

## 学成在线-第16天-讲义- Spring Security Oauth2 JWT

# 1 用户认证需求分析

## 1.1 用户认证与授权

截至目前，项目已经完成了在线学习功能，用户通过在线学习页面点播视频进行学习。如何去记录学生的学习过程呢？要想掌握学生的学习情况就需要知道用户的身份信息，记录哪个用户在什么时间学习什么课程；如果用户要购买课程也需要知道用户的身份信息。所以，去管理学生的学习过程最基本的要实现用户的身份认证。

### 什么是用户身份认证？

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。

### 什么是用户授权？

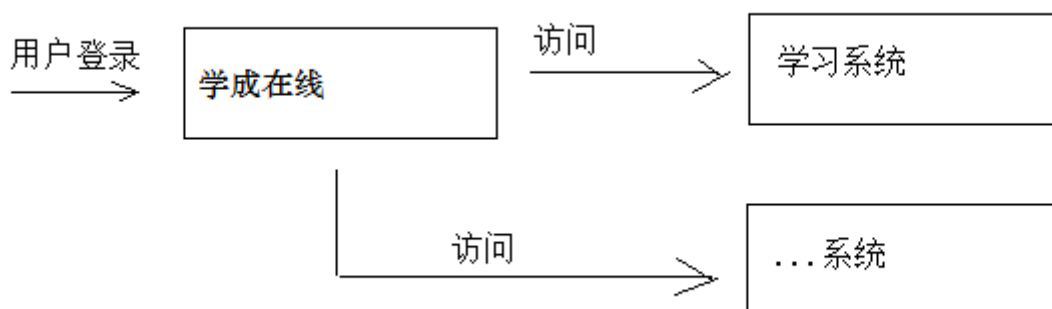
用户认证通过后去访问系统的资源，系统会判断用户是否拥有访问资源的权限，只允许访问有权限的系统资源，没有权限的资源将无法访问，这个过程叫用户授权。

## 1.2 单点登录需求

本项目包括多个子项目，如：学习系统，教学管理中心、系统管理中心等，为了提高用户体验性需要实现用户只认证一次便可以在多个拥有访问权限的系统中访问，这个功能叫做单点登录。

引用百度百科：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。

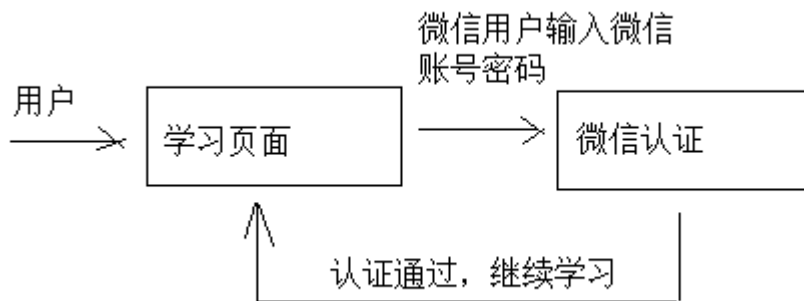
下图是SSO的示意图，用户登录学成网一次即可访问多个系统。



## 1.3 第三方认证需求

作为互联网项目难免需要访问外部系统的资源，同样本系统也要访问第三方系统的资源接口，一个场景如下：

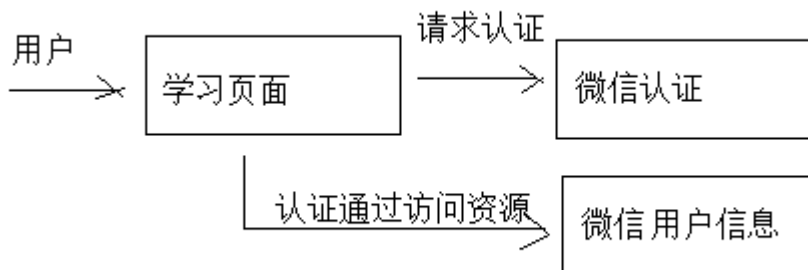
一个微信用户没有在本系统注册，本系统可以通过请求微信系统来验证该用户的身份，验证通过后该用户便可在本系统学习，它的基本流程如下：



从上图可以看出，微信不属于本系统，本系统并没有存储微信用户的账号、密码等信息，本系统如果要获取该用户的基本信息则需要首先通过微信的认证系统（微信认证）进行认证，微信认证通过后本系统便可获取该微信用户的基本信息，从而在本系统将该微信用户的头像、昵称等信息显示出来，该用户便不用在本系统注册却可以直接学习。

什么是第三方认证（跨平台认证）？

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。

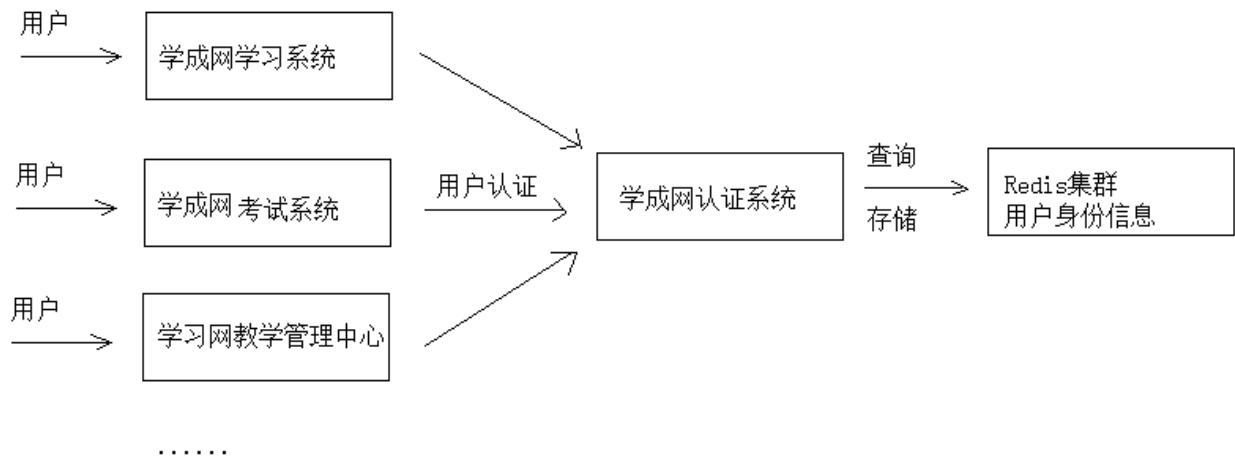


## 2 用户认证技术方案

### 2.1 单点登录技术方案

也可以看作第三方系统

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

## 2.2 OAuth2认证

### 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。

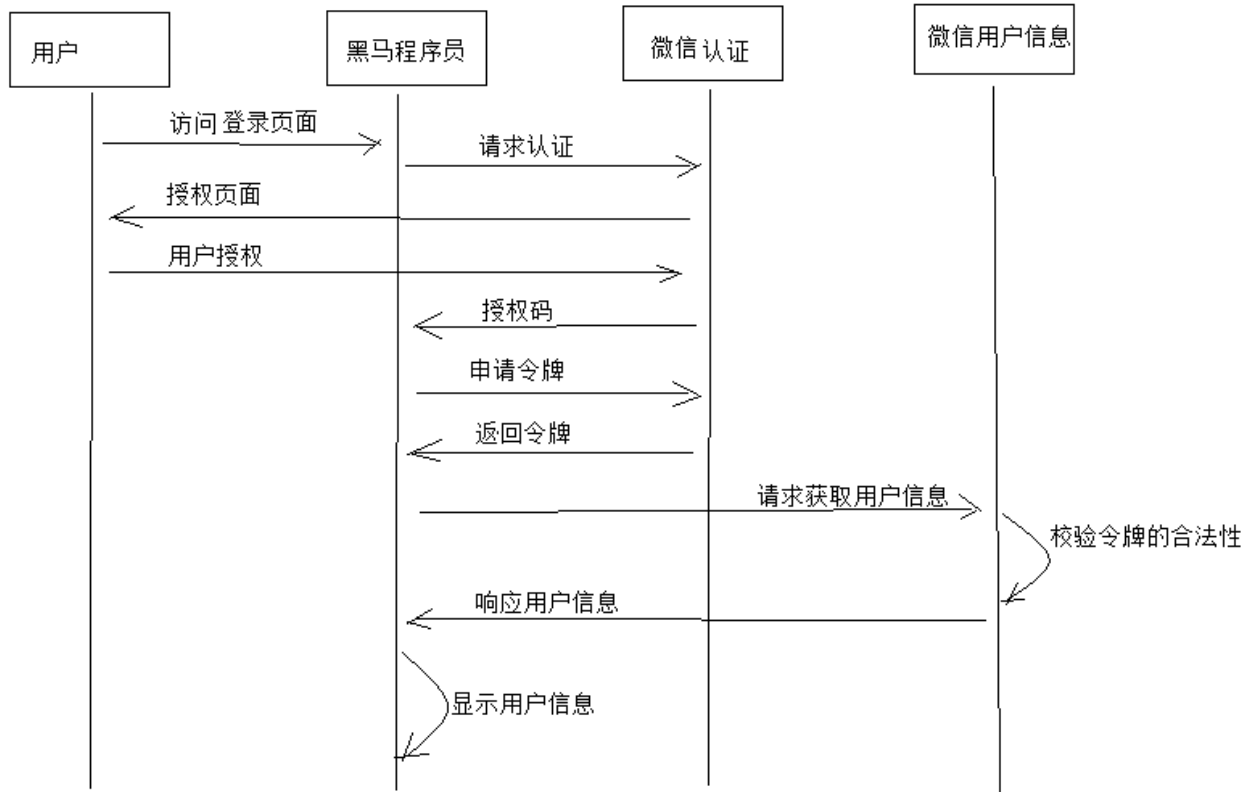
OAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。同时，任何第三方都可以使用OAuth认证服务，任何服务提供商都可以实现自身的OAuth认证服务，因而OAuth是开放的。业界提供了OAuth的多种实现如PHP、JavaScript，Java，Ruby等各种语言开发包，大大节约了程序员的时间，因而OAuth是简易的。互联网很多服务如Open API，很多大公司如Google，Yahoo，Microsoft等都提供了OAuth认证服务，这些都足以说明OAuth标准逐渐成为开放资源授权的标准。

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“微信”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

微信会先验证

就是用户

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，微信会询问用户是否给授权黑马程序员访问自己的微信数据，用户点击“确认登录”表示同意授权，微信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。



## 黑马程序员用户中心

即将登录黑马程序员用户中心，请确  
认是本人操作

- 使用你的帐号登录该应用

确认登录

取消

微信给的授权码，应该向微信认证服务器申请令牌

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。

微信

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

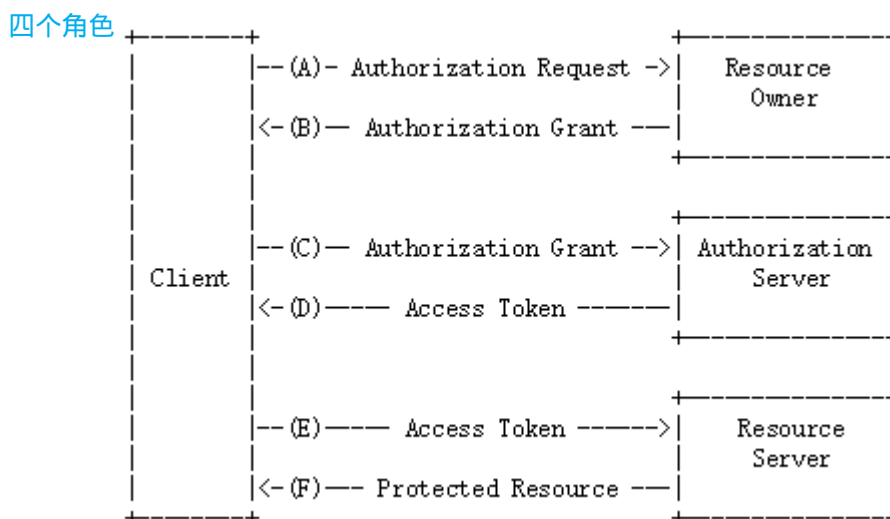
资源服务器校验令牌的合法性，如果合法则向用户响应资源信息内容。

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。

第三方认证成功后黑马程序员是不是也颁发一个自己的令牌给用户呢？

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：学成在线Android客户端、学成在线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） 刚才就是微信认证服务器

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 刚才就是微信的用户信息

存储资源的服务器，比如，学成网用户管理服务器存储了学成网的用户信息，学成网学习服务器存储了学生的学习信息，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。

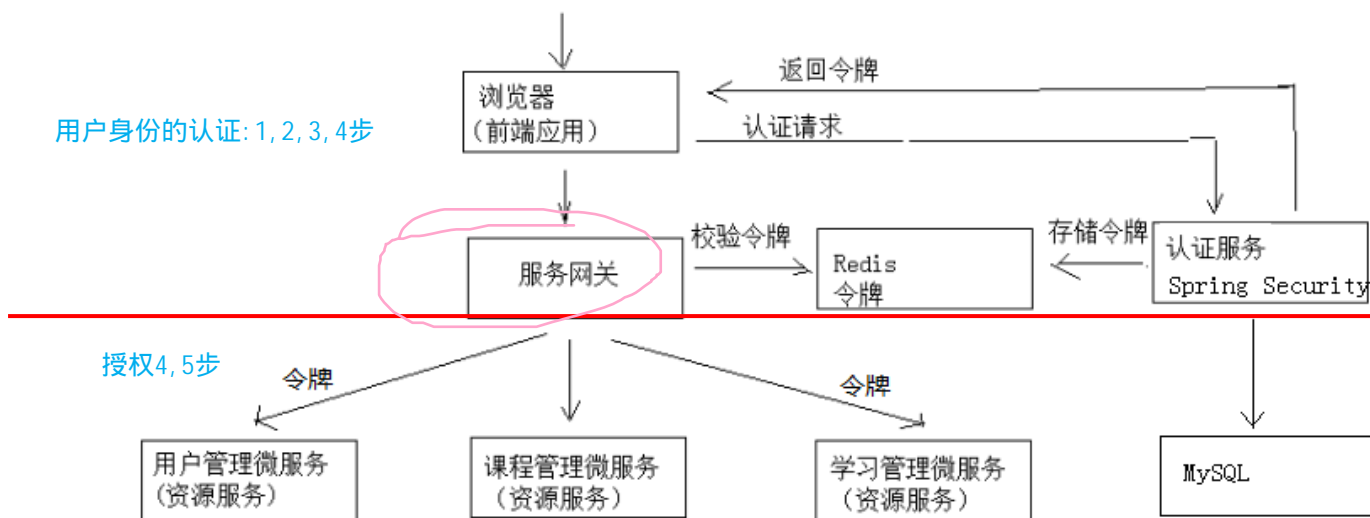
## 2.2.2 Oauth2在本项目的应用

Oauth2是一个标准的开放的授权协议，应用程序可以根据自己去使用Oauth2，本项目使用Oauth2实现如下目标：

- 1、学成在线访问第三方系统的资源
- 2、外部系统访问学成在线的资源
- 3、学成在线前端（客户端）访问学成在线微服务的资源。
- 4、学成在线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 Spring Security Oauth2研究

### 3.1 目标

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



### 3.2.1 导入基础工程

需要的包

导入“资料”目录下的 xc-service-ucenter-auth 工程，该工程是基于 Spring Security OAuth2 的一个二次封装的工程，导入此工程研究 OAuth2 认证流程。

### 3.2.2 创建数据库

导入资料目录下的 xc\_user.sql，创建用户数据库



以“oauth\_”开头的表都是 spring Security 自带的表。

本项目中 spring Security 主要使用 oauth\_client\_details 表：

	Field	Type	Comment
🔑	client_id	varchar(48) NOT NULL	
	resource_ids	varchar(256) NULL	
	client_secret	varchar(256) NULL	
	scope	varchar(256) NULL	
	authorized_grant_types	varchar(256) NULL	
	web_server_redirect_uri	varchar(256) NULL	
	authorities	varchar(256) NULL	
	access_token_validity	int(11) NULL	
	refresh_token_validity	int(11) NULL	
	additional_information	varchar(4096) NULL	
	autoapprove	varchar(256) NULL	

client\_id：客户端id [客户端id是什么？](#)

resource\_ids：资源id（暂时不用）

client\_secret：客户端密码

scope：范围

access\_token\_validity：访问token的有效期（秒）

refresh\_token\_validity：刷新token的有效期（秒）

authorized\_grant\_type：授权类型，authorization\_code,password,refresh\_token,client\_credentials

## 3.3 OAuth2授权码模式 直接向spring security内部自带的类申请令牌，不经过自己的controller

如何运作，配置类是什么，现在不能授权了

### 3.3.1 OAuth2授权模式

OAuth2有以下授权模式：

授权码模式（Authorization Code） 隐式授权模式（Implicit）  
密码模式（Resource Owner Password Credentials） 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。

### 3.3.2 授权码授权流程

**上边例举**的黑马程序员网站使用微信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，请求微信认证服务器申请令牌
- 4、微信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

### 3.3.2 申请授权码

1. 请求认证服务获取授权码：  
Get请求：

```
localhost:40400/auth/oauth/authorize?client_id=XcWebApp&response_type=code&scope=app&redirect_uri=http://localhost
```

参数列表如下：这些参数都是些啥玩意

client\_id：客户端id，和授权配置类中设置的客户端id一致。只有允许的客户端才能够颁发授权码，到时会向数据库查询是否有

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）。

2. 首先跳转到登录页面：XcWebApp

← → ↻ ⓘ localhost:40400/auth/login

### Login with Username and Password

User:

Password:

都是XcWebApp, 数据库中的是client\_secret, 经过加密的

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。

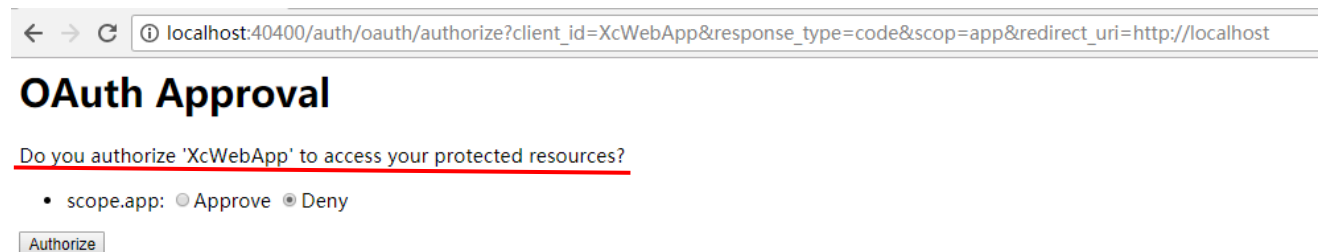
当前导入的基础工程中将正确的密码硬编码为“123”，所以这里账号随意输入，密码输入123即可认证通过。

密码账号同名

Your login attempt was not successful, try again.

Reason: No client with requested id: itcast

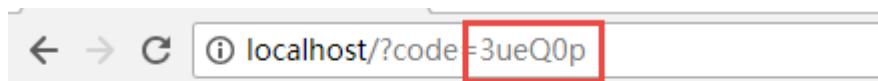
3. 接下来进入授权页面：



点击“同意”。

4. 接下来返回授权码： 每次都不同

认证服务携带授权码跳转redirect\_uri



### 3.3.3 申请令牌

拿到授权码后，申请令牌。

Post请求：<http://localhost:40400/auth/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？

为什么需要

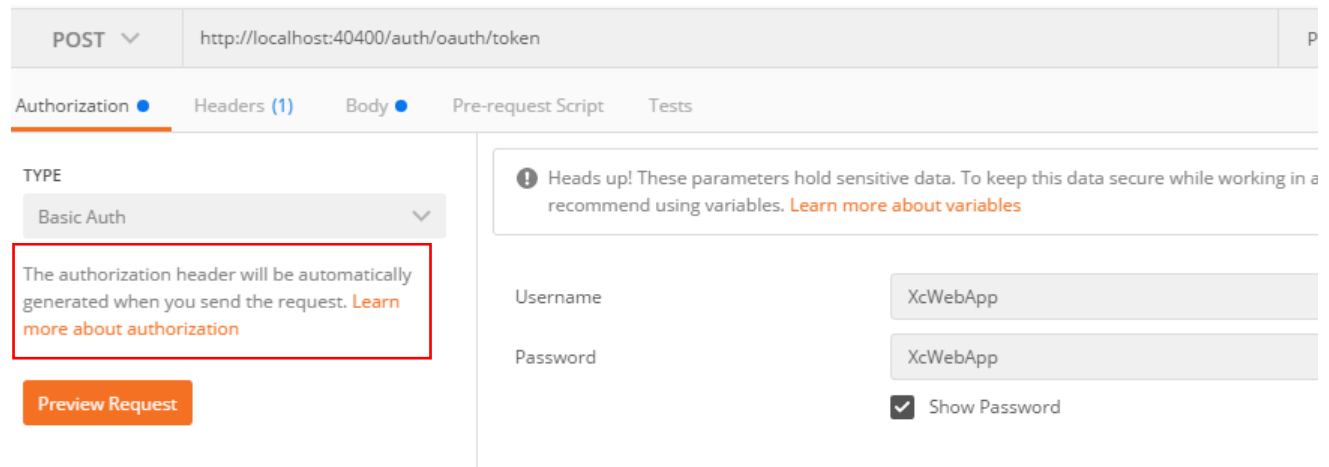
http协议定义的一种认证方式，将客户端id和客户端密码按照“客户端ID:客户端密码”的格式拼接，并用base64编码，放在header中请求服务端，一个例子：

Authorization : Basic WGNXZWJBcHA6WGNXZWJBcHA=

WGNXZWJBcHA6WGNXZWJBcHA= 是用户名:密码的base64编码。编码方式根据id和密码而定,岂不是很容易被截取?  
认证失败服务端返回 401 Unauthorized 跟密码授权的密码不一样的

以上测试使用postman完成：

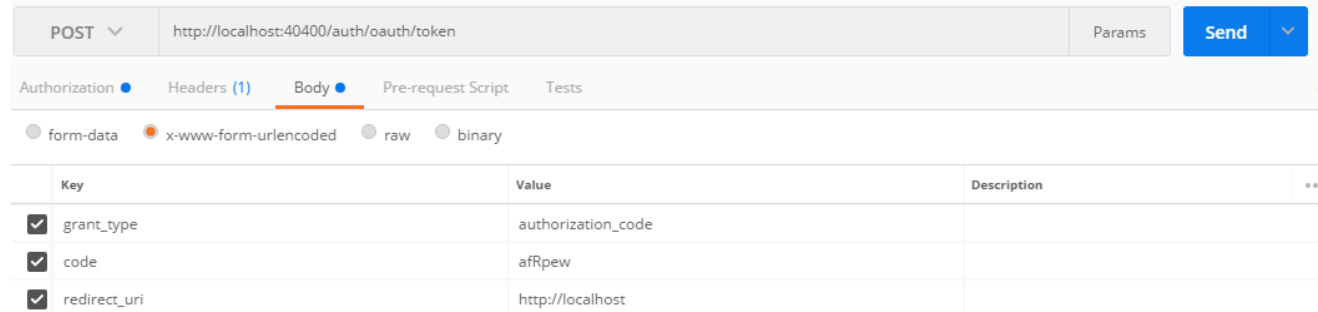
http basic认证：



The image shows the Postman interface for configuring a Basic Auth request. The URL is `http://localhost:40400/auth/oauth/token`. The 'Authorization' tab is selected. Under 'TYPE', 'Basic Auth' is chosen. A red box highlights a note: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'. The 'Username' and 'Password' fields both contain 'XcWebApp'. The 'Show Password' checkbox is checked. A 'Preview Request' button is visible.

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。

Post请求参数：



The image shows the Postman interface for configuring the body of a POST request. The URL is `http://localhost:40400/auth/oauth/token`. The 'Body' tab is selected. The 'x-www-form-urlencoded' radio button is chosen. A table lists the parameters:

Key	Value	Description
<input checked="" type="checkbox"/> grant_type	authorization_code	
<input checked="" type="checkbox"/> code	afrPew	
<input checked="" type="checkbox"/> redirect_uri	http://localhost	

点击发送：

申请令牌成功：



#### 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

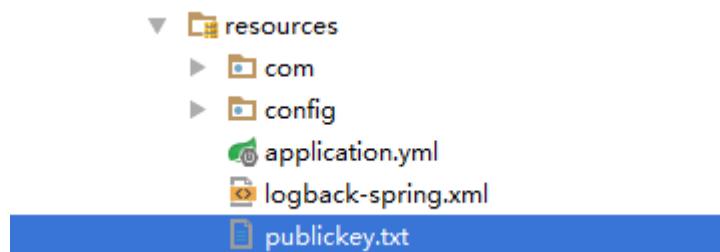
### 3.3.4.2 资源服务授权配置

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。

#### 1、配置公钥      令牌如何生效

认证服务生成令牌采用非对称加密算法，认证服务采用私钥加密生成令牌，对外向资源服务提供公钥，资源服务使用公钥 来校验令牌的合法性。

将公钥拷贝到 publickey.txt文件中，将此文件拷贝到资源服务工程的classpath下



#### 2、添加依赖

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

#### 4、在config包下创建ResourceServerConfig类：

```
@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 = "publickey.txt";
```



```
//定义JwtTokenStore，使用jwt令牌
@Bean
public TokenStore tokenStore(JwtAccessTokenConverter jwtAccessTokenConverter) {
    return new JwtTokenStore(jwtAccessTokenConverter);
}

//定义JJwtAccessTokenConverter，使用jwt令牌
@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
@Override
public void configure(HttpSecurity http) throws Exception {
    //所有请求必须认证通过
    http.authorizeRequests().anyRequest().authenticated();
}
}
```

### 3.3.4.3 资源服务授权测试

这里我们测试课程图片查询

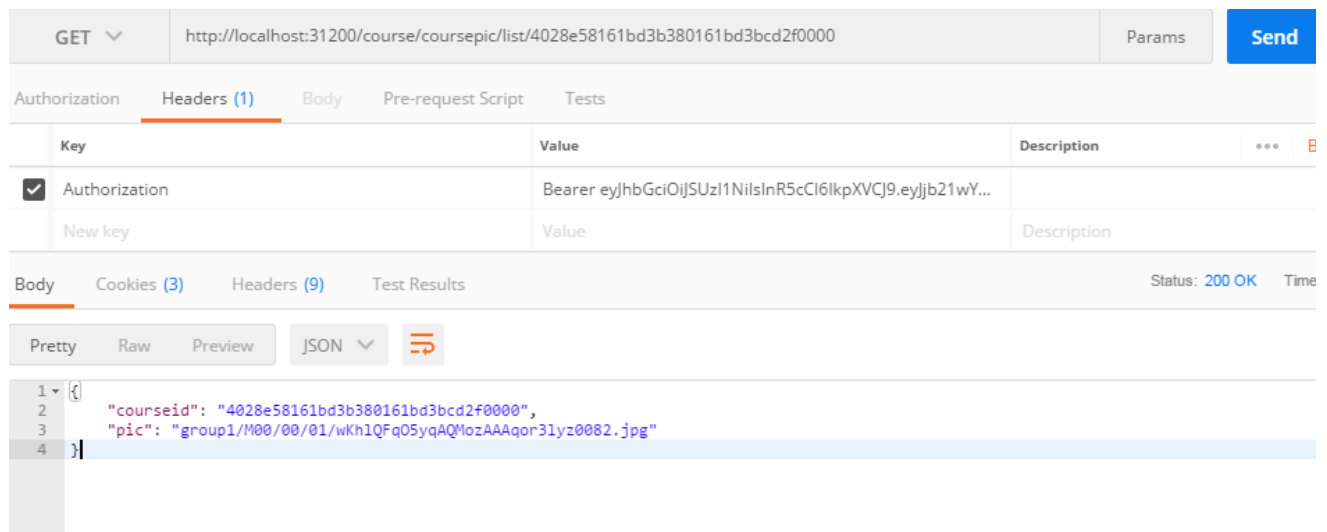
get <http://localhost:31200/course/coursepic/list/4028e58161bd3b380161bd3bcd2f0000>

请求时没有携带令牌则报错：

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

请求时携带令牌：

在http header中添加 Authorization : Bearer 令牌



GET http://localhost:31200/course/coursepic/list/4028e58161bd3b380161bd3bcd2f0000 Params Send

Authorization Headers (1) Body Pre-request Script Tests

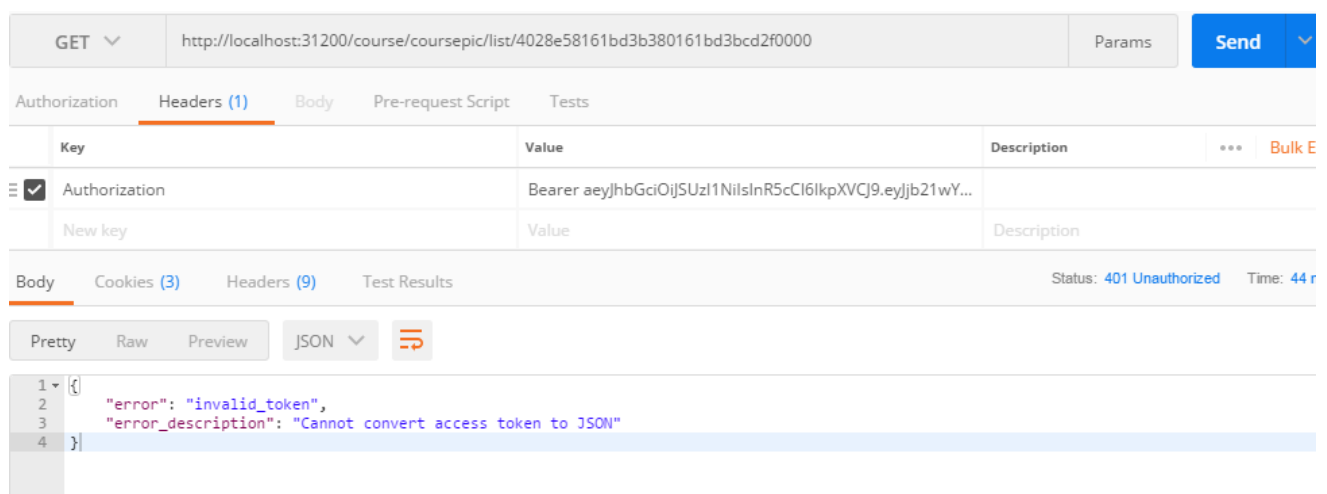
Key	Value	Description
Authorization	Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJjb21wY...	
New key	Value	Description

Body Cookies (3) Headers (9) Test Results Status: 200 OK Time

Pretty Raw Preview JSON

```
1 {
2   "courseid": "4028e58161bd3b380161bd3bcd2f0000",
3   "pic": "group1/M00/00/01/wKh1QFq05yqAQMoZAAAqor3lyz0082.jpg"
4 }
```

当输入错误的令牌也无法正常访问资源。



GET http://localhost:31200/course/coursepic/list/4028e58161bd3b380161bd3bcd2f0000 Params Send

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Authorization	Bearer aeyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJjb21wY...	
New key	Value	Description

Body Cookies (3) Headers (9) Test Results Status: 401 Unauthorized Time: 44 r

Pretty Raw Preview JSON

```
1 {
2   "error": "invalid_token",
3   "error_description": "Cannot convert access token to JSON"
4 }
```

### 3.3.4.4 解决swagger-ui无法访问

当课程管理加了授权之后再访问swagger-ui则报错：





修改授权配置类ResourceServerConfig的configure方法：

针对swagger-ui的请求路径进行放行：

都是直接向spring security内部自带的类申请令牌，不经过自己的controller

```
//Http安全配置，对每个到达系统的http请求链接进行校验
@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("/v2/api-docs", "/swagger-resources/configuration/ui",
            "/swagger-resources", "/swagger-resources/configuration/security",
            "/swagger-ui.html", "/webjars/**").permitAll()
        .anyRequest().authenticated();
}
//所有请求必须认证通过
http.authorizeRequests().anyRequest().authenticated();
```

注意：

通过上边的配置虽然可以访问swagger-ui，但是无法进行单元测试，除非去掉认证的配置或在上边配置中添加所有请求均放行（"/\*\*"）。 仅仅是放开认证，方法还需要检查权限的

### 3.4 Oauth2密码模式授权 都是直接向spring security内部自带的类申请令牌，不经过自己的controller

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:40400/auth/oauth/token>

参数：

grant\_type：密码模式授权填写password

username：账号 itcast

这里怎么有两个用户和密码？

password：密码 111111

并且此链接需要使用 http Basic认证。

POST

http://localhost:40400/auth/oauth/token

Params

Send

Save

Authorization

Headers (2)

Body

Pre-request Script

Tests

Cookies

TYPE

Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Username

XcWebApp

Password

XcWebApp

☒ Show Password

上边参数使用x-www-form-urlencoded方式传输，使用postman测试如下：

POST

http://localhost:40400/auth/oauth/token

Params

Send

Save

grant\_type

password

redirect\_uri

http://localhost

username

mrt

password

123

New key

Value

Description

Body

Cookies

Headers (11)

Test Results

Status: 200 OK

Time: 200 ms

Size: 1.77 KB

Pretty

Raw

Preview

JSON

🔍

```
1 {
2   "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
   .eyJjb21wYV55SWQlOm51bGwsInVzZXJ3aWwMiOm51bGwsInVzZXJ3bmFtZSI6Im1ydCIsInNjb3B1IjpbImFwcCJdLCJ1Ym11IjpbudWxsLCJ1dHlwZSI6bnVsbCwiaWQiOm51bGwsImV4cCI6MTUz
   MTI1NDY4MCIwIjoiMjZDZDAtMjZjYS00MzA3LTK2MmItNjIzNDk4ZGY3MjgyIiwiaW12xpZm50X21kIjoimGNXZWJ8cHA1fQ
   .bRcpNTE9wYQ89TwG_vx2RRQOQ37GKp7lwFj3Ck6j8SziF0pTgaPmugES-s1P6h0_SgxEZTi3Ug-Qs4LkfuFQZ11yepQiSDkHfwy7verHS0LLbfKpSrnIUDtq0w
   -dkqKy7rboirfxCZLGebP22ETSEKMHFTN7SjY3reekp90LCu34uCAchQHZ1v80LEKDeak4eNV2JDdz8q41ugbDbyte1qMv2yv3r2QZLMroOYF0R3PXXuy2U6
   -f89JI82w8jcq3ooRwdvYu_Mh6fB5YcvJz18VoGoFEB-f1LThBG0aebS6Z93IX7m3XZQ3cyoxdx12R1fLvoHjbn2jd5HDko1w",
3   "token_type": "bearer",
4   "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
   .eyJjb21wYV55SWQlOm51bGwsInVzZXJ3aWwMiOm51bGwsInVzZXJ3bmFtZSI6Im1ydCIsInNjb3B1IjpbImFwcCJdLCJ1dHlwZSI6bnVsbCwiaWQiOm51bGwsImV4cCI6MTUz
   MTI1NDY4MCIwIjoiMjZDZDAtMjZjYS00MzA3LTK2MmItNjIzNDk4ZGY3MjgyIiwiaW12xpZm50X21kIjoimGNXZWJ8cHA1fQ
   .h8a0MEH1GFwOn6mUcunH-apkc71gVclhwZ7U3eG3xDxzsruBj2hTf3Gj1UVQ44b1b4Ae6UMbitGXIkhpeMg31A4kTk0i1jgXV2HuaYuT-bJqopLXRdHcV545a8HvvpNMdKZ
   -kGNZsXb04QbwVgmK01JRECzEjEDsnIcwk42sDYTRnA1umG1v4K4kb0-FerLayuhNC173DcEazwRb0E-ILhVEXEv0S00f-YYeKQ
   -U9eqpw5gdrQ6Ev7TLn_KZdobeIAd0HN1FYTBcn2FFT0be226jgbNwbXmHgnRIEzv8j2zEKXt346bGRF5AXKRVt07RYOFAuN_E17r161jKw",
5   "expires_in": 43199,
6   "scope": "app",
7   "jti": "23f6d6d0-3bba-4307-961b-623498df7282"
```

经过UserDetailsService()方法处理

注意：当令牌没有过期时同一个用户再次申请令牌则不再颁发新令牌。

## 3.5校验令牌

Spring Security Oauth2提供校验令牌的端点，如下：

Get: [http://localhost:40400/auth/oauth/check\\_token?token=](http://localhost:40400/auth/oauth/check_token?token=)

参数：

token：令牌

使用postman测试如下：



结果如下：

```
{
  "companyId": null,
  "userpic": null,
  "user_name": "mrt",
  "scope": [
    "app"
  ],
  "name": null,
  "utype": null,
  "id": null,
  "exp": 1531254828,
  "jti": "6a00f227-4c30-47dc-a959-c0c147806462",
  "client_id": "XcWebApp"
}
```

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

## 3.5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:40400/auth/oauth/token>

参数：

refresh\_token：刷新令牌（注意不是access\_token，而是refresh\_token）

刷新令牌通常是在令牌快过期时进行刷新。

在介绍JWT之前先看一下**传统**校验令牌的方法，如下图：

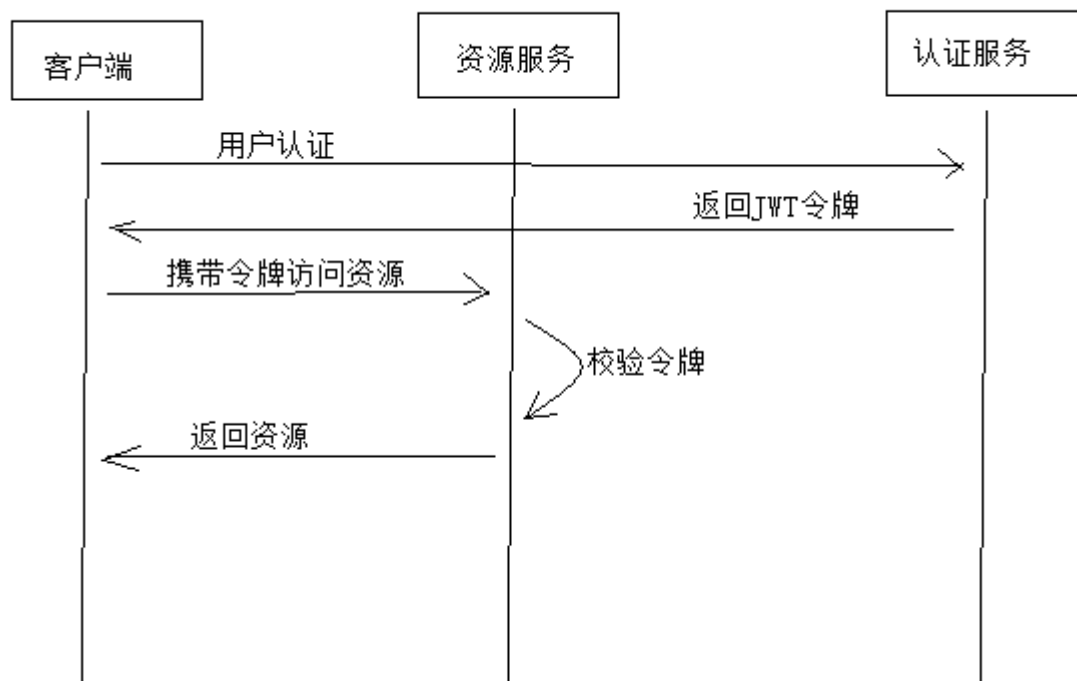


传统授权方法的问题是用户每次请求资源服务，资源服务都需要携带令牌访问认证服务去校验令牌的合法性，并根据令牌获取用户的相关信息，性能低下。

#### 解决：

使用JWT的思路是，用户认证通过会得到一个JWT令牌，JWT令牌中已经包括了用户相关的信息，客户端只需要携带JWT访问资源服务，资源服务根据事先约定的算法自行完成令牌校验，无需每次都请求认证服务完成授权。

JWT令牌授权过程如下图：



#### 什么是JWT？

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：

- 1、jwt基于json，非常方便解析。
- 2、可以在令牌中自定义丰富的内容，易扩展。
- 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。
5. 相比于session，它无需保存在服务器，不占用服务器内存开销; 同时也支持跨域访问

#### 缺点：

- 1、JWT令牌较长，占存储空间比较大。

### 3.6.1.1 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**    内容

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。[为什么要编码?大家都知道编码解码方式](#)

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：[可以还原](#)

```
{
  "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。[也就是根据密钥加密](#)

一个例子：[加密和签名区别](#)

```
HMACSHA256(
  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。

### 3.6.3 JWT入门 密钥证书中有公钥和私钥，私钥用于生成jwt令牌，公钥用于校验令牌

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。

#### 3.6.3.1 生成私钥和公钥

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 [在当前文件夹生成证书](#)

```
keytool -genkeypair -alias xckey -keyalg RSA -keypass xuecheng -keystore xc.keystore -storepass xuechengkeystore
```

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

查询证书信息：

```
keytool -list -keystore xc.keystore
```

删除别名

```
keytool -delete -alias xckey -keystore xc.keystore
```

##### 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\_0g.exe

配置openssl的path环境变量，本教程配置在D:\OpenSSL-Win64\bin [E:/](#)

cmd进入xc.keystore文件所在目录执行如下命令：



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

输入密钥库密码：

```
C:\Users\admin>keytool -list -rfc --keystore xc.keystore | openssl x509 -inform pem -pubkey
输入密钥库口令: xuechengkeystore
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsMcSgG1I1INC0QMMHxkT
CXwjR2AGZ30wJf98mdQxuce+GUX8Kw/E2gKiL6ZLYJH8iG8rtPR6wQvK17g6HjJZ
d8SfFpJpAPSFjM8AgYxm9kJ2LdSurbAjdXH7a+OucLQPYI1MerTCZWarcKmhFGP
JHAVFnDL/gAqcHiVnvov1zfLPquCetH1Y+Aowcj7bNcYsvZwdPyPmaP3gERa2smi
Jn4+f16u+ncPPIkSZBbueMXXnlykoI3w/BPSEui74VX1vPr/aUah+GXNLucGE5H4
e8r4C4SvBSQHnL6j+NWU0cVGLpnP0bKXHwL0w/6z+MG8gm1W1dv6i7yYc1Vb0+
RQIDAQAB
-----END PUBLIC KEY-----
-----BEGIN CERTIFICATE-----
MIIDPTCCAiWgAwIBAgIEdCHhYjANBgkqhkiG9w0BAQsFADBPMBQswCQYDVQQGEwJj
bJELMAkGA1UECBMCaG4xCzAIBgNVBACTAnc6MQswCQYDVQQKEwJtcTELMakGA1UE
CxMcbXEzDDAKBgNVBAMTA21ydAeFw0xODA1MjMwMzU2NTZaFw0xODA1MjMwMzU2
NTZaME8xChZBbnVBAWtAmNMQswCQYDVQQIEwJobjELMAkGA1UEBxMCenoxChZB
BgNVBAoTAmNMQswCQYDVQQLEwJtcTEEMMAoGA1UEAxMDbXJOMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsMcSgG1I1INC0QMMHxkTCXwjR2AGZ30wJf98
mdQxuce+GUX8Kw/E2gKiL6ZLYJH8iG8rtPR6wQvK17g6HjJZd8SfFpJpAPSFjM8Ag
Yxm9kJ2LdSurbAjdXH7a+OucLQPYI1MerTCZWarcKmhFGP JHAVFnDL/gAqcHiV
nvov1zfLPquCetH1Y+Aowcj7bNcYsvZwdPyPmaP3gERa2smiJn4+f16u+ncPPIkS
ZBbueMXXnlykoI3w/BPSEui74VX1vPr/aUah+GXNLucGE5H4e8r4C4SvBSQHnL6
j+NWU0cVGLpnP0bKXHwL0w/6z+MG8gm1W1dv6i7yYc1Vb0+RQIDAQABoyEwHzAd
BgNVHQ4EFggQU0At0QaoBEclY1JVYD3sgmawsUfkwDQYJKoZIhvcNAQELBQADggEB
ADDyrS6muic7gKcbssWVrOGIEQxX19fmSQd0QXaqIOWkk4MgboBWFcx1sFjH/BXf
fMU4JD6Yy6V8XknwYXIP0wTPBF2rXKkt970J+PcIZ5SbnfWb/mjCR1nwtAednP0x
EUnyLX1BryXFykcxN9cd3wlnhyifSof0n1GCUqMT8/Z04ZmCeiOH1fzuvBi1QgwZ
RsBqW+krMmAWjvsvb0DRNOfB3TN7/k9gTtA7eZ3UTqitwx8RI1Noy1rhWCavk2Yd
N5rWqQxajXaco+u1VvwnwnwzyZ95bRo2gNZ/aIddPvoh0tYPF+xziLgUGA8V4rg8
```

下边这一段就是公钥内容：

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

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAIijyxMdq4S6L1Af1rtB8SjCZHNGsQG8JTFgy55eYvzG0B
/E4AudR2prSRBvF7NYPL47scRCNPGlnvbQczBHbBug6uOr78qnWsYxHIW6Aa5d1NsmOD4DLtSw8eX0hFyK5F
j6ScYOSFBz9cd1nNTvx2+olV0lJdcpQdQhsfgsEr1ntvWterZt/8r7xNN83gHYuZ6TM5MYvjQNBc5qC7Krs9wM7U
oQuL+s0X6RIOib7/mcLn/IFLsLDdYQAZksDx/6+tt+1oHdMarChIPYT1sx9Dwj2j2mvFNDTKKKKAQ0cv14VrhZ67Vj
mz2yMJePDqUi0JYS2r0ilo7n8vN7s83v5uOQIDAQAB-----END PUBLIC KEY-----
```

将上边的公钥拷贝到文本文件中，合并为一行。

### 3.6.3.2 生成jwt令牌

在认证工程创建测试类，测试jwt令牌的生成与验证。

```
//生成一个jwt令牌
@Test
public void testCreateJwt(){
    //证书文件
    String key_location = "xc.keystore";
    //密钥库密码
    String keystore_password = "xuechengkeystore";
    //访问证书路径
    ClassPathResource resource = new ClassPathResource(key_location);
    //密钥工厂
    KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(resource,
    keystore_password.toCharArray());

    //密钥的密码，此密码和别名要匹配
```





```
String keypassword = "xuecheng";
//密钥别名
String alias = "xckey";
//密钥对 ( 密钥和公钥 )
KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias, keypassword.toCharArray());
//私钥
RSAPrivateKey aPrivate = (RSAPrivateKey) keyPair.getPrivate();
//定义payload信息
Map<String, Object> tokenMap = new HashMap<>();
tokenMap.put("id", "123");
tokenMap.put("name", "mrt");
tokenMap.put("roles", "r01,r02");
tokenMap.put("ext", "1");
//生成jwt令牌
Jwt jwt = JwtHelper.encode(JSON.toJSONString(tokenMap), new RsaSigner(aPrivate));
//取出jwt令牌
String token = jwt.getEncoded();
System.out.println("token="+token);

}
```

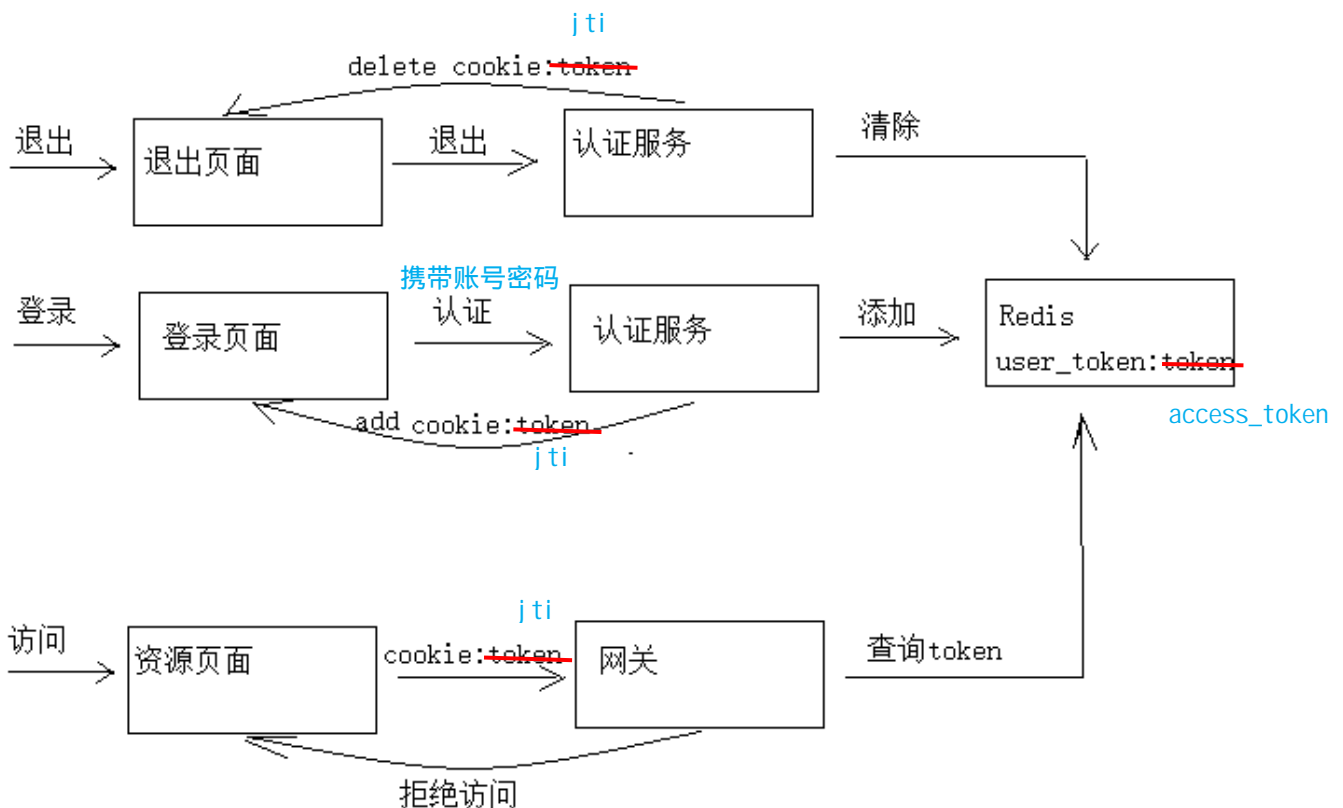
### 3.6.3.3 验证jwt令牌

```
//资源服务使用公钥验证jwt的合法性，并对jwt解码
@Test
public void testVerify(){
    //jwt令牌
    String token
    ="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHQiOiIxIiwicm9sZXMiOiJyMDEscjAyIiwibmFtZSI6Im1ydCIscImlkIjoiaMTIzIn0.KK7_67N5d1Dthd1PgDHMSbi0UlmjGRcm_XJUWseJ2eZyJJWoPP2IcEZgAU3tUaaKEHUf9wSRwaDgwhrwfyIcSHbs8oy3zOQEL8j5A0jzBBs7vnRmB7DbSaQD7eJiQVJOX01QpdmEFgjhc_IBCVTJCVWgZw60IEW1_Lg5tqaLvCiI126K48pJB5f-1e2zgYMzqR1L2LyTFkq39rG57VOqqSCi3dapsZQd4ctq95SJCXgGdrUDWtD52rp5o6_0uq-mrbRdRxkrQfsa1j8C5IW2-T4eUmiN3f9wF9JxUK1__XC10QkOn-ZTBCdqWwIygDFbU7sf6KzfHJTm5vfj6NIA";
    //公钥
    String publicKey = "-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAIjyxMdq4S6L1Af1rtB8SjCZHNGsQG8JTfGy55eYvzG0B/E4AudR2prSRBvF7NYPL47scRCPNgLn timer Bug6u0r78qnWsYxHlW6Aa5dI5NsmOD4DLtSw8eX0hFyK5Fj6ScYOSFBz9cd1nNTvx2+oIv0lJDcpQdQhsfgsEr1ntvWterZt/8r7xNN83gHYuZ6TM5MYvjQNbc5qC7Krs9wM7UoQuL+s0X6R10ib7/mcLn/1FLsLDdYQAZkSDx/6+t+1oHdMarChIPYT1sx9Dwj2j2mvFNDDTKKKKAQ0cv14Vrhz67Vjnz2yMJePDqUi0JYS2r0iIo7n8vN7s83v5uOQIDAQAB-----END PUBLIC KEY-----";
    //校验jwt
    Jwt jwt = JwtHelper.decodeAndVerify(token, new RsaVerifier(publicKey));
    //获取jwt原始内容 拿到jwt就说明验证成功
    String claims = jwt.getClaims();
    //jwt令牌
    String encoded = jwt.getEncoded();
    System.out.println(encoded);
}
```

## 4 认证接口开发

### 4.1 需求分析 间接向spring security申请令牌, 不过要处理redis储存, cookie保存问题

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：

- 1、实现用户退出注销功能，服务端清除令牌后，即使客户端请求携带token也是无效的。
- 2、由于jwt令牌过长，不宜存储在cookie中，所以将jwt令牌存储在redis，由客户端请求服务端获取并在客户端存储。  
不是有个session storage存储，每次都携带

## 4.2 Redis配置 OneNote中有用法

### 4.2.1 安装Redis

#### 1、安装Redis服务

下载Windows版本的redis : <https://github.com/MicrosoftArchive/redis/tags>

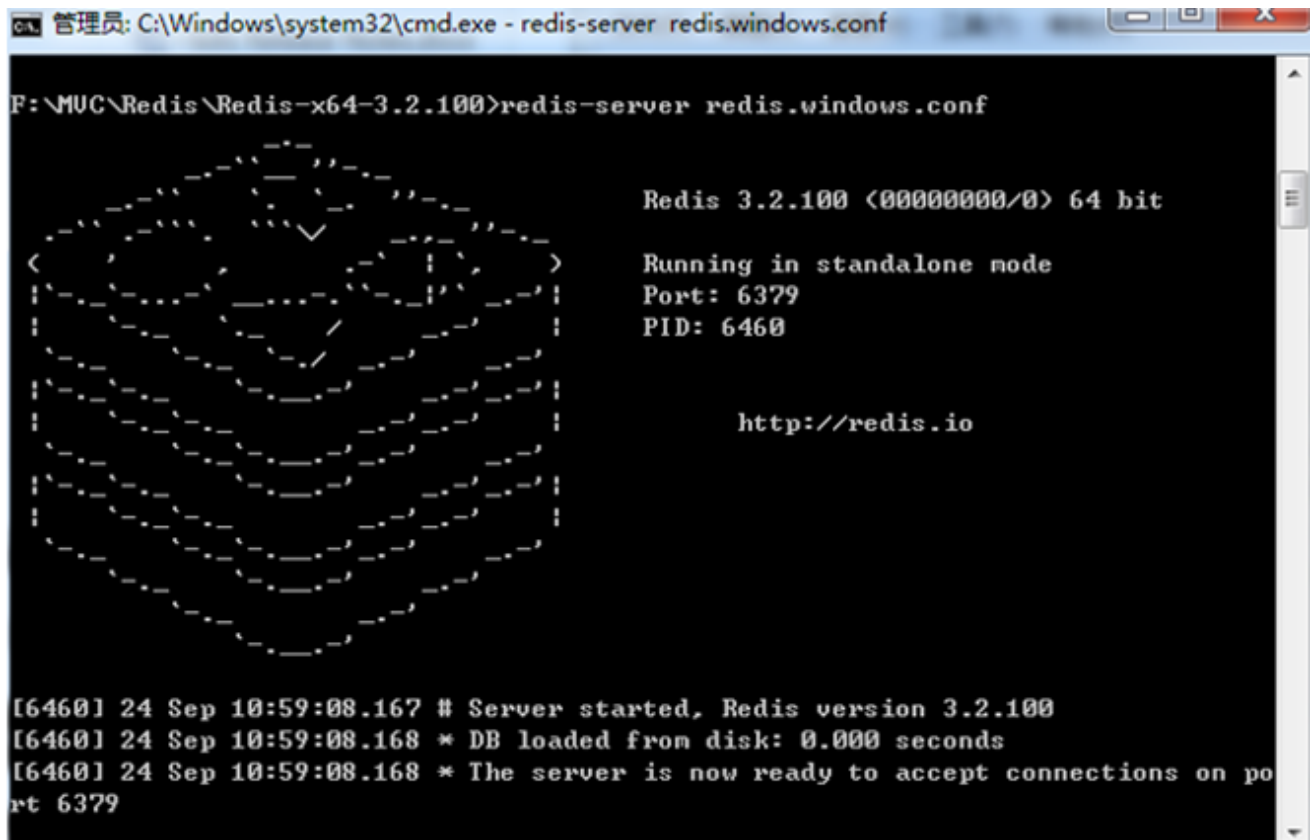
下载Redis-x64-3.2.100版本，解压Redis-x64-3.2.100.zip

进入cmd命令行，进入Redis-x64-3.2.100目录。

运行：

```
redis-server redis.windows.conf
```

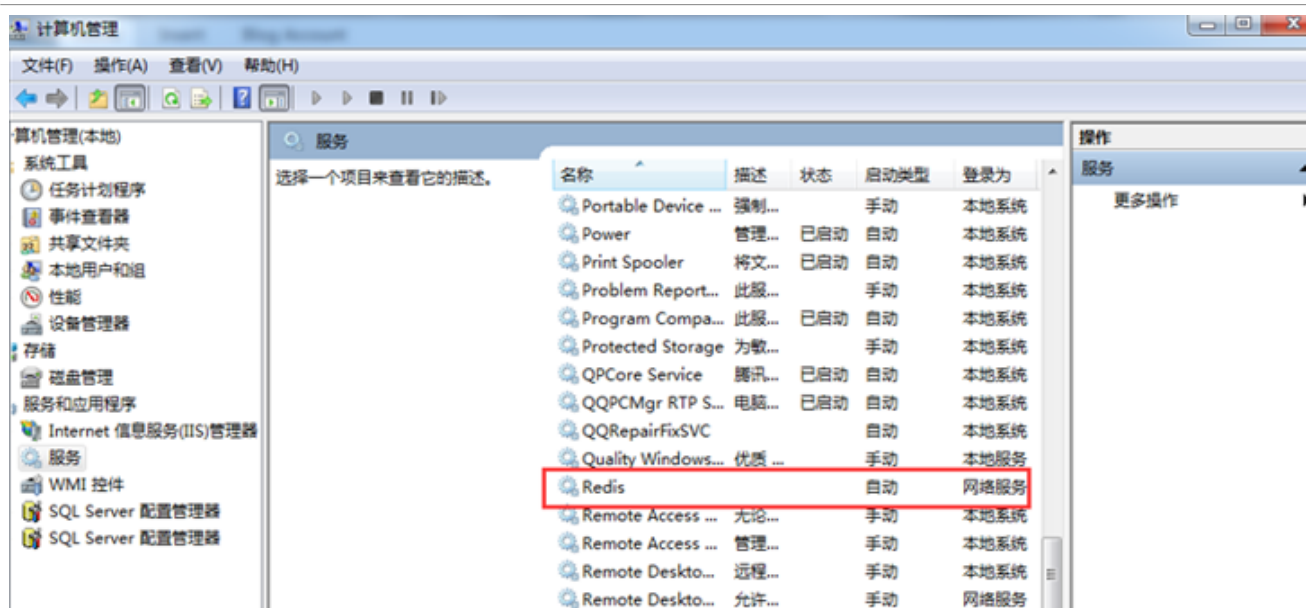
出现 下图说明 redis启动成功：



注册为服务：

```
redis-server --service-install redis.windows-service.conf --loglevel verbose
```

刷新服务，会看到多了一个redis服务。



常用的redis服务命令如下：

进入redis安装目录：

卸载服务：redis-server.exe --service-uninstall

开启服务：redis-server.exe --service-start

停止服务：redis-server.exe --service-stop

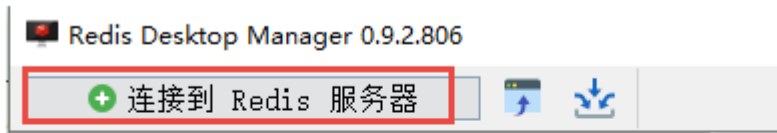
## 2、redis-desktop-manager

下载windows版本的redis客户端：<https://redisdesktop.com/download>

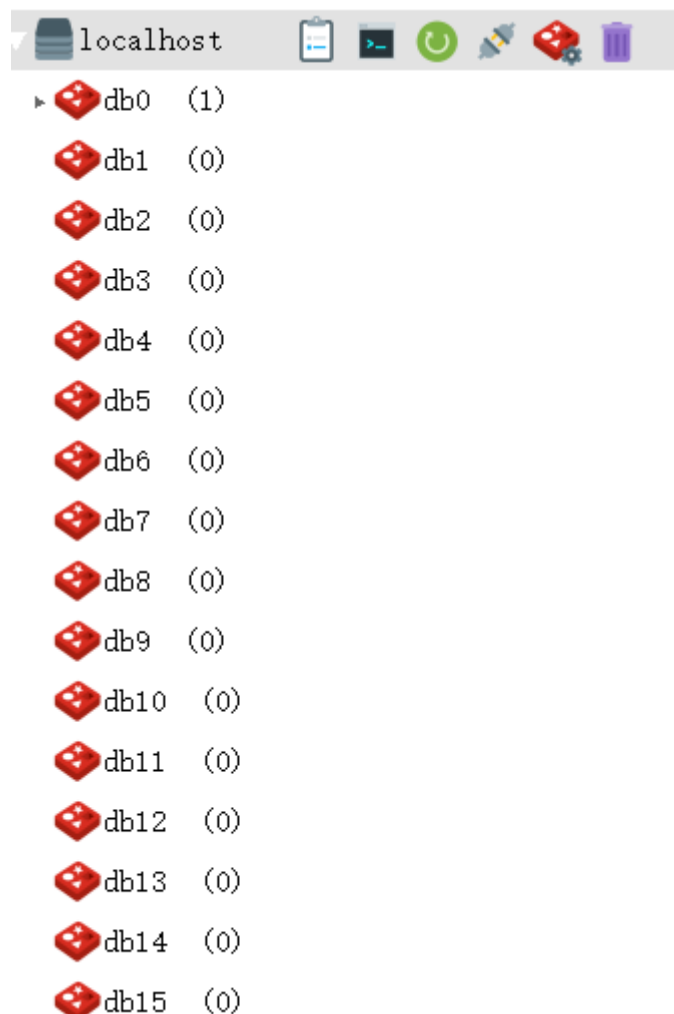
下载redis-desktop-manager-0.9.2.806.exe [这个不好用, 去Gi tHub下载一个](#)

安装后启动redis客户端：

配置redis链接：



连接成功：



## 4.2.2 redis连接配置

在认证服务的application.yml文件中添加如下配置：

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

```
spring:
  application:
    name: xc-service-ucenter-auth
  redis:
    host: ${REDIS_HOST:127.0.0.1}
    port: ${REDIS_PORT:6379}
    timeout: 5000 #连接超时 毫秒
    jedis:
      pool:
        maxActive: 3
        maxIdle: 3
        minIdle: 1
        maxWait: -1 #连接池最大等待时间 -1没有限制
```

## 4.2.3 测试

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class RedisTest {
    @Autowired
    private StringRedisTemplate stringRedisTemplate;
    @Test
    public void testRedis(){
        //定义key
        String key = "user_token:9734b68f-cf5e-456f-9bd6-df578c711390";
        //定义Map
        Map<String,String> mapValue = new HashMap<>();
        mapValue.put("id", "101");
        mapValue.put("username", "itcast");
        String value = JSON.toJSONString(mapValue);
        //向redis中存储字符串
        stringRedisTemplate.boundValueOps(key).set(value, 60, TimeUnit.SECONDS);
        //读取过期时间，已过期返回-2
        Long expire = stringRedisTemplate.getExpire(key);
        //根据key获取value
        String s = stringRedisTemplate.opsForValue().get(key);
        System.out.println(s);
    }
}
```

## 4.3 认证服务

### 4.3.1 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。

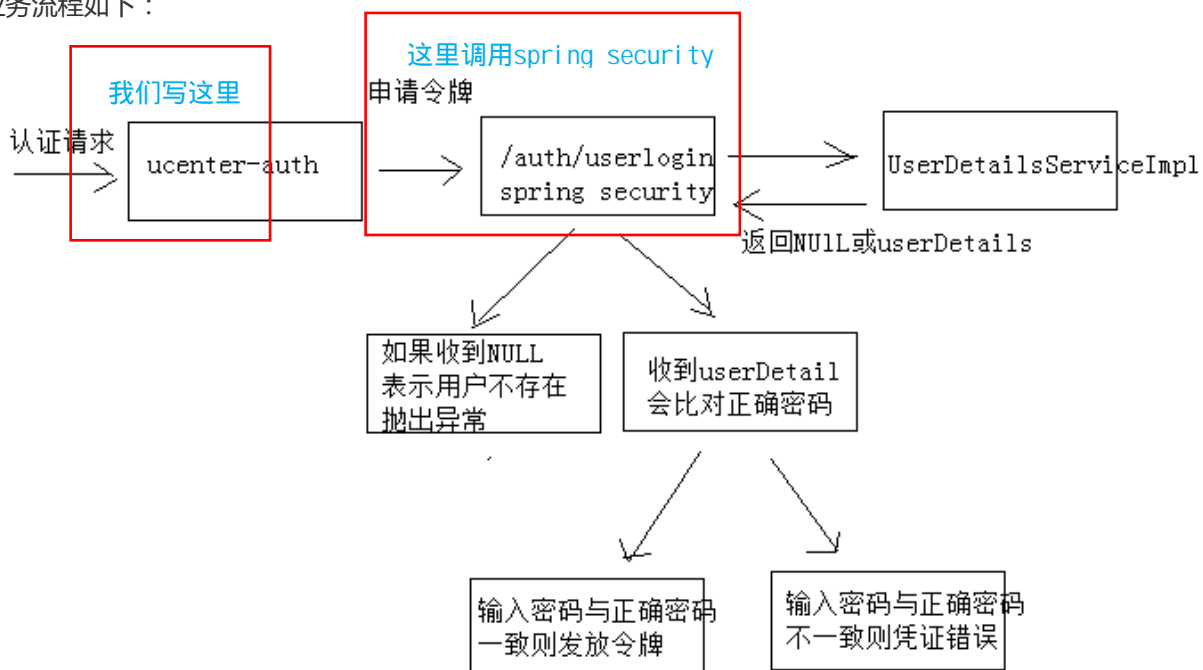
#### 2、退出接口

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。

业务流程如下：



### 4.3.2 Api接口

```
@Api(value = "用户认证",description = "用户认证接口")
public interface AuthControllerApi {

    @ApiOperation("登录")
    public LoginResult login(LoginRequest loginRequest);

    @ApiOperation("退出")
    public ResponseResult logout();

}
```

### 4.3.3 配置参数

在application.yml中配置参数

```
auth:
  tokenValiditySeconds: 1200 #token存储到redis的过期时间
  clientId: XcWebApp
  clientSecret: XcWebApp
  cookieDomain: localhost
  cookieMaxAge: -1 流浪其已关闭, 这个cookie就无效
```

### 4.3.4 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：

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class TestClient {

    @Autowired
    LoadBalancerClient loadBalancerClient;

    @Autowired
    RestTemplate restTemplate;

    @Test
    public void testClient(){

        //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(XcServiceList.XC_SERVICE_UCENTER_AUTH);
        URI uri = serviceInstance.getUri();

        String authUrl = uri+"/auth/oauth/token";
```





```
//URI url, HttpMethod method, HttpEntity<?> requestEntity, Class<T> responseType
// url就是 申请令牌的url /oauth/token
//method http的方法类型
//requestEntity请求内容
//responseType, 将响应的结果生成的类型

//请求的内容分两部分
//1、header信息, 包括了http basic认证信息
MultiValueMap<String, String> headers = new LinkedMultiValueMap<String, String>();
String httpbasic = httpbasic("XcWebApp", "XcWebApp");
//"Basic WGNXZWJBcHA6WGNXZWJBcHA="
headers.add("Authorization", httpbasic);
//2、包括: grant_type、username、password
MultiValueMap<String, String> body = new LinkedMultiValueMap<String, String>();
body.add("grant_type", "password");
body.add("username", "itcast");
body.add("password", "123");
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, 也要正常返回值
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 body1 = exchange.getBody();
System.out.println(body1);

}

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 = Base64.encode(string.getBytes());
    return "Basic "+new String(encode);
}
}
```

## 4.3.4 Dao

暂时使用静态数据, 待用户登录调通再连接数据库校验用户信息。

## 4.3.5 Service

调用认证服务申请令牌，并将令牌存储到 redis。

### 1、AuthToken

创建 AuthToken模型类，存储申请的令牌，包括身份令牌、刷新令牌、jwt令牌

身份令牌：用于校验用户是否认证

刷新令牌：jwt令牌快过期时执行刷新令牌

jwt令牌：用于授权

```
@Data
@ToString
@NoArgsConstructor
public class AuthToken {
    String access_token;//身份token
    String refresh_token;//刷新token
    String jwt_token;//jwt令牌
}
```

申请令牌的service方法如下：

```
@Service
public class AuthService {

    private static final Logger LOGGER = LoggerFactory.getLogger(AuthService.class);

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

    @Autowired
    RestTemplate restTemplate;
    @Autowired
    LoadBalancerClient loadBalancerClient;
    @Autowired
    StringRedisTemplate stringRedisTemplate;

    //认证方法
    public AuthToken login(String username,String password,String clientId,String clientSecret){
        //申请令牌
        AuthToken authToken = applyToken(username,password,clientId, clientSecret);
        if(authToken == null){
            ExceptionCast.cast(AuthCode.AUTH_LOGIN_APPLYTOKEN_FAIL);
        }
        //将 token存储到redis
        String access_token = authToken.getAccess_token();
        String content = JSON.toJSONString(authToken);
        boolean saveTokenResult = saveToken(access_token, content, tokenValiditySeconds);

        if(!saveTokenResult){
```



```
        ExceptionCast.cast(AuthCode.AUTH_LOGIN_TOKEN_SAVEFAIL);
    }
    return authToken;
}
//存储令牌到redis
private boolean saveToken(String access_token,String content,long ttl){
    //令牌名称
    String name = "user_token:" + access_token;
    //保存到令牌到redis
    stringRedisTemplate.boundValueOps(name).set(content,ttl, TimeUnit.SECONDS);
    //获取过期时间
    Long expire = stringRedisTemplate.getExpire(name);
    return expire>0;
}
//认证方法
private AuthToken applyToken(String username,String password,String clientId,String
clientSecret){
    //选中认证服务的地址
    ServiceInstance serviceInstance =
loadBalancerClient.choose(XcServiceList.XC_SERVICE_UCENTER_AUTH);
    if (serviceInstance == null) {
        LOGGER.error("choose an auth instance fail");
        ExceptionCast.cast(AuthCode.AUTH_LOGIN_AUTHSERVER_NOTFOUND);
    }
    //获取令牌的url
    String path = serviceInstance.getUri().toString()+"/auth/oauth/token";

    //定义body
    MultiValueMap<String,String> formData = new LinkedMultiValueMap<>();
    //授权方式
    formData.add("grant_type", "password");
    //账号
    formData.add("username",username);
    //密码
    formData.add("password", password);
    //定义头
    MultiValueMap<String,String> header = new LinkedMultiValueMap<>();
    header.add("Authorization", httpbasic(clientId,clientSecret));
    //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
    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);
            }
        }
    });
    Map map = null;
    try {
        //http请求spring security的申请令牌接口

        ResponseEntity<Map> mapResponseEntity = restTemplate.exchange(path, HttpMethod.POST,
```

```
new HttpEntity<MultiValueMap<String, String>>(formData, header), Map.class);
    map = mapResponseEntity.getBody();

    } catch (RestClientException e) {
        e.printStackTrace();
        LOGGER.error("request oauth_token_password error: {}",e.getMessage());
        e.printStackTrace();
        ExceptionCast.cast(AuthCode.AUTH_LOGIN_APPLYTOKEN_FAIL);
    }

    if(map == null ||
        map.get("access_token") == null ||
        map.get("refresh_token") == null ||
        map.get("jti") == null){//jti是jwt令牌的唯一标识作为用户身份令牌
        ExceptionCast.cast(AuthCode.AUTH_LOGIN_APPLYTOKEN_FAIL);
    }
    AuthToken authToken = new AuthToken();
    //访问令牌(jwt)
    String jwt_token = (String) map.get("access_token");
    //刷新令牌(jwt)
    String refresh_token = (String) map.get("refresh_token");
    //jti, 作为用户的身份标识
    String access_token = (String) map.get("jti");
    authToken.setJwt_token(jwt_token);
    authToken.setAccess_token(access_token);
    authToken.setRefresh_token(refresh_token);
    return authToken;
}

//获取httpbasic认证串
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 = Base64.encode(string.getBytes());
    return "Basic "+new String(encode);
}
}
```

## 4.3.6 Controller

AuthController代码如下：

```
@RestController
public class AuthController implements AuthControllerApi {
    @Value("${auth.clientId}")
    String clientId;
    @Value("${auth.clientSecret}")
    String clientSecret;
```



```
@Value("${auth.cookieDomain}")
String cookieDomain;
@Value("${auth.cookieMaxAge}")
int cookieMaxAge;
@Value("${auth.tokenValiditySeconds}")
int tokenValiditySeconds;
@Autowired
AuthService authService;

@Override
@PostMapping("/userlogin")
public LoginResult login(LoginRequest loginRequest) {
    //校验账号是否输入
    if(loginRequest == null || StringUtils.isEmpty(loginRequest.getUsername())){
        ExceptionCast.cast(AuthCode.AUTH_USERNAME_NONE);
    }
    //校验密码是否输入
    if(StringUtils.isEmpty(loginRequest.getPassword())){
        ExceptionCast.cast(AuthCode.AUTH_PASSWORD_NONE);
    }
    AuthToken authToken = authService.login(loginRequest.getUsername(),
loginRequest.getPassword(), clientId, clientSecret);
    //将令牌写入cookie
    //访问token
    String access_token = authToken.getAccess_token();
    //将访问令牌存储到cookie
    saveCookie(access_token);
    return new LoginResult(CommonCode.SUCCESS,access_token);
}

//将令牌保存到cookie
private void saveCookie(String token){
    HttpServletResponse response = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes()).getResponse();
    //添加cookie 认证令牌，最后一个参数设置为false，表示允许浏览器获取
    CookieUtil.addCookie(response, cookieDomain, "/", "uid", token, cookieMaxAge, false);
}

@Override
@PostMapping("/userlogout")
public ResponseResult logout() {
    return null;
}
}
```

### 4.3.7 登录url放行

认证服务默认都要校验用户的身份信息，这里需要将登录url放行。

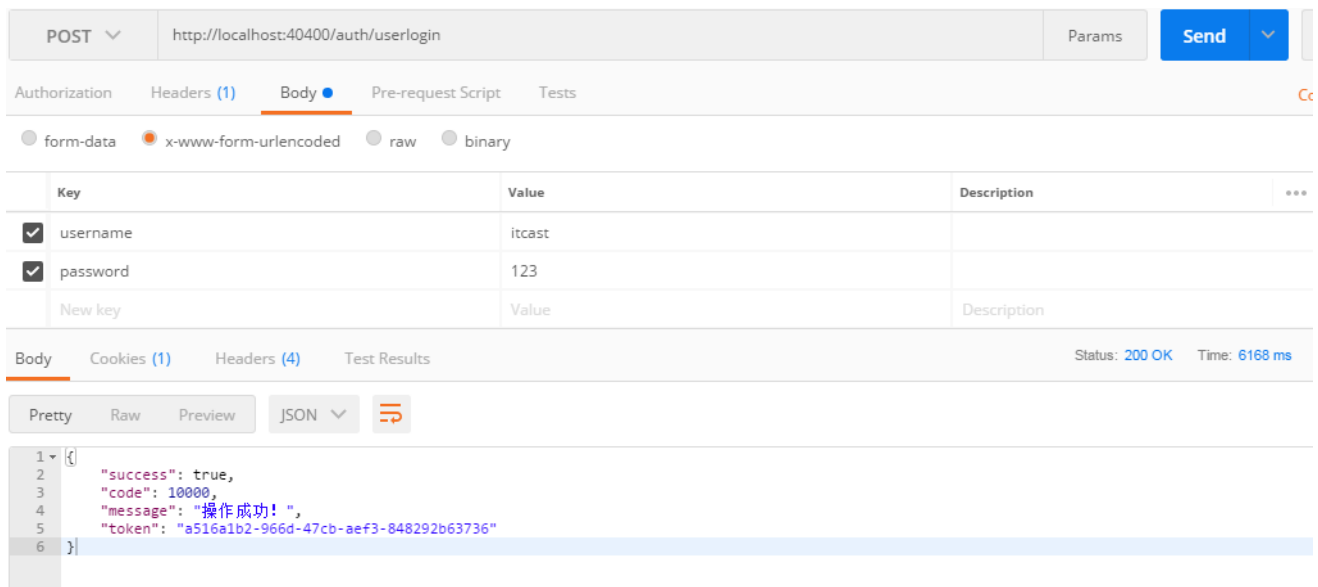
在WebSecurityConfig类中重写 configure(WebSecurity web)方法，如下：

```
@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers("/userlogin");
}
```

### 4.3.8 测试认证接口

使用postman测试：

Post请求：<http://localhost:40400/auth/userlogin>



### 4.3.9 测试写入Cookie

cookie最终会写到xuecheng.com域名下，可通过nginx代理进行认证，测试cookie是否写成功。

cookie还可以写到特定位置？

1、配置nginx代理

可能通过header来写入

在ucenter.xuecheng.com下配置代理路径

```
#认证
location ^~ /openapi/auth/ {
    proxy_pass http://auth_server_pool/auth/;
}
```

添加：

```
#认证服务
upstream auth_server_pool{
    server 127.0.0.1:40400 weight=10;
}
```

2、请求：<http://ucenter.xuecheng.com/openapi/auth/userlogin>

观察cookie写入结果

POST <http://ucenter.xuecheng.com/openapi/auth/userlogin> Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Cookies

form-data x-www-form-urlencoded raw binary

Key	Value	Description
username	itcast	
password	123	
New key	Value	Description

Body Cookies (1) Headers (6) Test Results Status: 200 OK Time: 6132 ms Size: 362 B

Name	Value	Domain	Path	Expires	HTTP	Secure
uid	4ceb0914-e42f-4022-8cc5-8c56098773c0	xuecheng.com	/		false	false

Body Cookies (1) Headers (6) Test Results

Server → nginx/1.14.0

Date → Mon, 08 Jul 2019 06:59:42 GMT

Content-Type → application/json; charset=UTF-8

Transfer-Encoding → chunked

Connection → keep-alive

Set-Cookie → uid=87bc5dd9-5416-4580-8bcb-3de5b1fa1b35; Domain=xuecheng.com; Path=/

```
auth:
  tokenValiditySeconds: 1200 #token存储到redis的过期时间
  clientId: XcWebApp
  clientSecret: XcWebApp
  cookieDomain: xuecheng.com
  cookieMaxAge: -1
encrypt:
```

