# Request对象学习之获取请求数据

当浏览器向服务器请求的时候，服务器会搜寻webapps->虚拟项目名（比如project）→Myservlet->service方法。同时会在服务器里，根据此次请求创建一个**request**对象，此对象封存了此次请求的相关数据。Request会作为实参传递给**service**方法！

**Request**对象（此次请求的的所有数据）：

①由服务器创建，一次请求创建一个

②生命周期为一次请求内，请求结束即销毁此次对象。

Service方法内部要进行：

①**获取**请求信息

②**请求**处理信息

③**响应**处理结果

**请求数据：**

**1请求行：1.1请求方式 1.2请求URL协议**

**2请求头**

**3请求实体**

**一、请求行信息**

**业务逻辑需要使用的信息**

**请求行**

**1.1获取—请求方式**

**String method = req.getMethod(); //method就是请求方式**

**1.2获取—请求URL信息**

**StringBuffer requestURL=req.getResultURL**

**System.out.println(requstURL.toString()); //http://localhost:8080/03/user**

**1.3获取—请求URI信息**

**String requestURI =req.getRequeURI（）； //03/user 找到我们要的servlet**

**1.4获取get请求中URL的请求数据**

**String queryString=req.getQueryString(); //url以后用？连接的内容**

**1.5 获取协议**

**String scheme = req. getScheme（）； // 返回结果为http**

**备注：alt＋shift＋L 可以快速出现返回值**

**二、请求头信息**

**网页搜集到的用户资料才是我们真正会需要的，核心信息。**

**2.0根据键名获取请求的头信息**

**Enumeration headerNames =req.getHeaderNames();//返回存储头键名的枚举类型**

**While(headerNames.hasMoreElements())//判断是否还有元素，枚举加一**

**{**

**//获取请求头键名**

**String name = （String）headerNames.nextElements();**

**//获取请求请求头键名对应的值**

**String value=req.getHeader(name);**

**System.out.println(name+“：”+value);**

**}**

**输出request对象所有头键名**

**2.1根据键名获取请求头信息**

**String header = req.getHeader(“User-Agent”)//获取浏览器版本（用户代理商）**

**如果没有这个键名，则会返回null**

三、请求实体（用户数据）

**键名就是前端页面中表单标签中的name属性或其他方式提交的**

**3.1根据键名获取数据**

**String uname=req.getParameter(uname);**

**String pwd= req.getParameter(pwd);**

**//这个键名必须与前端form一致。如果没有对应键名，返回null。**

**3.2获取一个键名但有多个不同值的实体数 据**

**String favs=req.getParameterValues(“fav”);//在前端设置fav为一个键**

**If（fav!=null）{**

**for(String s:favs){**

**System.out.println(“fav的值为”+s);**

**}**

**}**

**3.3获取请求实体中的键名枚举**

**Enumeration names = req.getParameterNames();**

**While(names.hasMoreElements()){**

**System.out.println(names.nextElements());**

**}**

**3.4根据键名拿值.**

**（我们想要的形式是 键名：Values。）**

**Enumeration names = req.getParameterNames();//不适合用多个值的情况**

**While(names.hasMoreElements()){**

**//获取键名**

**String name = (String)names.nextElements();**

**//如果fav有多个值，到fav的时候就可以单独对fav这个键名 进行getParameterValues判断。**

**If(“fav”.equals(name)){**

**String[] favs2=req.getParameterValues(“fav”);**

**If（fav!=null）{**

**For(String s :favs2)**

**System.out.println(name+”:”+s );**

**}**

**}else{**

**//获取值**

**String value =req.getParameter(name);**

**System.out.println(name+”:”value);**

**}**

**四、请求相关的网络的数据**

**4.1获取客户端IP地址**

**String remoteAddr = req.getRemoteAddr();**

**4.2获取客户端端口号（即浏览器）**

**Int remotePort = req.getRemotePort();**

**4.3 获取服务器主机IP**

**String localAddr=req.getLocalAddr();**

**4.4获取服务器的端口号:8080**

**String localPort = req.getLocalPort();**

# Response对象学习之响应处理结果

**一旦Response对象做了请求响应，则意味着本次请求结束。服务器会在响应以后将此次请求相关的req与resp对象销毁。**

1、设置响应行：协议、状态码、状态消息

2、设置响应头

3、设置响应实体

**1、设置响应行**

自定义响应异常

resp.sendError(404);//也可以设置500

**2、设置响应头**

2.1添加响应头信息

resp.addHeader(“mouse”,”thinkpad”);//name . value

2.2设置响应头信息

resp.setHeader(“mouse”,”two fly bird”);//如果没有就创建

**3、设置响应实体**

resp.getWriter().write(“resp object”);

**解决乱码的情况**

// 1、获取请求数据 2、处理请求数据 3、响应处理结果

req.setCharacterEncoding("utf-8");

String uname = req.getParameter("uname");//1、获取请求数据

System.out.println(uname); //2处理请求数据

resp.getWriter().write("响应请求结果");//3响应处理结果

**请求乱码：服务器获取的请求数据乱码**

**1、post请求方式乱码：**

req.setCharacterEncoding("utf-8");

**2、 get请求方式乱码：**

req.setCharacterEncoding("utf-8");

**并且在tomcat-conf里的server.xml文件的Connection标签属性中：userBodyEncodingForURI=”true”.**

**3、服务器乱码情况**

**resp.setContentType(“text/html;charset=utf-8”);**

# 请求转发

**解释请求转发**：在一个servlet中调用其他servlet。

**Req.getRequestDispatcher(“转发的路径”).forward(req,resp);**

特点：

①降低servlet的代码冗余

②一次请求转发内的servlet共享此次请求的request和Response对象

③浏览器地址栏信息不变，仅一次请求

LoginServlet：

req.getRequestDispatcher("login").forward(req,resp);

req.setAttribute("str","用户名密码错误");

UserServlet：

String str=((req.getAttribute("str"))==**null**?"":(String)

(req.getAttribute("str")));

resp.getWriter().write("<br/>"+str);

此为请求转发。

# 重定向

作用：保护第一次的请求，避免因为用户的刷新动频繁触发servlet的执行。

**特点：两次请求，地址栏的信息改变。**

登录成功以后我们常常会使用重定向，因为主页面可能拥有很多东西。

**请求转发：**

我们在login页面登录，输入错误的账号与密码。进入到user页面，同时利用请求转发实现了user页面也可以拥有登录的信息。

**重定位：**

我们在login页面登录，输入错误的账号与密码。进入到user页面，同时重定位到login页面。**两次请求，一次请求转发。**

**重定位两次请求会生成两次request与resp。**

**它保护了A这个servlet，但同时重定位不利于信息的流动性。**

resp.sendRedirect("login");

**Cookie:**

在服务器端进行声明哪些数据需要被存储。

以响应的方式告诉浏览器来存储。

作用：实现不同请求之间的数据共享。

|  |
| --- |
| Servlet-2  String name = **null**;  Cookie[] cookies =req.getCookies();  **for**(Cookie ck:cookies){  **if**("name".equals(ck.getName())){  name=ck.getValue();  }  }  String c = req.getParameter("c");  //处理请求信息  System.out.println(name+":"+c); |
| Servlet-1  //响应处理结果（浏览器）  //创建Cookie数据  Cookie c = **new** Cookie("name",name);  //设置cookie信息  c.setMaxAge(3600\*24\*3);  //设置有效路径  c.setPath("/MyServlet2/02") ;  //只有这次路径下，才会有Cookie的数据  //设置Cookie数据  resp.addCookie(c); |

如果不设置Cookie最大时间，cookie缓存在浏览器内存。

设置以后缓存在硬盘内。

**Session:解决的是两次请求.**

**解决重定向之间的共享、**

**ServletA**

|  |
| --- |
| HttpSession session = req.getSession();//create a session  session.setAttribute("username", username);//储存数据到Session对象当中  System.out.println("ServletA.service:"+session.getId());//打印Session的IDHttpSession session = req.getSession();//create a session  session.setAttribute("username", username);//储存数据到Session对象当中  System.out.println("ServletA.service:"+session.getId());//打印Session的ID |

**ServletB**

|  |
| --- |
| HttpSession session = req.getSession();//如果没有创建新的。  //获取A的处理对象结果数据  String username = (String) session.getAttribute("username");  //处理请求信息  System.out.println("ServletB"+username); |

拿到A创建的那个Session，既然是A创建的，那么A里面的属性username也可以拿到，即实现了共享

使用Session：

①创建Session

HttpSession session = req.getSession();

②存储数据到Session

Session.setAttrbute(String name ,Object value);

③获取Session

HttpSession session = req,getSession();

④获取Session的数据

Session.getAttribute(String name);注意返回的是Object，需要强制转换才可以使用

⑤删除Session的数据

Session.removeAttribute(String name);//如果有则删除。

流程：

1、浏览器发起请求到A这个Servlet，在A这个Servlet中使用req.getSession()来获取Session对象，如果此次请求中没有Session的ID，则创建一个新的Session对象，如果有则返回这个Session对象的，注意Session对象没有被销毁情况下。

注意：长期不操作，位于A的Session会失效，此时用户如果在B页面，则依赖于A的B界面没有数据。则重新登录A界面可以解决该问题。

2、检验session是否失效。

3、存储数据到session对象中或者获取session的数据或者删除session的数据。

特点：

Session解决了同一个用户的不同请求的数据共享问题。

Session的作用域：浏览器不关闭，session不失效，则同一用户的任意请求获取的都是同一个session。

一次会话。

Session设置：

Session默认配置有效时间为半小时，可以在toncat的xml文件中修改此配置。

注意：此种配置方式是所有tomcat的项目都是30min

也可以在代码中修改

**ServletContext对象使用：**

作用：解决了不同之间用户之间的数据共享问题。

使用：

**①创建ServletContext对象；**

|  |
| --- |
| //三种获取ServletContext的方式  ServletContext sc1 = **this**.getServletContext();  ServletContext sc2 = **this**.getServletConfig().getServletContext();  ServletContextsc3=  req.getSession().getServletContext(); |

**②存储共享数据**

sc1.setAttribute("str",str);

**③获取共享数据**

|  |
| --- |
| //获取ServletContext对象  ServletContext servletContext = **this**.getServletContext();  //获取共享数据  servletContext.getAttribute("str"); |

**④删除共享数据**

**Sc.removeAttribute(String name);**

**特点：**

服务器创建、所有用户共享

**生命 周期：**

服务器开启到服务器结束

其他作用：

获取web.xml全局配置属性

|  |
| --- |
| //获取web.xml的全局配置  String f=sc1.getInitParameter("flag");//flag为param-name名称 |

|  |
| --- |
| <context-param>  <param-name>flag</param-name>  <param-value>flase</param-value>  </context-param> |

获取WebRoot下资源的流对象

sc1.getResource(path);

path为相对路径，写从webRoot开始查找的资源；底层自动寻找；

|  |
| --- |
| //获取WebRoot资源下的流对象  InputStream resourceAsStream = sc1.getResourceAsStream("image/a.png");  //获取WebRoot下资源的绝对路径  String path=sc1.getRealPath("image/a.png");  System.out.println("path");//动态获取资源绝对路径！ |

**ServletConfig：实现动态的设置**

获取全局配置属性的时候，我们用到它来获取

比如这个编码的格式：Req.setCharacterEncoding(“utf-8”);

**通过全局配置属性来更改它：**

**ServletConfig sg = this.getServletConfig();**

**//获取Xml文件**

**String flag =sg.getinitParameter(“flag”);**

而在xml文件中的Servlet标签：

**<init-param>**

**<param-name>code</paarm-name>**

**<param-value>utf-8</param-value>**

**</init-param>**

则

**Req.setCharacterEncoding(sg.getInitParameter(“code”));**

等价于

**Req.setCharacterEncoding(“utf-8”);**