Servlet01

CS/BS都是什么

cs:Client/Server;客户端/服务器
比如qq等安装到电脑上的程序,就是客户端
BS:Brower/Server;浏览器/服务器

Web资源

web资源

html页面

JSP/Servlet

访问web资源

在浏览器中输入的地址,就是在访问web资源

web服务器

web服务器的作用就是接收客户端的请求,给客户端做出响应
JavaWeb服务器中,将需要的JSP/Servlet动态资源转换成静态资源,可以称他们为JSP/Servlet容器

Servlet

1. 什么是Servlet Servlet是JavaWeb的 **三大组件** 之一,它属于动态资源, Servlet的作用是处理请求,服务器会把接收到的请求交给Servlet来处理, 在Servlet中通常需要:
    - 接收请求数据;
    - 处理请求;
    - 完成响应;
2. 创建Servlet的方式
    - 实现javax.servlet.Servlet接口

    - 继承javax.servlet.GenericServlet类
    - 继承javax.servlet.http.HttpServlet类,会专门对Http请求提供一些方法

3. 在浏览器上访问Servlet
   在web.xml文件中配置Servlet的访问路径

4. Servlet的生命周期

    - init(config);
    - service(req,rsp);
    - destory();

5. Servlet的特性

    - 每个Servlet都是单例模式,但是可以存在多个Servlet类线程不安全,效率高
    - Servlet类由我们创建,Servlet对象由服务器(Tomcat)创建,Servlet对象的方法也是由服务器调用

ServletConfig接口

web.xml中的信息,会被加载进内存,就被抽象成了ServletConfig的实现类对象
获得名字,获得参数,遍历参数名

GenericServlet

HttpServlet

- 注意继承init();
- doGet()与doPost();
- 能方便的根据请求方式,操作浏览器发起请求时携带的参数.

ServletContext对象

是Servlet的上下文对象,什么是上下文?
寿与天齐

获得方式:

- SrevletConfig#getServletContext();
- GenericServlet#getServletContext();

什么是域对象?

常用方法

- setAttribute(String name,Object value);//存放一个对象
- getAttribute(String name)://用来获取ServletContext中的数据
- removeAttribute(String name)://移除ServletContext中的域属性
- Enumeration getAttributeNames()://获取所有域属性的名称

应用初始化参数与Servlet初始化参数

```
<context-param/>
<init-param/>
```

使用ServletContext获取项目资源

创建顺序

```
<load-on-startup>
```

关于url-pattern标签

快速创建Servlet的方式

服务器处理请求的过程

看图.

Response

用来设置服务器响应给客户端的内容.

状态码:

- 200:成功
- 302:重定向
- 404:客户端出错(访问的资源不存在)
- 505:服务端出错 #### 方法:
- setStatus(sc):发送正确的状态码
- sendError(sc):发送错误的状态码

响应头:

- Content-Type,Refresh,Location等
- 响应头有什么用呢?

设置响应头的方法:

- setHeader(name,value);单值
- addHeader(name,value);多值
- setIntHeader(name,intValue);int类型单值
- addIntHeader(name,intValue);int类型多值
- setDateHeader(name,longValue);毫秒单值
- addDateHeader(name,longValue);毫秒多值

示例:重定向

重定向的实现过程

- 设置Location响应头
  - setHeader(“Location”,”/项目名/Servlet路径”);
  - setStatus(302);

示例:定时刷新

- 定时刷新的实现过程
- 设置Refresh响应头
  - setHeader(“Refresh”,”秒数;URL=/项目名/Servlet路径”);

响应体:用于HTML中显示

- ServletOutputStream:发送字节数据的流
- PrintWriter:发送字符数据的流

- 两个流不能同时使用

示例:显示图片

- 使用FileInputStream读取图片
- 使用IOUtils将读取到的图片转换成字节数组
- 使用ServletOutputStream.write将字节数组写入到浏览器

示例:快捷的重定向设置

- sendRedirect(“/项目名/路径”);

Request

封装了客户端所有的请求数据.

获取常用信息:

```
获取客户端IP:getRemoteAddr();
获取请求方式:getMethod();
获取浏览器信息:getHeader(“User-Agent”);
    User-Agent是请求头
```

获取请求头:

```
getHeader(name);
getIntHeader(name);
getDateHeader(name);
getHeaders(name);多值请求头
```

示例:判断是否为本站的超链接发出的(防盗链)

根据请求头Referer的值判断,如果值为null,则
getHeader(“Referer”) == null
是从地址栏中输入的网址

示例:获取请求的URL
- http://localhost:8080/Request&Response/as?uname=abc&password=xxx(一个完整的URL路径)

- String getScheme();获取协议:http
- String getServerName();获取服务器名,localhost
- int getServerPort();获取服务器端口,8080
- String getContextPath();获取项目名,/Request&Response
- String getServletPath();获取Servlet路径,/as
- String getQueryString();获取参数部分,即问号后面的部分,uname=abc&password=xxx
- String getRequestURI();获取请求URI,等于项目名+Servlet路径,/Request&Response/as
- StringBuffer getRequestURL();获取请求URL,等于不包含参数的整个请求路径http://localhost:8080/Request&Response/as

示例:获取请求参数
请求参数是客户端发送给服务器的;
可能是在请求体中(POST),也可能在url的参数列表中(?后面)(GET)
getParameter(name);
getParameterValues(name);多值
getParameterMap();获取所有请求参数,返回值类型:Map

示例:html中超链接链接参数,CheckBox使用

请求转发,请求包含

什么是请求转发?

在一次请求中,先访问AServlet,在AServlet中直接访问了BServlet(B也可以再访问C)的请求过程.
看图.
注意:只有最后一个Servlet可以设置响应体,前面所有Servlet设置的响应体都不会有效果
注意:与重定向不同
注意:请求转发或包含只有一套Request/Response,而重定向有多个.

```
rd = getRequestDispatcher(“/路径“);注意这里没有项目名
rd.forward(req,rsp);转发
rd.include(req,rsp);包含
```

演示请求转发,演示请求包含
请求转发和重定向的区别.
响应次数
地址栏变化
目标区间
路径组成

效率

Request域

在Servlet中有三个域对象,request,session,application都有以下三个方法
setAttribute(name,objValue);这就是在设置域属性.
getAttribute(name);
removeAttribute(name);

域属性与参数的区别.
看图.

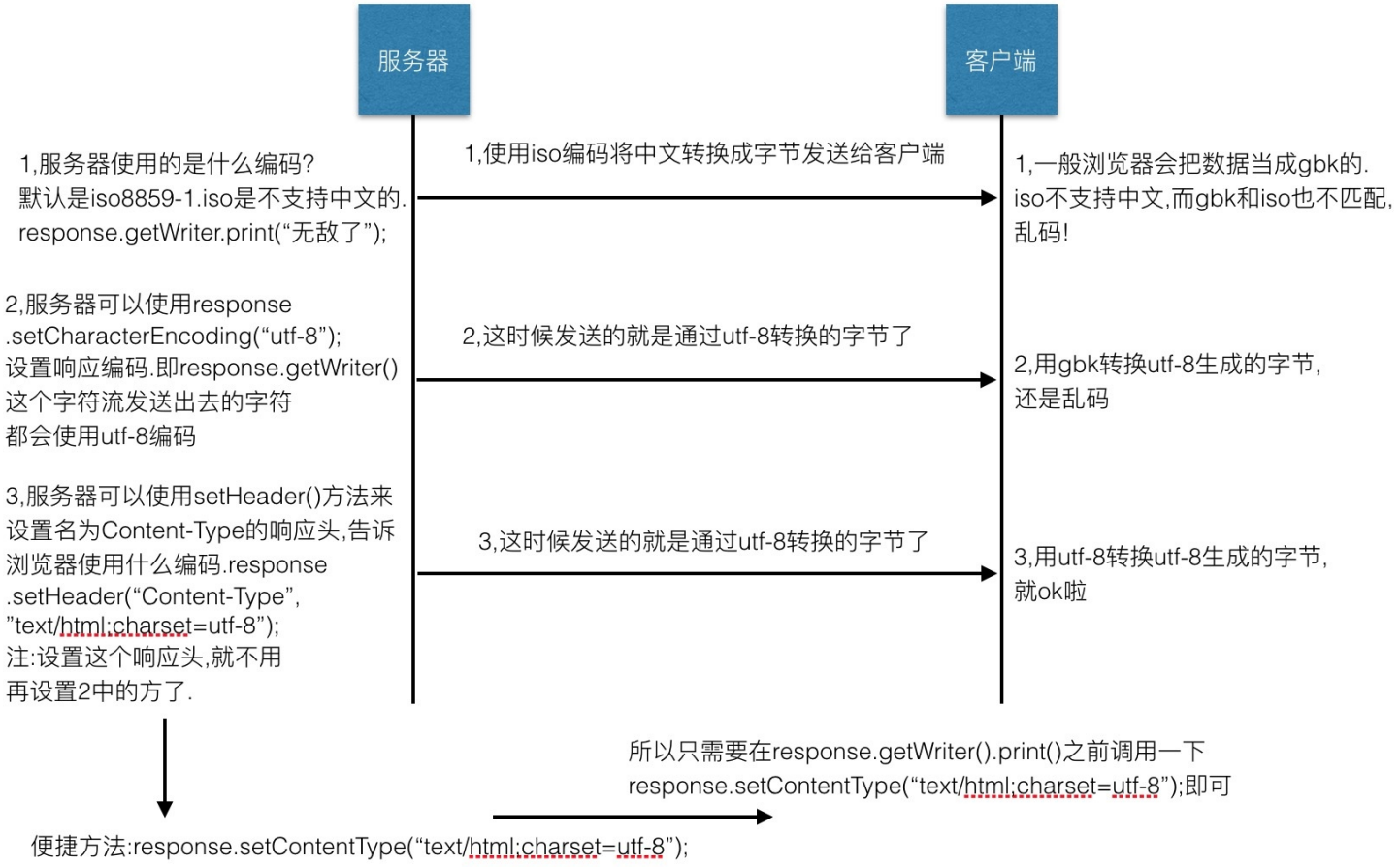
一,字符编码

字符在存入的时候,都会被转换成二进制的字节,如果存的时候,按照一个规范来存,取的时候按照另一个规范来取的话,就会出现乱码.
- iso-8859-1(国际标准化组织命名的不支持中文,在ASCII码基础上扩充了一些拉丁字符);
- gbk(系统默认编码,中国的国标码,是在gb2312基础上的扩展规范),gb18030是在gbk的基础上的又一次扩充;
- utf-8(万国码,支持全世界的编码,使用这个就可以了)

二,响应编码

- 当使用response.getWriter()来向客户端发送字符数据时,如果在之前没有设置编码,那么默认使用iso,因为iso不支持中文,一定会乱码.
- 在使用response.getWriter()之前,可以使用response.setCharacterEncoding()来设置字符流的编码为gbk或utf-8,一般都会设置成utf-8;
- 在使用response.getWriter()之前,可以使用response.setHeader("Content-Type","text/html;charset=utf-8");来通知浏览器,服务器使用的是utf-8编码.
- setHeader("Content-Type","text/html;charset=utf-8");可以完成设置服务器编码和通知浏览器服务器是什么编码两步,快捷写法是:response.setContentType("text/html;charset=utf-8");

编码的设置过程:响应编码

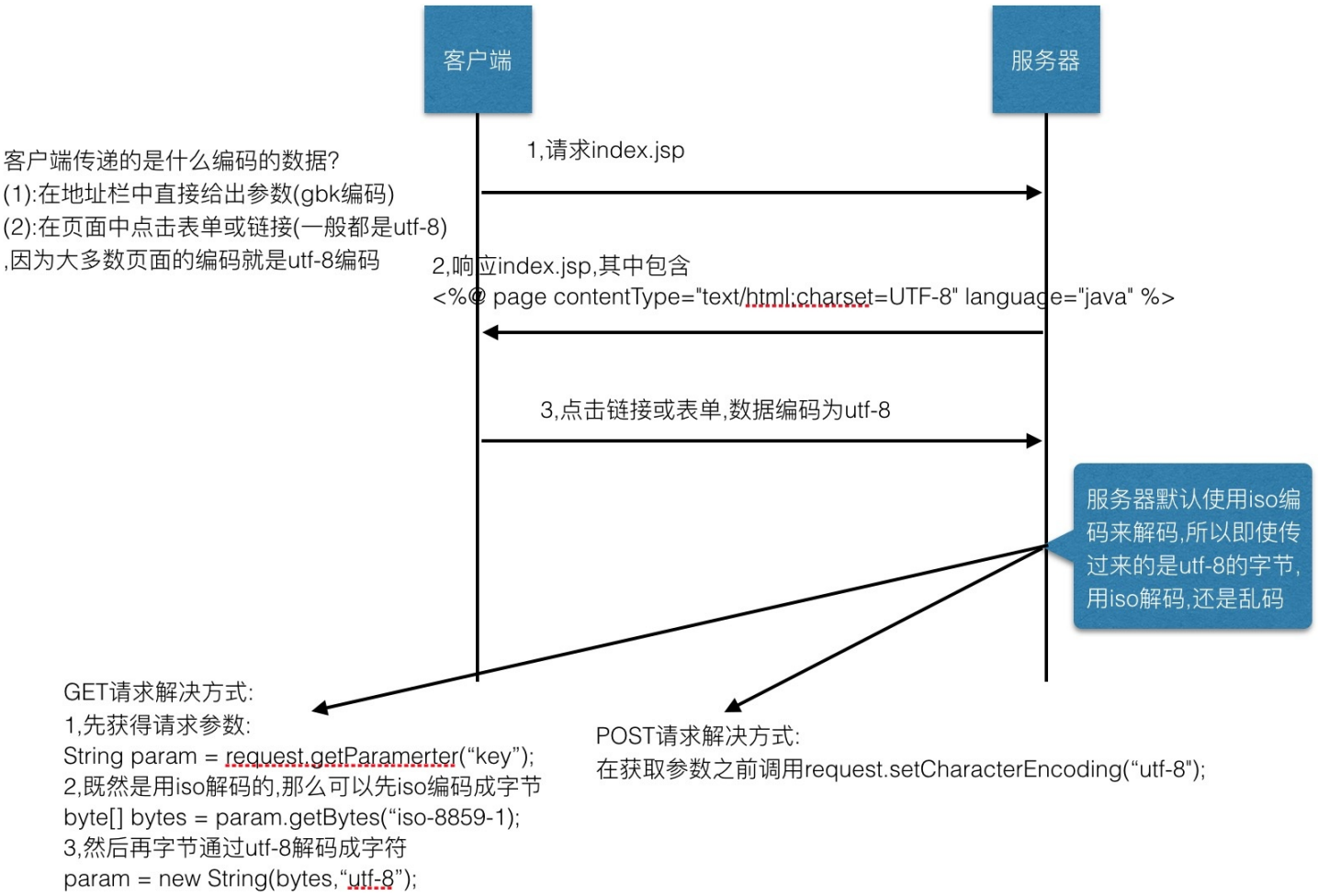


三,请求编码

- 客户端发送给服务器的请求参数是什么编码:
  - 客户端首先打开一个页面,然后在页面中提交表单或点击超链接,在请求这个页面时,服务器响应的编码是什么,那么客户端发送请求时的编码就是什么.
- 服务器默认使用什么编码来解码参数:
  - 服务器端默认使用ISO-8859-1来解码,所以这里一定会出现乱码,因为iso-8859-1是不支持中文的.
- 请求编码处理分为两种:GET和POST:GET请求参数不在请求体中,而POST请求参数在请求体中,所以它们的处理方式是不同的.
- GET请求编码处理:
  - String name = new String(request.getParameter("name").getBytes("iso-8859-1"),"utf-8");
  - 也可以在/conf/server.xml中配置端口号的标签中添加配置:URIEncoding=utf-8;但是这种修改只是本地的修改.
- POST请求编码处理:

- 在获取参数之前调用request.setCharacterEncoding("utf-8");

编码的设置过程:请求编码



四,URL编码

- 首先URL编码不是字符编码.
- URL编码,是客户端与服务端之间传递参数的一种方式.
- URL编码需要先指定一种字符编码,把字符串解码后,得到byte数组,然后把小于0的字节+256,再转换成16进制,前面再添加一个%.
- 就是将中文转换成%后面跟随两位16进制的格式.
- POST请求默认就使用URL编码,tomcat会自动使用URL编码.
- URL编码:String username = URLEncoder.encode(username,"utf-8");
- URL解码:String username = URLDecoder.encode(username,"utf-8");

- 需要将链接中的中文参数,使用url编码!通过jsp实现.

为什么要使用URL编码?

在客户端和服务器之间传递非英文时需要将文本转换成网络适合的方式.(传中文,有时会丢一个半个字节的);

使用URL编码可以方便,安全的传输;

```
String name = "张三";
//得到name进行utf-8编码后的字节数组
byte[] bytes = name.getBytes("utf-8");
//[-27, -68, -96, -28, -72, -119]
System.out.println(Arrays.toString(bytes));

//对name字符串,以utf-8编码的格式进行url编码得到
// un:%E5%BC%A0%E4%B8%89
//对[-27, -68, -96, -28, -72, -119]中的负数
//加255后,再转换成16进制,再在前面加上%得到最终结果
String un = URLEncoder.encode(name,"utf-8");
System.out.println(un);

//将url编码生成的带%的字符串解码成正常的字符串:张三
un = URLDecoder.decode(un,"utf-8");
System.out.println(un);
```

相关路径

- web.xml中路径:Servlet路径
  - 要么以星号开头,要么以"/"斜杠开头
- 转发和包含路径:使用以斜杠开头
  - 以"/"斜杠开头:相对当前项目路径
  - 例如:http://localhost:8080/项目名/
  - request.getRequestDispacher("/BServlet").forward(request,response);这里的/BServlet就是相对于上面的地址
  - 不以"/"开头:相对当前Servlet路径
- 重定向路径(客户端路径):使用绝对路径最好
  - 以"/"开头:相对当前主机,例如:http://localhost:8080/
  - 所以需要手动添加项目名,如:response.sendRedirect("/DemoProject/DemoServlet");
- 页面中超链接和表单路径:使用绝对路径最好
  - 与重定向相同,都是客户端路径,需要添加项目名称.
  - <form action="/DemoProject/DemoServlet"/>

    - <a href="/DemoProject/DemoServlet"/>
    - <a href="DemoServlet"/>,如果不以"/"开头,那么相对当前页面所在路径,
    - 如果当前页面为:http://localhost:8080/DemoProject/demo.html
    - 则不以"/"开头的实际路径为:http://localhost:8080/DemoProject/DemoServlet
- ClassLoader获取资源路径
    - 相对Classes目录
- Class获取资源路径
    - 以"/"开头相对classes目录
    - 不以"/"开头相对当前.class文件所在目录

JSP01

一,JSP简介

JSP是一种运行在服务器端的脚本语言,是用来开发动态网页的,该技术是JavaWeb程序开发的重要技术.

JSP是一种动态网页技术标准,它是在静态网页HTML代码中加入Java程序片段(Scriptlet)和JSP标签(Tag),构成JSP网页文件,该文件扩展名为:.jsp.

当客户端请求JSP文件时,服务器执行该JSP文件,然后以HTML的格式返回给客户端,及JSP程序的执行是由Web服务器来完成的.

二,JSP的作用

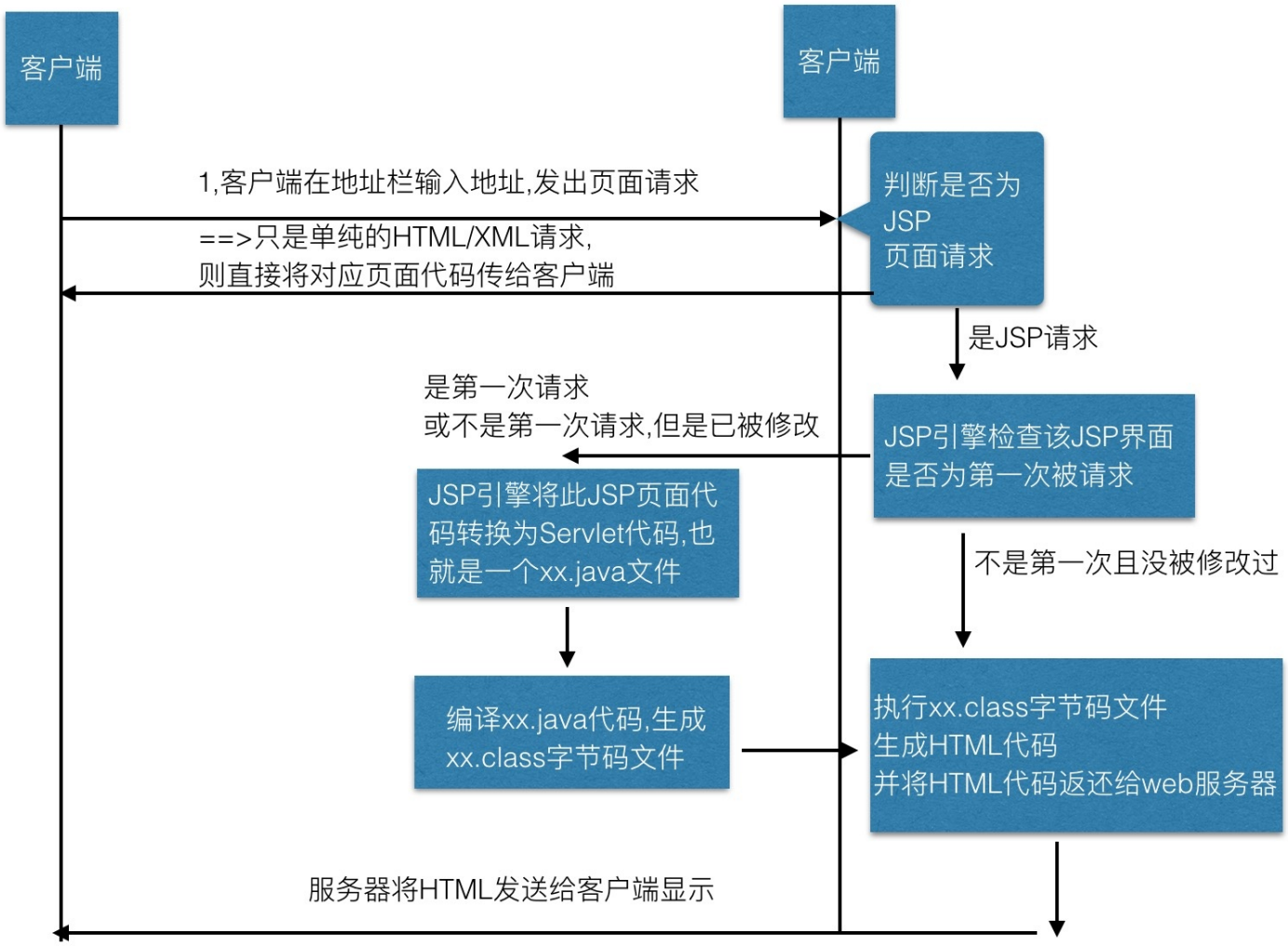
- Servlet:
    - 缺点:不适合设置html响应体,需要大量的response.getWriter().print("xxx");
    - 优点:动态资源,可以编程.
- html:
    - 缺点:html是静态页面,不能包含动态信息
    - 优点:不用为输出html标签而发愁
- JSP:(Java Server Page)
    - 优点:在原有html的基础上添加java脚本,构成JSP页面.

三,JSP和Servlet的分工

- JSP:
  - 作为请求发起页面,例如显示表单,超链接.
  - 作为请求结束页面,例如显示数据.
- Servlet:
  - 作为请求中处理数据的环节.

四,JSP的组成

- JSP=html+java脚本+jsp标签(指令)
- JSP中无需常见即可使用的对象一共有九个,也就是九个JSP内置对象.例如:Request对象,out对象等
- 3种java脚本:
  - <% ... %>:java代码片段(常用),用于定义0~N条java语句;
  - <%= ... %>:java表达式,用于输出(常用),用于输出一条表达式(或变量)的结果.
  - <%! ... %>:声明,用来创建类的成员变量和成员方法(基本不用)
- 基本原理:
  - 当JSP页面被第一次访问时,服务器会把JSP文件编译成java文件(就是一个Servlet类)
  - 然后在把.java文件编译成.class字节码文件
  - 执行.class文件,创建该类对象
  - 调用该对象的service()方法
  - 第二次请求同一jsp文件时(该文件未被修改过),直接调用service()放方法.



五,JSP语法

JSP页面主要是将JSP代码放在特定标签中,再嵌入到HTML代码中形成的.

开始标签,结束标签,元素内容三部分组成的整体,称为JSP元素.

JSP元素分为三种类型,基本元素,指令元素,动作元素.

1,基本元素

(1)JSP注释

- 格式:<%-- ... --%> 当服务器把jsp编译成.java文件时,已经忽略了注释部分,也就是说jsp的注释是不会被发送到客户端浏览器上的
- HTML的注释会被编译进.java文件中,会被发送到客户端浏览器上,只不过不显示出来. ###

(2)JSP声明 在JSP页面中可以声明变量和方法,声明后的变量和方法可以在本JSP页面内的任何位置使用.并在JSP页面初始化时被初始化.

```
<!-- 语法格式 -->
<%!
  声明变量,方法,类
%>
<%!
  int i = 0;
  public void fun(){

  }
  class Fun{
    public Fun(){
        //TODO 构造方法
    }
  }
%>
```

(3)JSP表达式

JSP表达式是由变量,常量组成的算式,它将JSP生成的数值转换成字符串嵌入HTML页面,直接显示出来.

```
html
<!-- 语法格式 -->
<%=表达式内容%>
<p>i的值为:<%=i%></p>
```

(4)JSP代码块

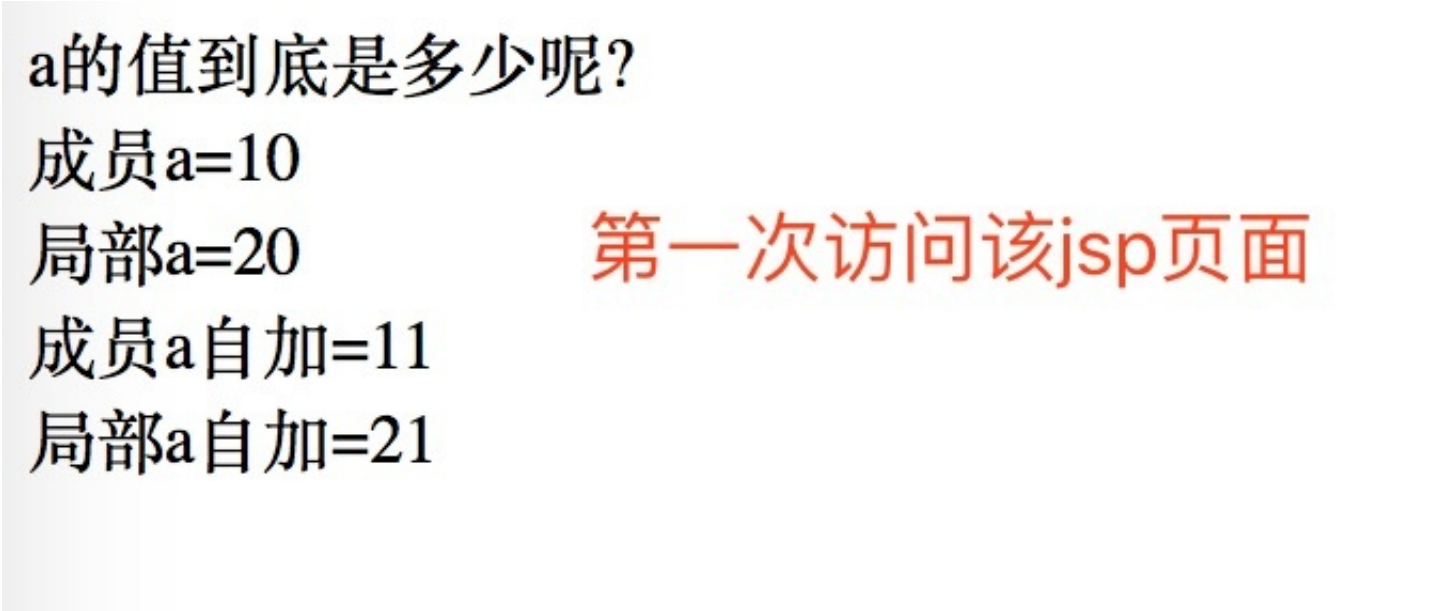
JSP代码块可以包含任意合法的Java语句.

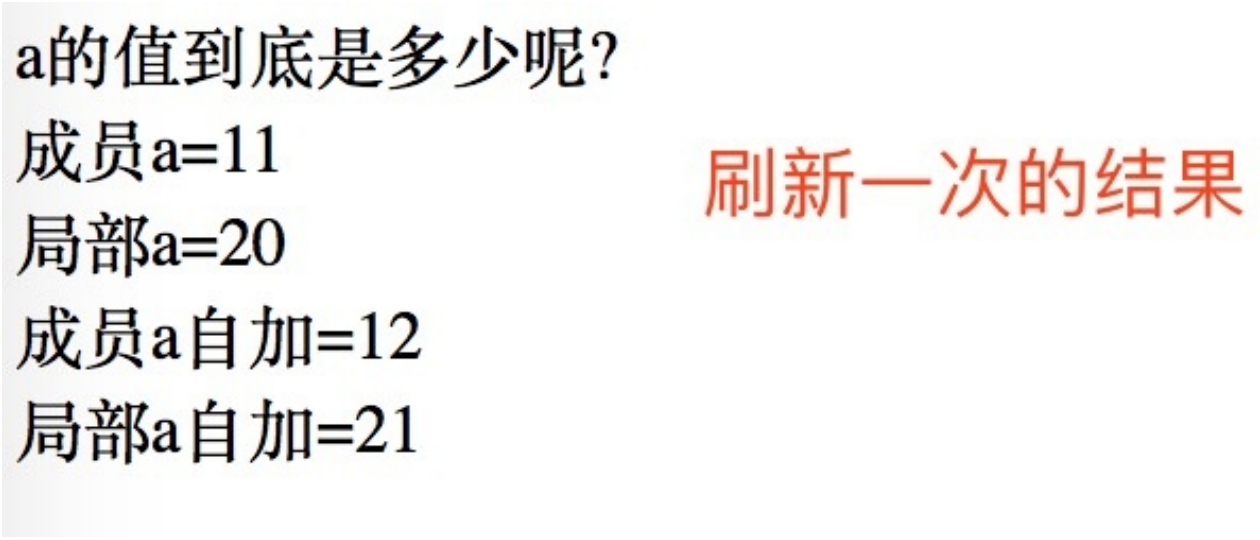
```
jsp
<% 合法的Java代码%>
```

(5)关于成员变量与局部变量

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>演示成员变量与局部变量</title>
</head>
<body>
<%!
    int a = 10;
%>
<%
    int a = 20;
%>
a的值到底是多少呢?<br/>
成员a=<%=this.a%><br/>
局部a=<%=a%><br/>
成员a自加=<%=++this.a%><br/>
局部a自加=<%=++a%>
</body>
</html>
```

结果:





结论:

- 如果不使用this指代,那么使用的就是局部变量.

- 多次访问同一个jsp页面,也就是在操作同一个对象,所以做自加操作的话该对象的成员变量才会一次次自加.而局部变量不会.

2,指令元素

JSP指令是被服务器解析并执行的,通过指令元素可以使服务器按照指令的设置执行动作或设置在整个JSP页面范围有效的属性.

JSP指令不直接产生可见输出.

指令的分类:page指令,include指令,taglib指令.

JSP指令的语法格式:

```
<%@ 指令名称 属性="属性值1" 属性2="属性值2" ...%>
```

(1)page指令

page指令用来定义JSP页面的全局属性,它描述了与页面相关的一些信息,其作用域为它所在的JSP文件页面.

page指令的语法规则:

```jsp
<%@page
    language="脚本语言"//(基本没用,指定jsp编译后的语言类型,只能编译java,默认为java)
    extends="继承的父类名称"//让jsp生产的Servlet去继承该属性指定的类,(基本不用)
    import="导入的Java包或类的名称"
    session="true/false"//当前页面是否支持session,如果为false,那么当前页面就没有session这个内置对象
    buffer="none/8kb/自定义缓冲区大小"//指定缓冲区大小,默认为8kb,通常不需要修改
    autoFlush="true/false"//指定,当jsp输出流缓冲区满时,是否自动刷新(要输出的东西多了,是不是自动刷新),默认为true,如果为false,那么在缓冲区满时抛出异常.
    isThreadSafe="true/false"//是否支持并发访问,默认为false(没用)
    isELIgnored="true/false"//是否忽略el表达式,默认值为false,不忽略,老版本的jsp默认是true(基本看不到了),现在已经改成false了
    info="页面信息"//(没用)
    errorPage="当前页面若是发生错误,要转向哪个页面,就由这个属性来指定(相对路径,不包含项目名)(状态码为200)"
    isErrorPage="true/false"//它指定当前页面是否为处理错误的页面(状态码为500)(只有设置这个属性为true时,才能使用九大内置对象中的exception)
    pageEncoding="页面编码类型"//当Tomcat要把jsp编译成.java时,需要使用pageEncoding属性
    contentType="text/html; ISO-8859-1" // 添加一个contentType="MIME类型和字符集"响应头
    //等同于response.setContentType="text/html;ISO-8859-1"
    //页面编码类型pageEncoding和contentType响应头,如果只设置一个为utf-8,那么另一个会默认也被设置未utf-8
    //如果都不设置,则默认为iso
%>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <!-- 在web.html页面中设置error-page标签 -->
    <error-page>
      <!-- 如果错误信息为404,则跳转到error404.jsp页面 -->
        <error-code>404</error-code>
        <location>/jsps/error404.jsp</location>
    </error-page>

    <error-page>
        <error-code>500</error-code>
        <location>/jsps/error500.jsp</location>
    </error-page>

    <error-page>
      <!-- 如果抛出空指针异常,则跳转到errorDemo.jsp页面 -->
```

```
        <exception-type>java.lang.NullPointerException</exception-type>
        <location>/jsps/errorDemo.jsp</location>
    </error-page>

</web-app>
```

注:
- 一个页面可以有多个<%@page%>指令来分别描述属性.
- 在一个<%@page%>指令中,除了import属性可以被使用多次外,其他属性只能被使用一次.
- <%@page%>指令区别大小写的.

常用的使用格式:

```
<%@page
  language="java"
  <!-- 设置的是请求,服务器返回的HTML页面代码的编码 -->
  contentType="text/html;charset=utf-8"
  <!-- 设置JSP页面本身的编码 -->
  pageEncoding="utf-8"
  <!-- 当JSP页面中使用了某些java类库时需要设置下面的import属性 -->
  <!-- 一般都会采用MVC设计模式,也就不需要使用import属性 -->
  import="java.util.*"
%>
```

(2)include指令(静态包含)

include指令称为文件加载指令,可以将其他的文件插入JSP网页,被插入的文件可以是JSP文件,HTML文件或者其他文本文件,必须保证插入后形成的新文件符合JSP页面的语法规则.

- 与RequestDispatcher的include()方法的功能相似!区别在于包含的时间点不同.
- <%@include%>是在jsp编译成java文件时完成的--->包含的文件与当前文件共同生成一个java(一个Servlet)文件,然后再生成一个class.
- RequestDispatcher.include()是一个方法,包含和被保护的是两个Servlet,即两个.class,他们只是把响应的内容在运行时合并了.
- 作用:将页面分解了,使用包含的方式组合在一起,这样一个页面中不变的部分,就可以作为一个独立的jsp存在,而我们只需要处理变化的变量.实现可重用.

include指令语法规则:

```
Java
<%@include file="文件名"%>
```

注: 被插入的文件必须与当前JSP页面在同一Web服务目录下.

(3)taglib指令

taglib指令用来定义一个标记库以及标记的前缀,不要用jsp,jspx,java,javax,servlet,sum和sunw为自定义标签的前缀.

taglib指令语法规则:

```Java
<%@taglib uri="指定标签库所在位置" prefix="标记前缀" %>
<%@taglib uri="/libs/commons-dd" prefix="d" %>
// 当使用commons-dd库中的text标签时的写法
// 这就是d标记前缀的作用,处理名字冲突,如果多个标签库中都有text标签,就可以通过标记前缀区分
<d:text>
```

使用JSTL核心标签库时会常用.

3,动作标签

动作标签由服务器(Tomcat)执行,在服务器端执行,一共有二十个,很少用.

html标签由浏览器来执行.
JSP动作元素是用来控制JSP引擎的行为,JSP标准动作标签均以"jsp"为前缀,主要有如下6个动作元素.

- jsp:include:在页面得到请求时动态包含一个文件.
- jsp:forward:引导请求进入新的页面(转向到新页面)(请求转发)
- jsp:plugin:连接客户端的Applet或Bean插件
- jsp:useBean:应用JavaBean组件
- jsp:setProperty:设置JavaBean的属性值
- jsp:getProperty:获取JavaBean的属性值并输出

(1)jsp:include动作

语法格式:

```java
<jsp:include page="文件的名字"/>
```

功能:
- 当前JSP页面动态包含一个文件,即将当前JSP页面,被包含的文件各自独立编译为字节码文件.
- 执行到该动作标签处,才加载执行被包含文件的字节码.

(2)jsp:forward动作

语法格式:

```
java
<jsp:forward page="文件的名字"/>
```

功能:

- 该动作用于停止当前页面的执行,转向另一个HTML或JSP页面
- 该动作用于控制界面间的前后跳转,跳转时携带着内置对象Request中的信息

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>动态转发</title>
</head>
<body>
<h1>a.jsp</h1>
<%--动态转发--%>
<jsp:forward page="b.jsp">
    <%--也可以携带参数--%>
    <jsp:param name="user" value="zz"/>
    <jsp:param name="password" value="123"/>
</jsp:forward>
</body>
</html>
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>b</title>
</head>
<body>
<h1>b.jsp</h1>
<%
    String user = request.getParameter("user");
    String password = request.getParameter("password");
```

```
    out.print(user+"--"+password);
%>
</body>
</html>
```

六,JSP内置对象

九大内置对象

主要是为了方便页面之间的数据共享,JSP专门设置了九个内置对象,这些对象不需要创建,在写代码的时候直接使用就ok.

对象名称	所属类型	有效范围	说　明
application	javax.servlet.ServletContext	application	代表应用程序上下文,允许JSP页面与包括在同一应用程序中的任何Web组件共享信息调用该对象的setAttribute()设置的属性在项目的任何地方都能get到.
config	javax.servlet,.ServletConfig	page	允许将初始化数据传递给一个JSP页面,每个JSP页面都有自己的config对象
exception	java.lang.Throwable	page	该对象含有只能由指定的"JSP错误处理页面"访问的异常数据. 就是说该对象中存放的时候,只能由带有处理错误页面功能的JSP页面访问.
out	javax.servlet.jsp.JspWriter	page	提供对象输出流的访问
page	javax.servlet.jsp.HttpJspPage	page	代表JSP页面对应的Servlet实例
pageContext	javax.servlet.jsp.PageContext	page	是JSP页面本身的上下文,它提供了唯一一组方法来管理具有不同作用域的属性
request	javax.servlet.http.HttpServletResquest	request	提供对请求数据的访问,同时还提供用于加入特定请求数据的上下文
response	javax.servlet.http.HttpServletResponse	page	该对象用来向客户端传输数据
session	javax.servlet.http.HttpSession	session	用来保存在服务器与一个客户之间需要保存的数据,当客户端关闭网站的所有网页时,session变量会自动消失

session对象

用户在浏览某个网站时,从进入网站到浏览器关闭所经过的这一段时间成为一次会话.

session对象的生命周期就是一次会话的过程,session对象保存的信息,可以在该会话过程中的不同页面之间共享.

一般用来保存用户名和密码,在登录后的后续页面中,直接判断session中的内容即可.

pageContext对象

JSP中有四大域:

- servletContext(application)
- 整个应用程序
- session
- 整个会话(一个会话只有一个用户)
- request
- 一个请求链(可能有多个Servlet)
- pageContext
- 当前的jsp页面,这个域是在当前jsp页面和当前jsp页面中使用的标签之间共享数据(当前jsp页面与当前jsp页面中使用的标签之前就属于跨域的关系)
- 域对象,可以代理其他域:
- pageContext.setAttribute("xxx","XXX",PageContext.SESSION_SCOPE);将属性保存到了session域中
- 全域查找:pageContext.findAttribute("xxx");该方法会在四个域中都查找,从小到大依次查找.(重要)
- 所以在设置属性名时可以给一个前缀区分:session_xxx,request_xxx等.
- 获取其它八个内置对象:pageContext.getXXX();即可

JSP02

一,Http协议与Cookie

Cookie是服务器保存到客户端的.由服务器创建,保存到客户端.当客户端又一次访问服务器时,会将上次请求得到的Cookie再发送给服务器

Cookie是由服务器创建保存到客户端浏览器的一个键值对.

服务器保存Cookie的响应头:Set-Cookie: aaa=AAA Set-Cookie: bbb=BBB

- response.addHeader("Set-Cookie","aaa=AAA");

- response.addHeader("Set-Cookie","bbb=BBB");
当浏览器请求服务器时,会将该服务器保存的Cookie随着请求发送给服务器.

浏览器归还Cookie的请求头:Cookie:aaa=AAA;bbb=BBB

- 不会直接通过请求头获取Cookie的键和值(麻烦),有专门的类可以实现.

Http协议规定:
- 一个Cookie最大4kb
- 一个服务器最多向一个浏览器保存20个Cookie
- 一个浏览器最多可以保存300个Cookie

有些浏览器会为了竞争,违反Http规定,但是也不会让一个Cookie占用太大的空间.

二,Cookie的用途

- 服务器使用Cookie来跟踪客户端状态
- 保存购物车(购物车中的商品不能使用request保存,因为它是一个用户向服务器发送的多个请求信息)
- 显示上次登录名(也是一个用户多个请求)

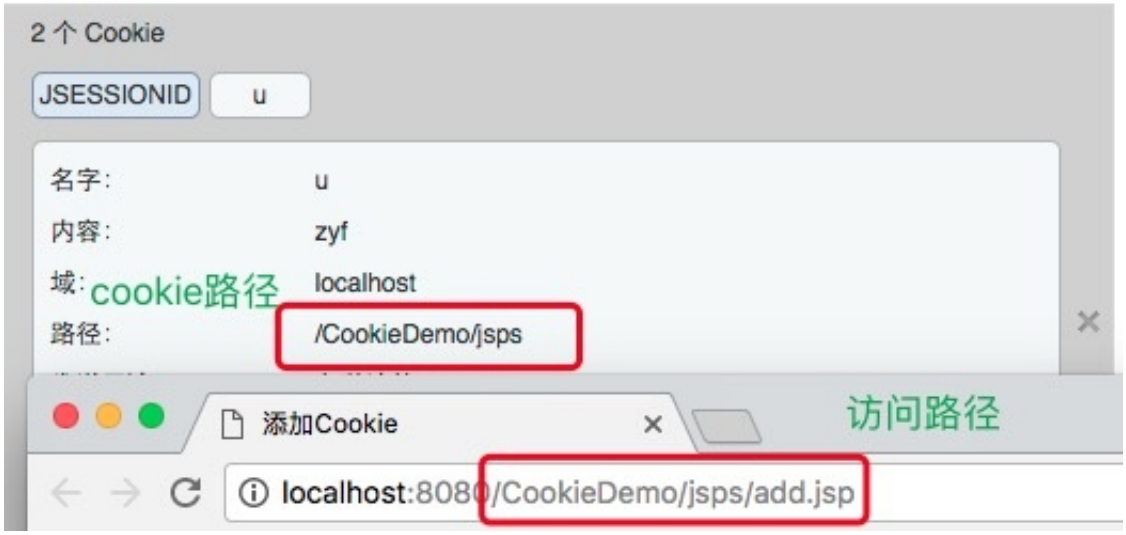
注: Cookie是不能跨浏览器的!

三,JavaWeb中使用Cookie

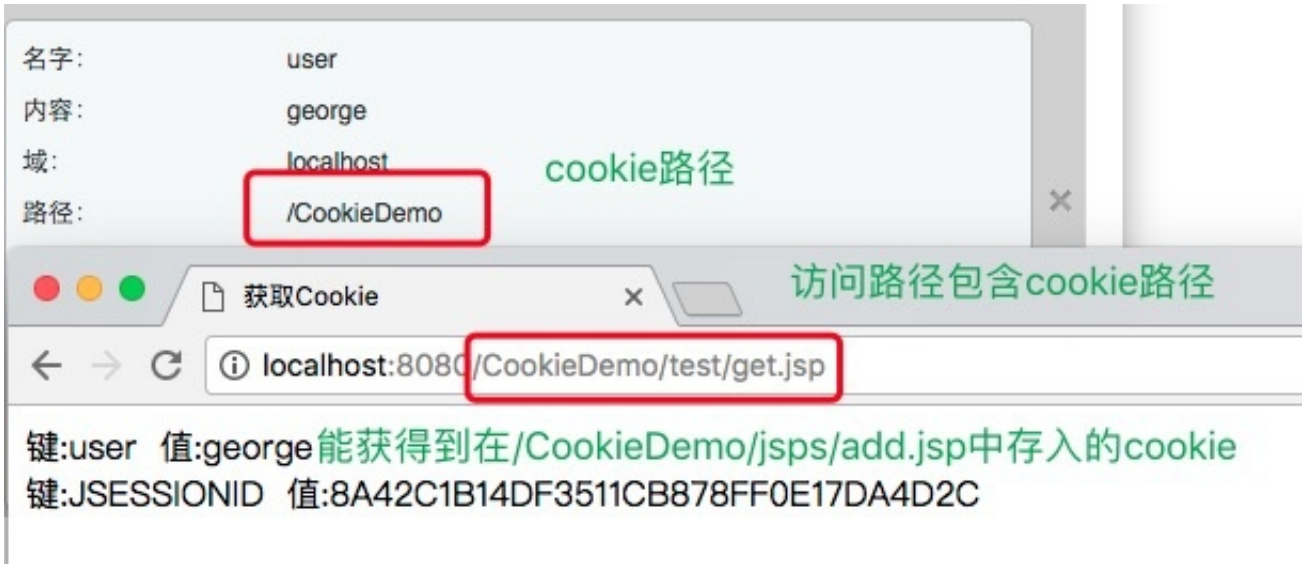
- 原始方式:
  - 使用response发送Set-Cookie响应头
  - 使用request获取Cookie请求头
- 便捷方式:
  - 使用response.addCookie()方法向浏览器保存Cookie
  - 使用request.getCookies()方法获取浏览器归还的Cookie

四,Cookie相关

- Cookie不只有name和value两个属性

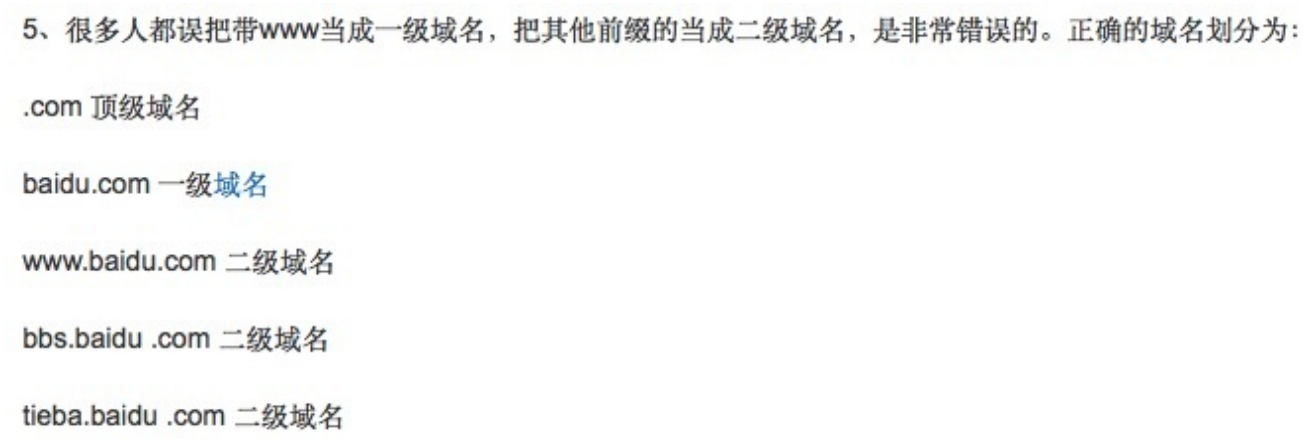
- Cookie的maxAge:Cookie可保存的最大时长.以秒为单位.
  - maxAge>0:浏览器会将Cookie保存到客户机硬盘上,有效时长为maxAge的值决定.
  - maxAge<0:浏览器只将Cookie保存到浏览器内存中,当用户关闭浏览器时,浏览器进程结束,Cookie被销毁.
  - maxAge=0:浏览器会直接删除该Cookie(可用来删除原有Cookie)
  - 如果不设置maxAge属性,则Cookie默认会保存到当次会话结束时(关闭浏览器时);
- 关于Cookie的path:
  - 首先要明确Cookie的path并不是设置这个Cookie在客户端的保存路径.
  - Cookie的path由服务器创建Cookie时设置.
  - 一个服务器的路径有很多,这个服务器保存到客户端的Cookie也不唯一,那么当浏览器访问某一个路径时,要上传哪一个Cookie就是由Cookie的path决定的.
  - 浏览器访问服务器的路径,如果包含某个Cookie的路径,那么就会归还这个Cookie
  - 

  - 



- 例如:(可打开浏览器Cookie观察)
- aCookie.path=/DemoProject/;bCookie.path=/DemoProject/jsps/;cCookie.path=/DemoPro
- 访问:/DemoProject/index.jsp时,归还:aCookie
- 访问:/DemoProject/jsps/demo.jsp时,归还:aCookie,bCookie
- 访问:/DemoProject/jsps/cookie/test.jsp时,归还:aCookie,bCookie,cCookie
- Cookie的path默认值:当前访问路径的父路径.
- 在访问/DemoProject/jsps/demo.jsp时,响应的cookie的path就为:/DemoProject/jsps

- Cookie的domain
  - domain用来指定Cookie的域名! 当多个二级域中共享Cookie时才有用.
- 什么是二级域名:



  - 上述二级域名之间默认是不能共享Cookie的,可以通过设置domain来共享.

    - 设置:cookie.setDomain(".baidu.come");
    - 设置path为:cookie.setPath("/");(必须为斜杠,不能带项目名)
    - 了解一下就行.

一,HttpSession

- HttpSession是由JavaWeb提供的,用来会话跟踪的类. **Session是服务器端对象,保存在服务器端!**
- HttpSession是Servlet三大域对象之一,涉及到域对象,所以也有setAttribute(),getAttribute(),removeAttribute()方法.
    - request域
    - session域
    - application域
- HttpSession底层依赖Cookie,或者是URL重写.

二,HttpSession的作用

- 会话范围:会话范围是某个用户从首次访问服务器开始,到该用户关闭浏览器结束.
    - 会话:用户在浏览某个网站时,从进入网站到浏览器关闭所经过的这段时间成为一次会话.
- 服务器会为每个客户端创建一个session对象,session主要用于解决单个客户端访问同一网站的不同页面资源时信息的传递与控制.
- session被保存到服务器的一个Map中,这个Map被称为session缓存.
    - Servlet中获得session对象:Session session = resquest.getSession();
    - JSP中得到session对象:session是JSP的内置对象,可以直接使用.
- session域的相关方法:

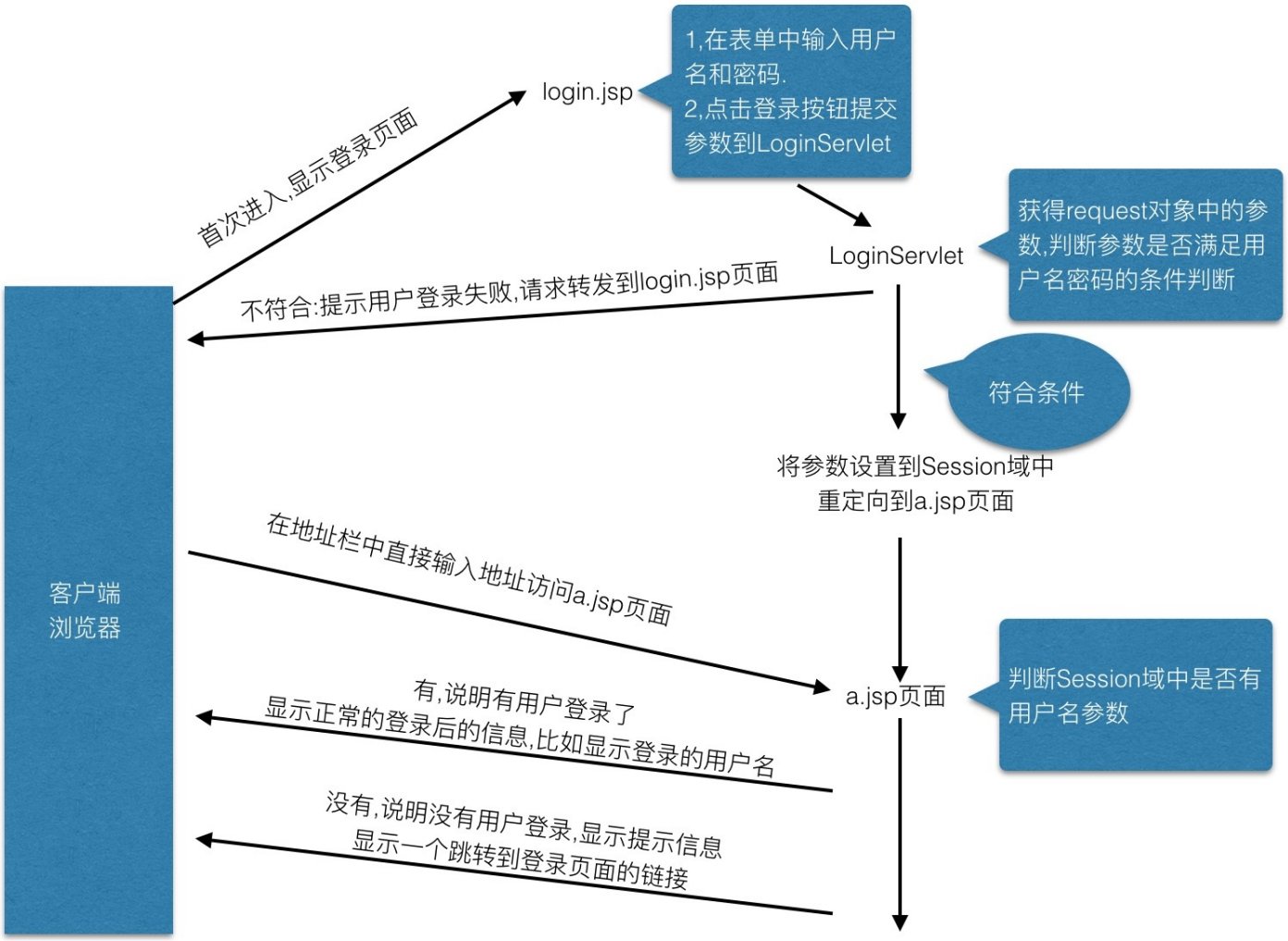
三,示例

1,单次会话中,不同Servlet之间的数据共享

- 访问AServlet存储数据.
- 访问BServlet获得数据.

2,用户登录状态的保存示例(Session的重要用途)

Session应用,保存用户登录信息



四,HttpSession原理

- 服务器不会直接创建session对象,而是在第一次执行request.getSession方法获取时,才会创建session对象
- request.getSession()方法:
  - 获取Cookie中的JSESSIONID(如果Cookie中无JSESSIONID,那么会去获取url中是否有该JSESSIONID参数==>URL重写的基本支持):
  - 如果sessionId不存在,创建session,把session保存起来,把新建的sessionId保存到Cookie中.
  - 如果sessionId存在,通过sessionId查找到session对象,如果没有查找到,创建session,把session保存起来,把新建的sessionId保存到Cookie中.
  - 如果sessionId存在,通过sessionId查找到了session对象,那么就不会再创建session了.

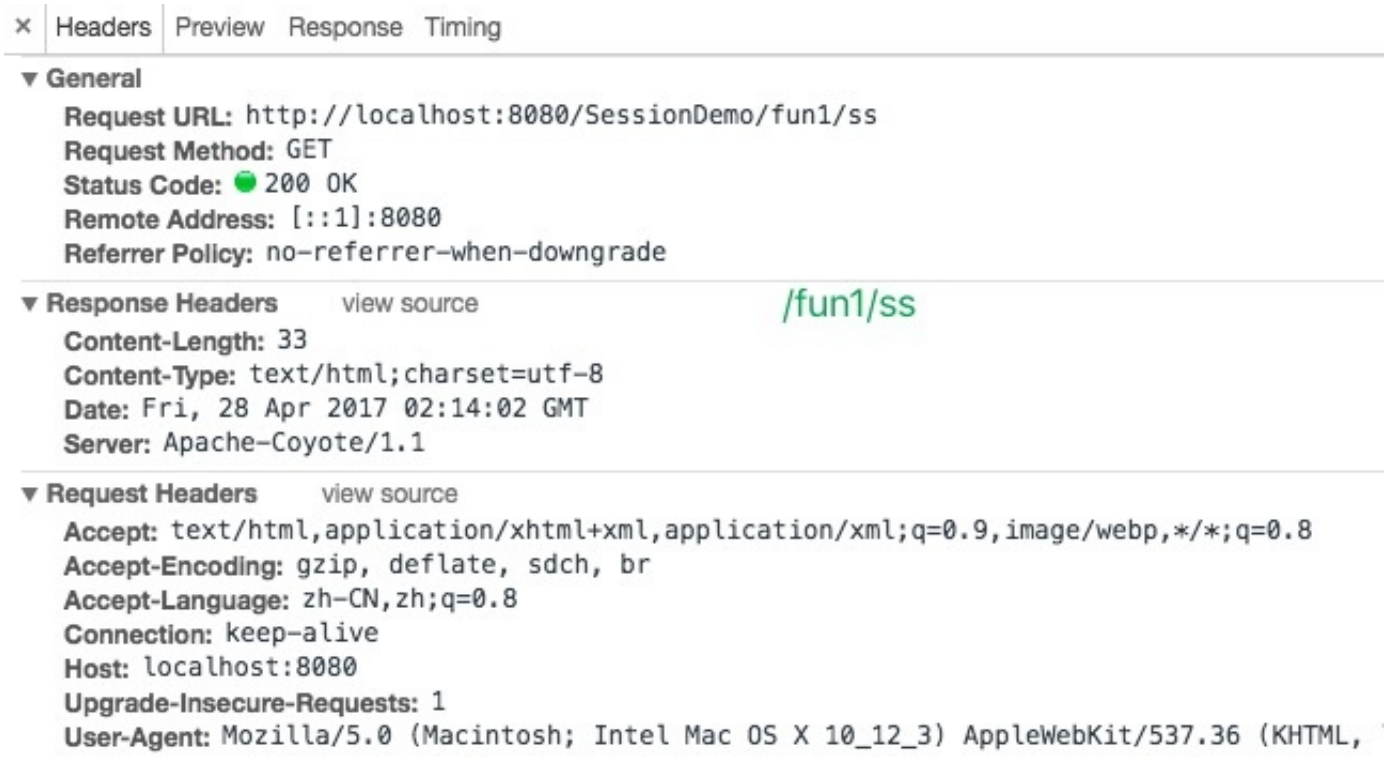
- 找到了session后,返回session对象
- 如果创建了新的session,浏览器会得到一个包含了sessionId的Cookie,这个Cookie的生命为-1,即只在浏览器内存中存在, 如果不关闭浏览器,那么Cookie就一直存在.
- 下次请求时,再次执行request.getSession()方法,因为可以通过Cookie中的sessionId找到对应的session对象,所以与上一处请求使用的是同一个session对象.
- 这也就是一次会话中,多次请求明明已经是不同的request对象了,还能找到相同的session对象的原因.

```
package fun1;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * Created by zyf on 2017/4/28.
 */
@WebServlet(name = "SessionServlet",urlPatterns = "/fun1/ss")
public class SessionServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse respons
e) throws ServletException, IOException {

    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response
) throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        response.getWriter().print("<h1>查看是否有sessionId</h1>");
    }
}
```

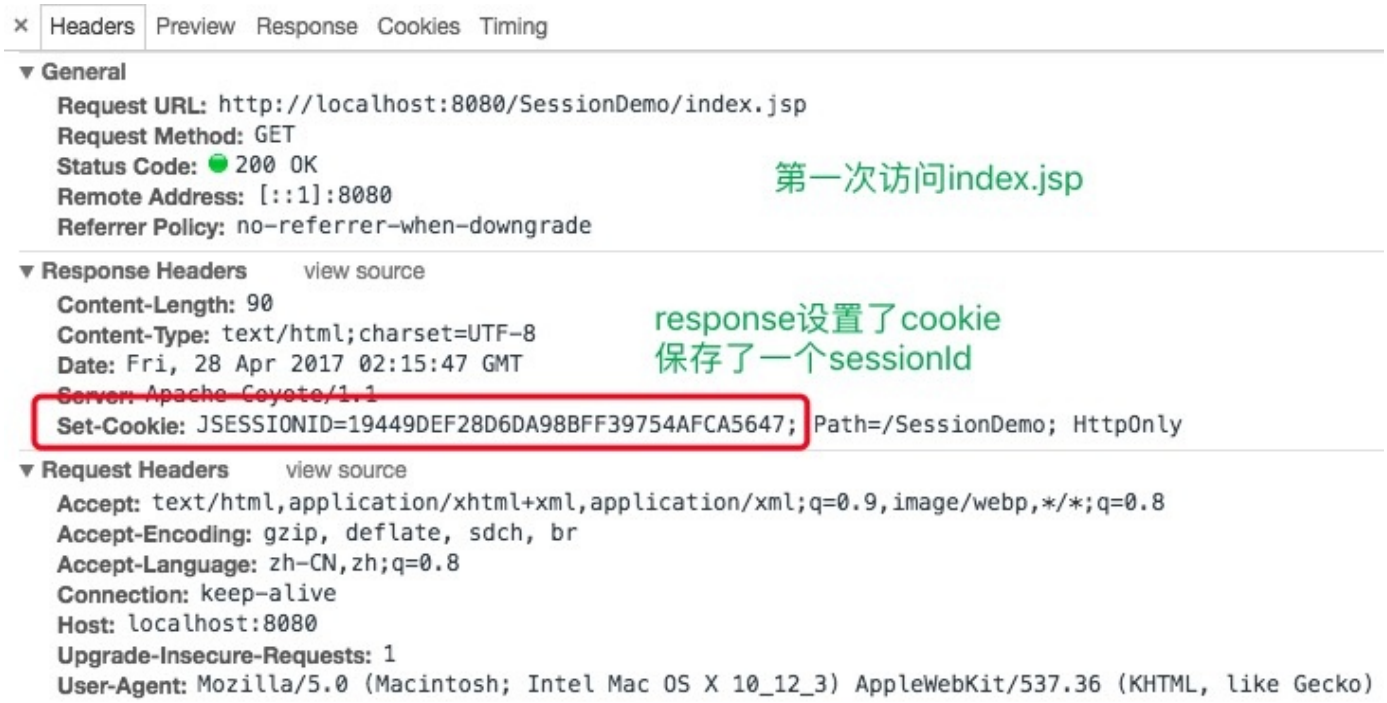
```
<%--
  Created by IntelliJ IDEA.
  User: zyf
  Date: 2017/4/27
  Time: 上午11:08
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
```

```
    <title>$Title$</title>
  </head>
  <body>
  $END$
  </body>
</html>
```

- 第一次访问/fun1/ss(Servlet)

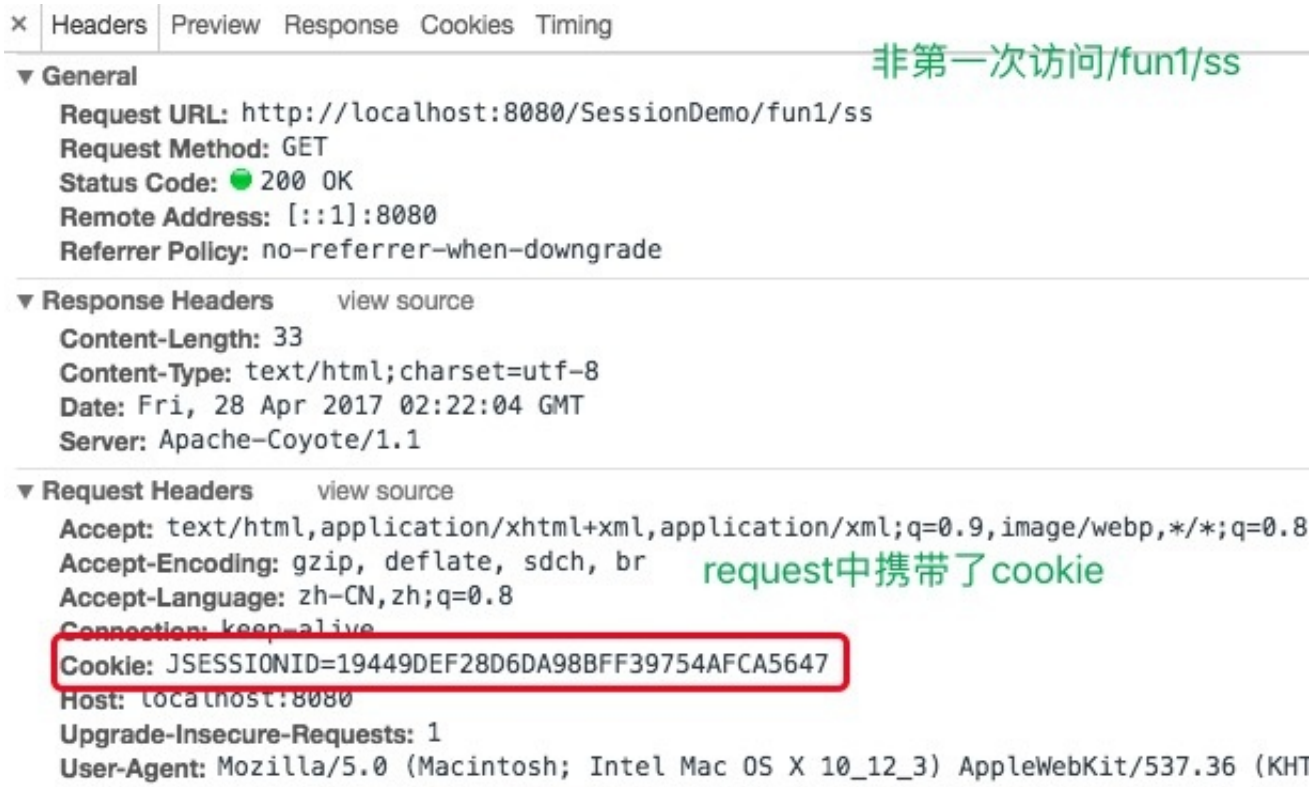


- 第一次访问index.jsp



- 非第一次访问index.jsp

- 非第一次访问/fun1/ss



- 结论:
  - 访问jsp页面时,jsp会自动使用Cookie,也就自动创建了一个携带SessionId的Cookie
  - 访问Servlet时,如果没有调用过request.getSession()方法,则不会创建携带SessionId的Cookie
  - 如果没有调用过request.getSession()方法,即使在Servlet中,通过response.addCookie方法保存了Cookie,也不会保存携带SessionId的Cookie
- 关于request.getSession()和request.getSession(boolean var)
  - request.getSession(true)和request.getSession()效果一样
  - request.getSession(false):如果session缓存中不存在session(存在Cookie不存session或不存在该Cookie),那么返回null,不再创建.

五,session的部分方法

方法	说明
Object getAttribute(String attriname)	用于获取与指定名字相练习的属性,如果属性不存在,将返回null
void setAttriburte(String name,Object value)	用于设定指定名字的属性值,并且把它存储在session对象中
void removeAttribute(String attriname)	用于删除指定的属性(包含属性名,属性值)
Enumeration getAttributeNames()	用于返回session对象中存储的每一个属性对象,返回值是一个Enumeration的对象
long getCreationTime()	得到session对象被创建的时间,单位为毫秒
long getLstAccessedTime()	得到session最后发送请求的时间,单位为毫秒
String getId()	获取sessionId,底层使用UUID生成随机id(32位长度)
int getMaxInactiveInterval(int var)	设置session可以的最大不活动时间(秒),默认为30分钟.当session在30分钟内没有使用,那么Tomcat会在session池中移除该对象.
void setMaxInactiveInterval()	设置最大不活动时间
boolean isNew()	用于判断request.getSession()是在创建session后获得(第一次调用)还是在获得session(非第一次调用)
void invalidate()	销毁session,调用该方法后,再次调用request.getSession时获得的就是一个新的Session对象了

六,URL重写

- session依赖Cookie,目的是让客户端发出请求时归还sessionId,这样才能找到对应的session
- 如果客户端禁用了Cookie,那么就无法得到sessionId,session也就无法使用了
- 也可以使用URL重写来替代Cookie
  - 让网站的所有超链接,表单中都添加一个特殊的请求参数,即jsessionId
  - 这样服务器可以通过获取请求参数得到jsessionId,从而找到session对象
- response.encodeURL(String url)
  - 该方法会对url进行智能的重写:
  - 当请求中没有归还sessionId这个cookie,那么该方法会重写url,否则不重写,当然url必须是指向本站的url

代码示例

```
<%--
  Created by IntelliJ IDEA.
  User: zyf
  Date: 2017/4/28
  Time: 下午2:19
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
```

```
<head>
    <title>URL重写session</title>
</head>
<body>
<%--如果客户端禁用Cookie,那么就无法用Cookie保存JSESSIONID,
也就无法通过JSESSIONID来获得session对象--%>
<%--可以在每一个超链接的后面,增加一个JSSEIONID的参数,来向服务器提交sessionId--%>
<%--连接JSESSIONID参数,必须用分号;--%>
<a href="/SessionDemo/fun1/as;JSESSIONID=<%=session.getId()%>">点击这里到AServlet</a
>
<a href="/SessionDemo/fun1/as;JSESSIONID=<%=session.getId()%>">点击这里到AServlet</a
>
<a href="/SessionDemo/fun1/as;JSESSIONID=<%=session.getId()%>">点击这里到AServlet</a
<br/>
<%
    //会查看cookie是否存在,如果不存在,在指定的url后添加JSESSIONID参数
    //response.encodeURL()方法:如果cookie存在,不会在url后添加任何东西
    //也就是说,第一次访问此页面,无cookie,所以会在/SessionDemo/fun1/as后面拼接/sessionID
    //第二次访问此页面,cookie已经存在了,所以不会在url后面拼接sessionID了
    out.print(response.encodeURL("/SessionDemo/fun1/as"));
%>

</body>
</html>
```

一,JavaBean概述

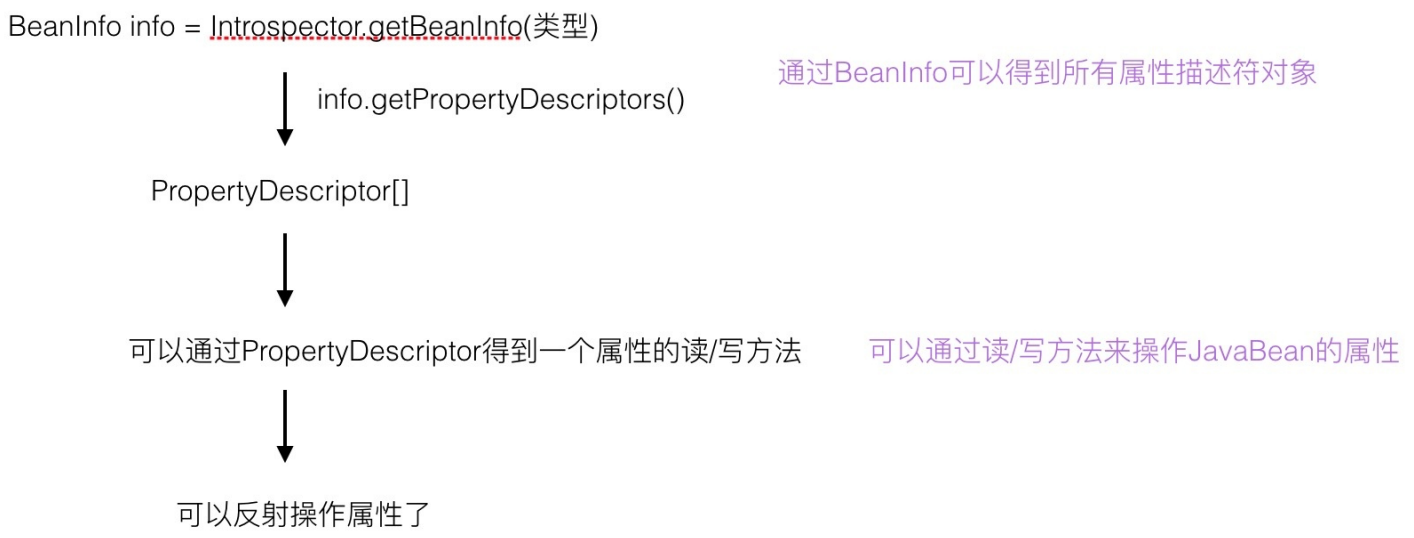
JavaBean是在编写类时,对类的某些要求,是一种规范.

JavaBean规范:
- 1,必须要有一个空参的默认构造方法.
- 2,提供get/set方法,如果只有get方法,那么这个属性为只读属性.
- 3,属性:有get/set方法的成员变量,即使没有成员变量,但是有get/set方法,那么也是一个属性,属性名由get/set方法名决定.
- 4,方法名称满足一定的规范,那么它就是属性. boolean类型的属性,它的读方法,可以是is开头,也可以是get开头.

二,Introspector(内省)

内省就是通过反射来操作JavaBean,只是比使用反射方便一些.

我们需要提供JavaBean类.



通过反射,得到对应类所有的属性,公共的方法和事件的过程,称为内省.

内省底层是依赖反射的.

```
BeanInfo bi = Introspector.getBeanInfo(xxx.class);

//所有满足了JavaBean规范的方法们,提取出来的一大堆属性(getName,setName方法等,即使没有name这个
//通过属性描述符对象,可以获得对应属性的get/set方法,也就可以通过得到的方法去修改或获取这个属性的
//PropertyDescriptors类就是属性描述符类
```

```
PropertyDescriptor[] ds = bi.getPropertyDescriptors();
```

三,commons-beanutils工具类

依赖内省完成.在项目中就使用它来完成上述的操作.

- 导包:
- commons-beanutils.jar
- commons-logging.jar

```
@Test
    public void fun2() throws IllegalAccessException, InstantiationException, ClassNot
        String classPath = "domain.Student";
        Class clazz = Class.forName(classPath);
        Object obj = clazz.newInstance();

        //JavaBean的操作就使用BeanUtils类即可
        BeanUtils.setProperty(obj,"name","张三");
        //age是int类型,这里会自动转换的
        BeanUtils.setProperty(obj,"age","28");
        BeanUtils.setProperty(obj,"gender","男");

        //找不到xxx属性,也就无法赋值,但是不会抛出异常
        BeanUtils.setProperty(obj,"xxx","XXX");

        System.out.println(obj);

        String ageS = BeanUtils.getProperty(obj, "age");
        int age = Integer.decode(ageS);
        System.out.println(age+"");
    }

    /**
     * 把map中的数据直接封装到一个Bean中
     *
     *
     * Map:{"username":"张三","password":"123}
     * 要把map的数据封装到一个JavaBean中,要求map的key与JavaBean的属性名一致
     */
    @Test
    public void fun3() throws InvocationTargetException, IllegalAccessException {
```

```
        Map<String,String> map = new HashMap<>();

        map.put("user","张三");
        map.put("password","123");

        User user = new User();
        //这个方法,就会将map中的数据直接装进user对象中
        BeanUtils.populate(user,map);

    }
```

四,jsp中与JavaBean相关的标签:(基本不用)

- jsp:useBean: 查询bean如果查询不到会创建
  -
  - bean的变量名,bean的类型,bean所在的域
- jsp:setProperty:
  -
  - 要设置的属性名,被设置的bean对象名,要设置的值
- jsp:getProperty:
  -
  - 要获得的用户名,bean对象名

一,EL表达式概述

EL是JSP内置的表达式语言.在jsp2.0后,不让再使用java脚本,而是使用el表达式和动态标签来替代java脚本.

EL替代的是<%=...%>,也就是只能作为输出.

二,简单使用

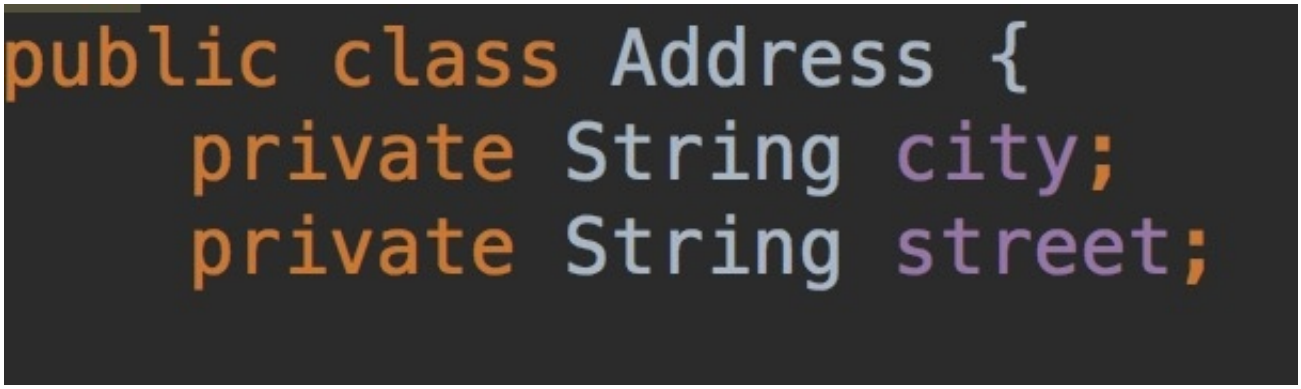
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
```

```
    <head>
      <title>$Title$</title>
    </head>
    <body>
    <%
      pageContext.setAttribute("demo","pageContext");
      request.setAttribute("demo","request");
      session.setAttribute("demo","session");
      application.setAttribute("demo","application");
    %>
    <%--当这个属性不存在时,不会显示null,会输出空字符串--%>
    ${xx}
    <%--同名的话,优先级顺序:从范围小的到范围大的--%>
    <%--相当于全域查找--%>
    ${demo}
    <!-- 获取指定域属性 -->
    ${pageScope.demo}
    ${requestScope.demo}
    ${sessionScope.demo}
    ${applicationScope.demo}
    </body>
</html>
```

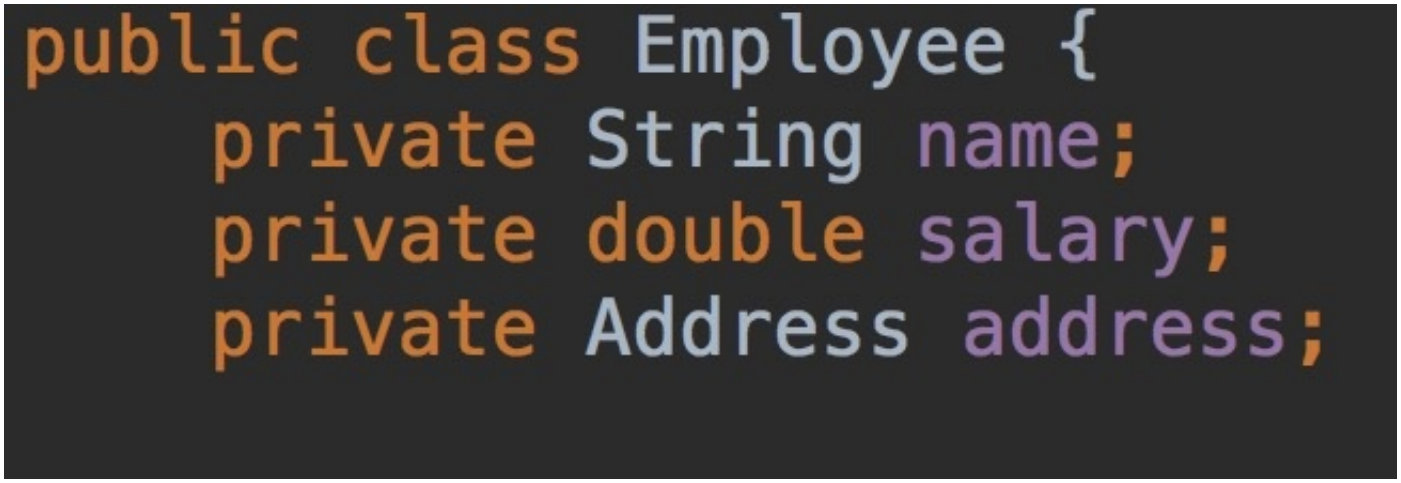
三,JavaBean导航

建立两个JavaBean类:Address和Employee.

Address类:



Employee类:




```html
<%@ page import="domain.Address" %>
<%@ page import="domain.Employee" %>

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




<%
Address address = new Address();
address.setCity("北京");
address.setStreet("兴工街");
Employee employee = new Employee();
employee.setAddress(address);
employee.setName("张三");
employee.setSalary(130.23);

    request.setAttribute("emp",employee);

%>
```

使用el获取request域的emp

<%--JavaBean导航,就是一顿.一个个属性--%>
<%--等同于在使用get方法获取--%>
${requestScope.emp.address.street}

${emp.gender}

```

# 四,EL运算符

EL运算符

| 运算符 | 说明 | 示例 | 结果 |
| --- | --- | --- | --- |
| + | 加 | ${100+5} | 105 |
| - | 减 | ${100-5} | 95 |
| * | 乘 | ${100*5} | 500 |
| /或div | 除 | ${100/5}或{$100 div 5} | 20 |
| %或mod | 取余 | ${100%5}或{$100 mod 5} | 0 |
| ==或eq | 等于 | ${5 == 5}或${5 eq 5} | TRUE |
| !=或ne | 不等于 | ${5 != 5}或${5 ne 5} | FALSE |
| <或lt | 小于 | ${98<100}或${98 lt 100} | TRUE |
| >或gt | 大于 | ${100 > 98}或${100 gt 98} | TRUE |
| <=或le | 小于等于 | ${100 <= 98}或${100 le 98} | FALSE |
| >=或ge | 大于等于 | ${100 >= 98}或${100 ge 98} | TRUE |
| &&或and | 并且 | ${true && false}或${true and false} | FALSE |
| !或not | 非 | ${!true}或${not true} | FALSE |
| ll或or | 或者 | ${true ll false}或${true or false} | TRUE |
| empty | 是否为空 | ${empty “”}<br>可以判断字符串,数据,集合的长度是否为0,为0返回true.<br>empty还可以与not或!一起使用.<br>${not empty “”} | true<br><br><br>false |

# 五,EL内置对象

十一个内置对象,其中十个是Map,只有pageContext不是Map,是PageContext类型.
```

- 参数相关:

- param:
- paramValues:
- 请求头相关:
- header:
- headerValues:
- 全局参数:
- initparam:
- Cookie相关:
- cookie:

十一个内置对象

内置对象名称	类型	用处	说明
pageScope	Map<String,String>		各种域
requestScope	Map<String,String>		
sessionScope	Map<String,String>		
applicationScope	Map<String,String>		
param	Map<String,String>	获得请求参数,key是参数名,value是参数值(单值)	参数相关
paramValues	Map<String,String[]>	获得请求参数,当一个参数名对应多个参数值时使用(多值)	
header	Map<String,String>	对应请求头,key头名称,value是头值(单值)	请求头相关
headerValues	Map<String,String[]>	对应请求头,一个头名称对应多个值(多值)	
initParam	Map<String,String>	获取在web.xml中配置的全局初始化参数:<context-param>内的参数 <context-param> <param-name>name</param-name> <param-value>value</param-value> </context-param> <context-param> <param-name>a</param-name> <param-value>A</param-value> </context-param>	全局初始化参数相关
cookie	Map<String,Cookie>	注意value的类型是Cookie. key:Cookie的name value:Cookie对象	cookie相关
pageContext	PageContext	可以通过该对象获得jsp九大内置对象中的其它八个	

六,EL函数库

EL函数库是由JSTL提供的.

- 导入标签库:<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
- 相关方法:
- fn:contains
- 判断字符串是否包含另外一个字符串
-

- fn:containsIgnoreCase
- 判断字符串是否包含另外一个字符串(大小写无关)
-

- fn:endsWith
- 判断字符串是否以另外字符串结束
-

- fn:escapeXml
- 把一些字符转成XML表示
- 例如 <字符应该转为<
$fn:escapeXml(param:info)$ — $fn:indexOf$ — 子字符串在母字符串中出现的位置— {fn:indexOf(name, "-")}

- fn:join
- 将数组中的数据联合成一个新字符串并,使用指定字符格开
- $fn:join(array,";")$ — $fn:length$ — 获取字符串的长度，或者数组的大小— {fn:length(shoppingCart.products)}
- fn:replace
- 替换字符串中指定的字符
- $fn:replace(text,"-","•")$ — $fn:split$ — 把字符串按照指定字符切分— {fn:split(customerNames, ";")}

- fn:startsWith
- 判断字符串是否以某个子串开始
-

- fn:substring

- 获取子串
-
$fn:substring(zip, 6, -1) - fn:substringAfter$ — 获取从某个字符所在位置开始的子串—{fn:substringAfter(zip, "-")}

- fn:substringBefore
- 获取从开始到某个字符所在位置的子串
- $fn:substringBefore(zip, "-") - fn:toLowerCase$ — 转为小写—{fn.toLowerCase(product.name)}

- fn:toUpperCase
- 转为大写字符
- $fn.UpperCase(product.name) - fn:trim$ — 去除字符串前后的空格—{fn.trim(name)}

七,自定义EL函数

- 写一个java类,类中可以定义0-n个方法,但方法必须是静态方法,并且有返回值.
- 在WEB-INF目录下创建一个.tld文件
- 在jsp页面中导入标签库

一,JSTL简介

JSTL(JSP Standard Tag Library)是JSP标准标签库,使用JSTL中的标签,可以提高开发效率,减少JSP页面中代码量,保持页面的简洁性和良好的可读性,可维护性.

JSTL是apache对EL表达式的扩展,JSTL依赖EL,JSTL是一种标签语言.

JSTL含有四个定制标记库:
- core标记库:
- 主要功能为,操作作用域的变量,流程控制,URL生成和操作等,通常和EL表达式结合使用,EL作为JSTL标签的属性值.是重点
- format标记库:
- 主要用来格式化数据,比如数字和日期等,支持使用本地化资源束进行JSP页面的国际化.学两个即可
- xml标记库:
- 该标记库包含一些标记,这些标记用来操作以XML表示的数据.过时了

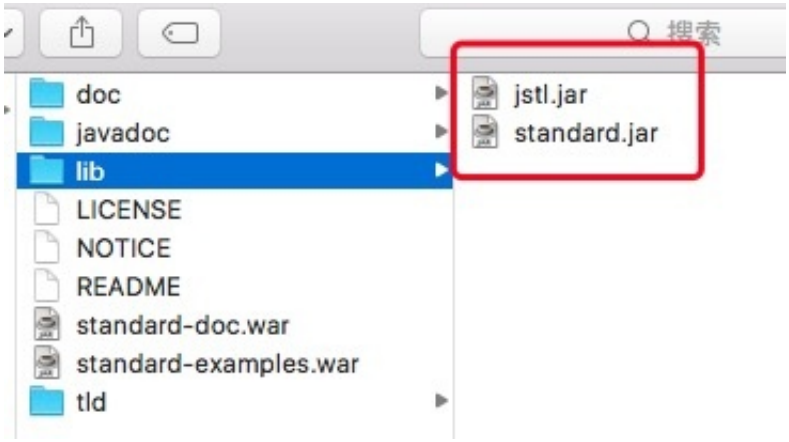
- sql标记库:
- 该标记库定义了用来查询关系数据库的操作.过时了

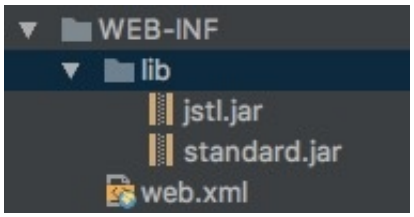
二,导入JSTL

1,下载Jar包

Jar包下载链接

2,将jstl.jar,standard.jar两个jar包拷贝到/WEB-INF/lib/下





3,选中lib下的jar包,右键,选Add as Library后直接回车即可.

4,在要使用JSTL标签库的jsp页面中通过指令元素(标签)引入要使用的标记库.

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

5,也可以在web.xml中配置,这样每个jsp页面就不用再单独配置了

三,常用JSTL标签

1,标签

其功能为创建变量,保存数据,语法格式为:

```
jstl
<c:set target="" value="" var="" property="" scope=""/>
```

- var:可选项,创建需要保存信息的变量.
- value:可选项,要保存的信息,可以是EL表达式或常量.
- scope:可选项,保存信息的变量范围(page,request,session,application).
- target:可选项,需要修改属性的变量名,一般为JavaBean实例(就是一个对象),若指定了target属性,则也必须指定property属性.
- property:可选项,需要修改的JavaBean的属性.

例:

在session域中创建一个变量gender,值为男

```
java
<body>
<c:set scope="session" var="gender" value="男"/>
<%
System.out.println(session.getAttribute("gender"));
%>
</body>
```

2,标签

其功能为在JSP页面中显示数据,语法格式为:

```
java
<c:out value="" default=""/>
```

- value:输出的信息,可以是EL表达式或常量.

- default:可选项,当value为null时,显示default的信息.

例:

显示gender变量的值

3,标签

其功能为删除变量,语法格式为:

```
java
<c:remove var="" scope=""/>
```

- var:要删除的变量.
- scope:可选项,被删除变量的范围(page,request,session,application).

例:

删除session中的变量gender

```
java
<c:remove var="gender" scope="session"/>
```

4,<c:if>标签

其功能为判断表达式的值,如果表达式的值为真则只需其主体内容,语法格式为:

```java
<c:if test="test-condition" var="" scope="">
body content
</c:if>

<c:if test="判断条件" var="" scope="">
要执行的代码
</c:if>
```

- test:需要判断的条件.

- var:要求保存条件结果的变量名,这样做的目的是为了避免在页面中多次进行相同的判断.
- scope:可选项,保存条件结果的变量范围.

例:

如果gender为男,则输出女,并将判断结果保存在man变量中
```java
//如果gender的值是男,则在页面中显示女
<c:if test="${gender.equals('男')}">

</c:if>

//在session域中,保存一个变量为man,man的值就是这个判断条件的结果
//这样如果又要判断test这个条件的话,则可以直接判断man是否为true
<c:if test="
$gender.equals('男') " var =" man " scope =" session "><c:outvalue =" 女 " /> \
{man}"/>
```

5,<c:forEach>标签

其功能为循环迭代一个集合中的对象.语法格式为:

```
jstl
\<c:forEach var="" items="" varStatus="" begin="" end="" step="">
body content
\</c:forEach>
```

- items:可选项,进行循环的元素
- var:可选项,代表当前元素的变量名
- begin:可选项,开始条件
- end:可选项,结束条件
- step:可选项,步长,默认值为1
- varStatus:可选项,显示循环变量的状态.

例:

循环显示颜色集合的值

```
java
\<c:forEach var="color" items="red,blue">
<c:out value="${color}"/>
<br/>
\</c:forEach>
```

JavaWeb三大组件

都需要在web.xml中进行配置.

- Servlet
- Listenr(两个感知监听器不需要配置)
- Filter

过滤器(Filter)

过滤器是JavaWeb三大组件之一,与Servlet很相似,不过过滤器是用来拦截请求的,而Servlet是用来处理请求的.

当用户请求某个Servlet时,会先执行部署在这个请求上的Filter,如果Filter通过,那么会继续执行用户请求的Servlet,如果无法通过Filter,那么就无法执行用户请求的Servlet.

过滤器会在一组资源(jsp,servlet,.css,.html等等)的前面执行.

可以放过请求,也可以拦截请求.

过滤器的编写步骤

写一个类实现Filter接口

```
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;
```

```java
// 以下三个方法就是Filter的生命周期
@WebFilter(filterName = "AFilter")
public class AFilter implements Filter {

    /**
     * 销毁之前执行,用来做对非内存资源进行释放
     * 在服务器关闭时销毁
     */
    public void destroy() {
    }

    /**
     * 每次过滤都会执行
     * @param req
     * @param resp
     * @param chain
     * @throws ServletException
     * @throws IOException
     */
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain ch
ain) throws ServletException, IOException {
    // 注意下面这句代码,如果写了这句代码,那么就会执行目标Servlet的service方法
    // 就相当于放行,不拦截了(当执行完目标资源后,还会回来执行doFilter后面的语句)
        // chain.doFilter(req, resp);
    // System.out.println("放行后还会执行这条输出语句");
    }

    /**
     * Filter会在服务器启动时就创建
   * 创建之后马上执行,用来做初始化
     * @param config 与ServletConfig相似
     * @throws ServletException
     */
    public void init(FilterConfig config) throws ServletException {
    // config.getInitParamerterNames();//获取所有初始化参数的名称
    // config.getInitParamerter(name);//获得Filter的参数配置,与<init-param>标签对应
    // config.getFilterName();//获得Filter的配置名称
    // config.getServletContext();
    }
}
```

FilterChain接口

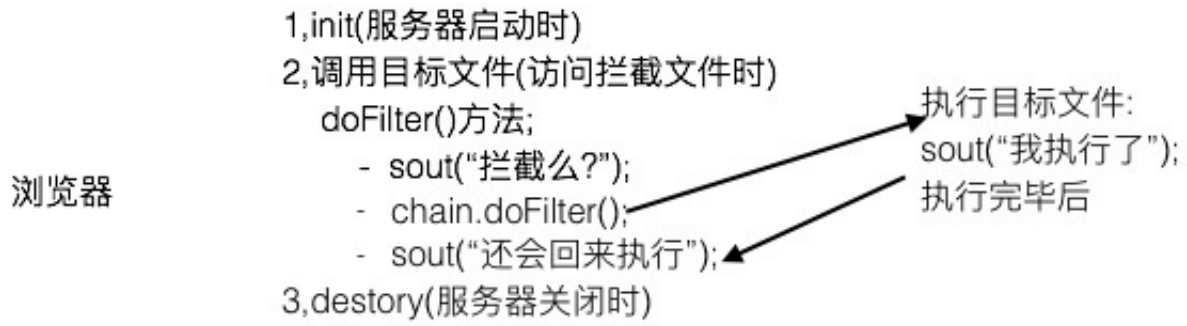
```
public interface FilterChain {
    //该接口只有一个方法,就是放行
    //表示要执行下一个Filter(若没有,则执行目标资源)
    void doFilter(ServletRequest var1, ServletResponse var2) throws IOException,
ServletException;
}

Filter#doFilter(request,response,chain){
  //在目标资源执行前执行
  System.out.println("之前");
  chain.doFilter(request,response);
  //在目标资源执行后执行
  System.out.println("之后");
}
```

在web.xml中进行配置

```
<!-- 如果没有使用注解的方式构建AFilter类的话,需要在web.xml中这样配置 -->
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.or
g/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <filter>
        <filter-name>af</filter-name>
        <filter-class>AFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>af</filter-name>
        <!-- 过滤器拦截的路径:下面的就是只拦截AServlet -->
        <url-pattern>/as</url-pattern>
        <!-- 往往这里使用的都是带*的 -->
        <url-pattern>/*</url-pattern>
        <!-- 点名要拦截哪个Servlet -->
        <servlet-name>AServlet</servlet-name>
    </filter-mapping>
</web-app>
```

过滤流程

Filter执行流程



程序输出内容:
拦截么
我执行了
还会回来执行

注意事项

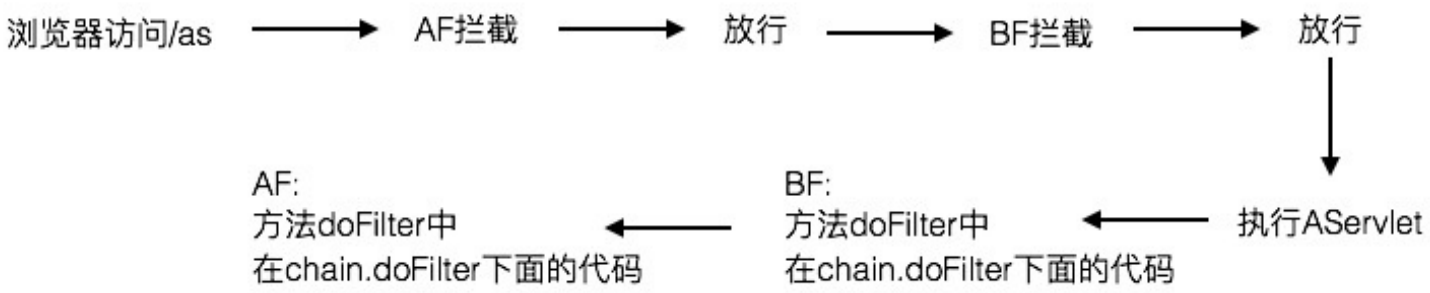
Filter是单例的.整个服务器中只有这一个AFilter类的对象.

多个Filter的执行顺序

多个Filter执行顺序

AServlet:路径为:/as

AF:拦截/*
BF:拦截/as



如果没有下一个过滤器,则直接执行目标资源,如果有下一个过滤,那么执行下一个过滤器.

过滤器的排序:

可以在web.xml中通过的顺序排序,由上到下执行.

如果基于注解的方式,那么会根据类名的顺序执行(0-9,A-Z的顺序).

Filter的四种拦截方式

可以在中进行配置,内容为下方的大写英文即可.

注解形式的写法:

```
java
@WebFilter(filterName = "AFilter",dispatcherTypes =
{DispatcherType.REQUEST,DispatcherType.FORWARD},...)
```

中进行配置:

```
html
<filter-mapping>
```

```
<filter-name>af</filter-name>
<url-pattern>/*</url-pattern>
<dispatcher>REQUEST</dispatcher>
<dispatcher>FORWARD</dispatcher>
...
</filter-mapping>
```

注: 若不写dispatcher属性,则默认为REQUEST.

注: 若写了dispatcher,则默认的会被覆盖.

- 拦截请求:拦截直接请求,不会拦截转发.
- DISPATCHER
- 比如:拦截了AS,但是如果先访问BS,再从BS转发或包含到AS,那么依然能访问过去.
- 拦截转发
- FORWARD
- 拦截包含
- INCLUDE
- 拦截错误:基本没啥用
- ERROR

Filter的应用场景

- 执行目标资源之前做预处理工作,例如设置编码,这种方式通常都会放行,只是为了在执行目标资源前做点小动作.
  - 几乎所有的Servlet都需要些request.setCharacterEncoding(),可以将这句代码放到一个Filter中.
  - 也就是说可以将在Servlet中重复性的代码,放到Filter中执行.
- 通过条件判断是否放行,例如校验当前用户是否已经登录,或者用户IP是否已经被禁用.
- 在目标资源执行后,做一些后续的处理工作,例如把目标资源输出的数据进行处理.
  - 回程拦截!

一,监听器Listner

JavaWeb三大组件:Servlet,Listener,Filter.

- 监听器:
- 是一个接口,我们自己来添加实现.
- 需要注册,例如注册在按钮上.
- 监听器中的方法,会在特殊事件发生时被调用.当按钮被点击的时候,要执行什么代码.
- 观察者模式:
- 事件:
- 偷东西(事件对象)
- 事件源:
- 小偷(被监听的事件源)
- 监听器:
- 警察:会观察小偷的动作(将监听器注册在事件源上),当小偷偷东西时(事件发生),警察会执行抓小偷方法(执行监听器的方法)

二,JavaWeb中的监听器(八个)

事件源

事件源基本就是三大域.(每个域两个就是六个监听器)

- ServletContext:在服务器启动时创建,在服务器关闭时销毁
- 生命周期监听器:ServletContextListener(两个方法,分别监听创建和销毁)
- 属性监听:ServletContextAttributeListener(三个方法,监听添加属性,替换属性,移除属性等三个事件)
- 监听替换属性的方法:

```
java
@Override
public void attributeReplaced(ServletContextAttributeEvent se) {
// 需要注意的是这里se.getValue得到的是旧值
// 新值可以用如下方法获得,因为新值肯定已经被保存到application域中了
System.out.println("修改 "+se.getName() +"=旧值"+se.getValue()
+" 新值" +se.getServletContext().getAttribute(se.getName()));
}
```

- HttpSession:
- 生命周期监听器:HttpSessionListener(两个方法,分别监听创建和销毁)

- 属性监听:HttpSessionAttributeListener(三个方法,监听添加属性,替换属性,移除属性等三个事件)
- ServletRequest:请求一次动态资源,就会创建一个
- 生死生命周期监听器监听:ServletRequestListener(两个方法,分别监听创建和销毁)
- 属性监听:ServletRequestAttributeListener(三个方法,监听添加属性,替换属性,移除属性等三个事件)

JavaWeb中编写监听器

- 写一个监听器类:要求必须去实现某个监听器接口.
- 注册,是在web.xml中配置来完成注册.

事件对象:

- ServletContextEvent: ServletContext getServletContext()
- HttpSessioinEvent: HttpSession getSession()
- ServletRequest:
  - ServletContext getServletContext()
  - ServletRequest getServletRequest()

与HttpSession相关的监听器(两个)

HttpSessionBindingListener:添加到JavaBean上,JavaBean就可以监听到自己被添加进session或被从session中移除这两个事件.

HttpSessionActivationListener(Session的活化监听):实体类对象跟着session保存到硬盘上的过程,或从硬盘获取的过程可以被这个监听器监听到,不过若要将实体类对象保存到硬盘上,该实体类需要实现Serializable接口.

特点:

- 用来添加到JavaBean上,而不是添加到三大域上.
- 这两个监听器都不需要在web.xml中注册.

其他:

禁止session的序列化:

在apache-tomcat-7.0.77/conf/context.xml文件中加入一个标签:

```html

```
<!-- Default set of monitored resources -->
<WatchedResource>WEB-INF/web.xml</WatchedResource>
<Manager pathname=""/>

<!-- Uncomment this to disable session persistence across Tomcat restarts -->
<!--
<Manager pathname="" />
-->

<!-- Uncomment this to enable Comet connection tacking (provides events
     on session expiration as well as webapp lifecycle) -->
<!--
<Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
-->
```

```

session的活化和钝化

若session一定时间不活动,服务器会将session保存到硬盘,可以在apache-tomcat-7.0.77/conf/context.xml文件中设置.

保存到硬盘的过程就叫钝化,从硬盘中获取的过程叫做活化.

```
html
<!-- maxIdleSwap的值最小可以填1,单位为分钟 -->
<!-- directory为保存session的文件夹 -->
<!-- session文件的格式:文件名:sessionId,文件拓展名:.session -->
<Manager className="org.apache.catalina.session.PersistentManager" maxIdleSwap="1">
<Store className="org.apache.catalina.session.FileStore" directory="mysession"/>
</Manager>
```