我们知道HTTP协议是无状态的，那么真的就没有办法让他有记忆了吗？我们可以通过Cookie、Session和Token来稍加弥补HTTP没有记忆的问题，本篇文章来简单讨论下这三位。

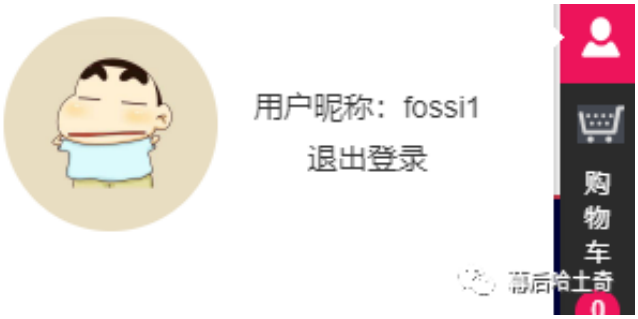
一、Cookie

我们先来聊聊Cookie。

用户浏览器第一次访问服务器的时候，服务器不认识本次请求身份，因此可以创建一个身份标识数据，格式是”key=value“放到 Set-Cookie 字段里，随着响应报文一同发给浏览器。

浏览器收到响应报文后，看到了Set-Cookie，知道是服务器返回的身份标识数据，就会保存起来。用户下次打开浏览器的时候，如果这个身份标识数据还在，浏览器则可以直接使用它。

我们拿用户信息的Cookie为例，比如登录商城后，前端用户名和头像是如何通过cookie来展示呢?



在JAVA后端代码中，通过登录接口拿到用户名密码后，进行校验，校验通过后生成Cookie写入HttpServletResponse返回给前端浏览器。当然了，前端JS也是可以设置Cookie的。

接口层:

```
@ApiOperation(value = "用户登录",notes = "用户登录",httpMethod =
"POST")
@PostMapping("/login")
public CommonJsonResult login(@ApiParam(name = "UserBO",value = "用
户登录实体",required = true)
```

```
                                @RequestBody UserBO userBO,
                                HttpServletRequest request,
                                HttpServletResponse response) {
    String username = userBO.getUsername();
    String password = userBO.getPassword();
//        1、判断用户名和密码不能为空
    if(StringUtils.isBlank(username) ||
StringUtils.isBlank(password)){
        return CommonJsonResult.errorMsg("用户名或密码不能为空");
    }
//        2、实现登录
    Users userResult =
userService.queryUserForLogin(username,password);
    if(userResult == null){
        return CommonJsonResult.errorMsg("用户名或密码不正确");
    }
//        3、去除一些敏感信息返回给前端
    userResult = setNullProperty(userResult);
//        4、设置cookie
    CookieUtils.setCookie(request,response, "user",
            JsonUtils.objectToJson(userResult),true);

    //TODO 生成用户TOKEN，存入redis
    //TODO 同步购物车数据
    return CommonJsonResult.ok(userResult);
}
```

其中工具类写Cookie的逻辑为：

```
private static final void doSetCookie(HttpServletRequest request,
HttpServletResponse response,
        String cookieName, String cookieValue, int cookieMaxage,
boolean isEncode) {
    try {
        if (cookieValue == null) {
            cookieValue = "";
        } else if (isEncode) {
            cookieValue = URLEncoder.encode(cookieValue, "utf-8");
        }
        Cookie cookie = new Cookie(cookieName, cookieValue);
        if (cookieMaxage > 0)
            cookie.setMaxAge(cookieMaxage);
        if (null != request) {// 设置域名的cookie
          String domainName = getDomainName(request);
            logger.info("========== domainName: {} ==========",
domainName);
```

```
            if (!"localhost".equals(domainName)) {
              cookie.setDomain(domainName);
            }
        }
        cookie.setPath("/");
        response.addCookie(cookie);
    } catch (Exception e) {
       e.printStackTrace();
    }
}
```

在这段代码中，我们对Cookie进行了一些属性上的设置。我们可以设置过期时间，一旦超过这个期限浏览器就认为是 Cookie 失效，在存储里删除，也不会发送给服务器。同时设定了domain和path，分别指定Cookie的所属域名和路径，浏览器在发送 Cookie 前会从 URI 中提取出 host 和 path 部分，对比 Cookie 的属性。如果不满足条件，就不会在请求头里发送 Cookie。

这样，浏览器就可以根据返回的Cookie信息存储起来，前端JS则读取此信息在页面展示，只要这个Cookie没有过期，下次打开浏览器就仍然可以看到用户信息。我们可以查看下Cookie的信息：

我们把这种前端缓存的小片段信息称为Cookie。

通过抓取登录报文，可以看到以下信息：

在响应报文首部字段中，我们会看到有个字段叫做：

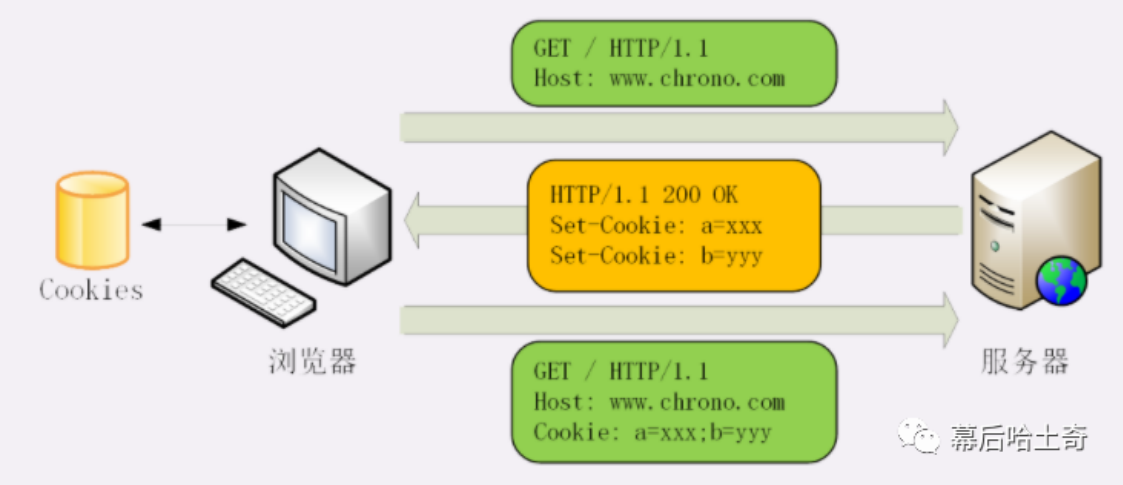
```
Set-Cookie:
user=%7B%22id%22%3A%22201118H62G5ZWPH0%22%2C%22username%22%3A%22fossi%22%2C%22password%22%3Anull%2C%22nickname%22%3A%22fossi%22%2C%22realname%22%3Anull%2C%22face%22%3A%22http%3A%2F%2Fbloghello.oursnail.cn%2Favatar.png%22%2C%22mobile%22%3Anull%2C%22email%22%3Anull%2C%22sex%22%3A2%2C%22birthday%22%3Anull%2C%22createdTime%22%3Anull%2C%22updatedTime%22%3Anull%7D; Domain=.oursnail.cn; Path=/
```

其中user后面一坨子就是url编码后的字符串（编码问题我们下篇文章就来探讨），
我们将其解码看下：

```
{"id":"201118H62G5ZWPH0","username":"fossi","password":null,"nickname":"fossi","realname":null,"face":"http://bloghello.oursnail.cn/avatar.png","mobile":null,"email":null,"sex":2,"birthday":null,"createdTime":null,"updatedTime":null}
```

原来就是关于用户的基本信息，比如用户名、头像，前端工程即可根据此cookie信息读取出来并展示了。

浏览器根据此响应字段写入Cookie，缓存起来用户相关的信息，前端项目在用到用户信息展示的地方都可以从这个Cookie中获取。



从这张图中我们也能够看到，Cookie 是由浏览器负责存储的，而不是操作系统。所以，它是“浏览器绑定”的，只能在本浏览器内生效。

如果你换个浏览器或者换台电脑，新的浏览器里没有服务器对应的 Cookie，就好像是脱掉了贴着纸条的衣服，“健忘”的服务器也就认不出来了，只能再走一遍 Set-Cookie 流程。

值得注意的是，如果中间存在代理服务器，很有可能对HTTP头部报文长度有限制，因此Cookie的长度也应该尽量减少避免不必要的麻烦。

此外，Cookie本身确实会带来一些问题：

- Cookie会被附加在每个HTTP请求头中，无形中增加了流量；
- 由于在HTTP请求中的Cookie是明文传输的，存在安全性问题，可以使用HTTPS来解决；
- Cookie的大小不应超过4KB，原因是某些浏览器或者代理服务器有限制，所以对复杂的存储需求来说是不够用的；

二、Session

Session是另一种记录客户状态的机制，保存在服务器上，客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。

客户端浏览器再次访问时只需要从该Session中查找该客户的状态即可。好了，Session就是这么个情况，实际上就是用ConcurrentHashMap结构把会话信息存储在内存中。相同的会话过来时，服务器可以根据SessonID找到对应的用户信息。

Session 可以与 Cookie 一起使用，下面展示Session Cookie的流程，即会话Cookie，

- 首先，客户端会发送一个http请求到服务器端。

- 服务器端接受客户端请求后，建立一个session对象，并发送一个http响应到客户端，这个响应头，其中就包含Set-Cookie头部。该头部包含了sessionId：Set-Cookie：JSESSIONID=XXXXXXX
- 在客户端发起的第二次请求，假如服务器给了set-Cookie，浏览器会自动在请求头中添加cookie
- 服务器接收请求，分解cookie，验证信息，核对成功后返回response给客户端

三、Session VS Cookie

- 存放位置不同：Cookie在客户端浏览器，而Session存放在服务端。
- 安全性不同：基于第一点，因此Session更加安全。
- 对服务器压力不同：基于第一点，Session存放在内存，当客户端连接很多时，会占用很多内存资源，对服务器压力较大。
- 存储大小不同：基于上一点，**单个 Cookie 保存的数据不能超过 4K**，Session 可存储数据远高于 Cookie，但是当访问量过多，会占用过多的服务器资源。
- 有效期不同：Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般失效时间较短，客户端关闭（默认情况下）或者 Session 超时都会失效。
- 存取值的类型不同：Cookie 只支持存字符串数据，想要设置其他类型的数据，需要将其转换成字符串，Session 可以存任意数据类型。

不过值得注意的是，cookie有可能会走向灭亡，比如Chrome浏览器就将逐步淘汰现有的第三方Cookie技术，基于cookie的一些广告推荐机制可能将发生变化，何去何从让我们拭目以待。

四、Token

熟悉Session的同学知道，Session在一台服务器上逻辑是没问题的，但是当遇到了集群或者分布式的时候，就会有问题。因为多服务器不共享Session，而请求可能会打到任意一台机器上，就会出现读取不到Session的问题。

这个时候可以使用Token来解决。

Token 也称为令牌，token在客户端一般存放于localStorage，cookie，或sessionStorage中。在服务器一般存于数据库中。

token 的认证流程与cookie很相似：

- 用户登录，成功后服务器返回Token给客户端。
- 客户端收到数据后保存在客户端
- 客户端再次访问服务器，将token放入headers中
- 服务器端采用filter过滤器校验。校验成功则返回请求数据，校验失败则返回错误码

面对集群或者分布式的场景时，就可以根据存储在数据库的Token进行校验，而不会有Session这种问题。

此外，如果你的用户数据可能需要和第三方共享，或者允许第三方调用 API 接口，可以用 Token 。

而Session只提供一种简单的认证，即只要有此 SessionID ，即认为有此 User 的全部权利。是需要严格保密的，这个数据应该只保存在站方，不应该共享给其它网站或者第三方 App。

最后小小总结下：

- session存储于服务器，可以理解为一个状态列表，拥有一个唯一识别符号sessionId，通常存放于cookie中。服务器收到cookie后解析出sessionId，再去session列表中查找，才能找到相应session，依赖cookie。
- cookie存储在客户端，可以通过HTTP协议的Set-Cookie响应头让浏览器自动添加。
- token是指令牌，无状态，服务端通过token可判断出该用户是否是合法用户。