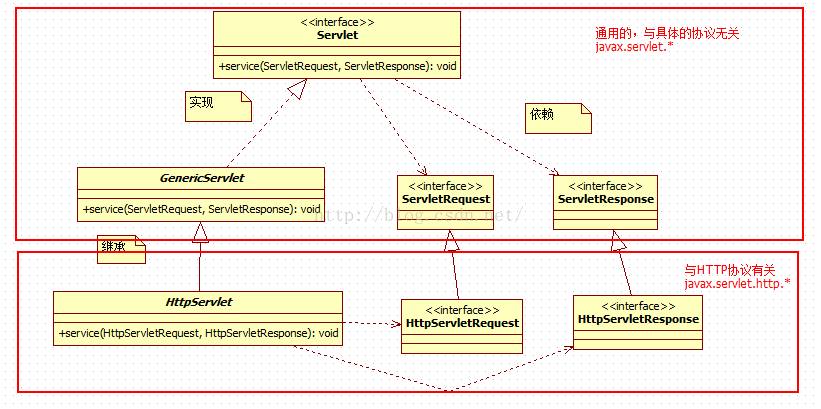
Servlet详解

# Servlet API详解



Servlet<---GenericServlet<--- HttpServlet

ServletRequest<---HttpServletRequest

ServletResponse<---HttpServletReponse

HttpSession，Cookie

## Servlet接口

|  |
| --- |
| **init(ServletConfig config)**：在Servlet实例化之后，Servlet容器会调用init()方法，来初始化该对象，主要是为了让servlet对象在处理客户请求前完成一些初始化工作。例如建立数据库的连接，获取配置信息等。对于每个Servlet实例，init()方法只能被调用一次。Servlet容器通过config参数向Servlet传递配置信息。可以通过config获取web应用程序的初始化参数，还可以获取ServletContext对象  **service(ServletRequest req,ServletResponse res)**： 容器调用service()方法处理客户端的请求，在调用service()方法被容器调用之前，必须确保Init()方法正确完成，容器会构造一个ServletRequest对象和ServletResponse对象作为参数传递给service()方法。  **destroy()**：当容器检测到一个Servlet对象应该从服务中被移除的时候，容器会调用该对象的destory方法，以便让Servlet对象可以释放它所使用的资源，保存数据到数据库中。  **ServletConfig getServletConfig()**：返回容器调用init()方法时传递给Servlet对象的ServletConfig对象  **String getServletInfo()**：返回一个String类型的字符串，包含了关于Servlet的信息，例如版权，作者等。 |

## ServletRequest接口

对于每一个HTTP请求都会创建一个ServletRequest。并将它传给Servlet的service方法。

|  |
| --- |
| **Object getAttribute(String name)**：返回以name为名字的属性的值，如果不存在，返回null  **Enumeration getAttributeNames()**：返回请求中所有可用的属性的名字。返回一个枚举集合  **void removeAttribute(String name)**：移除请求中名字为name的属性  **void setAttribute(String key,Object val)**：在请求中保存一个键值对。  **getCharacterEncoding()**：返回请求正文使用的字符编码的名字  **int getContentLength()**：以字节为单位，返回请求正文的长度。  **String getContentType()：**返回请求主体的MIME类型  **ServletInputStream getInputStream()**  **BufferedReader getReader()**  **String getLocalAddr()**：返回接收到请求的网络接口的IP地址。  **String getRemoteAddr()**：返回发送请求的客户端或最后一个代理服务器的IP地址  **String getRemoteHost()**：返回发送请求的客户端或者最后一个代理服务器的完整限定名  **String getLocalName()**：返回接收到请求的IP接口的主机名  **int getLocalPort()**：返回接收到请求的网络接口的IP端口号  **int getRemotePort()**：返回发送请求的客户端或最后一个代理服务器的IP源端口。  **String getParameter(String name)**：返回请求中name参数的值，如果name参数有多个值，该方法返回值列表中的第一个值。如果请求中没有找到该参数，返回null  **Enumeration getParameterNames()**：返回请求中包含的所有的参数的名字。  **String[] getParameterValues(String name)**：返回请求中name参数所有的值  **String getProtocal()**：返回请求使用的协议的名字和版本 例如： HTTP/1.1  **RequestDispatcher getRequestDispatcher(String path)**：返回Request Dispather对象，作为path所定位的资源的封装。  **String getServerName()**：返回请求发送到服务器的主机名  **int getServerPort()**：返回请求发送到服务器的端口号  **setCharacterEncoding(String env)**：覆盖在请求正文中所使用的字符编码的名字。 |

## ServletResponse接口

对于每一个HTTP请求都会在调用service方法之前创建一个ServletRequest。并将它传给Servlet的service方法。

|  |
| --- |
| **flushBuffer()**：强制把任何在缓存中的内容发送到客户端  **int getBufferSize()**：返回实际用于响应的缓存的大小，如果没有使用缓存，这个方法返回0  **String getCharacterEncoding()**：返回响应中发送的正文所使用的字符编码（MIME字符集）  **String getContentType():**返回响应中发送的正文所使用的MIME类型  **ServletOutputStream getOutputStream()**  **PrintWriter getWriter()**  **boolean isCommitted()**：返回一个布尔值，指示是否已经提交了响应。  **void reset()**：清除在缓存中的任何数据，包括状态代码和消息报头。  **void resetBuffer()**：清除在缓存中的响应内容，保留状态代码和消息包头。  **void setBufferSize(int size**):设置响应正文的缓存大小。  **void setCharacterEncoding(String charset)**：设置发送到客户端的响应的字符编码  **void setContentLength(int len):**设置内容正文的长度  **void setContentType(String type):**设置要发送到客户端的响应的内容类型。例如："text/html;charset=UTF-8" |

## ServletConfig接口

Servlet容器初始化Servlet时，将ServletConfig传递给Servlet的init方法。ServletConfig封装可以通过@WebServlet或者描述符传递。

|  |
| --- |
| **String getInitParameter(String name)**：返回名字为name的初始化参数的值，初始化参数在web.xml配置文件中进行配置。  **Enumeration getInitParameterNames()：**返回Servlet所有初始化参数的名字的枚举集合。  **ServletContext getServletContext()**：返回Servlet上下文对象的引用。  **String getServletName()：**返回Servlet实例的名字，这个名字是web应用程序的部署描述符中指定。  @WebServlet(name = "ServletConfigDemoServlet",  urlPatterns = { "/servletConfigDemo" },  initParams = {  @WebInitParam(name="admin", value="Harry Taciak"),  @WebInitParam(name="email", value="admin@example.com")  }  )  ServletConfig servletConfig = getServletConfig();  String admin = servletConfig.getInitParameter("admin");  String email = servletConfig.getInitParameter("email"); |

## ServletContext

ServletContext表示Servlet应用程序。每个web应用程序只有一个context。

## GenericServlet抽象类

如果要编写一个通用的Servlet，只需要从GenericServlet继承，并且实现抽象方法service()。GnnericServlet实现了Servlet和ServletConfig接口。

不必亲自保存ServletConfig，只需要覆盖service()方法。

## HttpServlet

在绝大多数的网络应用中，都是客户端通过HTTP协议去访问服务器端的资源我们编写的Servlet也主要是应用于HTTP协议的请求和响应，为了快速开发应用于HTTP协议的Servlet类，sun公司在javax.servlet.http包中给我们提供了一个抽象类HttpServlet，它继承自GenericServlet类，用于创建适合WEB站点的HTTP Servlet。

|  |
| --- |
| public void service(ServletRequest req,ServletResponse res);  protected void service(HttpServletRequest req,HttpServletResponse resp)  针对HTTP1.1定义的7种请求方法GET,POST,HEAD,PUT,DELETE,TRACE,OPTION ， HttpServlet提供了7个处理方法，常用一下两个方法。  doGet(HttpServletRequest req,HttpServletResponse resp)  doPost(HttpServletRequest req,HttpServletResponse resp)  注意：HttpServlet不需要覆盖service()方法，而是覆盖doGet,doPost方法。其他几种方法基本上不需要覆盖。 |

## HttpServletRequest接口

继承自javax.servlet.servletRequest,在该接口中新增的方法如下：

|  |
| --- |
| **String getContextPath()**：返回请求URI中表示请求上下文的部分，上下文路径是请求URI的开始部分  **Cookie[] getCookies()**：返回客户端在此次请求中发送的所有的Cookie对象  **String getHeader(String name):**返回名字为name的请求报头的值  **Enumeration getHeaderNames()**：返回此次请求中包含的所有的报头名字的枚举集合  **Enumeration getHeaders(String name)**：返回名字为name的请求报头的所有值的枚举集合  **String getMethod()**：返回此次请求所使用的HTTP方法的名字  **String getServletPath()**：返回请求URI中调用Servlet的部分  **HttpSession getSession()**：返回和此次请求相关联的Session  **HttpSession getSession(boolean create)**：返回此次请求相关联的Session,如果没有给客户端分配Session,并且create参数为true,则创建一个新的Session,如果为false，此次请求没有一个有效的HttpSession,返回null |

## HttpServletResponse

该接口继承ServletResponse,新增的方法入下

|  |
| --- |
| **addCookie(Cookie cookie)**  **addHeader(String name,String value);**  **boolean containsHeader(String name)**  **encodeRedirectURL(String url)**：使用SessionID 对于重定向的url编码，以便用于sendRedirect()方法中  **encodeURL(String url)**：使用SessionID 指定的url编码  **sendRedirect(String location)**：发送一个临时的重定向响应到客户端，让客户端访问新的url |

# Servlet基础

servlet是sun公司制订的一种用来扩展web服务器功能的组件规范。扩展web服务器功能。**组件规范**(符合一定规范，实现部分功能，需要部署到相应的容器上才能运行的软件模块。) **容器**是符合一定规范，为组件提供运行环境的程序。

## 开发一个Servlet程序

第一步：tomcat的安装和配置（jdk的配置）

第二步：写一个java类，实现Servlet接口或者继承HttpServlet抽象类。然后编译打包。目录结构如下:

--appname

--WEB-INF

--classes(字节码文件)

--lib(可选,放.jar文件)

--web.xml(部署描述文件)

注意：除了appname可以自己命名，其它都不能改。

然后部署，生成的文件夹拷贝到容器特定的文件夹下面，比如，tomcat对应的是webapps文件夹。启动tomcat，访问servlet。

### 常用部署描述符

|  |
| --- |
| <servlet>  <servlet-name>**helloServlet**</servlet-name>  <servlet-class>web.HelloServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>**helloServlet**</servlet-name>  <url-pattern>/hello</url-pattern>  </servlet-mapping>  context-param, filter,listener |

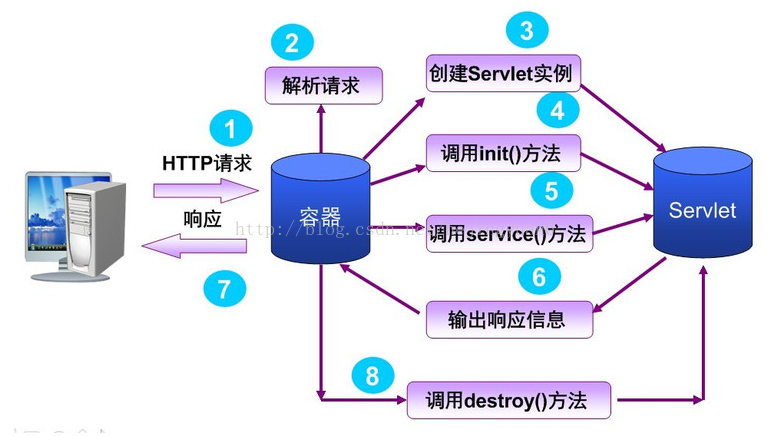
### 常用注解

|  |
| --- |
| @WebServlet()  @WebFilter() |

## Servlet的运行过程

|  |
| --- |
| ⒈ 客户端发送请求至服务器端；  ⒉服务器端根据web.xml文件中的Servlet相关配置信息，将客户端请求转发到相应的Servlet  ⒊ Servlet引擎调用Service()方法，根据request对象中封装的用户请求与数据库进行交互，返回数据之后，Servlet会将返回的数据封装到response对象中；  ⒋ Servlet生成响应内容并将其传给服务器。响应内容动态生成，通常取决于客户端的请求  ⒌ 服务器将响应返回给客户端 |

## Servlet生命周期



1） **加载和实例化**；容器调用servlet的构造器创建一个servlet对象，在默认情况下，不管有多少个请求，容器只会创建一个servlet对象。在默认情况下，容器只有收到请求之后，才会创建servlet对象。另外，容器在启动的时候，就将某些servlet对象创建好。这些servlet必须配置一个参数<load-on-startup>参数</load-on-startup>参数：大于等于0的整数，值越小，优先级越高。

2） **初始化**；容器创建好servlet对象之后，会立即调用init方法。一般情况下，不需要写init方法，因为GenericServlet已经实现了init方法：将容器创建的ServletConfig对象保存下来，并且提供了getServletConfig方法来获得该对象。调用了一个空的init方法(该init方法用于子类去override)。如果要实现自己的初始化处理逻辑，只要override init()方法。初始化方法只会执行一次。ServletConfig对象可以用来访问servlet的初始化参数。

3） **响应请求**：Servlet初始化之后，就处于响应请求的就绪状态，此时如有客户端请求发送，就会调用Servlet实例的service()方法（在调用之前，首先会创建HttpServletRequest和HttpServletResponse对象），并且根据用户的请求方式，调用doPost或者doGet方法；

4）**Servlet的销毁**：容器依据自身的算法来决定是否销毁servlet对象。容器在销毁servlet对象之前，会调用servlet对象的destroy方法。destroy方法只会执行一次。

## 常见问题解析

### get请求与post请求

|  |
| --- |
| **发送get请求**：在浏览器地址栏输入某个地址；点击链接；表单默认的提交方式  **发送post请求**：设置表单method="post"。  **get请求的特点** ：请求参数会添加到请求资源路径的后面，只能添加少量的参数(因为请求行只有一行，大约只能存放2K左右的数据)。请求参数会显示在浏览器地址栏)路由器会记录请求地址)。  **post请求的特点**：请求参数添加到实体内容里面，可以添加大量的参数。相对安全，但是，post请求不会对请求参数进行加密处理(可以使用https协议来保证数据安全)。 |

### 参数获取

|  |
| --- |
| String request.getParameter(String paramName);  String[] request.getParameterValues(String paramName);  **注意**：如果paraName对应的值不存在，返回null。  **url中的参数**：如果paraName对应的值不存在，返回null。  **表单中的参数**：  -- 输入域(文本密码，隐藏域)：值没有返回空白字符。  -- 单选按钮：没有选中获取值返回null  -- 复选框：没有选中的复选框返回null值。选中的字符返回  -- 下拉框：默认选中一个。  -- 多选框：没有选中的复选框返回null值。  **注意**：如果表单有多个同名输入元素，所有值都会被提交。如果调用getParameter方法的话，只会返回最后一个值。 |

### 乱码问题

**乱码产生的原因**：编码不支持输出中文字符；没有按编码格式进行解码。

|  |
| --- |
| **# servlet输出中文**  默认情况下，out.println会使用"iso-8859-1"来进行编码。response.setContent Type("text/html;charset=utf-8"); //注意，编码格式只要支持中文即可  text/html;charset=utf-8作用：生成一个消息头(content-type),告诉浏览器，服务器返回的数据类型和编码格式；也设置了out.println在输出时使用的编码格式。  **#获取url参数中的中文参数**  <Connector **URIEncoding="utf-8"** connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>  **#获取表单中中文参数**  当表单提交时，浏览器会对表单中的中文使用。浏览器打开表单所在的页面时的编码格式进行编码。所有我们要以正确的编码进行解码。  解决方式一:  step1,要保证表单所在的页面按照指定的编码格式打开。  <meta http-equiv="content-type" content="text/html;charset=utf-8">  step2,让服务器按照指定的编码格式进行解码。  request.setCharacterEncoding("utf-8"); //只对post请求有效。    方式二:  step1,同上。  step2, name = new String(name.getBytes("iso-8859-1"),"utf-8"); |

### 重定向和转发

**重定向**：服务器发送一个302状态码和一个Location消息头 (值是一个地址，称为重定向地址),当浏览器收到之后，会立即向重定向地址发请求。

response.sendRedirect(String url); //url:重定向地址

**特点**：重定向地址是任意的;重定向之后，浏览器地址栏的地址变成重定向地址。

**注意**：重定向之前，不能调用out.close方法；重定向之前，会先清空response上缓存的数据。

**转发**：一个web组件(servlet或者jsp)将未完成的处理通过容器交给另一个web组件继续完成。常见的情况是：一个servlet获得数据之后，转发给一个jsp，由这个jsp来负责展现这些数据。

|  |
| --- |
| //1. 绑订数据到request对象上。  request.setAttribute(String name, Object obj);  //2. 获得转发器，url: 转发的目的地地址  RequestDispatcher rd = request.getRequestDispatcher(String url);  //3. 转发  rd.forward(request,response);  //4.依据绑订名称找到绑订值，如果值不存在返回null。  Object request.getAttribute(String name); |

**转发的特点**：转发之后，浏览器地址栏的地址不变。转发的目的地仅限于同一个应用。

**注意**：转发之前，容器会先清空response对象上缓存的数据。且不能够调用out.close()，

**转发与重定向的区别**：1) 转发所涉及的各个组件可以共享同一个request对象和同一个response对象,而重定向不行。当容器收到请求，会创建request,  
response,当容器发送响应之后，会立即删除request,response。也就是说，request,response的生存时间很短暂 (一次请求与响应期间)。

2)转发的目的地地址必须是同一个应用内部某个地址，而重定向地址不限。

3)转发之后，浏览器地址栏的地址不变；而重定向会变。

4)转发是一件事未做完，而重定向是一件事已经做完，然后再去做另一件事。

### 资源路径问题

**-- 资源路径查找方式**

servlet容器如何处理请求资源路径比如，在浏览器地址栏输入http://ip:port/  
web04\_2/abc.html请求资源路径("/web04\_2/abc.html")

第一步：容器要依据请求资源路径找到应用所在的文件夹,接下来，容器会假设访问的是一个servlet,然后查找web.xml文件中与之匹配的servlet。

第二步：匹配(查找web.xml文件中与之匹配的servlet):

方式一: 精确匹配，要求web.xml文件当中，必须这样配置:

<url-pattern>/abc.html</url-pattern>

方式二: 通配符匹配，使用"\*"匹配任意的0个，1个或者多个字符。比如

<url-pattern>/\*</url-pattern>

方式三:后缀匹配,使用"\*."开头，后接任意的1个或者多个字符。比如

<url-pattern>\*.do</url-pattern> 匹配所有以".do"结尾的请求。

第三步：如果以上都不匹配，容器会查找相应的文件，如果找到，则返回，找不到，返回404。

-- 相对路径和绝对路径

**相对路径**：不以"/"开头的路径，相对于当前页面url(ip:port/demo/hello  
/1.html）相对路径是/demo/hello。

绝对路径：以"/"开头的路径。链接地址、表单提交地址、重定向地址从**应用名开始写**；**转发从应用名之后开始写（**转发只能发生在同一应用中**）**。

String request.getContextPath();//获得应用名

### servlet上下文

servlet上下文：容器在启动之后，会为每一个web应用都创建一个符合ServletContext接口要求的对象，该对象称为servlet上下文。

唯一性: 一个web应用一个servlet上下文。

一直存在: 只要容器不关闭，servlet上下文会一直存在。

#获得servlet上下文ServletConfig, FilterConfig, GenericServlet, Http  
Session都提供了getServletContext方法。

servlet上下文的作用：绑订数据setAttribute,removeAttribute,  
getAttribute注意: request,session,servlet上下文都能绑订数据，

区别:区别1: 绑订的数据存在的时间不一样， request < session < servlet上下文。如果都能解决绑订的问题，应该优先使用生命周期短的。

区别2: 访问的范围不一样，request对象上绑订的数据可以被同一个请求中所涉及的各个组件访问；session绑订的数据可以被同一个会话所涉及的组件访问；servlet上下文绑订的数据可以被同一个应用所涉及的所有组件访问。

### Servlet多线程安全性问题

servlet存在线程安全问题的原因：容器收到一个请求，会启动一个线程来处理该请求。容器在默认情况下，对于某个servlet类型，只会创建一个实例。当容器收到多个请求，则会有多个线程同时访问某个servlet实例的情况，就有可能产生线程安全问题。比如，多个线程同时去修改servlet实例的属性。

解决方式

-- 加锁：使用synchronized对有线程安全问题的代码加锁。

-- servlet实现SingleThreadModel接口： (不建议使用)。容器会为servlet创建多个实例(一个线程一个实例),因为有可能会产生过多的实例，影响性能，所以不建议使用。

### Servlet异常处理

第一步：将异常交给servlet容器来处理将异常抛给容器，比如

throw new ServletException(e);

第二步：配置一个错误处理页面。

|  |
| --- |
| <error-page>  <exception-type>javax.servlet.ServletException</exception-type>  <location>/error.jsp</location>  </error-page>  告诉容器，捕获到指定的异常之后，调用location指定的页面。 |

# 会话跟踪

HTTP协议本身是基于请求/响应模式的，无状态的协议。也就是说，当客户端的请求到来，服务器端做出响应之后，连接就被关闭了。用户活动发送在多个请求和响应之中，作为web服务器来说，必须采用一种机制唯一的标识一个用户，同时记录该用户的状态。

## URL重写

网页链接的过程中，可以将识别信息添加到url中。

## 隐藏域

页面经过表单跳转后，可以使用隐藏域保留识别信息（id)。

## cookie

当浏览器访问服务器时，服务器会发送少量的数据给浏览器(使用set-cookie消息头),浏览器会将这些数据保存下来；当浏览器再次访问服务器时，会将之前保存的这些数据发送给服务器(使用cookie消息头)。

### Cookie API详解

|  |
| --- |
| **#创建cookie**  Cookie c = new Cookie(String name,String value);  response.addCookie(c);  **#查询cookie**  Cookie[] cookies = request.getCookies(); **注意**：该方法有可能返回null。  String cookie.getName(); //获得cookie的名称  String cookie.getValue(); //获得cookie的值  **#cookie的生存时间**  cookie.setMaxAge(int seconds);seconds：单位是秒  seconds > 0: 浏览器会把cookie保存到硬盘上,超过指定的时间(seconds)，浏览器会删除该cookie。  seconds < 0: 缺省值,浏览器会把cookie保存到内存里面，只要浏览器关闭，cookie就会被销毁。  seconds = 0: 删除cookie比如，要删除一个名叫userprofile的cookie:  Cookie c = new Cookie("userprofile","");  c.setMaxAge(0);  response.addCookie(c); |

### cookie编码问题

cookie只能保存合法的ascii字符，中文很显示不是ascii字符，需要将其转换成合法的ascii字符。

|  |
| --- |
| String URLEncoder.encode(String str, String code);  比如 URLEncoder.encode("李白","utf-8");  URLDecoder.decode(String str,String code); |

### Cookie路径问题

浏览器在访问服务器的某个地址的时候，会比较cookie的路径是否与该地址匹配，只有匹配的cookie才会发送。

匹配的规则：要访问的地址必须是cookie的路径或者其子路径。cookie有一个缺省的路径，值等于创建该cookie的web组件的路径：比如: /web  
07/sub01/addCookie.jsp添加了一个cookie,则该cookie的缺省的路径等于

"/web07/sub01"。

/web07/findCookie.jsp error

/web07/sub01/findCookie2.jsp ok

/web07/sub01/aaa/findCookie3.jsp ok

修改cookie的路径： cookie.setPath(String path);一般情况下，会将path设置为应用名，比如cookie.setPath("/web07");添加一个cookie

|  |
| --- |
| Cookie c = new Cookie("cookie1",  URLEncoder.encode("李白","utf-8"));  c.setMaxAge(3600);  c.setPath("/appname");  response.addCookie(c); |

### cookie的限制

|  |
| --- |
| cookie可以被用户禁止。  cookie保存在客户端(浏览器),不安全，所以，如果要将敏感数据(比如帐号、密码)保存在客户端，一定要加密。  cookie保存的数据大小有限制，大约是4k左右(跟浏览器有关系)。  浏览器保存的cookie的数量也有限制，大约是300个左右(跟浏览器有关系)。 |

## Session

浏览器访问服务器时，服务器会创建一个session对象(该对象有一个唯一的id,一般称之为sessionId),然后，服务器在默认情况下，会使用cookie机制(使用set-cookie消息头)将sessionId发送给浏览器。浏览器再次访问服务器时，会将sessionId发送给服务器，服务器依据sessionId找到之前创建的session对象。Session对象主要有唯一识别的sessionID，和key-value数据

### HttpSession API使用

|  |
| --- |
| **#获取session**  方式一：HttpSession s = request.getSession(boolean flag);  **flag = true**：先查看请求当中是否有sessionId,如果没有，则创建一个session对象；如果有,则依据sessionId查找对应的session对象，如果找到，则返回，找不到，创建一个新的session对象。  **flag = false**：先查看请求当中是否有sessionId,如果没有，则返回null；如果有,则依据sessionId查找对应的session对象，如果找到，则返回，找不到，返回null。    方式二：HttpSession s = request.getSession();等价于 request.getSession(true)。  **#常用方法**  getId();//获得sessionIdString  setAttribute(String name,Object obj); //设置属性和值  Object getAttribute(String name); //获取属性值  removeAttribute(String name); //删除属性  **# session超时**  容器会将空闲时间过长的session对象删除掉。容器默认的超时时间限制一般是30分钟。修改容器缺省的超时限制：conf/web.xml  <session-config>  <session-timeout>30</session-timeout>  </session-config>  也可以将以上配置放到某个具体的应用的web.xml文件里面。  setMaxInactiveInterval(int seconds); //  **#删除session**  invalidate() |

### 禁止cookie使用session的方法

可以使用url重写来解决这个问题，浏览器访问服务器上的某个地址时，需要使用服务器提供的地址(该地址后面会添加sessionId)。

response.encodeURL(String url);//方法用于链接地址、表单提交地址的处理。

response.encodeRedirectURL(String url);//该方法用于重定向地址的处理

比如response.sendRedirect(response.encodeRedirectURL("list.do"));

转发不用考虑。

### session的优缺点(跟cookie相比)

|  |
| --- |
| **优点**：  **安全**(session将状态保存在服务器端，而cookie将状态保存在浏览器端)。  **保存的数据量更大**(cookie大约只能保存4k左右)。  **保存的数据类型更丰富**(cookie只能保存字符串)。  **缺点**：  session要将状态保存的服务器端，会占用服务器  更多的内存空间，而cookie将状态保存在客户端(浏览器)，对服务器不会占用过多的内存空间。 |