

# 第9章 Spring Security OAuth2 JWT

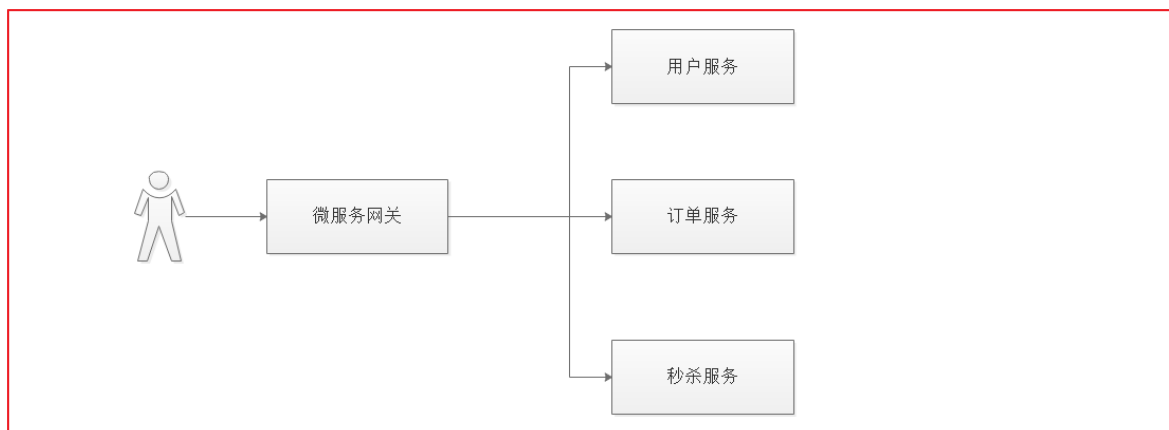
## 学习目标

- 用户认证分析
- 认证技术方案了解（单点登录+第三方授权认证）
- SpringSecurity OAuth2.0入门

- 1 | oauth2.0认证模式
- 2 | 授权码授权模式
- 3 | 密码授权模式
- 4 | 授权流程

- 用户授权认证开发

## 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 认证与授权

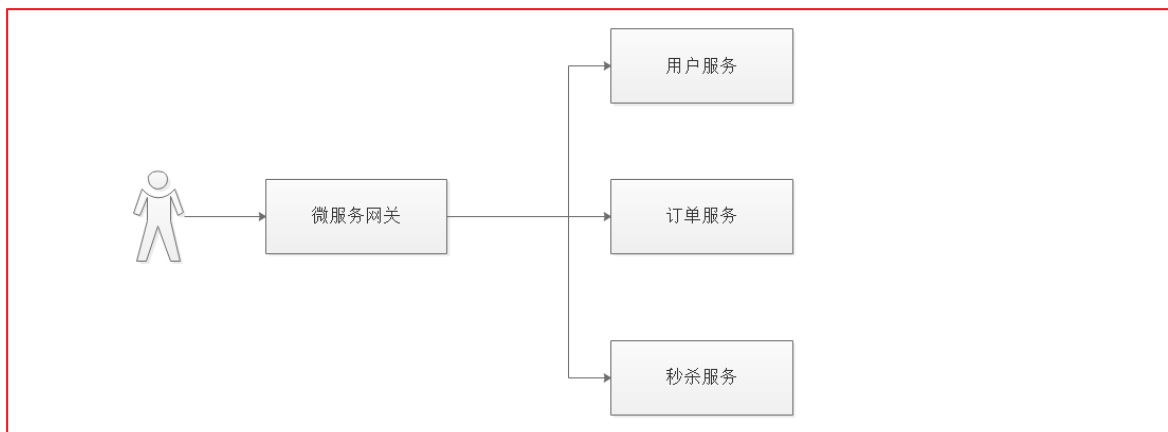
#### 身份认证

用户身份认证即用户去访问系统资源时系统要求验证用户的身份信息，身份合法方可继续访问。常见的用户身份认证表现形式有：用户名密码登录，指纹打卡等方式。说通俗点，就相当于校验用户账号密码是否正确。

#### 用户授权

用户认证通过后去访问系统的资源，系统会判断用户是否拥有访问资源的权限，只允许访问有权限的系统资源，没有权限的资源将无法访问，这个过程叫用户授权。

## 1.2 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

## 1.3 第三方账号登录

### 1.3.1 第三方登录介绍

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。

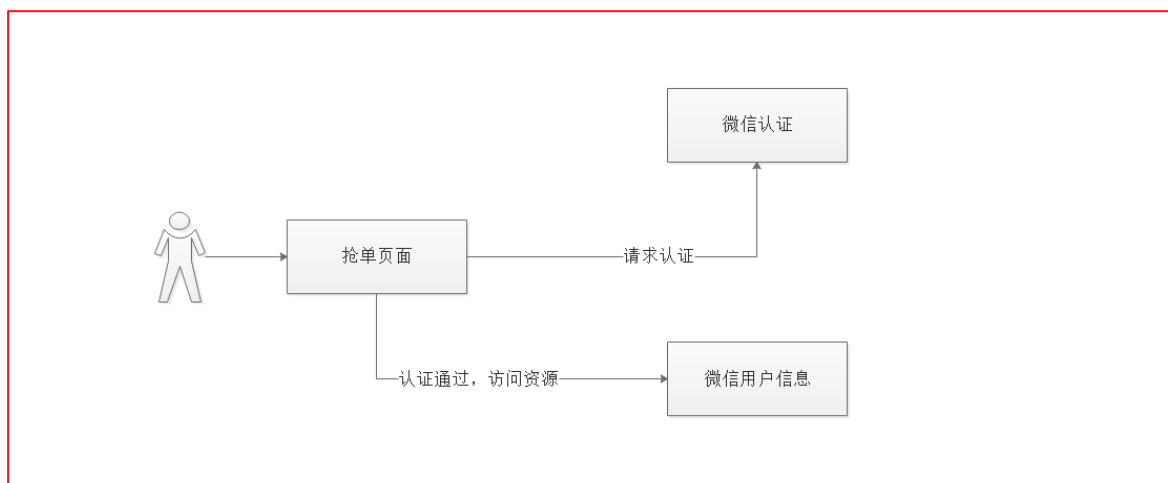


### 1.3.2 第三方登录优点

- 1 1. 相比于本地注册，第三方登录一般来说比较方便、快捷，能够显著降低用户的注册和登录成本，方便用户实现快捷登录或注册。
- 2 2. 不用费尽心思地应付本地注册对账户名和密码的各种限制，如果不考虑昵称的重复性要求，几乎可以直接一个账号走遍天下，再也不用在大脑或者什么地方记住N多不同的网站或App的账号和密码，整个世界一下子清静了。
- 3 3. 在第一次绑定成功之后，之后用户便可以实现一键登录，使得后续的登录操作比起应用内的登录来容易了很多。
- 4 4. 对于某些喜欢社交，并希望将更多自己的生活内容展示给朋友的人来说，第三方登录可以实现把用户在应用内的活动同步到第三方平台上，省去了用户手动发布动态的麻烦。但对于某些比较注重个人隐私的用户来说，则会有一些担忧，所以龙哥所说的这个优点是有所前提的。
- 5 5. 因为降低了用户的注册或登录成本，从而减少由于本地注册的繁琐性而带来的隐形用户流失，最终提高注册转化率。
- 6 6. 对于某些应用来说，使用第三方登录完全可以满足自己的需要，因此不必要设计和开发一套自己的账户体系。
- 7 7. 通过授权，可以通过在第三方平台上分享用户在应用内的活动在第三方平台上宣传自己，从而增加产品知名度。
- 8 8. 通过授权，可以获得该用户在第三方平台上的好友或粉丝等社交信息，从而后续可以针对用户的社交关系网进行有目的的营销宣传，为产品的市场推广提供另一种渠道。

### 1.3.3 第三方认证

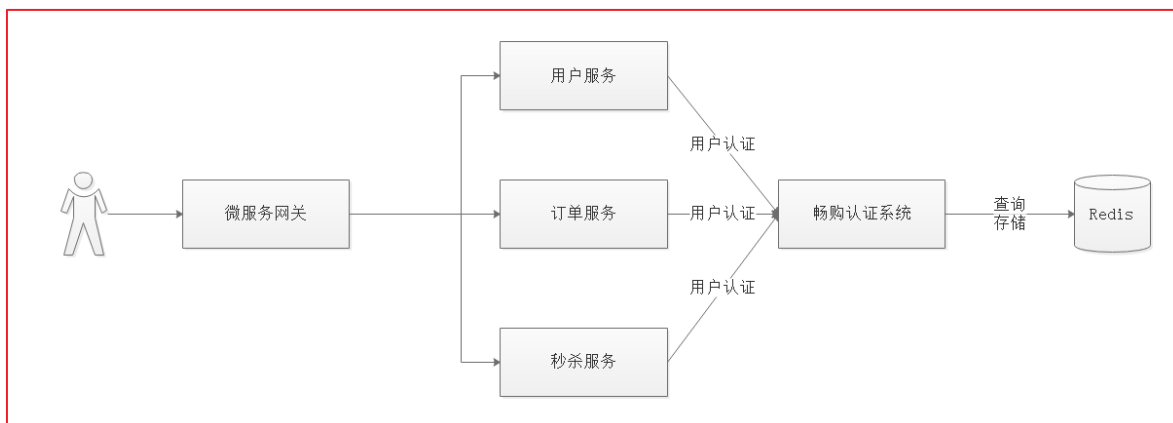
当需要访问第三方系统的资源时需要首先通过第三方系统的认证（例如：微信认证），由第三方系统对用户认证通过，并授权资源的访问权限。



## 2 认证技术方案

### 2.1 单点登录技术方案

分布式系统要实现单点登录，通常将认证系统独立抽取出来，并且将用户身份信息存储在单独的存储介质，比如：MySQL、Redis，考虑性能要求，通常存储在Redis中，如下图：



单点登录的特点是：

- 1 1、认证系统为独立的系统。
- 2 2、各子系统通过Http或其它协议与认证系统通信，完成用户认证。
- 3 3、用户身份信息存储在Redis集群。

Java中有很多用户认证的框架都可以实现单点登录：

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

## 2.2 OAuth2认证

OAuth（开放授权）是一个开放标准，允许用户授权第三方移动应用访问他们存储在另外的服务提供者上的信息，而不需要将用户名和密码提供给第三方移动应用或分享他们数据的所有内容，OAuth2.0是OAuth协议的延续版本。

理解：

- 1 OAuth2.0属于一个开放的授权认证标准，允许用户授权第三方移动应用(例如：黑马官网)访问自己(例如：微信信息)服务器上的资源，用户无需向第三方应用提供账号或者密码。

### 2.2.1 OAuth2认证流程(授权码模式的流程)

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(授权码授权)：



### 1.客户端请求第三方授权

用户进入黑马程序的登录页面，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帐号登录”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。



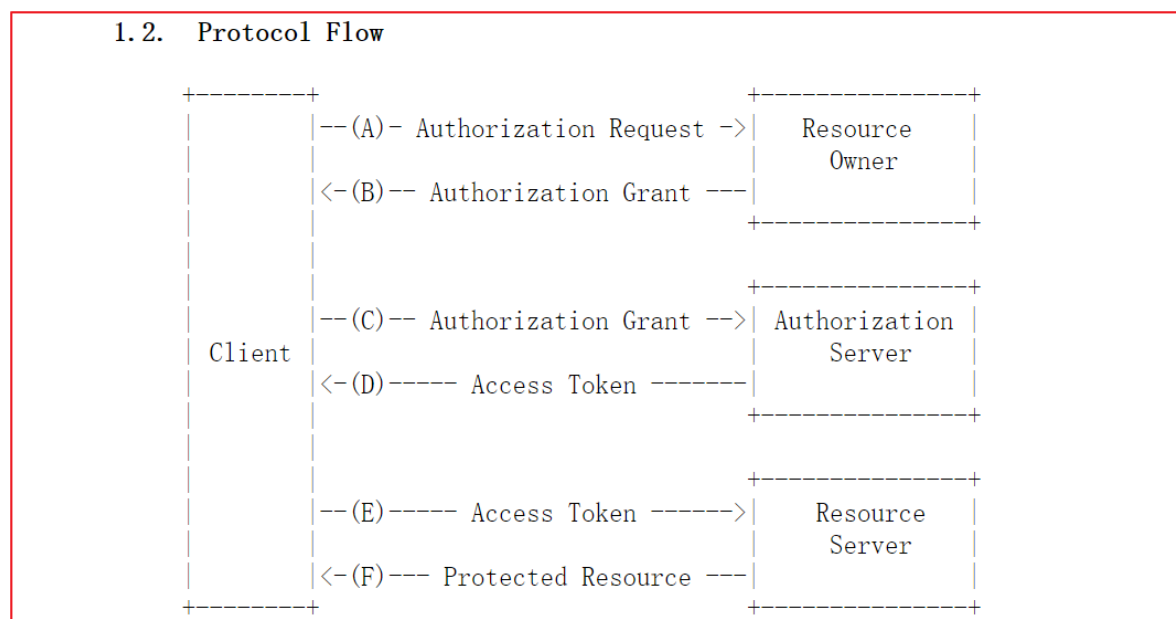
3.客户端获取到授权码，请求认证服务器申请令牌 此过程用户看不到，客户端应用程序请求认证服务器，请求携带授权码。

4.认证服务器向客户端响应令牌 认证服务器验证了客户端请求的授权码，如果合法则给客户端颁发令牌，令牌是客户端访问资源的通行证。此交互过程用户看不到，当客户端拿到令牌后，用户在黑马程序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：畅购在线 Android客户端、畅购在 线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，畅购网用户管理服务器存储了畅购网的用户信息等。客户端最终访问资源服务器获取资源信息。

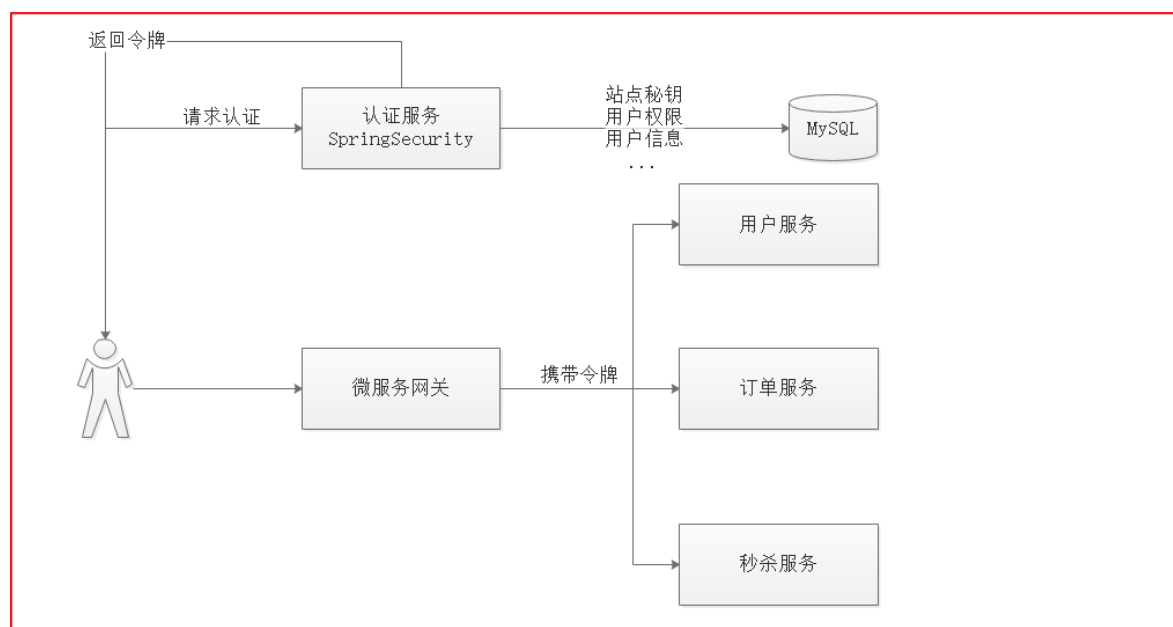
### 2.2.2 OAuth2在项目的应用

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，本项目使用 OAuth2实现如 下目标：

- 1、畅购访问第三方系统的资源
- 2、外部系统访问畅购的资源
- 3、畅购前端（客户端） 访问畅购微服务的资源。
- 4、畅购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。
- 5 为什么要用oauth2.0:1.方便 2.不暴露任何的用户信息

## 2.3 Spring security OAuth2认证解决方案

本项目采用 Spring security + OAuth2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 SpringSecurity OAuth2.0入门

### 3.1 学习知识点说明

本项目认证服务基于Spring Security OAuth2进行构建，并在其基础上作了一些扩展，采用JWT令牌机制，并自定义了用户身份信息的内容。本教程的主要目标是学习在项目中集成Spring Security OAuth2的方法和流程，通过 spring Security OAuth2的研究需要达到以下目标：

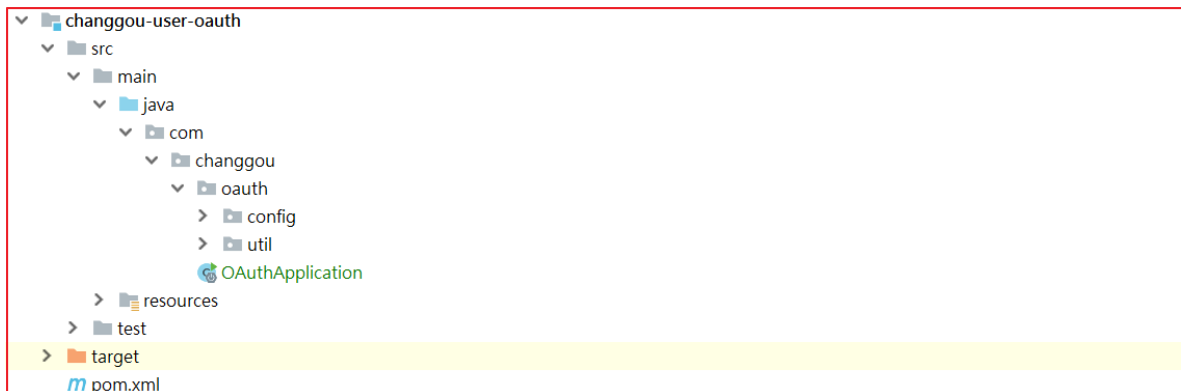
- 1、理解OAuth2的授权码认证流程及密码认证的流程。
- 2、理解spring Security OAuth2的工作流程。
- 3、掌握资源服务集成spring Security框架完成OAuth2认证的流程。

### 3.2 搭建认证服务器

关于oauth2.0服务搭建的详细流程，如果有兴趣可以参考课件中提供的oauth2.o搭建手册完成搭建。

#### 3.2.1 导入认证工程

将课件中 `day09\oauth2.0\changgou-user-oauth` 的工程导入到项目中去，如下图：



#### 3.2.2 application.yml配置



```
server:
  port: 9001
spring:
  application:
    name: user-auth
  redis:
    host: 192.168.211.132
    port: 6379
    password:
    jedis:
      pool:
        max-active: 8
        max-idle: 8
        min-idle: 0
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://192.168.211.132:3306/changgou_user?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=UTC
    username: root
    password: 123456
  main:
    allow-bean-definition-overriding: true
  eureka:
    instance:
      prefer-ip-address: true
    client:
      service-url:
        defaultZone: http://127.0.0.1:7001/eureka
  auth:
    ttl: 3600 #token存储到redis的过期时间
    clientId: changgou
    clientSecret: changgou 密钥
    cookieDomain: localhost
    cookieMaxAge: -1
  encrypt:
    key-store:
      location: classpath:/changgou.jks 证书位置
      secret: changgou 密钥
      alias: changgou
      password: changgou
```

### 3.2.2 启动授权认证服务

启动之前，记得先启动eureka，再启动该授权认证工程。

## 3.3 Oauth2授权模式

### 3.3.1 Oauth2授权模式

Oauth2有以下授权模式：

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

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。

### 3.3.2 授权码授权实现

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(资源拥有者)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

## (1)申请授权码

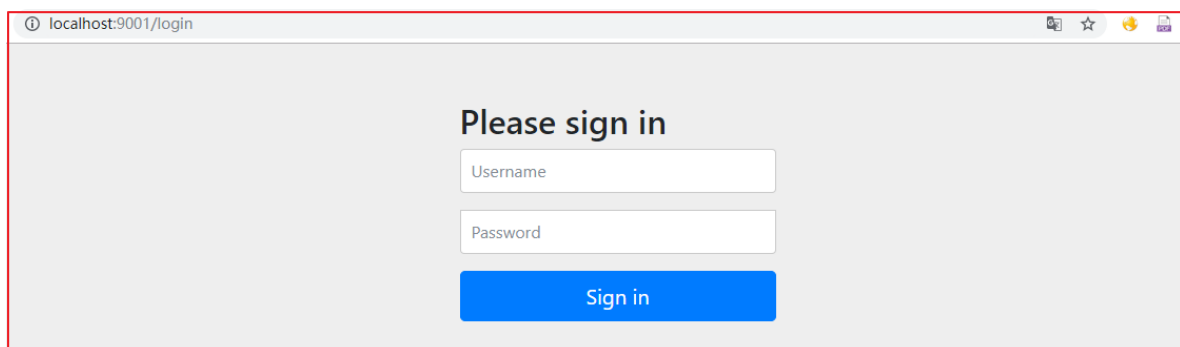
请求认证服务获取授权码：

```
1 Get请求:
2 http://localhost:9001/oauth/authorize?
  client_id=changgou&response_type=code&scop=app&redirect_uri=http://localhost
```

参数列表如下：

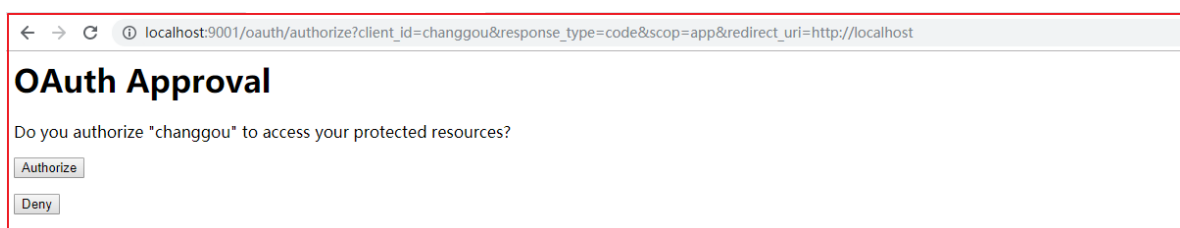
```
1 client_id: 客户端id, 和授权配置类中设置的客户端id一致。
2 response_type: 授权码模式固定为code
3 scop: 客户端范围, 和授权配置类中设置的scop一致。
4 redirect_uri: 跳转uri, 当授权码申请成功后会跳转到此地址, 并在后边带上code参数（授权码）
```

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，密钥也为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



## (2)申请令牌

拿到授权码后，申请令牌。Post请求：<http://localhost:9001/oauth/token> 参数如下：

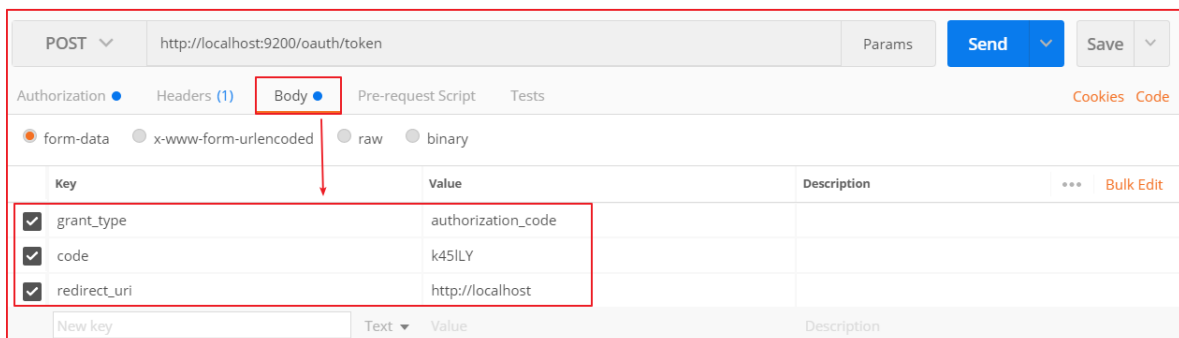
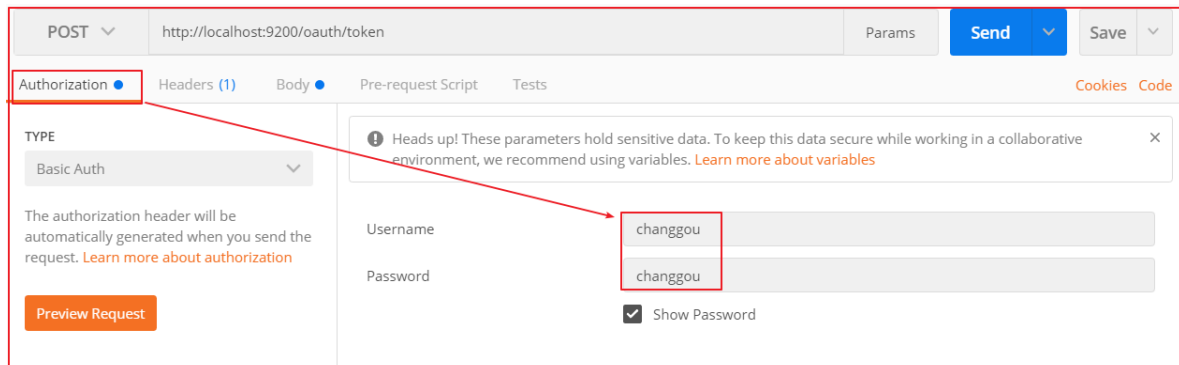
- 1 **grant\_type**: 授权类型，填写**authorization\_code**，表示授权码模式
- 2 **code**: 授权码，就是刚刚获取的授权码，注意：授权码只使用一次就无效了，需要重新申请。
- 3 **redirect\_uri**: 申请授权码时的跳转url，一定和申请授权码时用的**redirect\_uri**一致。

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：

- 1 **access\_token**: 访问令牌，携带此令牌访问资源
- 2 **token\_type**: 有MAC Token与Bearer Token两种类型，两种的校验算法不同，RFC 6750建议Oauth2采用 Bearer Token (<http://www.rfcreader.com/#rfc6750>)。
- 3 **refresh\_token**: 刷新令牌，使用此令牌可以延长访问令牌的过期时间。
- 4 **expires\_in**: 过期时间，单位为秒。
- 5 **scope**: 范围，与定义的客户端范围一致。
- 6 **jti**: 当前token的唯一标识

### (3)令牌校验

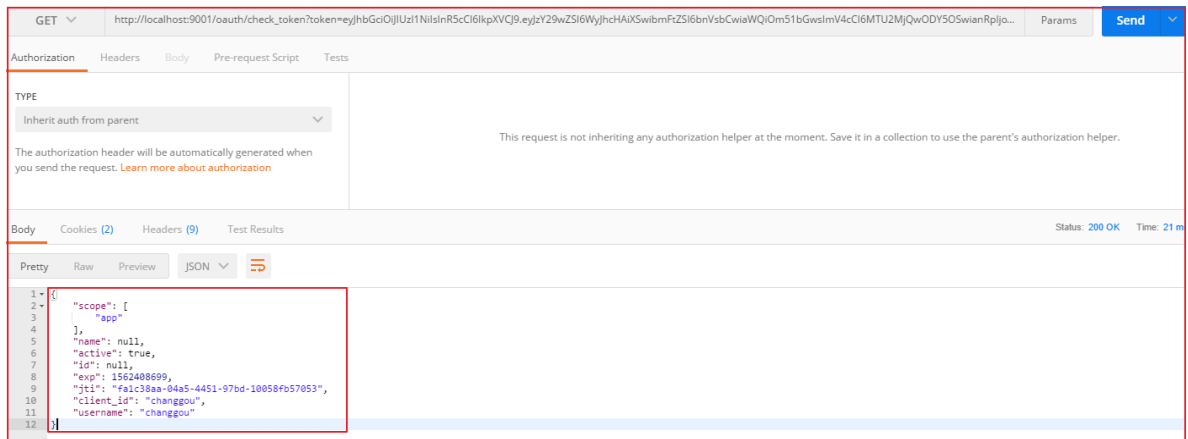
Spring Security OAuth2提供校验令牌的端点，如下：

Get: [http://localhost:9001/oauth/check\\_token?token=\[access\\_token\]](http://localhost:9001/oauth/check_token?token=[access_token])

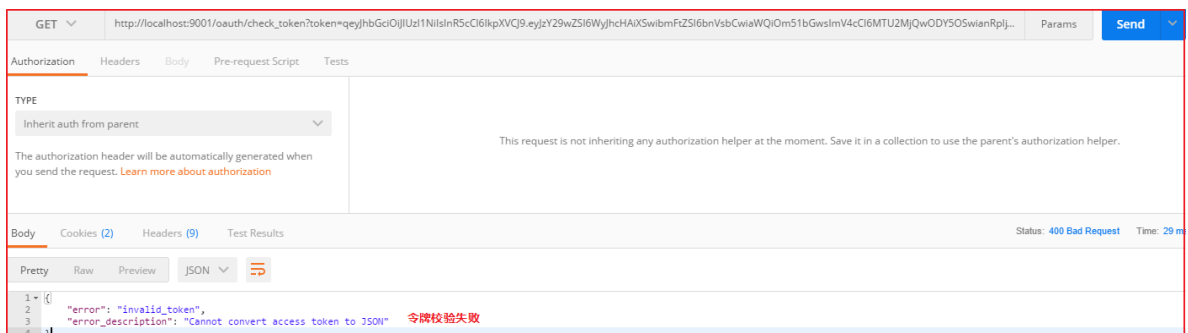
参数：

token：令牌

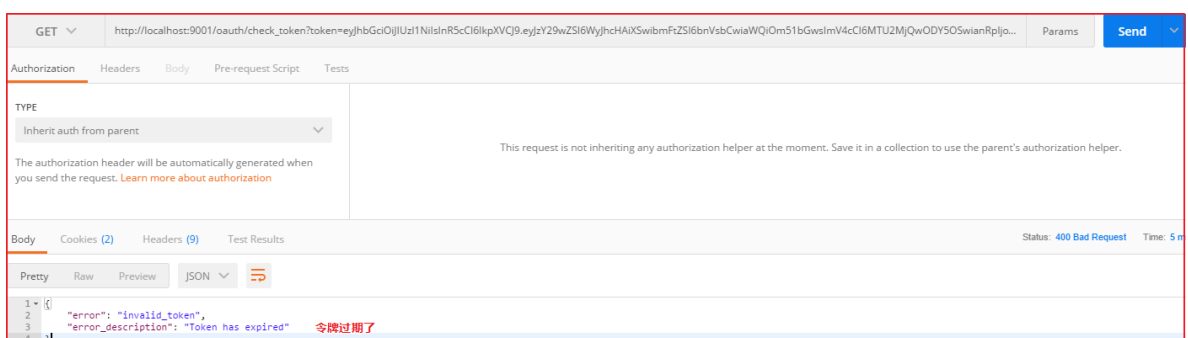
使用postman测试如下：



如果令牌校验失败，会出现如下结果：



如果令牌过期了，会如下如下结果：



### (4)刷新令牌

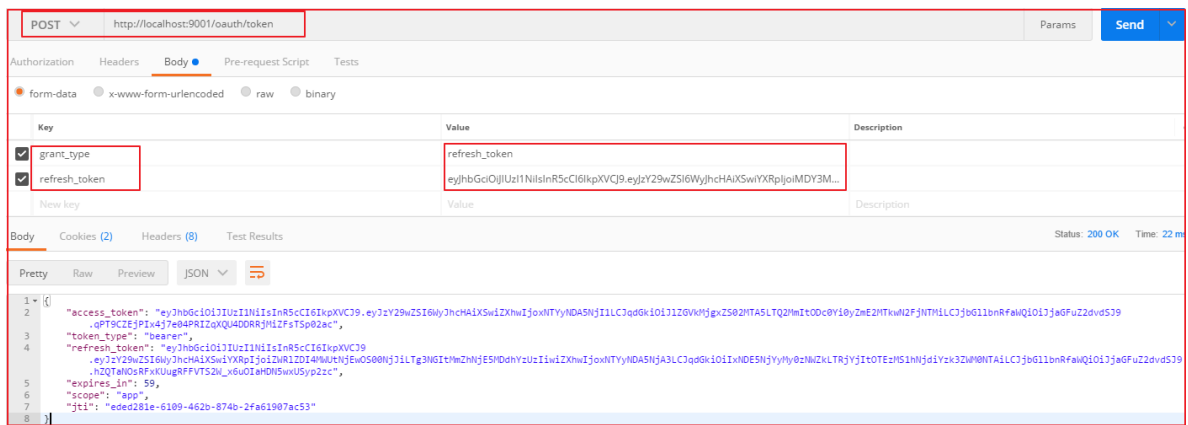
刷新令牌是当令牌快过期时重新生成一个令牌，它于授权码授权和密码授权生成令牌不同，刷新令牌不需要授权码也不需要账号和密码，只需要一个刷新令牌、客户端id和客户端密码。

测试如下：Post: <http://localhost:9001/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）



### 3.3.3 密码授权实现

#### (1)认证

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。

测试如下：

Post请求：<http://localhost:9001/oauth/token>

参数：

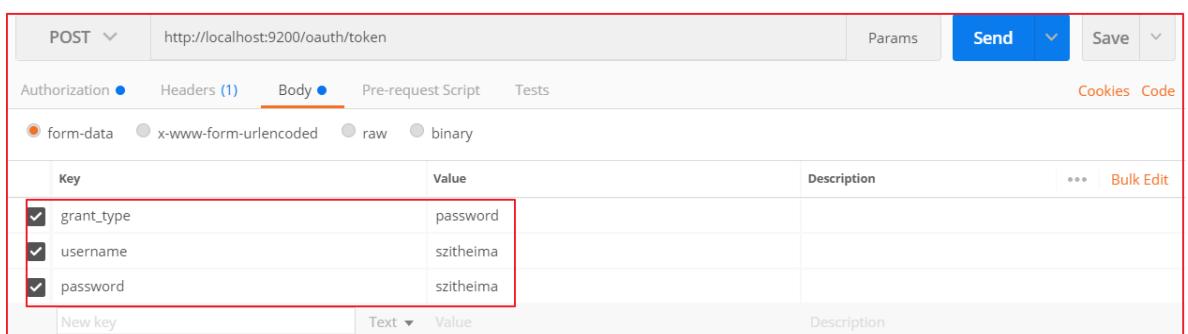
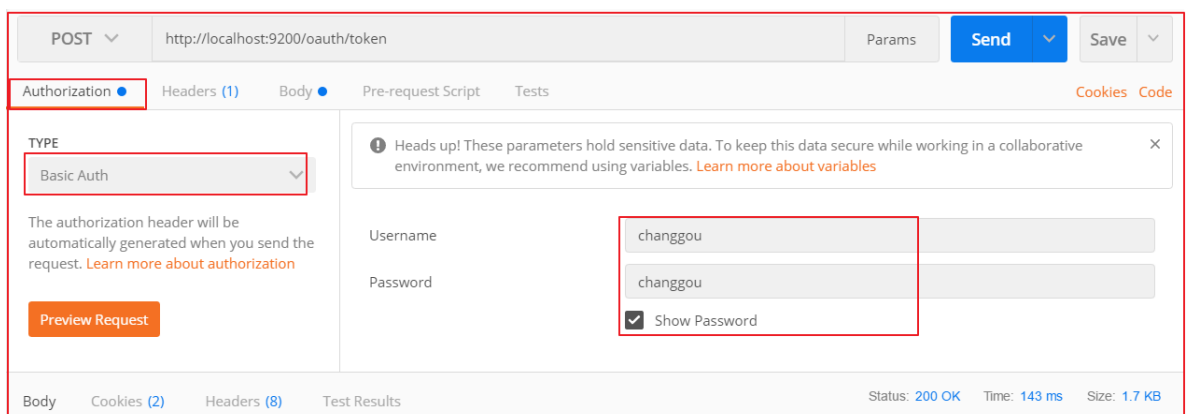
grant\_type：密码模式授权填写password

username：账号 -用户账号

password：密码 -用户密码

即使用密码授权，客户端ID和客户端秘钥也必须要传到后台认证。

并且此链接需要使用 http Basic认证。



测试数据如下：



## (2)校验令牌

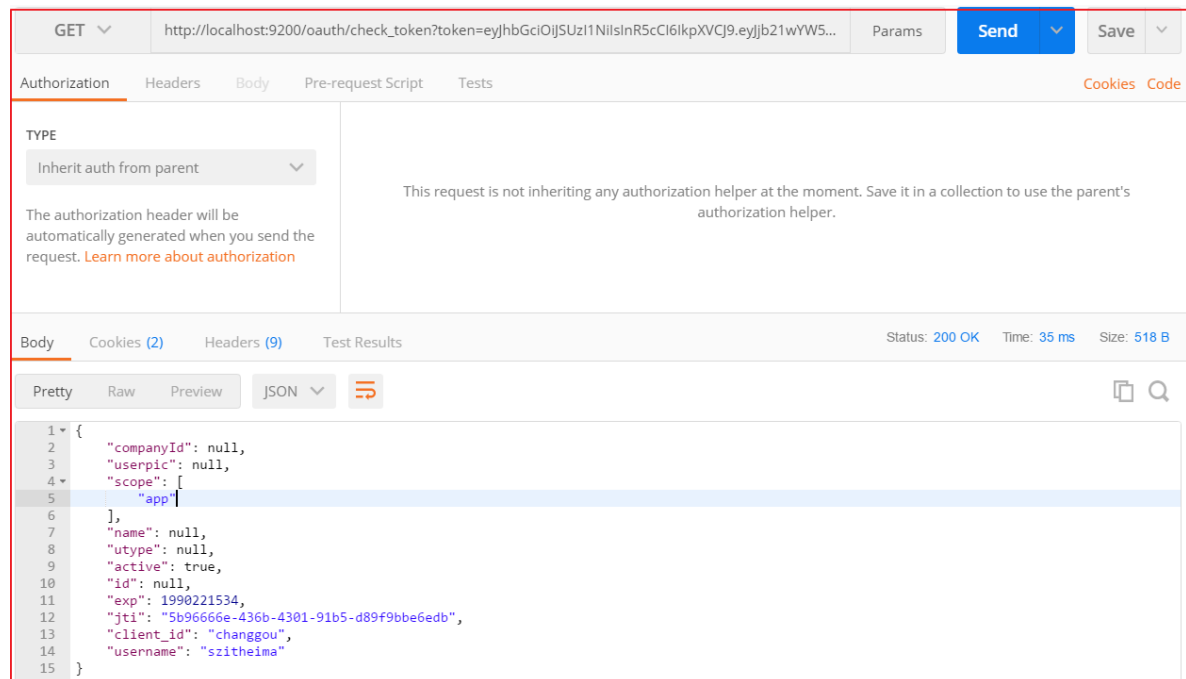
Spring Security Oauth2提供校验令牌的端点，如下：

Get: [http://localhost:9001/oauth/check\\_token?token=](http://localhost:9001/oauth/check_token?token=)

参数:

token: 令牌

使用postman测试如下:



返回结果：

```
1 {
2     "companyId": null,
3     "userpic": null,
4     "scope": [
5         "app"
6     ],
7     "name": null,
8     "utype": null,
9     "active": true,
10    "id": null,
11    "exp": 1990221534,
12    "jti": "5b96666e-436b-4301-91b5-d89f9bbe6edb",
13    "client_id": "changgou",
```

```
14     "username": "szitheima"
15 }
```

exp: 过期时间, long类型, 距离1970年的秒数 (new Date().getTime()可得到当前时间距离1970年的毫秒数)。

user\_name: 用户名

client\_id: 客户端Id, 在oauth\_client\_details中配置

scope: 客户端范围, 在oauth\_client\_details表中配置

jti: 与令牌对应的唯一标识 companyld、userpic、name、utype、

id: 这些字段是本认证服务在Spring Security基础上扩展的用户身份信息

### (3)刷新令牌

刷新令牌是当令牌快过期时重新生成一个令牌，它于授权码授权和密码授权生成令牌不同，刷新令牌不需要授权码 也不需要账号和密码，只需要一个刷新令牌、客户端id和客户端密码。

测试如下： Post: <http://localhost:9001/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)

POST

http://localhost:9200/oauth/token

Params

Send

Save

Authorization

Headers

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

Key	Value	Description
<input checked="" type="checkbox"/> grant_type	refresh_token	
<input checked="" type="checkbox"/> refresh_token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjb21wYW55...	
New key	Value	Description

Body

Cookies (2)

Headers (8)

Test Results

Status: 200 OK Time: 59 ms Size: 1.46 KB

Pretty

Raw

Preview

JSON

```

1 {
2   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
   .eyJzY29wZSI6WyJhcHAiXSwiZXhwIjoxOTkwMjIzZC00MzVlTg1NW
   EtNzcg40GUMnZzODY3IiwiaXN0bGciOiJwOTZiZjZlNSIjODQ1L
   TQyZDktYmF0MTM5MTM5MDY0MTg1LjBjBg1bnRfaWQiOiJjaGZuZ2
   dvdS39.WU1Gcrgy8iLiZWtTKdG4CX1_AKVPzrjzE58tUfwC1cu03
   PLU2sMmoAZdNjrdhFT5afI9VZV0v0gT2WjA8GLPx6RX3YDLukFj
   pAS0KGJDHmUBdo96iKk8vngYagIleo6tSmJ4S2G531gXZdayq16
   JfnU1SeK42sTrn5halNu1S0JgK0ZLxrhXmNlgGauy9CfVdy_1sF1
   MsCirhwftWgZ0rq-oU1_AIDnVPUCU44jLUTMYbSPF1z9r2rTumh4
   hX-sEWcLYi6YfDKbnL91E8xUosKrydaYAvqptny_BVe874pNRZ1Q
   qYbT2T9fz6fbBzXIjZQ1s0NB7ejff4C7vA",
3   "token_type": "bearer",
4   "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
   .eyJzY29wZSI6WyJhcHAiXSwiYXRpIjo1NmMZA0ZmEtOTYyZC00M
   zVlTg1NWEtNzcg40GUMnZzODY3IiwiaXN0bGciOiJwOTZiZjZlNS
   IjODQ1L TQyZDktYmF0MTM5MTM5MDY0MTg1LjBjBg1bnRfaWQiOi
   JjaGZuZ2dv dS39.oXcs2vsqpy9TAey06hv7qfy7_53j2LlRRDj3
   9KNI6K6vt555zqHnjp-HUG6b7Y3xH61UEsa_Vo2-zlGeqEoKTYg_a1
   U4A9xaqB36cZV4tFk1D30tYsNzb7eeHU9nBSj85GRK3Y-9_75ID
   52btkzLsQHexx-04zXP0g0mY1YcM0Fw0Z8IyglDZzXySymKwFZp3-n1
   FMVfZWAwZ3bueDt_xeQ5usjJWgZFT_Dc3p25Wn34j8z1tUgA4j6
   ndfvmv3FMM1t12-QayNsfdH-wh1KZvtgkC1Eb0lHYHHXyDWA6f1
   JSLXNLIAL4Bto9Bjg04kACDGxmdn190wQ",
5   "expires_in": 431999999,
6   "scope": "app",
7   "jti": "6c3004fa-962d-43eb-855a-7788e076a867"
8 }
```

刷新令牌成功，会重新生成新的访问令牌和刷新令牌，令牌的有效期的也比旧令牌长。

刷新令牌通常是在令牌快过期时进行刷新。

## OAuth2.0总结

授权模式4种：

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

授权码模式授权：

- 1 1. 申请授权码 [http://localhost:9001/oauth/authorize?](http://localhost:9001/oauth/authorize?client_id=changgou&response_type=code&scope=app&redirect_uri=http://localhost)  
[client\\_id=changgou&response\\_type=code&scope=app&redirect\\_uri=http://localhost](http://localhost:9001/oauth/authorize?client_id=changgou&response_type=code&scope=app&redirect_uri=http://localhost)
- 2
- 3 2. 登录界面输入客户端ID和客户端密钥
- 4
- 5 3. 点击authorize授权获取授权码
- 6
- 7 4. 用户授权码获取令牌 <http://localhost:9001/oauth/token>

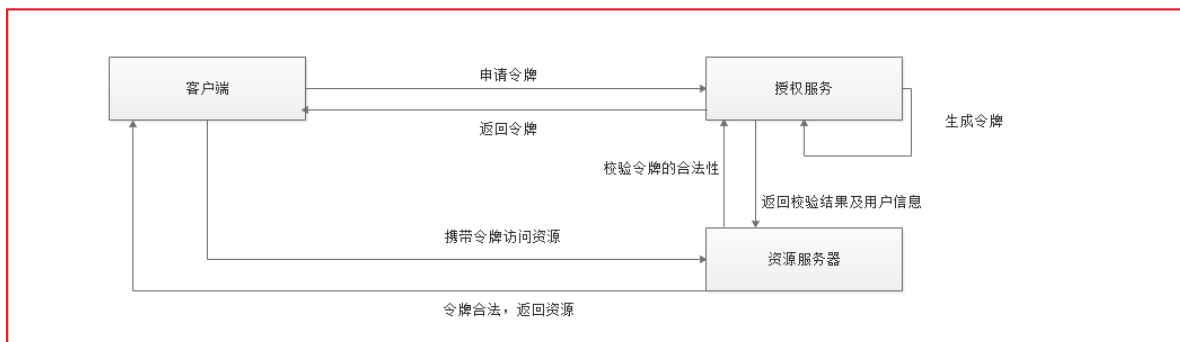
密码模式授权：

- 1 1. 客户端ID和客户端密钥需要传递到后台
- 2
- 3 2. 用户账号和密码

## 4 资源服务授权

### 4.1 资源服务授权流程

(1)传统授权流程

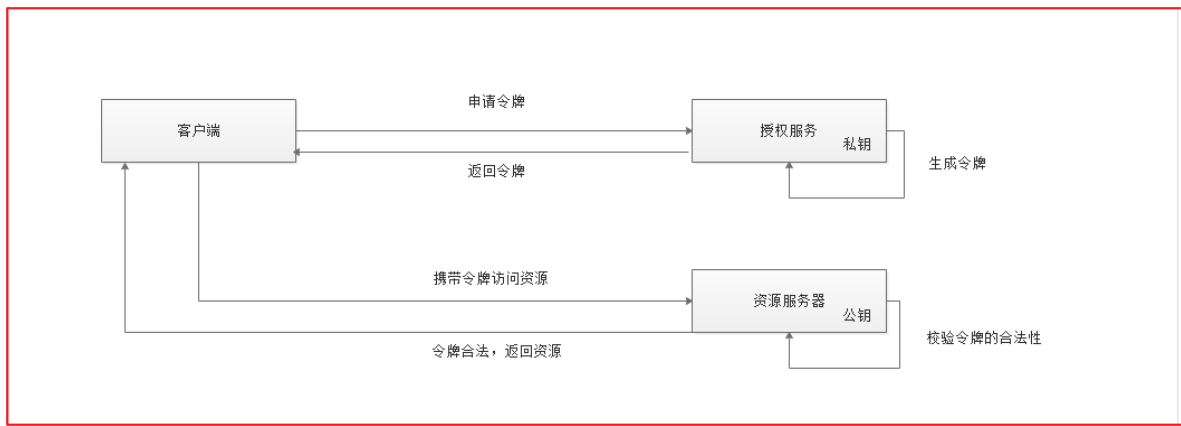


资源服务器授权流程如上图，客户端先去授权服务器申请令牌，申请令牌后，携带令牌访问资源服务器，资源服务器访问授权服务校验令牌的合法性，授权服务会返回校验结果，如果校验成功会返回用户信息给资源服务器，资源服务器如果接收到的校验结果通过了，则返回资源给客户端。

传统授权方法的问题是用户每次请求资源服务，资源服务都需要携带令牌访问认证服务去校验令牌的合法性，并根据令牌获取用户的相关信息，性能低下。

(2)公钥私钥授权流程





传统的授权模式性能低下，每次都需要请求授权服务校验令牌合法性，我们可以利用公钥私钥完成对令牌的加密，如果加密解密成功，则表示令牌合法，如果加密解密失败，则表示令牌无效不合法，合法则允许访问资源服务器的资源，解密失败，则不允许访问资源服务器资源. 非对称加密

上图的业务流程如下：

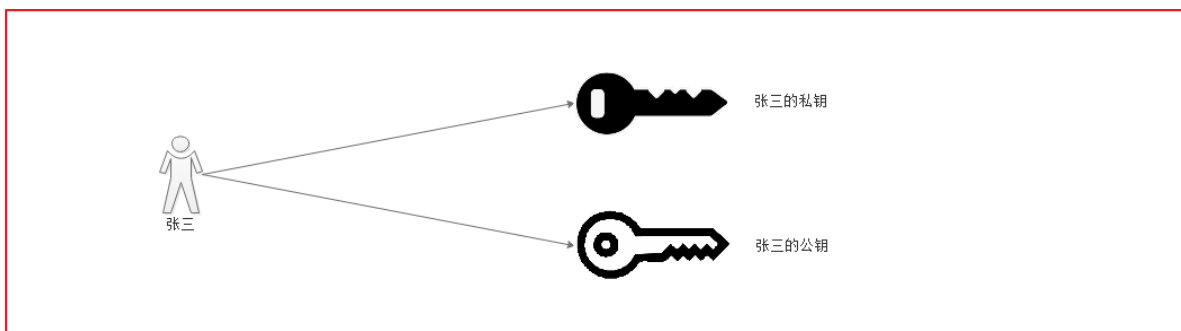
- 1、客户端请求认证服务申请令牌
- 2、认证服务生成令牌认证服务采用非对称加密算法，使用私钥生成令牌。
- 3、客户端携带令牌访问资源服务客户端在Http header 中添加： **Authorization: Bearer 令牌**。
- 4、资源服务请求认证服务校验令牌的有效性资源服务接收到令牌，使用公钥校验令牌的合法性。
- 5、令牌有效，资源服务向客户端响应资源信息

## 4.2 公钥私钥

在对称加密的时代，加密和解密用的是同一个密钥，这个密钥既用于加密，又用于解密。这样做有一个明显的缺点，如果两个人之间传输文件，两个人都要知道密钥，如果是三个人呢，五个人呢？于是就产生了非对称加密，用一个密钥进行加密（公钥），用另一个密钥进行解密（私钥）。

### 4.2.1 公钥私钥原理

张三有两把钥匙，一把是公钥，另一把是私钥。



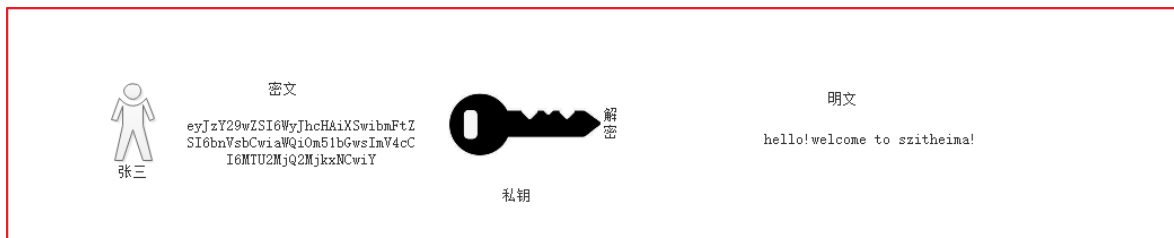
张三把公钥送给他的朋友们----李四、王五、赵六----每人一把。



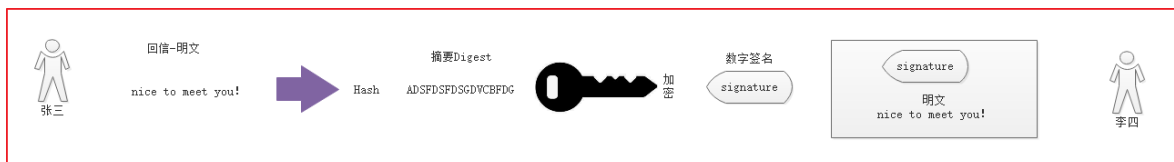
李四要给张三写一封保密的信。她写完后用张三的公钥加密，就可以达到保密的效果。



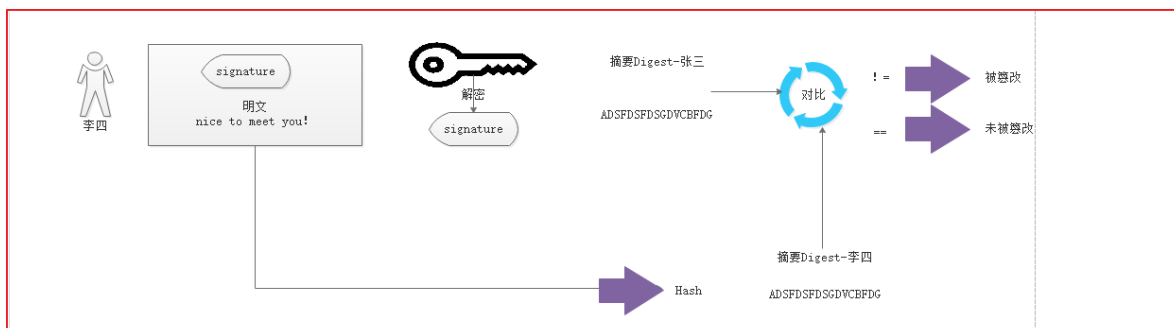
张三收信后，用私钥解密，就看到了信件内容。这里要强调的是，只要张三的私钥不泄露，这封信就是安全的，即使落在别人手里，也无法解密。



张三给李四回信，决定采用"数字签名"。他写完后先用Hash函数，生成信件的摘要（digest）。张三将这个签名，附在信件下面，一起发给李四。



李四收信后，取下数字签名，李四再对信件本身使用Hash函数，将得到的结果，与上一步得到的摘要进行对比。如果两者一致，就证明这封信未被修改过。



## 4.2.2 生成私钥公钥

Spring Security 提供对JWT的支持，本节我们使用Spring Security 提供的JwtHelper来创建JWT令牌，校验JWT令牌 等操作。这里JWT令牌我们采用非对称算法进行加密，所以我们要先生成公钥和私钥。

(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

创建一个文件夹，在该文件夹下执行如下命令行（生成证书：包含公钥、私钥-是一对）：

```
1 | 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

```
1 | -alias: 密钥的别名
2 | -keyalg: 使用的hash算法
3 | -keypass: 密钥的访问密码
4 | -keystore: 密钥库文件名，xc.keystore保存了生成的证书
5 | -storepass: 密钥库的访问密码
```

(2)查询证书信息

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

(3)删除别名

```
1 | keytool -delete -alias changgou -keystore changgou.jks
```

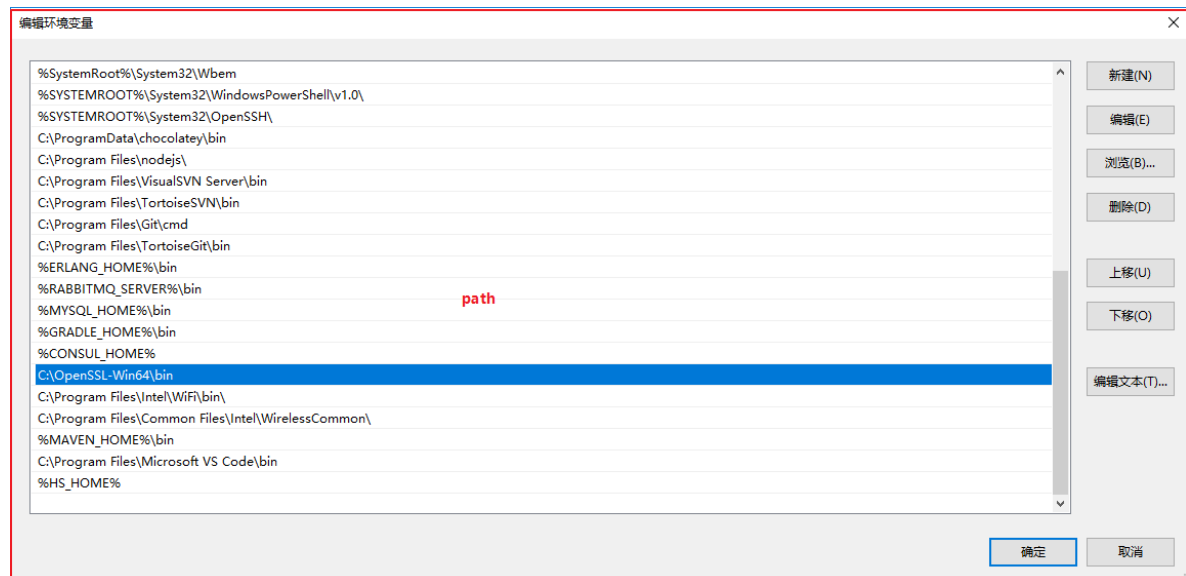
## 4.2.3 导出公钥

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\_0g.exe

配置openssl的path环境变量，如下图：



本教程配置在C:\OpenSSL-Win64\bin

cmd进入changgou.jks文件所在目录执行如下命令(如下命令在windows下执行，会把-变成中文方式，请将它改成英文的-)：

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

```
C:\Users\71075\Desktop\cret>keytool -list -rfc --keystore changgou.jks | openssl x509 -inform pem -pubkey
输入密钥库口令: changgou
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAfSEiaLvij9C1Mz+oyAm
t47whAaRkRu/8kePM+X8760UGU0RMwGti6Z9y3LQ0RvK6I0brXmbGB/RsN38PVnh
cP8ZfxGUH26kX0RK+t1rxcrG+HkPYOH4XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEm
oLKyMsvTviG5f1TpDprT25unWgE4md1kthRWXOnfWHATVY7Y/r4obiOL1mS5bEa/
iNKotQnnvIAKtjBM4RlIDWMA6dmz+1HtLtqDD2LF1qwoiSIHI75LQZ/CNYaHCfZS
xtOydpNKq8eb1/PGiLNo1D4La2zf0/1d1cr5mkesV570NxRmU1tFm8Zd3MZ1Zmyv
9QIDAQAB
-----END PUBLIC KEY-----
-----BEGIN CERTIFICATE-----
MIIDXzCCAQegAwIBAgIEQkT0oJANBgkqhkiG9w0BAQsFADBgMQswCQYDVQQGEwJj
bJELMAkGA1UECmCZ2QxZCZAJBgNVBACTAuNGMREwDwYDVQQKEWhjaGFuZ2dvZDTER
MA8GA1UECmIY2hhbmdnb3UxETAPBgNVBAMTCGNoYW5nZ291MB4XDTE5MDUxNzIw
MDMyNFoXDTE5MDg5NTIwMDMyNFoYDDELMAkGA1UEBhMCY24xZCZAJBgNVBAGTAmdk
MQswCQYDVQQHJEwJzejERMA8GA1UEChMIY2hhbmdnb3UxETAPBgNVBAsTCGNoYW5n
Z291MREwDwYDVQQDEWhjaGFuZ2dvZDTCASIdQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBALxbBImi74o/QtTM/qMgJreO8IQGkZEbv/JHjzP1/O+tFB1NETMBrYum
fcty0NEbyuiNG615mxgf0bDd/D1Z4XD/GXSR1B9upF9ESvrZa8XKxvh5D2Dh+Fzw
C/ENZbtZ/cd7S3CD8avGWbbiMimBjQcysjLL074huX5U6Q6a09ubploB0JndZLYU
V1zp31hwE1W02P6+KG4ji9ZkuWxGv4jSgLUdZ7yACrYwTOEzSA1jGunZs/pr7S7a
gw9ixdasKlkiByO+S0GfwjWChwn2UbsTsnATsqvHm9fzxoizaJQ+C2ts39P9XZXK
+ZpHrFee9DcUZ1NBRZvGXdzGZWZsr/UCAwEAAAMhMB8wHQYDVRO0BBYEFD1S24hM
6d3PjUW4QNA91iPVq+TMA0GCSqGSIb3DQEBCwUAA4IBAQBvLqfGerXnoxT4roBo
8/PY7+Irl1QfpQWSYWiDeY7+ek5c7AGCg0k5dVC4GxdgsCdCmNM9VlcS7r2j1mD1
t04g2MABBYWkkaVunSdYm73KhF4ktz+QPrBnUKisZnE6wc1P1+MBE6J61uJR47Sx
Cd/bbqwr7JjfpvpeAux3NVNaL6YN8K4M4Zwapu+GtiwNyoBXk6bTPp4fdgWKpW6X
i+DX64vCOWWqiQSRMSIZ+1RzPBRjaOXqk1JuFbYIbnez9eVd1I264ziUIUvBQfES
bqZrk/bvyt51PnmpZJBthiEe/C3mMmsLwsUTN7jsIldtJaZ1lffmqar3bqxTxRdc
w60x
-----END CERTIFICATE-----
C:\Users\71075\Desktop\cret>_
```

下面段内容是公钥

```
1 | -----BEGIN PUBLIC KEY-----
2 | MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAfSEiaLvij9C1Mz+oyAm
3 | t47whAaRkRu/8kePM+X8760UGU0RMwGti6Z9y3LQ0RvK6I0brXmbGB/RsN38PVnh
4 | cP8ZfxGUH26kX0RK+t1rxcrG+HkPYOH4XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEm
5 | oLKyMsvTviG5f1TpDprT25unWgE4md1kthRWXOnfWHATVY7Y/r4obiOL1mS5bEa/
6 | iNKotQnnvIAKtjBM4RlIDWMA6dmz+1HtLtqDD2LF1qwoiSIHI75LQZ/CNYaHCfZS
7 | xtOydpNKq8eb1/PGiLNo1D4La2zf0/1d1cr5mkesV570NxRmU1tFm8Zd3MZ1Zmyv
8 | 9QIDAQAB
9 | -----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。

## 4.2.4 JWT令牌

### (1)创建令牌数据

在changgou-user-oauth工程中创建测试类com.changgou.token.CreateJwtTest，使用它来创建令牌信息，代码如下：

```
1 | public class CreateJwtTest {
2 |
3 |     /**
4 |      * 创建令牌测试
5 |      */
6 |     @Test
7 |     public void testCreateToken(){
8 |         //证书文件路径
9 |         String key_location="changgou.jks";
10 |         //秘钥库密码
```

```

11     String key_password="changgou";
12     //密钥密码
13     String keypwd = "changgou";
14     //密钥别名
15     String alias = "changgou";
16
17     //访问证书路径
18     ClassPathResource resource = new ClassPathResource(key_location);
19
20     //创建密钥工厂
21     KeyStoreKeyFactory keyStoreKeyFactory = new
KeyStoreKeyFactory(resource, key_password.toCharArray());
22
23     //读取密钥对(公钥、私钥)
24     KeyPair keyPair =
keyStoreKeyFactory.getKeyPair(alias, keypwd.toCharArray());
25
26     //获取私钥
27     RSAPrivateKey rsaPrivate = (RSAPrivateKey) keyPair.getPrivate();
28
29     //定义Payload
30     Map<String, Object> tokenMap = new HashMap<>();
31     tokenMap.put("id", "1");
32     tokenMap.put("name", "itheima");
33     tokenMap.put("roles", "ROLE_VIP,ROLE_USER");
34
35     //生成Jwt令牌
36     Jwt jwt = JwtHelper.encode(JSON.toJSONString(tokenMap), new
RsaSigner(rsaPrivate));
37
38     //取出令牌
39     String encoded = jwt.getEncoded();
40     System.out.println(encoded);
41 }
42 }

```

运行后的结果如下：

```

1 eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlcyI6I1JPTeVfVklQLFJPTeVfVWVNFuiIs
Im5hbWUiOiJpdGhlaw1hIiwiaWQiOiIxIn0. IR9Qu9ZqYZ2gu2qgAzyT38UhEeL4oi69ko-
dzc_P9-Vjz40hwZDqx18wZ-W2WAw1ewGIHV1EYDjg0-eilogJZ5Uikyww1bewXCpv1M-
ZRtyQQqHFT1fDiVcFetyTayaskwa-x_BVS4pTWaskiaIKbKR4KcME2E5o1rEek-
3YPkqAiZ6WP1UOmpaCJDaaFsdninQG0gzSCuGvLuG40x0Ngpfk7mPOecsIi5cbJE1pdYusCr9oXc5
3RoyfvyPhjzV7c2D5eIZu3leUPXRvvVAPJFECSBiisxUSEeiGpmuQhaFZd1g-
yJ1lWQrixFvehMeLX2XU6W1n1L5ARTpQf_Jjiw

```

## (2)解析令牌

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：

```

1 public class ParseJwtTest {

```

```

2
3     /**
4      * 校验令牌
5      */
6     @Test
7     public void testParseToken(){
8         //令牌
9         String token =
10         "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlcyI6I1JPTeVfVklQLFJPTeVfVWVNFui
11         IsIm5hbWUiOiJpdGh1aw1hIiwiaWQiOiIxIn0. IR9Qu9ZqYZ2gu2qgAzyT38UhEeL40i69ko-
12         dzC_P9-Vjz40hwZDqx18wZ-W2WAw1ewGIHV1EYDjg0-eilogJZ5Uikyww1bewXCpv1M-
13         ZRtYQQqHFT1fDiVcFetyTayaskwa-x_BVS4pTWaskiaIKbKR4KcME2E5o1rEek-
14         3YPkqAiZ6WP1UompacJDaaFSdninqG0gzSCuGvLuG40X0Ngpfk7mPOecsIi5cbJElpdYUSCr9oXc
15         53ROyfvYPHjzV7c2D5eIZu31eUPXRvvVAPJFECSBiisxUSEeiGpmuQhaFZd1g-
16         yJ1wQrixFvehMeLX2XU6W1n1L5ARTpQf_jjiw";
17
18         //公钥
19         String publicKey = "-----BEGIN PUBLIC KEY-----
20         MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvFsEiaLvi9C1Mz+oyAmt47whAaRku/
21         8kePM+X8760UGU0RMWgti6Z9y3LQ0RvK6I0brXmbGB/RsN38PVnhcP8ZfxGUH26kX0RK+t1rxcrG
22         +HkPYOH4XPAL8Q11u1n9x3tLcIPxq8ZZtuIyKYEmoLKyMsvTviG5f1TpDprt25unWgE4md1kthRW
23         XonfWHATVY7Y/r4obiOL1mS5bEa/iNKotQNNvIAktjBM4RlIDWma6dmz+lHtLtqDD2LF1qwoiSIH
24         I75LQZ/CNYaHCFZSxtOydpNKq8eb1/PGiLNo1D4La2zf0/1dlcr5mkesv570NxRmU1tFm8Zd3MZ1
25         Zmyv9QIDAQAB-----END PUBLIC KEY-----";
26
27         //校验Jwt
28         Jwt jwt = JwtHelper.decodeAndVerify(token, new
29         Rsaverifier(publickey));
30
31         //获取Jwt原始内容
32         String claims = jwt.getClaims();
33         System.out.println(claims);
34         //jwt令牌
35         String encoded = jwt.getEncoded();
36         System.out.println(encoded);
37     }
38 }

```

运行后的结果如下：

```

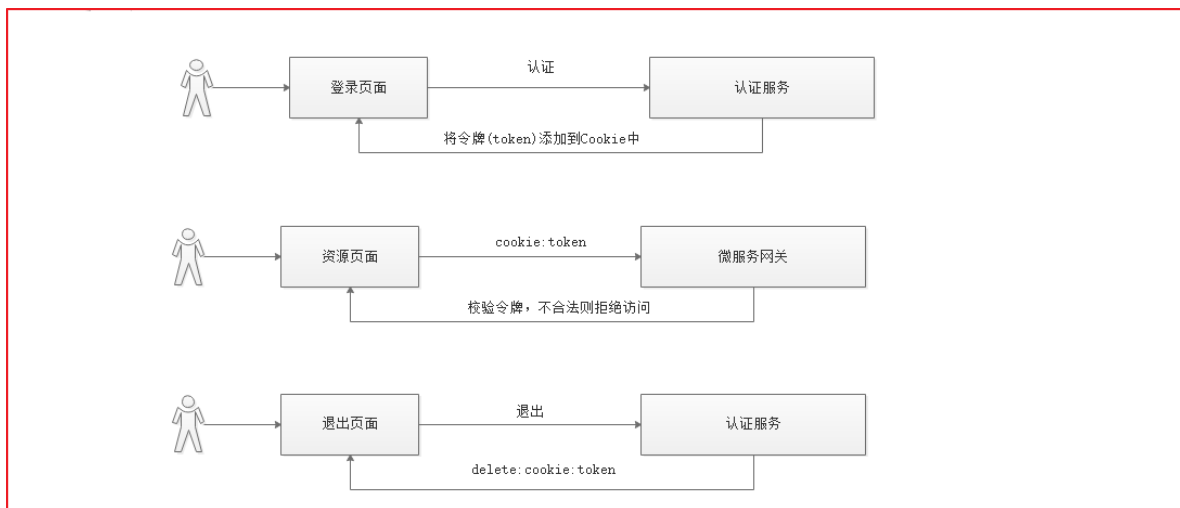
{"roles":["ROLE_VIP","ROLE_USER"],"name":"itheima","id":"1"}
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlcyI6I1JPTeVfVklQLFJPTeVfVWVNFuiIsIm5hbWUiOiJpdGh1aw1hIiwiaWQiOiIxIn0. IR9Qu9ZqYZ2gu2qgAzyT38UhEeL40i69ko-dzC_P9-

```

## 5 认证开发

### 5.1 需求分析

用户登录的流程图如下：



执行流程：

- 1 1、用户登录，请求认证服务
- 2 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cookie
- 3 3、用户访问资源页面，带着cookie到网关
- 4 4、网关从cookie获取token，如果存在token，则校验token合法性，如果不合法则拒绝访问，否则放行
- 5 5、用户退出，请求认证服务，删除cookie中的token

## 5.2 认证服务

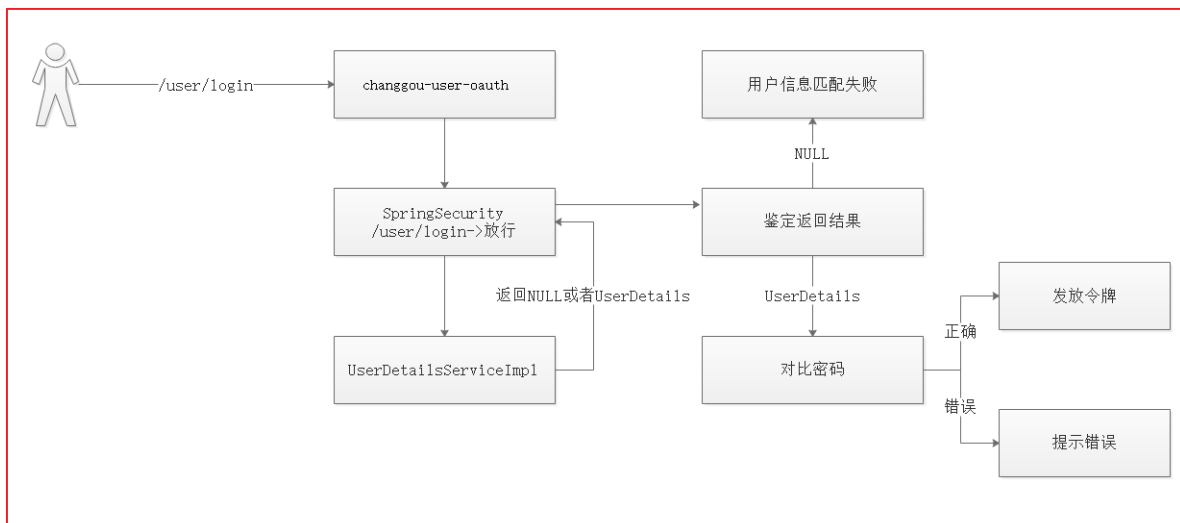
### 5.2.1 认证需求分析

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，并将令牌写入cookie。  
(header/cookie/param)

#### 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cookie中删除。



UserDetails:用户信息的封装->用户信息的载体->令牌的载荷

## 5.2.2 工具封装

在changgou-user-oauth工程中添加如下工具对象，方便操作令牌信息。

创建com.changgou.oauth.util.AuthToken类，存储用户令牌数据，代码如下：

```
1 public class AuthToken implements Serializable{
2
3     //令牌信息
4     String accessToken;
5     //刷新token(refresh_token)
6     String refreshToken;
7     //jwt短令牌
8     String jti;
9
10    //...get...set
11 }
```

创建com.changgou.oauth.util.CookieUtil类，操作Cookie,代码如下：

```
1 public class CookieUtil {
2
3     /**
4      * 设置cookie
5      *
6      * @param response
7      * @param name      cookie名字
8      * @param value      cookie值
9      * @param maxAge      cookie生命周期 以秒为单位
10     */
11     public static void addCookie(HttpServletResponse response, String
12 domain, String path, String name,
13                                     String value, int maxAge, boolean httpOnly)
14     {
15         Cookie cookie = new Cookie(name, value);
16         cookie.setDomain(domain);
17         cookie.setPath(path);
18         cookie.setMaxAge(maxAge);
19         cookie.setHttpOnly(httpOnly);
20         response.addCookie(cookie);
21     }
22
23     /**
24      * 根据cookie名称读取cookie
25      * @param request
26      * @return map<cookieName,cookieValue>
27     */
28     public static Map<String,String> readCookie(HttpServletRequest request,
29 String ... cookieNames) {
30         Map<String,String> cookieMap = new HashMap<String,String>();
31         Cookie[] cookies = request.getCookies();
32         if (cookies != null) {
33             for (Cookie cookie : cookies) {
34                 String cookieName = cookie.getName();
35                 String cookieValue = cookie.getValue();
36                 for(int i=0;i<cookieNames.length;i++){
```



```

35         if(cookieNames[i].equals(cookieName)){
36             cookieMap.put(cookieName,cookieValue);
37         }
38     }
39 }
40 }
41 return cookieMap;
42 }
43 }

```

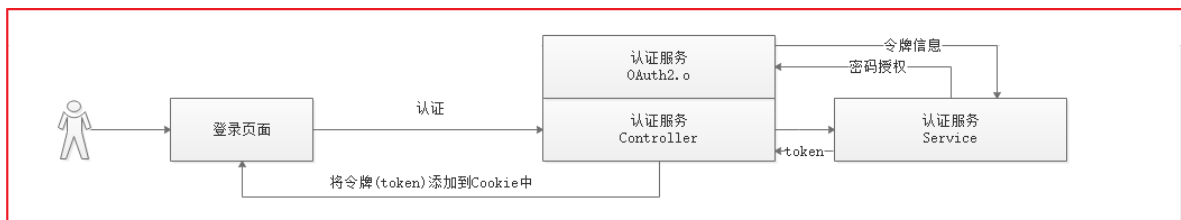
创建com.changgou.oauth.util.UserJwt类，封装SpringSecurity中User信息以及用户自身基本信息,代码如下：

```

1 public class UserJwt extends User {
2     private String id;    //用户ID
3     private String name;  //用户名字
4
5     public UserJwt(String username, String password, Collection<? extends
GrantedAuthority> authorities) {
6         super(username, password, authorities);
7     }
8
9     //...get...set
10 }

```

### 5.2.3 业务层



如上图，我们现在实现一个认证流程，用户从页面输入账号密码，到认证服务的Controller层，Controller层调用Service层，Service层调用OAuth2.0的认证地址，进行密码授权认证操作，如果账号密码正确了，就返回令牌信息给Service层，Service将令牌信息给Controller层，Controller层将数据存入到Cookie中，再响应用户。

登录过程总结：

```

1 参数传递：
2 1.账号      username=szitheima
3 2.密码      password=szitheima
4 3.授权方式   grant_type=password
5
6 请求头传递
7 4.Basic Base64(客户端ID:客户端密钥)  Authorization=Basic
Y2hhbmdnb3U6Y2hhbmdnb3U=
8 changgou:changgou

```

创建com.changgou.oauth.service.AuthService接口，并添加授权认证方法：

```
1 public interface AuthService {
2
3     /**
4      * 授权认证方法
5      */
6     AuthToken login(String username, String password, String clientId, String
7     clientSecret);
8 }
```

创建com.changgou.oauth.service.impl.AuthServiceImpl实现类，实现获取令牌数据，这里认证获取令牌采用的是密码授权模式，用的是RestTemplate向OAuth服务发起认证请求，代码如下：

```
1 @Service
2 public class AuthServiceImpl implements AuthService {
3
4     @Autowired
5     private LoadBalancerClient loadBalancerClient;
6
7     @Autowired
8     private RestTemplate restTemplate;
9
10    /**
11     * 授权认证方法
12     * @param username
13     * @param password
14     * @param clientId
15     * @param clientSecret
16     * @return
17     */
18    @Override
19    public AuthToken login(String username, String password, String
20    clientId, String clientSecret) {
21        //申请令牌
22        AuthToken authToken = applyToken(username,password,clientId,
23        clientSecret);
24        if(authToken == null){
25            throw new RuntimeException("申请令牌失败");
26        }
27        return authToken;
28    }
29
30    /**
31     * 认证方法
32     * @param username: 用户登录名字
33     * @param password: 用户密码
34     * @param clientId: 配置文件中的客户端ID
35     * @param clientSecret: 配置文件中的密钥
36     * @return
37     */
38 }
```

```

36     */
37     private AuthToken applyToken(String username, String password, String
clientId, String clientSecret) {
38         //选中认证服务的地址
39         ServiceInstance serviceInstance = loadBalancerClient.choose("user-
auth");
40         if (serviceInstance == null) {
41             throw new RuntimeException("找不到对应的服务");
42         }
43         //获取令牌的url
44         String path = serviceInstance.getUri().toString() +
"/oauth/token";
45         //定义body
46         MultivalueMap<String, String> formData = new LinkedMultivalueMap<>
();
47         //授权方式
48         formData.add("grant_type", "password");
49         //账号
50         formData.add("username", username);
51         //密码
52         formData.add("password", password);
53         //定义头
54         MultivalueMap<String, String> header = new LinkedMultivalueMap<>
();
55         header.add("Authorization", httpbasic(clientId, clientSecret));
56         //指定 restTemplate当遇到400或401响应时候也不要抛出异常，也要正常返回值
57         restTemplate.setErrorHandler(new DefaultResponseErrorHandler() {
58             @Override
59             public void handleError(ClientHttpResponse response) throws
IOException {
60                 //当响应的值为400或401时候也要正常响应，不要抛出异常
61                 if (response.getRawStatusCode() != 400 &&
response.getRawStatusCode() != 401) {
62                     super.handleError(response);
63                 }
64             }
65         });
66         Map map = null;
67         try {
68             //http请求spring security的申请令牌接口
69             ResponseEntity<Map> mapResponseEntity =
restTemplate.exchange(path, HttpMethod.POST, new
HttpEntity<MultivalueMap<String, String>>(formData, header), Map.class);
70             //获取响应数据
71             map = mapResponseEntity.getBody();
72         } catch (RestClientException e) {
73             throw new RuntimeException(e);
74         }
75         if (map == null || map.get("access_token") == null ||
map.get("refresh_token") == null || map.get("jti") == null) {
76             //jti是jwt令牌的唯一标识作为用户身份令牌
77             throw new RuntimeException("创建令牌失败! ");
78         }
79
80         //将响应数据封装成AuthToken对象
81         AuthToken authToken = new AuthToken();
82         //访问令牌(jwt)
83         String accessToken = (String) map.get("access_token");

```

```

84         //刷新令牌(jwt)
85         String refreshToken = (String) map.get("refresh_token");
86         //jti, 作为用户的身份标识
87         String jwtToken= (String) map.get("jti");
88         authToken.setJti(jwtToken);
89         authToken.setAccessToken(accessToken);
90         authToken.setRefreshToken(refreshToken);
91         return authToken;
92     }
93
94
95     /**
96      * base64编码
97      * @param clientId
98      * @param clientSecret
99      * @return
100     */
101     private String httpbasic(String clientId,String clientSecret){
102         //将客户端id和客户端密码拼接, 按“客户端id:客户端密码”
103         String string = clientId+":"+clientSecret;
104         //进行base64编码
105         byte[] encode = Base64Utils.encode(string.getBytes());
106         return "Basic "+new String(encode);
107     }
108 }

```

## 5.2.4 控制层

创建控制层com.changgou.oauth.controller.AuthController, 编写用户登录授权方法, 代码如下:

```

1  @RestController
2  @RequestMapping(value = "/user")
3  public class AuthController {
4
5      //客户端ID
6      @Value("${auth.clientId}")
7      private String clientId;
8
9      //密钥
10     @Value("${auth.clientSecret}")
11     private String clientSecret;
12
13     //Cookie存储的域名
14     @Value("${auth.cookieDomain}")
15     private String cookieDomain;
16
17     //Cookie生命周期
18     @Value("${auth.cookieMaxAge}")
19     private int cookieMaxAge;
20
21     @Autowired
22     AuthService authService;
23
24     @PostMapping("/login")
25     public Result login(String username, String password) {

```

```

26         if(StringUtils.isEmpty(username)){
27             throw new RuntimeException("用户名不允许为空");
28         }
29         if(StringUtils.isEmpty(password)){
30             throw new RuntimeException("密码不允许为空");
31         }
32         //申请令牌
33         AuthToken authToken =
34             authService.login(username,password,clientId,clientSecret);
35
36         //用户身份令牌
37         String access_token = authToken.getAccessToken();
38         //将令牌存储到cookie
39         saveCookie(access_token);
40
41         return new Result(true, StatusCode.OK, "登录成功! ");
42     }
43
44     /**
45      * 将令牌存储到cookie
46      * @param token
47      */
48     private void saveCookie(String token){
49         HttpServletResponse response = ((ServletRequestAttributes)
50             RequestContextHolder.getRequestAttributes()).getResponse();
51
52         CookieUtil.addCookie(response,cookieDomain,"/", "Authorization", token, cookie
53             MaxAge, false);
54     }
55 }

```

### 5.2.5 测试认证接口

使用postman测试:

Post请求: <http://localhost:9001/user/login>

[illegible]

