

第9章 用户认证

1 用户认证分析



上面流程图描述了用户要操作的各个微服务，用户查看个人信息需要访问客户微服务，下单需要访问订单微服务，秒杀抢购商品需要访问秒杀微服务。每个服务都需要认证用户的身份，身份认证成功后，需要识别用户的角色然后授权访问对应的功能。

1.1 单点登录

用户访问的项目中，至少有3个微服务需要识别用户身份，如果用户访问每个微服务都登录一次就太麻烦了，为了提高用户的体验，我们需要实现让用户在一个系统中登录，其他任意受信任的系统都可以访问，这个功能就叫单点登录。

单点登录（Single Sign On），简称为 SSO，是目前比较流行的企业业务整合的解决方案之一。SSO的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统

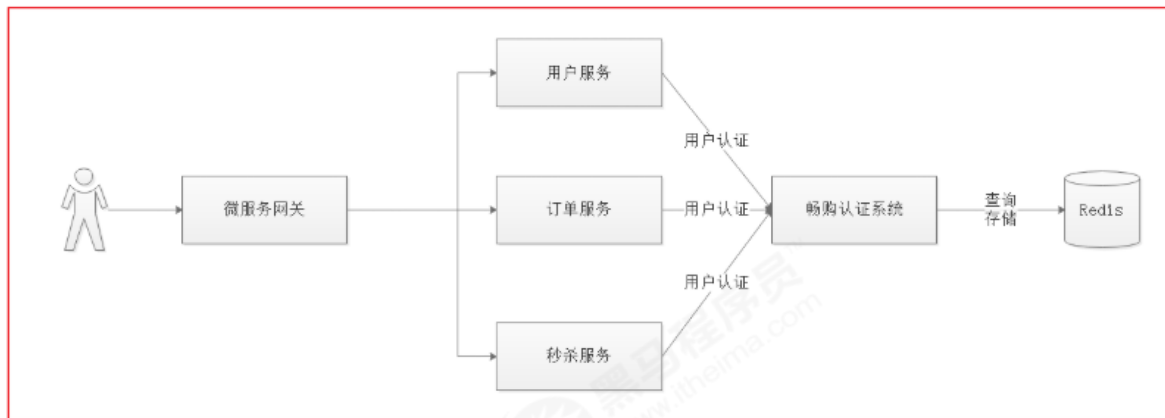
1.2 第三方账号登录

随着国内及国外巨头们的平台开放战略以及移动互联网的发展，第三方登录已经不是一个陌生的产品设计概念了。所谓的第三方登录，是说基于用户在第三方平台上已有的账号和密码来快速完成己方应用的登录或者注册的功能。而这里的第三方平台，一般是已经拥有大量用户的平台，国外的比如 Facebook，Twitter等，国内的比如微博、微信、QQ等。



2 认证解决方案

2.1 单点登录技术方案



Java中有很多用户认证的框架都可以实现单点登录：

- 1、Apache Shiro.
- 2、CAS
- 3、Spring security

2.2 第三方登录技术方案

2.2.1 OAuth2认证流程

第三方认证技术方案最主要是解决认证协议的通用标准问题，因为要实现跨系统认证，各系统之间要遵循一定的接口协议。OAUTH协议为用户资源的授权提供了一个安全的、开放而又简易的标准。同时，任何第三方都可以使用OAUTH认证服务，任何服务提供商都可以实现自身的OAUTH认证服务，因而OAUTH是开放的。业界提供了OAUTH的多种实现如PHP、JavaScript，Java，Ruby等各种语言开发包，大大节约了程序员的时间，因而OAUTH是简易的。互联网很多服务如Open API，很多大公司如Google，Yahoo，Microsoft等都提供了OAUTH认证服务，这些都足以说明OAUTH标准逐渐成为开放资源授权的标准。OAuth协议目前发展到2.0版本，1.0版本过于复杂，2.0版本已得到广泛应用。参考：<https://baike.baidu.com/item/oAuth/7153134?fr=aladdin> OAuth协议：<https://tools.ietf.org/html/rfc6749> 下边分析一个OAuth2认证的例子，黑马程序员网站使用微信认证的过程：



1. 客户端请求第三方授权

用户进入黑马程序员的登录页面，点击微信的图标以微信账号登录系统，用户是自己在微信里信息的资源拥有者。



点击“用QQ账号登录”出现一个二维码，此时用户扫描二维码，开始给黑马程序员授权。



2.资源拥有者同意给客户端授权

资源拥有者扫描二维码表示资源拥有者同意给客户端授权，微信会对资源拥有者的身份进行验证，验证通过后，QQ会询问用户是否给授权黑马程序员访问自己的QQ数据，用户点击“确认登录”表示同意授权，QQ认证服务器会颁发一个授权码，并重定向到黑马程序员的网站。



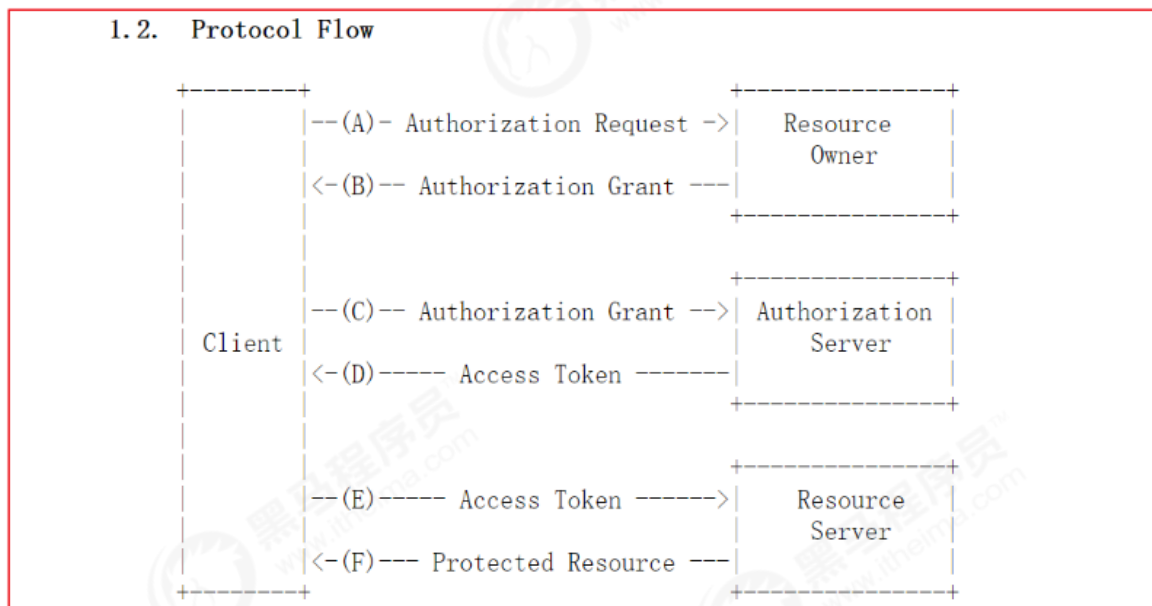
3.客户端获取到授权码，请求认证服务器申请令牌 此过程用户看不到，客户端应用程序请求认证服务器，请求携带授权码。

员看到已经登录成功。

5.客户端请求资源服务器的资源 客户端携带令牌访问资源服务器的资源。黑马程序员网站携带令牌请求访问微信服务器获取用户的基本信息。

6.资源服务器返回受保护资源 资源服务器校验令牌的合法性，如果合法则向用户响应资源信息内容。
注意：资源服务器和认证服务器可以是一个服务也可以分开的服务，如果是分开的服务资源服务器通常要请求认证 服务器来校验令牌的合法性。

Oauth2.0认证流程如下：引自Oauth2.0协议rfc6749 <https://tools.ietf.org/html/rfc6749>



Oauth2包括以下角色：

- 1、客户端 本身不存储资源，需要通过资源拥有者的授权去请求资源服务器的资源，比如：畅购Android客户端、畅购Web客户端（浏览器端）、微信客户端等。
- 2、资源拥有者 通常为用户，也可以是应用程序，即该资源的拥有者。
- 3、授权服务器（也称认证服务器）用来对资源拥有的身份进行认证、对访问资源进行授权。客户端要想访问资源需要通过认证服务器由资源拥有者授权后方可访问。
- 4、资源服务器 存储资源的服务器，比如，畅购用户管理服务器存储了畅购的用户信息，微信的资源服务存储了微信的用户信息等。客户端最终访问资源服务器获取资源信息。

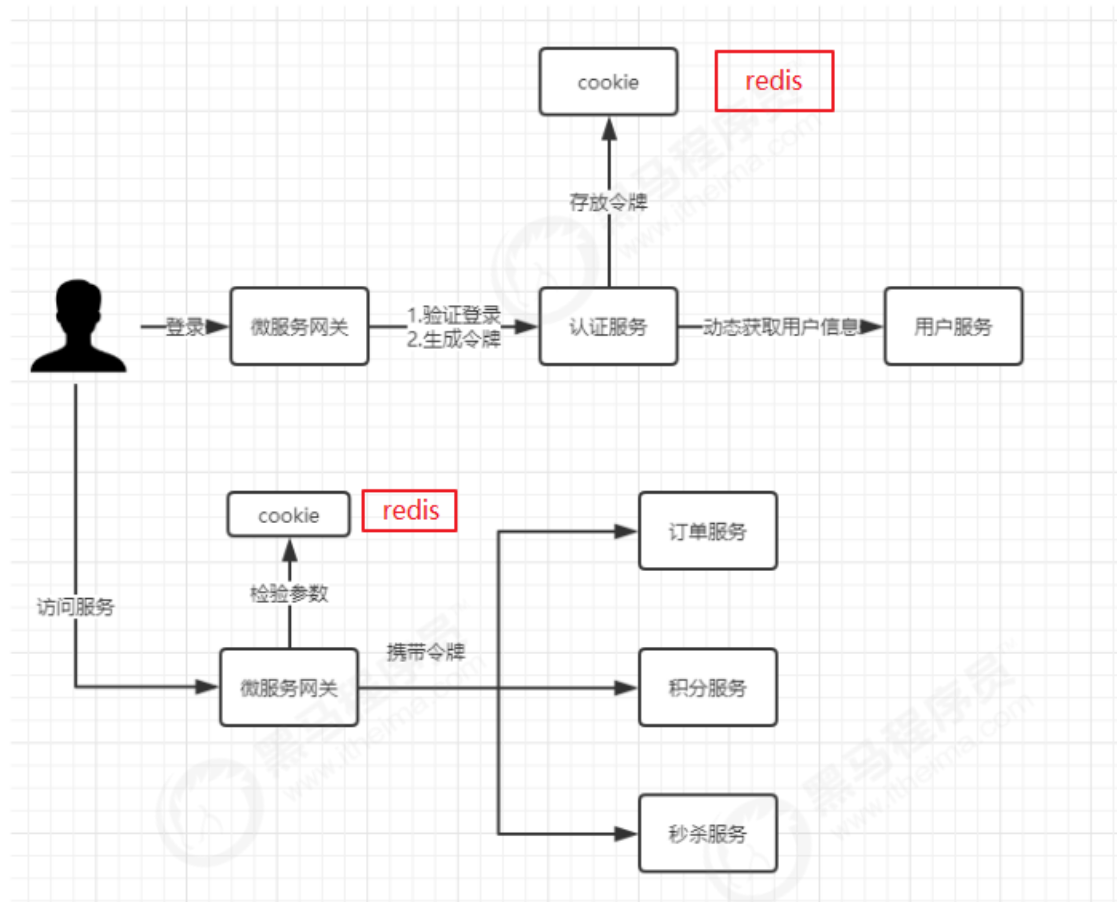
2.2.2 Oauth2在项目的应用

Oauth2是一个标准的开放的授权协议，应用程序可以根据自己的要求去使用Oauth2，项目中使用Oauth2可以实现如下功能：

- 1、本系统访问第三方系统的资源
- 2、外部系统访问本系统的资源
- 3、本系统前端（客户端）访问本系统后端微服务的资源。
- 4、本系统微服务之间访问资源，例如：微服务A访问微服务B的资源，B访问A的资源。

2.3 Spring security OAuth2认证解决方案

本项目采用 Spring security + OAuth2+JWT完成用户认证及用户授权，Spring security 是一个强大的和高度可定制的身份验证和访问控制框架，Spring security 框架集成了OAuth2协议，下图是项目认证架构图：



- 1、用户请求认证服务完成认证。
- 2、认证服务下发用户身份令牌，拥有身份令牌表示身份合法。
- 3、用户携带令牌请求资源服务，请求资源服务必先经过网关。
- 4、网关校验用户身份令牌的合法，不合法表示用户没有登录，如果合法则放行继续访问。
- 5、资源服务获取令牌，根据令牌完成授权。
- 6、资源服务完成授权则响应资源信息。

3 Jwt令牌回顾

JSON Web Token (JWT) 是一个开放的行业标准 (RFC 7519)，它定义了一种简介的、自包含的协议格式，用于在通信双方传递json对象，传递的信息经过数字签名可以被验证和信任。JWT可以使用 HMAC算法或使用RSA的公钥/私钥对来签名，防止被篡改。

官网：<https://jwt.io/>

标准：<https://tools.ietf.org/html/rfc7519>

JWT令牌的优点：

- 3、通过非对称加密算法及数字签名技术，JWT防止篡改，安全性高。
- 4、资源服务使用JWT可不依赖认证服务即可完成授权。

缺点：

- 1、JWT令牌较长，占存储空间比较大。

3.1 令牌结构

通过学习JWT令牌结构为自定义jwt令牌打好基础。

JWT令牌由三部分组成，每部分中间使用点(.)分隔，比如：xxxxx.yyyyy.zzzzz

Header

头部包括令牌的类型（即JWT）及使用的哈希算法（如HMAC SHA256或RSA）

一个例子如下：

下边是Header部分的内容

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

将上边的内容使用Base64Url编码，得到一个字符串就是JWT令牌的第一部分。

Payload

第二部分是负载，内容也是一个json对象，它是存放有效信息的地方，它可以存放jwt提供的现成字段，比如：iss（签发者），exp（过期时间戳），sub（面向的用户）等，也可自定义字段。

此部分不建议存放敏感信息，因为此部分可以解码还原原始内容。

最后将第二部分负载使用Base64Url编码，得到一个字符串就是JWT令牌的第二部分。

一个例子：

```
{
  "sub": "1234567890",
  "name": "456",
  "admin": true
}
```

Signature

第三部分是签名，此部分用于防止jwt内容被篡改。

这个部分使用base64url将前两部分进行编码，编码后使用点(.)连接组成字符串，最后使用header中声明签名算法进行签名。

一个例子：



```
base64UrlEncode(header) + "." +  
base64UrlEncode(payload),  
secret)
```

base64UrlEncode(header) : jwt令牌的第一部分。

base64UrlEncode(payload) : jwt令牌的第二部分。

secret : 签名所使用的密钥。

3.2 生成私钥公钥

JWT令牌生成采用非对称加密算法

1、生成密钥证书 下边命令生成密钥证书，采用RSA 算法每个证书包含公钥和私钥

```
keytool -genkeypair -alias changgou -keyalg RSA -keypass changgou -keystore  
changgou.jks -storepass changgou
```

Keytool 是一个java提供的证书管理工具

```
-alias: 密钥的别名  
-keyalg: 使用的hash算法  
-keypass: 密钥的访问密码  
-keystore: 密钥库文件名，changgou.jks保存了生成的证书  
-storepass: 密钥库的访问密码
```

查询证书信息：

```
keytool -list -keystore changgou.jks
```

2、导出公钥

openssl是一个加解密工具包，这里使用openssl来导出公钥信息。

安装 openssl : <http://slproweb.com/products/Win32OpenSSL.html>

安装资料目录下的Win64OpenSSL-1_1_1b.exe

配置openssl的path环境变量，

cmd进入changgou.jks文件所在目录执行如下命令：

```
keytool -list -rfc --keystore changgou.jks | openssl x509 -inform pem -pubkey
```



```
t47whAaRkRu/8kePM+X8760UGU0RMwGti6Z9y3LQ0RvK6I0brXmbGB/RsN38Pvnh
cP8ZfxGUH26kX0RK+t1rxcrG+HkPYOH4XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEm
oLKymSVtviG5f1TpDprT25unWgE4md1kthRWXOnfWHATVY7Y/r4obiOL1ms5bEa/
iNKotQNNvIAktjBM4R1IDWMA6dmz+1HtLtqDD2LF1qwoiSIHI75LQZ/CNYaHcfZS
xtOydpNKq8eb1/PGiLNo1D4La2zf0/1d1cr5mkesV570NxRmU1tFm8Zd3MZ1Zmyv
9QIDAQAB
-----END PUBLIC KEY-----
-----BEGIN CERTIFICATE-----
MIIDXzCCAkegAwIBAgIEQkT0oJANBgkqhkiG9w0BAQsFADBgMQswCQYDVQQGEwJj
bJELMAkGA1UEBMCZ2QxZCZAJBgNVBACTAjEwMDYwYDELMakGA1UEBhMCY24xZCZAJBgNVBAGTAmdk
MQswCQYDVQQHEwJzejERMA8GA1UEChMIY2hhbmdnb3UxETAPBgNVBAsTCGNoYW50ZW50
Z291MRREwDwDVQQDEWhjaGFuZ2dvZ2d0dCCASiWdQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBALxbBImi74o/QtTM/qMgJre08IQGkZEBv/JHjzP1/O+tFB1NETMBrYum
fcty0NEbyuiNC615mxgf0bDd/D1Z4XD/GXSR1B9upF9ESvrZa8XKxvh5D2Dh+Fzw
C/ENZbtZ/cd7S3CD8avGwbbiMimBjQcysjLL074huX5U6Q6a09ubploBOJndZLYU
V1zp31hwE1W02P6+KG4ji9ZkuWxGv4jSgLUdZ7yACrYwTOE2SA1jGunZs/pr7S7a
gw9ixdasKikiBy0+S0GfwjWghwn2UbsTsnATsqvHm9fzxoizaJQ+C2ts39P9XZK
+ZpHrFee9DcUZ1NbRZvGXdzGZWsr/UCaWEAAAMhMB8wHQYDVRO0BBYEFd1S24hM
6d3CpJuw4QNA91iPVq+TMA0GCSqGSIb3DQEBwUAA4IBAQBvLqfGerXnoxT4roBo
8/PY7+Irr1lQfpQWSYWiDeY7+ek5c7AGG0k5dVC4GxdgsCdCmNM9V1cS7r2j1mD1
t04g2MABbYWKaVunSdYm73KhF4ktz+QPrBnUKisZnE6wciP1+MBE6J61uJR47Sx
Cd/bbqwr7JjfpvpeAux3NVNaL6YN8K4M4Zwapu+GtiwNyoBXk6bTPp4fdgWkpW6X
i+DX64vCOWwqiQSRMSIZ+1RzPBRjaOXk1JuFbYIbnez9eVd1I264ziUIUvBQfES
bqZrk/bvyt51PnmpZJBthiEe/C3mMmsLwsUTN7jsI1dtJaZ11ffmqar3bqxTxRdc
#60x
-----END CERTIFICATE-----
C:\Users\71075\Desktop\cret>
```

下面段内容是公钥

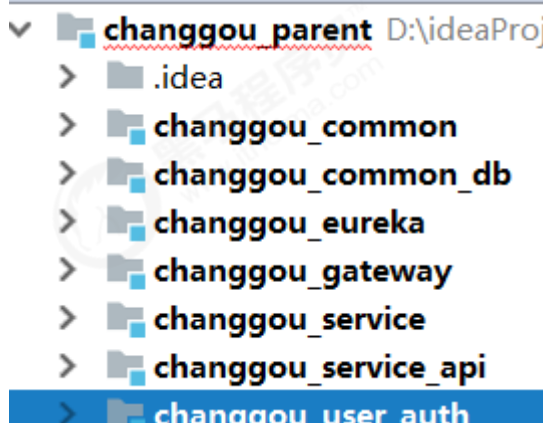
```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCGKCAQEAVFsEiaLvij9C1Mz+oyAm
t47whAaRkRu/8kePM+X8760UGU0RMwGti6Z9y3LQ0RvK6I0brXmbGB/RsN38Pvnh
cP8ZfxGUH26kX0RK+t1rxcrG+HkPYOH4XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEm
oLKymSVtviG5f1TpDprT25unWgE4md1kthRWXOnfWHATVY7Y/r4obiOL1ms5bEa/
iNKotQNNvIAktjBM4R1IDWMA6dmz+1HtLtqDD2LF1qwoiSIHI75LQZ/CNYaHcfZS
xtOydpNKq8eb1/PGiLNo1D4La2zf0/1d1cr5mkesV570NxRmU1tFm8Zd3MZ1Zmyv
9QIDAQAB
-----END PUBLIC KEY-----
```

将上边的公钥拷贝到文本public.key文件中，合并为一行,可以将它放到需要实现授权认证的工程中。

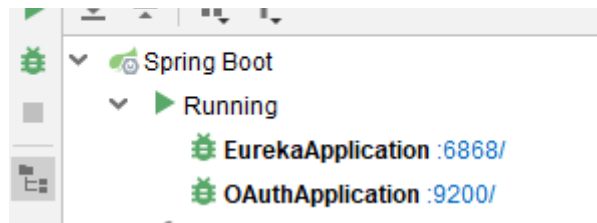
3.3 基于私钥生成jwt令牌

3.3.1 导入认证服务

1) 将课件中 changgou_user_auth 的工程导入到项目中去，如下图：



2) 启动eureka，再启动认证服务



3.3.2 认证服务中创建测试类

```
public class CreateJwtTest {

    /**
     * 创建令牌测试
     */
    @Test
    public void testCreateToken(){
        //证书文件路径
        String key_location="changgou.jks";
        //密钥库密码
        String key_password="changgou";
        //密钥密码
        String keypwd = "changgou";
        //密钥别名
        String alias = "changgou";

        //访问证书路径
        ClassPathResource resource = new ClassPathResource(key_location);

        //创建密钥工厂
        KeyStoreKeyFactory keyStoreKeyFactory = new
        KeyStoreKeyFactory(resource, key_password.toCharArray());

        //读取密钥对(公钥、私钥)
        KeyPair keyPair =
        keyStoreKeyFactory.getKeyPair(alias, keypwd.toCharArray());

        //获取私钥
        RSAPrivateKey rsaPrivate = (RSAPrivateKey) keyPair.getPrivate();

        //定义Payload
        Map<String, Object> tokenMap = new HashMap<>();
        tokenMap.put("id", "1");
        tokenMap.put("name", "itheima");
        tokenMap.put("roles", "ROLE_VIP,ROLE_USER");

        //生成Jwt令牌
        Jwt jwt = JwtHelper.encode(JSON.toJSONString(tokenMap), new
        RsaSigner(rsaPrivate));

        //取出令牌
        String encoded = jwt.getEncoded();
        System.out.println(encoded);
    }
}
```



3.4 基于公钥解析jwt令牌

上面创建令牌后，我们可以对JWT令牌进行解析，这里解析需要用到公钥，我们可以将之前生成的公钥public.key拷贝出来用字符串变量token存储，然后通过公钥解密。

在changgou-user-oauth创建测试类com.changgou.token.ParseJwtTest实现解析校验令牌数据，代码如下：

```
public class ParseJwtTest {

    /**
     * 校验令牌
     */
    @Test
    public void testParseToken(){
        //令牌
        String token =
            "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlcyI6IjJPTeVfVklQLFJPTeVfVWVNFUIiIsIm5hbWUiOiJpdGhlaw1hIiwiaWQiOiIxIn0.IR9Qu9ZqYZ2gu2qgAzyT38UhEeL40i69ko-dzC_P9-vjz40hwZDqx18wZ-w2WAw1eWGIHV1EYDjg0-eilogJZ5Uikyww1bewXCpv1M-ZRtYQQqHFTlfdiVcFetyTayaskwa-x_BVS4pTWaskiaIKbKR4KcME2E5o1rEek-3YPkqAiZ6WP1UOmpacJDaaFSdninqG0gzSCuGvLuG40x0Npgfk7mPOecsIi5cbJE1pdYUsCr9oXc53R0yfvYphjzV7c2D5eIZu3leUPXRvvVAPJFECSBiisxUSEEigpmuQhaFzd1g-yJ1WQrixFvehMeLX2XU6W1n1L5ARTpqf_Jjiw";

        //公钥
        String publickey = "-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAVFSeialVij9C1Mz+oyAmt47whAarkRu/8keP
M+x8760UGU0RMWgti6Z9y3LQ0RvK6I0brXmbGB/RsN38PvnhcP8ZfxGUH26kX0RK+t1rxcrG+HkPYOH4
XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEmoLKyMsvTviG5f1TpDprT25unWgE4md1kthRWXOnfWHATVY7Y
/r4obiOL1ms5bEa/inKotQNNvIAktjBM4RlIDWma6dmz+1HtLtqDD2LF1qwoiSIHI75LQZ/CNYaHcfZS
xtOydpNKq8eb1/PGiLNo1D4La2zf0/1dlcr5mkesV570NxRmU1tFm8Zd3MZ1Zmyv9QIDAQAB-----END
PUBLIC KEY-----";

        //校验Jwt
        Jwt jwt = JwtHelper.decodeAndVerify(token, new Rsaverifier(publickey));

        //获取Jwt原始内容
        String claims = jwt.getClaims();
        System.out.println(claims);
        //jwt令牌
        String encoded = jwt.getEncoded();
        System.out.println(encoded);
    }
}
```

4 OAuth2.0入门

4.1 准备工作

1) 搭建认证服务器之前，先在用户系统表结构中增加如下表结构：



```
`resource_ids` varchar(256) DEFAULT NULL,  
`client_secret` varchar(256) DEFAULT NULL COMMENT '客户端密钥，  
BCryptPasswordEncoder加密',  
`scope` varchar(256) DEFAULT NULL COMMENT '对应的范围',  
`authorized_grant_types` varchar(256) DEFAULT NULL COMMENT '认证模式',  
`web_server_redirect_uri` varchar(256) DEFAULT NULL COMMENT '认证后重定向地址',  
`authorities` varchar(256) DEFAULT NULL,  
`access_token_validity` int(11) DEFAULT NULL COMMENT '令牌有效期',  
`refresh_token_validity` int(11) DEFAULT NULL COMMENT '令牌刷新周期',  
`additional_information` varchar(4096) DEFAULT NULL,  
`autoapprove` varchar(256) DEFAULT NULL,  
PRIMARY KEY (`client_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

导入1条初始化数据,其中加密字符明文为changgou :

```
INSERT INTO `oauth_client_details` VALUES ('changgou', null,  
'$2a$10$YvKp3xzDcri6MASPIqnzzeGBHez1QZR3A079XdMNU4R725Krkxi2', 'app',  
'authorization_code,password,refresh_token,client_credentials',  
'http://localhost', null, '43200', '43200', null, null);
```

4.2 OAuth2授权模式介绍

OAuth2有以下授权模式：

1. 授权码模式 (Authorization Code)
2. 隐式授权模式 (Implicit)
3. 密码模式 (Resource Owner Password Credentials)
4. 客户端模式 (Client Credentials)

其中授权码模式和密码模式应用较多，本小节介绍授权码模式。

4.2.1 授权码模式

4.2.1.1 授权码授权流程

上边例举的黑马程序员网站使用QQ认证的过程就是授权码模式，流程如下：

- 1、客户端请求第三方授权
- 2、用户同意给客户端授权
- 3、客户端获取到授权码，请求认证服务器申请 令牌
- 4、认证服务器向客户端响应令牌
- 5、客户端请求资源服务器的资源，资源服务校验令牌合法性，完成授权
- 6、资源服务器返回受保护资源

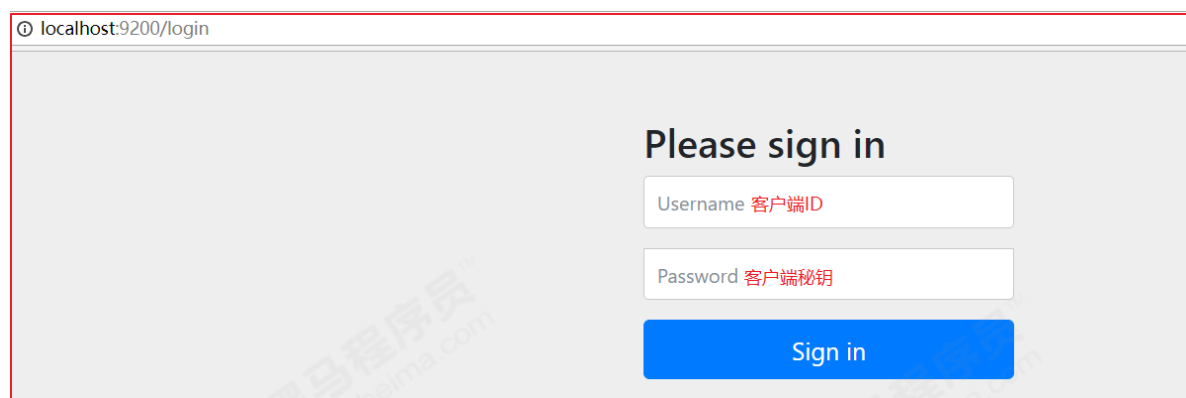
请求认证服务获取授权码：

Get请求：
`http://localhost:9200/oauth/authorize?`
`client_id=changgou&response_type=code&scope=app&redirect_uri=http://localhost`

参数列表如下：

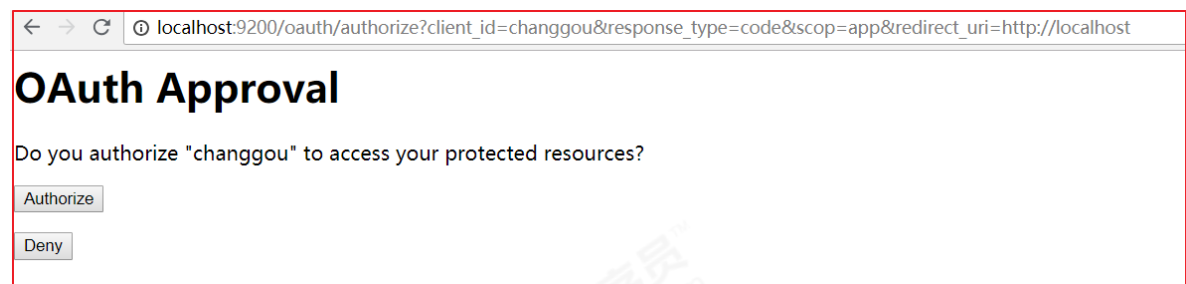
`client_id`: 客户端id, 和授权配置类中设置的客户端id一致。
`response_type`: 授权码模式固定为code
`scope`: 客户端范围, 和授权配置类中设置的scope一致。
`redirect_uri`: 跳转uri, 当授权码申请成功后会跳转到此地址, 并在后边带上code参数(授权码)

首先跳转到登录页面：



输入账号和密码，点击Login。Spring Security接收到请求会调用UserDetailsService接口的loadUserByUsername方法查询用户正确的密码。当前导入的基础工程中客户端ID为changgou，秘钥也为changgou即可认证通过。

接下来进入授权页面：



点击Authorize,接下来返回授权码：认证服务携带授权码跳转redirect_uri,code=k45iLY就是返回的授权码, 每一个授权码只能使用一次



4.2.1.3 申请令牌

拿到授权码后，申请令牌。

Post请求：
`http://localhost:9200/oauth/token`

grant_type: 授权类型，填写**authorization_code**，表示授权码模式

code: 授权码，就是刚刚获取的授权码，注意：授权码只使用一次就无效了，需要重新申请。

redirect_uri: 申请授权码时的跳转url，一定和申请授权码时用的**redirect_uri**一致。

此链接需要使用 http Basic认证。

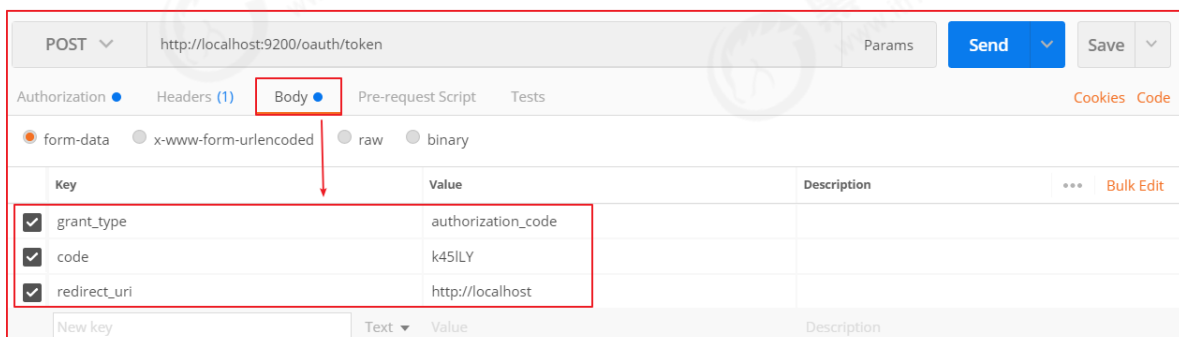
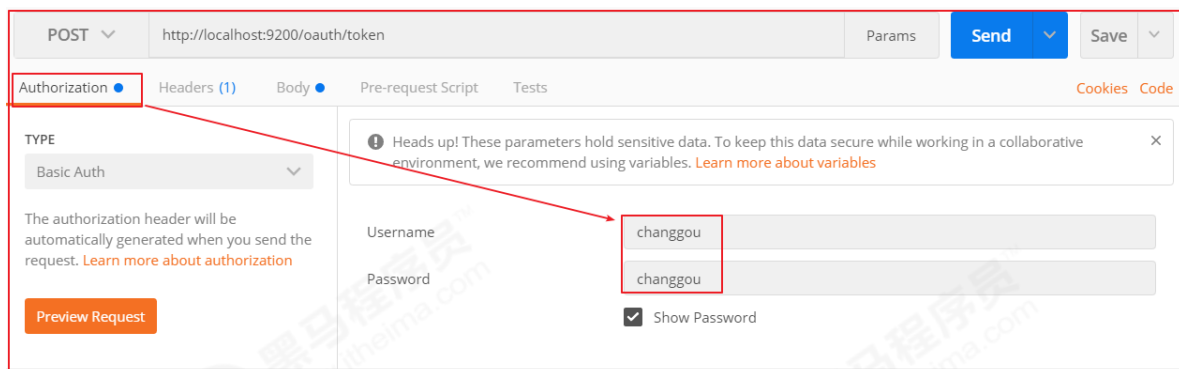
什么是http Basic认证？

http协议定义的一种认证方式，将客户端id和客户端密码按照“客户端ID:客户端密码”的格式拼接，并用base64编码，放在header中请求服务端，一个例子：

Authorization : Basic WGNXZWJBcHA6WGNXZWJBcHA=WGNXZWJBcHA6WGNXZWJBcHA= 是用户名:密码的base64编码。认证失败服务端返回 401 Unauthorized。

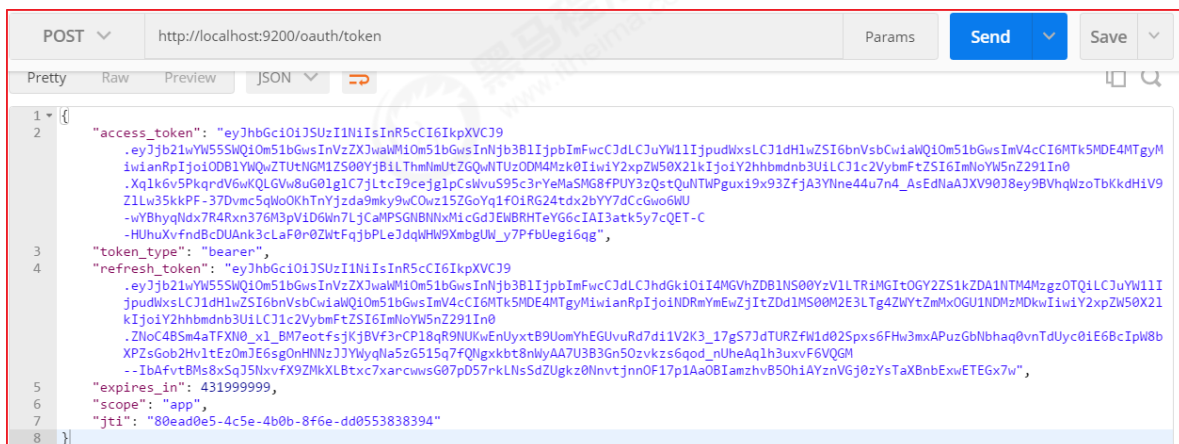
以上测试使用postman完成：

http basic认证：



客户端id和客户端密码会匹配数据库oauth_client_details表中的客户端id及客户端密码。

点击发送：申请令牌成功



返回信如下：

token_type: Bearer Token (Bearer Token的中文意思是“持有者的令牌”，RFC 6750是以OAuth 2.0为基准，使用 Bearer Token (<http://www.rfcreader.com/#rfc6750>)。

refresh_token: 刷新令牌，使用此令牌可以延长访问令牌的过期时间。

expires_in: 过期时间，单位为秒。

scope: 范围，与定义的客户端范围一致。

jti: 当前token的唯一标识

4.2.1.4 令牌校验

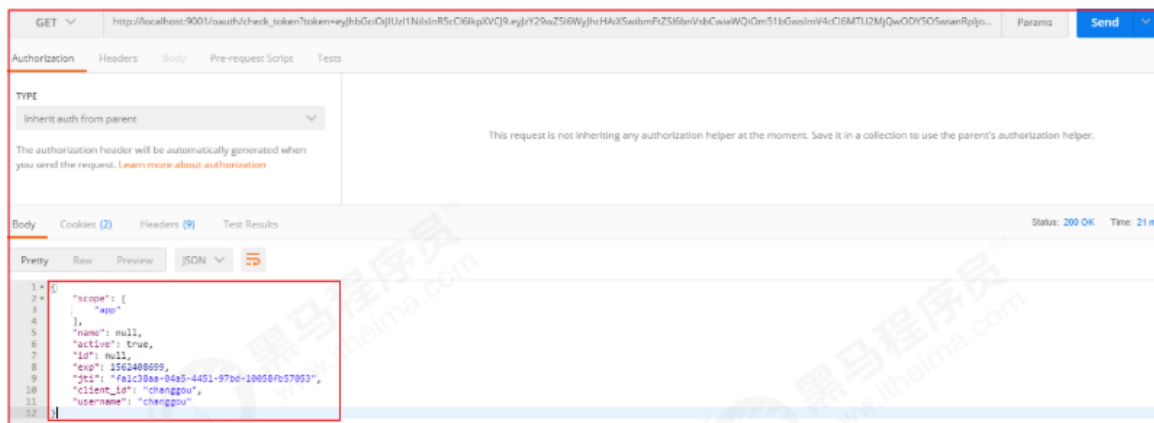
Spring Security OAuth2提供校验令牌的端点，如下：

Get: [http://localhost:9200/oauth/check_token?token=\[access_token\]](http://localhost:9200/oauth/check_token?token=[access_token])

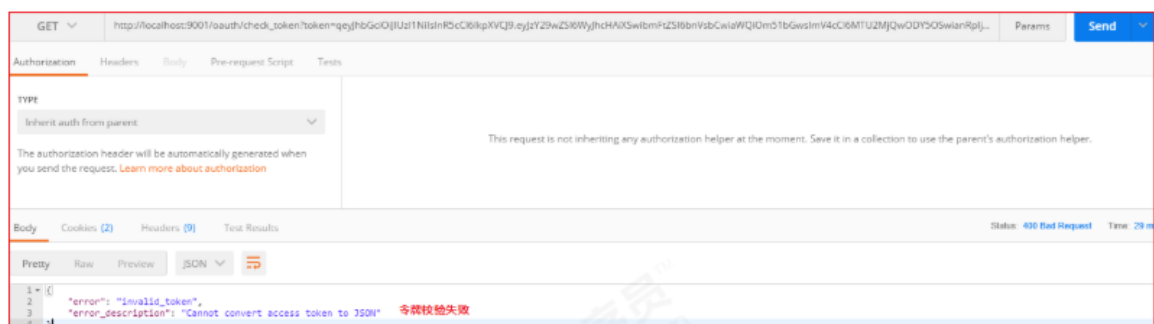
参数：

token：令牌

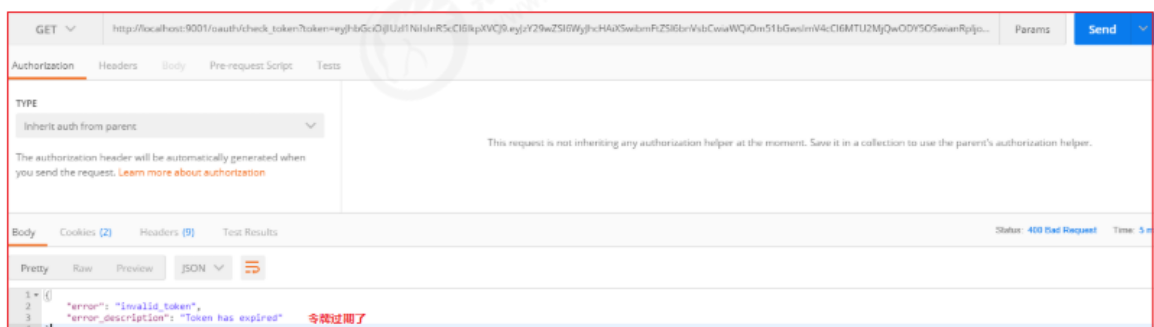
使用postman测试如下：



如果令牌校验失败，会出现如下结果：



如果令牌过期了，会出现如下结果：



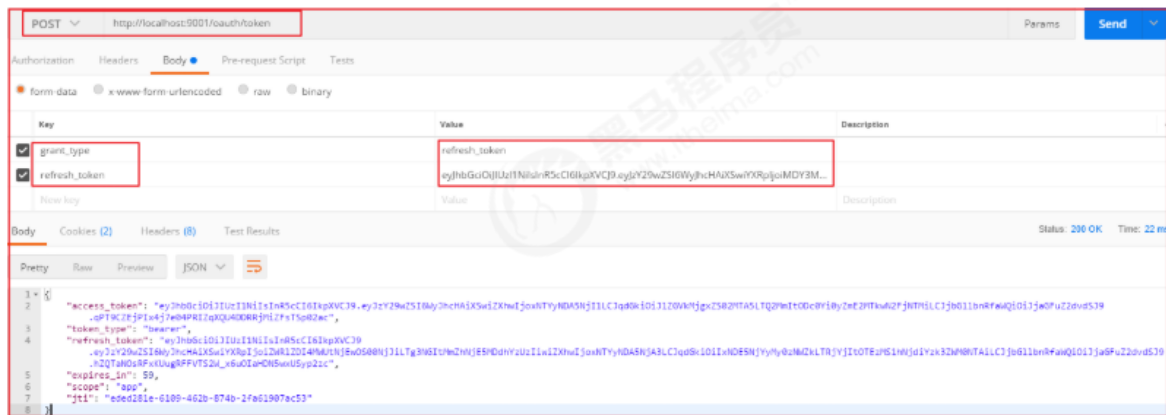
4.2.1.5 刷新令牌

测试如下：Post：<http://localhost:9200/oauth/token>

参数：

grant_type : 固定为 refresh_token

refresh_token：刷新令牌（注意不是access_token，而是refresh_token）



4.2.2 密码模式

密码模式 (Resource Owner Password Credentials) 与授权码模式的区别是申请令牌不再使用授权码, 而是直接通过用户名和密码即可申请令牌。

4.2.2.1 申请令牌

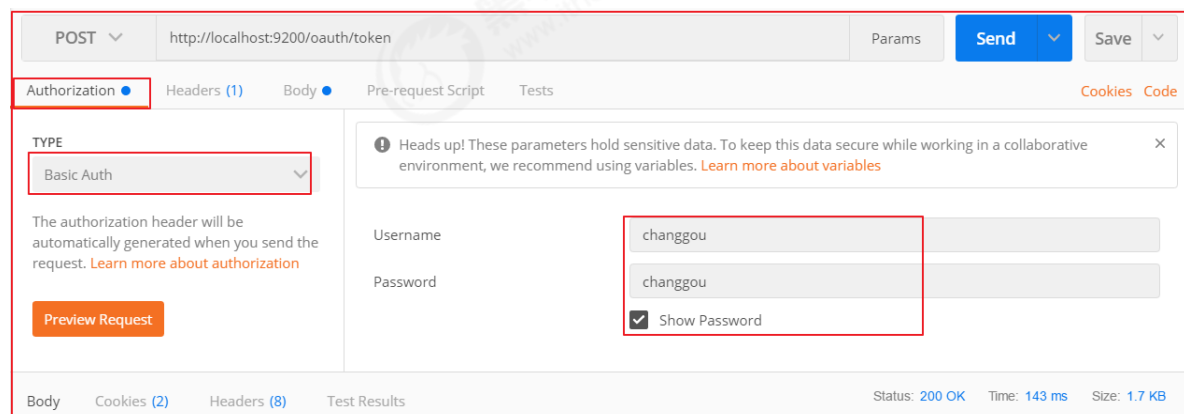
测试如下：

Post请求:
`http://localhost:9200/oauth/token`

携带参数：

- grant_type**: 密码模式授权填写password
- username**: 账号
- password**: 密码

并且此链接需要使用 http Basic认证。



Key	Value	Description
grant_type	password	
username	szitheima	
password	szitheima	

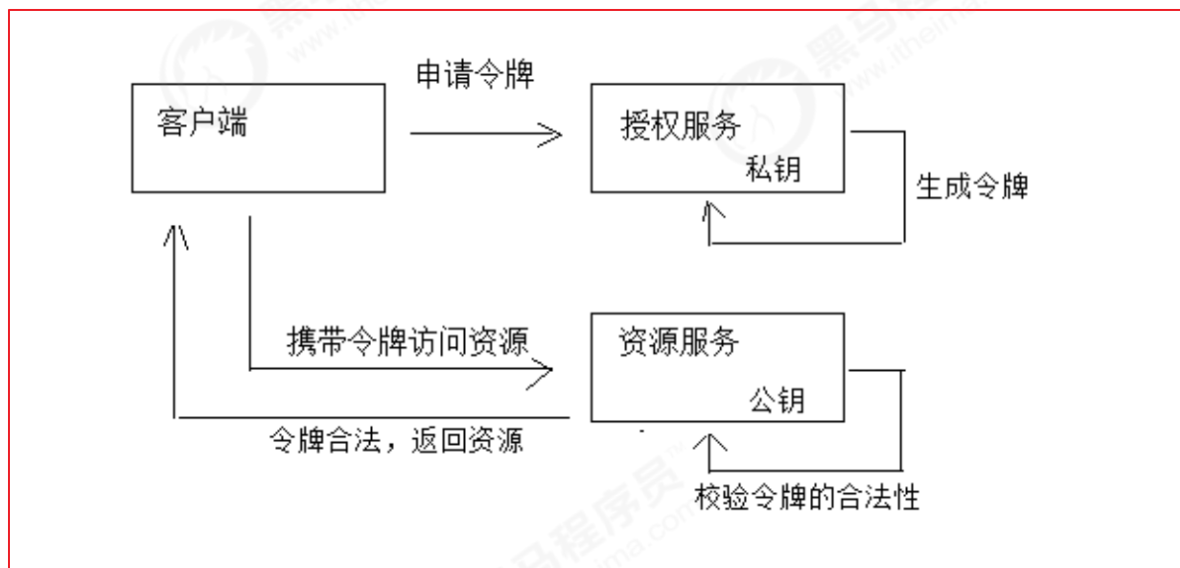
测试数据如下：

```

1 {
2   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
   .eyJjb21wYW55S5wQ10m51bGwsInVzZXJwYW10m51bGwsInNjb3B1IjpbImFwcCJdLCJhdGkiOiI3MmEzMjRkOC1kMjN1LTQ4YzUtYmU0Zi11MmMwMjM4NmExNWE1LCJyZW11Ij
   jpdWxsLCJ1dHlwZSI6bnVsbCw1aWQ10m51bGwsImV4cCI6MTk5MDIyMTM0MjN1LTQ4YzUtYmU0Zi11MmMwMjM4NmExNWE1LCJyZW11Ij
   kIjo1Y2hhbmdnb3U1c2VybWVtZSI6InN6aXRoZW1tYSJ9
   .q8kMdEpo1V18uJ0e1jTY9U50davl1b_3H5pUDyxOY2SmE0IK5W_DQ3rUwhH3H9xsb7qAbP1hewpzDNjsFxsRpmY1I6syyvaEi_ea6bG4bpNekaN
   -iD7yOJKysj9wpnbekHY2QrbRUDRu-ZhAGIphUVZkknBy8d11hAeR7ZCMjokzRzgbgSMmW7v7iv8Y6yN-EVpoclVh9jQQMz-12Nqxs10Tjpd
   -7C1glYbfo_SvOUx111AVVGaiZK5suPrpWP87jo8WsIQ3CK5PwXUkUnapv92jC19p1NW1bz1RRmmsxvHus6tIP_cWymDVef_8JWQbIeia-USOWwKyUdeIQ",
3   "token_type": "bearer",
4   "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
   .eyJjb21wYW55S5wQ10m51bGwsInVzZXJwYW10m51bGwsInNjb3B1IjpbImFwcCJdLCJhdGkiOiI3MmEzMjRkOC1kMjN1LTQ4YzUtYmU0Zi11MmMwMjM4NmExNWE1LCJyZW11Ij
   jpdWxsLCJ1dHlwZSI6bnVsbCw1aWQ10m51bGwsImV4cCI6MTk5MDIyMTM0MjN1LTQ4YzUtYmU0Zi11MmMwMjM4NmExNWE1LCJyZW11Ij
   kIjo1Y2hhbmdnb3U1c2VybWVtZSI6InN6aXRoZW1tYSJ9
   .q8kMdEpo1V18uJ0e1jTY9U50davl1b_3H5pUDyxOY2SmE0IK5W_DQ3rUwhH3H9xsb7qAbP1hewpzDNjsFxsRpmY1I6syyvaEi_ea6bG4bpNekaN
   -iD7yOJKysj9wpnbekHY2QrbRUDRu-ZhAGIphUVZkknBy8d11hAeR7ZCMjokzRzgbgSMmW7v7iv8Y6yN-EVpoclVh9jQQMz-12Nqxs10Tjpd
   -7C1glYbfo_SvOUx111AVVGaiZK5suPrpWP87jo8WsIQ3CK5PwXUkUnapv92jC19p1NW1bz1RRmmsxvHus6tIP_cWymDVef_8JWQbIeia-USOWwKyUdeIQ",
5   "expires_in": 431999999,
6   "scope": "app",
7   "jti": "72a324d8-d23e-48c5-be4f-b1c02386a15a"
8 }
    
```

4.3 资源服务授权

资源服务拥有要访问的受保护资源，客户端携带令牌访问资源服务，如果令牌合法则可成功访问资源服务中的资源，如下图：

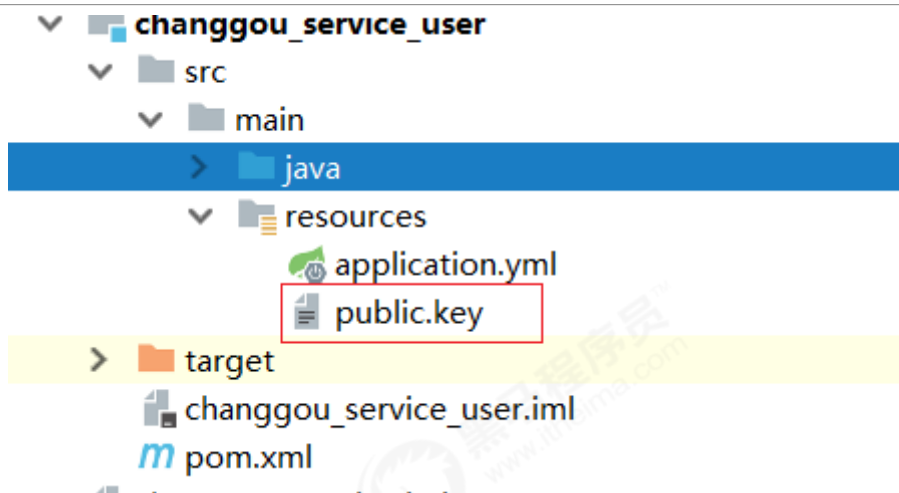


上图的业务流程如下：

- 1、客户端请求认证服务申请令牌
- 2、认证服务生成令牌认证服务采用非对称加密算法，使用私钥生成令牌。
- 3、客户端携带令牌访问资源服务客户端在Http header 中添加：Authorization: Bearer令牌。
- 4、资源服务请求认证服务校验令牌的有效性资源服务接收到令牌，使用公钥校验令牌的合法性。
- 5、令牌有效，资源服务向客户端响应资源信息

4.3.1 用户服务对接Oauth2

基本上所有微服务都是资源服务，这里我们在课程管理服务上配置授权控制，当配置了授权控制后如要访问课程信息则必须提供令牌。



2、添加依赖

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
```

3、配置每个系统的Http请求路径安全控制策略以及读取公钥信息识别令牌，如下：

```
@Configuration
@EnableResourceServer
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)//激活方法上的PreAuthorize注解
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {

    //公钥
    private static final String PUBLIC_KEY = "public.key";

    /**
     * 定义JwtTokenStore
     * @param jwtAccessTokenConverter
     * @return
     */
    @Bean
    public TokenStore tokenStore(JwtAccessTokenConverter
    jwtAccessTokenConverter) {
        return new JwtTokenStore(jwtAccessTokenConverter);
    }

    /**
     * 定义JJwtAccessTokenConverter
     * @return
     */
    @Bean
    public JwtAccessTokenConverter jwtAccessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setVerifierKey(getPubKey());
        return converter;
    }

    /**
     * 获取非对称加密公钥_key
     */
}
```



```
private String getPubKey() {
    Resource resource = new ClassPathResource(PUBLIC_KEY);
    try {
        InputStreamReader inputStreamReader = new
        InputStreamReader(resource.getInputStream());
        BufferedReader br = new BufferedReader(inputStreamReader);
        return br.lines().collect(Collectors.joining("\n"));
    } catch (IOException ioe) {
        return null;
    }
}

/**
 * Http安全配置，对每个到达系统的http请求链接进行校验
 * @param http
 * @throws Exception
 */
@Override
public void configure(HttpSecurity http) throws Exception {
    //所有请求必须认证通过
    http.authorizeRequests()
        //下边的路径放行
        .antMatchers(
            "/user/add"). //配置地址放行
        permitAll()
        .anyRequest().
        authenticated(); //其他地址需要认证授权
}
}
```

4.3.2 资源服务授权测试

不携带令牌访问<http://localhost:9005/user>

由于该地址受访问限制，需要授权，所以出现如下错误：

```
{
  "error": "unauthorized",
  "error_description": "Full authentication is required to access this
resource"
}
```

携带令牌访问<http://localhost:9005/user>

在http header中添加 Authorization : Bearer 令牌

Params Authorization Headers (9) Body Pre-request Script Tests Cookies Code

▼ Headers (1)

KEY	VALUE	DESCRIPTION
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1NjQx...	
Key	Value	Description

► Temporary Headers (8)

Body Cookies (1) Headers (9) Test Results Status: 200 OK Time: 19ms Size: 8.43 KB Save

Pretty Raw Preview JSON

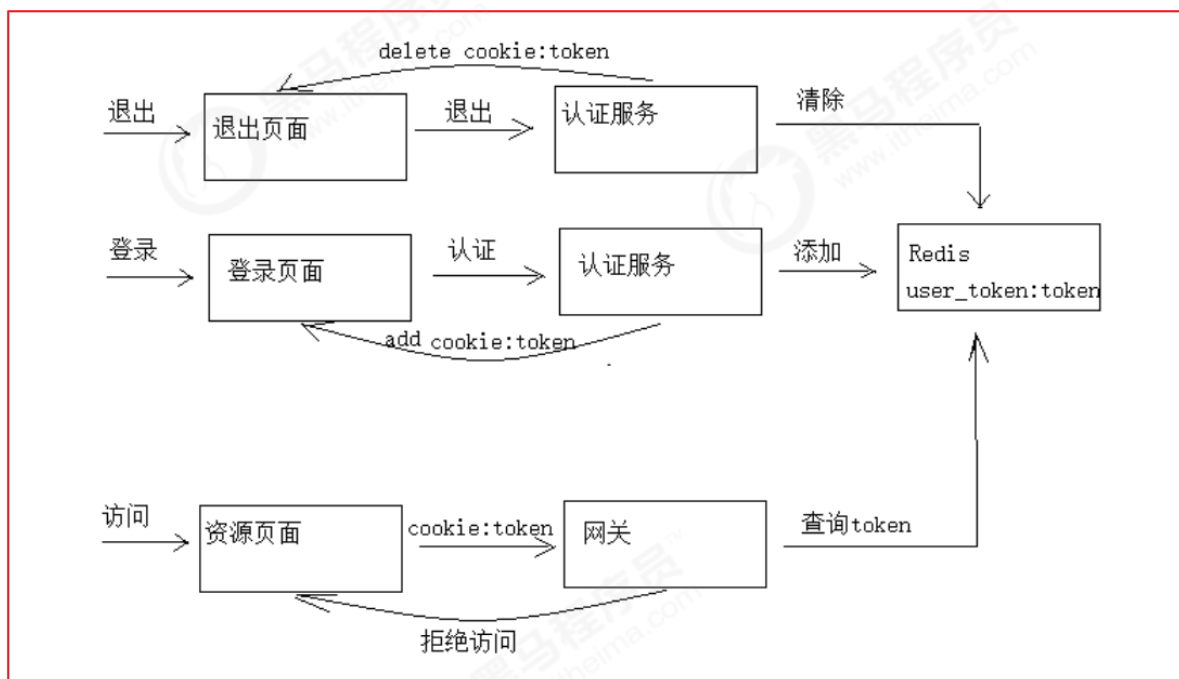
```
1 {
2   "flag": true,
3   "code": 20000,
4   "message": "查询成功",
5   "data": [
6     {
7       "username": "13904211939",
8       "password": "123123",
9       "phone": "13904211939",
10      "email": null,
11      "created": "2019-03-06T15:40:47.000+0000",
12      "updated": "2019-03-06T15:40:47.000+0000",
13      "sourceType": null,
14      "nickname": null,
15      "avatar": null
16    }
17  ]
18 }
```

当输入错误的令牌也无法正常访问资源。

5 认证开发

5.1 需求分析

功能流程图如下：



执行流程：

- 1、用户登录，请求认证服务
- 2、认证服务认证通过，生成jwt令牌，将jwt令牌及相关信息写入Redis，并且将身份令牌写入cookie
- 3、用户访问资源页面，带着cookie到网关
- 4、网关从cookie获取token，并查询Redis校验token，如果token不存在则拒绝访问，否则放行
- 5、用户退出，请求认证服务，清除redis中的token，并且删除cookie中的token

使用redis存储用户的身份令牌有以下作用：

客户端存储。

5.2 Redis配置

将认证服务changgou_user_auth中的application.yml配置文件中的Redis配置改成自己对应的端口和密码。

5.3 认证服务

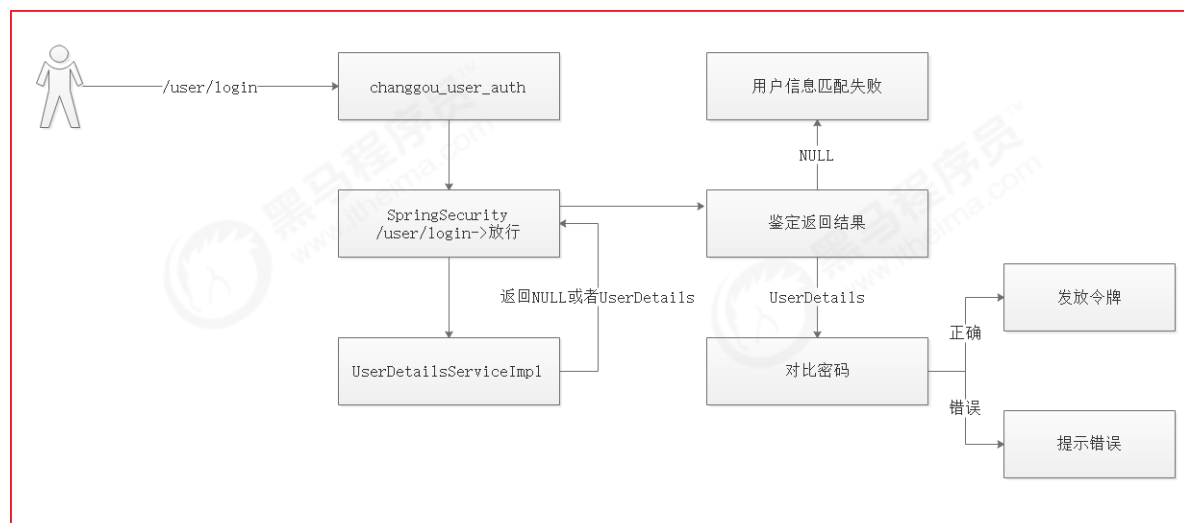
5.3.1 认证需求分析

认证服务需要实现的功能如下：

1、登录接口

前端post提交账号、密码等，用户身份校验通过，生成令牌，并将令牌存储到redis。将令牌写入cookie。

2、退出接口 校验当前用户的身份为合法并且为已登录状态。将令牌从redis删除。删除cookie中的令牌。



5.3.2 授权参数配置

修改changgou_user_auth中application.yml配置文件，修改对应的授权配置

```
auth:
  ttl: 1200 #token存储到redis的过期时间
  clientId: changgou #客户端ID
  clientSecret: changgou #客户端秘钥
  cookieDomain: localhost #Cookie保存对应的域名
  cookieMaxAge: -1 #Cookie过期时间，-1表示浏览器关闭则销毁
```

5.3.3 申请令牌测试

为了不破坏Spring Security的代码，我们在Service方法中通过RestTemplate请求Spring Security所暴露的申请令牌接口来申请令牌，下边是测试代码：



```
public class TokenTest {

    @Autowired
    private LoadBalancerClient loadBalancerClient;

    @Autowired
    private RestTemplate restTemplate;

    /**
     * 发送Http请求创建令牌
     */
    @Test
    public void testCreateToken() throws InterruptedException {
        //采用客户端负载均衡，从eureka获取认证服务的ip 和端口
        ServiceInstance serviceInstance = loadBalancerClient.choose("USER-
AUTH");
        URI uri = serviceInstance.getUri();
        //申请令牌地址
        String authUrl = uri + "/oauth/token";

        //1、header信息，包括了http basic认证信息
        MultivalueMap<String, String> headers = new LinkedMultivalueMap<String,
String>();
        //进行Base64编码，并将编码后的认证数据放到头文件中
        String httpbasic = httpbasic("changgou", "changgou");
        headers.add("Authorization", httpbasic);
        //2、指定认证类型、账号、密码
        MultivalueMap<String, String> body = new LinkedMultivalueMap<String,
String>();
        body.add("grant_type", "password");
        body.add("username", "itheima");
        body.add("password", "123456");
        HttpEntity<MultivalueMap<String, String>> multivalueMapHttpEntity = new
HttpEntity<MultivalueMap<String, String>>(body, headers);
        //指定 restTemplate当遇到400或401响应时候也不要抛出异常，也要正常返回
        restTemplate.setErrorHandler(new DefaultResponseErrorHandler() {
            @Override
            public void handleError(ClientHttpResponse response) throws
IOException {
                //当响应的值为400或401时候也要正常响应，不要抛出异常
                if (response.getRawStatusCode() != 400 &&
response.getRawStatusCode() != 401) {
                    super.handleError(response);
                }
            }
        });

        //远程调用申请令牌
        ResponseEntity<Map> exchange = restTemplate.exchange(authUrl,
HttpMethod.POST, multivalueMapHttpEntity, Map.class);
        Map result = exchange.getBody();
        System.out.println(result);
    }

    /**

```



```
    * @param clientSecret
    * @return
    */
    private String httpbasic(String clientId,String clientSecret){
        //将客户端id和客户端密码拼接，按“客户端id:客户端密码”
        String string = clientId+": "+clientSecret;
        //进行base64编码
        byte[] encode = Base64Utils.encode(string.getBytes());
        return "Basic "+new String(encode);
    }
}
```

5.3.4 业务层

AuthService接口：

```
public interface AuthService {
    AuthToken login(String username, String password, String clientId, String
    clientSecret);
}
```

AuthServiceImpl实现类：

基于刚才写的测试实现申请令牌的service方法如下：

```
@Service
public class AuthServiceImpl implements AuthService {

    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private LoadBalancerClient loadBalancerClient;

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    @Value("${auth.ttl}")
    private long ttl;

    /**
     * 申请令牌
     * @param username
     * @param password
     * @param clientId
     * @param clientSecret
     * @return
     */
    @Override
    public AuthToken applyToken(String username, String password, String
    clientId, String clientSecret) {
```



```
URI uri = serviceInstance.getUri();
String url = uri+"/oauth/token";

Multimap<String, String> body = new LinkedMultimap<>();
body.add("grant_type", "password");
body.add("username", username);
body.add("password", password);

Multimap<String, String> headers = new LinkedMultimap<>();
headers.add("Authorization", this.getHttpBasic(clientId, clientSecret));

HttpEntity<Multimap<String, String>> requestEntity = new
HttpEntity<>(body, headers);

restTemplate.setErrorHandler(new DefaultResponseErrorHandler(){
    @Override
    public void handleError(ClientHttpResponse response) throws
IOException {
        if (response.getRawStatusCode() != 400 &&
response.getRawStatusCode() != 401){
            super.handleError(response);
        }
    }
});
ResponseEntity<Map> responseEntity = restTemplate.exchange(url,
HttpMethod.POST, requestEntity, Map.class);

Map map = responseEntity.getBody();

if (map == null || map.get("access_token") == null ||
map.get("refresh_token") == null || map.get("jti") == null){
    throw new RuntimeException("申请令牌失败");
}

AuthToken authToken = new AuthToken();
authToken.setAccessToken((String) map.get("access_token"));
authToken.setRefreshToken((String) map.get("refresh_token"));
authToken.setJti((String) map.get("jti"));

stringRedisTemplate.boundValueOps(authToken.getJti()).set(authToken.getAccessTo
ken(), ttl, TimeUnit.SECONDS);

return authToken;
}

private String getHttpBasic(String clientId, String clientSecret) {

    String value = clientId+":"+clientSecret;
    byte[] encode = Base64Utils.encode(value.getBytes());
    return "Basic "+new String(encode);
}
}
```



5.3.5 控制层

AuthController编写用户登录授权方法，代码如下：

```
@RestController
@RequestMapping("/oauth")
public class AuthController {

    @Autowired
    private AuthService authService;

    @Value("${auth.clientId}")
    private String clientId;

    @Value("${auth.clientSecret}")
    private String clientSecret;

    @Value("${auth.cookieDomain}")
    private String cookieDomain;

    @Value("${auth.cookieMaxAge}")
    private int cookieMaxAge;

    @PostMapping("/login")
    public Result login(String username, String password) {

        if (StringUtils.isEmpty(username)) {
            throw new RuntimeException("用户名不存在");
        }
        if (StringUtils.isEmpty(password)) {
            throw new RuntimeException("密码不存在");
        }

        AuthToken authToken =
            authService.applyToken(username, password, clientId, clientSecret);

        this.saveJtiToCookie(authToken.getJti());

        return new Result(true, StatusCode.OK, "登录成功");
    }

    private void saveJtiToCookie(String jti) {
        HttpServletResponse response = ((ServletRequestAttributes)
            RequestContextHolder.getRequestAttributes()).getResponse();

        CookieUtil.addCookie(response, cookieDomain, "/", "uid", jti, cookieMaxAge, false);
    }
}
```

5.3.6 登录请求放行

修改认证服务WebSecurityConfig类中configure()，添加放行路径

北京市昌平区建材城西路金燕龙办公楼一层 电话：400-618-9090

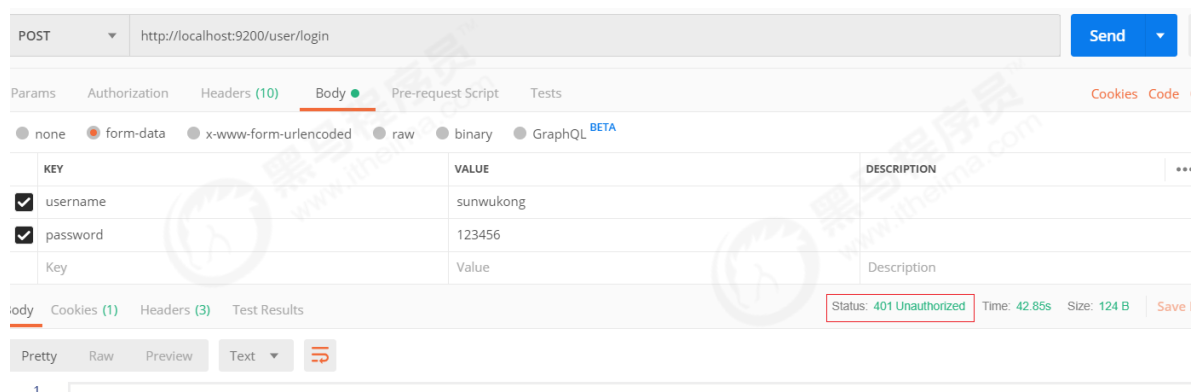
```
@Order(-1)
class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /**
     * 忽略安全拦截的URL
     * @param web
     * @throws Exception
     */
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers(
            ...antPatterns: "/oauth/login", ←
            "/oauth/logout");
    }
}
```

5.3.7 测试认证接口

使用postman测试：

1)Post请求：<http://localhost:9200/oauth/login>



5.3.8 动态获取用户信息

当前在认证服务中，用户密码是写死在用户认证类中。所以用户登录时，无论帐号输入什么，只要密码是itheima都可以访问。因此需要动态获取用户帐号与密码。

5.3.8.1 定义被访问接口

用户微服务对外暴露根据用户名获取用户信息接口

```
@GetMapping("/load/{username}")
public User findUserInfo(@PathVariable("username") String username){
    return userService.findById(username);
}
```

5.3.8.2 放行该接口，修改ResourceServerConfig类



```
// 为所有请求认证通过
http.authorizeRequests()
    // 下边的路径放行
    .antMatchers(
        ...antPatterns: "/user/add", "/user/load/**") // 配置地址放行
    permitAll()
    .anyRequest().
    authenticated(); // 其他地址需要认证授权
}
```

5.3.8.3 定义feign接口

changgou_user_server_api新增feign接口

```
@FeignClient(name="user")
public interface UserFeign {

    @GetMapping("/user/load/{username}")
    public User findUserInfo(@PathVariable("username") String username);
}
```

5.3.8.4 认证服务添加依赖

```
<dependency>
    <groupId>com.changgou</groupId>
    <artifactId>changgou_service_user_api</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

5.3.8.5 修改认证服务启动类

```
@EnableFeignClients(basePackages = "com.changgou.user.feign")
```

5.3.8.6 修改用户认证类

```
@Autowired
private UserFeign userFeign;

if (StringUtils.isEmpty(username)) {
    return null;
}

// 根据用户名查询用户信息
//String pwd = new BCryptPasswordEncoder().encode("itheima");
com.changgou.user.pojo.User userInfo = userFeign.findUserInfo(username);
// 创建User对象
String permissions = "goods_list,seckill_list";
UserJwt userDetails = new UserJwt(username, userInfo.getPassword(), AuthorityUtils.commaSeparatedStringToAuthorityList(permissions));
return userDetails;
}
```

测试：重新启动服务并申请令牌

6 认证服务对接网关

6.1 新建网关工程changgou_gateway_web

1) changgou_gateway添加依赖



```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<!--redis-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
  <version>2.1.3.RELEASE</version>
</dependency>
</dependencies>
```

2) 新建工程changgou_gateway_web,并创建启动类

```
@SpringBootApplication
@EnableDiscoveryClient
public class WebGatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(WebGatewayApplication.class, args);
    }
}
```

3) 创建application.yml

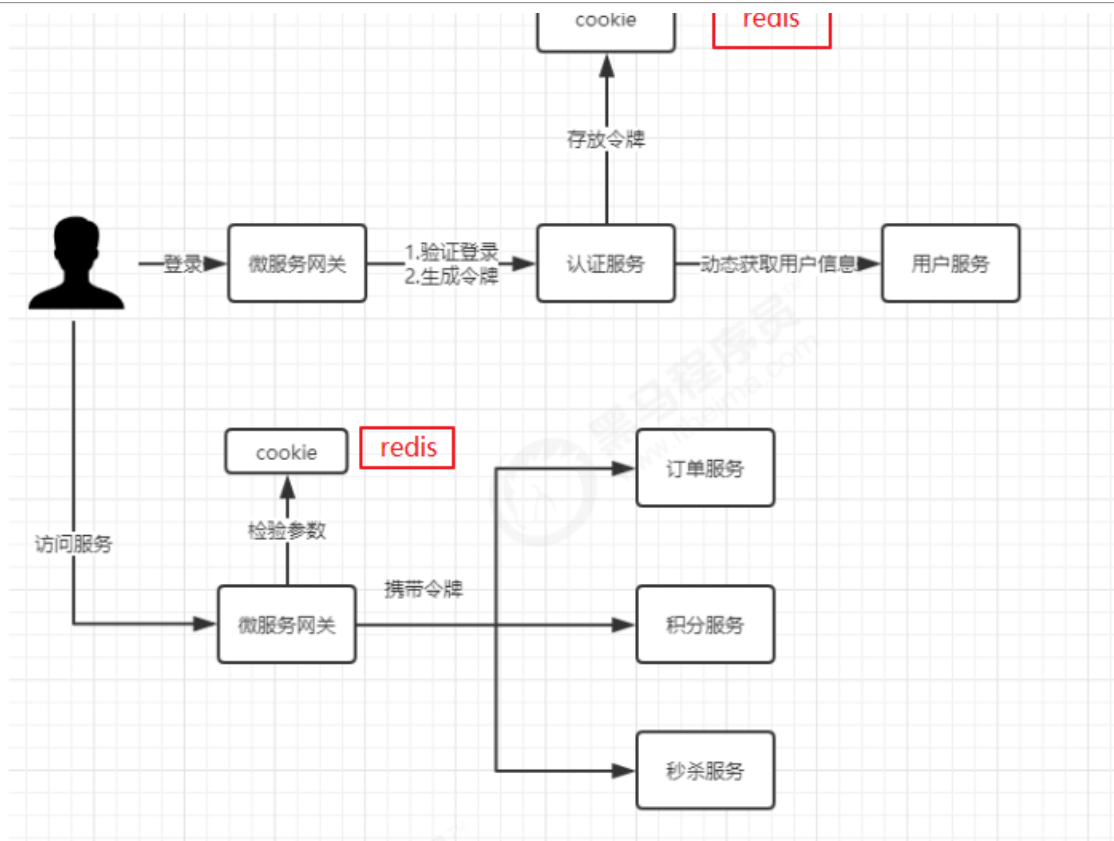
```
spring:
  application:
    name: gateway-web
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]': # 匹配所有请求
            allowedOrigins: "*" #跨域处理 允许所有的域
            allowedMethods: # 支持的方法
              - GET
              - POST
              - PUT
              - DELETE
      routes:
        - id: changgou_goods_route
          uri: lb://goods
          predicates:
```



```
ry/**,/api/para/**,/api/pref/**,/api/sku/**,/api/spec/**,/api/spu/**,/api/stockB
ack/**,/api/template/**
  filters:
    #- PrefixPath=/brand
    - StripPrefix=1
    #用户微服务
  - id: changgou_user_route
    uri: lb://user
    predicates:
      -
Path=/api/user/**,/api/address/**,/api/areas/**,/api/cities/**,/api/provinces/**
  filters:
    - StripPrefix=1
    #认证微服务
  - id: changgou_oauth_user
    uri: lb://user-auth
    predicates:
      - Path=/api/oauth/**
  filters:
    - StripPrefix=1

redis:
  host: 192.168.200.128
server:
  port: 8001
eureka:
  client:
    service-url:
      defaultZone: http://127.0.0.1:6868/eureka
  instance:
    prefer-ip-address: true
management:
  endpoint:
    gateway:
      enabled: true
  web:
    exposure:
      include: true
```

6.2 网关全局过滤器



新建过滤器类AuthorizeFilter,对请求进行过滤

业务逻辑：

- 1) 判断当前请求是否为登录请求，是的话，则放行
- 2) 判断cookie中是否存在信息，没有的话，拒绝访问
- 3) 判断redis中令牌是否存在，没有的话，拒绝访问

```
@Component
public class AuthFilter implements GlobalFilter, Ordered {

    public static final String Authorization = "Authorization";

    @Autowired
    private AuthService authService;

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {

        //获取当前请求对象
        ServerHttpRequest request = exchange.getRequest();
        ServerHttpResponse response = exchange.getResponse();

        String path = request.getURI().getPath();
        if ("/api/oauth/login".equals(path)){
            //放行
            return chain.filter(exchange);
        }
    }
}
```



```
        if (StringUtils.isEmpty(jti)){
            //拒绝访问,请求跳转
            response.setStatus(HttpStatus.UNAUTHORIZED);
            return response.setComplete();
        }

        //判断redis中token是否存在
        String redisToken = authService.getTokenFromRedis(jti);
        if (StringUtils.isEmpty(redisToken)){
            //拒绝访问,请求跳转
            response.setStatus(HttpStatus.UNAUTHORIZED);
            return response.setComplete();
        }

        //校验通过,请求头增强,放行
        request.mutate().header(Authorization, "Bearer "+redisToken);
        return chain.filter(exchange);
    }

    @Override
    public int getOrder() {
        return 0;
    }
}
```

新建业务逻辑类AuthService

```
@Service
public class AuthService {

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    /**
     * 判断cookie中jti是否存在
     * @param request
     * @return
     */
    public String getJtiFromCookie(ServerHttpRequest request) {
        HttpCookie cookie = request.getCookies().getFirst("uid");
        if (cookie!=null){
            return cookie.getValue();
        }
        return null;
    }

    /**
     * 判断redis中令牌是否过期
     * @param jti
     * @return
     */
    public String getTokenFromRedis(String jti) {
```

```
}  
}
```

测试:

清除postman中的cookie数据，在未登录的情况下，并访问：<http://localhost:8001/api/user>。会返回401异常信息

访问：<http://localhost:8001/api/oauth/login>，并重新测试。可以发现测试通过，拿到返回结果数据

7 自定义登录页面

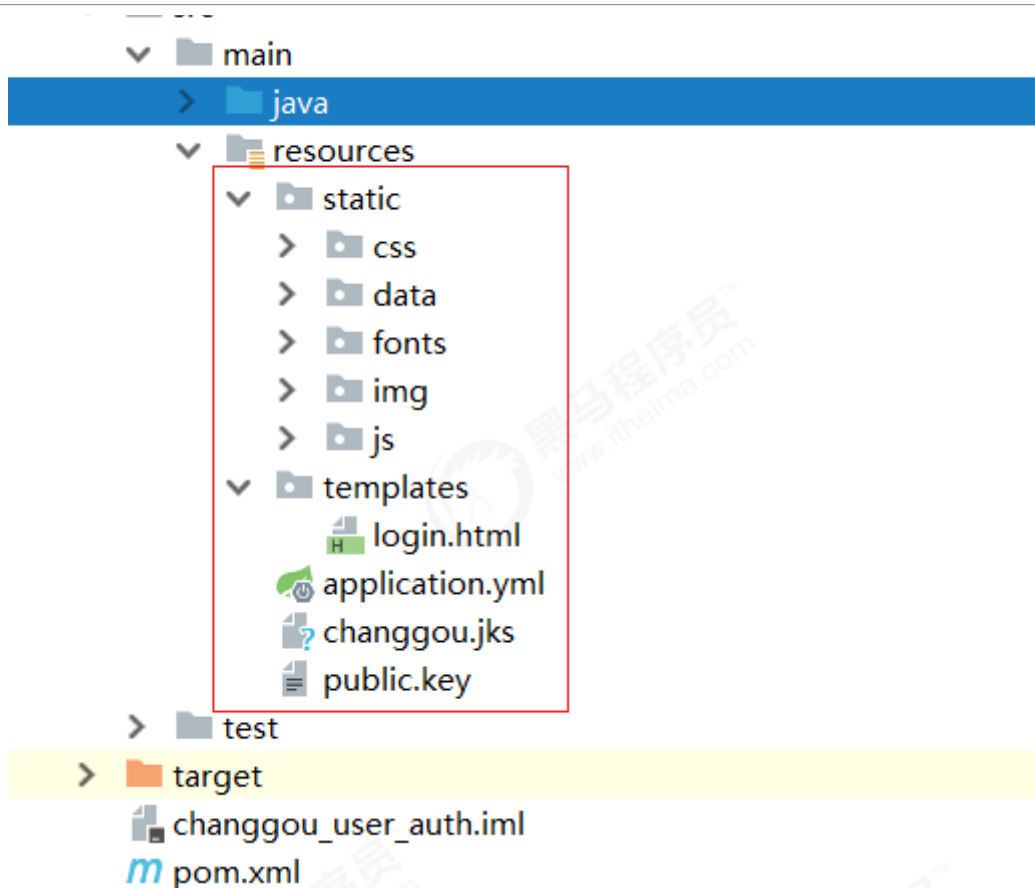
7.1 认证服务添加依赖

```
<!--thymeleaf-->  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>
```

7.2 资源\成品页面\登录页面

名称	修改日期	类型	大小
css	2019/7/12 18:12	文件夹	
data	2019/7/12 18:12	文件夹	
fonts	2019/7/12 18:12	文件夹	
img	2019/7/12 18:12	文件夹	
js	2019/7/12 18:12	文件夹	
login.html	2019/7/12 14:22	Chrome HTML D...	6 KB

7.3 把页面放到下面的项目中



7.4 静态资源放行

修改WebSecurityConfig类

```
@Configuration
@EnableWebSecurity
@Order(-1)
class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /**
     * 忽略安全拦截的URL
     * @param web
     * @throws Exception
     */
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers(
            ...antPatterns: "/oauth/login",
            "/oauth/logout", "/oauth/toLogin", "/login.html", "/css/**", "/data/**", "/fonts/**", "/img/**", "/js/**");
    }
}
```

```
web.ignoring().antMatchers(
    "/oauth/login",

    "/oauth/logout", "/oauth/toLogin", "/login.html", "/css/**", "/data/**", "/fonts/**",
    "/img/**", "/js/**");
```

添加开启表单登录


```

        .httpBasic() //启用Http基本身份验证
        .and()
        .formLogin() //启用表单身份验证
        .and()
        .authorizeRequests() //限制基于Request请求访问
        .anyRequest()
        .authenticated(); //其他请求都需要经过验证
http.formLogin().loginPage("/oauth/toLogin");//配置springSecurity要跳转登录页面的访问路径
        .loginProcessingUrl("/oauth/login"); //配置springSecurity执行登录操作的访问路径
    }
}

```

7.5 LoginRedirectController

```

@Controller
@RequestMapping("/oauth")
public class LoginRedirectController {

    @RequestMapping("/toLogin")
    public String toLogin(){
        return "login";
    }
}

```

7.6 修改登录页面实现用户登录

```

<div id="profile" class="tab-pane active">
    <form class="sui-form">
        <div class="input-prepend"><span class="add-on loginname"></span>
            <input id="inputName" v-model="username" type="text" placeholder="邮箱/用户名/手机号" class="span2 input-xfat">
        </div>
        <div class="input-prepend"><span class="add-on loginpwd"></span>
            <input id="inputPassword" type="password" v-model="password" placeholder="请输入密码" class="span2 input-xfat">
        </div>
        <div class="setting">
            <label class="checkbox inline">
                <input name="m1" type="checkbox" value="2" checked="">
                自动登录
            </label>
            <span class="forget">忘记密码? </span>
        </div>
        <div class="logged">
            <div style="text-align: center; color: red">{{msg}}</div>
            <!--<a class="sui-btn btn-block btn-xlarge btn-danger" href="home.html" target=" blank">登&nbsp;&nbsp;&nbsp;录</a-->
            <button class="sui-btn btn-block btn-xlarge btn-danger" @click="login()" type="button">登 录</button>
        </div>
    </form>

```

7.7 定义login方法

```

<script th:inline="javascript">
    var app = new Vue({
        el:"#app",
        data:{
            username:"",
            password:"",
            msg:""
        },
        methods:{
            login:function(){
                app.msg="正在登录";
                axios.post("/api/oauth/login?
username="+app.username+"&password="+app.password).then(function (response) {
                    if (response.data.flag){
                        app.msg="登录成功";

```

```
    }  
    })  
}  
})  
</script>
```

定义路径过滤

```
public class URLFilter {  
  
    public static String filterPath =  
        "/api/wseckillorder,/api/seckill,/api/wxpay,/api/wxpay/**,/api/worder/**,/api/us  
er/**,/api/address/**,/api/wcart/**,/api/cart/**,/api/categoryReport/**,/api/ord  
erConfig/**,/api/order/**,/api/orderItem/**,/api/orderLog/**,/api/preferential/**  
*,/api/returnCause/**,/api/returnOrder/**,/api/returnOrderItem/**";  
  
    public static boolean hasAuthorize(String url){  
  
        String[] split = filterPath.replace("/**", "").split(",");  
  
        for (String value : split) {  
  
            if (url.startsWith(value)){  
                return true;  
            }  
        }  
  
        return false;  
    }  
}
```

7.8 测试

访问：<http://localhost:9200/oauth/toLogin>

