# 简介

Servlet是J2EE 规范中的一种，主要是为了扩展java作为web服务的功能.从92年到的J2EE 1.2到现在J2EE8 从12个规范到现在20多个规范，越来越完善他的作用就是为java程序提供一个统一的web应用的规范，方便程序员统一的使用这种规范来编写程序，应用容器可以使用提供的规范来实现自己的特性。比如tomcat的代码和jetty的代码就不一样是吧，但作为程序员你只需要了解servlet规范就可以从request中取值，你可以操作session等等。不用在意应用服务器底层的实现的差别而影响你的开发。

当然你也可以自己写一个http 服务器，自己定义一套API，比如你在底层接受到一个http请求后，你把这个http请求的header、cookie和param等封装成一个MyRequest.class 。然后你要得到，你在你的MyServlet中从MyRequest对象中拿到param请求参数，校验成功后需要返还给浏览器一个HTTP response。其中必须要有一个session，所以你往cookie中写了一个字段，LAOZIDESESSIONID=878361839QWQWEQEQE,同时把这个sessionid放在了自己的内存中。下一次浏览器再访问你就会带上这个LAOZIDESESSIONID这个cookie，你就知道他原来已经访问过了，而且上一次访问的数据你都有(在第一次保存在内存中)。

但是有没有想过，如果每个程序员都写一个自己的HTTP服务器，该程序员离职了咋办。而且你用你的，我用我的，遇到问题都不能一起解决，你一会儿只支持http/1.0 ，别人都支持http/2.0了(虽然这个是在底层的实现了，和servlet没半毛钱关系，大家注意了，打个比方而已)。别人都支持注解了，你还在写配置呢！肯定不能啊，所有J2EE要出一个规范，要管住你们这群人，大家都要同步走。大家都用我这套规范，所有的请求都放在Request中，返回都放在response中。sessionID的名称也都可以自己设置，比如tomcat你可以Context sessionCookieName="zheshilaozidesessionid" class="ztext-empty-paragraph"><br>

讲了这么多废话，总结来说Servlet就是一群人来制定java应用中使用web时的各种规范，统一接口，其他内部实现由厂商自己实现，tomcat jetty jboss等等应运而生。面向接口编程！！很熟悉吧</p><p>关于他如何工作的：一个http请求到来，容器将请求封装成servlet中的request对象，在request中你可以得到所有的http信息，然后你可以取出来操作，最后你再把数据封装成servlet的response对象，应用容器将respose对象解析之后封装成一个http response。完了

# 实现方式

## 实现Servlet接口

|  |
| --- |
| /\*\*  \* Servlet的实现要实现Servlet接口  \*  \* **@author** mahao  \* **@date**: 2019年3月27日 下午6:26:55  \*/  **public** **class** Servlet1 **implements** Servlet {  /\* 只执行一次，当第一次请求时，执行这个操作 \*/  **public** **void** **init**(ServletConfig config) **throws** ServletException {  System.***out***.println("1. init--------");  }  **public** ServletConfig **getServletConfig**() {  System.***out***.println("getServletConfig-----------");  **return** **null**;  }  /\* 多次执行，每次来请求，就有执行 \*/  **public** **void** **service**(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {  System.***out***.println(req + "----Service--------" + req.getProtocol());  res.setContentLength(25000);//指定回复的属性信息  res.setContentType("Text/htmls4;charset=UTF-8");  res.getWriter().println("2. 你好++++");  }  **public** String **getServletInfo**() {  System.***out***.println("getServletInfo-----");  **return** **null**;  }  /\* 服务器停止时，调用这个方法 \*/  **public** **void** **destroy**() {  System.***out***.println("3. destroy----------");  }  } |

## 继承GenercServlet

|  |
| --- |
| /\*\*  \* Servlet实现方式二： 继承GenericServlet类  \*  \* **@author** mahao  \* **@date**: 2019年3月27日 下午6:28:54  \*/  **public** **class** Servlet2 **extends** GenericServlet {  @Override  **public** **void** **service**(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {  System.***out***.println("通过GenericServlet实现-----");  }  } |

## 继承HttpServlet实现

|  |
| --- |
| /\*\*  \* Servlet实现方式三： <br>  \* 继承HttpServlet类， 重写doGet/doPost方法；  \*  \* **@author** mahao  \* **@date**: 2019年3月27日 下午6:28:54  \*/  **public** **class** Servlet3 **extends** HttpServlet {  @Override  **protected** **void** **doGet**(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println("获得请求方式是--" + req.getMethod());  }  @Override  **protected** **void** **doPost**(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  doGet(req, resp);  }  } |

# Servlet生命周期

**初始化阶段**：  
Servlet容器加载Servlet，加载完成后，Servlet容器会创建一个Servlet实例并调用init()方法，init()方法只会调用一次  
Servlet容器会在一下几种情况装载Servlet：

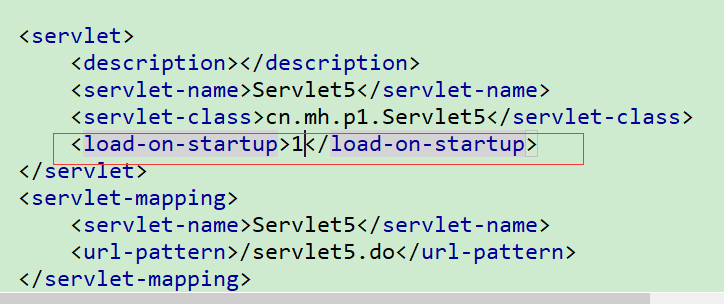
1. Servlet容器启动时自动装载某些servlet，实现这个需要在web.xml文件中添加<loadstartup>1</load-on-startup>
2. 在Servlet容器启动后，客户首次向Servlet发送请求
3. Servlet类文件被更新后，重新装载

**处理客户端请求阶段：**  
每收到一个客户端请求，服务器就会产生一个新的线程去处理。  
对于用户的Servlet请求，Servlet容器会创建一个特定于请求的ServletRequest和ServletResponse。  
对于tomcat来说，它会将传递来的参数放入一个HashTable中，这是一个String-->String[]的键值映射

**终止阶段：**  
当web应用被终止，或者Servlet容器终止运行，或者Servlet重新装载Servlet新实例时，Servlet容器会调用Servlet的destroy()方法

*Servlet不能独立运行，是供Servlet引擎调用的类，他的运行完全由Servlet引擎控制和调度。对客户端的多次Servlet请求，Servlet是第一次被请求时，加载到服务器容器中，servlet被加载一次，因此只存在一个实例，这个实例常驻内存。在处理请求时，服务器对每个请求开辟一个新的线程去处理请求，Servlet引擎创建一个新的HttpServletRequest和HttpServletResponse，并调用内存中的Servlet实例的service方法，service根据请求方式调用doXXX方法。但是Servlet存在并发问题，对成员变量要少写。*

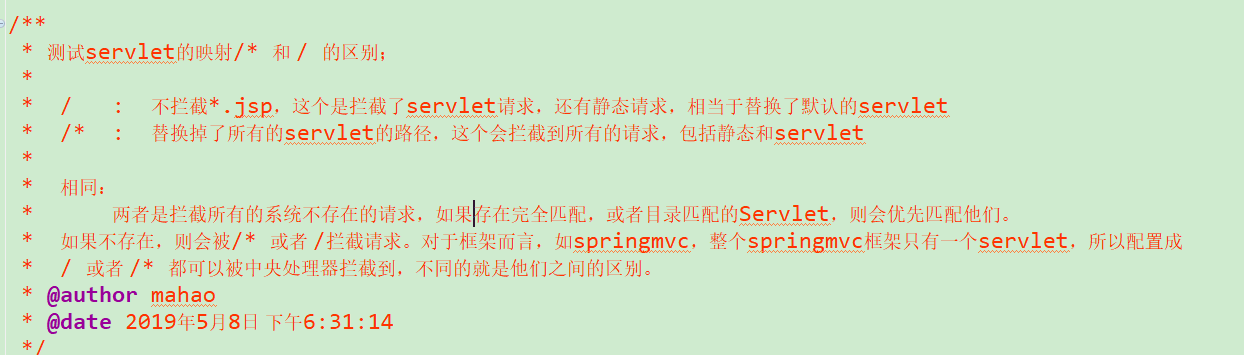
# Servlet自动加载



# Servlet映射配置和优先级

对于Servlet的虚拟目录访问路径配置一共有三种，优先级由大到小为

完全路径匹配 目录匹配 扩展名匹配



## 5.1 完全路径匹配

以/开头， /day09/demo1

访问地址： localhost:8080/day09/demo1

## 5.2 目录匹配

以 / 开头 ，含有相同目录即可匹配

/\* 可匹配项目下所有的访问

比如 /day09/dsal 或者/day09 或者/day09/aa/bb.do

/aa/\* 访问得到/aa下的所有Servlet

## 5.3 后缀名访问

\*.do 能访问所有的以do结尾的servlet

URL-Pattern的三种配置方式

1、完全路径匹配 以/开始 不能包含统配符\* 例如： /hello /init

2、目录匹配 以/开始, /\*结尾 例如： /\* /aa/\* /aaa/bbb/\*

3、扩展名匹配 不能以/开始，以\*开始 例如： \*.do \*.action

经典错误：/\*.do

|  |
| --- |
| **对于如下的一些映射关系：**   * Servlet1 映射到 /abc/\* * Servlet2 映射到 /\* * Servlet3 映射到 /abc * Servlet4 映射到 \*.do   **问题：**   * 当请求URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应   **Servlet引擎将调用Servlet1。**   * 当请求URL为“/abc”时，“/abc/\*”和“/abc”都匹配，哪个servlet响应   **Servlet引擎将调用Servlet3。**   * 当请求URL为“/abc/a.do”时，“/abc/\*”和“\*.do”都匹配，哪个servlet响应   **Servlet引擎将调用Servlet1**。   * 当请求URL为“/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应   **Servlet引擎将调用Servlet2.**   * 当请求URL为“/xxx/yyy/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应   **Servlet引擎将调用Servlet2。** |

# Web中的路径问题

## 6.1 相对路径（以 . 开头 ，或者不写）

当前目录表示： ./ 或者 不写

相对路径是的项目名对应eclipse中的WebRoot，路径相对是针对于服务器中的文件目录而言

|-项目名

|- class文件

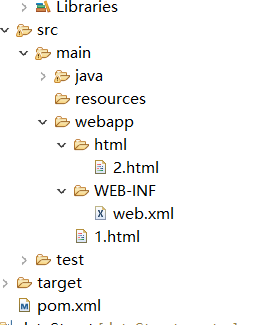
|-WEB-INF

|-lib

|-web.xml

|-js/css/html

对于./ 是指当前文件所在的文件夹



对于2.html而言，./是指html文件夹 ../ 是指 当前项目根目录，和1.html同一文件。

## 6.2 绝对路径 （以/ 开头）

用绝对地址访问项目;

重要：

客户端：

<http://localhost:8080/bServlet/html/2.html>

简写： /bServlet/html/2.html

服务端：

不用加项目名：

比如在文件下载时：访问根目录下的upload文件夹下a.png

客户端：/day09/upload/a.png

服务端 /upload/a.png

相对路径：

服务端：./upload/a.png

客户端：根目录下： ./upload/a.png

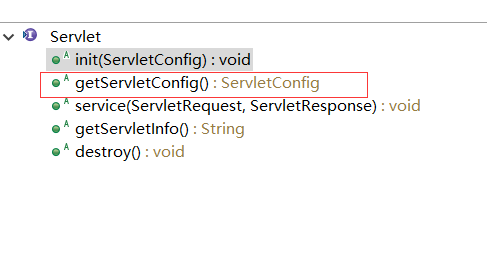
|  |
| --- |
| 路径问题  在WebRoot下 新建 1.html  在WebRoot/aa 下新建 2.html  在1.html 和 2.html 分别通过 超链接 访问 HelloServlet  1.html  <h1>相对路径</h1>  <a href="./hello">HelloServlet</a>  <a href="hello">HelloServlet</a>  <h1>绝对路径</h1>  <a href="http://localhost/day5/hello">HelloServlet</a>  <a href="/day5/hello">HelloServlet</a>  2.html  <h1>相对路径</h1>  <a href="../hello">HelloServlet</a>  <h1>绝对路径</h1>  <a href="http://localhost/day5/hello">HelloServlet</a>  <a href="/day5/hello">HelloServlet</a>  </body>  路径分为相对路径和绝对路径两种写法  1. 相对路径，根据当前资源路径 与 目标资源路径，寻找相对位置关系，通过 . (当前目录) 和 .. (上一级目录) 访问目标资源  1.html 访问 HelloServlet  当前路径 http://localhost/day5/1.html  目标路径 http://localhost/day5/hello  位于同一个目录中 ./hello 、hello ===== 替换当前路径最后一个资源  2.html 访问 HelloServlet  当前路径 http://localhost/day5/aa/2.html  目标路径 http://localhost/day5/hello  上一级目录中 ../hello ===== 替换上一级目录资源  \*\*\*\*\* 相对路径，总需要分析当前路径与目标路径对应关系，编写规则会根据当前路径不同而不同  2、绝对路径  带有协议完整路径 （跨网站） http://localhost/day5/hello  以/ 开始路径 （同一个站点内） : /day5/hello  服务器端和客户端对于/ 的区别  客户端访问路径：/day5/hello  服务器内部路径：/hello  结论：web访问中所有资源路径，都使用绝对路径  =========================================================================================================================== |

# ServletConfig

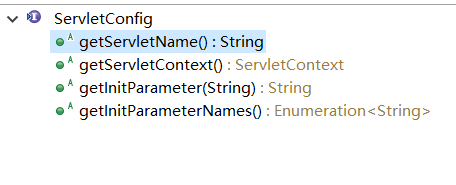


servlet启动的配置文件，可以读取web.xml中的servlet中的数据；

1. 获取对象方式

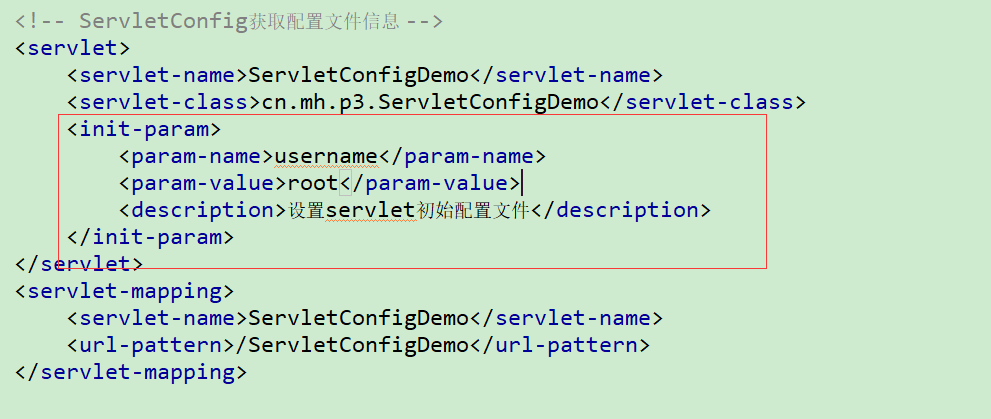


1. ServletConfig对象API



1. 测试使用：

在Servlet中填写配置文件



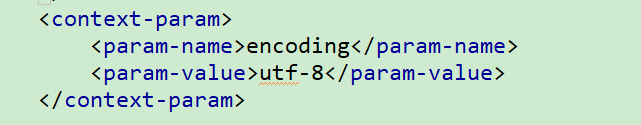


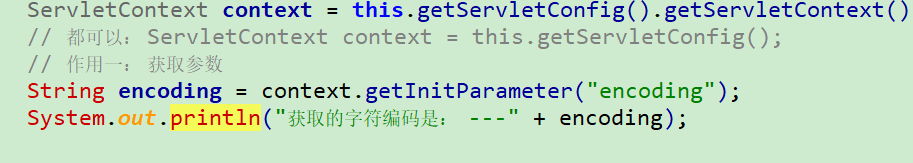
# ServletContext Servlet的上下文对象

获取方式通过Servletconfig对象，ServletContext对象包含在ServletConfig对象中，ServletConfig对象是servlet初始化时，由web服务器提供给servlet；

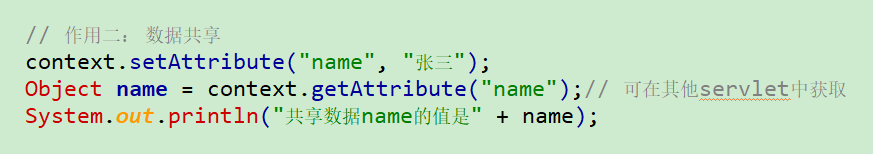
## 8.1 提供全局的参数初始化，比如编码

通过servletContext的[**getInitParameter**](mk:@MSITStore:D:\文档\java_ee_api_中英文对照版.chm::/javax/servlet/ServletContext.html#getInitParameter(java.lang.String))**（String name）获取web.xml中的配置信息**





## 8.2 web程序内数据共享



## 8.3 获取资源（web配置文件的路径问题）



注意： 在web应用程序中WebRoot对应的是项目名，发布到服务后，webRoot是虚拟目录，就会变成<项目名，src源码包，则WEB-INF下的classes文件，context.getResourceAsStream("/WEB-INF/classes/conf/db.properties");

# 缺省Servlet

# Response

## 10.1 重定向



## 10.2 页面定时跳转

response.setHeader("refresh","5;url=/bservlet/servlet1.do");

5秒后跳转到servlet1.do请求；

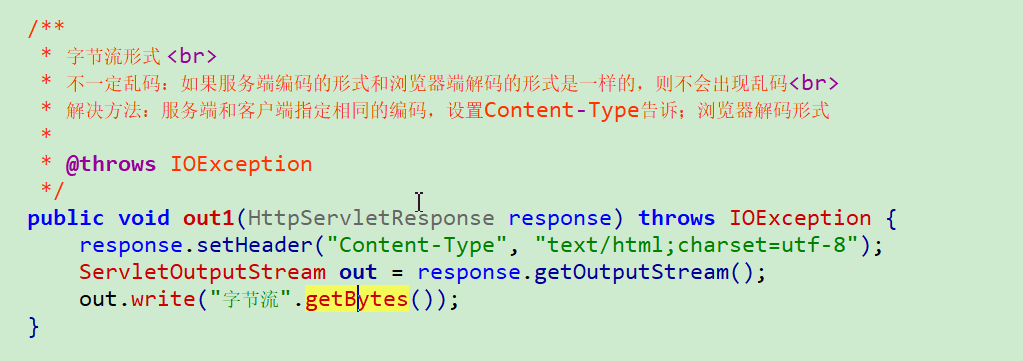
## 10.3 禁用浏览器缓存

|  |
| --- |
| response.setHeader("Cache-Control", "no-cache");  response.setHeader("Pragma", "no-cache");  response.setDateHeader("Expires", -1); |

## 10.4 解决中文问题乱码

字节流：

服务端编码和浏览器打开文件编码一致



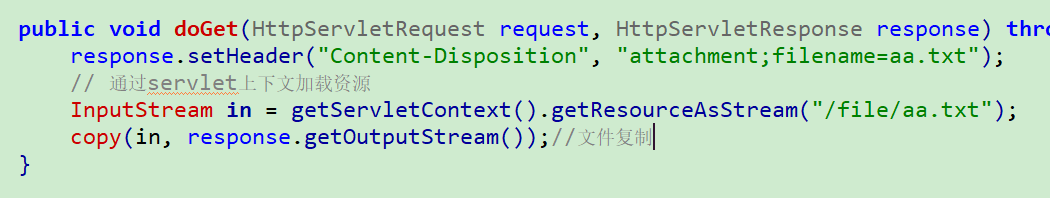
字符流：

设置浏览器编码,设置服务端缓冲区编码；



## 10.5 文件下载

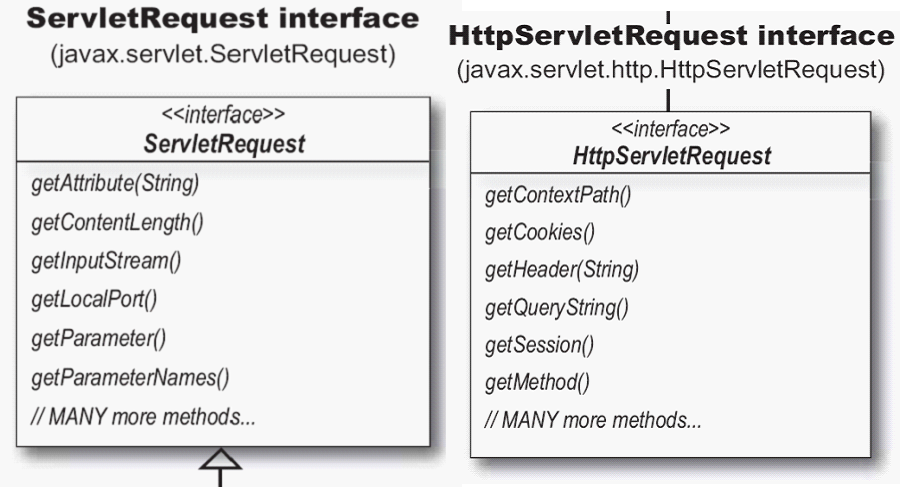
Content-Disposition: attachment;filename=aa.txt



中文下载名乱码

|  |
| --- |
|  |

# Request



## 11.1 获取客户机信息

|  |
| --- |
| **public** **void** **doGet**(HttpServletRequest request, HttpServletResponse response) {  response.setContentType("text/html;charset=utf-8");  System.*setOut*(**new** PrintStream(response.getOutputStream()));    String **ip** = request.getRemoteAddr();  System.***out***.println("<br>ip地址是： " + ip);  String **method** = request.getMethod();  System.***out***.println("<br>请求方式是： " + method);  String **contextPath** = request.getContextPath();  System.***out***.println("<br>获取虚拟路径 "+contextPath);  StringBuffer **url** = request.getRequestURL();  System.***out***.println("<br>获取请求完整url "+url);  String **uri** = request.getRequestURI();  System.***out***.println("<br>获取请求资源uri "+uri);  }  ip地址是： 127.0.0.1  请求方式是： GET  获取虚拟路径 /bServlet  获取请求完整url http://127.0.0.1:8080/bServlet/P6Servlet1  获取请求资源uri /bServlet/P6Servlet1 |

## 11.2获取请求头信息

|  |
| --- |
| //遍历头信息  Enumeration<String> **names** = req.getHeaderNames();  **while**(names.hasMoreElements()) {  String **name** = names.nextElement();  System.***out***.println(names+"---------"+req.getHeader(name));  } |

重要的头：

referrer 当前请求来源地址

User-Agent 客户端浏览器类型

if-Modified-Since 配合last-Modified 和304状态码控制缓存

## 11.3获取请求参数

|  |
| --- |
| // 1.获取input单个数据  String **name** = request.getParameter("username");  // 2.获取爱好多个数据  String[] **loves** = request.getParameterValues("love");  // 3.通过getParameterMap获取数据  Map<String,string[]> **parameterMap**= request.getParameterMap(); |

### 11.3.1 解决请求乱码问题

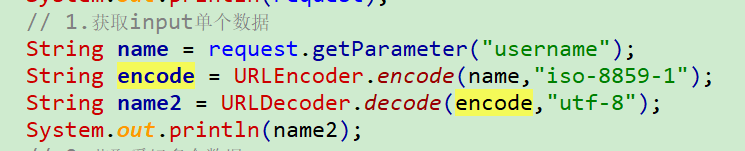
#### 1. POST请求

request.setCharacterEncoding（“utf-8”）；

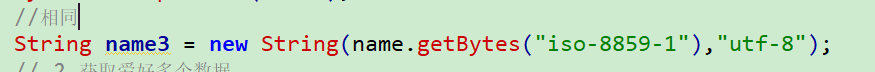
#### 2. GET请求

对于get方式请求，浏览器对中文进行编码（username=%E9%A9%AC%E8%B1%AA 马豪）utf-8编码表，提交到后台后，服务器对提交的数据新进行解码，tomcat在8之前默认采用的编码表示ISO-8859-1，所以会产生乱码。

解决： 对采用iso-8859-1解码的数据在用iso-8859-1进行编码，则将数据恢复成了解码之前的数据，在采用正确的编码表进行解码；



//简写形式



## 11.4 利用请求域传递对象

## 11.5重定向和转发的区别

重定向：

是浏览器发送请求后，服务器设置302状态码，并设置Location指定请求路径。这是浏览器发送两次请求，请求的HttpServletRequrst对象不同。重定向可以去任何url；

转发：

转发是浏览器发送请求后，服务器送请求，请求其他数据，是浏览器发送，对客户端说，只有一次请求，所有requrst是同一个。所以可以使用request域传递数据。转发只能在程序内进行转发

# JSP

## 12.1 简介

jsp用于servlet生成html页面，

java server jsp java服务端网页，是在服务端执行

jsp执行在服务端，也是servlet程序，翻译成Servlet—编译—执行

## 12.2 jsp脚本元素（jsp页面写java代码）

<%! java代码 %> 定义类，常量方法（不常用）成员变量

<%= java代码 %> 输出语句

<% java代码 %> 逻辑代码，定义变量

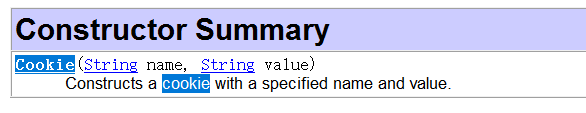
案列：

|  |
| --- |
| <table>  <%  **for** (**int** i = 0; i < 10; i++) {  %>  <tr>  <%  **for** (**int** j = 0; j < 10; j++) {  %>  <td>  <%=i%>--<%=j%>  </td>  <%  }  %>  </tr>  <%  }  %>  </table> |

# EL

# cookie

是将信息写入浏览器，以后浏览器访问服务器自动带着这个信息请求，数据是存在客户端的；



## 14.1 setMaxAge（int expiry）有效时间

设定cookie的有效时间，cookie默认情况下，是一个会话级别的保存时间，cookie默认保存到浏览器内存中。setMaxAge（）是将cookie持久化化的操作，将保存到硬盘上，参数是秒。



## 14.2 setPath（String url）有效路径

设置cookie的有效路径，指定某些请求访问时，携带cookie，默认的是跟从servlet的上级请求。这样指定cookie的有效路径，指定cookie作用于某些请求，减少网络传输；

当前servlet的访问路径是 /cookie/P2Cookie2 --默认的cookie作用范围是/cookie/下 <br>

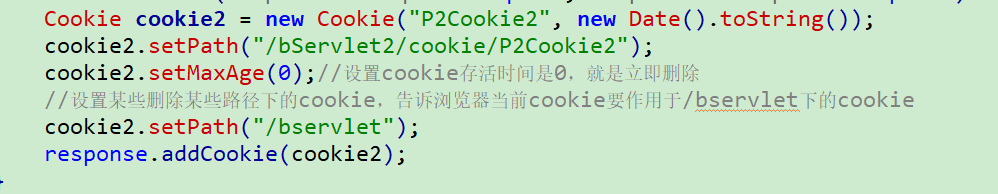
\* P2Cookie的访问路径是/P2Cookie ---默认的请求头携带的是 /项目名 <br>

\* 所以： 当浏览器中同时存在两个cookie，访问/cookie/P2Cookie2时，请求时携带者两个cookie，

\* 但是请求/P2cookie时，只有一个cookie，/cookie/P2Cookie2不携带

**setPath 和 setMaxAge（）删除cookie**

**为cookie设置存活时间是0，设置删除的cookie，设置为 / ，确保删除。**



## 14.3 setDomain(String pattern) 有效域名

# session

session是将数据存储在服务端，服务端对每个会话都有一个session，服务器可以接受多个会话，在同一个会话内，所有的请求共享同一个会话中的数据。 如何区别每个请求所属于那个会话呢，这个就是要根据cookie，浏览器第一次请求时，服务端生成一个JSESSIOID，放入到cookie中，这个cookie则会在以后的请求中，携带者sessionid，服务器根据这个sessionID去寻找这个会话的session，所以每个会话都有独特的session。

# JSP

## 16.1 JSP的指令元素

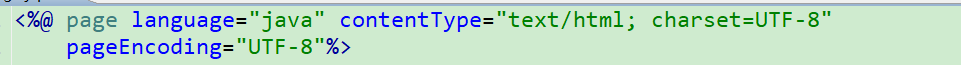
语法： <%@ 指令元素 属性名=属性值 %>

\*page ： 设定jsp的属性

\*include ：包含页面

\*taglib ：引入标签库文件

1.**page**



1. language ： 设置jsp文件嵌入的语言
2. extends ： 继承，指定jsp翻译成servlet继承的类
3. session： session="false" 关闭页面是否开启session
4. import <%@ page import="java.util.List" %>
5. pageEncoding : 设定jsp文件翻译成servlet的编码
6. content-type： 设定服务器输出页面的编码（5 6 没有联系）

2.**include**

1. 导入其他页面的代码；

静态包含的原理： 在jsp文件翻译成java之前，将需要静态导入的jsp文件，合并到一个jsp文件中，里面内容是原封不动的，然后在一起翻译成java文件。

2.动态导入：

3.**taglib**

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

## 16.2 jsp内置对象

1.request --触发服务调用的请求

2.response --对请求的应答

3.session --为请求客户创建的session对象

4.application –从servlet配置对象中获取的servlet的上下文（ServletContext）。

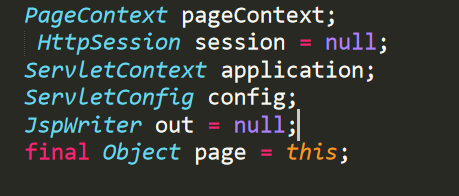
5.out --JspWriter

6.pageContext –域对象，对当前页面有效

7. page

8.config --ServletConfig servlet的配置

9.exception

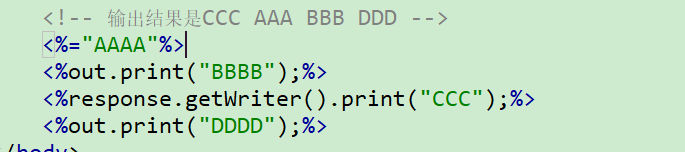


* out详解：

输出有两个对象

jsp内置out对象 对应 JspWriter

servlet中的PrintWriter 对应response.getWriter();



分析：out对象调用的方法也是response.getWriter();才能将数据写回客户端。out将jsp页面上的html代码存放到jspWriter的缓冲区中，等缓冲区满或者html代码写入完毕，将调用getWriter写回客户端。所以response.getWriter().write（），是先于out执行的。

* pageContext详解

