**畅购电商系统开发**

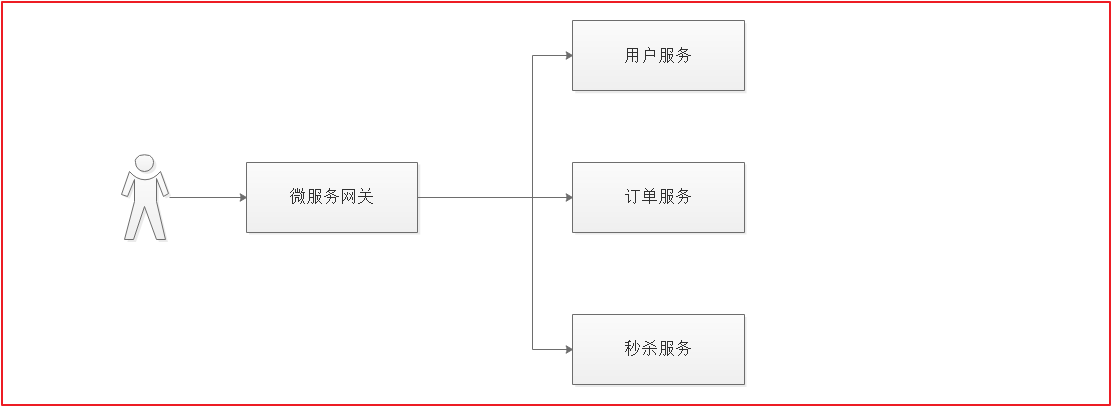
**第9天**

传智播客.黑马程序员.深圳

# 学习目标

* 用户认证分析
* 认证技术方案了解
* SpringSecurity Oauth2.0入门
* 用户授权认证开发

# 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。

## 认证与授权

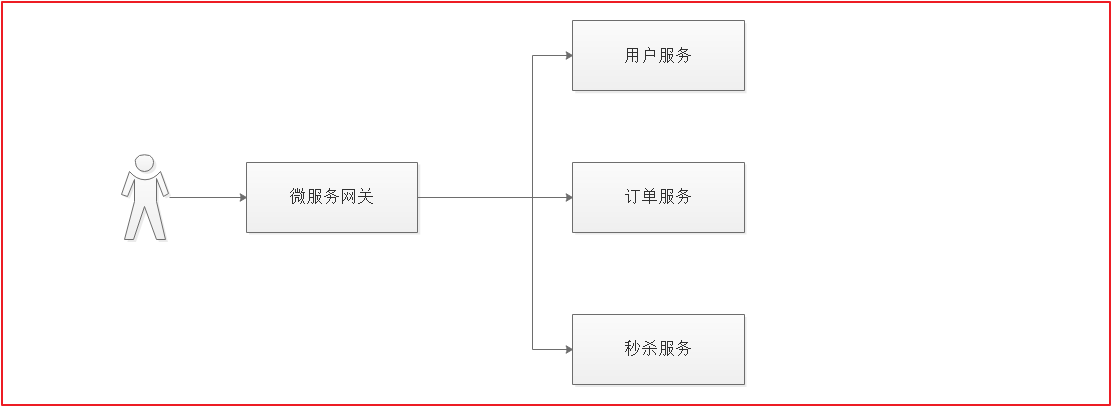
**身份认证**

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。

## 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

## 第三方账号登录

获取第三方账户的信息后保存在自己的数据库中，下次在使用这个第三方账号进行登录时，使用自己数据库的第三方账号信息与当前请求登录的第三方账号返回信息进行比对，看是否符合；

就是使用QQ登录，获取QQ的授权，获取QQ的信息保存在自己的数据库，下次再用QQ登录，用数据库的QQ信息，与当前QQ信息进行比对

### 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。



### 第三方登录优点

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.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.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.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前提的。

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。

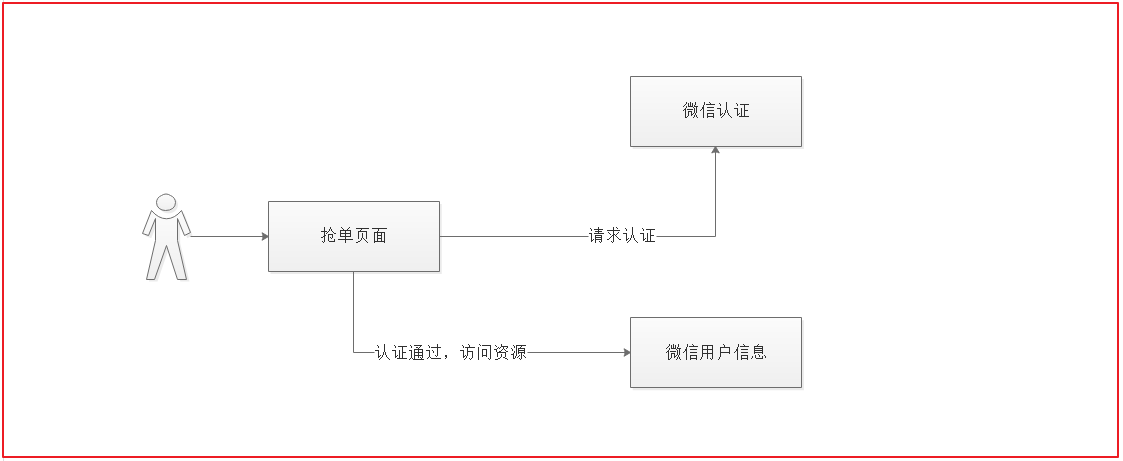
6.对于某些应用来说，使用第三方登录完全可以满足自己的需要，因此不必要设计和开发一套自己的账户体系。

7.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.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性的营销宣传，为产品的市场推广提供另一种渠道。

### 第三方认证

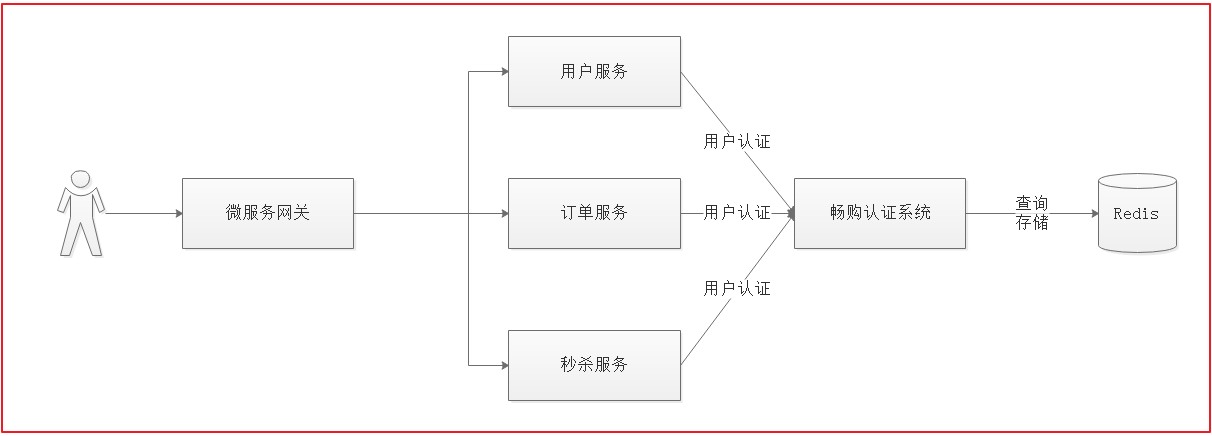
当需要访问第三方系统的资源时需要首先通过第三方系统的认证（例如：微信认证），由第三方系统对用户认证通过，并授权资源的访问权限。



# 认证技术方案

## 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、认证系统为独立的系统。

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、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、Apache Shiro.

2、CAS

3、Spring security CAS

## 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。

### 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。

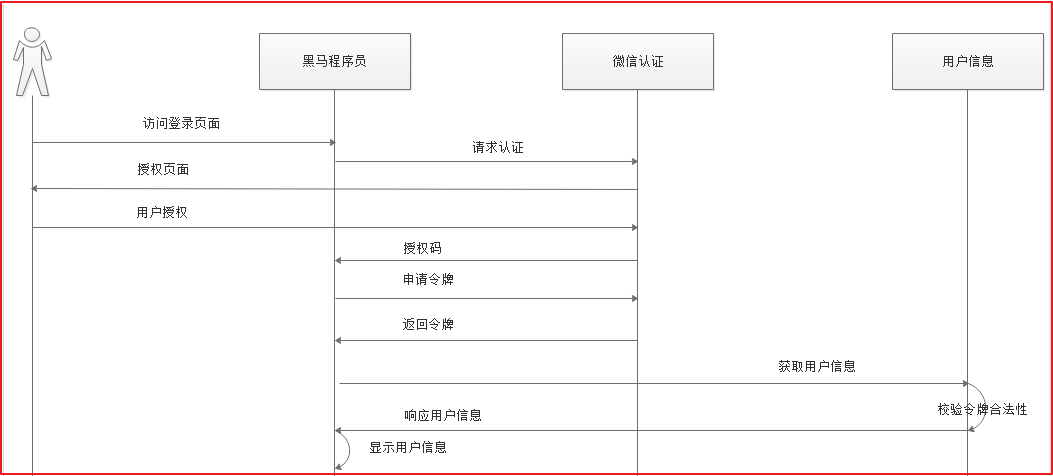
OAUTH协议为用户资源的授权提供了一个安全的、开放而又简易的标准。同时，任何第三方都可以使用OAUTH认证服务，任何服务提供商都可以实现自身的OAUTH认证服务，因而OAUTH是开放的。业界提供了OAUTH的多种实现如PHP、JavaScript，Java，Ruby等各种语言开发包，大大节约了程序员的时间，因而OAUTH是简易的。互联网很多服务如Open API，很多大公司如Google，Yahoo，Microsoft等都提供了OAUTH认证服务，这些都足以说明OAUTH标准逐渐成为开放资源授权的标准。

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。

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。

### 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

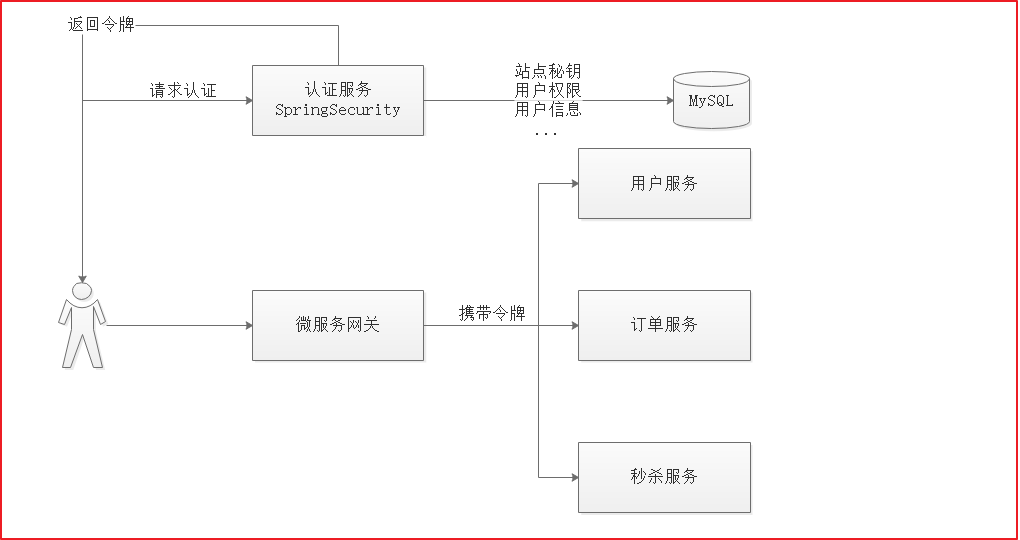
2、外部系统访问畅购的资源

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。

## 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。

# Security Oauth2.0入门

## 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。

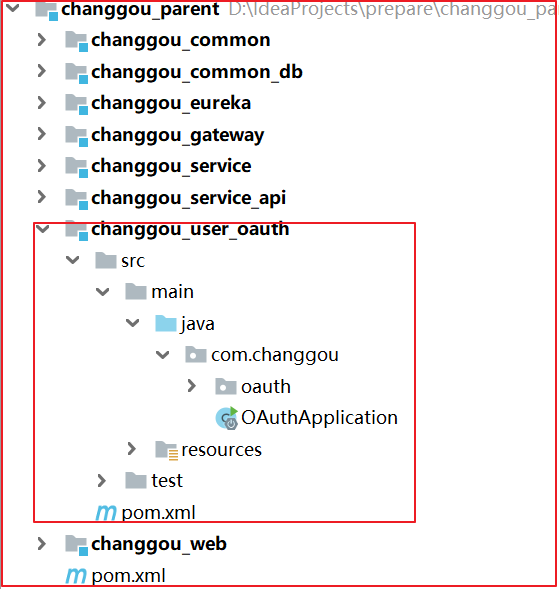
2、理解spring Security Oauth2的工作流程。

3、掌握资源服务集成spring Security框架完成Oauth2认证的流程。

## 搭建认证服务器

### 导入认证工程

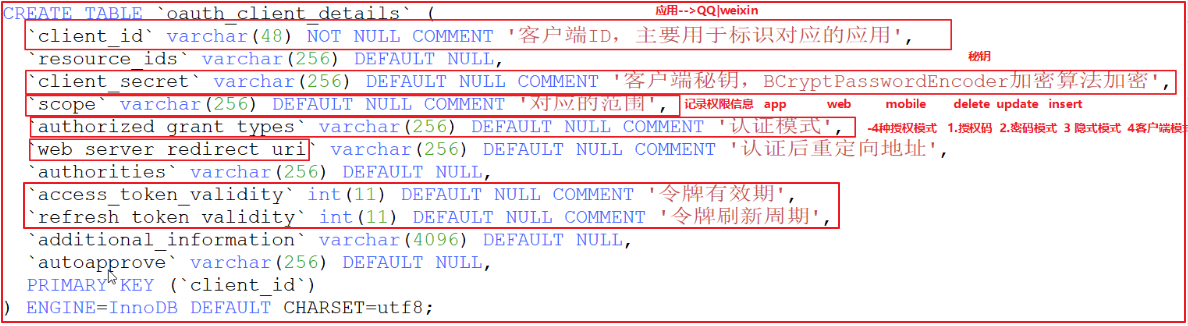
将课件中资料\Oauth2.0\changgou\_user\_oauth的工程导入到项目中去，如下图：



### 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\_oauth?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

### 使用的数据库介绍



### 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。

## Oauth2授权模式

### Oauth2授权模式

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。

### 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

#### 申请授权码

请求认证服务获取授权码：

Get请求：

<http://localhost:9001/oauth/authorize?client_id=changgou&response_type=code&scop=app&redirect_uri=http://localhost>

参数列表如下：

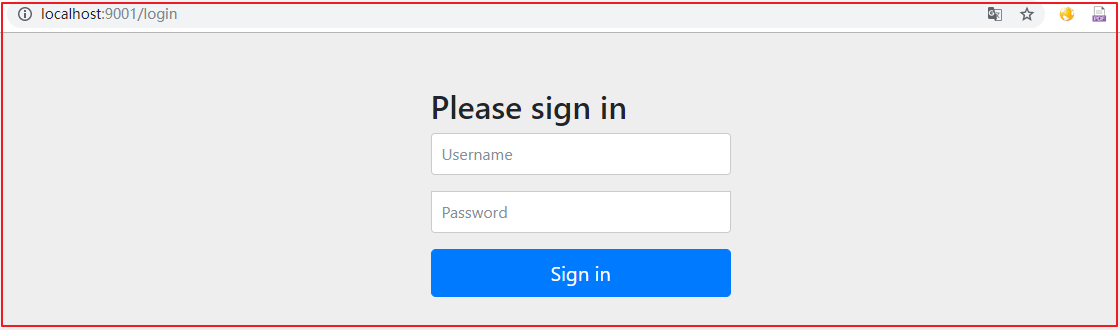
client\_id：客户端id，和授权配置类中设置的客户端id一致。

response\_type：授权码模式固定为code

scop：客户端范围，和授权配置类中设置的scop一致。

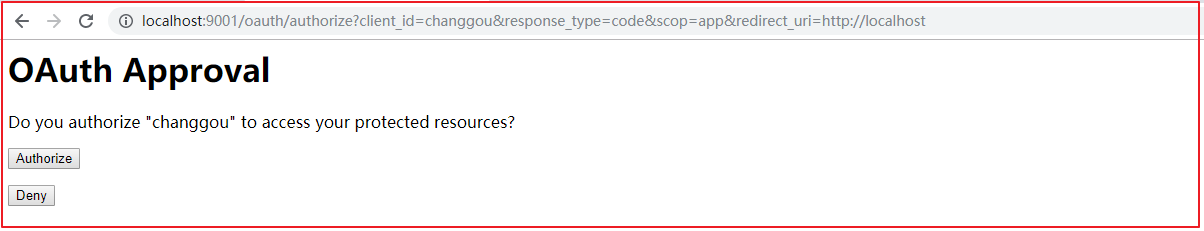
redirect\_uri：跳转uri，当授权码申请成功后会跳转到此地址，并在后边带上code参数（授权码）

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



#### 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：

grant\_type：授权类型，填写authorization\_code，表示授权码模式

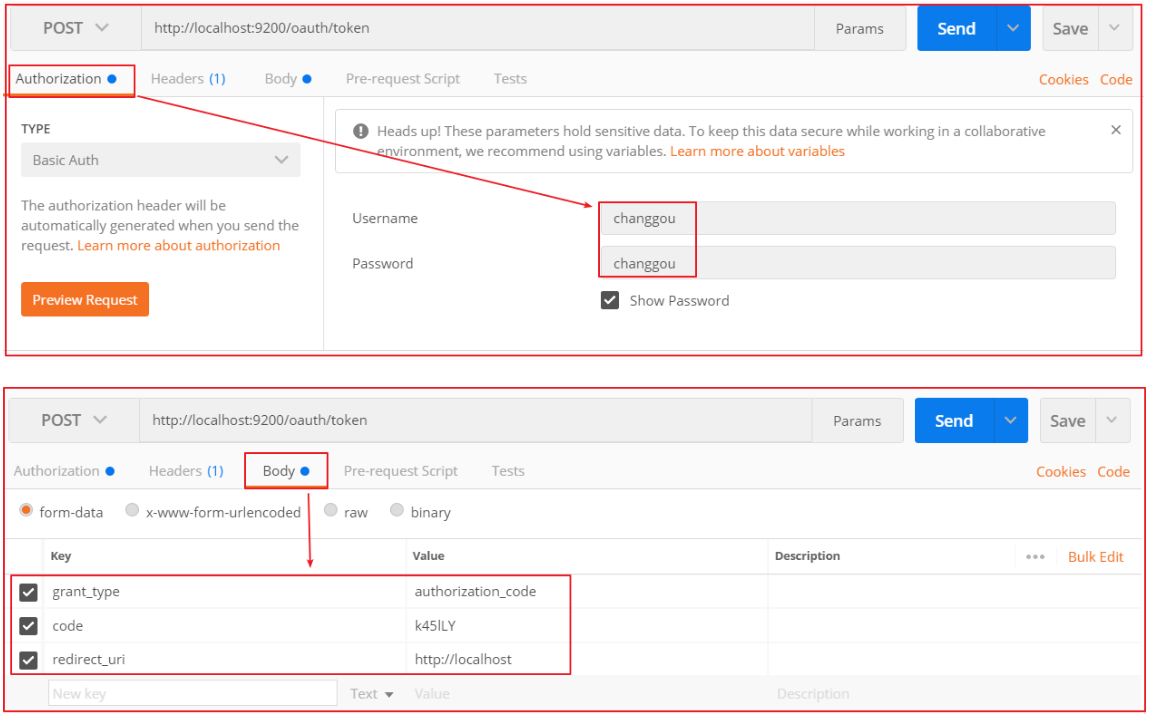
code：授权码，就是刚刚获取的授权码，注意：授权码只使用一次就无效了，需要重新申请。

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。

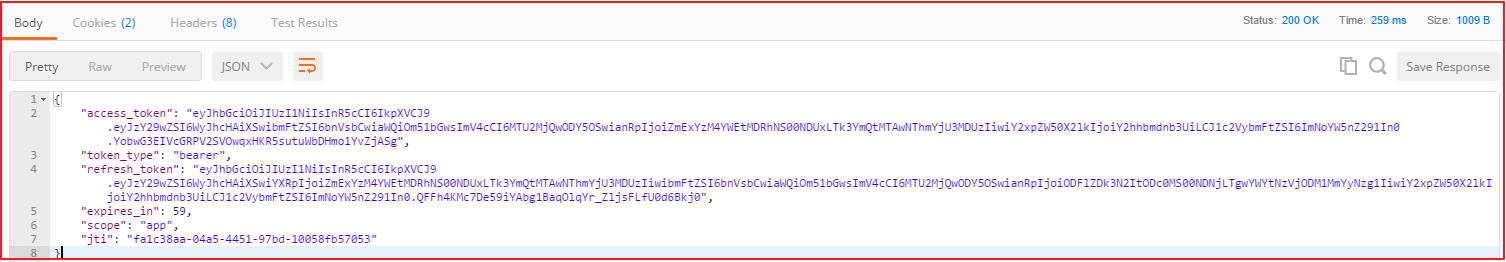
以上测试使用postman完成：

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



返回信息如下:

access\_token：访问令牌，携带此令牌访问资源

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）。

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

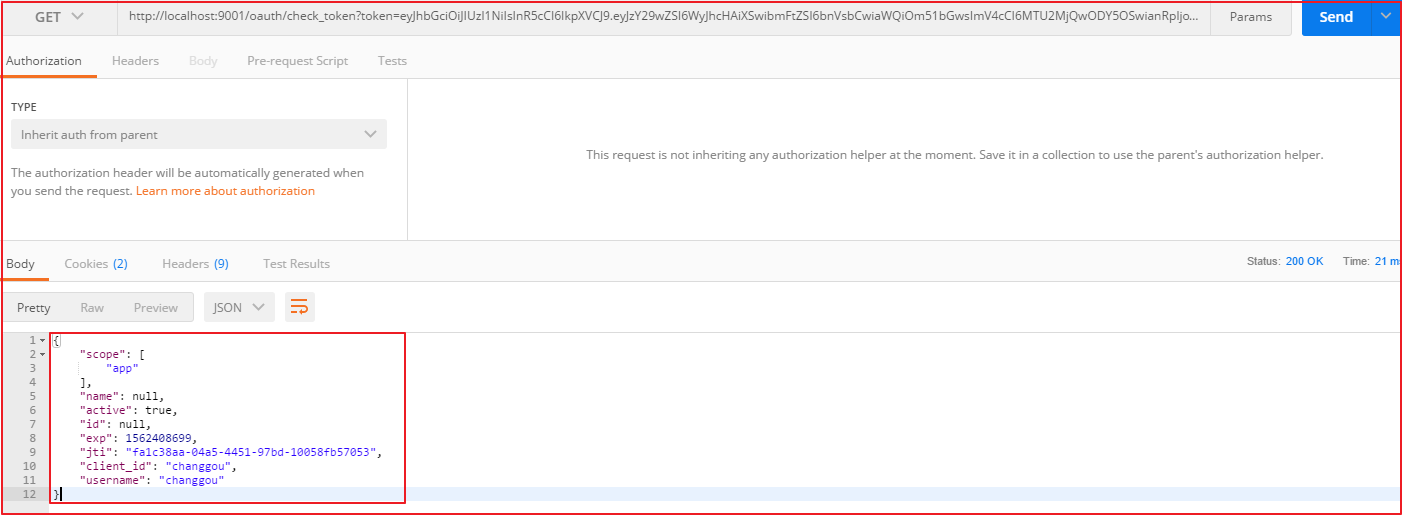
#### 令牌校验

Spring Security Oauth2提供校验令牌的端点，如下：

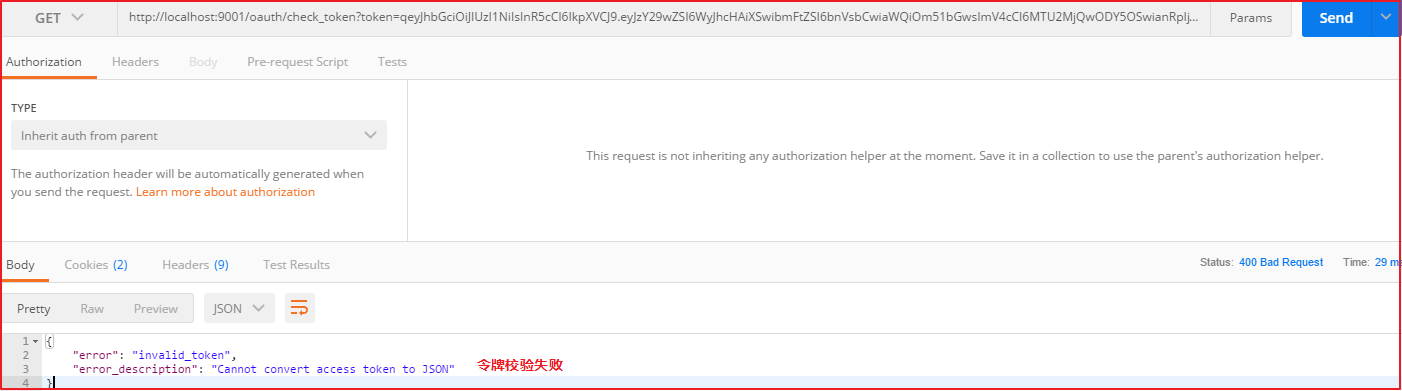
Get: <http://localhost:9001/oauth/check_token?token>= [access\_token]

参数：token：令牌

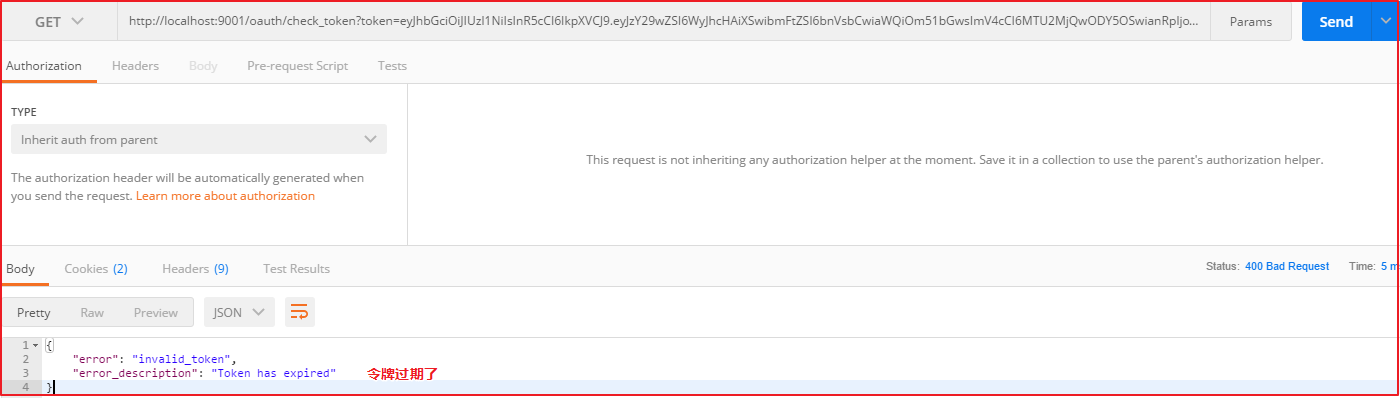
使用postman测试如下:



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：



#### 刷新令牌

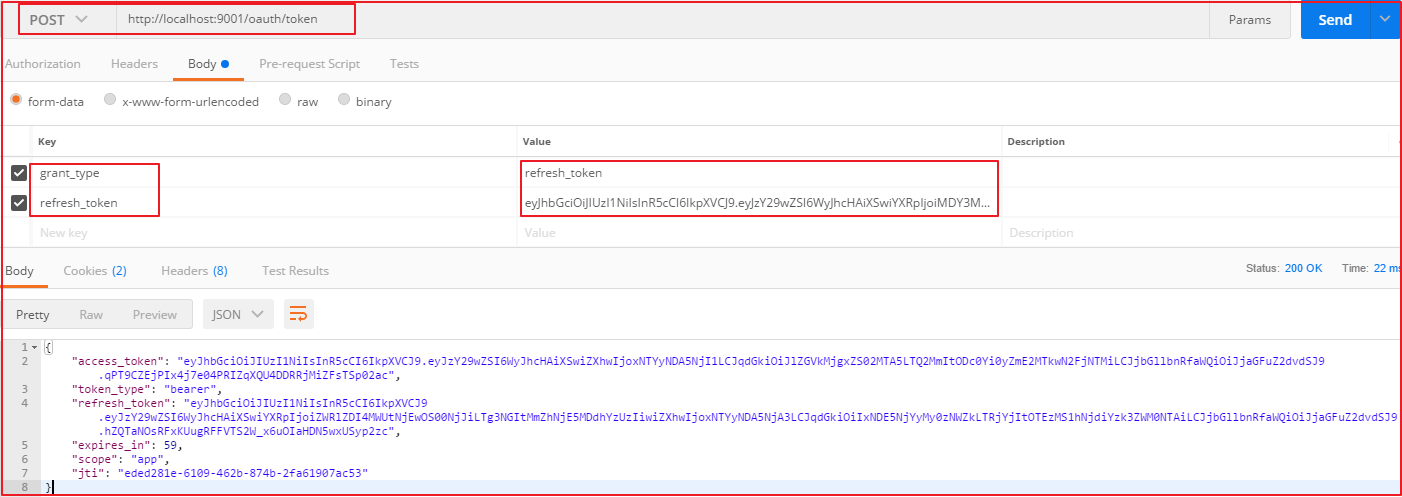
刷新令牌是当令牌快过期时重新生成一个令牌，它于授权码授权和密码授权生成令牌不同，刷新令牌不需要授权码 也不需要账号和密码，只需要一个刷新令牌、客户端id和客户端密码。

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）令牌刷新码



### 密码授权实现

#### 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>

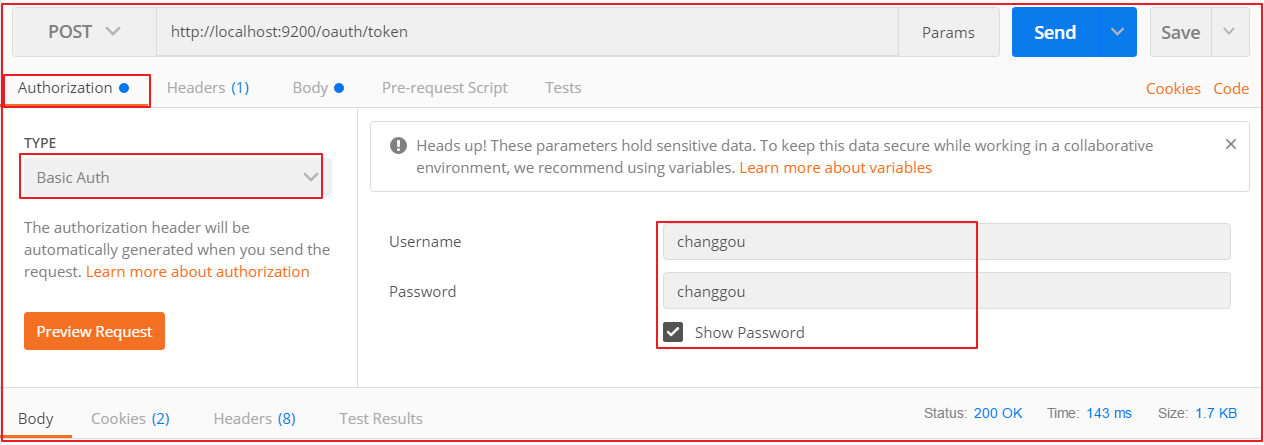
参数：

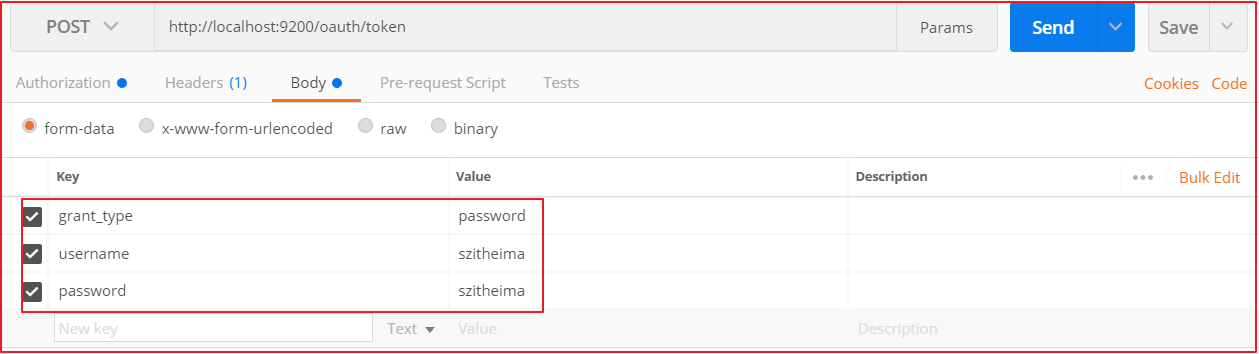
grant\_type：密码模式授权填写password

username：账号

password：密码

并且此链接需要使用 http Basic认证。





测试数据如下：



#### 校验令牌

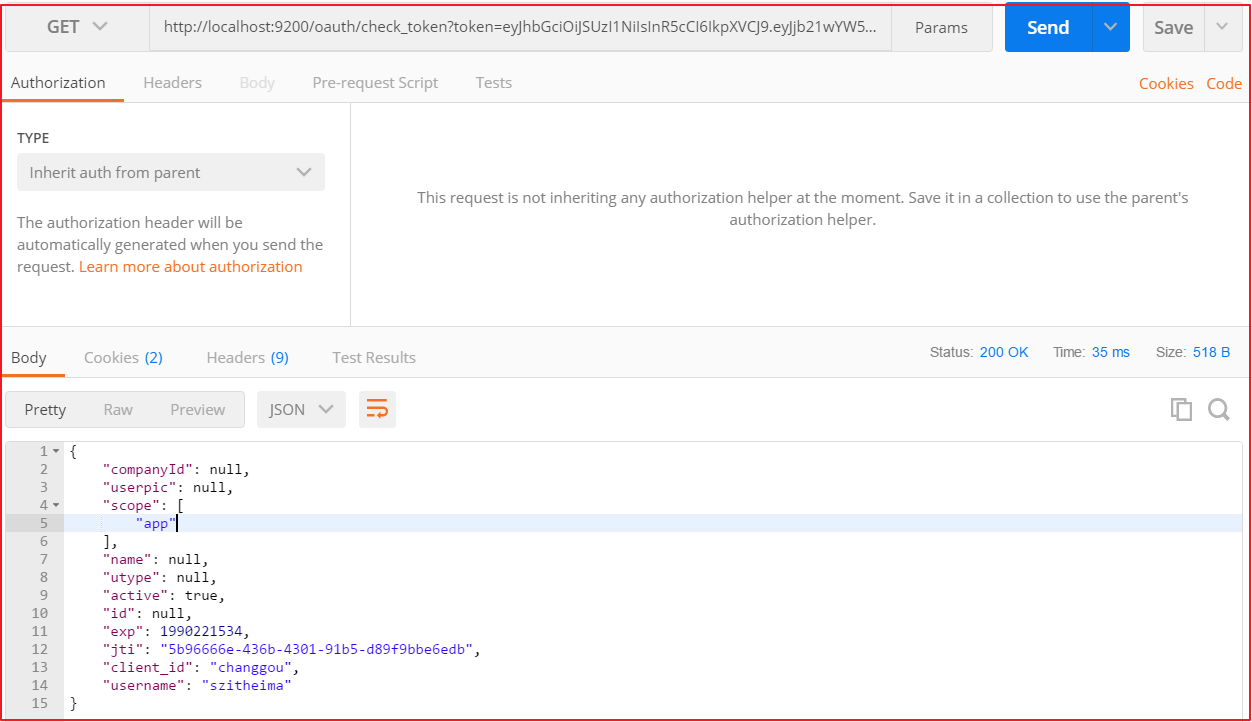
Spring Security Oauth2提供校验令牌的端点，如下：

Get: <http://localhost:9001/oauth/check_token?token>=

参数：

token：令牌

使用postman测试如下:



返回结果：

{

"**companyId**": null,

"**userpic**": null,

"**scope**": [

"app"

],

"**name**": null,

"**utype**": null,

"**active**": true,

"**id**": null,

"**exp**": 1990221534,

"**jti**": "5b96666e-436b-4301-91b5-d89f9bbe6edb",

"**client\_id**": "changgou",

"**username**": "szitheima"

}

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 companyId、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

#### 刷新令牌

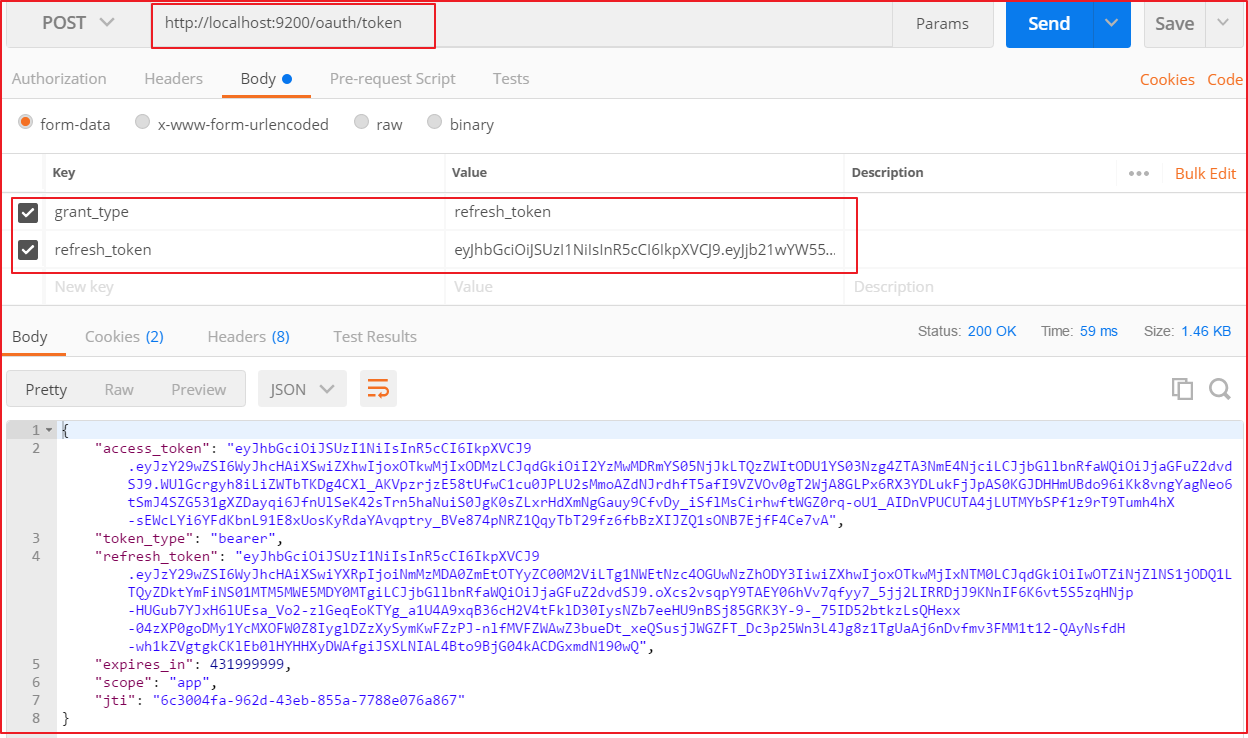
刷新令牌是当令牌快过期时重新生成一个令牌，它于授权码授权和密码授权生成令牌不同，刷新令牌不需要授权码 也不需要账号和密码，只需要一个刷新令牌、客户端id和客户端密码。

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）



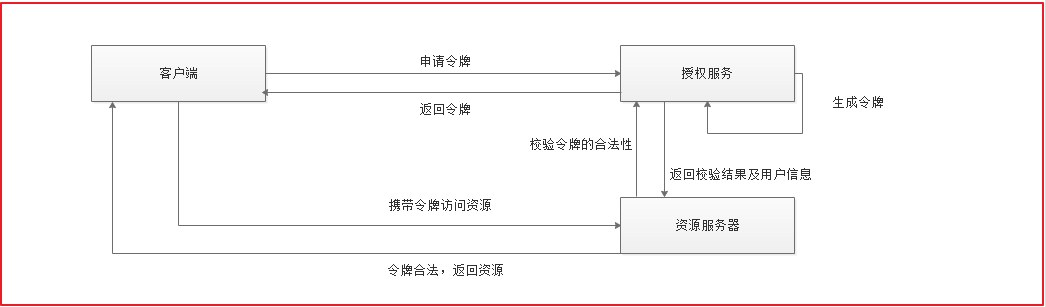
刷新令牌成功，会重新生成新的访问令牌和刷新令牌，令牌的有效期也比旧令牌长。

刷新令牌通常是在令牌快过期时进行刷新 。

# 资源服务授权

## 资源服务授权流程

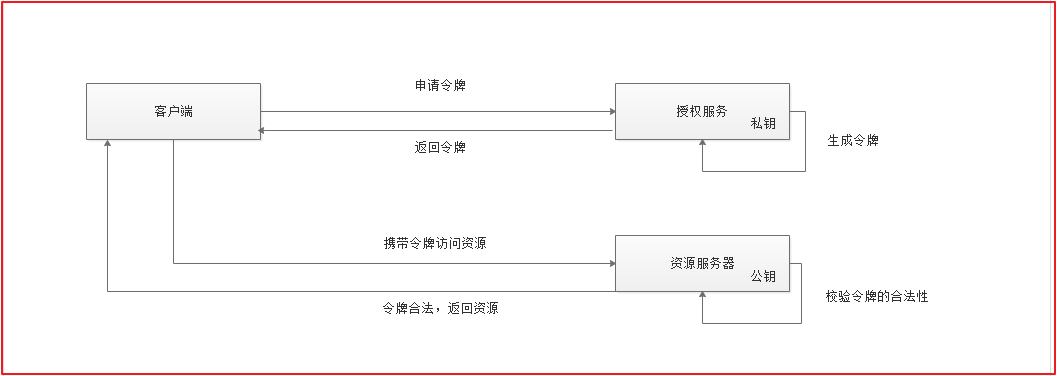
### 传统授权流程



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。

### 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。

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(RSA)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。

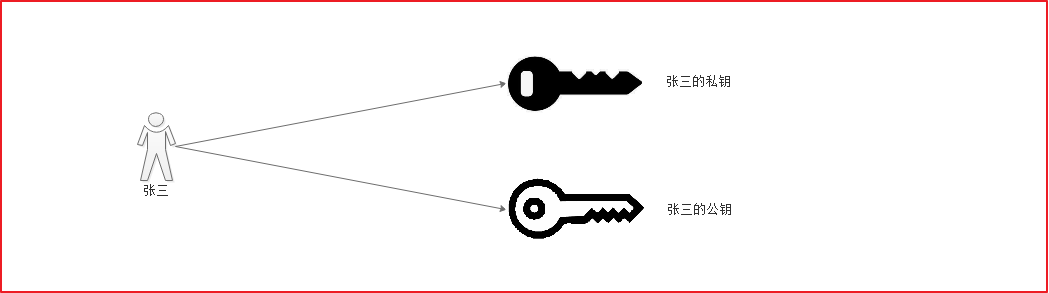
5、令牌有效，资源服务向客户端响应资源信息

## 公钥私钥

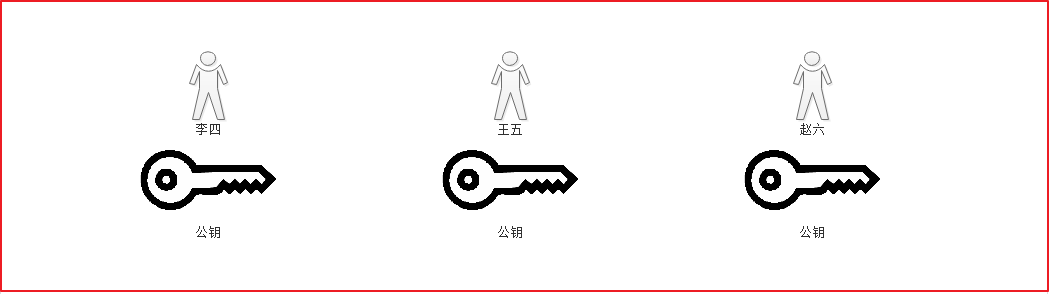
在对称加密的时代，加密和解密用的是同一个密钥，这个密钥既用于加密，又用于解密。这样做有一个明显的缺点，如果两个人之间传输文件，两个人都要知道密钥，如果是三个人呢，五个人呢？于是就产生了非对称加密，用一个密钥进行**加密（公钥）**，用另一个密钥进行**解密（私钥）**。

### 公钥私钥原理

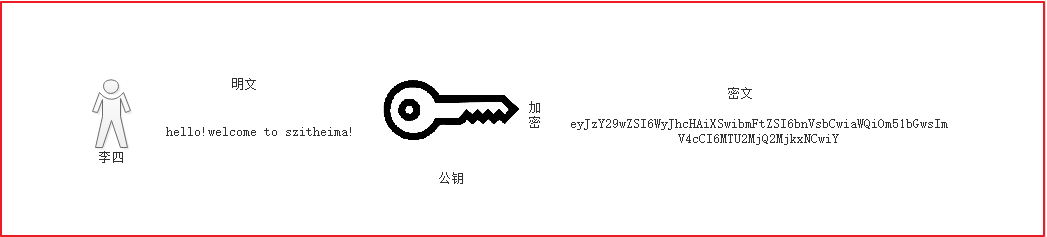
张三有两把钥匙，一把是公钥，另一把是私钥。



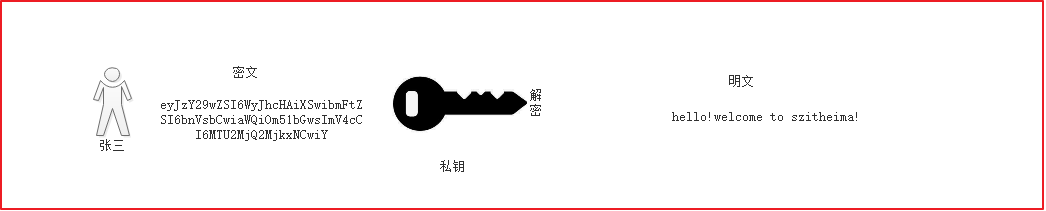
张三把公钥送给他的朋友们—-李四、王五、赵六—-每人一把。



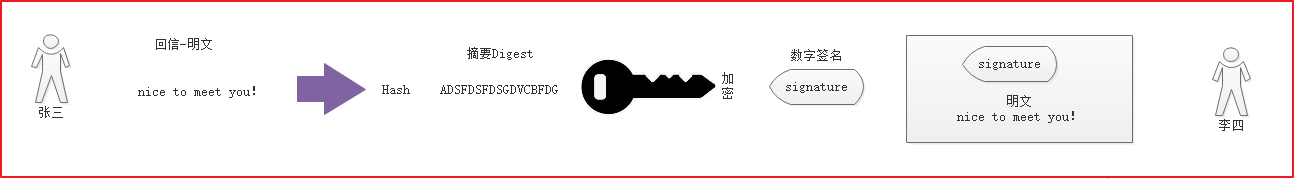
李四要给张三写一封保密的信。她写完后用张三的公钥加密，就可以达到保密的效果。



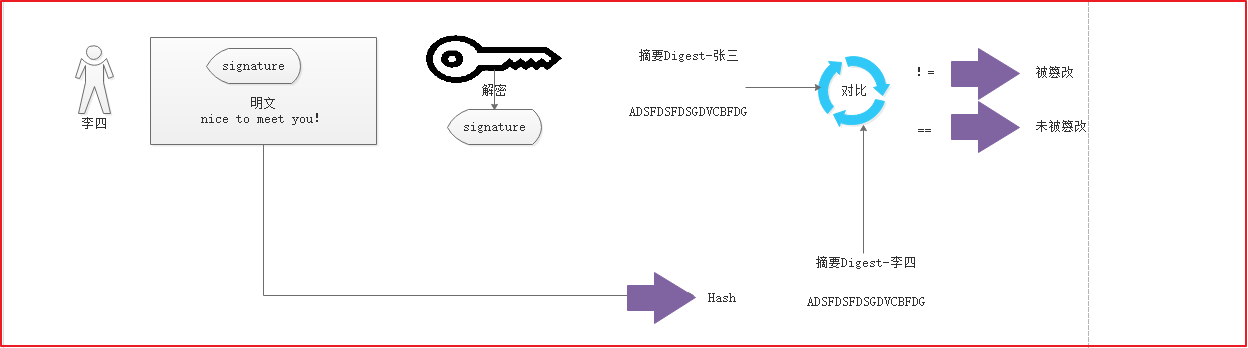
张三收信后，用私钥解密，就看到了信件内容。这里要强调的是，只要张三的私钥不泄露，这封信就是安全的，即使落在别人手里，也无法解密。



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，用张三的公钥解密，得到信件的摘要。由此证明，这封信确实是张三发出的。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。



### 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：

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，xc.keystore保存了生成的证书

-storepass：密钥库的访问密码

(2)查询证书信息

keytool -list -keystore changgou.jks

(3)删除别名

keytool -delete -alias changgou -keystore changgou.jsk

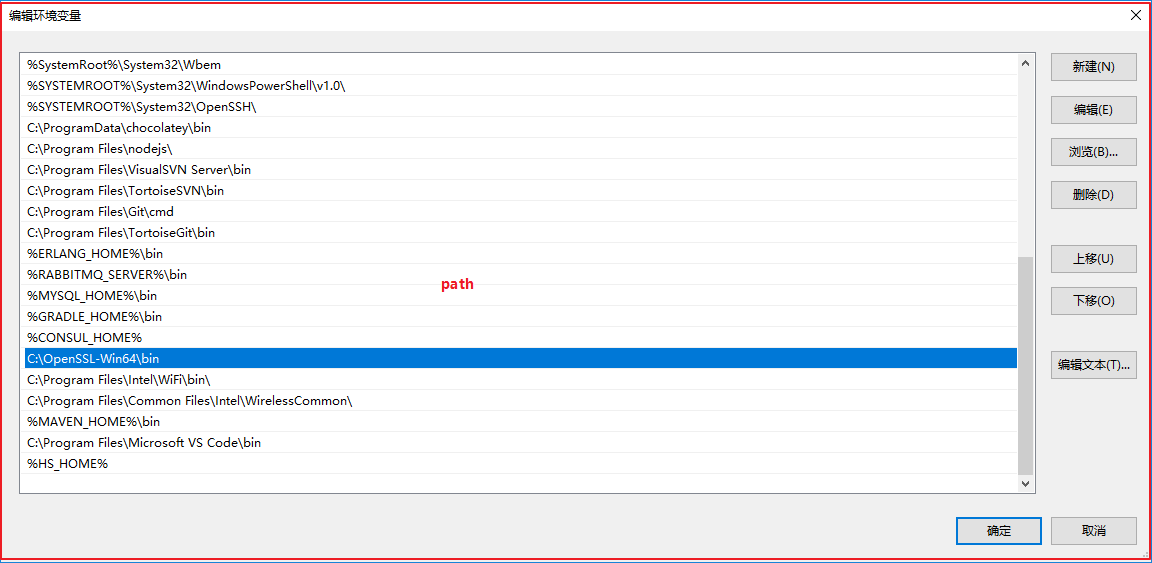
### 导出公钥

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

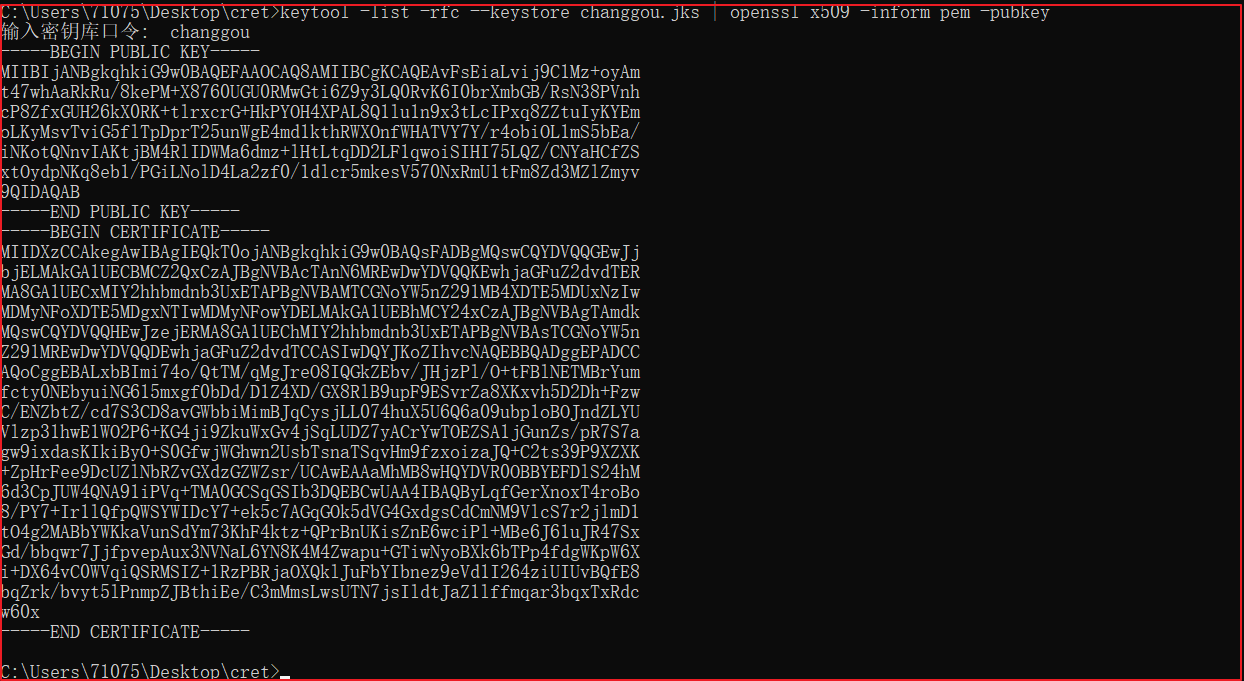
配置openssl的path环境变量，如下图：



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：

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



下面段内容是公钥

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvFsEiaLvij9C1Mz+oyAm

t47whAaRkRu/8kePM+X8760UGU0RMwGti6Z9y3LQ0RvK6I0brXmbGB/RsN38PVnh

cP8ZfxGUH26kX0RK+tlrxcrG+HkPYOH4XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEm

oLKyMsvTviG5flTpDprT25unWgE4md1kthRWXOnfWHATVY7Y/r4obiOL1mS5bEa/

iNKotQNnvIAKtjBM4RlIDWMa6dmz+lHtLtqDD2LF1qwoiSIHI75LQZ/CNYaHCfZS

xtOydpNKq8eb1/PGiLNolD4La2zf0/1dlcr5mkesV570NxRmU1tFm8Zd3MZlZmyv

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。

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

### JWT令牌

#### 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：

*/\*\*\*\*\*  
 \** ***@Author:*** *Steven  
 \** ***@Date:*** *2019/7/7 13:42  
 \** ***@Description:*** *com.changgou.token  
 \* 创建JWT令牌，使用私钥加密  
 \*\*\*\*/***public class** CreateJwtTest {  
  
 */\*\*\*  
 \* 创建令牌测试  
 \*/* @Test  
 **public void** testCreateToken(){  
 *//证书文件路径* String key\_location=**"changgou.jks"**;  
 *//秘钥库密码* String key\_password=**"changgou"**;  
 *//秘钥密码* String keypwd = **"changgou"**;  
 *//秘钥别名* String alias = **"changgou"**;  
  
 *//访问证书路径* ClassPathResource resource = **new** ClassPathResource(key\_location);  
  
 *//创建秘钥工厂* KeyStoreKeyFactory keyStoreKeyFactory = **new** KeyStoreKeyFactory(resource,key\_password.toCharArray());  
  
 *//读取秘钥对(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-使用私钥* Jwt jwt = JwtHelper.*encode*(JSON.*toJSONString*(tokenMap), **new** RsaSigner(rsaPrivate));  
  
 *//取出令牌* String encoded = jwt.getEncoded();  
 System.***out***.println(encoded);  
 }  
}

运行后的结果如下：

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlcyI6IlJPTEVfVklQLFJPTEVfVVNFUiIsIm5hbWUiOiJpdGhlaW1hIiwiaWQiOiIxIn0.IR9Qu9ZqYZ2gU2qgAziyT38UhEeL4Oi69ko-dzC\_P9-Vjz40hwZDqxl8wZ-W2WAw1eWGIHV1EYDjg0-eilogJZ5UikyWw1bewXCpvlM-ZRtYQQqHFTlfDiVcFetyTayaskwa-x\_BVS4pTWAskiaIKbKR4KcME2E5o1rEek-3YPkqAiZ6WP1UOmpaCJDaaFSdninqG0gzSCuGvLuG40x0Ngpfk7mPOecsIi5cbJElpdYUsCr9oXc53ROyfvYpHjzV7c2D5eIZu3leUPXRvvVAPJFEcSBiisxUSEeiGpmuQhaFZd1g-yJ1WQrixFvehMeLX2XU6W1nlL5ARTpQf\_Jjiw

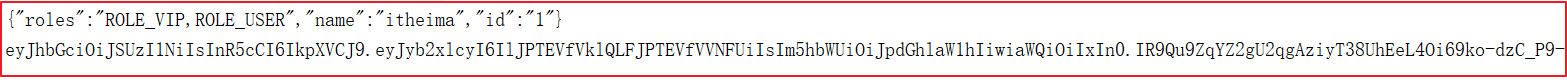
#### 解析令牌

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。

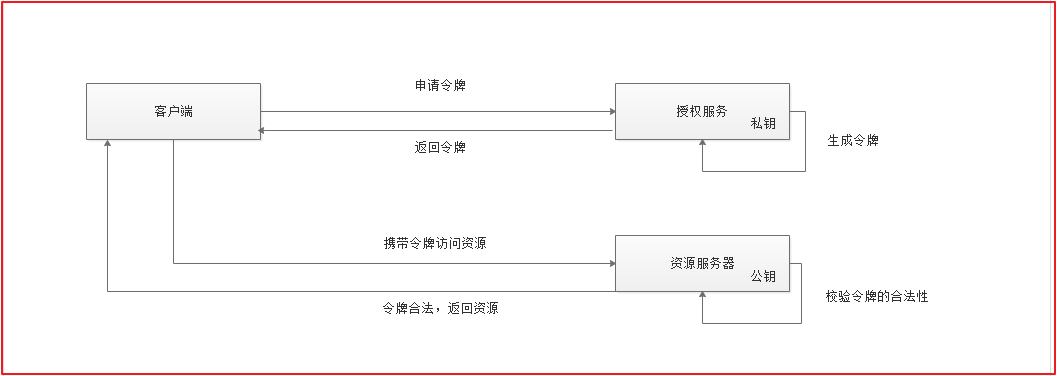
在changgou-user-oauth创建测试类com.changgou.token.ParseJwtTest实现解析校验令牌数据，代码如下：

*/\*\*\*\*\*  
 \** ***@Author:*** *Steven  
 \** ***@Date:*** *2019/7/7 13:48  
 \** ***@Description:*** *com.changgou.token  
 \* 使用公钥解密令牌数据  
 \*\*\*\*/***public class** ParseJwtTest {  
  
 */\*\*\*  
 \* 校验令牌  
 \*/* @Test  
 **public void** testParseToken(){  
 *//令牌* String token = **"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlcyI6IlJPTEVfVklQLFJPTEVfVVNFUiIsIm5hbWUiOiJpdGhlaW1hIiwiaWQiOiIxIn0.IR9Qu9ZqYZ2gU2qgAziyT38UhEeL4Oi69ko-dzC\_P9-Vjz40hwZDqxl8wZ-W2WAw1eWGIHV1EYDjg0-eilogJZ5UikyWw1bewXCpvlM-ZRtYQQqHFTlfDiVcFetyTayaskwa-x\_BVS4pTWAskiaIKbKR4KcME2E5o1rEek-3YPkqAiZ6WP1UOmpaCJDaaFSdninqG0gzSCuGvLuG40x0Ngpfk7mPOecsIi5cbJElpdYUsCr9oXc53ROyfvYpHjzV7c2D5eIZu3leUPXRvvVAPJFEcSBiisxUSEeiGpmuQhaFZd1g-yJ1WQrixFvehMeLX2XU6W1nlL5ARTpQf\_Jjiw"**;  
  
 *//公钥* String publickey = **"-----BEGIN PUBLIC KEY-----MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvFsEiaLvij9C1Mz+oyAmt47whAaRkRu/8kePM+X8760UGU0RMwGti6Z9y3LQ0RvK6I0brXmbGB/RsN38PVnhcP8ZfxGUH26kX0RK+tlrxcrG+HkPYOH4XPAL8Q1lu1n9x3tLcIPxq8ZZtuIyKYEmoLKyMsvTviG5flTpDprT25unWgE4md1kthRWXOnfWHATVY7Y/r4obiOL1mS5bEa/iNKotQNnvIAKtjBM4RlIDWMa6dmz+lHtLtqDD2LF1qwoiSIHI75LQZ/CNYaHCfZSxtOydpNKq8eb1/PGiLNolD4La2zf0/1dlcr5mkesV570NxRmU1tFm8Zd3MZlZmyv9QIDAQAB-----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);  
 }  
}

运行后的结果如下：



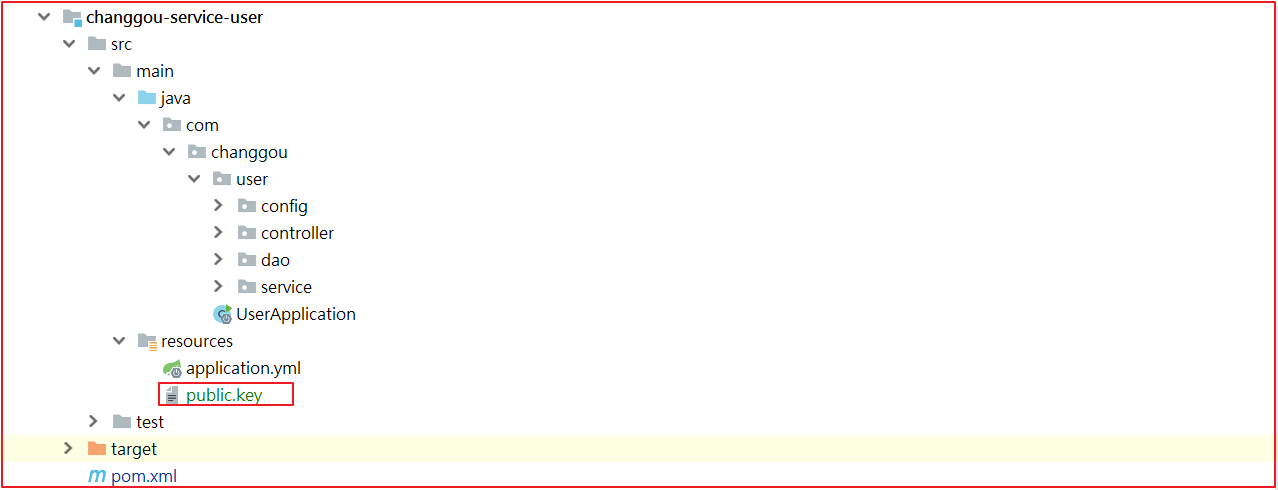
## 资源服务授权配置



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。

### 用户微服务资源授权

**(1)将上面生成的公钥public.key拷贝到changgou-service-user微服务工程的resources目录下，如下图：**



**(2)在changgou-service-user微服务工程pom.xml中引入oauth依赖**

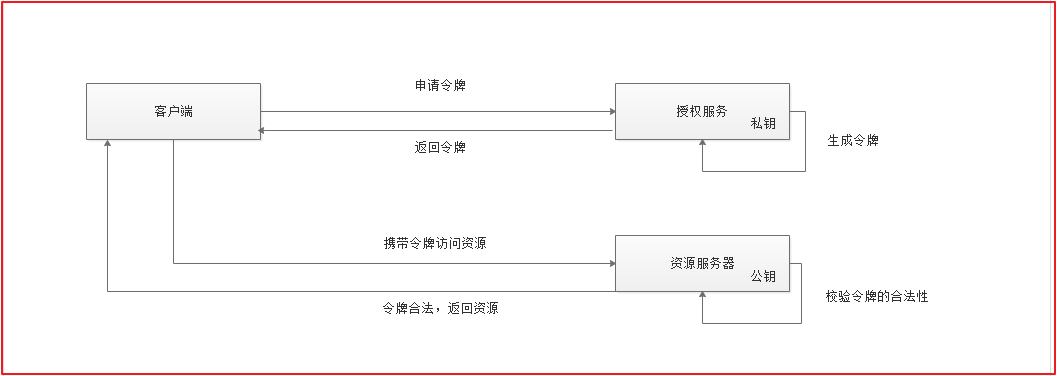
<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-oauth2</**artifactId**>  
</**dependency**>

**(3)资源授权配置**

在changgou-service-user工程中创建com.changgou.user.config.ResourceServerConfig，代码如下：

*/\*\*  
 \* 实现信息校验,这个类对每个访问微服务的请求进行检验  
 \* 1、获取公钥  
 \* 2、使用公钥作为校验密钥  
 \* 3、校验令牌  
 \*/*@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  
 \** ***@return*** *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* }  
}

### 授权测试

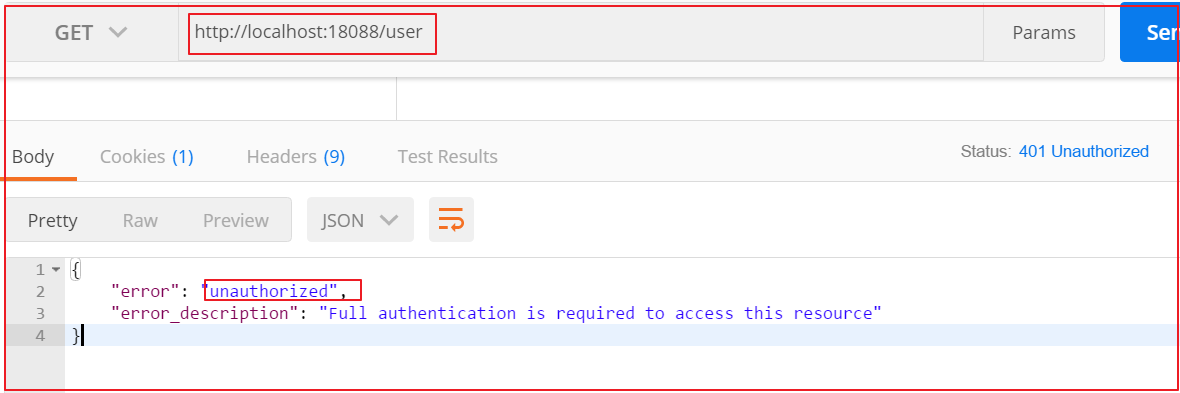


用户每次访问微服务的时候，需要先申请令牌，令牌申请后，每次将令牌放到头文件中，才能访问微服务。

头文件中每次需要添加一个Authorization头信息，头的结果为bearer token。

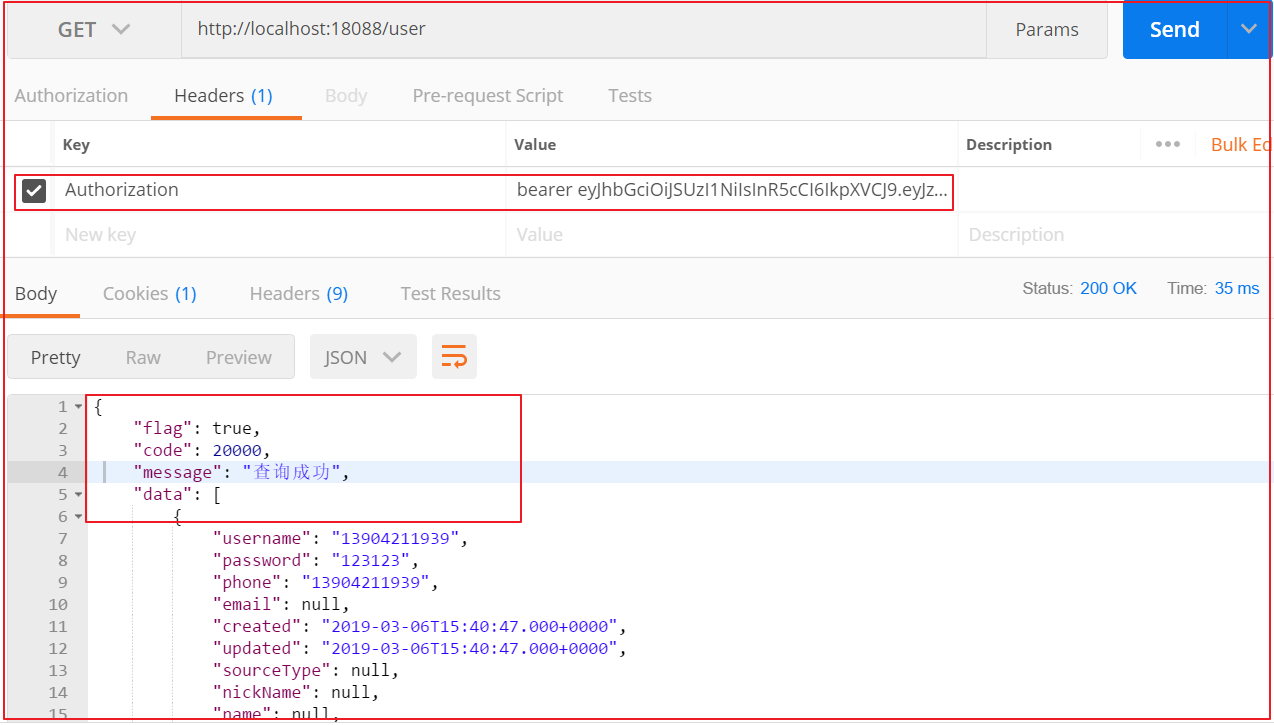
**(1)不携带令牌测试**

访问http://localhost:18088/user 不携带令牌，结果如下：



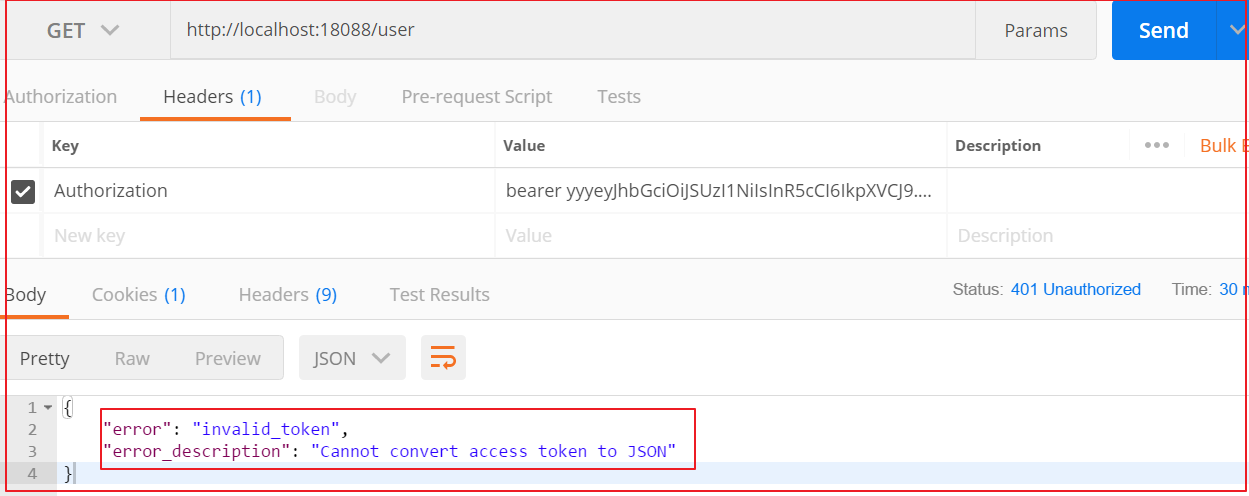
**(2)携带正确令牌访问**

访问<http://localhost:18088/user> 携带正确令牌，结果如下：



**(3)携带错误令牌**

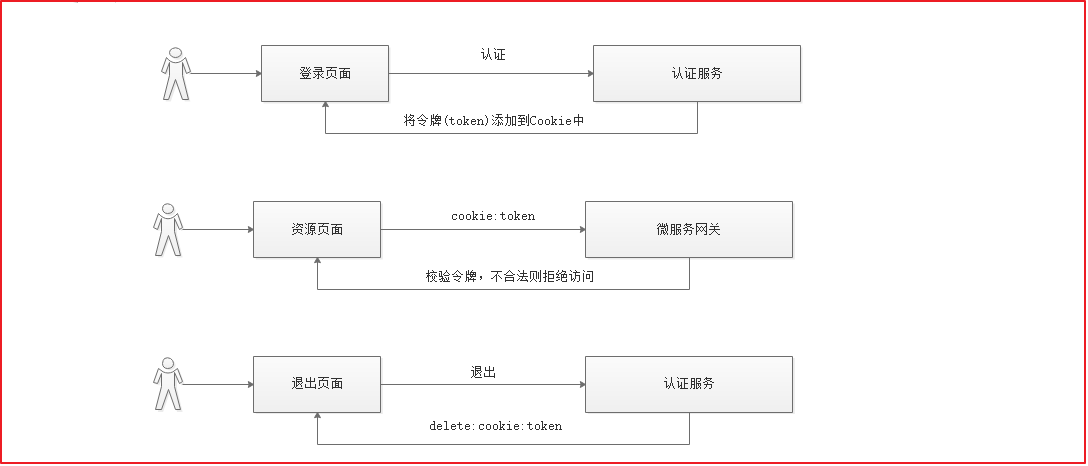
访问<http://localhost:18088/user> 携带不正确令牌，结果如下：



# 认证开发

## 需求分析：登录页面进行认证授权

用户登录的流程图如下：



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cookie

3、用户访问资源页面，带着cookie到网关

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、用户退出，请求认证服务，删除cookie中的token

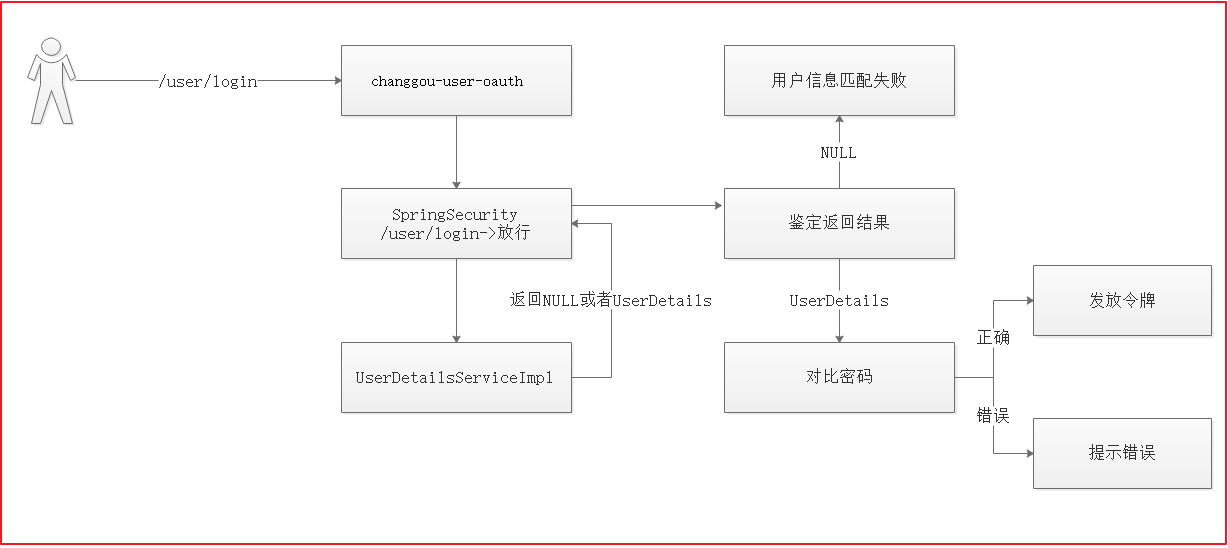
## 认证服务

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。

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。



### 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：

**public** **class** **AuthToken** **implements** **Serializable**{

//令牌信息

String accessToken;

//刷新token(refresh\_token)

String refreshToken;

//jwt短令牌

String jti;

//...get...set

}

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

**public** **class** **CookieUtil** {

/\*\*

\* 设置cookie

\*

\* @param response

\* @param name cookie名字

\* @param value cookie值

\* @param maxAge cookie生命周期 以秒为单位

\*/

**public** **static** **void** **addCookie**(HttpServletResponse response, String domain, String path, String name,

String value, **int** maxAge, **boolean** httpOnly) {

Cookie cookie = **new** Cookie(name, value);

cookie.setDomain(domain);

cookie.setPath(path);

cookie.setMaxAge(maxAge);

cookie.setHttpOnly(httpOnly);

response.addCookie(cookie);

}

/\*\*

\* 根据cookie名称读取cookie

\* @param request

\* @return map<cookieName,cookieValue>

\*/

**public** **static** Map<String,String> readCookie(HttpServletRequest request, String ... cookieNames) {

Map<String,String> cookieMap = **new** HashMap<String,String>();

Cookie[] cookies = request.getCookies();

**if** (cookies != **null**) {

**for** (Cookie cookie : cookies) {

String cookieName = cookie.getName();

String cookieValue = cookie.getValue();

**for**(**int** i=0;i<cookieNames.length;i++){

**if**(cookieNames[i].equals(cookieName)){

cookieMap.put(cookieName,cookieValue);

}

}

}

}

**return** cookieMap;

}

}

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：

**public** **class** **UserJwt** **extends** **User** {

**private** String id; //用户ID

**private** String name; //用户名字

**public** **UserJwt**(String username, String password, Collection<? extends GrantedAuthority> authorities) {

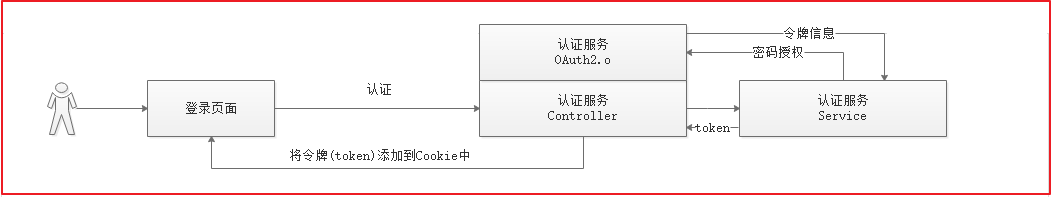
**super**(username, password, authorities);

}

//...get...set

}

### 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。

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

*/\*\*  
 \* 认证服务  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.oauth.service  
 \*/***public interface** LoginService {  
 */\*\*  
 \* 授权认证方法  
 \** ***@param username*** *用户名  
 \** ***@param password*** *密码  
 \** ***@param clientId*** *客户端id(密钥的别名)  
 \** ***@param clientSecret*** *客户端密钥(访问密码)  
 \** ***@return*** *\*/* AuthToken login(String username, String password, String clientId, String clientSecret);  
}

创建com.changgou.oauth.service.impl.LoginServiceImpl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：

**思路：**

//1.选中认证服务的地址-serviceInstance = loadBalancerClient.choose("user-auth")

//2.如果服务器实例为空，返回找不到服务异常信息

//3.拼接令牌的请求地址-url = serviceInstance.getUri().toString() + "/oauth/token"

//4.定义body = new LinkedMultiValueMap<String, String>()

//4.1 设置授权方式-body.add(名字，"password")

//4.2 设置账号

//4.3 设置密码

//5.定义响应头信息-headers = new LinkedMultiValueMap<String, String>()

//5.1 使用base64编码拼接 [Basic 客户端id:客户端密钥]-Base64Utils.encode

//5.2 设置头信息-Authorization

//6.使用restTemplate请求spring security的申请令牌接口-restTemplate.exchange(url,请求方式,参数,返回类型)

//7.获取响应数据-responseEntity.getBody()

//8.将响应数据封装成AuthToken对象

**代码实现：**

*/\*\*  
 \* 认证实现  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.oauth.service.impl  
 \*/*@Service  
**public class** LoginServiceImpl **implements** LoginService {  
 @Autowired  
 **private** LoadBalancerClient **loadBalancerClient**;  
 @Autowired  
 **private** RestTemplate **restTemplate**;  
 @Override  
 **public** AuthToken login(String username, String password, String clientId, String clientSecret) {  
 *//1.选中认证服务的地址-serviceInstance = loadBalancerClient.choose("user-auth")* ServiceInstance serviceInstance = **loadBalancerClient**.choose(**"user-auth"**);  
 *//2.如果服务器实例为空，返回找不到服务异常信息* **if**(serviceInstance == **null**){  
 **throw new** RuntimeException(**"找不到相应的服务!"**);  
 }  
 *//3.拼接令牌的请求地址-url = serviceInstance.getUri().toString() + "/oauth/token"* String url = serviceInstance.getUri().toString() + **"/oauth/token"**;  
 *//4.定义body = new LinkedMultiValueMap<String, String>()* MultiValueMap<String, String> body = **new** LinkedMultiValueMap<String, String>();  
 *//4.1 设置授权方式-body.add(名字，"password")* body.add(**"grant\_type"**, **"password"**);  
 *//4.2 设置账号* body.add(**"username"**, username);  
 *//4.3 设置密码* body.add(**"password"**, password);  
 *//5.定义响应头信息-heads = new LinkedMultiValueMap<String, String>();* MultiValueMap<String, String> headers = **new** LinkedMultiValueMap<String, String>();  
 *//5.1 使用base64编码拼接 [Basic 客户端id:客户端密钥]-Base64Utils.encode* String AuthorizationStr = clientId + **":"** + clientSecret;  
 *//转64编码* **byte**[] encode = Base64Utils.*encode*(AuthorizationStr.getBytes());  
 AuthorizationStr = **"Basic "** + **new** String(encode);  
 *//5.2 设置头信息-Authorization* headers.add(**"Authorization"**, AuthorizationStr);  
 *//6.使用restTemplate请求spring security的申请令牌接口-restTemplate.exchange(urt,请求方式,参数,返回类型)* ResponseEntity<Map> responseEntity = **restTemplate**.exchange(url, HttpMethod.***POST***, **new** HttpEntity<>(body, headers), Map.**class**);  
 *//7.获取响应数据-responseEntity.getBody()* Map map = responseEntity.getBody();  
 *//8.将响应数据封装成AuthToken对象* AuthToken authToken = **new** AuthToken();  
 *//访问令牌(jwt)* String accessToken = (String) map.get(**"access\_token"**);  
 *//刷新令牌(jwt)* String refreshToken = (String) map.get(**"refresh\_token"**);  
 *//jti，作为用户的身份标识* String jwtToken= (String) map.get(**"jti"**);  
 authToken.setJti(jwtToken);  
 authToken.setAccessToken(accessToken);  
 authToken.setRefreshToken(refreshToken);  
 **return** authToken;  
 }  
}

### 控制层

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

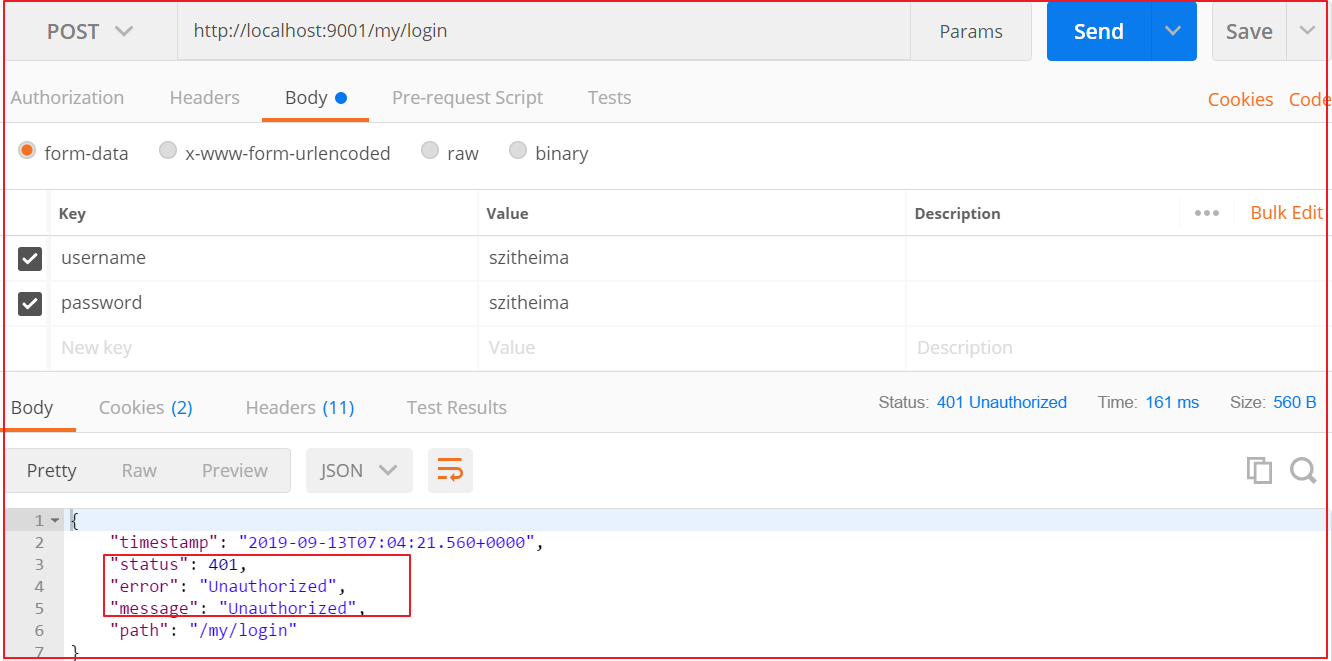
*/\*\*\*\*\*  
 \** ***@Author:*** *Steven  
 \** ***@Description:*** *com.changgou.oauth.controller  
 \*\*\*\*/*@RestController  
@RequestMapping(value = **"my"**)  
**public class** LoginController {  
  
 *//客户端ID* @Value(**"${auth.clientId}"**)  
 **private** String **clientId**;  
  
 *//秘钥* @Value(**"${auth.clientSecret}"**)  
 **private** String **clientSecret**;  
  
 *//Cookie存储的域名* @Value(**"${auth.cookieDomain}"**)  
 **private** String **cookieDomain**;  
  
 *//Cookie生命周期* @Value(**"${auth.cookieMaxAge}"**)  
 **private int cookieMaxAge**;  
  
 @Autowired  
 **private** LoginService **loginService**;  
  
 @PostMapping(**"login"**)  
 **public** Result<AuthToken> login(String username, String password) {  
 **if**(StringUtils.*isEmpty*(username)){  
 **throw new** RuntimeException(**"用户名不允许为空"**);  
 }  
 **if**(StringUtils.*isEmpty*(password)){  
 **throw new** RuntimeException(**"密码不允许为空"**);  
 }  
 *//申请令牌* AuthToken authToken = **loginService**.login(username,password,**clientId**,**clientSecret**);  
  
 *//用户身份令牌* String access\_token = authToken.getAccessToken();  
 *//将令牌存储到cookie* saveCookie(access\_token);  
  
 **return new** Result<AuthToken>(**true**, StatusCode.***OK***,**"登录成功！"**,authToken);  
 }  
 */\*\*\*  
 \* 将令牌存储到cookie  
 \** ***@param token*** *\*/* **private void** saveCookie(String token){  
 HttpServletResponse response = ((ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*()).getResponse();  
 CookieUtil.*addCookie*(response,**cookieDomain**,**"/"**,**"Authorization"**,token,**cookieMaxAge**,**false**);  
 }  
}

### 测试认证接口

#### 直接访问报错

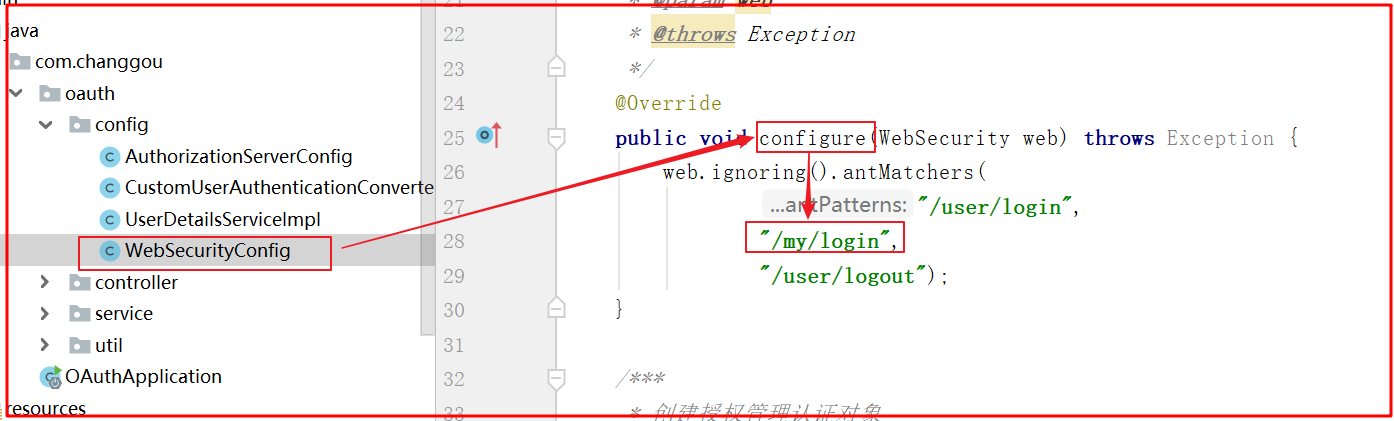
使用postman测试：

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

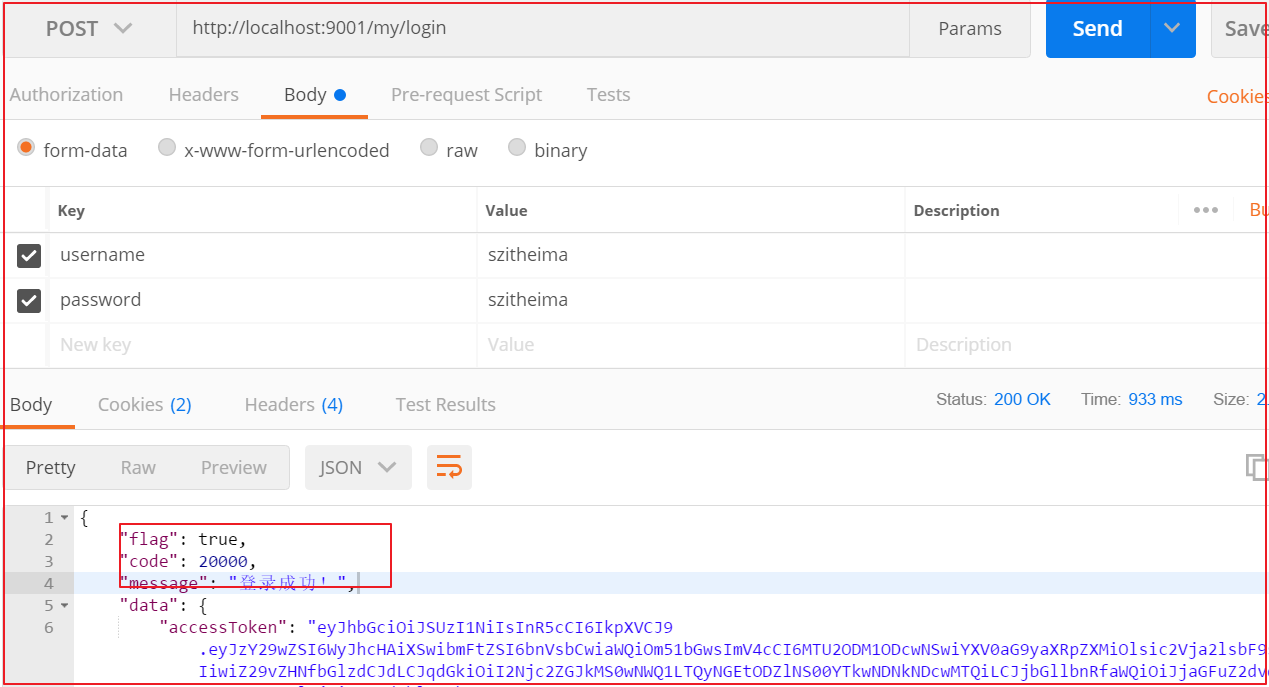


#### 放行/my/login

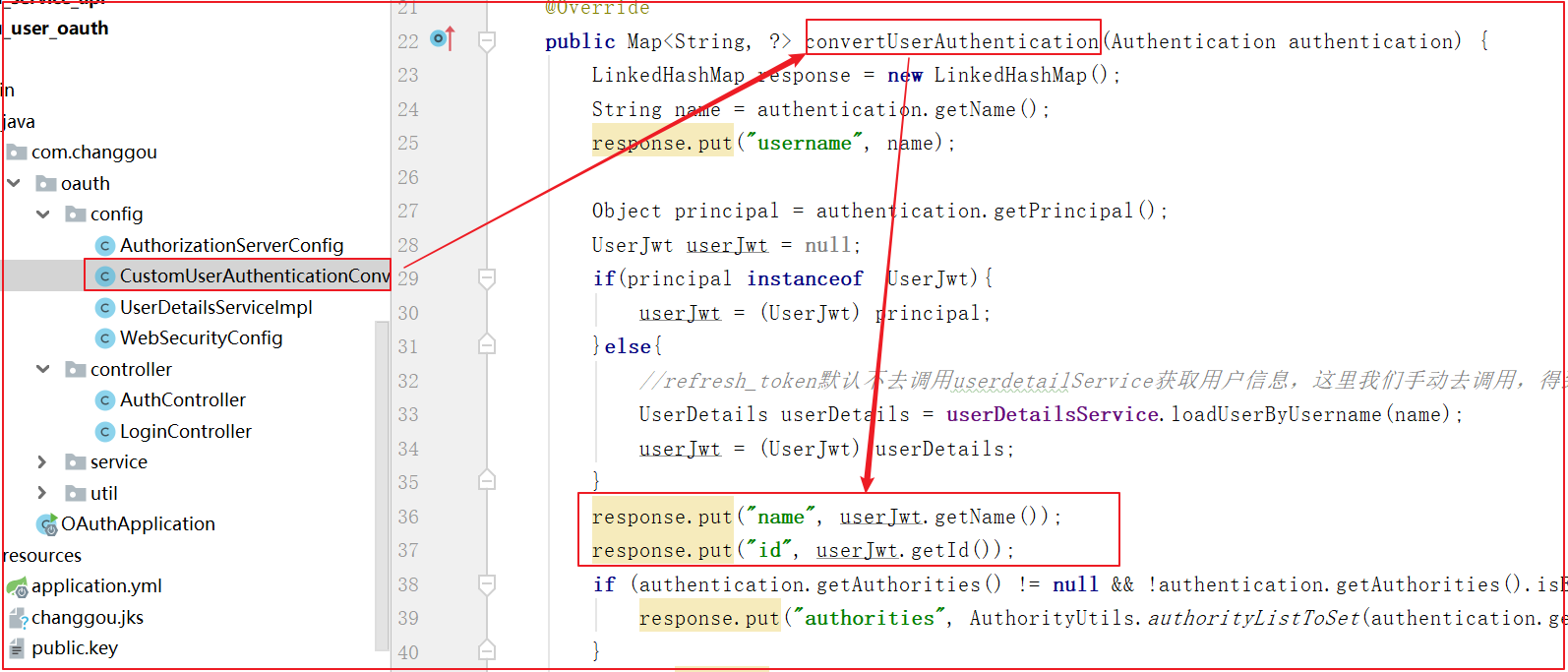
报错了未经许可，这时是因为，我们没有令牌的情况下去访问/my/login是不合法的，所以我们要在没有令牌的情况下放行此操作。



重启认证服务器，再次访问<http://localhost:9001/my/login>



扩展信息，如果想添加载荷，多返回些信息，可以在如下对象中添加：



注意：测试时，以解析令牌得到的结果查询。