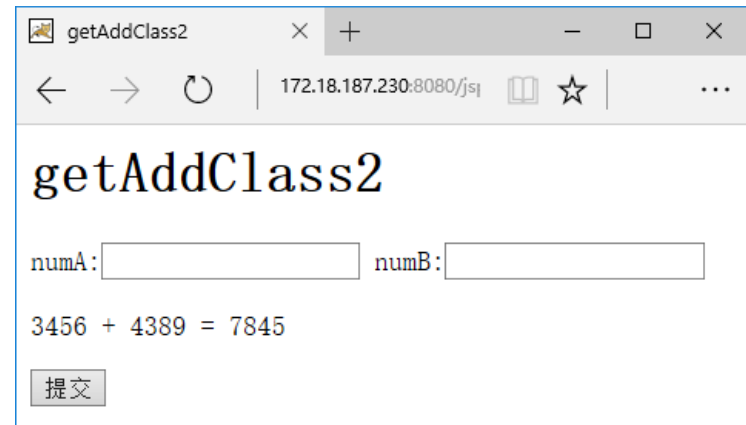


getAddClass2

numA: 3456 numB: 4389

0 + 0 = 0 |

提交



getAddClass2

numA: numB:

3456 + 4389 = 7845

提交

[getSumClass3.jsp](#)

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>getAddClass3</title>
</head>
<body>
  <h1>getAddClass3</h1>
  <jsp:useBean id="add" scope="page" class="com.group.bean.Add" />
  <% add.setNumA(33);add.setNumB(45);%>
  <p><% out.print(add.getNumA()+" + "+add.getNumB()+" = "+add.getSum()); %>
    </p>
</body>
</html>
```



附录

附录1、利用底层功能上传文件

```
<%@ page language="java" import="java.util.*,java.io.*"
    contentType="text/html; charset=utf-8"%>
<%request.setCharacterEncoding("utf-8");%>
<%!// 读取http请求的正文并转化为字符串。
String getRequestBody(HttpServletRequest request)throws Exception{
    int contentLength = request.getContentLength();
    DataInputStream in
        = new DataInputStream(request.getInputStream());
    byte dataBytes[]=new byte[contentLength];
    int bytesRead = 0;
    int totalBytesRead = 0;
    while(totalBytesRead<contentLength){
        bytesRead = in.read(dataBytes,totalBytesRead,contentLength);
        totalBytesRead += bytesRead;
    }
    String requestBody = new String(dataBytes,"ISO-8859-1");
    return requestBody;
}
// 找出用来把正文划分为多个部分的边界字符串
String getBoundary(String contentType){
    int lastIndex = contentType.lastIndexOf("=");
    return contentType.substring(lastIndex+1,contentType.length());
}
```

// 找出正文的一个划分部分的头部中找出文件名

```
String getFileName(String partOfBody) throws Exception {
    final String RET = "\r\n";
    String fileName = "";
    int lineStart = 0;
    int lineEnd = partOfBody.indexOf(RET);
    while(lineEnd>=0){
        String line = partOfBody.substring(lineStart, lineEnd);
        lineStart = lineEnd+RET.length();
        lineEnd = partOfBody.indexOf(RET,lineStart);
        if(line.isEmpty()){
            break;
        }
        else{
            if(line.toLowerCase().indexOf("filename")>=0){
                fileName = line.substring(line.lastIndexOf("=")+1).trim();
                fileName = fileName.substring(1, fileName.length()-1);
                fileName = new String(fileName.getBytes("ISO-8859-1"), "UTF-8");
                break;
            }
        }
    }
    return fileName;
}
```



```

boolean saveFile(String contentPart,String path,String fileName){
    final String RET = "\r\n"; int bodyStart = 0;int lineStart = 0;
    int lineEnd = contentPart.indexOf(RET);
    while(lineEnd>=0){
        String line = contentPart.substring(lineStart, lineEnd);
        lineStart = lineEnd+RET.length();
        lineEnd = contentPart.indexOf(RET,lineStart);
        if(line.isEmpty()){
            bodyStart = lineStart;
            break;
        }
    }
    if(bodyStart==0) return false;
    String body=contentPart.substring(bodyStart);
    try{
        FileOutputStream fileOut
            = new FileOutputStream(path+fileName);
        fileOut.write(body.getBytes("ISO-8859-1"));
        fileOut.flush();
        fileOut.close();
        return true;
    }
    catch(Exception e){
        return false;
    }
}

```

%>

```

<% String msg = "";
    if(request.getMethod().equalsIgnoreCase("post")){
        String requestBody = getRequestBody(request);
        String contentType = request.getContentType();
        String boundary = getBoundary(contentType);
        int startPos = requestBody.indexOf(boundary);
        while(startPos>=0){
            int endPos = requestBody.indexOf(boundary,startPos+1);
            String partOfBody = "";
            if(endPos<0){
                partOfBody = requestBody.substring(startPos);
            }
            else{
                partOfBody= requestBody.substring(startPos
                                                    + boundary.length()+2,endPos);
            }
            String fileName = getFileName(partOfBody);
            if(!fileName.isEmpty()){
                if(saveFile(partOfBody,"c:\\temp\\",fileName)){
                    msg = fileName+"上传成功! ";
                }
                else{
                    msg = fileName+"上传失败! ";
                }
            }
            if(endPos<0) break;
            startPos = partOfBody.indexOf(boundary,endPos);
        }
    }
%>

```

```
<!DOCTYPE HTML>
<html>
<head>
<title>uploadFile.jsp</title>
</head>
<body>
<form action="uploadFile.jsp" method="post" enctype="multipart/form-data">
    选择上传文件: <br /> <input type="file" name="file1" /><br />
    <br /> <input type="submit" name="submit" value="submit" /><br />
    <br />
</form>
<%=msg%>
</body>
</html>
```

附录2、Cookie对象的方法

类型	方法名	方法解释
String	getComment()	返回cookie中注释,如果没有注释的话将返回空值.
String	getDomain()	返回cookie中Cookie适用的域名. 使用getDomain() 方法可以指示浏览器把Cookie返回给同一域内的其他服务器,而通常Cookie只返回给与发送它的服务器名字完全相同的服务器。注意域名必须以点开始（例如.yesky.com）
int	getMaxAge()	返回Cookie过期之前的最大时间，以秒计算。
String	getName()	返回Cookie的名字。名字和值是我们始终关心的两个部分。
String	getPath()	返回Cookie适用的路径。如果不指定路径，Cookie将返回给当前页面所在目录及其子目录下的所有页面。
boolean	getSecure()	如果浏览器通过安全协议发送cookies将返回true值，如果浏览器使用标准协议则返回false值。
String	getValue()	返回Cookie的值。笔者也将在后面详细介绍getValue/setValue。
int	getVersion()	返回Cookie所遵从的协议版本。
void	setComment(String purpose)	设置cookie中注释。
void	setDomain(String pattern)	设置cookie中Cookie适用的域名
void	setMaxAge(int expiry)	以秒计算，设置Cookie过期时间。
void	setPath(String uri)	指定Cookie适用的路径。
void	setSecure(boolean flag)	指出浏览器使用的安全协议，例如HTTPS或SSL。
void	setValue(String newValue)	cookie创建后设置一个新的值。
void	setVersion(int v)	设置Cookie所遵从的协议版本。

附录3、request对象的其它方法

<code>getProtocol()</code>	获取http响应所使用的通信协议，如http/1.1等。
<code>getScheme()</code>	获取URL协议，如http等。
<code>getContextPath()</code>	获取URL相对路径(虚拟目录)，如/image等。
<code>getServletPath()</code>	获取请求的JSP页面所在的目录。
<code>getContentLength()</code>	获取HTTP请求的长度。
<code>getMethod()</code>	获取表单提交信息的方式，如POST或GET。
<code>getHeader(String s)</code>	获取请求中头的值。
<code>getHeaderNames()</code>	获取头名字的一个枚举。
<code>getHeaders(String s)</code>	获取头的全部值的一个枚举。
<code>getRemoteAddr()</code>	获取客户的IP地址。
<code>getRemoteHost()</code>	获取客户机的名称(如果获取不到，就获取IP地址)。
<code>getServerName()</code>	获取服务器的名称。
<code>getServerPort()</code>	获取服务器的端口号。
<code>getParameterNames()</code>	获取表单提交的name参数值的一个枚举。
<code>getRequestURI()</code>	获得URI
<code>getQueryString()</code>	获得URL的查询字符串（整个的，没有分开key-value对）

[参考1](#)

例： 如果点击链接的URL为<http://www.groupbbs.com:8080/jsp/file/request.jsp?x=1&y=2>，通过下面的方法可以得到值：

<code>request.getScheme():</code>	<code>http</code>
<code>request.getContextPath():</code>	<code>/jsp</code>
<code>request.getServletPath():</code>	<code>/file/request.jsp</code>
<code>request.getContentLength():</code>	<code>-1(没有正文)</code>
<code>request.getMethod():</code>	<code>GET</code>
<code>request.getRemoteAddr():</code>	<code>132.236.174.191</code>
<code>request.getRemoteHost():</code>	<code>132.236.174.191</code>
<code>request.getServerName():</code>	<code>www.groupbbs.com</code>
<code>request.getServerPort():</code>	<code>8080</code>
<code>request.getRequestURI():</code>	<code>/jsp/file/request.jsp</code>
<code>request.getQueryString():</code>	<code>x=1&y=2</code>
<code>request.getRequestURL():</code>	<code>http://www.groupbbs.com:8080/jsp/file/request.jsp</code>
<code>request.getRealPath("/"):</code>	<code>C:\Program Files\Apache Software Foundation\Tomcat 8.0\webapps\jsp\</code>
<code>request.getServletContext().getRealPath("/"):</code>	<code>C:\Program Files\Apache Software Foundation\Tomcat 8.0\webapps\jsp\</code>

可以用读文件的方法读取**http**请求的正文：

```
int contentLength = request.getContentLength();  
DataInputStream in = new DataInputStream(request.getInputStream());
```

附录4、JSP页面构成元素

- JSP页面主要是在HTML页面中加入JSP的内容。JSP内容均在<%和%>之间定义。
- 生成响应页面时HTML模版直接输出到客户端，JSP内容要执行语句或给出编译指示，只有输出的内容会被发送到客户端。
- JSP页面由五种元素组成：
 - (1) HTML模版
 - (2) 指令标签
 - (3) 动作标签
 - (4) 脚本程序：声明片段、脚本片段和注释
 - (5) 表达式

<%@ 指令%>	对后面的JSP页面给出编译指示，主要指令有： page ， include ， taglib
<% 动作 %>	用来实现特殊的功能，主要标签： <jsp:include> <jsp:forward> <jsp:useBean> <jsp:setProperty> <jsp:getProperty>
<%! 声明%>	声明全局JSP变量和方法。它们被所有线程所共享。
<% 脚本片段 %>	有效的java程序段，其中定义的变量为局部变量。
<%-- 注释--%>	JSP注释
<%= 表达式%>	计算JSP表达式的值并输出到页面。

附录5、JSP指令标签(Directive)

- 概述

JSP指令标签指示JSP引擎对JSP页面需要做什么。

- 主要指令

- (1) Page指令
- (2) include指令
- (3) taglib指令(后面再讲)

• page指令

功能

用来定义整个JSP页面的属性。一个JSP页面可以有多条page指令。
除了import属性，其它属性只能定义一次。

语法(红色为默认值)

<code><%@ page [language="java"]</code>	定义所用语言(默认为java)
<code>[extends="package.class"]</code>	定义JSP文件的继承类
<code>[import="{package.class, ...}"]</code>	指明想要引入的java类包
<code>[session="true false"]</code>	是否允许使用session对象
<code>[buffer="none 8kb xkb"]</code>	输出缓冲的大小
<code>[autoFlush="true false"]</code>	当缓冲满时是否自动输出
<code>[isThreadSafe="true false"]</code>	采用多线程(默认)还是单线程

[info="text"]	定义JSP页面的文本信息
[contentType="text/html ; charset=ISO-8859-1"]	定义传送给浏览器的内容（或文件）类型和字符编码。
[errorPage="relativeURL"]	处理意外的页面
[isErrorPage="true false"]	指定当前页面是否可以处理来自另一个页面的错误，缺省为"false"。
[pageCoding="ISO-8859-1"]	定义JSP源程序使用的字符编码
%>	

- (1) **JSP**默认已引入以下包：java.lang.*，javax.servlet.*，javax.servlet.jsp.*，javax.servlet.http.*。其它类包必须用指令引入：
- ```
<%@ page import="java.util.*" %>
```
- ```
<%@ page import="java.io.*,java.net.*,java.sql.*" %>
```

- (2) **autoFlush**指明每当缓冲区满时是否自动把缓冲区的内容输出给客户端。如果autoFlush为false，当buffer不足时，会抛出错误。当buffer设置为none, autoFlush必须为true。

```
<%@ page buffer="24kb" autoFlush=false %>
```

- (3) 设置所支持的语言，目前JSP支持的语言只有java:

```
<%@ page language="java" %>
```

- (4) 用<%@ page contentType="text/html; charset=utf-8"%>

设置http响应的内容（或文件）类型和编码。

其它文件类型: *text/plain*(文本), *image/gif* (gif图像), *image/jpeg* (jpeg图像), *application/xshockwave* (flash文件), *application/msword* (msword文件), ... 。

支持的显示字符集(charset): *ISO-8859-1*, *GBK*, *GB2312*, *UTF-8*, ... 。

也可以采用<% response.setContentType("text/html; charset=utf-8"); %>进行设置。

这个指令会影响HTTP响应:

```
HTTP/1.1 200 OK\r\n
```

```
...
```

```
Content-Type: text/html; charset=utf-8\r\n
```

```
\r\n
```

正文

- (5) 用`<%@page errorPage="errorPage.jsp" %>`指出出错处理程序为errorPage.jsp。
在这个程序中用命令 `<%@page isErrorPage="true" %>` 说明是错误处理程序。用 `<%=exception.toString()%>` 显示出错信息。
- (6) 用`<%@page info="增加学生信息" %>` 定义页面信息
用`<%=getServletInfo()%>` 可以显示出该信息。
- (7) 用`<%@page pageEncoding="GBK" %>`。pageEncoding定义源文件编码，charset定义响应中的html文件的编码，同时出现时它们必须一致。两个都没定义时默认使用ISO-8859-1。只charset编码而没有pageEncoding时JSP源码采用charset编码。

附录6、JavaBean的scope

getServerTime.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<jsp:useBean id="getServerTime" class="Java.util.Date" scope="page" />
<%= getServerTime.toString() %>
```

对象名

当scope=page/request时，每次访问该网页(刷新页面)都会创建新的JavaBean对象。
当scope=session时，只有获得新的sessionID时，才会创建新的JavaBean对象。
当scope=application时，只有重启Web服务器才会创建新的JavaBean对象。在不同主机和浏览器访问时都会取到同一个的时间。

page表示将JavaBean实例对象存储在PageContext对象中，作用范围是当前JSP页面有效。
request表示将JavaBean实例对象存储在ServletRequest对象中，即可以被属于同一个请求的所有Servlet和JSP页面访问。
session表示将JavaBean实例对象存储在HttpSession对象中，存储在HttpSession对象中的JavaBean对象可以被属于同一个会话的所有Servlet和JSP页面访问。要求当前JSP页面支持Session，即page指令的session属性设置为true（默认）。
application表示将JavaBean实例对象存储在ServletContext对象中，存储在ServletContext对象中的JavaBean对象可以被同一个Web应用程序中的所有Servlet和JSP页面访问。也就是所有用户都可以使用。

附录7、配置虚拟主机和虚拟路径

- * 如果希望jsp文件被修改后自动编译运行，就需要在子目录META-INF下新建context.xml文件，内容为：<Context reloadable="true"/>
- * 所有应用程序都要使用的jar包应该放在子目录tomcat\lib中。jsp不需要配置classpath。
- * 下面为增加虚拟主机和虚拟目录的方法。webapps下的子目录都是虚拟目录。要在其它地方定义虚拟目录，需要修改配置文件tomcat/conf/server.xml。为了定义lab和test两个虚拟目录，下面在server.xml增加了两个配置：

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
  <Context path="lab" docBase="D:\Lab" reloadable="true"></Context>
  <Context path="test" docBase="D:\test" reloadable="true"></Context>
</Host>
<Host name="www.group.com" appBase="d:\group" unpackWARs="true" autoDeploy="true">
  <Context path="tech" docBase="D:\Tech" reloadable="true"></Context>
  <Context path="teach" docBase="D:\Teach" reloadable="true"></Context>
</Host>
```