## 2.HTTP-浏览器与服务器的交互交流

请求：客户端根据用户的地址信息将数据发送给服务端的过程

相应：服务器将处理好的结果返回给浏览器的过程

## HTTP协议

### 3.1 HTTP协议的概念

HTTP协议(Hypertext Transfer Protocol,超文本传输协议) ,是-个客户端请求和响应的标准协议,这个

协议详细规定了浏览器和万维网服务器之间互相通信的规则。用户输入地址和端口号之后就可以从服务器上取得所

需要的网页信息。

通信规则规定了客户端发送给服务器的内容格式，也规定了服务器发送给客户端的内容格式。客户端发送给

服务器的格式叫"请求协议":服务器发送给客户端的格式叫"响应协议”。

### 3.2. HTTP协议的特点

1.支持客户/服务器模式。

2.简单快速:客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET. POST。 每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小， 因而通信速度很快。

3.灵活: HTTP允许传输任意类型的数据对象。传输的类型由Content-Type加以标记。

4.无连接:无连接是表示每次连接只处理一个请求。 服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

HTTP1.1版本后支持可持续连接。通过这种连接就有可能在建立-一个TCP连接后，发送请求并得到回应，然后发送更多的请求并得到更多的回应，通过把建立和释放TCP连接的开销分摊到多个请求上，则对于每个请求而言，由于TCP而造成的相对开销被大大地降低了。而且还可以发送流水线请求也就是说在发送请求1之后的回应到来之前就可以发送请求2.也可以认为, -次连接发送多个请求，由客户机确认是否关闭连接，而服务器会认为这些请求分别来自不同的客户端。

5.无状态: HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面。在服务器不需要先前信息时它的应答就较快。

### 3.3. HTTP之URL

HTTP (超文本传输协议)是一个基于请求与响应模式的、应用层的协议，常基于TCP的连接方式,绝大多数

的Web开发,都是构建在HTTP协议之上的Web应用。

HTTP URL (URL是- -种特殊类型的URI，包含了用于查找某个资源的足够的信息)的格式如下:

|  |
| --- |
| http://host[:port]/ [abc\_ path]  http://IP(主机名/域名) :端口/访问的资源路径 |

●http表示要通过HTTP协议来定位网络资源;

●host表示合法的Internet主机域名或者IP地址;

●port 指定-个端口号, 为空则使用缺省端口80;

●abs. path指定请求资源的URI;如果URL中没有给出abs, path,那么当它作为请求URI时，必须以"/"的形式给出，通常这个工作浏览器自动帮我们完成。

### 3.4. HTTP请求

HTTP请求由三部分组成，分别是:请求行、请求头、请求正文。

通过chrome浏览器，F12 - > Network查看。

格式:

请求行

请求头1

请求头2

请求空行

请求体

请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本。

格式如下: Method Request-URl HTTP-Version CRLF

Method表示请求方法:

Request-URI是一个统- 资源标识符;

HTTP-Version表示请求的HTTP协议版本;

CRLF表示回车和换行;

1. Get请求(没有请求体)



1. Post请求

### 3.5. HTTP响应

在接收和解释请求消息后，服务器返回-个HTTP响应消息。HTTP 响应也是由三个部分组成，分别是:状态行、消息报头、响应正文。



格式:

状态行

响应头1

响应头2

...

响应空行

响应体

### 3.6.消息头

HTTP消息由客户端到服务器的请求和服务器到客户端的响应组成。请求消息和响应消息都是由开始行(对于请求消息，开始行就是请求行，对于响应消息,开始行就是状态行)，消息报头(可选) , 空行(只有CRLF的行)，消息正文(可选)组成。

每一个报头域都是由名字+\*.\*+空格+值组成，消息报头域的名字是大小写无关的.

请求头

请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。

●Referer: 该请求头指明请求从哪里来。

如果是地址栏中输入地址访问的都没有该请求头地址栏输入地址，通过请求可以看到，此时多了一一个Referer的请求头，并且后面的值为该请求从哪里发出。比如:百度竞价，只能从百度来的才有效果，否则不算;通常用来做统计工作、防盗链。

响应头

响应报头允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对Request-URI所标识的资源进行下一步访问的信息。

Location: Location响应报头域用于重定向接受者到一个新的位置。

Location响应报头域，常用在更换域名的时候。

response. sendRedi rect("http://www. baidu. com");

Refresh: 自动跳转(单位是秒).可以在 页面通过meta标签实现，也可在后台实现。

<meta http- equiv=" refresh" content=" 3 ; ur1=http://www. baidu. com">

## 4 Tomcat服务器

请求：

客户端发送数据服务端的过程

响应：

服务端返回数据给客户端的过程

### 4.1.什么是Tomcat

Tomcat是- -个符合JavaEE WEB标准的最小的WEB容器，所有的JSP程序-定要有WEB容器的支持才能运行，而且在给定的WEB容器里面都会支持事务处理操作。

### 4.2.Tomcat的安装与启动

### 4.3.Tomcat的目录结构

1. bin: 启动和关闭tomcat的脚本文件（bat文件）

2. conf: 配置文件server.xml该文件用于配置server相关的信息，比如tomcat启动的端口号， 配置主机.(Host); web.xml 文件配置与web应用(web 应用相当于-个web站点) ; tomcat user.xml配置用户名密码和相关权限

3. lib: 该目录放置运行tomcat运行需要的jar包

4. logs: 存放日志，当我们需要查看日志的时候,可以查询信息

5.temp: 存放Tomcat运行时产生的临时文件

6.webapps: web应用所在的目录.即外界访问的web资源的存放目录

7.work: Tomca的工作目录，该目录用于存放jsp被访问后生成对应的server文件和.class文件

### 4.4. ID EA配置Tomcat

1.点击settings, 选择Build, Execution, Deployment，

然后选择"Aplction Servers"，点击右侧的"+"号，选择"Tomcat Server"

## 5. Servlet的实现

Servlet是Server与Applet的缩写，是服务端小程序的意思。使用Java语言编写的服务器端程序,可以像生成动态的WEB页，Servlet 主要运行在服务器端，井由服务器调用执行，是一 种按照Servlet标准来开发的类。是SUN公司提供的一门用于开发动态Web资源的技术。(言外之意: 要实现web开发，需要实现Servlet标准)

Servlet本质上也是Java类，但要遵循Servlet规范进行编写，没有main()方法，它的创建、使用、销毁都由Servlet容器进行管理(如Tomcat)。(言外之意: 写自己的类，不用写main方法，别人自动调用)

Servlet是和HTTP协议是紧密联系的，其可以处理HTTP协议相关的所有内容。这也是Servlet应用广泛的原因之一。

提供了Servlet功能的服务器，叫做Servlet容器，其常见容器有很多，如Tomcat, Jtty,, WebLogic Server,WebSphere, JBoss等等。

### 5.1创建web项目

### 5.2 tomcat设置及@webservelt注解

|  |
| --- |
| @WebServlet(**"/ser01"**) *//@WebServlet(name="servlet01", value = "/ser01") //@WebServlet(name="servlet02", value = {"/ser01","/ser001"}) //@WebServlet(name="servlet01", urlPatterns = "/ser01") //@WebServlet(name="servlet02", urlPatterns = {"/ser01","/ser001"})* |

### 5.3 servlet的实现方式

1.继承HttpServlet

2.继承GenericServlet

3.实现Servlet

4.重写 doGet和doPost方法

### 5.4. Servlet的生命周期

Servlet没有main()方法，不能独立运行，它的运行完全由Servlet引学来控制和调度。所谓生命周期， 指的是servlet容器何时创建servlet实例、何时调用其方法进行请求的处理、何时并 销毁其实例的整个过程。

●实例和初始化时机

当请求到达容器时，容器查找该servlet对象是否存在，如果不存在,则会创建实例并进行初始化。

●就绪/调用/服务阶段

有请求到达容器，容器调用servlet对象的service()方法，处理请求的方法在整个生命周期中可以被多次调用;HttpServlet的service()方法，会依据请求方式来调用doGet)或者doPost()方法。但是，这两个 do方法默认情况下，会抛出异常，需要子类去override.

●销毁时机

当容器关闭时(应用程序停止时)， 会将程序中的Servlet实例进行销毁。

上述的生命周期可以通过Servlet中的生命周期方法来观察。在Servlet中有三个生命周期方法。不需要用户手动调用，而是在特定的时机有容器自动调用,观察这三个生命周期方法即可观察到Servlet的生命周期。

Servlet的生命周期，简单的概括这就分为四步: servlet 类加载->实例化-->服务-->销毁。

init方法,在Servlet实例创建之后执行(证明该Servlet有实例创建了)

service方法，每次有请求到达某个Servlet方法时执行，用来处理请求(证明该Servlet 进行服务了)

destroy方法，Servlet 实例销毁时执行(证明该Servlet的实例被销毁了)

Servlet的工作流程：

1. Web Client向Servlet容器(Tomcat) 发出Http请求

2. Servlet容器接收Web Client的请求

3. Servlet容器创建一个HttpServletRequest对象,将Web Clent请求的信息封装到这个对象中

4. Servlet容器创建一个HttpServletResponse对象

5. Servlet容器调HttpServlet对象service方法，把Request与Response作为参数，传给HttpServlet

6. HttpServlet调用HttpServletRequest对象的有关方法，获取Http请求信息

## 6. HttpServletRequest对象

HttpServletRequest对象:主要作用是用来接收客户端发送过来的请求信息,例如:请求的参数,发送的头信息等都属于客户端发来的信息，service0)方法中形 参接收的是HttpServletRequest接口的实例化对象，表示该对象主要应用在JUTTP协议上，该对象是由Tomcat封装好传递过来。

HttpServletRequest是ServletRequest的子接口，ServletRequest 只有一个子接口，就是

HttpServletRequest。既然只有一个子接口为什么不将两个接口合并为一个?

从长远上讲:现在主要用的协议是HTTP协议,但以后可能出现更多新的协议。若以后想要支持这种新协议，只需要直接继承ServletRequest接口就行了。

在HttpServletRequest接口中，定义的方法很多，但都是围绕接收客户端参数的。但是怎么拿到该对象呢?

不需要，直接在Service方法中由容器传入过来,而我们需要做的就是取出对象中的数据，进行分析、处理。

### 6.1.接收请求

6.1.1.常用方法

1.方法

getRequestURL()

获取客户端发出请求时的完整URL

getRequestURI

获取请求行中的资源名称部分(项目名称开始)

getQueryString()

获取请求行中的参数部分

getMethod()

获取客户端请求方式

getProtocol)

获取HTTP版本号

getContextPath()

获取webapp名字

6.1.2.获取请求参数

1.方法

getParameter(name)

获取指定名称的参数

getParameterValus(String name)

获取指定名称参数的所有值

### 6.2.请求乱码问题

由于现在的request属于接收客户端的参数,所以必然有其默认的语言编码，主要是由于在解析过程中默认使用的编码方式为ISO 8859-1(此编码不支持中文),所以解析时一定会出现乱码。要想解决这种乱码问题，需要设置request中的编码方式,告诉服务器以何种方式来解析数据。或者在接收到乱码数据以后，再通过相应的编码格式还原。

方式一:

request. setCharacterEncodingC"UTF-8");

这种方式只针对POST有效(必须在接收所有的数据之前设定)

方式二:

new stringCrequest . getParameter(name) . getBytes(" I50-8859-1") ,"UTF-8");

借助了String对象的方法，该种方式对任何请求有效，是通用的。

Tomcat8起，以后的GET方式请求是不会出现乱码的。

### 6.3.请求转发

请求转发，是一种服务器的行为，当客户端请求到达后，服务器进行转发,此时会将请求对象进行保存地址栏中的URL地址不会改变,得到响应后，服务器端再将响应发送给客户端，从始至终只有一一个请求发出。

实现方式如下，达到多个资源协同响应的效果。

request. getRequestDispatcher(ur1) . forward(request, response) ;

### 6.4. request作用域

通过该对象可以在一次请求中传递数据，作用范围:在一次请求中有效，即服务器跳转有效。

//设置域对象内容

request . setAttribute(string name, Object value);

//获取域对象内容

request . getAttribute(string name);

//删除域对象内容

request . removeAttribute(string name) ;

request域对象中的数据在一次请求中有效,则经过请求转发, request 域中的数据依然存在，则在请求转发的过程中可以通过request来传输/共享数据。

## 7. HttpServletResponse对象

Web服务器收到客户端的http请求，会针对每一次请求， 分别创建一个用于代表请求的 request对象和代表响应的response对象。

request和response对象代表请求和响应：获取客户端数据，需要通过request对象；向客户端输出数据，需要通过response对象。

HttpServletResponse的主要功能用于服务器对客户端的请求进行响应，将Web服务器处理后的结果返回给客户端。service(方法中形参接收的是HttpServletResponse接口的实例化对象，这个对象中封装了向客户端发送数据、发送响应头，发送响应状态码的方法。|

### 7.1.响应数据

接收到客户端请求后，可以通过HttpsServletResponse对象直接进行响应,响应时需要获取输出流。

有两种形式: .

getWriter()获取字符流(只能响应回字符)

getOutputStream()获取字节流(能响应一切数据)

响应回的数据到客户端被浏览器解析。

注意:两者不能同时使用。

### 7.2.响应乱码问题

在响应中，如果我们响应的内容中含有中文,则有可能出现乱码。这是因为服务器响应的数据也会经过网络传输，服务器端有一种编码方式， 在客户端也存在一种编码方式, 当两端使用的编码方式不同时则出现乱码。

getWriter()的字符乱码

对于getWriter()获取到的字符流，响应中文必定出乱码，由于服务器端在进行编码时默认会使用ISO-8859-1格式的编码，该编码方式并不支持中文。

要解决该种乱码只能在服务器端告知服务器使用一种能够支持中文的编码格式，比如我们通常用的"UTF-8"。

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

此时还只完成了一半的工作, 要保证数据正确显示，还需要指定客户端的解码方式。

|  |
| --- |
| response.setHeader(**"content-type"**,**"text/html;CharsetEncoder=utf-8"**); |

两端指定编码后,乱码就解决了，-句话:保证发送端和接收端的编码一致

以上两端编码的指定也可以使用一句替代，同时指定服务器和客户端

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

|  |
| --- |
| *//设置服务端的响应编码格式* resp.setCharacterEncoding(**"utf-8"**); *//设置客户端的响应编码格式* resp.setHeader(**"content-type"**,**"text/html;CharsetEncoder=utf-8"**); *//同时设置服务端和客户端的响应编码格式* resp.setContentType(**"text/html;charset=utf-8"**); |

getOutputStream()字节乱码

对于getOutputStream()方式获取到的字节流，响应中文时，由于本身就是传输的字节，所以此时可能出现乱码，也可能正确显示。当服务器端给的字节恰好和客户端使用的编码方式一致时则文本正确显示， 否则出现乱码。无论如何我们都应该准确掌握服务器和客户端使用的是那种编码格式，以确保数据正确显示。

指定客户端和服务器使用的编码方式一致。

|  |
| --- |
| response.setHeader(**"content-type"**,**"text/html;Charset=utf-8"**); |

### 7.3.重定向

重定向是一种服务器指导，客户端的行为。客户端发出第一一个请求， 被服务器接收处理后，服务器会进行响应，在响应的同时，服务器会给客户端一个新的地址(下次请求的地址response.sendRedirect(ur);)，当客户端接收到响应后，会立刻、马上、自动根据服务器给的新地址发起第二二个请求，服务器接收请求并作出响应，重定向完成。

从描述中可以看出重定向当中有两个请求存在，并且属于客户端行为。

//重定向跳转到index .jsp

response. sendRedirect("index. jsp");

通过观察浏览器我们发现第一次请求获得的响应码为302,并且含有一个location头信息。并且地址栏最终

看到的地址是和第一次请求地址不同的， 地址栏已经发生了变化。

### 7.4.请求转发与重定向的区别

请求转发和重定向比较:

|  |  |
| --- | --- |
| 请求转发req.getRequestDispatcher().forward() | 重定向(resp.sendRedirect() ) |
| 一次请求，数据在request域中共享 | 两次请求，request 域中数据不共享 |
| 服务器端行为 | 客户端行为 |
| 地址栏不发生变化 | 地址栏发生变化 |
| 绝对地址定位到站点后 | 绝对地址可写到http:// |

两者都可进行跳转，根据实际需求选取即可。

## 8. Cookie对象

Cookie是浏览器提供的一种技术，通过服务器的程序能将一些只须保存在客户端， 或者在客户端进行处理的数据，放在本地的计算机上,不需要通过网络传输，因而提高网页处理的效率，并且能够减少服务器的负载，但是由于Cookie是服务器端保存在客户端的信息，所以其安全性也是很差的。例如常见的记住密码则可以通过Cookie来实现。

有一个专门操作Cookie的类javax.servlet.http.Cookie.随着服务器端的响应发送给客户端，保存在浏览器。当下次再访问服务器时把Cookie再带回服务器。

Cookie的格式：键值对用"="链接，多个键值对间通过";“隔开。

### 8.1. Cookie的创建的发送

通过new Cookie("key'"value' ):来创建一个Cookie对象,要想将Cookie随响应发送到客户端，需要先添加到response对象中，response.addookie(cookie);此时该 cookie对象则随着响应发送至了客户端。在浏览器上可以看见。

//创建Cookie对象

Cookie cookie = new CookieC"uname"，“zhangsan");

//发送Cookie对象

response. addcookie(cookie);

### 8.2. cgokie的获取

在服务器端只提供了一一个getCookies()的方法用来获取客户端回传的所有cookie组成的一一个数组，如果需要获取单个cookie则需要通过遍历，getName()获取 Cookie的名称，getValue()获取 Cookie的值。

|  |
| --- |
| Cookie[] cookie=req.getCookies(); **if**(cookie!=**null**&& cookie.**length**>=0){  **for**(Cookie cookies:cookie){  String name=cookies.getName();  String value=cookies.getValue();  System.***out***.println(**"名称："**+name+**"\t值："**+value);  } } |

### 8.3. Cookie设置到期时间

除了Cookie的名称和内容外，我们还需要关心一个信息，到期时间，到期时间用来指定该cookie何时失效。默认为当前浏览器关闭即失效。我们可以手动设定cookie的有效时间(通过到期时间计算)， 通过setMaxAge(int time);方法设定cookie的最大有效时间，以秒为单位。

到期时间的取值

●负整数

若为负数，表示不存储该cookie.

cookie的maxAge属性的默认值就是-1,表示只在浏览器内存中存活，一旦关闭浏览器窗口,那么cookie 就会消失。

●正整数

若大于0的整数，表示存储的秒数。

表示cookie对象可存活指定的秒数。当生命大于0时，浏览器会把Cookie保存到硬盘上,就算关闭浏览器，就算重启客户端电脑，cookie 也会存活相应的时间。

●零

若为0,表示删除该cookie.

cookie生命等于0是-个特殊的值，它表示cookie被作废!也就是说，如果原来浏览器已经保存了这个

Cookie,那么可以通过Cookie的setMaxAge(0)来删除这个Cookie.无论是在浏览器内存中，还是在客户端硬盘上都会删除这个Cookie.

设置Cookie对象指定时间后失效

|  |
| --- |
| *//到期时间：负整数（默认值-1，表示在浏览器内存中存活，关闭浏览器失效）* Cookie cookie1=**new** Cookie(**"name1"**,**"zhangsan"**); cookie1.setMaxAge(-1); resp.addCookie(cookie1);  *//到期时间：整数（表示存活指定秒数，会将数据存在磁盘中）* Cookie cookie2=**new** Cookie(**"name2"**,**"lisi"**); cookie2.setMaxAge(60); resp.addCookie(cookie2);  *//到期时间：零（表示删除cookie）* Cookie cookie3=**new** Cookie(**"name3"**,**"wangwu"**); cookie3.setMaxAge(0); resp.addCookie(cookie3); |

### 8.4. Cookie的注意点

1. Cookie保存在当前浏览器中。

在一般的站点中常常有记住用户名这样一个操作, 该操作只是将信息保存在本机上,换电脑以后这些信息就无效了。而且cookie还不能跨浏览器。

2. Cookie存中文问题

Cookie中不能出现中文,如果有中文则通过URLEncoder.encode()来进行编码，获取时通过

URLDecoder .decode0)来进行解码。

|  |
| --- |
| String name=**"姓名"**; String value=**"张三"**;  *//将中文通过URLEncoder进行编码* name= URLEncoder.*encode*(name); value=URLEncoder.*encode*(value);  *//创建cookie* Cookie cookie1=**new** Cookie(name,value); *//响应cookie* resp.addCookie(cookie1);  *//获取cookie时通过URLEncoder进行解码* Cookie[] cookie= req.getCookies(); **if**(cookie!=**null**&&cookie.**length**>0){  **for**(Cookie cookies:cookie){  System.***out***.println(URLDecoder.*decode*(cookies.getName()));  System.***out***.println(URLDecoder.*decode*(cookies.getValue()));  } } |

3.同名Cookie问题

如果服务器端发送重复的Cookie那么会覆盖原有的Cookie.

4.浏览器存放Cookie的数量

不同的浏览器对Cookie也有限定，Cookie的存储有是上限的。Cookie是存储在客户端(浏览器)的，而且一般是由服务器端创建和设定。后期结合Session来实现回话跟踪，

### 8.5. Cookie的路径

Cookie的setPath设置cookie的路径，这个路径直接决定服务器的请求是否会从浏览器中加载某些cookie。

情景一:当前服务器下任何项目的任意资源都可获取Cookie对象

/\*当前项目路径为: s01 \*/

Cookie cookie = new Cookie("xx" ,"xXX");

//设置路径为"/"，表示在当前服务器下任何项目都可访问到Cookie对象

cookie. setpath("/");

response. . addcookie(cookie);

情景二:当前项目下的资源可获取Cokie对象(默认不设置Cookie的path)

/\*当前项目路径为: s01 \*/

Cookie cookie . new Cookie("xx" ,"xxx");

//设置路径为"/s01",表示在当前项目下任何项目都可访问到cookie对象

cookie. setPathC"/s01"); //默认情况，可不设置path的值

response.。addcookie(cookie);

情景三:指定项目下的资源可获取Cookie对象

/\*当前项目路径为: s01 \*/

Cookie cookie = new CookieC"xxx" ,"XXX");

//设置路径为"/s02"，表示在s02项目下才可访问到cookie对象

cookie. setPath("/s02"); // 只能在s02项目下获取Cookie,就算cookie是s01产生的， s01也不能获取它

response . addCookie(cookie);

情景四:指定目录下的资源可获取Cookie对象

/\*当前项目路径为: s01 \*/

Cookie cookie =new CookieC"xx" ,"XX");

//设置路径为" /s01/cook",表示在s02/cook目录下才可访问到Cooki e对象

cookie. setpath("/s01/cook");

response. addcookie(cookie);

如果我们设置path,如果当前访问的路径包含了cookie的路径（当前访问路径在cookie路径基础上要比cookie的范围小） cookie就会加载到request对象之中。

cookie的路径指的是可以访问该cookie的顶层目录，该路径的子路径也可以访问该cookie.

总结:

当访问的路径包含了cookie的路径时，则该请求将带上该cookie;如果访问路径不包含cookie路径,则该请求不会携带该cookie.

## 9. HttpSession对象

HttpSession对象是javx.servlet.http.HttpSession的实例，该接口并不像HttpServletRequest或

HttpServletResponse还存在-个父接口，该接口只是一一个纯粹的接口。这因为session本身就属于HTTP协议的范围。

对于服务器而言，每一个连接到它的客户端都是一个session, servlet 容器使用此接口创建HTTP客户端和HTTP服务器之间的会话。会话将保留指定的时间段，跨多个连接或来自用户的页面请求。一个会话通常对应于一个用户，该用户可能多次访问一个站点。可以通过此接口查看和操作有关某个会话的信息，比如会话标识符、创建时间和最后一次访问时间。 在整个session中,最重要的就是属性的操作。

session无论客户端还是服务器端都可以感知到，若重新打开一个新的浏览器，则无法取得之前设置的session,因为每一个session只保存在当前的浏览器当中，并在相关的页面取得。

Session的作用就是为了标识一次会话，或者说确认一个用户;并且在一次会话（一个用户的多次请求）期间共享数据。我们可以通过request.getSession0方法，来获取当前会话的session对象。

|  |
| --- |
| *//获取session对象,如果session对象存在，则获取，不存在，则创建* HttpSession session=req.getSession(); |

### 9.1.标识符JSESSIONID

Session既然是为了标识一次会话，那么此次会话就应该有一个唯一的标这个标志就是sessonld.每当一次请求到达服务器，如果开启了会话(访问了session)。服务器第一步会查看是否从客户端回传一个名为JSESSIONID的cookie,如果没有则认为这是-次新的会话，会创建-个新的session对象，并用唯一的sessionld为此次会话做一个标志。如果有JESSIONID这个cookie回传，服务器则会根据JSESSIONID这个值去查看是否含有Id为JSESSION值的session对象，如果没有则认为是一个新的会话，重新创建一个新的session对象，并标志此次会话:如果找到了 相应的session对象，则认为是之前标志过的一次会话，返回该session对象，数据达到共享。

这里提到一个叫做JSESSIONID的cookie,这是-个比较特殊的cookie,当用户请求服务器时，如果访问了session,则服务器会创建一一个名为 JSESSIONID,值为获取到的session (无论是获取到的还是新创建的)的sessionld的cookie对象,并添加到response对象中,响应给客户端，有效时间为关闭浏览器。

所以Session的底层依赖Cookie来实现。

### 9.2. session域对象

Session用来表示-次会话，在- -次会话中数据是可以共享的，这时session作为域对象存在，可以通过setAttribute(name,value)方法向域对象中添加数据，通过gettribute(name)从域对象中获取数据，通过removeAttribute(name)从域对象中移除数据。

|  |
| --- |
| *//获取session域对象*  HttpSession session=req.getSession();  *//设置session域对象* session.setAttribute(**"username"**,**"admin"**);  *//获取指定名称的session域对象* String username= (String) request.getSession().getAttribute(**"username"**);  *//移除指定名称的session域对象* session.removeAttribute(**"password"**); |

数据存储在session域对象中，当session对象不存在了，或者是两个不同的session对象时，数据也就不能共享了。这就不得不谈到session的生命周期。

### 9.3. session对象的销毁

**9.3.1.默认时间到期**

当客户端第一次请求serlet并且操作session时，session 对象生成，Tomcat中session默认的存活时间为30min,即你不操作界面的时间，一旦有操作，session 会重新计时。

那么session的默认时间可以改么?答案是肯定的。

可以在Tomcat中的conf目录下的web.xml文件中进行修改。

|  |
| --- |
| <!-- session 默认的最大不活动时间。单位：分钟。-->  <session-config>  <session-timeout> 30</session-timeout>  </session- config> |

**9.3.2.自己设定到期时间**

当然除了以上的修改方式外，我们也可以在程序中自己设定session的生命周期，通过

session.setMaxlnactiveInteral(int)来设定session的最大不活动时间，单位为秒。

|  |
| --- |
| *//获取session对象* HttpSession session=req.getSession(); *//修改最大不到期时间（15秒失效）* session.setMaxInactiveInterval(15); |

当然我们也可以通过getMaxInactiveInterval()方法来查看当前Session对象的最大不活动时间。

|  |
| --- |
| *//手动设置session对象最大不到期时间* int time=session.getMaxInactiveInterval()); |

**9.3.3.立刻失效**

或者我们也可以通过session.invalidate0)方法让session立刻失效I

|  |
| --- |
| //销毁session对象  session. Invalidate(); |

**9.3.4.关闭浏览器**

从前面的JESSION可知道，session 的底层依赖cookie实现，并且该cookie的有效时间为关闭浏览器，从而session在浏览器关闭时也相当于失效了(因为没有JSESSION再与之对应) .

**9.3.5.关闭服务器**

当关闭服务器时，session 销毁.

Session失效则意味着此次会话结束，数据共享结束。

## 10. ServletContext对象

每一个web应用都有且仅有一个ServletContext 对象，又称Application对象,从名称中可知，该对象是应用程序相关的。在WEB容器启动的时候，会为每一个WEB应用程序创建一个对应的 ServletContext对象。

该对象有两大作用：

第一、作为域对象用来共享数据，此时数据在整个应用程序中共享;

1. 该对象中存了当前应用程序相关信息。例如可以通过getServerlnfo()方法获取当前服务器信息，getRealPath(Stringpath)获取资源的真实路径等。

### 10.1. ServletContext对象的获取

获取ServletContext对象的途径有很多。比如:

1.通过request对象获取

ServletContext servletContext = request . getservletContext();

2.通过session对象获取

ServletContext servletContext = request . getsession() . getServletContext();

3.通过servletConfig对象获取，在Serlet标准中提供了ServletConfig方法

ServletConfig servletConfig = getServletConfig();

ServletContext servletContext = servletconfig .getServletContext();

4.直接获取。Servlet类中提供了直接获取ServletContext对象的方法

ServletContext servletContext4=getServletContext();

### 10.2. ServletContext域对象

ServletContext也可当做域对象来使用，通过向ServletContext中存取数据，可以使得整个应用程序共享某些数据。当然不建议存放过多数据，因为ServletContext中的数据一旦存储进去没有手动移除将会一直保存。

//获取ServletContext对象

ServletContext servletContext = request .getservletContext();

//设置域对象

servletContext. setattribute("name" ," zhangsan");

//获取域对象

string name = (string) servletContext , getAttri bute("name");

//移除域对象

letContext. removeAttribute("name");

Servlet的三大域对象

1. request域对象

在一次请求中有效。请求转发有效，重定向失效。

2. session域对象

在一次会话中有效。请求转发和重定向都有效，session销毁后失效。

3. servletContext域对象

在整个应用程序中有效。服务器关闭后失效。

## 11.文件上传和下载

在上网的时候我们常常遇到文件上传的情况，例如上传头像、上传资料等;当然除了上传，遇见下载的情况也很多，接下来看看我们servlet中怎么实现文件的上传和下载。

### 11.1.文件上传

文件.上传涉及到前台页面的编写和后台服务器端代码的编写，前台发送文件，后台接收并保存文件,这才是一个完整的文件上传。

**11.1.1.前台页面**

在做文件上传的时候，会有一个上传文件的界面，首先我们需要一个表单， 并且表单的请求方式为POST;其次我们的form表单的enctype必须设为multipart/form-data",即enctype="multipart/form-data",意思是设置表单的类型为文件.上传表单。默认情况下这个表单类型是applicationwx-www.form-urlencoded",不能用于文件.上传。只有使用了multipartform-data才能完整地传递文件数据。

|  |
| --- |
| <**form action="uploadservlet" method="post" enctype="multipart/form-data"**>  姓名：<**input type="text" name="username"**>  文件：<**input type="file" name="myfile"**>  <**input type="submit" value="ok"**> </**form**> |

**11.1.2.后台实现**

使用注解@MultipartConfig将-个Servlet标识为支持文件上传。Servlet 将multipart/form-data的POST请求封装成Part,通过Part对上传的文件进行操作。

|  |
| --- |
| */\*\*  \* 使用@MultipartConfig注解将-个Servlet标识为支持文件上传。  \* Servlet 将multipart/form-data的POST请求封装成Part,通过Part对上传的文件进行操作。  \*/* @WebServlet(**"/fileuploads"**) @MultipartConfig **public class** UploadServlet **extends** HttpServlet {  @Override  **protected void** service(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println(**"文件上传"**);   *//设置编码格式* req.setCharacterEncoding(**"utf-8"**);  resp.setCharacterEncoding(**"utf-8"**);   *//获取表单对象* String username=req.getParameter(**"username"**);  System.***out***.println(**"name:"**+username);   *//获取Part对象(Servlet将multipart/form-data的POST请求封装成Part对象)* Part part=req.getPart(**"myfile"**);  *//通过part对象获取上传的文件名* String filename=part.getName();  System.***out***.println(**"上传的文件名:"**+filename);   *//得到文件存放的路径* String filepath=req.getServletContext().getRealPath(**"/"**);  System.***out***.println(**"文件存放的路径:"**+filepath);  *//上传文件到指定目录* part.write(filepath+**"/"**+filename);  } } |

### 11.2.文件下载

文件下载，即将服务器上的资源下载(拷贝)到本地,我们可以通过两种方式下载。第一种是通过超链接本身的特性来下载，第二种是通过代码下载。

**11.2.1.超链接下载**

当我们在HTML或JSP页面中使用a标签时，原意是希望能够进行跳转。但当超链接遇到浏览器不识别的资源时会自动下载:当遇见浏览器能够直接显示的资源，浏览器就会默认显示出来，比如txt, png. jpg等。当然我们也可以通过download属性规定浏览器进行下载。但有些浏览器并不支持。

默认下载

|  |
| --- |
| <!--当超链接遇到浏览器不识别的资源时，会自动下载-->  <a href-"test. zip">超链接下载</a> |

指定download属性下载

|  |
| --- |
| <!--当超链接遇到浏览器识别的资源时，默认不会下载。通过download属性可进行下载-->  <a href-"test. txt" download>超链接 下载</a> |

download属性可以不写任何信息，会自动使用默认文件名。如果设置了download属性的值，则使用设置的值做为文件名。当用户打开浏览器点击链接的时候就会直接下载文件。

**11.2.2.后台实现下载**

实现步骤

1. 需要通过response.setContentType方法设置Content-type头字段的值，为浏览器无法使用某种方式或激活某个程序来处理的MIME类型，

例如**"application/octet-stream"** 或 **"application/x-msdownload"** 等。

需要通过reponse.setHeader()方法设置**"Content-Disposition"**头的值为**"attachment;filename="**+文件名

3.读取下载文件，调用response.getOutputStream方法向客户端写入附件内容。