# Servlet

## 一、在Eclipse中创建Servlet

Servlet其实就是一个java类，其对象的生成和销毁都由tomcat完成。

### 1.创建Servlet类

①新建一个项目，选择Web下的Dynamic web Project，然后点击next，勾选生成web.xml然后点击finish。

②在Java Resource目录下的src目录中建一个包，并新建和存放一个Servlet类。

**新建Servlet类的方法**：

**①实现Servlet接口，要重写全部的方法**

**②继承GenericServlet抽象类，重写service()方法**。

**③继承HttpServlet抽象类，重写需要的方法，一般为doGet()和doPost()方法。**

### 2.配置Servlet

在WebContent下的WEB-INF下的web.xml进行配置Servlet



其中<servlet-name>中填的是Servlet的别名，一般直接填对应的类名。<servlet-class>中填的是创建的Servlet类的全名(包名+类名)，这个的作用是方便系统使用Class.forName()获取Servlet的对象。注意<servlet>和<servlet-mapping>中的<servlet-name>中的别名必须相同。<url-pattern>是Servlet映射的请求地址，用户通过该地址访问Servlet，必须以斜线开头/，斜线之后的内容可以是任意(不能全是数字，因为java识别不了)。

**特别注意，修改了tomcat的web.xml配置信息后，一定要重启tomcat，即要用start启动tomcat，而不是Debug的方式。**注意只是修改了一般的java代码而没有修改配置信息，一般不用重新启动tomcat，tomcat会自动进行重新编译。

### 3.客户端访问Servlet

在浏览器中输入**localhost:8088/项目名称/ url-pattern中的路径(不用斜线)**。tomcat中的web.xml内的默认端口号是8080，由于自己的这个端口号老是被其它进程占用，自己改为了8088端口号。

浏览器根据配置的url-pattern访问servlet，tomcat会执行servlet的service方法处理请求。

[Servlet有关的图片\Servlet类运行过程.png](file:///F:\1.java笔记（和盈）\4.javaEE基础\Servlet有关的图片\Servlet类运行过程.png)

## 二、Servlet/GenericServlet/HttpServlet之间的关系

**Servlet**是一个**接口**，其service方法中的参数为ServletRequest、ServletResponse。Servlet是所有Servlet类必须直接或间接实现的一个接口。

GenericServlet是一个抽象类，其实现了Servlet接口。所有的servlet的初始化都是通过init方法实现的。该类的service方法中的参数为ServletRequest、ServletResponse。

HttpServlet也是一个抽象类，继承了GenericServlet抽象类。要特别注意HttpServlet中的request和response都是HttpServletRequest和HttpServletResponse。因为HttpServlet中对ServletRequest和response进行了强转。

GenericServlet和HttpServlet都有一个构造器，且都为无参构造器。

## 三、Servlet的启动优先级

正整数中，数字小的优先级大于数字大的。

0的优先级最高，默认情况优先级是-1。

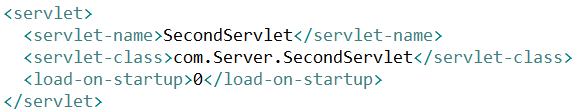
当load-on-startup 取值为负整数时，Tomcat 不会报错，能够正常启动，但是Servlet 不能被初始化。

当优先级取异常值(浮点数、字符串)，Tomcat 不会赋值 0，而是报启动异常，不能正常初始化 Servlet。

<load-on-startup></load-on-startup>这种情况也不能正常启动Tomcat。

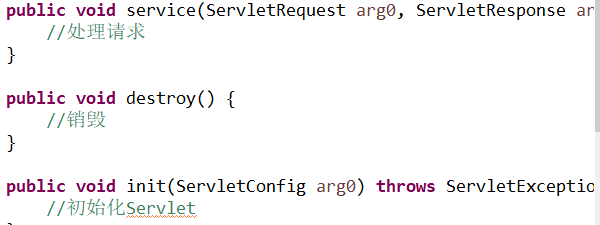


在web.xml中设置了多个servlet的时候，可以使用load-on-startup来指定servlet的加载顺序，服务器会根据load-on-startup的大小依次对servlet进行初始化。**(重要)配置load-on-startup后，servlet在web容器启动后立即加载，但只是调用servlet的init()方法**，用以**初始化该servlet相关的资源**。初始化成功后，该servlet可响应web请求；**如未配置load-on-startup，容器一般在第一次响应web请求时，会先检测该servlet是否初始化，如未初始化，则调用servlet的init()先初始化，初始化成功后，再响应请求**。



## 四、Servlet的生命周期(重点)

与Servlet生命周期有关的三个常用方法：



Sevlet的生命周期是指**Servlet对象从被创建到被销毁的过程。**

servlet对象的初始化和销毁只会发生一次，因此init()和destroy()方法只能被servlet容器调用一次，而service()方法取决于servlet被客户端访问的次数。

**Servlet对象的生命周期由tomcat(web容器)来维护**

### 1.创建对象并初始化过程：init()方法

①生成Servlet对象：**默认情况下，第一次访问servlet的时候生成servlet的对象**，即默认情况下不是tomcat启动就生成Servlet对象，而是在浏览器发送请求之前、tomcat调用service()方法之前就创建的对象。**如果需要在tomcat启动的时候就创建对象需要配置<load-on-startup>（配置为0或者正数）**。

②在生成servlet对象后会立即调用init方法初始化servlet对象。**该方法只执行一次**。

具体过程：

**<1>Servlet容器加载Servlet类，把它的. Class文件中的数据读到内存中。**

**<2>Servlet容器创建ServletConfig对象。ServletConfig对象包含了servlet的初始化配置信息。此外servlet容器还会使得ServletConfig对象与当前的web应用的servletContext对象关联。**

**<3>Servlet容器创建servlet对象**

**<4>Servlet容器将ServletConfig对象进行数据封装，然后作为参数传到servlet对象的init(ServletConfig   config)方法中初始化Servlet。只调用一次init方法， init方法必须在servlet接收到任何请求之前完成。**

### 2.运行过程：service()方法

执行service方法处理请求，不会生成多个servlet对象。**该方法可多次执行**。用户发送一次请求就调用一次。

### 3.销毁过程：destroy()方法

Web项目从服务器中移除或者tomcat关闭的时候会销毁servlet对象，执行destroy()方法。**该方法只执行一次**。

一般情况下，修改了java代码，tomcat会自动进行重新编译，然后原来的Servlet对象就会被销毁，并再次创建新的对象。

## 五、Servlet的工作模式

Tomcat是Servlet的运行环境，即一个Servlet容器。Servlet容器的作用是负责处理客户请求，当客户请求来到时，Servlet容器获取请求，然后调用某个Servlet的service()方法，并把Servlet的执行结果返回给客户。

**步骤**：

①服务器启动，客户端发送请求至服务器。

②服务器调用对应的Servlet的service()方法处理请求，Servlet处理完请求之后生成响应内容并将其传给服务器。

③服务器将响应返回客户端。

**详细过程**：

**1.客户端向web服务器(Tomcat)发送请求，此前的Servlet并未被初始化。**

**2.Tomcat从磁盘中加载Servlet，加载Servlet完成。**

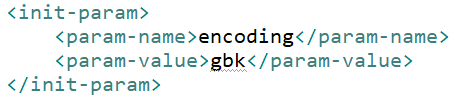
**3.Tomcat解析Http请求为request对象，然后将request转发至相应的Servlet中进行处理，Servlet处理之后返回response。**

**4.Tomcat将response转为Http响应。然后Tomcat再将Http响应返回给客户端。**

[Servlet有关的图片\Servlet工作模式动画.gif](file:///F:\1.java笔记（和盈）\4.javaEE基础\Servlet有关的图片\Servlet工作模式动画.gif)

## 六、在web.xml中配置init

<init-param>是配置初始化参数，该参数可以在init()方法中得到。这个<init-param>是在<servlet>之中。一个Servlet中可以配置多个初始化参数。



ServletConfig**.getInitParameter(String name)**：该方法是获得初始化参数，返回值为String类型的。

## 七、Servlet知识补充

**在一个应用程序中，每种Servlet类型只能有一个实例**。

用户请求会使Servlet容器调用Servlet的service()方法，并传入一个ServletRequest对象和一个ServletResponse对象。ServletRequest对象和ServletResponse对象都是由Servlet容器（例如TomCat）封装好的，程序员可以直接使用这两个对象。

**对于每一个应用程序，Servlet容器还会创建一个ServletContext对象。这个对象中封装了上下文（应用程序）的环境详情。每个应用程序只有一个ServletContext。每个Servlet对象也都有一个封装Servlet配置的ServletConfig对象。**

**Servlet是单例模式(即一个对象)，是以多线程的方式调用service()方法。Servlet不是线程安全，所以尽量不要再service()方法中操作全局变量。**

Servlet每次处理请求时服务器都会创建一个新的线程，访问service()方法，用来处理请求，会调用多次。

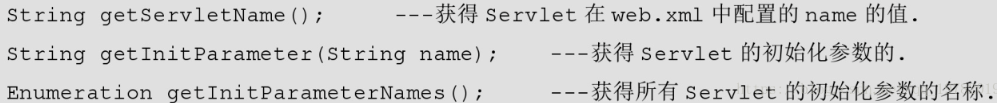
ServletConfig中包含了Servlet的配置信息，每一个Servlet都有其唯一对应的ServletConfig。注意ServletConfig对象是由Servlet容器(Tomcat服务器)创建的。

一个WEB应用对应一个唯一的ServletContext对象，ServletContext对象在项目启动时创建，在项目卸载时销毁。

service()方法抛出两个异常：**ServletException、IOException**

在调用Servlet的service( )方法前，Servlet容器会先创建一个ServletRequest对象和一个ServletResponse对象，并把它们作为参数传给service( )方法。

ServletConfig接口有如下几个方法：



Servlet(接口)、GenericServlet(抽象类)、HttpServlet(抽象类)、ServletRequest/Response(接口)、HttpServletRequest/Response(接口)、ServletConfig(接口)、ServletContext(接口)。

有了ServletContext对象，就可以共享从应用程序中的所有资料处访问到的信息，并且可以动态注册Web对象。

**HttpServlet有两个特性是GenericServlet所不具备的：**

**①不用重写service方法，而是重写doGet或者doPost方法。少数情况，还会重写其他方法。**

**②使用的是HttpServletRequest和HttpServletResponse对象**。

## 八、HttpServlet中两个service()方法

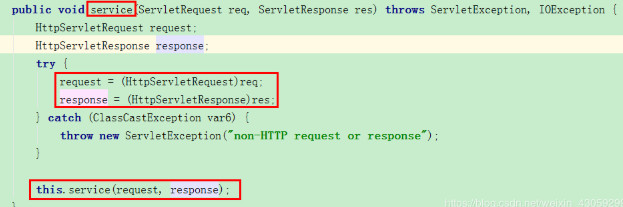
一**个是public修饰的service方法**，该方法的参数为ServletRequest和ServletResponse类型的，该方法用于将ServletRequest和ServletResponse转化为HttpServletRequest和HttpServletResponse，然后将强转得到的HttpServletRequest和HttpServletResponse作为参数传入protect修饰的service()方法中。该**public修饰的service方法是重写父类GenericServlet的方法。**之所以能这样强制的转换，是因为在调用Servlet的service方法时，Servlet容器总会传入一个HttpServletRequest对象和HttpServletResponse对象，预备使用HTTP。

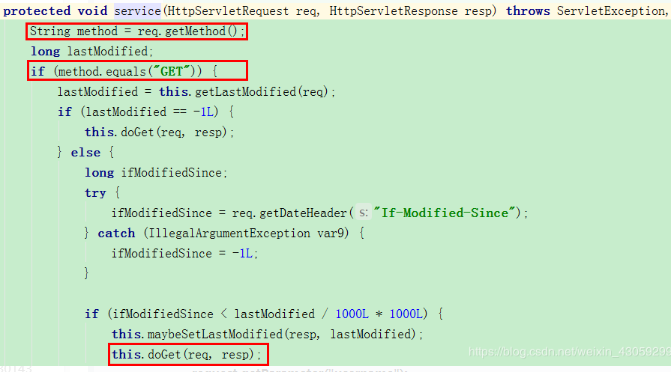
另一个是**protected修饰的service()方法**，该方法的参数是HttpServletRequest和HttpServletResponse类型的，该方法的作用是根据request请求的getMethod()方法判断请求的是的方式是get、post类型，然后重写相应的doGet()、doPost()方法，**该方法是重载的方法**。

**平时继承HttpServlet类的Servlet类重写的是protected修饰的方法，然后根据要处理的请求的类型，来重写不同的方法doGet()、doPost()方法**。

[Servlet有关的图片\HttpServlet.png](file:///F:\1.java笔记（和盈）\4.javaEE基础\Servlet有关的图片\HttpServlet.png)

部分源码如下：





## 九、HttpServletRequest

HttpServletRequest是客户端给服务器发送的请求报文。该对象由Tomcat服务器创建，最终作为参数传递到doGet或doPost方法中。Servlet容器对于接受到的每一个Http请求，都会创建一个ServletRequest对象，并把这个对象传递给Servlet的service( )方法。所有和请求有关的内容封装在请求对象中。

### 1. 常用的方法

**①获取用户表单提交的数据**

request**.getParameter(String name)**：括号内的参数与表单元素的name一致，返回的是String类型的数据。

request**.getParameterValues(String name)：返回的是String[]类型的数据。**

注意用上面的方法获取的单选框、多选框和下拉列表的数据是value属性值，而不是标签之间的文本值。

考虑怎么在Servlet类中获得获得单选框选中的文本值而不是value值。

**②请求的转发**：**这是服务器跳转**

**request.getRequestDispatcher(String url).forward(request, response);**

**③**request**.getMethod()：**返回值为String类型的，获得Http请求的方式，一般为post、get。

**④**request**.getHeader(String name)**：获得请求头，返回值为String类型。

**⑤**request**.getServerName()**：返回值为String类型的数据，返回的是发送请求的服务器的主机名。

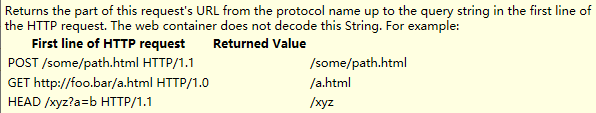
**⑥**req**.getScheme()**：返回值为String类型的数据，返回的是发送请求所采用的协议，如http、https。

**⑦**req**.getServerPort()**：返回值为int类型，返回的是发送请求的服务器的端口号。

**⑧**req**.getServletPath()**：返回值为String类型，返回的是Request URL中调用Servlet类的部分。**就是处理请求的Servlet类的url路径，即配置Servlet的url-pattern中的内容(包括斜线)**。

**⑨**req**.getContextPath()**：返回值为String类型。返回的是Request URL中表示请求内容的部分。**就是项目名称**。

**⑩**req**.getRequestURI()**：返回值为String类型，**获取的是从项目名称开始的资源路径名称**。即端口后面的内容，如/mvcProject/login.jsp。



**⑩**req**.getRequestURL()**：返回值为StringBuffer类型。返回的是Request URL。**得到的是全路径**。



在网页的开发者模式中点击NetWork可以看到Request URL，上面代码的url为：。

**Http请求有：请求行、请求头、请求体。**详细介绍在Servlet文件夹中的Http请求的图片中。[Servlet有关的图片\Http请求.png](file:///F:\1.java笔记（和盈）\4.javaEE基础\Servlet有关的图片\Http请求.png)

**get提交的请求参数在url地址的后面。post提交的请求参数在的请求体中。**

## 十、HttpServletResponse

HttpServletResponse是服务器给客户端发送的响应报文。该对象由Tomcat服务器创建，最终作为参数传递到doGet或doPost方法中。

### 1.常用的方法

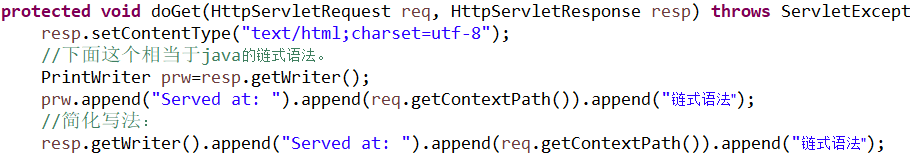
**⑴通过response的输出流输出动态页面内容。**

response.**getWriter()**：该方法返回的是一个PrintWriter对象，默认情况下，PrintWriter对象使用ISO-8859-1编码（该编码在输入中文时会发生乱码）**。**

PrintWriter**.writer(String s)：**参数为要向浏览器页面显示的内容。

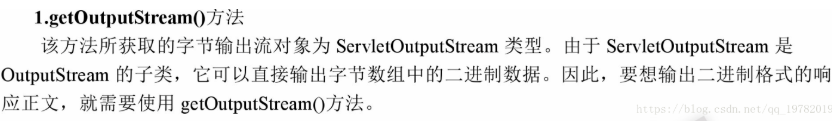
注意最后面要写PrintWriter**.flush()**、PrintWriter**.close()**。

PrintWriter有个append(CharSequence csq)方法， 可以写成下面那样，这就相当于java的链式语法。下面这部分内容可以直接显示在页面上。





response**.getOutputStream()**：获得字节流，通过该字节流的write(byte[] bytes)可以向response缓冲区中写入字节，再由Tomcat服务器将字节内容组成Http响应返回给浏览器。



**注意：虽然response对象的getOutSream()和getWriter()方法都可以发送响应消息体，但是他们之间相互排斥，不可以同时使用，否则会发生异常。**

**⑵响应的重定向**：即**将浏览器跳转到指定的位置，这是客户端跳转**。

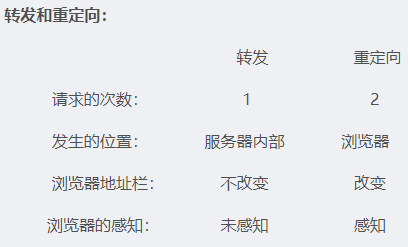
**response.sendRedirect(String url);**

**⑶**response**.setCharacterEncoding(String charset)**：设置响应的编码格式。一般写成response.setCharacterEncoding(“utf-8”)。

**⑷**response**.setHeader(String name,String value)**：设置给定名称和值的响应头。一般可以写成这样resp.setHeader("Content-type", "text/html;charset=utf-8")。若是普通文本，则是“text/plain”。

**⑸**response**.setContentType(String type)**：设置发送给客户的响应内容的类型。一般写成response.setContextType(“text/html;charset =utf-8”)。

**⑹**response.**addCookie(Cookie cookie)**：给响应内容添加Cookie。



**Http响应有：响应行、响应头、响应体。**详细介绍在Servlet文件夹中的Http响应的图片中。[Servlet有关的图片\Http响应.png](file:///F:\1.java笔记（和盈）\4.javaEE基础\Servlet有关的图片\Http响应.png)

## 十一、客户端与服务器之间的乱码问题

### 1.编码和解码

浏览器和服务器之间通信时，中文内容不能直接发送，需要对中文进行编码。

**编码**：将字符转换为二进制码的过程叫编码。

**解码**： 将二进制码转换为普通字符的过程叫解码。

编码和解码所采用的规则我们称为字符集。常见的字符集：ASCII、ISO8859-1、GBK、GB2312、UTF-8。

中文属于Unicode编码，占4个字节，英文属于ASCII编码，占2个字节。

产生乱码的根本原因是编码和解码的所采用的格式不一样。

### 2.产生乱码的两种情况

**⑴**用户通过表单向Servlet发送中文请求时，Servlet获取到内容会产生乱码。

**⑵**Servlet向浏览器响应中文内容时，也会产生乱码。

解决乱码问题：将编码和解码的格式统一为utf-8

### 3.具体解决乱码问题

#### ⑴请求编码和服务器解码

**①请求编码**：请求是浏览器发送给服务器的，浏览器会自动使用网页的字符集对参数进行编码。故统一网页的字符集为UTF-8即可。

**②服务器解码**：tomcat7以上的版本，对于path的内容，会自动用utf-8去解码，也可以在server.xml中去配置。而tomcat7以及以下的版本，默认的字符集依然是iso-8859-1。

**针对post请求：** request解码时默认字符集时iso8859-1，但iso不支持中文，故post请求在servlet中解码需要指定request的字符集。采用如下方法，来设置request的字符集：

**request.setCharacterEncoding("utf-8");**

**特别注意该方法要在request.getParameter()第一次调用之前调用。**

**针对get请求：**get请求是通过url地址传递请求参数，url中的请求参数将会被Tomcat服务器自动解码，若Tomcat的版本在7以下，则其默认编码是iso8859-1，因此需要修改Tomcat的解码的默认字符集，修改配置文件server.xml。否则可以不用更改Tomcat的解码方式。在server.xml的Connector标签中（改端口号的那个标签）添加如下属性：

**URIEncoding="utf-8"**



**特别注意修改完配置文件以后，get请求的编码就不用再处理了，但若是post请求还是需要用post请求中的设置解码方式。**

#### ⑵响应编码和浏览器解码

**①响应编码：**响应是服务器发送给浏览器，需要指定服务器的编码字符集为UTF-8。指定response的字符集的方式如下：

**response.setCharacterEncoding("utf-8");**

**该方法必须在获取输出流之前设置响应编码。**

**②浏览器解码：**虽然已经指定了response的字符集为utf-8，但浏览器默认使用GB2312解码的，所以依然乱码。因此需要设置浏览器的解码字符集。

法一：通过浏览器来设置，这种方式不靠谱。

法二：可通过服务器来告诉浏览器，我们的内容的编码格式为utf-8。就是通过一个响应头来告诉浏览器，内容的编码格式为：**“Content-Type:text/html;charset=utf-8”。**通过response的方法，来设置响应头：

**response.setHeader("Content-Type","text/html;charset=utf-8");**

针对设置服务器的响应编码设置和浏览器的解码设置，可以直接使用下面这种方法，下面这种方式就包含了上面的响应编码和浏览器的解码设置：

**response.setContentType("text/html;charset=utf-8")；**



### 4.解决乱码小结

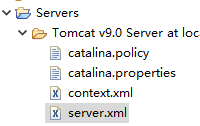
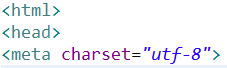
针对post请求，在request.getParameter()方法第一次调用之前，设置编码格式为：**request.setCharacterEncoding("utf-8")。**

针对get请求，修改server.xml配置文件中Connector标签中的编码为**URIEncoding="utf-8"。**注意这个更改如果没有修改Eclipse中tomcat的server Locations的内容(即双击tomcat修改)，使用的workspace是Eclipse的metadata，则应该更改的是Eclipse中项目那一栏的Servers文件夹中server.xml中添加编码格式，而不是在tomcat软件包中的config中修改server.xml。如果将Eclipse中的tomcat的server Locations改为了电脑安装的tomcat，则可以直接在Tomcat的config中更改server.xml中的内容。

响应过程，设置一个Content-Type响应头：

**response.setContentType("text/html;charset=utf-8")；这个只有响应的内容通过输出流输出到页面上该设置才有用。**

**html中的编码格式、(get提交)server.xml中的编码、(post提交)Servlet类中的编码、响应的编码格式都尽量要一致。要么都是gbk、要么都是utf-8。最好是utf-8。**









## 十二、路径问题

**1.URL和URI：**URI有两种实现方式URL和URN，其中URN用的很少。URL是URI的一种实现，也是URI最常见的实现方式。

**URL地址的格式**：**http://主机名:端口号/项目名/资源名**。

**2.相对路径：**指的是相对于当前资源所在路径。**不用/开头，用指定的路径替换当前路径最后一个/后面的部分。**

相对路径会经常发生变化，容易出现错误的链接。故在开发中一般使用绝对路径。

**3.绝对路径：**绝对路径使用**/**开头表示，绝对路径的形式一般是下面这样

**http://主机名:端口号/项目名/资源名称**

如果访问同一个项目内的资源，绝对路径用/开头：

**①服务器端路径：web.xml中配置的url-pattern路径、注解@WebServlet(String url)中的路径、服务器端跳转(请求转发)的路径request.getRequestDispatcher(String url).forward(request, response)。**

用绝对路径表示：/开头，路径url只需要写**/资源名称**，因为这里的斜线/已经表示前面的路径是**http://主机名:端口号/项目名**。因此**只需补充资源路径名即可**。

**②客户端路径：form表单中的路径(即action属性中填写的路径)、链接中的路径(超链接<a>、location.href、图片<img src>、window.open()中的路径)、js中的路径、引入js或css的路径、客户端跳转(响应重定向)的路径response.sendRedirect(String url)**。

用绝对路径表示：/开头，路径url要写**/项目名称/资源名称**，因为这里的斜线/表示前面的路径是**http://主机名:端口号**。因此**需补充项目名称和资源路径名**。

注意：在确定了路径没写错的情况下，直接访问Servlet还是报404错误，可能是Servlet的问题，重新启动一下Servlet即可。

例子：









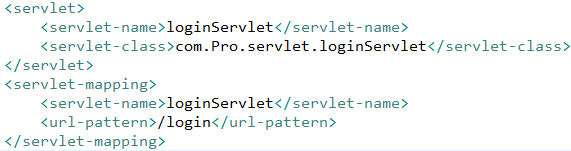
## 十三、内容补充

在Eclipse中创建html、css文件，右键点击Web项目中的WebContent文件夹，选择New一个other文件，然后找到对应的文件即可。

**每写完一个Servlet类就对它进行配置。**

### 1.例子解析：





例子分析：html文件名为hello.html，将请求内容从hello.html中的表单form中的内容通过action中的路径提交到服务器tomcat中。

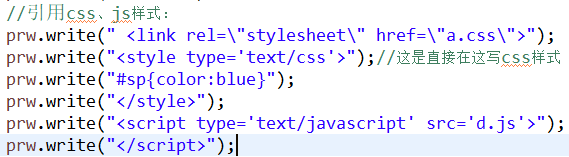
当在浏览器中输入localhost:8088/SecondServletPro(项目名称)/hello.html回车时就会立马提交请求。

之后tomcat创建loginServlet对象，并将得到的请求封装成HttpServletRequst然后作为参数传给loginServlet对象，然后通过request对象的getParameter()方法拿到表单中的内容。loginServlet对于请求作出了response，然后tomcat将该response封装成HttpservletResponse对象并通过response的getWriter()方法获得输出流，然后通过该输出流的方式发送给客户端即浏览器。

这时，浏览器中的路径变为了localhost:8088/SecondServletPro/(这个从表单文件的hello.html变成了Servlet类的配置中的url-pattern中的路径)login?account=xxx&password=xxx。

**注意form表单中的action属性内填的是用来指定处理form表单请求的Servlet类。action的内容与Servlet配置的url-pattern的斜线/后的内容一致**。

在Servlet类的输出流中引入css、js样式：



### 2.http常用的请求方式

**⑴get：提交的数据拼接在路径的后面，**如下所示：

url?name1=value1&name2=value2......

**⑵post：作为请求体的内容提交，路径看不到，**进入网页的开发者模式中的NetWork中可以看到。

**对比这两种方式**：

①get请求会把数据拼接在路径后面，直接能看到。如果提交密码等保密数据，建议使用post，有一定的安全性。

②整个路径的长度有限，get不能提交太大的数据。如果涉及文件提交等功能时只能用post。

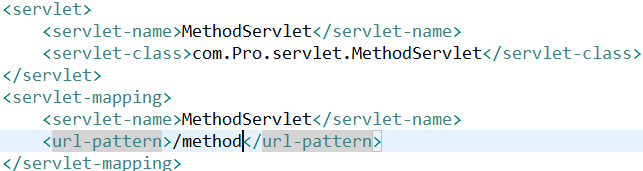
除了在表单中设置method=”post”方式，默认的请求方式都是get(包括直接输入url，链接，js等)。

注意当自己写的web项目页面显示无法访问此网站时，可能是自己的tomcat没启动就直接访问了。

### 3.页面跳转

从一个html的页面文件跳转到一个Servlet类中：<a>链接中的href和location.href以及window.open()中的内容都是同form表单的action一样，都是填指定处理请求的Servlet类的url-pattern斜线之后的内容。







### 4.表单元素的提交

**表单中每个元素一定要有name属性，因为表单提交内容是根据name来提交的，因此要想获得表单所提交的内容也要根据name属性获得**，如req.getParameter(String name)中的参数就是表单元素中的name。**表单元素中具体被提交的值则是表单元素中对应的value属性的值被提交了。name相当于提交方式，即以什么名字来提交，value相当于提交的具体值，即提交的内容是value对应的值(个人理解)。**

**下拉列表框中的name属性是在<select>中，而不是在<option>中。**

**①输入框中，如text、password等提交的内容就是输入的内容，也就是value的属性值。如果输入框中什么也没有输入，则会提交空的(不是null，而是空字符串)。**

**②单选框和多选框提交的内容则是选中的内容的value属性值，如果没有选中，则不会提交，采用方法获得的值则是null**。**当单选框或者多选框没写属性value时，若选中了，则提交的结果是on。**

**③下拉列表提交的则是选中的option对应的value属性值。**

**④注意文本域textarea提交的内容是输入文本域中的内容，也就是文本域显示在页面上的内容，而不是value属性值的内容。也就是说textarea可以不用写value属性。**

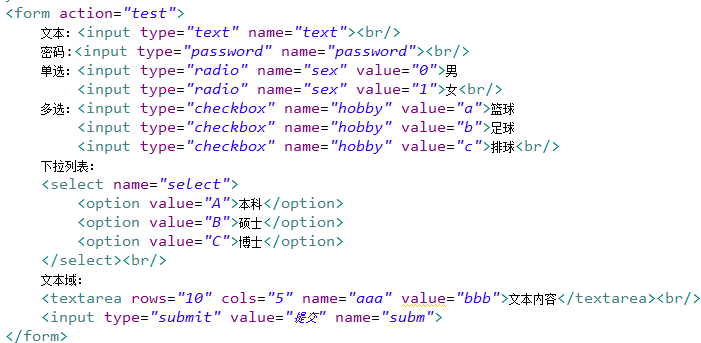
**⑤按钮如button、submit提交的是value属性值。**

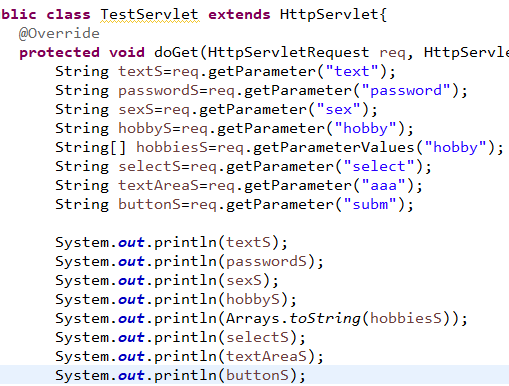
**通过request.getParameter(String name)方法得到的数据有如下几种情况**：

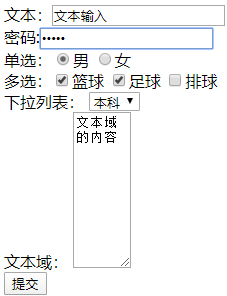
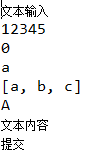
**正常的数据：提交了name属性，有了具体内容的数据**

**空字符串：提交了name属性，但是对应的value属性值是空串(文本框没有输入内容)**。

**null：没有提交name属性，如单选或多选没有任何一项选中则不会提交name属性**。



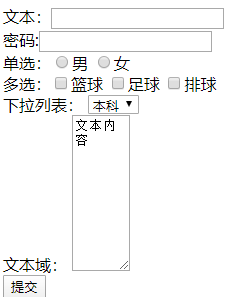
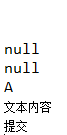


点击提交之后得到结果

文本域的内容

显示在页面上为：



点击提交之后得到结果

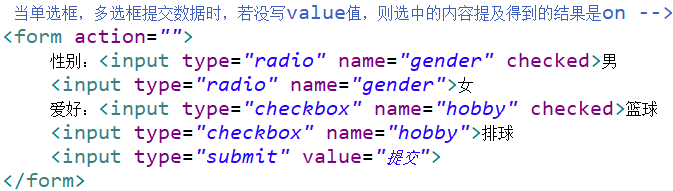
显示在页面上为：



对比上面显示在页面上的图，可以知道**当单选框或多选框没有选中任何一项，则其数据不会提交**。

**下面的单选、多选框没有写value值，当选中的内容提交之后，采用get方式提交在地址栏上显示提交的内容为on**。





### 5.连接数据库

**注意Web中项目中需要连接数据库时， JDBC中的jar包不需要BuildPath过程，只需将驱动包复制到WEB-INF目录下的lib目录中即可**，用BuildPath的方法是不行的。**WEB-INF中的lib目录就是用来存放jar包的**。

### 6.提交数据的方式：

**①通过form表单的action属性来提交数据**

**②仿造get方法提交数据的形式，将数据拼接在路径后面，可以利用<a>链接或者js中的location.href来提交数据。**

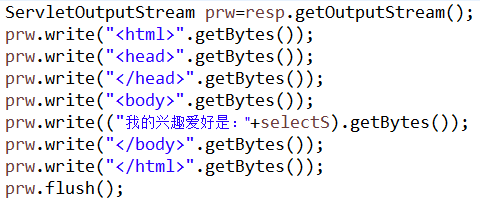
**<a href="url?name1=value1&name2=value2...."></a>**

**location.href="url?name1=value1...."**

### 7.浏览器出现500错误

浏览器出现500错误一般是服务器端代码中出现了问题。

如果tomcat服务器出现异常的信息为getWriter() has already been called for this response，可以用response.getOutputStream()方法代替getWriter()方法。该方法返回的是一个**ServletOutputStream**类型，其对应的write()方法的参数只能为byte[]或者int类型。使用完这个方法之后，再改回response.getWriter()方法则可以解决上面的异常问题，也可以直接用response.getOutputStream()方法而不使用getWriter()方法。



### 8.在Eclipse中创建html、js等文件

在Eclipse中创建html、css文件可以点击Web-Content右键New—>other—>在Web文件夹下找到相应的文件格式就可以。创建js类型的文件只能点击Web-Content右键-->New-->File，然后直接写xxx.js然后finsh就可以了，css、html也可以用这种方法创建。

### 9创建Servlet类不用配置web.xml的方法：

⑴正常生成一个Web项目

⑵右键Web项目，New-->Servlet，进入create Servlet页面，依次填写内容，然后点击next，直到点击finish完成。

**注意@WebServlet(String url)是注解：参数就是Servlet类配置中的url-pattern。**

## 十四、重定向、请求转发

**①(重定向)客户端跳转：response的方法**

**response.sendRedirect(String url)**

**注意客户端跳转路径是会发生变化。**

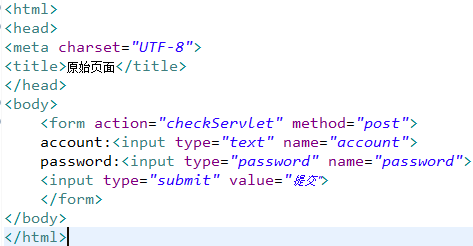
**②(请求转发)服务器跳转：request的方法**

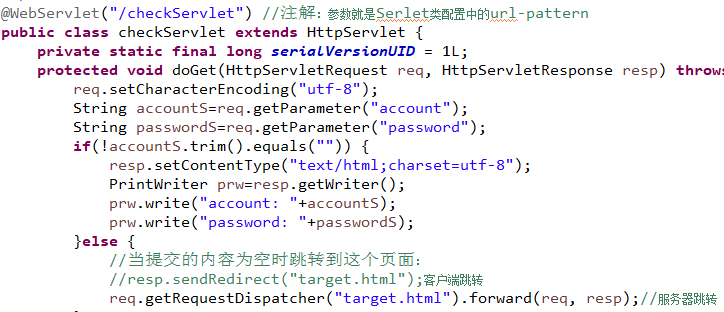
**request.getRequestDispatcher(String url).forward(request, response);**

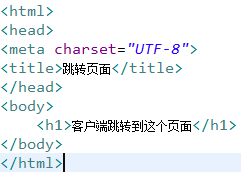
**注意服务器跳转路径是不会发生变化的。服务器跳转要特别注意不要忘了写forward()，这个一定不能漏了。**

### 1.例子：客户端跳转和服务端跳转

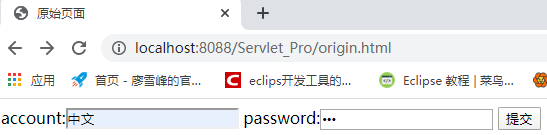
**例1：**



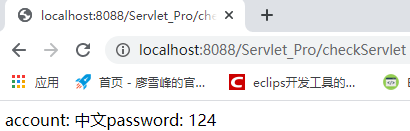




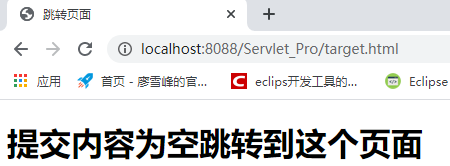
在网页中输入下面相应的地址如下页面：地址为.../origin.html

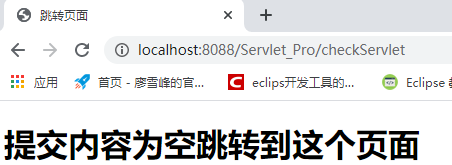


点击提交之后的页面：地址变为了..../checkServlet

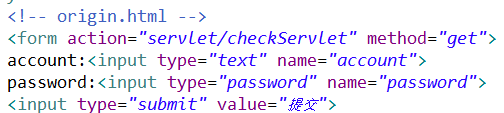


当提交的内容为空时，采用客户端跳转的方式，则网页上的地址变为：...../target.html。而采用服务器跳转的方式，网页上的地址还是：..../checkServlet。

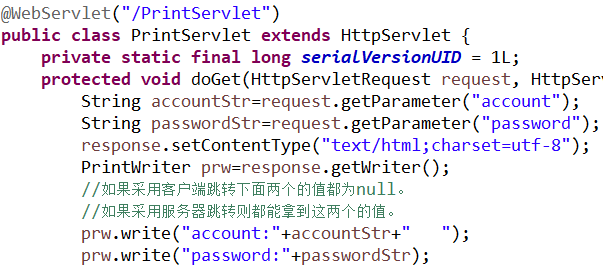




**例2**：



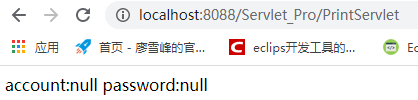


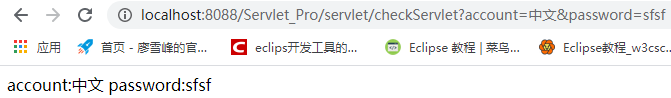




点击提交之后，下面分别是客户端和服务器跳转的到的结果：

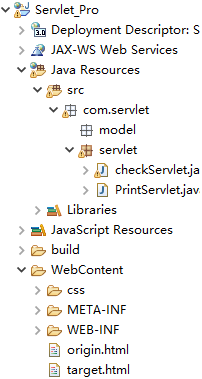
注意采用客户端跳转得到null的值，是因为客户端跳转发送的是新的请求，而新的请求中没有携带account和password以及它们的数据，所以为null。若采用服务器跳转，account和password中没有填写数据点提交，account和password的数据也会提交，只不过为空字符串，而不是null。注意这俩个结果的区别。



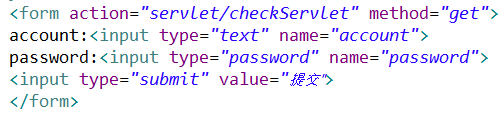


### 2.实例：服务器中跳转到服务器(好好理解)

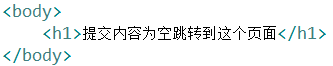
<1>包结构



2.origin.html文件：



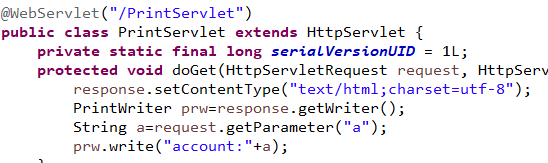
3.target.html文件：



4.checkServlet类：



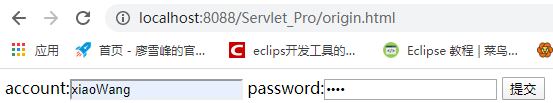
5.PrintServlet类：

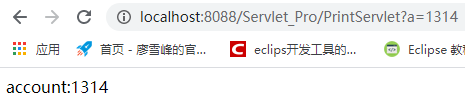


思路：从网页上输入....origin.html，如果提交的表单不为空则从checkServlet服务器中跳转到PrintServlet服务器中。如果提交的表单为空，则从checkServlet服务器中跳转到....target.html显示的页面中。注意上面checkServlet配置的路径是**/servlet/checkServlet，**而PrintServlet的路径是**/printSevlet**，从包结构中可以看出PrintServlet和CheckServlet都是在servlet包中的。

**注意下面浏览器地址栏中的内容变化。**

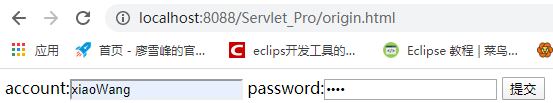
**⑴执行代码1**：将代码2-5都注释结果如下：**注意代码1中的绝对路径不能写/Servlet\_Pro/servlet/PrintServlet，这样写会找不到报404，理解这里绝对路径的写法。**

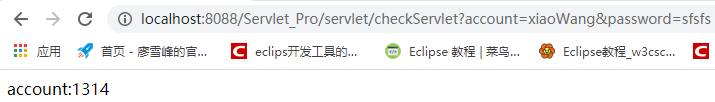




分析：采用客户端跳转的方式跳转，地址栏一开始为......origin.html，提交数据之后服务器原来的路径应该为....../servlet/checkServlet，后来变成了目标路径即....../PrintServlet。这是因为客户端跳转会改变原来的路径。同时原本拼接在跳转目标路径后面的数据如a=1314显示在了地址栏中。而account和password中的数据则没有显示在地址栏中，这是因为采用客户端跳转发送的是新请求，新的请求中没有包含account和passsword以及它们的数据。

**(2)执行代码2**：将其他if中的其他代码注释。理解这里绝对路径的写法。

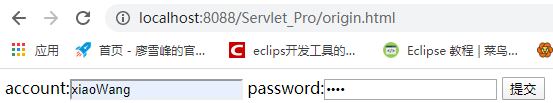


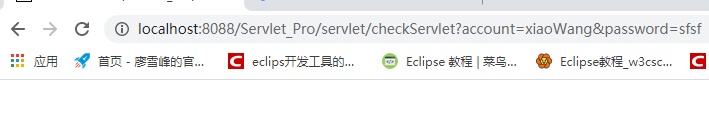


分析：采用服务器端跳转的方式跳转，地址栏一开始为......origin.html，提交数据之后服务器的地址没改变，还是原来的路径....../server/checkServlet。注意当点击提交按钮时，account和password中的数据拼接在了地址栏路径的后面。而原本拼接在跳转目标地址后面的数据a=1314则没有拼接在该地址栏的后面(注意这个数据也是成功提交了)。

代码3和代码4只是将代码1和代码2的绝对路径改为了相对路径，**注意这里的相对路径表示方法**。

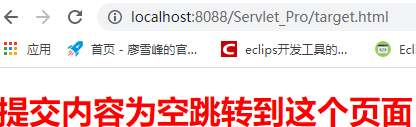
**⑶执行代码5**：得到如下结果



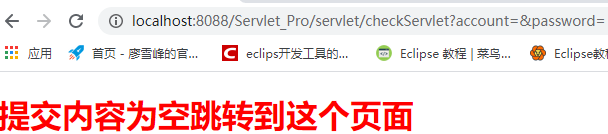


分析：代码5、2、4采用的都是服务器跳转的方式，代码5和代码2(或代码4)比较，代码5写的服务器跳转少了**.forward(request,responsee)**，得到的结果发现拼接在目标路径上的数据a=1314并没有显示在页面上(上面红框内)，而代码1～4拼接在目标路径上面的数据都显示在了页面上。这是因为服务器跳转通过**forward(request,responsee)**方法将request和response向下传递了。因此这个一定不能忘了**。**

**⑷执行代码6**：得到如下结果



执行代码7得到如下结果：



### 3.请求转发和重定向的区别

**①重定向使用的是response.sendRedirect(String url)方法，而请求转发使用的是request.getRequestDispatcher(String url).forward(request,response)方法。**

**②重定向是客户端跳转，请求转发是服务器跳转**

**③重定向过程中浏览器的url地址会改变，重定向会把原本的路径改成跳转目标的路径，而请求转发则会维持原来的路径，不会变成跳转目标的路径。**

**④重定向是客户端至少发出了两次访问的请求，请求转发是客户端只发出了一次访问的请求。**

**⑤重定向每次发出的请求都是一个新的request，request中的数据会丢失。请求转发发出一个request，在request作用域中的数据不会丢失。**

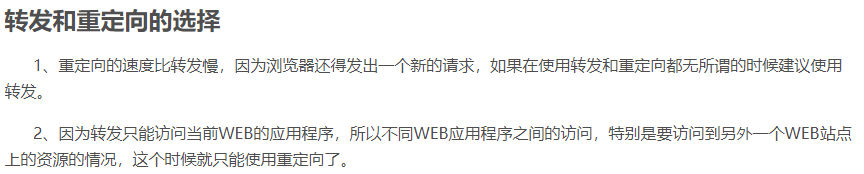
**⑥重定向和请求转发的绝对路径表示不一样，重定向要加上项目名称即/项目名称/资源名，而求转发路径直接写/资源名。**

**⑦重定向可以访问自己Web项目以外的资源，请求转发只能访问本web项目中的资源**。比如重定向跳转到百度页面，写成response.sendRedirect("http://www.baidu.com")。

**⑧重定向跳转后必须加上return，要不然页面虽然跳转了，但是还是会执行跳转后面的语句，请求转发是执行了跳转页面，下面的代码就不会执行了(这个好像有问题)。**

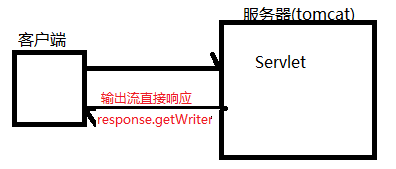
**请求转发过程：浏览器发送http请求给服务器，服务器接受请求，在内部调用方法完成请求处理和转发动作，并将目标资源发送给浏览器。（整个过程是一次请求，该请求也可以被转发n次，这n次都使用同一对request和response）**。

**重定向过程：浏览器发送http请求给服务器，服务器接受后发送302状态码和要重定向的目标地址给浏览器，浏览器发现响应头中是302，则自动给新的目标地址发送一个新的http请求，服务器接收这个请求，处理并发送资源给客户。（整个过程有两次请求，每次请求都会产生新的request和response，所以这就是将信息存放到request域中后，不能使用重定向的原因）。**

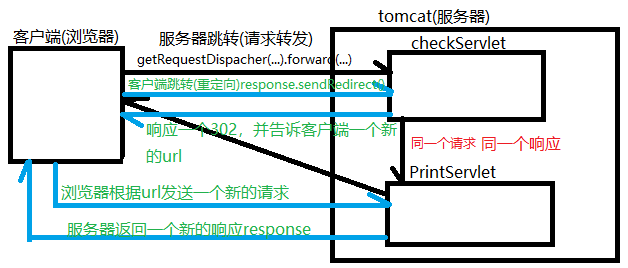


### 4.Servlet三种响应方式：

**①response.getWriter()输出流直接响应**：



**②客户端跳转(重定向)和③服务器跳转(请求转发)的视图**：



注意要定期清理浏览器中的cookie中的内容，不然在浏览器中进行访问时可能会使用cookie中缓冲的内容，而不是自己更新之后要访问的内容。

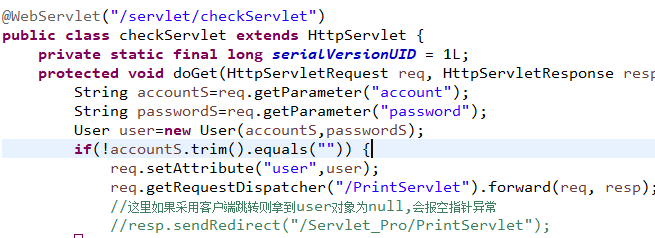
## 十五、作用域

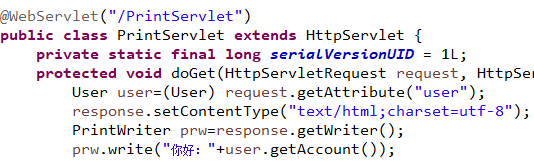
向作用域中存放数据：request.**setAttribute(String name，Object o)**，如果作用域中已经存在该name，若继续存放该name的数据则原数据会被替换。

向作用域中取数据：request.**getAttribute(String name)**，返回值为Object类型。request**.removeAttribute(String name)**：移除属性。

### 1.(HttpServletRequest)request

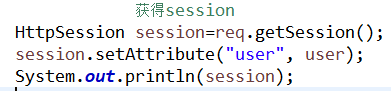
**获取request作用域中的数据必须是同一个请求，即将数据放在request作用域中，必须用服务器跳转，若使用客户端跳转拿到的是null，拿不到想要的值**。注意客户端(浏览器)每发送一个新请求，则request作用域中存放的数据则是一个新数据。如不断刷新浏览器则是不断发送新请求。





### 2.(HttpSession)session

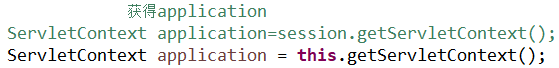
从客户端请求服务器开始到客户端关闭为止，浏览器不关闭都是使用的同一个session对象。**存放在session中的数据即使使用客户端跳转也可以拿到**。将浏览器关闭再重新打开或使用另一个浏览器打开则是另一个session。**一般只把登录用户的信息放在session中**。手动销毁session对象，调用invalidate()方法。移除session中存放的属性，调用removeAttribute()方法。





### 3(ServletContext)application

**application为全局作用域，匹配整个项目**。只要项目的代码没变则是同一个application，即使浏览器关闭了重新打开也是同一个application对象。若项目的代码改变了或重启了则是另一个application对象。**和具体用户挂钩的数据不能放在application中**。和整个项目有关的数据放在这里。





用request.getParameter()和request.getParameterValues()方法拿值和getAttribute()拿值的区别：只有放在作用域中用setAttribute()设置的值才能用getAttribute()方法拿，其余的都是用getParameter()方法取值。