java web

张海宁[1]

April 20, 2018

[1]hnzhang1@gzu.edu.cn

Contents

Chapter 1

cookie and session

在计算机领域，如果一个程序如果能够记忆用户的操作历史或者用户的交互历史，那么这个程序就是有状态的。这些被记住的信息就称为系统的状态。

Http 协议是无状态 的，每次的客户端请求都会被当做一个新的请求，服务端不会记住你曾经访问过，那么就无法追踪用户[1]。为了实现有状态的通信，人们提出了 cookie 和 session 技术[2]。这两种技术都是用来保存与用户有关的信息的。

1.1 cookie

Cookie[3]是由服务器端生成并发送给客户端保存的一小段文本数据。Cookie 中不能包含空格，值的存储形式是一系列的 “属性-值”。

以下为两次请求同一个 jsp 网站，抓取到的相关 http 请求信息。可以看到:

1. 第一次请求的 request header 中并无 cookie 相关信息，但是 response header 中有一个 set-cookie，也就是说，服务器看到请求中没有 cookie，那么就自动创建了一个 cookie 给客户端（这个和具体的 web 容器有关）;

2. 第二次请求的时候，request header 直接带上了这个 cookie 去访问服务器，服务器看到这个 cookie 就知道是这个特定的用户又来访问了。

第一次访问网站:
General
Request URL: http://localhost:8080/lab-java/
Request Method: GET
Status Code: 200
Remote Address: [::1]:8080
Referrer Policy: no-referrer-when-downgrade
Response Headers
HTTP/1.1 200
Set-Cookie: JSESSIONID=8DF2DE79C1951FF63FAC3B89D47FD44F; Path=/lab-java; HttpOnly
Content-Type: text/html;charset=UTF-8

[1]一个比较典型的场景就是电商网站的购物车
[2]`https://tools.ietf.org/html/rfc6265`
[3]`https://en.wikipedia.org/wiki/HTTP_cookie`

Content-Length: 1300
Date: Fri, 20 Apr 2018 10:06:57 GMT
Request Headers
GET /lab-java/ HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7
第二次访问网站:
General
Request URL: http://localhost:8080/lab-java/
Request Method: GET
Status Code: 200
Remote Address: [::1]:8080
Referrer Policy: no-referrer-when-downgrade
Response Headers
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Content-Length: 1300
Date: Fri, 20 Apr 2018 10:14:19 GMT
Request Headers
GET /lab-java/ HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
DNT: 1
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7
Cookie: JSESSIONID=8DF2DE79C1951FF63FAC3B89D47FD44F

1.1.1 cookie 的常用方法

1. 获取 cookie

   Cookie[] ck = request.getCookies();
   一个服务器可能会向一个客户端设置若干个 cookie

2. 获取某个 cookie 对象的名字和值

   ckName=ck[i].getName(); ckValue=ck[i].getValue();

3. 向客户端添加 cookie

(a) 新建 cookie
Cookie my2ndCookie = new Cookie(”username”,”unknown.yes#or#no?”);

(b) 向客户端发送 cookie
response.addCookie(my2ndCookie);

1.2 session

一个客户端与服务器端之间的通讯过程称为一次会话 (session)。

Session[4]是保存在服务器端的，是一个容器，其中保存的是一对一对的数据（属性-值）。在 jsp 中，每当一个客户端的请求到来时，服务器端都会检查其请求中是否包含 cookie，若有，则从 cookie 中读取数据（JSESSIONID）；若无，则为这一个用户端与服务器端的会话过程创建一个 session。这个 session 会在用户关闭了这个会话之后就消失了，下一次这个用户再来的时候，服务器会把他当做一个全新用户来对待。如何让服务器记住某个特定的用户呢？方法有若干种，这里介绍其中一种最常用的。在1.1部分介绍了客户端第一次访问服务器的时候，会得到一个服务器端生成的 cookie，这个 cookie 的名字是 JSESSIONID，值是一串十六进制数。这个 cookie 是用来保存用户 seesion 信息的，那个 JSESSIONID 就是服务器为那次会话生成的一个标识符，通常叫做 session id。客户端在这之后再访问服务器的话，就会带上这个 cookie 来访问，服务器看到这个 cookie 里的 session id 就会知道某个特定的用户来了。

1.2.1 session 的常用方法

1. 获取 session
   session 是 servlet 的一个内置对象，可以像 request 一样直接使用，一个客户端和一个服务器在一段时间内只拥有一个 session
2. 往 session 里写数据
   session.setAttribute(”id”, id);
   注意是一对一对的写，与 cookie 类似
3. 读 session 里的数据
   session.getAttribute(”id”);
4. 销毁 session
   session.invalidate();
   注销登陆

1.3 cookie 和 session 的生命周期

cookie 和 session 是有其生命周期的。

- 有些网站会提供一个记住密码，多长时间不用再重新登陆的功能，使用的方法就是设置 cookie 的过期时间；
- 有时候停留在某个页面长时间不进行操作的话，再去操作就可能会被提示登陆超时，请重新登陆，这个功能就很可能是 session 超时了。

cookie 和 session 与时效相关的一些方法如 Table1.1和 Table1.2所示。

[4] https://en.wikipedia.org/wiki/Session_(computer_science)

Method	Description
int getMaxAge()	Returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.an integer specifying the maximum age of the cookie in seconds; if negative, means the cookie is not stored; if zero, deletes the cookie.

Table 1.1: cookie 时效相关的方法

Method	Description
long getCreationTime()	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
long getLastAccessedTime()	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
int getMaxInactiveInterval()	Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
void setMaxInactiveInterval(int interval)	Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never timeout.

Table 1.2: session 时效相关的一些方法

1.4 Other

偶然间发现 stanford 的一个关于网站方面的资源, `https://crypto.stanford.edu/cs142/lectures/`; 还有 cookie 和 session 方面的`https://web.stanford.edu/~ouster/cgi-bin/cs142-fall10/lecture.php?topic=cookie`。

Chapter 2

servlet

servlet 是运行在 web 容器中的 java 程序，很适合用来进行处理 web 应用中的业务逻辑。

通过 servlet 和 jsp 的配合使用，可以使其各负其责：

- jsp

  负责显示页面

- servlet

  负责处理业务逻辑

jsp 和 servlet 的使用场景就是：jsp 负责页面显示、与用户交互，jsp 页面中一些需要处理的数据就送到 servlet 来处理。

2.1 servlet

通过编写一个负责处理注册信息的 servlet 来展示 servlet 的使用方法。

在 eclipse 中，右键当前的项目，选择新建，找到 servlet，即可建立一个 servlet(这是一个 java 类)。

建立完成后，打开刚才建立的 servlet，可以看到其中会有一些 eclipse 自动填写的代码，我们关注的是其中的两个方法 doGet 和 doPost。一般来说，servlet 是用来处理 jsp 页面传递过来的数据，而传递数据的方式通常情况下是通过表单来进行的，表单中的数据提交通常使用的是 post 方法。本部分涉及到两个文件：teacher.jsp 和 Reg.java，其中 teacher.jsp 负责页面的显示以及数据的收集，Reg.java 负责处理 teacher.jsp 页面传递过来的数据。

2.1.1 jsp 页面

teacher.jsp 页面中的关键代码如列表2.1所示。从列表2.1中可以看出，当前 form 的 action 是指向的 teacher_add.jsp 页面，这是在学习 sevlet 之前的数据处理方式，从现在开始需要把 action 的值改为新建的 servlet，即：action=”Reg”。

2.1.2 servlet 代码

因为 teacher.jsp 页面中 form 的数据提交方式为 post，所以我们要完成 Reg.java 中的 doPost() 方法。Reg.java 中的关键代码如列表2.2所示。

```
<form action="teacher_add.jsp" method="post">
<table>
<tr>
  <td>姓名：</td>
  <td><input type="text" name="nm"></td>
</tr>
<tr>
  <td>职工号：</td>
  <td><input type="number" name="staffid" min=20060001 ></td>
</tr>
<tr>
  <td><input type="submit"></td>
  <td><input type="reset"></td>
</tr>
</table>
</form>
```

Figure 2.1: teacher.jsp 中的关键代码

2.2 filter

filter 是 servlet 规范中定义的一种特殊类。filter 可以理解成介于客房端和目标资源之间的一个过滤器，即它会对客户端的请示进行过滤后才可以到达服务器上的目标资源，或者访问到目标资源后，对服务器端产生的响应进行处理后才送回客户端（这两个活动可以在一个过滤器中同时进行，即双向过滤）。filter 的示意图如 Figure 2.3所示。在用户注册的时候，如果想禁止某个姓名被使用，可以通过部署一个 filter 介于注册页面和业务逻辑处理模块之间。接下来的例子中，注册页面为 teacher.jsp，业务逻辑处理模块为 Reg.java（请注意这是一个 servlet）。

2.2.1 建立 filter

与建立一个 servlet 的方法类似，可以建立一个 filter，将其命名为 FilterReg.java。可以看到这个新建的 java 文件中，有若干个方法，其中的 doFilter 方法是我们需要关注的，因为过滤功能就是在此处实现的。Filter 文件 FilterReg.java 的代码如 Figure 2.4所示。Figure 2.4中的代码会判断用户提交的姓名，如果姓名为“zq”，则会截断请求，使得用户请求不能到达 Reg 这个 servlet，从而不能完成注册，达到了禁止特定用户名注册的目的。

2.2.2 部署 filter

在上一小节中，讲述了如何建立一个 filter，其实要使 filter 发挥作用还必须进行正确的配置。从 servlet 3.0 开始就不需要在 web.xml 文件中进行配置了，只需要在 servlet 和 filter 的源代码文件中进行声明即可，如 Figure 2.5和 Figure 2.6所示。

特别要注意的是 filter 和 servlet 里声明的 urlPatterns 需要保持一致：filter 里的 @WebFilter(filterName = "/FilterReg",urlPatterns = "/Reg") 和 servlet 里的 @WebServlet(name="reg",urlPatterns= "/Reg")。

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // 设置request传递过来值的编码，并获取传递值
        request.setCharacterEncoding("utf-8");

        String id = request.getParameter("staffid");
        String name = request.getParameter("nm");

        //get current date and time
        //LabDate ld = new LabDate();
        //String time = ld.getDtTm();

        //write to database
        //String[] fields= {"id","name","logDate"};
        //String[] values= new String[3];
        //values[0]=id;
        //values[1]=name;
        //values[2]=time;
        //Db db = new Db();
        //int i = db.writeDb("teachers", fields, values);
        //db.getClose();
        //String rz="";
        //if(i==1) {
        //      rz = "done! Will return to the former page in 3 seconds.";
        //}else {
        //      rz = "Something wrong! Will return in 3 seconds.";
        //}
        //response.getWriter().print(rz);
        //response.setHeader("refresh","3,URL=teacher.jsp");
}
```

Figure 2.2: Reg.java 中的关键代码

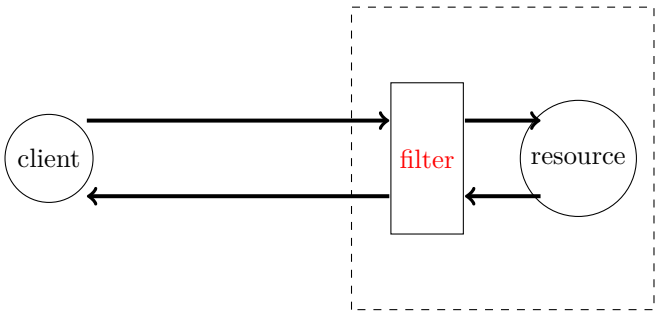

Figure 2.3: the position and function of a filter

```
public void doFilter(ServletRequest request,
 ServletResponse response, FilterChain chain)
 throws IOException, ServletException {
        // TODO Auto-generated method stub
        // place your code here
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;
        String id = request.getParameter("staffid");
        String name = request.getParameter("nm");
        //System.out.println("staffid is: "+id);
        System.out.println("filter says: the name is "+name);
        if(name.equals("zq")) {
          resp.getWriter().println("The name "+name+" is forbidden!");
        }else {
        // pass the request along the filter chain
          chain.doFilter(request, response);
          resp.setHeader("refresh", "3,URL=teacher.jsp");
        }
}
```

Figure 2.4: FilterReg.java 中的关键代码

```
package cs;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Reg
 */
@WebServlet(name="reg",urlPatterns= {"/Reg"})
public class Reg extends HttpServlet {
```

Figure 2.5: servlet 中的声明 @WebServlet

```
package filter;

import java.io.IOException;
import javax.servlet.DispatcherType;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet Filter implementation class FilterReg
 */
@WebFilter(filterName = "/FilterReg",urlPatterns = {"/Reg"})
```

Figure 2.6: filter 中的声明 @WebFilter

Listener 接口	**Event 类**
ServletContextListener	ServletContextEvent
ServletContexAttributeListener	ServletContextAttributeEvent
HttpSessionListener HttpSessionActivationListener	HttpSeesionEvent
HttpSessionAttributeListener HttpSessionBindingListener	HttpSessionBindingEvent
ServletRequestListener	ServletRequestEvent
ServletRequestAttributeListener	ServletRequestAttributeEvent

Table 2.1: Listener 接口与 Event 类

2.3 listener

listener 是 servlet 规范中定义的一种特殊类。listener 是监听器，其作用就是用来监听 servlet 容器中一些事件 (event) 的发生。从大的分类上来说，listener 可以监听以下三个对象的事件：

1. ServletContex
2. HttpSession
3. ServletRequest

listener 和 event 的对应关系，如 Table 2.1所示。

监听器通常可以被用来统计在线人数。接下来以此为例展示监听器的使用方法。

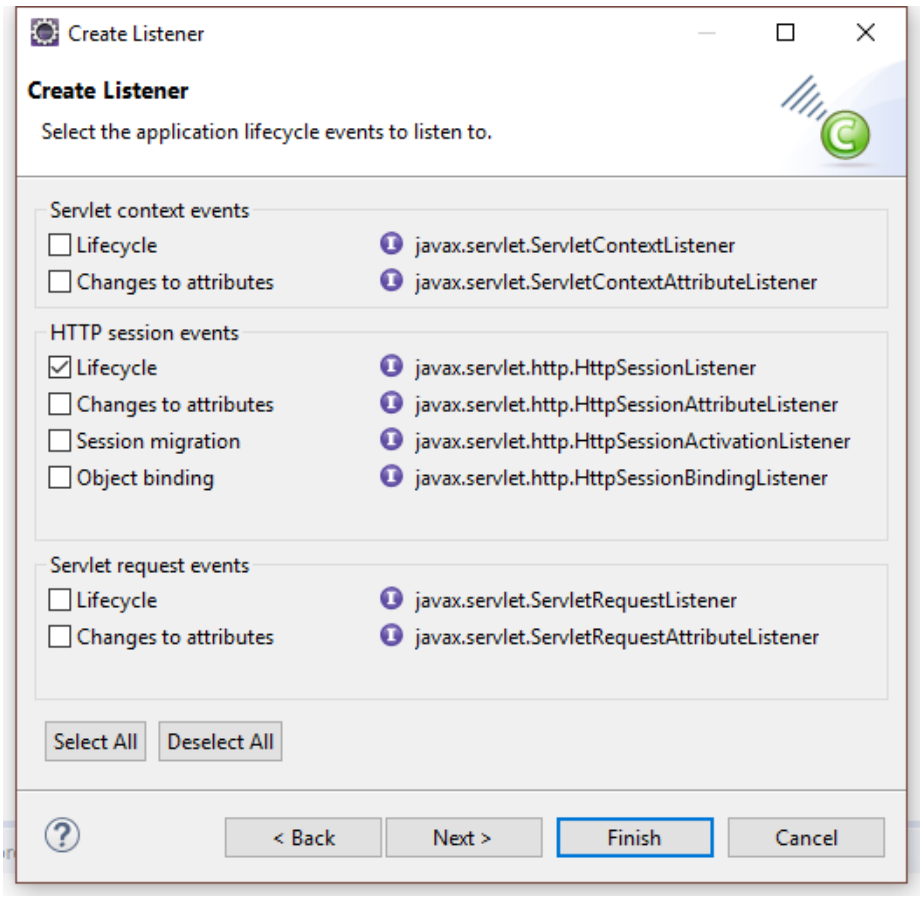

Figure 2.7: 选择 listener 监听事件

2.3.1 新建 listener

与 servlet 和 filter 的新建方式一样，新建一个 listener。这里需要注意的是，在下一步选择监听事件的时候，选择 HttpSessionEvent 里面的 lifecycle，如 Figure 2.7所示，因为这里统计在线人数是通过统计当前活动的 session 来计数的。

其代码如 Figure 2.8所示。

2.3.2 部署 listener

listener 的部署很简单，不需要另外配置，在创建 listener 的时候 eclipse 会自动帮助创建一条注释：@WebListener，这条注释就完成了 listener 的部署。

通常，每个 http 请求都会与一个 session 关联在一起。如果该请求没有与之相关联的 session，那么服务器会为其创建一个 session，此时会触发 sessionCreated 事件；该 session 会在用户不再活动的一定时间之后失效，或者用户主动注销（调用 session.invalidate() 方法），此时会触发 sessionDestoryed 事件，这两个事件会被监听器监听到。

```
package listener;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

import cs.LabDate;
/**
 * Application Lifecycle Listener implementation class OnlineNum
 *
 */
@WebListener
public class OnlineNum implements HttpSessionListener {
        private int count;

        public int getNum() {
                return count;
        }

        public synchronized void setCount(int c) {
                count+=c;
        }

    /**
     * Default constructor.
     */
    public OnlineNum() {
        // TODO Auto-generated constructor stub
        count=0;
        System.out.println("当前在线人数被初始化为0");
    }

        /**
     * @see HttpSessionListener#sessionCreated(HttpSessionEvent)
     */
    public void sessionCreated(HttpSessionEvent arg0)  {
         // TODO Auto-generated method stub
        String dt = new LabDate().getDtTm();
        String sid = arg0.getSession().getId();
//      count++;
        setCount(1);
        System.out.println("at "+dt+","+sid+" coming, 当前在线人数为: "+count);
    }

        /**
     * @see HttpSessionListener#sessionDestroyed(HttpSessionEvent)
     */
    public void sessionDestroyed(HttpSessionEvent arg0)
{
         // TODO Auto-generated method stub
//      count--;
        setCount(-1);
        String dt = new LabDate().getDtTm();
        String sid = arg0.getSession().getId();
        System.out.println("at "+dt+","+sid+" leave, 当前在线人数为: "+count);
    }
}
```

Figure 2.8: listener 代码