一、HttpServletRequest 对象

1、介绍

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，service()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 HTTP 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。

2、使用

1)、常用的方法

请求方式+url+版本号

getRequestURL：返回客户端发出请求时的完整 URL

getRequestURI：返回请求行中的资源名部分(项目名称开始)

getQueryString ：返回请求行中的参数部分

getMethod：得到客户端请求方式

getProtocol:获取 HTTP 版本号

getContextPath:获取 webapp 名字

获取客户端若干消息头

getHeader (String name)：获取单个请求头内容

Enumeration<String> getHeaderNames():获取所有的请求头名称集合

获得客户机请求参数(客户端提交的数据)

getParameter(name):获取指定名称的参数值，这是最为常用的方法之一。

getParameterValues（String name）:获取指定名称参数的所有值数组。它适用于一个参数名对应多个值的情况：如页面表单中的复选框，多选列表提交的值。

getParameterNames():返回一个包含请求消息中的所有参数名的 Enumeration 对象。通过遍历这个 Enumeration 对象，就可以获取请求消息中所有的参数名。

getParameterMap():返回一个保存了请求消息中的所有参数名和值的 Map 对象。Map 对象的 key 是字符串类型的参数名，value 是这个参数所对应的 Object 类型的值数组

2)、乱码解决

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。

①req.setCharacterEncoding("UTF-8");这种方式只针对 POST 有效

②new String(req.getParameter(name).getBytes(“ISO-8859-1”));

Tomcat8 之后 GET 不会出现乱码

3)、转发、重定向

区分是客户端跳转还是服务器端跳转：

看地址栏： 地址栏改变-->客户端跳转； 地址栏不改变--> 服务器端跳转

在写路径时： 以/开头或者 http://都是绝对路径

不以/开头就是相对路径：服务器和客户端一样都是相对当前资源所在的路径

代码中：

服务器端路径(转发)：/ --> http://localhost:8080/servlet03

客户端路径(重定向)：/ -->http://localhost:8080

各种表单输入项数据的获取

现在遇到的客户端跳转： 超链接、form 表单、重定向

服务器端跳转： 请求转发

4)、request 域对象

通过该对象可以在一个请求中传递数据，作用范围：在一次请求中有效，即服务器跳转有效。request.setAttribute() :设置域对象内容；request.getAttribute(String name): 获取域对象内容

二、HttpServletResponse 对象

1、介绍

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 对象既然代表请求和响应 获取客户端数据 ,通过 request 对象；向客户端输出数据，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。

2、常用方法

addCookie(Cookie cookie)：将指定的 Cookie 加入到当前的响应中

addHeader(String name,String value)：将指定的键值加入到响应头信息中

containsHeader(String name) ：返回一个布尔值，判断响应的头部是否被设置

encodeURL(String url)：编码指定的 URL

sendError(int sc)：使用指定状态码发送一个错误到客户端

sendRedirect(String location)：发送一个临时响应到客户端，重定向

setHeader(String name,String value)：将给出的名字和值设置响应的头部

setStatus(int sc)：给当前响应设置状态

setContentType(String ContentType)：设置响应的 MIME 类型

getWriter()：获取输出字符流

getOutputStream()：获取输出的字节流

3、设置头信息

1)、刷新、跳转页面

所有头信息都是随着请求和回应自动发送到服务器端（客户端），在 response 中一个比较常用的头信息就是刷新的指令，可以完成定时刷新的功能。

```
resp.setHeader("refresh","2");
```

对于刷新的头信息，除了定时的功能外，还具备了定时跳转的功能，可以让一个页面定时跳转到一个指定的页面。(已经注册成功，两秒后跳转到登陆页面)

```
response.setHeader("refresh","3;URL=ok.html");
```

但是这种跳转不是万能的，有时候根本就无法进行跳转操作，返回后刷新不会跳转；

对于这种定时跳转的头信息，也可以采用 HTML 的方式进行设置，HTML 本身也可以设置头信息。(客户端跳转)

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

2)、返回状态码

setStatus 方法用来设置 Servlet 向客户端返回的状态码，它用来设置没有出错的状态，通常不使用。如果 Servlet 运行出错，Servlet 可以使用 sendError 方法设置状态码，如 sendError(int sc)方法设置错误状态代码。sendError(int sc,String msg)方法除了设置状态码，还向客户发出一条错误信息。

3)、添加 Cookie

addCookie 方法可以在 Web 服务器响应中加入 Cookie 对象，这个对象将被浏览器所保存。Cookie 机制也被用来维护会话状态。

4、跳转

通过 response 也可以进行跳转，该种跳转称为请求重定向，属于客户端跳转，地址栏改变，存在两个请求。请求中的参数不能在此种跳转间传递。

1)、请求转发(Dispatcher)

转发过程：客户浏览器发送 http 请求---->web 服务器接受此请求-->调用内部的一个方法在容器内部完成请求处理和转发动作---->将目标资源发送给客户。此种跳转从头到尾只有一个请求，即只有一个 Request 对象

注意：转发的路径必须是同一个 web 容器下的 url，由于中间传递的是自己的容器内的 request。在客户浏览器路径栏显示的仍然是其第一次访问的路径，也就是说客户是感觉不到服务器做了转发。request.getRequetsDispatcher（“url”）.forward(req, resp)

2)、请求重定向(Redirect)

重定向过程：客户浏览器发送 http 请求--->web 服务器接受后发送 302 状态码响应及对应新的 location 给客户浏览器-->浏览器发现是 302 响应，自动再发送一个新的 http 请求，请求的 url 是新的 location 地址-->服务器根据此请求寻找资源并发送给客户。

response.sendRedirect（“url”）

注意：在这里 location 可以重定向到任意 URL，既然是浏览器重新发出了请求，就没有 request 传递的概念了，在客户浏览器路径栏显示的是其重定向的路径，客户可以观察到地址的变化。

5、解决乱码问题

当服务器端响应给客户端中文时，可能存在乱码

两种方式响应回数据：PrintWriter 和 ServletOutputStream

getOutputStream 和 getWriter 这两个方法互相排斥，调用了其中的任何一个方法后，就不能再调用另一方法。

PrintWriter()：一定乱码，服务器自动使用 ISO-8859-1 编码，通过如下方式解决乱码

```
response.setCharacterEncoding("utf-8");
response.setHeader("content-type", "text/html;charset=utf-8");
```

一句顶两句：

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

OutputStream() ：可能乱码

使用 OutputStream 流输出中文注意问题：

服务器端，数据输出的编码方式，就是客户端打开的编码方式

例：outputStream.write("中国".getBytes("utf-8"));使用 OutputStream 流向浏览器输出中文，编码方式是 utf-8；此时客户端浏览器也要以 utf-8 的编码打开，否则会出现中文乱码。在服务器端控制客户端浏览器以 utf-8 的编码方式显示，此时既可以通过上述解决 PrintWriter 的方式，也可以通过设置响应头信息的方式：

response.setHeader("content-type", "text/html;charset=UTF-8");

6、响应图片

通过图片访问 servlet

img.html

```
<img alt="美女" src="imgServlet">
```

Servlet 中设置响应头，表明响应的数据类型，学过的有 text/html 文本类型，其他 MIME 类型见 Tomcat 服务中的 web.xml

注意：此时用流的形式读取再输出

```
        resp.setHeader("Content-Type", "image/gif");
        //获取资源路径
        String realPath=req.getServletContext().getRealPath("/girl.gif");
        ServletOutputStream out=resp.getOutputStream();
        InputStream in=new FileInputStream(new File(realPath));
        byte[]car =new byte[1024];
        int len=0;
        while((len=in.read(car))!=-1){
            out.write(car,0,len);
```

```
        }
        out.flush();
        in.close();
        out.close();
```

<mime-mapping></mime-mapping> 定义某一个扩展名和某一个 MIME Type 做对应，包含两个子元素：

①<extension></extension> 扩展名的名称

②<mime-type></mime-type> MIME 格式

三、HttpSession 对象

1、简介

HttpSession 对象是 javax.servlet.http.HttpSession 的实例，该接口并不像 HttpServletRequest 或 HttpServletResponse 还存在一个父接口，该接口只是一个纯粹的接口。因为 session 属于 HTTP 协议的范畴

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取得，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。

2、使用

1)、常用方法

setAttribute(String name,Object value)：将 value 对象以 name 名称绑定到会话

getAttribute(String name)：取得 name 的属性值，如果属性不存在则返回 null

removeAttribute(String name)：从会话中删除 name 属性，若不存在不会执行，也不会抛处错误。

Enumeration getAttributeNames()：返回和会话有关的枚举值

invalidate()：使会话失效，同时删除属性对象

isNew()：用于检测当前客户是否为新的会话

long getCreationTime()：返回会话创建时间

getLastAccessedTime()：返回会话时间内 web 容器接收客户最后发出的请求时间

int getMaxInactiveInterval()：返回在会话期间内客户请求的最长时间为秒

setMaxInactiveInterval(int seconds)：允许客户客户请求的最长时间

ServletContext getServletContext()：返回当前会话的上下文环境，ServletContext 对象可以使 Servlet 与 web 容器进行通信

String getId()：返回会话期间的识别号

Session：默认是在第一次使用 Session 时被创建

Cookie：大部分都是通过服务器主动(程序手动)设置到浏览器的，只有 JSESSIONID 是服务自己设定的， 不需要程序员操心

2)、session id

对于每一个用户而言，实际上都表示一个个不同的 session，服务器通过 session id 来区分这些用户，即每一个通过浏览器连接到服务器上的用户都会由服务器分配一个唯一

的不会重复的编号。

cookie 自动设置的 JSESSIONID 的值就是每一个用户的 session id，所以 session 在进行操作的时候用到了 cookie 的处理机制。

session id 的结束：

① 生命周期到期(默认一次浏览器的打开与关闭)

②非正常关闭服务器 session 销毁

正常关闭服务器，session 对象不会被销毁，而是会被序列化到磁盘上

工作空间 work 目录下 SESSION.ser

③ 调用 session 的 invalidate() 方法

session 在所有的项目开发中，用得最多的地方就是登陆验证及注销操作功能

3)、域对象

HttpServletRequest、ServletContext 、HttpSession 都是域对象 ,它们三个是 Servlet 中可以使用的域对象。

HttpServletRequest：一个请求创建一个 request 对象，所以在同一个请求中可以共享 request。例如一个请求从 AServlet 转发到 BServlet，那么 AServlet 和 BServlet 可以共享 request 域中的数据；

ServletContext：一个应用只创建一个 ServletContext 对象，所以在 ServletContext 中的数据可以在整个应用中共享，只要不重启服务器。

HttpSession：一个会话创建一个 HttpSession 对象，同一会话中的多个请求中可以共享 session 中的数据；

4)、会话跟踪

会话：两个人一次见面的交谈。什么时候结束，分别的时候。在这次交谈中，双方都可以说多句话，即请求响应可能有多次。

javaweb 中，客户发送第一个请求开始，见面成功，即会话开始，发请求响应，请求响应…直到客户关闭浏览器，两人分别，会话结束。

为什么需要会话跟踪技术？

①http 协议无状态

②为了在一个会话中多个请求间共享数据

③前面的请求转发，数据只能在一个请求

④重定向，不能共享数据

HTTP 协议是无状态协议 ,也就是说每个请求都是独立的 !无法记录前一次请求的状态。但 HTTP 协议中可以使用 Cookie 来完成会话跟踪！在 JavaWeb 中，使用 session 来完成会话跟踪，session 底层依赖 Cookie 技术。

判断新用户

在 session 的对象中可以使用 isNew()判断一个用户是否是新用户

取得用户操作时间

如果要想取得一个 session 的操作时间，可以通过计算方法求出

3、Cookie

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。

有专门提供的 javax.servlet.http.Cookie 的操作类

首先 Cookie 就是一个键和一个值构成的，随着服务器端的响应发送给客户端浏览器。然后客户端浏览器会把 Cookie 保存起来，当下一次再访问服务器时把 Cookie 再发送给服务器。由服务器创建，客户端保存

格式：Cookie: a=A; b=B; c=C。即多个 Cookie 用分号离开；

注意：不同浏览器之间是不共享 Cookie 的。

在一般的站点中经常都会存在记住用户名的这样一个操作，只是将信息保存在本机上，换电脑后这些信息无效。

设置 Cookie：在服务器端 ,发送到客户端 response.addCookie();

取得 Cookie：request.getCookie();

获取 Cookie：Cookie.getName+Cookie.getValue()

在客户端每次发送请求时都会发送一个 Cookie 的头信息，所以想要取得 Cookie 则必须使用 request 对象完成

Cookie 由于没有设置保存的时间，所以默认是在浏览器关闭时，Cookie 失效。若想要真正的保存在客户端上一段时间，则必须使用 Cookie 类提供的 setMaxAge()指定 cookie 的最大生存时间(以秒为单位)大于 0 的整数；若为负数，则表示不存储该 cookie；若为 0，则删除该 cookie

cookie.setMaxAge()三种设置

负数：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：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：cookie 生命等于 0 是一个特殊的值，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。

JSESSION 是在第一次请求完成之后才被设置到浏览器上，所以第一次请求并不带这个 Cookie，只有当有了 Session 才会发送该 Cookie

response 对象主要表示的就是服务器端对客户端的回应，对于 Cookie 的操作本身是存在安全隐患的。Cookie 是 servlet 发送到 Web 浏览器的少量信息，这些信息由浏览器保存，然后发送回服务器。cookie 的值可以唯一地标识客户端，因此 Cookie 常用于会话管理。

一个 cookie 拥有一个名称、一个值和一些可选属性，比如注释、路径和域限定符、最大生存时间和版本号。一些 Web 浏览器在处理可选属性方面存在 bug，因此有节制地使用这些属性可提高 servlet 的互操作性。

浏览器支持每台 Web 服务器最多保存 20 个 cookie；一个浏览器最多保存 300 个 cookie(概数)；并且可能将每个 cookie 的大小限定为 4 KB。

Cookie 的覆盖

如果服务器端发送重复的 Cookie 那么会覆盖原有的 Cookie。

Cookie 的路径

浏览器在访问 BServlet 时，是否会带上 AServlet 保存的 Cookie，这取决于 Cookie 的 path。

在项目中的配置文件 web.xml

```
<servlet-mapping>
    <servlet-name>cookie</servlet-name>
```

```
    <url-pattern>/coo</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>cookie01</servlet-name>
    <url-pattern>/cookie/cookie01</url-pattern>
</servlet-mapping>
  <servlet-mapping>
    <servlet-name>cookie02</servlet-name>
    <url-pattern>/cookie/cookie02</url-pattern>
</servlet-mapping>
```

在 Cookie01.java 中添加一个 cookie

```
        Cookie cookie=new Cookie("xxx","XXX");//默认浏览器关闭时失效
        cookie.setPath("/servlet03/cookie");
        resp.addCookie(cookie);//默认是浏览器关闭时失效
        System.out.println(cookie.getPath());
```

在 Cookie02.java 中获取 cookie01.java 中添加的 cookie

```
        Cookie[] cookies = req.getCookies();
        if(cookies!=null){
            for(Cookie temp : cookies){
System.out.println(temp.getName() + "=" + temp.getValue() + " cookie02");
            }
        }
```

在 Cookie.java 中获取 cookie01.java 中添加的 cookie

```
    Cookie[] cookies = req.getCookies();
    if(cookies!=null){
        for(Cookie temp : cookies){
    System.out.println(temp.getName() + "=" + temp.getValue() + " cookie");
        }
    }
    System.out.println("cookie");
```

在 Servlet 中保存的 Cookie 如果没有设置 path，那么它的 path 默认为当前 Servlet 的所在路径；

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

Cookie 保存中文

Cookie 的 name 和 value 保存成中文会报错；但可以先把中文转换成 URL 编码，然后在保存到 Cookie 的 name 和 value 中。

```
String name="姓名";
String value="张三";
name=URLEncoder.encode(name,"utf-8");
value=URLEncoder.encode(value, "utf-8");
Cookie cookie=new Cookie(name,value);
resp.addCookie(cookie);
//获取cookie
Cookie[]cookies=req.getCookies();
if(cookies!=null){
    for(Cookie coo:cookies){
        String name=URLDecoder.decode(coo.getName(), "utf-8");
        String value=URLDecoder.decode(coo.getValue(),"utf-8");
        System.out.println(name+":"+value);
    }
}
```

四、 ServletContext 对象

一个 web 应用只有一个 ServletContext 对象，所有的 servlet 都共享这个 ServletContext 对象，又称为 Application 对象。WEB 容器在启动时，会为每个 WEB 应用程序创建一个对应的 ServletContext 对象,其代表当前 WEB 应用。ServletContext 定义了一组方法，servlet 使用这些方法与其他 servlet 容器进行通信，ServletContext 对象包含在 ServletConfig 对象中，ServletConfig 对象在初始化 servlet 时由 web 服务器提供给 servlet。

public void	setAttribute(String name, Object object)
public Object	getAttribute(String name)
public java.util.Enumeration<E>	getAttributeNames()
public String	getContextPath()

public String	getInitParameter(String name)
public String	getRealPath(String path)
public java.io.InputStream	getResourceAsStream(String path)
public String	getServerInfo()
public String	getServletContextName()