Passport用户认证

# 1 介绍

|  |
| --- |
| 1 在购物车之前的功能，不需要对用户登录进行判定  2 在购物车之后的功能，必须要求用户登录的判定通过  3 在购物车功能中，必须对用户登录进行判定，但是判定失败也可以继续使用(程序执行另一个分支) |

# 2 登录的入口

直接点击登录功能

登录

结算时被登录功能拦截

# 3 认证中心的设计

业务功能

请求

用户服务

拦截器

具体的业务

认证中心（只负责护照的颁发和护照信息的验证）

Redis

Mysql

|  |
| --- |
| 1 给用户颁发通行证(token)  2 验证其他业务功能接收token(用户访问所携带的)的真伪 |

# 4 认证中心的整合

|  |  |  |
| --- | --- | --- |
| 1 引入认证中心的静态资源    2 在search模块的页面点击登录链接上加入认证中心的url    3 增加一个测试的结算功能(真正的结算功能在dd模块)      4 加入拦截器(在web-util中加入拦截器，让所有请求到web的请求都被拦截器拦截)    5 加入拦截器   |  | | --- | | @Component **public class** AuthInterceptor **extends** HandlerInterceptorAdapter {  **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {  *// 拦截代码* **return true**;  } } |   6 加入拦截器的整合类   |  | | --- | | @Configuration **public class** WebMvcConfiguration **extends** WebMvcConfigurerAdapter {  @Autowired  AuthInterceptor **authInterceptor**;  @Override  **public void** addInterceptors(InterceptorRegistry registry){  registry.addInterceptor(**authInterceptor**).addPathPatterns(**"/\*\*"**);  **super**.addInterceptors(registry);  } } |   7 有的方法走拦截器，有的方法不走拦截器    除了可以通过web模块是否扫描拦截器来决定拦截器的使用之外  还可以通过注解的方式来标识具体的方法是否需要通过拦截器  @LoginRequired  第一类方法：不需要进行拦截的方法(没有拦截器注解)，直接放行，不加@LoginRequired  第二类方法：需要拦截但是拦截校验失败(用户没登陆或者登录过期了)也可以继续访问的方法，比如说购物车中的所有方法@LoginRequired(loginSuccess=false)  第三类方法：需要拦截，并且拦截校验一定要通过(用户登录成功了)才能访问的方法@LoginRequired(loginSuccess=true) |

用户认证

Jwt

Jwt

Userinfo

Redis

Jwt

Userinfo

Session共享

业务模块

jsessionid

浏览器

jsessionid

Token

业务模块

Token

浏览器

Token

E

e

e

Session

Session

Redis

Session们

(user)

认证中心

Token

# 5 用jwt实现用户登录的校验(token)

服务器

Key

UserInfo

token

# 6 passport登录功能

|  |
| --- |
| 1 点击登录按钮，在登录页面输入用户名和密码通过用户名和密码验证用户登录是否成功后，生成jwt的token返回给search.gmall.com/index，在拦截器里面将返回的token写入cookie  2 点击结算按钮，拦截器拦截请求  A该用户没有登录，并且结算请求时必须登录的，将用户打回认证中心进行登录  B该用户没有登录，并且结算请求时没有登录也可以访问，放行  3 在被拦截登录后，返回原始请求(携带登录成功后颁发的token)  4 原始请求的拦截器第二次拦截请求，拦截请求后，将返回的token写入cookie |

认证中心

业务模块

to

浏览器(token)

# 7 从首页点击登录的流程

主动登录

|  |
| --- |
| 1 首页访问登录页，携带ReturnUrl会跳地址    2 登录页保存会跳地址    3 登录页通过异步方法验证用户名和密码    4 验证通过后颁发token给异步ajax  5 异步ajax得到token，然后根据会跳地址ReturnUrl访问原始功能 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 请求原始应用  1 被拦截器拦截  2 拦截器需要判断注解  是否需要进行身份验证  是否必须通过身份验证   |  |  |  | | --- | --- | --- | |  | 老token空 | 老token不空 | | 新token空 | 从未登录过 | 之前登录过 | | 新token不空 | 刚刚登录 | 过期 |   3 如果是必须登录但是没有登录，会被重定向认证中心进行登录  4 在认证中心登陆后，进行重新的请求，请求原始应用  输入用户名和密码后，进行第二次请求(请求原始地址)  5 验证  通过httpClient(apach一个通用工具类)  6 验证结果结合注解的情况 |

# 8 用户userService服务的对接

|  |
| --- |
| 1 认证中心调用用户服务进行login和信息查询的处理  2 用户服务可以调用缓存  User:userName:password  存储密码  User:password:info  存储用户信息  K：username+password  V：用户信息  User:memberId:info  根据id存储用户信息 |

request.getRemoteAddr()

Nginx

request.getRemoteAddr()

request

Login->token(ip)

verify->token(ip)

request

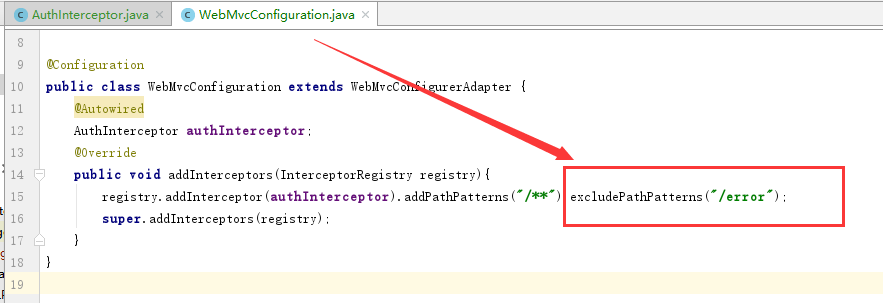
X拦截器

request



# 9 springboot默认有全局异常处理(error)

排除全局异常处理



# 10 第三方社交登录

王

刘

宋

刘

1 授权操作

第三方服务器

2 Code

3 Code->access\_token

4 Access\_token

用户数据

Gmall

Code->access\_token

|  |
| --- |
| 1 授权  2 返回授权code  3 交换access\_token  4 用access\_token调用用户数据 |

# 11 准备工作

|  |
| --- |
| 1    2    3    4    5    6    7    8 获得自己的授权id和授权密钥    9 修改回调地址 |

# 12 四步完成授权操作

|  |  |
| --- | --- |
| 1 请求授权地址，用户和第三方进行授权协议签订  https://api.weibo.com/oauth2/authorize?client\_id=187638711&response\_type=code&redirect\_uri=http://passport.gmall.com:8085/vlogin    2 通过回调地址获得授权码  <http://网站回调地址?code=fef987b3f9ad1169955840b467bfc661>    3 用授权码code交换access\_token   |  | | --- | | *// 3 换取access\_token // client\_secret=a79777bba04ac70d973ee002d27ed58c* String s3 = **"https://api.weibo.com/oauth2/access\_token?"**;*//?client\_id=187638711&client\_secret=a79777bba04ac70d973ee002d27ed58c&grant\_type=authorization\_code&redirect\_uri=http://passport.gmall.com:8085/vlogin&code=CODE";* Map<String,String> paramMap = **new** HashMap<>(); paramMap.put(**"client\_id"**,**"187638711"**); paramMap.put(**"client\_secret"**,**"a79777bba04ac70d973ee002d27ed58c"**); paramMap.put(**"grant\_type"**,**"authorization\_code"**); paramMap.put(**"redirect\_uri"**,**"http://passport.gmall.com:8085/vlogin"**); paramMap.put(**"code"**,**"117df4b8b5d435c9dccd93370f65b02b"**);*// 授权有效期内可以使用，没新生成一次授权码，说明用户对第三方数据进行重启授权，之前的access\_token和授权码全部过期* String access\_token\_json = HttpclientUtil.*doPost*(s3, paramMap); |   第三步返回access\_token的同时，会将用户id信息返回给调用者  4 用access\_token查询用户数据 |

# 13 社交登录和项目的整合

|  |
| --- |
| 1 用户在passport认证中心准备登录时可以点击第三方社交登录按钮  跳转到  <https://api.weibo.com/oauth2/authorize?client_id=YOUR_CLIENT_ID&response_type=code&redirect_uri=YOUR_REGISTERED_REDIRECT_URI>  引导用户进入第三方网站的授权页面  2 用户在授权完成后  <http://网站回调地址?code=fef987b3f9ad1169955840b467bfc661>  第三方网站将调用我们在第三方网站所创建的应用的回调地址，将授权码写入到我们的服务器中  我们网站需要将授权码保存到我们的数据库中  3 我们通过授权码code发送post请求到第三方网站，换取access\_token  https://api.weibo.com/oauth2/access\_token?client\_id=YOUR\_CLIENT\_ID&client\_secret=YOUR\_CLIENT\_SECRET&grant\_type=authorization\_code&redirect\_uri=YOUR\_REGISTERED\_REDIRECT\_URI&code=CODE  换取来的token，和相关信息写入我们用户数据库  4 在用户使用的过程中通过access\_token获取用户数据(第三方网站的用户数据)  <https://api.weibo.com/2/users/show.json?access_token=2.00S4TE5GH4psNBf534795d1677YqPB&uid=1>  通过第三方社交登录的用户在本网站的信息需要补全，在该用户使用网站的级别高的功能时，要求用户补全信息甚至进行实名认证 |

# 14 整合社交登录功能

|  |
| --- |
| 1 地址一：  引导用户与第三方平台进行授权交互地址（第三方平台地址，页面上有我们自己的授权应用的信息）  <https://api.weibo.com/oauth2/authorize>  2 地址二  授权成功后，第三方平台将授权码回调给我们的地址(我们自己的地址)  3 地址三  使用授权码code交换access\_token的地址，需要加入client\_secret应用密钥，并且使用post请求进行交换，保证access\_token的安全性  https://api.weibo.com/oauth2/access\_token  4 地址四  <https://api.weibo.com/2/users/show.json>  使用access\_token换取用户信息的地址 |