

第 5 章 认证授权 v3.1

1 模块需求分析

1.1 什么是认证授权

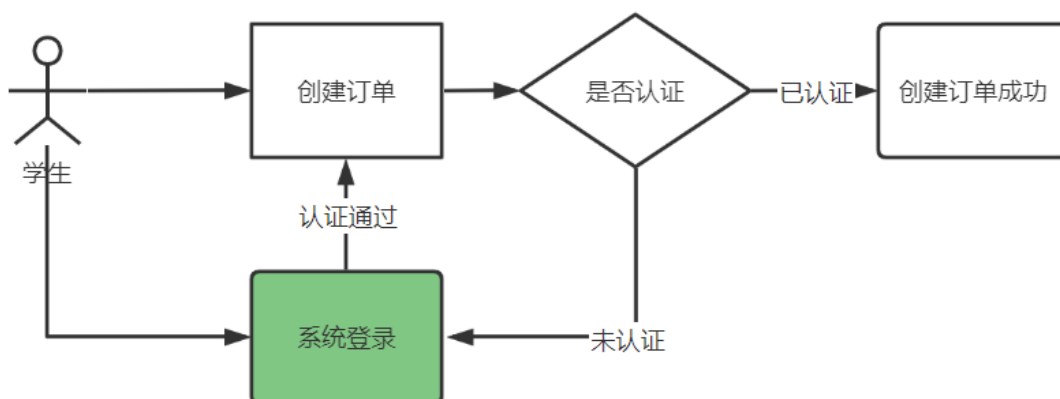
截至目前，项目已经完成了课程发布功能，课程发布后用户通过在线学习页面点播视频进行学习。如何去记录学生的学习过程呢？要想掌握学生的学习情况就需要知道用户的身份信息，记录哪个用户在什么时间学习什么课程，如果用户要购买课程也需要知道用户的身份信息。所以，去管理学生的学习过程最基本的要实现用户的身份认证。

认证授权模块实现平台所有用户的身份认证与用户授权功能。

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的表现形式有：用户名密码登录，微信扫码等方式。

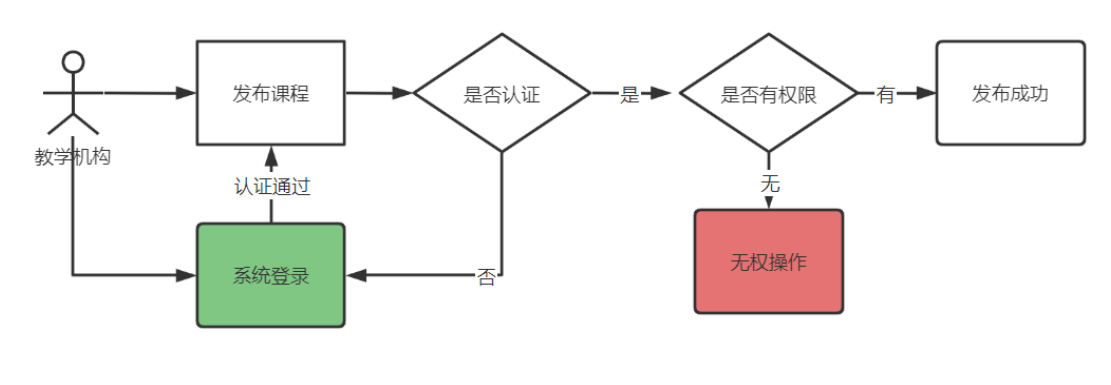
项目包括学生、学习机构的老师、平台运营人员三类用户，不管哪一类用户在访问项目受保护资源时都需要进行身份认证。比如：发布课程操作，需要学习机构的老师首先登录系统成功，然后再执行发布课程操作。创建订单，需要学生用户首先登录系统，才可以创建订单。如下图：



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。比如：用户去发布课程，系统首先进行用户身份认证，认证通过后继续判断用户是否有发布课程的权限，如果没有权限则拒绝继续访问系统，如果有权限则继续发布课程。

如下图：



1.2 业务流程

1.2.1 统一认证

项目包括学生、学习机构的老师、平台运营人员三类用户，三类用户将使用统一的认证入口，如下图：

学成在线 在线有教，学有所成

首页 课程 职业规划

登陆 | 注册 我的学习 教学机构

输入查询关键词 搜索

登录 注册

账户名

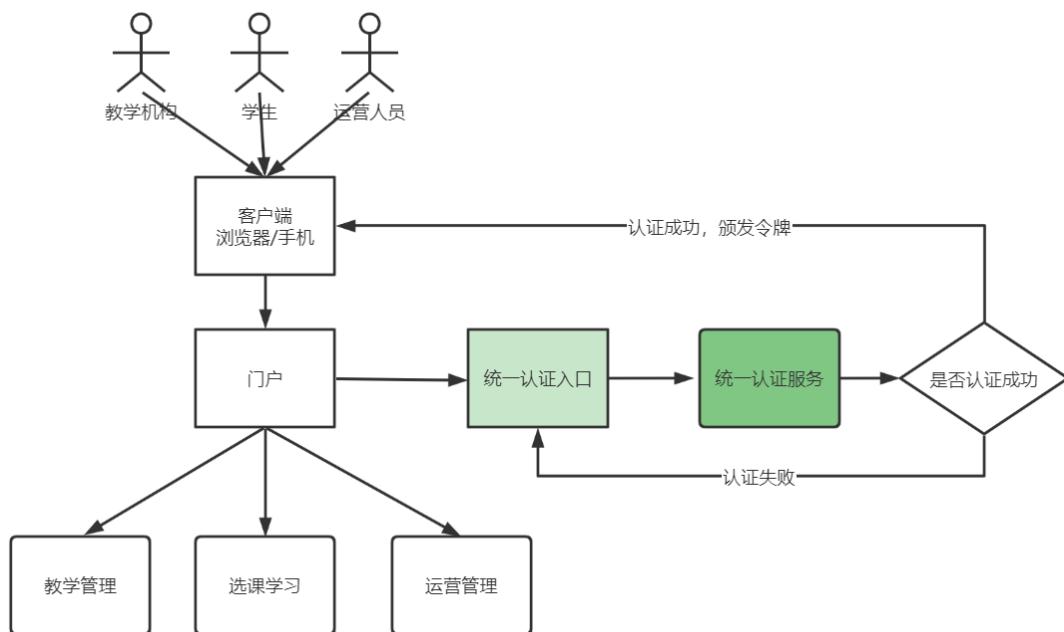
登录密码

☐ 自动登录 忘记密码

登录

用户输入账号和密码提交认证，认证通过则继续操作。

项目由统一认证服务受理用户的认证请求，如下图：



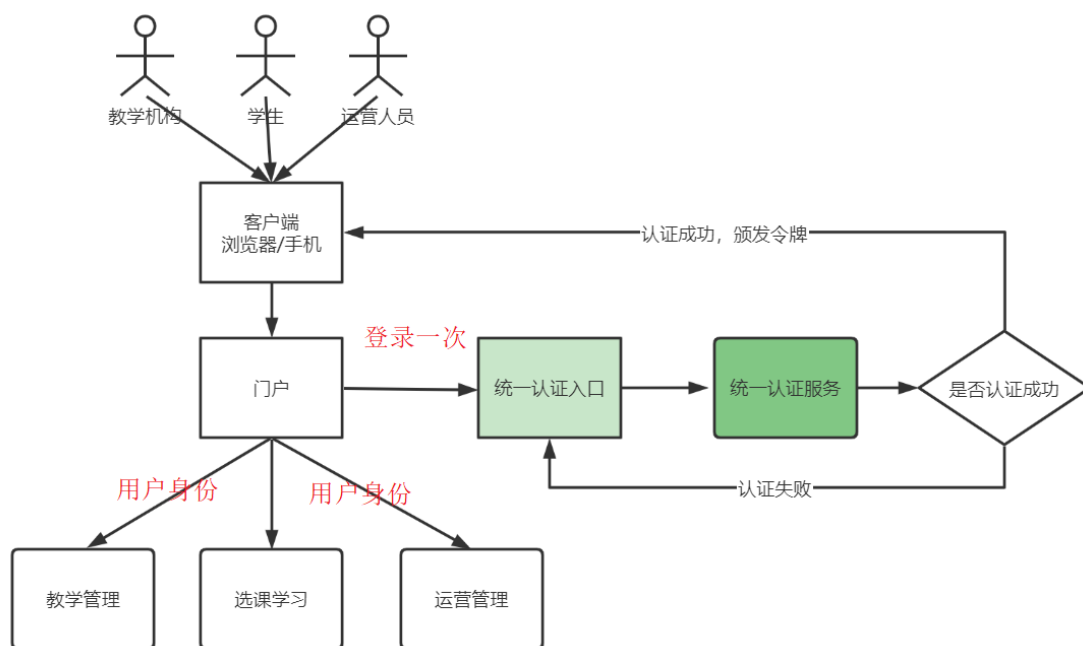
认证通过由认证服务向给用户颁发令牌，相当于访问系统的通行证，用户拿着令牌去访问系统的资源。

1.2.2 单点登录

本项目基于微服务架构构建，微服务包括：内容管理服务、媒资管理服务、学习中心服务、系统管理服务等，为了提高用户体验性，用户只需要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。

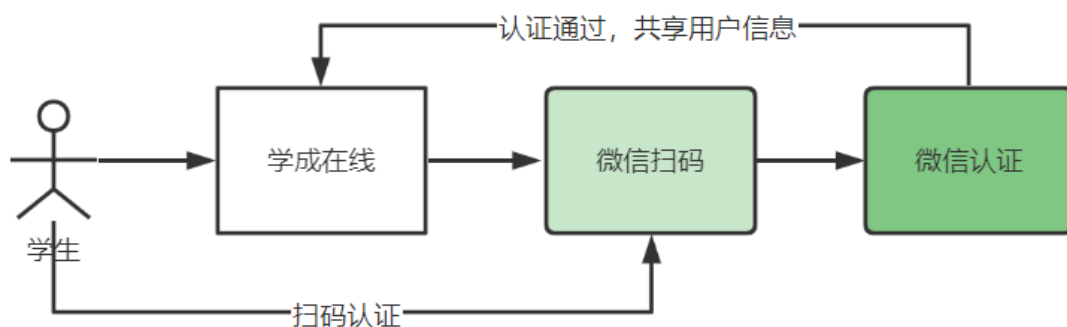
如下图，用户只需要认证一次，便可以在多个拥有访问权限的系统中访问。



1.2.3 第三方认证

为了提高用户体验，很多网站有扫码登录的功能，如：微信扫码登录、QQ 扫码登录等。扫码登录的好处是用户不用输入账号和密码，操作简便，另外一个好处就是有利于用户信息的共享，互联网的优势就是资源共享，用户也是一种资源，对于一个新网站如果让用户去注册是很困难的，如果提供了微信扫码登录将省去用户注册的成本，是一种非常有效的推广手段。

微信扫码登录其中的原理正是使用了第三方认证，如下图：



2 Spring Security 认证研究

2.1 Spring Security 介绍

认证功能几乎是每个项目都要具备的功能，并且它与业务无关，市面上有很多认证框

架，如：Apache Shiro、CAS、Spring Security 等。由于本项目基于 Spring Cloud 技术构建，Spring Security 是 spring 家族的一份子且和 Spring Cloud 集成的很好，所以本项目选用 Spring Security 作为认证服务的技术框架。

Spring Security 是一个功能强大且高度可定制的身份验证和访问控制框架，它是一个专注于为 Java 应用程序提供身份验证和授权的框架。

项目主页：<https://spring.io/projects/spring-security>

Spring cloud Security：<https://spring.io/projects/spring-cloud-security>

2.2 认证授权入门

2.2.1 创建认证服务工程

下边我们使用 Spring Security 框架快速构建认证授权功能体系。

1、部署认证服务工程

从课程资料中拷贝 xuecheng-plus-auth 工程到自己的工程目录下。

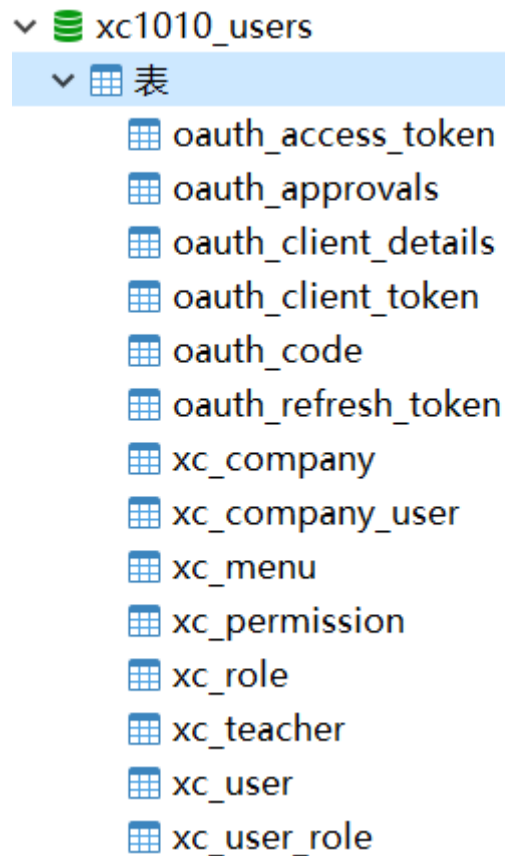
此工程是一个普通的 spring boot 工程，可以连接数据库。

此工程不具备认证授权的功能。

2、创建数据库

创建 xc_users 数据库

导入课程资料中的 xcplus_users.sql 脚本。



在 nacos 中新增 auth-service-dev.yaml:

```
YAML
server:
  servlet:
    context-path: /auth
    port: 63070
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://192.168.101.65:3306/xc1010_users?serverTimezone=UTC&u
serUnicode=true&useSSL=false&
    username: root
    password: mysql
```

初始工程自带了一个 Controller 类，如下：

```
Java
package com.xuecheng.auth.controller;
```

```

/**
 * @author Mr.M
 * @version 1.0
 * @description 测试 controller
 * @date 2022/9/27 17:25
 */
@Slf4j
@RestController
public class LoginController {

    @Autowired
    XcUserMapper userMapper;

    @RequestMapping("/login-success")
    public String loginSuccess(){

        return "登录成功";
    }

    @RequestMapping("/user/{id}")
    public XcUser getuser(@PathVariable("id") String id){
        XcUser xcUser = userMapper.selectById(id);
        return xcUser;
    }

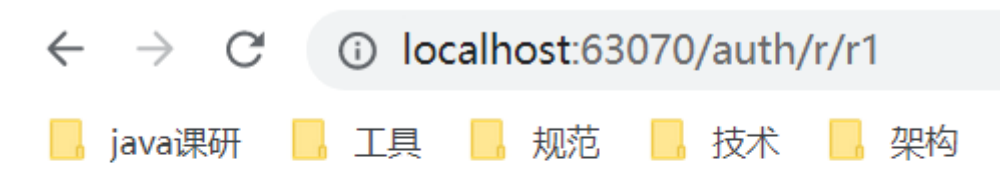
    @RequestMapping("/r/r1")
    public String r1(){
        return "访问 r1 资源";
    }

    @RequestMapping("/r/r2")
    public String r2(){
        return "访问 r2 资源";
    }

}

```

启动工程，尝试访问 <http://localhost:63070/auth/r/r1> :



访问r1资源

访问用户信息: <http://localhost:63070/auth/user/52>



以上测试一切正常说明此工程部署成功。

2.2.2 认证测试

下边向 auth 认证工程集成 Spring security, 向 pom.xml 加入 Spring Security 所需要的依赖

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

重启工程, 访问 <http://localhost:63070/auth/r/r1>

自动进入/login 登录页面, /login 是 spring security 提供的,此页面有几个 css 样式加载会稍微慢点, 如下图:

Please sign in

Username

Password

Sign in

账号和密码是多少呢？下一步需要进行安全配置。

拷贝课程资料下的 WebSecurityConfig.java 到 config 下需要三部分内容：

1、用户信息

在内存配置两个用户：zhangsan、lisi

zhangsan 用户拥有的权限为 p1

lisi 用户拥有的权限为 p2

2、密码方式

暂时采用明文方式

3、安全拦截机制

/r/**开头的请求需要认证

登录成功到成功页面

代码如下：

```
Java
//配置用户信息服务
@Bean
public UserDetailsService userDetailsService() {
    //这里配置用户信息,这里暂时使用这种方式将用户存储在内存中
    InMemoryUserDetailsManager manager = new
```

```

InMemoryUserDetailsManager();

manager.createUser(User.withUsername("zhangsan").password("123").authorities("p1").build());

manager.createUser(User.withUsername("lisi").password("456").authorities("p2").build());
    return manager;
}

@Bean
public PasswordEncoder passwordEncoder() {
    //密码为明文方式
    return NoOpPasswordEncoder.getInstance();
}

//配置安全拦截机制
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .antMatchers("/r/**").authenticated()//访问/r 开始的请求需要认证通过
        .anyRequest().permitAll()//其它请求全部放行
        .and()
        .formLogin().successForwardUrl("/login-success");//登录成功跳转到/login-success
        http.logout().logoutUrl("/logout");//退出地址
    }
}

```

重启工程

1、访问 <http://localhost:63070/auth/user/52> 可以正常访问

2、访问 <http://localhost:63070/auth/r/r1> 显示登录页面

账号 zhangsan，密码为 123，如果输入的密码不正确会认证失败，输入正确显示登录成功。

为什么 <http://localhost:63070/auth/user/52> 可以正常访问，访问 <http://localhost:63070/auth/r/r1> 显示登录页面？

`http.logout().logoutUrl("/logout");`配置了退出页面，认证成功后访问/logout 可退出登录。

2.2.3 授权测试

用户认证通过去访问系统资源时 spring security 进行授权控制，判断用户是否有该资源的访问权限，如果有则继续访问，如果没有则拒绝访问。

下边测试授权功能：

1、配置用户拥有哪些权限。

在 WebSecurityConfig 类配置 zhangsan 拥有 p1 权限，lisi 拥有 p2 权限。

```
Java
@Bean
public UserDetailsService userDetailsService() {
    //这里配置用户信息,这里暂时使用这种方式将用户存储在内存中
    InMemoryUserDetailsManager manager = new
InMemoryUserDetailsManager();

    manager.createUser(User.withUsername("zhangsan").password("123").a
uthorities("p1").build());

    manager.createUser(User.withUsername("lisi").password("456").autho
rities("p2").build());
    return manager;
}
```

2、指定资源与权限的关系。

什么是系统的资源？

比如：查询一个用户的信息，用户信息就是系统的资源，要访问资源需要通过 URL，所以我们在 controller 中定义的每个 http 的接口就是访问资源的接口。

下边在 controller 中配置/r/r1 需要 p1 权限，/r/r2 需要 p2 权限。

hasAuthority('p1')表示拥有 p1 权限方可访问。

代码如下：

```
Java
@RestController
public class LoginController {
    ....
    @RequestMapping("/r/r1")
    @PreAuthorize("hasAuthority('p1')")//拥有 p1 权限方可访问
    public String r1(){
        return "访问 r1 资源";
    }
}
```

```

}

@RequestMapping("/r/r2")
@PreAuthorize("hasAuthority('p2')")//拥有 p2 权限方可访问
public String r2(){
    return "访问 r2 资源";
}
...

```

现在重启工程。

当访问以/r/开头的 url 时会判断用户是否认证，如果没有认证则跳转到登录页面，如果已经认证则判断用户是否具有该 URL 的访问权限，如果具有该 URL 的访问权限则继续，否则拒绝访问。

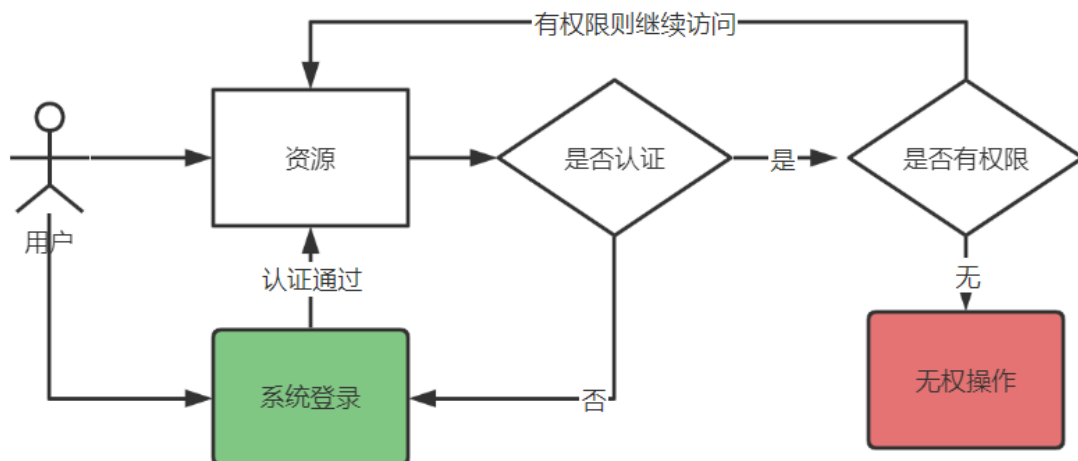
例如：

访问/r/r1，使用 zhangsan 登录可以正常访问，因为在/r/r1 的方法上指定了权限 p1，zhangsan 用户拥有权限 p1,所以可以正常访问。

访问/r/r1，使用 lisi 登录则拒绝访问，由于 lisi 用户不具有权限 p1 需要拒绝访问

注意：如果访问上不加@PreAuthorize，此方法没有授权控制。

整理授权的过程见下图所示：

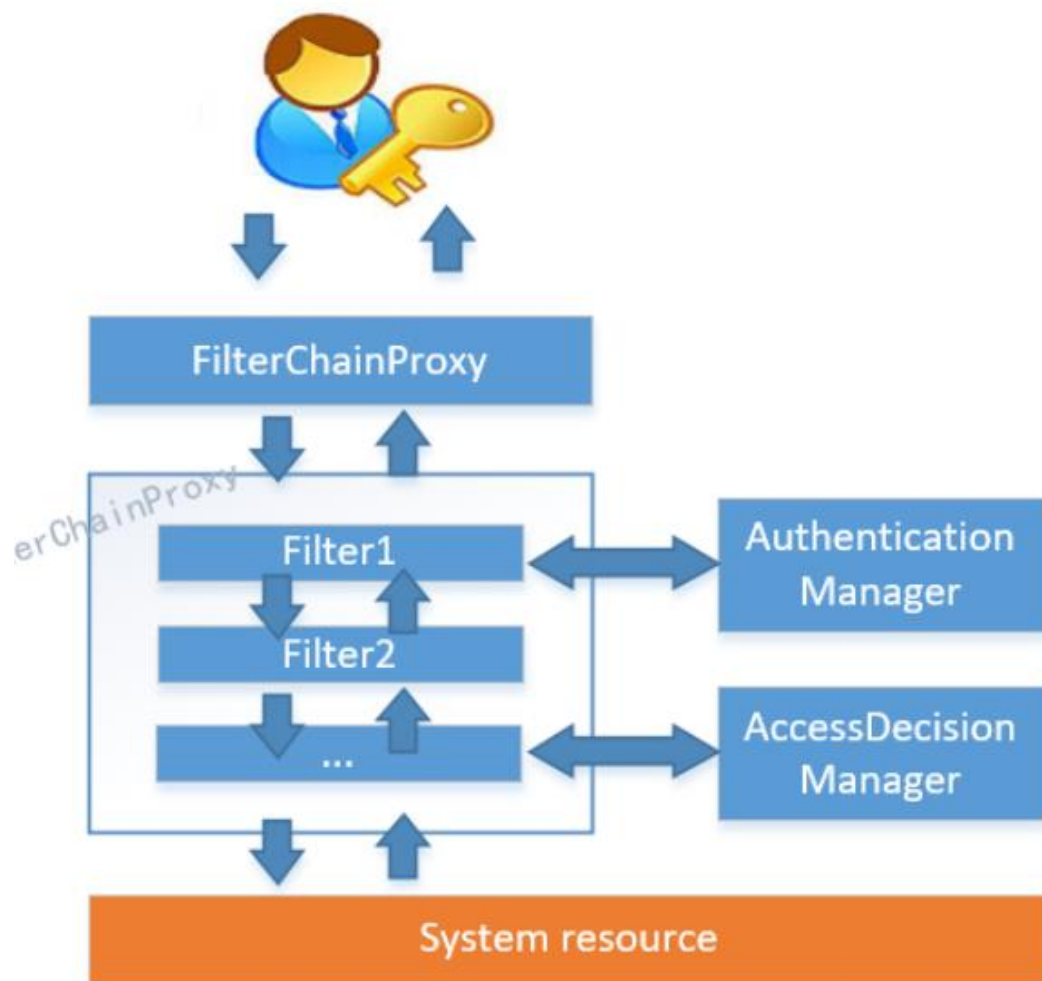


2.2.4 工作原理

通过测试认证和授权两个功能，我们了解了 Spring Security 的基本使用方法，下边了解它的工作流程。

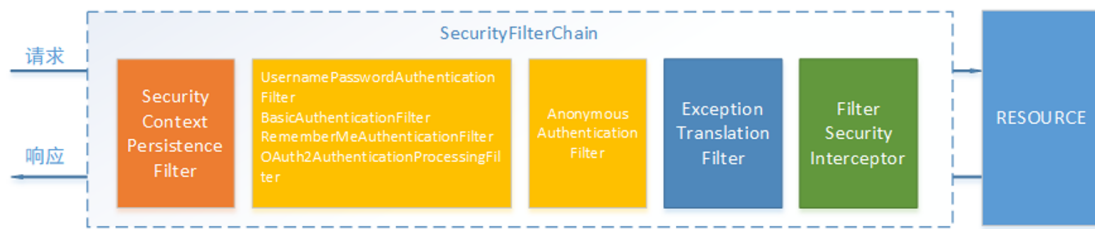
Spring Security 所解决的问题就是**安全访问控制**，而安全访问控制功能其实就是对所有进入系统的请求进行拦截，校验每个请求是否能够访问它所期望的资源。根据前边知识的学习，可以通过 Filter 或 AOP 等技术来实现，Spring Security 对 Web 资源的保护是靠 Filter 实现的，所以从这个 Filter 来入手，逐步深入 Spring Security 原理。

当初初始化 Spring Security 时，会创建一个名为 `SpringSecurityFilterChain` 的 Servlet 过滤器，类型为 `org.springframework.security.web.FilterChainProxy`，它实现了 `javax.servlet.Filter`，因此外部的请求会经过此类，下图是 Spring Security 过滤器链结构图：



`FilterChainProxy` 是一个代理，真正起作用的是 `FilterChainProxy` 中 `SecurityFilterChain` 所包含的各个 Filter，同时这些 Filter 作为 Bean 被 Spring 管理，它们是 Spring Security 核心，各有各的职责，但他们并不直接处理用户的**认证**，也不直接处理用户的**授权**，而是把它们交给了认证管理器（`AuthenticationManager`）和决策管理器（`AccessDecisionManager`）进行处理。

spring Security 功能的实现主要是由一系列过滤器链相互配合完成。



下面介绍过滤器链中主要的几个过滤器及其作用：

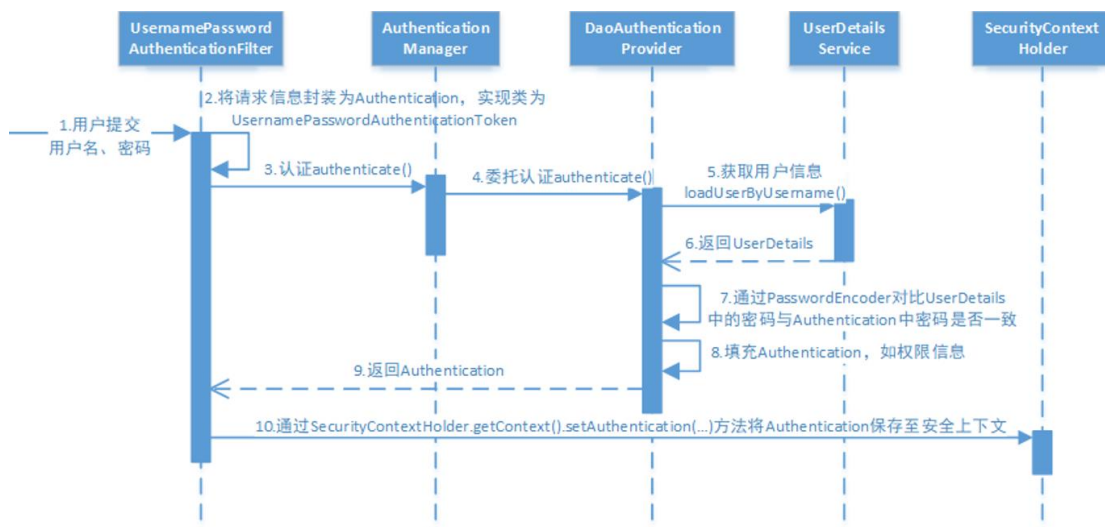
SecurityContextPersistenceFilter 这个 Filter 是整个拦截过程的入口和出口（也就是第一个和最后一个拦截器），会在请求开始时从配置好的 **SecurityContextRepository** 中获取 **SecurityContext**，然后把它设置给 **SecurityContextHolder**。在请求完成后将 **SecurityContextHolder** 持有的 **SecurityContext** 再保存到配置好的 **SecurityContextRepository**，同时清除 **securityContextHolder** 所持有的 **SecurityContext**；

UsernamePasswordAuthenticationFilter 用于处理来自表单提交的认证。该表单必须提供对应的用户名和密码，其内部还有登录成功或失败后进行处理的 **AuthenticationSuccessHandler** 和 **AuthenticationFailureHandler**，这些都可以根据需求做相关改变；

FilterSecurityInterceptor 是用于保护 web 资源的，使用 **AccessDecisionManager** 对当前用户进行授权访问，前面已经详细介绍过了；

ExceptionTranslationFilter 能够捕获来自 **FilterChain** 所有的异常，并进行处理。但是它只会处理两类异常：**AuthenticationException** 和 **AccessDeniedException**，其它的异常它会继续抛出。

Spring Security 的执行流程如下：



1. 用户提交用户名、密码被 **SecurityFilterChain** 中的 **UsernamePasswordAuthenticationFilter** 过滤器获取到，封装为请求

Authentication，通常情况下是 UsernamePasswordAuthenticationToken 这个实现类。

2. 然后过滤器将 Authentication 提交至认证管理器（AuthenticationManager）进行认证

3. 认证成功后，AuthenticationManager 身份管理器返回一个被填充满了信息的（包括上面提到的权限信息，身份信息，细节信息，但密码通常会被移除）

Authentication 实例。

4. SecurityContextHolder 安全上下文容器将第 3 步填充了信息的 Authentication，通过 SecurityContextHolder.getContext().setAuthentication(...)方法，设置到其中。

5. 可以看出 AuthenticationManager 接口（认证管理器）是认证相关的核心接口，也是发起认证的出发点，它的实现类为 ProviderManager。而 Spring Security 支持多种认证方式，因此 ProviderManager 维护着一个

List<AuthenticationProvider>列表，存放多种认证方式，最终实际的认证工作是由 AuthenticationProvider 完成的。咱们知道 web 表单的对应的 AuthenticationProvider 实现类为 DaoAuthenticationProvider，它的内部又维护着一个 UserDetailsService 负责 UserDetails 的获取。最终 AuthenticationProvider 将 UserDetails 填充至 Authentication。

2.3 什么是 OAuth2

2.3.1 OAuth2 认证流程

在前边我们提到微信扫码认证，这是一种第三方认证的方式，这种认证方式是基于 OAuth2 协议实现，

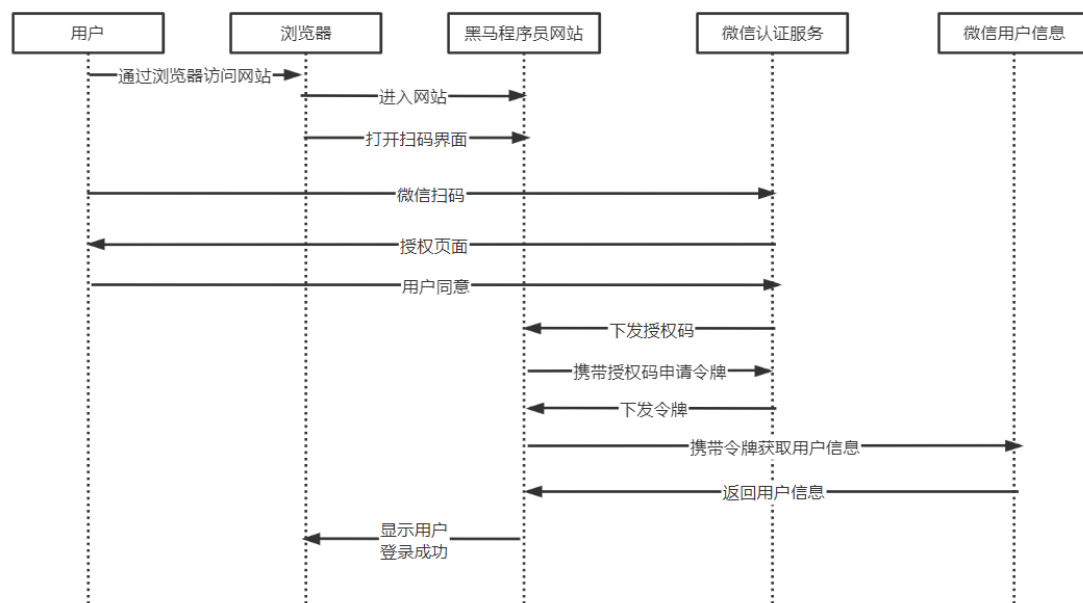
OAUTH 协议为用户资源的授权提供了一个安全的、开放而又简易的标准。同时，任何第三方都可以使用 OAUTH 认证服务，任何服务提供商都可以实现自身的 OAUTH 认证服务，因而 OAUTH 是开放的。业界提供了 OAUTH 的多种实现如 PHP、JavaScript，Java，Ruby 等各种语言开发包，大大节约了程序员的时间，因而 OAUTH 是简易的。互联网很多服务如 Open API，很多大公司如 Google，Yahoo，Microsoft 等都提供了 OAUTH 认证服务，这些都足以说明 OAUTH 标准逐渐成为开放资源授权的标准。

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扫码登录的过程:



具体流程如下:

1、用户点击微信扫码

用户进入黑马程序的登录页面, 点击微信的图标开打微信扫码界面。





微信扫码的目的是通过微信认证登录黑马程序员官网，黑马程序员网站需要从微信获取当前用户的身份信息才会让当前用户在黑马网站登录成功。

现在搞清楚几个概念：

资源：用户信息，在微信中存储。

资源拥有者：用户是用户信息资源的拥有者。

认证服务：微信负责认证当前用户的身份，负责为客户端颁发令牌。

客户端：客户端会携带令牌请求微信获取用户信息，黑马程序员网站即客户端，黑马网站需要在浏览器打开。

2、用户授权黑马网站访问用户信息

资源拥有者扫描二维码表示资源拥有者请求微信进行认证，微信认证通过向用户手机

返回授权页面，如下图：



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

询问用户是否授权黑马程序员访问自己在微信的用户信息，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给黑马程序员的网站。

只有资源拥有者同意微信才允许黑马网站访问资源。

3、黑马程序员的网站获取到授权码

4、携带授权码请求微信认证服务器申请令牌

此交互过程用户看不到。

5、微信认证服务器向黑马程序员的网站响应令牌

此交互过程用户看不到。

6、黑马程序员网站请求微信资源服务器获取资源即用户信息。

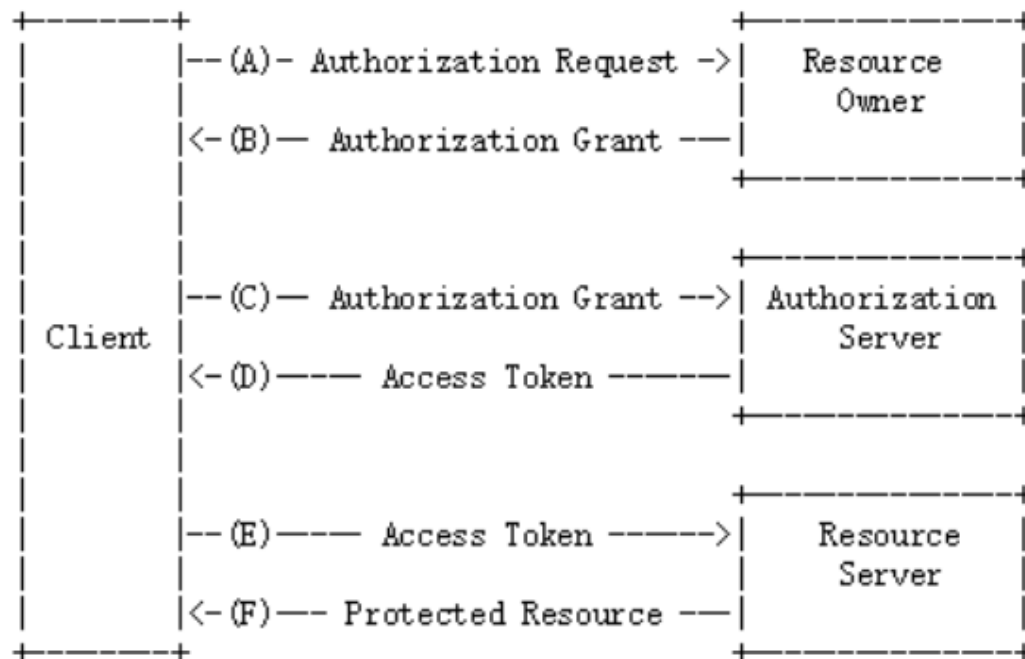
黑马程序员网站携带令牌请求访问微信服务器获取用户的基本信息。

7、资源服务器返回受保护资源即用户信息

8、黑马网站接收到用户信息，此时用户在黑马网站登录成功。

理解了微信扫码登录黑马网站的流程，接下来认识 OAuth2.0 的认证流程，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：手机客户端、浏览器等。

上边示例中黑马网站即为客户端，它需要通过浏览器打开。

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。

A 表示 客户端请求资源拥有者授权。

B 表示 资源拥有者授权客户端即黑马网站访问自己的用户信息。

3、授权服务器（也称认证服务器）

认证服务器对资源拥有者进行认证，还会对客户端进行认证并颁发令牌。

C 客户端即黑马网站携带授权码请求认证。

D 认证通过颁发令牌。

4、资源服务器

存储资源的服务器。

E 表示客户端即黑马网站携带令牌请求资源服务器获取资源。

F 表示资源服务器校验令牌通过后提供受保护资源。

2.3.2 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的要求去使用 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。

本项目要接入微信扫码登录所以本项目要使用 OAuth2 协议访问微信中的用户信息。

2、外部系统访问学成在线的资源。

同样当第三方系统想要访问学成在线网站的资源也可以基于 OAuth2 协议。

3、学成在线前端（客户端）访问学成在线微服务的资源。

本项目是前后端分离架构，前端访问微服务资源也可以基于 OAuth2 协议进行认证。

2.3.3 OAuth2 的授权模式

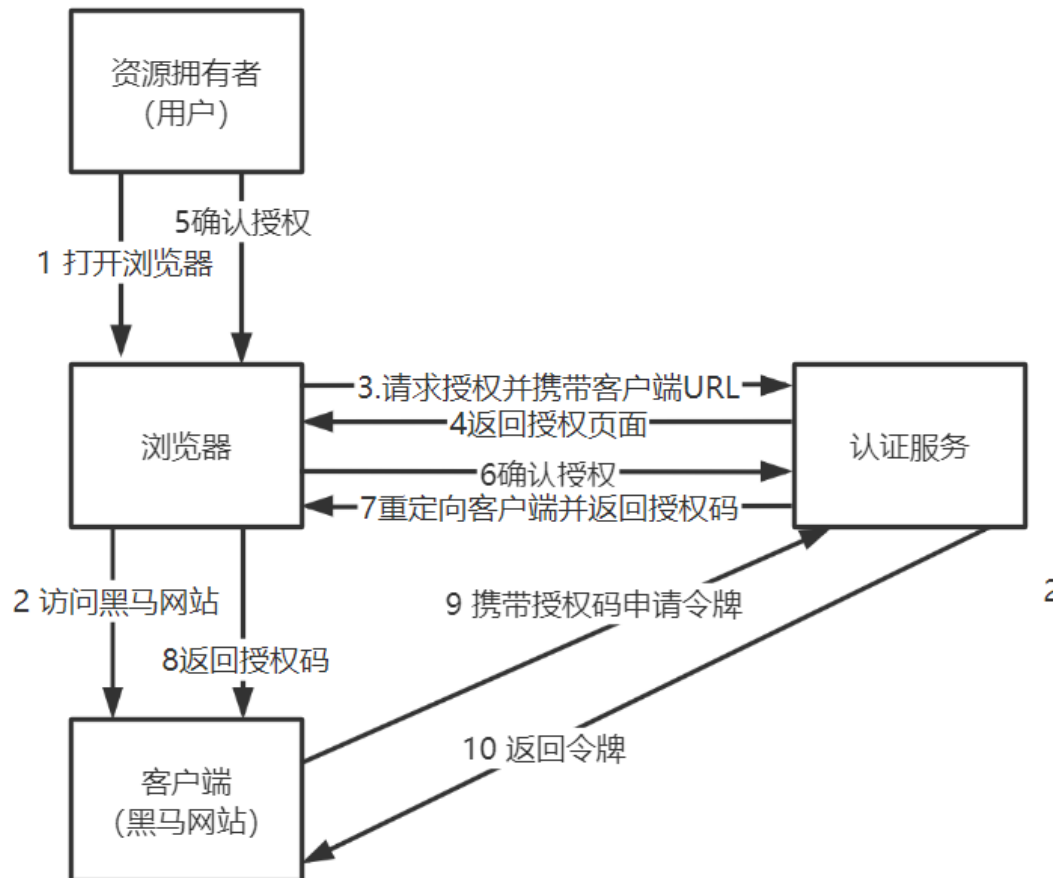
Spring Security 支持 OAuth2 认证，OAuth2 提供授权码模式、密码模式、简化模式、客户端模式等四种授权模式，前边举的微信扫码登录的例子就是基于授权码模式，这四种模式中授权码模式和密码模式应用较多，本节使用 Spring Security 演示授权码模式、密码模式，其余两种请自行查阅相关资料。

2.3.3.1 授权码模式

OAuth2 的几个授权模式是根据不同的应用场景以不同的方式去获取令牌，最终目的是要获取认证服务颁发的令牌，最终通过令牌去获取资源。

授权码模式简单理解是使用授权码去获取令牌，要想获取令牌先要获取授权码，授权码的获取需要资源拥有者亲自授权同意才可以获取。

下图是授权码模式的交互图：



还以黑马网站微信扫码登录为例进行说明：

- 1、用户打开浏览器。
- 2、通过浏览器访问客户端即黑马网站。
- 3、用户通过浏览器向认证服务请求授权，请求授权时会携带客户端的 URL，此 URL 为下发授权码的重定向地址。
- 4、认证服务向资源拥有者返回授权页面。
- 5、资源拥有者亲自授权同意。
- 6、通过浏览器向认证服务发送授权同意。
- 7、认证服务向客户端地址重定向并携带授权码。
- 8、客户端即黑马网站收到授权码。
- 9、客户端携带授权码向认证服务申请令牌。
- 10、认证服务向客户端颁发令牌。

2.3.3.2 授权码模式测试

要想测试授权模式首先要配置授权服务器即上图中的认证服务器，需要配置授权服务及令牌策略。

1、从课程资料中拷贝 AuthorizationServer.java、TokenConfig.java 到认证服务的 config 包下。

说明“：AuthorizationServer 用 @EnableAuthorizationServer 注解标识并继承 AuthorizationServerConfigurerAdapter 来配置 OAuth2.0 授权服务器。

```
Java
package com.xuecheng.auth.config;
*/
@Configuration
@EnableAuthorizationServer
public class AuthorizationServer extends
AuthorizationServerConfigurerAdapter {
    ...
}
```

AuthorizationServerConfigurerAdapter 要求配置以下几个类：

```
Java
public class AuthorizationServerConfigurerAdapter implements
AuthorizationServerConfigurer {
    public AuthorizationServerConfigurerAdapter() {}
    public void configure(AuthorizationServerSecurityConfigurer
security) throws Exception {}
    public void configure(ClientDetailsServiceConfigurer clients)
throws Exception {}
    public void configure(AuthorizationServerEndpointsConfigurer
endpoints) throws Exception {}
}
```

1) ClientDetailsServiceConfigurer：用来配置客户端详情服务 (ClientDetailsService)，

随便一个客户端都可以随便接入到它的认证服务吗？答案是否定的，服务提供商会给批准接入的客户端一个身份，用于接入时的凭据，有客户端标识和客户端密钥，在这里配置批准接入的客户端的详细信息。

2) AuthorizationServerEndpointsConfigurer：用来配置令牌 (token) 的访问端点和令牌服务(token services)。

3) AuthorizationServerSecurityConfigurer：用来配置令牌端点的安全约束。

2、TokenConfig 为令牌策略配置类

暂时先使用 InMemoryTokenStore 在内存存储令牌，令牌的有效期等信息配置如下：

Java

```
// 令牌管理服务
@Bean(name="authorizationServerTokenServicesCustom")
public AuthorizationServerTokenServices tokenService() {
    DefaultTokenServices service=new DefaultTokenServices();
    service.setSupportRefreshToken(true); // 支持刷新令牌
    service.setTokenStore(tokenStore); // 令牌存储策略
    service.setAccessTokenValiditySeconds(7200); // 令牌默认有
    效期2 小时
    service.setRefreshTokenValiditySeconds(259200); // 刷新令牌
    默认有效期3 天
    return service;
}
```

3、配置认证管理 bean

Java

```
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true,prePostEnabled =
true)
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {

    @Bean
    public AuthenticationManager authenticationManagerBean()
throws Exception {
        return super.authenticationManagerBean();
    }
    ....
}
```

重启认证服务

1、get 请求获取授权码

地址：

http://localhost:63070/auth/oauth/authorize?client_id=XcWebApp&response_type=code&scope=all&redirect_uri=http://www.51xuecheng.cn

参数列表如下：

- client_id: 客户端准入标识。
- response_type: 授权码模式固定为 code。

- scope: 客户端权限。
- 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)。

输入账号 zhangsan、密码 123 登录成功, 输入

/oauth/authorize?client_id=XcWebApp&response_type=code&scope=all&redirect_uri=http://www.51xuecheng.cn

显示授权页面

OAuth Approval

Do you authorize "XcWebApp" to access your protected resources?

- scope.all: ☒ Approve ☐ Deny

Authorize

授权“XcWebApp”访问自己的受保护资源？

选择同意。

2、请求成功, 重定向至 <http://www.51xuecheng.cn/?code=授权码>, 比如:

<http://www.51xuecheng.cn/?code=Wqjb5H>

3、使用 httpclient 工具 post 申请令牌

/oauth/token?client_id=XcWebApp&client_secret=XcWebApp&grant_type=authorization_code&code=授权码&redirect_uri=http://www.51xuecheng.cn/

参数列表如下

- client_id: 客户端准入标识。
- client_secret: 客户端密钥。
- 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。

httpclient 脚本如下:

```
Plain Text
### 授权码模式
```



```
### 第一步申请授权码(浏览器请求)/oauth/authorize?client_id=c1&response_type=code&scope=all&redirect_uri=http://www.51xuecheng.cn
### 第二步申请令牌
POST
{{auth_host}}/auth/oauth/token?client_id=XcWebApp&client_secret=XcWebApp&grant_type=authorization_code&code=CTvCrB&redirect_uri=http://www.51xuecheng.cn
```

申请令牌成功如下所示：

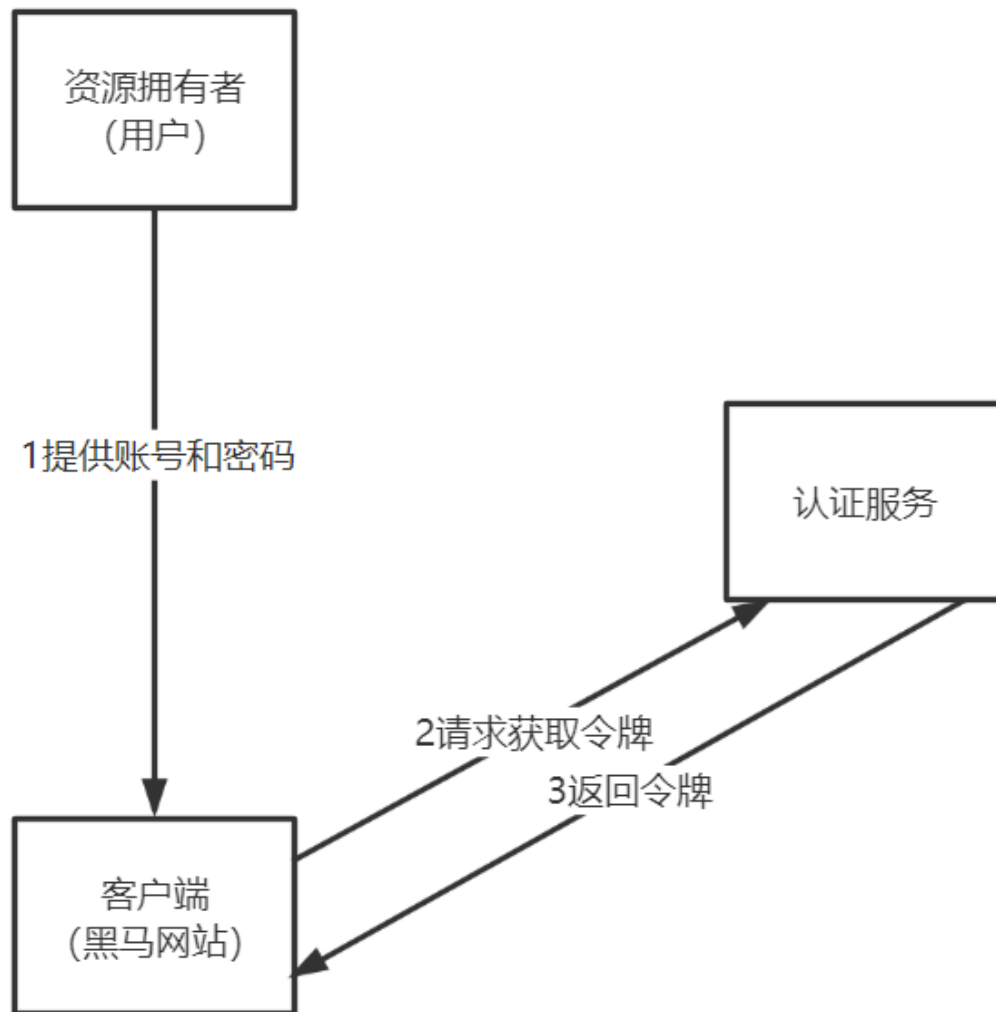
```
JSON
{
  "access_token": "368b1ee7-a9ee-4e9a-aae6-0fcab243aad2",
  "token_type": "bearer",
  "refresh_token": "3d56e139-0ee6-4ace-8cbe-1311dfaa991f",
  "expires_in": 7199,
  "scope": "all"
}
```

说明：

- 1、access_token，访问令牌，用于访问资源使用。
- 2、token_type，bearer 是在 RFC6750 中定义的一种 token 类型，在携带令牌访问资源时需要在 head 中加入 bearer 空格 令牌内容
- 3、refresh_token，当令牌快过期时使用刷新令牌可以再次生成令牌。
- 4、expires_in：过期时间（秒）
- 5、scope，令牌的权限范围，服务端可以根据令牌的权限范围去对令牌授权。

2.3.3.3 密码模式

密码模式相对授权码模式简单，授权码模式需要借助浏览器供用户亲自授权，密码模式不用借助浏览器，如下图：



- 1、资源拥有者提供账号和密码
- 2、客户端向认证服务申请令牌，请求中携带账号和密码
- 3、认证服务校验账号和密码正确颁发令牌。

开始测试：

1、POST 请求获取令牌

`/oauth/token?client_id=XcWebApp&client_secret=XcWebApp&grant_type=password
&username=shangsan&password=123`

参数列表如下：

- `client_id`：客户端准入标识。
- `client_secret`：客户端密钥。
- `grant_type`：授权类型，填写 `password` 表示密码模式
- `username`：资源拥有者用户名。

- password: 资源拥有者密码。

2、授权服务器将令牌 (access_token) 发送给 client

使用 httpclient 进行测试

Java

密码模式

POST

```
{{auth_host}}/auth/oauth/token?client_id=XcWebApp&client_secret=XcWebApp&grant_type=password&username=zhangsan&password=123
```

返回示例:

JSON

```
{
  "access_token": "368b1ee7-a9ee-4e9a-aae6-0fcab243aad2",
  "token_type": "bearer",
  "refresh_token": "3d56e139-0ee6-4ace-8cbe-1311dfaa991f",
  "expires_in": 6806,
  "scope": "all"
}
```

这种模式十分简单，但是却意味着直接将用户敏感信息泄漏给了 client，因此这就说明这种模式只能用于 client 是我们自己开发的情况下。

2.3.3.4 本项目的应用方式

通过演示授权码模式和密码模式，授权码模式适合客户端和认证服务非同一个系统的情况，所以本项目使用授权码模式完成微信扫码认证。本项目采用密码模式作为前端请求微服务的认证方式。

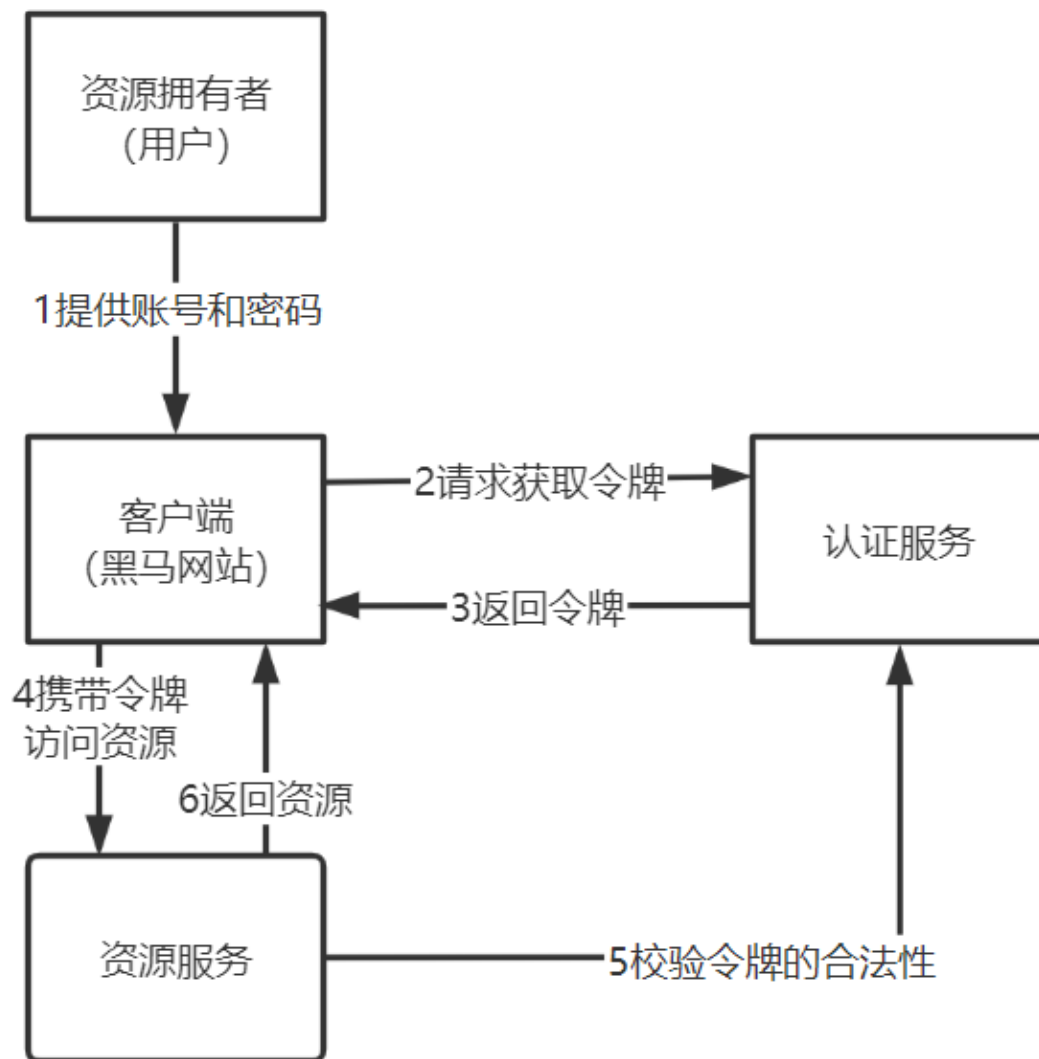
2.4 JWT

2.4.1 普通令牌的问题

客户端申请到令牌，接下来客户端携带令牌去访问资源，到资源服务器将会校验令牌的合法性。

资源服务器如何校验令牌的合法性？

我们以 OAuth2 的密码模式为例进行说明：



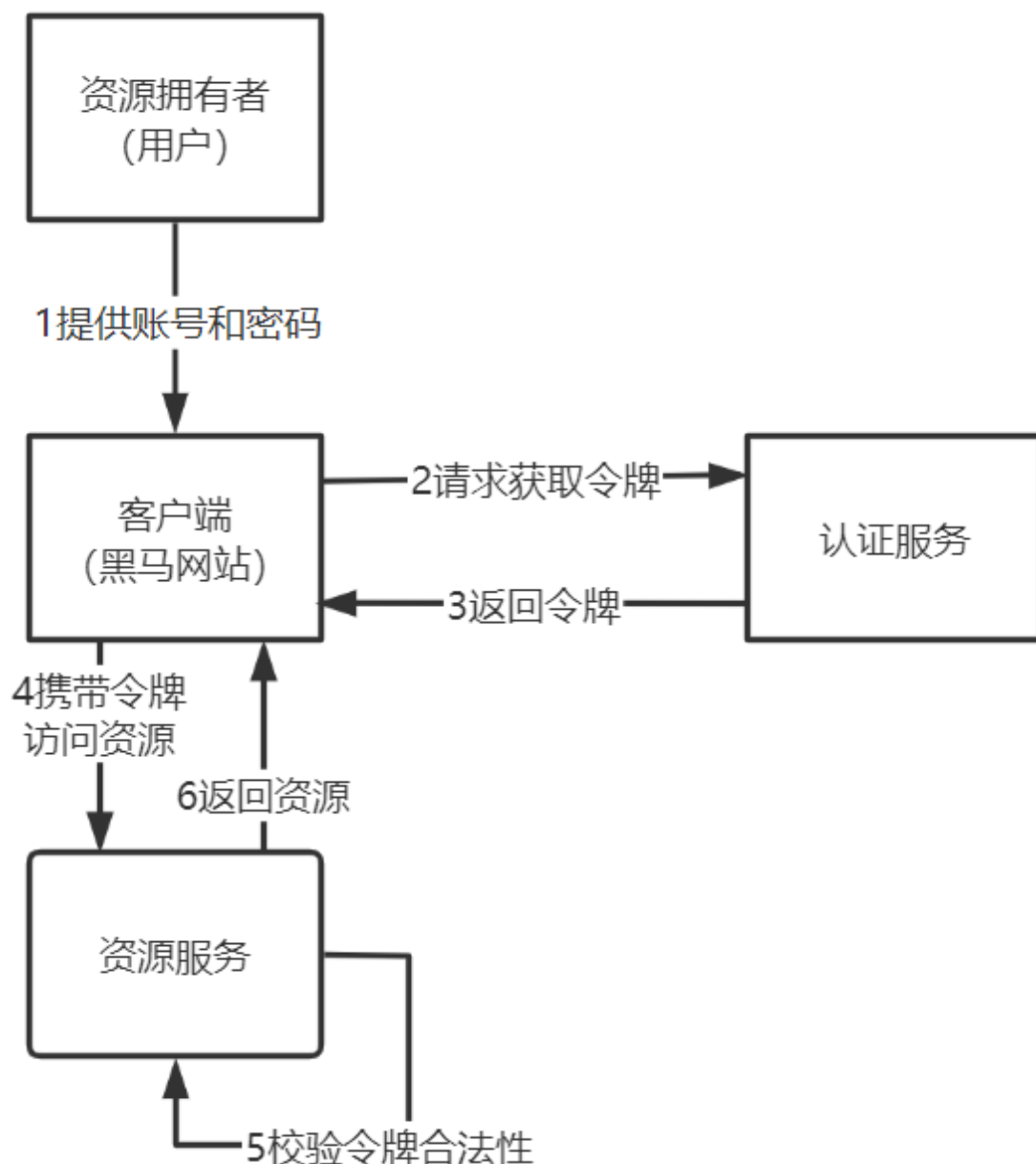
从第 4 步开始说明：

- 1、客户端携带令牌访问资源服务获取资源。
- 2、资源服务远程请求认证服务校验令牌的合法性
- 3、如果令牌合法资源服务向客户端返回资源。

这里存在一个问题：

就是校验令牌需要远程请求认证服务，客户端的每次访问都会远程校验，执行性能低。

如果能够让资源服务自己校验令牌的合法性将省去远程请求认证服务的成本，提高了性能。如下图：



如何解决上边的问题，实现资源服务自行校验令牌。

令牌采用 JWT 格式即可解决上边的问题，用户认证通过后会得到一个 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。

2.4.2 什么是 JWT

什么是 JWT？

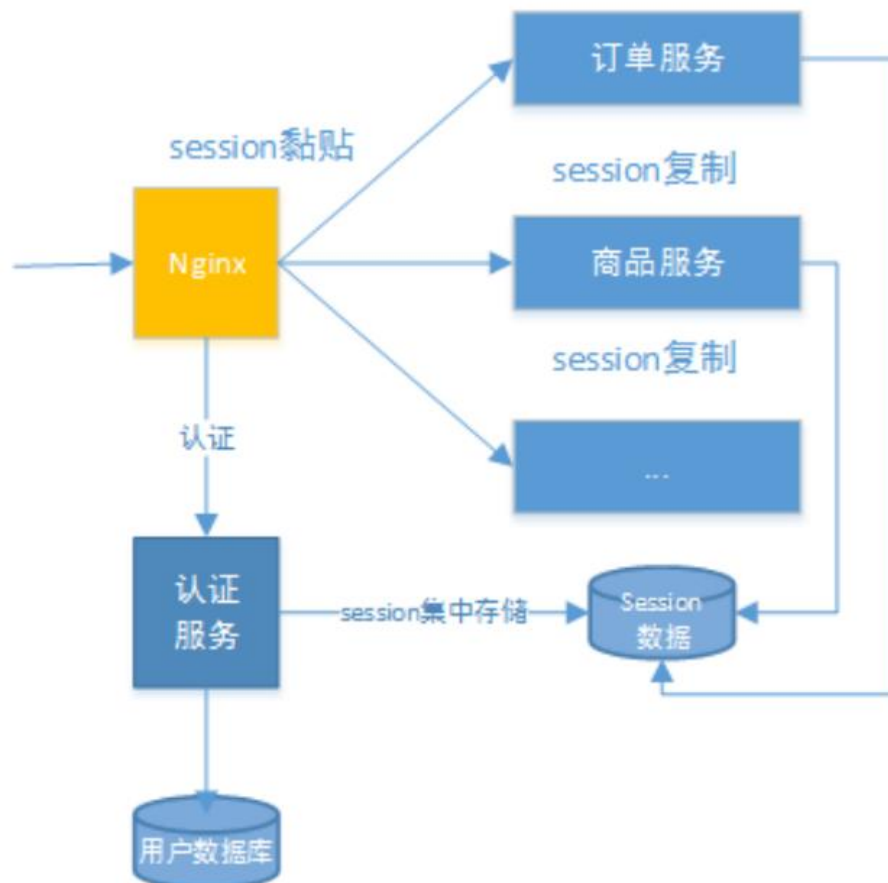
JSON Web Token (JWT) 是一种使用 JSON 格式传递数据的网络令牌技术，它是一个开放的行业标准 (RFC 7519)，它定义了一种简洁的、自包含的协议格式，用于在通信双方传递 json 对象，传递的信息经过数字签名可以被验证和信任，它可以使用

HMAC 算法或使用 RSA 的公钥/私钥对来签名，防止内容篡改。官网：<https://jwt.io/>

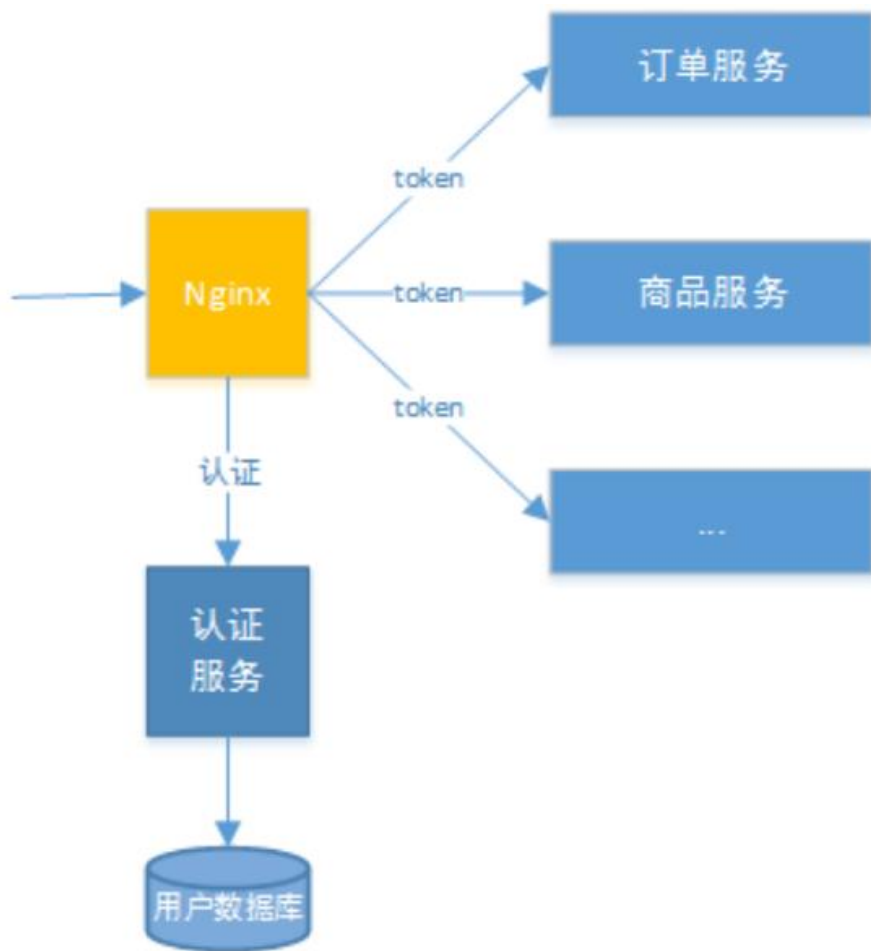
使用 JWT 可以实现无状态认证，什么是无状态认证？

传统的基于 session 的方式是有状态认证，用户登录成功将用户的身份信息存储在服务端，这样加大了服务端的存储压力，并且这种方式不适合在分布式系统中应用。

如下图，当用户访问应用服务，每个应用服务都会去服务器查看 session 信息，如果 session 中没有该用户则说明用户没有登录，此时就会重新认证，而解决这个问题的方法是 Session 复制、Session 黏贴。



如果是基于令牌技术在分布式系统中实现认证则服务端不用存储 session，可以将用户身份信息存储在令牌中，用户认证通过后认证服务颁发令牌给用户，用户将令牌存储在客户端，去访问应用服务时携带令牌去访问，服务端从 jwt 解析出用户信息。这个过程就是无状态认证。



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。

缺点：

- 1、JWT 令牌较长，占存储空间比较大。

下边是一个 JWT 令牌的示例：

Plain Text

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOi0lsicmVzMSJdLCJ1c2VyX25hbWUiOiJ6aGFuZ3NhbiIsInNjb3BlIjpbImFsbCJdLCJleHAiOjE2NjQyNTQ2NzIsImF1dG8iOiJhbnR1eSIsImVudF9pZCI6ImMxIn0.wkDBL7roLrvdBG2oGnX
```

```
eoXq-zZRgE9IwV2nxd-ez_oA
```

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

1. 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

```
JSON
{
  "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。

2. Payload

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

此部分不建议存放敏感信息，因为此部分可以解码还原原始内容。

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

一个例子：

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

3. 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 中声明的签名算法进行签名。

一个例子：

```
JSON
```



```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret)
```

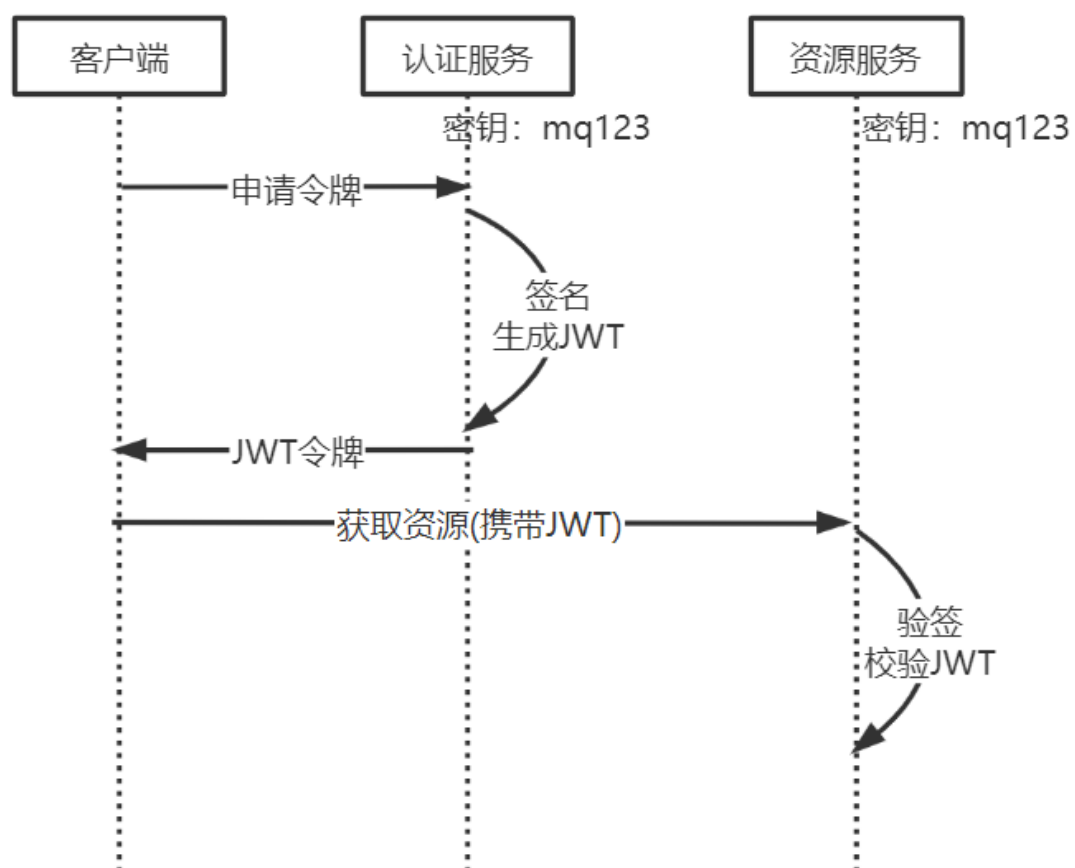
base64UrlEncode(header): jwt 令牌的第一部分。

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

secret: 签名所使用的密钥。

为什么 JWT 可以防止篡改?

第三部分使用签名算法对第一部分和第二部分的内容进行签名，常用的签名算法是 HS256，常见的还有 md5, sha 等，签名算法需要使用密钥进行签名，密钥不对外公开，并且签名是不可逆的，如果第三方更改了内容那么服务器验证签名就会失败，要想保证验证签名正确必须保证内容、密钥与签名前一致。



从上图可以看出认证服务和资源服务使用相同的密钥，这叫对称加密，对称加密效率高，如果一旦密钥泄露可以伪造 jwt 令牌。

JWT 还可以使用非对称加密，认证服务自己保留私钥，将公钥下发给受信任的客户端、资源服务，公钥和私钥是配对的，成对的公钥和私钥才可以正常加密和解密，非对称

加密效率低但相比对称加密非对称加密更安全一些。

2.4.3 测试生成 JWT 令牌

在认证服务中配置 jwt 令牌服务，即可实现生成 jwt 格式的令牌，

```
Java
package com.xuecheng.auth.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.oauth2.provider.token.AuthorizationSe
rverTokenServices;
import
org.springframework.security.oauth2.provider.token.DefaultTokenSer
vices;
import
org.springframework.security.oauth2.provider.token.TokenEnhancerCh
ain;
import
org.springframework.security.oauth2.provider.token.TokenStore;
import
org.springframework.security.oauth2.provider.token.store.InMemoryT
okenStore;
import
org.springframework.security.oauth2.provider.token.store.JwtAccess
TokenConverter;
import
org.springframework.security.oauth2.provider.token.store.JwtTokenS
tore;

import java.util.Arrays;

/**
 * @author Administrator
 * @version 1.0
 */
@Configuration
public class TokenConfig {

    private String SIGNING_KEY = "mq123";
```

```

    @Autowired
    TokenStore tokenStore;

//    @Bean
//    public TokenStore tokenStore() {
//        //使用内存存储令牌（普通令牌）
//        return new InMemoryTokenStore();
//    }

    @Autowired
    private JwtAccessTokenConverter accessTokenConverter;

    @Bean
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }

    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new
JwtAccessTokenConverter();
        converter.setSigningKey(SIGNING_KEY);
        return converter;
    }

//令牌管理服务
    @Bean(name="authorizationServerTokenServicesCustom")
    public AuthorizationServerTokenServices tokenService() {
        DefaultTokenServices service=new DefaultTokenServices();
        service.setSupportRefreshToken(true);//支持刷新令牌
        service.setTokenStore(tokenStore);//令牌存储策略

        TokenEnhancerChain tokenEnhancerChain = new
TokenEnhancerChain();

tokenEnhancerChain.setTokenEnhancers(Arrays.asList(accessTokenConv
erter));
        service.setTokenEnhancer(tokenEnhancerChain);

        service.setAccessTokenValiditySeconds(7200); // 令牌默认有
效期 2 小时
        service.setRefreshTokenValiditySeconds(259200); // 刷新令牌

```

默认有效期 3 天

```
        return service;
    }
}
```

重启认证服务。

使用 httpclient 通过密码模式申请令牌

Java

密码模式

POST

```
{{auth_host}}/oauth/token?client_id=XcWebApp&client_secret=XcWebApp
p&grant_type=password&username=zhangsan&password=123
```

生成 jwt 的示例如下:

JSON

```
{
  "access_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiIscmVzMSJdLCJ1c2VyX25hbWUiOiIj6aGFuZ3Nhb2IiInNjb3BlIjpbImFsbCJdLCJleHAiOjE2NjQzMzE2OTUsImF1dGhvcml0aWVzIjpbInAxIl0sImp0aSI6ImU5ZDNkMGZkLTl0Y2ItNDRjOC04YzEwLTl0NmIzNGY4ZGZjYyIsImNsaWVudF9pZCI6ImMxIn0.-9SKI-qUqKhKcs8Gb80Rascx-JxqsNZxxXoPo82d8SM",
  "token_type": "bearer",
  "refresh_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiIscmVzMSJdLCJ1c2VyX25hbWUiOiIj6aGFuZ3Nhb2IiInNjb3BlIjpbImFsbCJdLCJhdGkiOiIjLOWQzZDBmZC0yNGNiLTQ0YzgtOGMxMC0yNTZiZmZrM0GRmY2MiLCJleHAiOjE2NjQ1ODM2OTUsImF1dGhvcml0aWVzIjpbInAxIl0sImp0aSI6ImRjNTRjNTRkLTl0YTMtNDIzNS04MmY3LTFkOWZkMmFjM2VmNSIsImNsaWVudF9pZCI6ImMxIn0.Wsw1Jc-Kd_GFqEugzdfoSsMY6inC80QsraA21WjWtT8",
  "expires_in": 7199,
  "scope": "all",
  "jti": "e9d3d0fd-24cb-44c8-8c10-256b34f8dfcc"
}
```

- 1、access_token，生成的 jwt 令牌，用于访问资源使用。
- 2、token_type，bearer 是在 RFC6750 中定义的一种 token 类型，在携带 jwt 访问资源时需要在 head 中加入 bearer jwt 令牌内容
- 3、refresh_token，当 jwt 令牌快过期时使用刷新令牌可以再次生成 jwt 令牌。
- 4、expires_in：过期时间（秒）

6、jti: 令牌的唯一标识。

Plain Text

POST

响应示例如下:

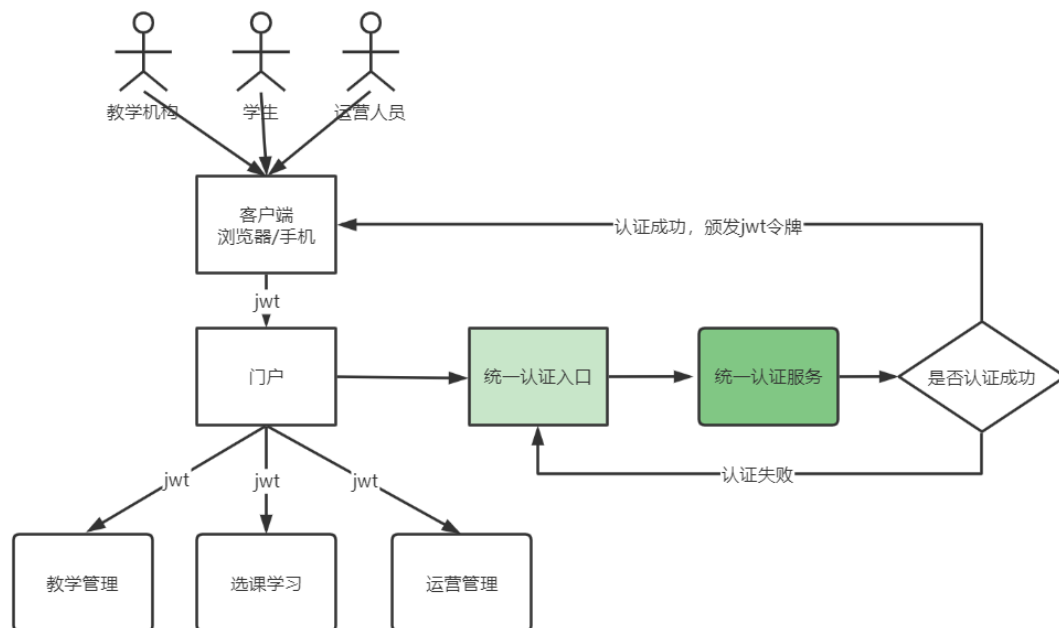
JSON

```
{
  "aud": [
    "res1"
  ],
  "user_name": "zhangsan",
  "scope": [
    "all"
  ],
  "active": true,
  "exp": 1664371780,
  "authorities": [
    "p1"
  ],
  "jti": "f0a3cdeb-399d-48f0-8804-eca638ad8857",
  "client_id": "c1"
}
```

2.4.4 携带令牌访问资源服务

拿到了 `jwt` 令牌下一步就要携带令牌去访问资源服务中的资源，本项目各个微服务就是资源服务，比如：内容管理服务，客户端申请到 `jwt` 令牌，携带 `jwt` 去内容管理服务查

询课程信息，此时内容管理服务要对 jwt 进行校验，只有 jwt 合法才可以继续访问。如下图：



1、在内容管理服务的 content-api 工程中添加依赖

XML

```
<!--认证相关-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
```

2、从资料中拷贝 资源服务配置类 中的 TokenConfig 和 ResouceServerConfig 到内容管理的 api 工程的 config 包下。

可以在 ResouceServerConfig 类中配置需要认证的 url。

Java

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeRequests()
            .antMatchers("/r/**", "/course/**").authenticated()/
    /所有/r/**的请求必须认证通过
```

```
.anyRequest().permitAll()  
;  
}
```

重启内容管理服务

使用 httpclient 测试:

1、访问根据课程 id 查询课程接口

Plain Text

查询课程信息

GET http://localhost:63040/content/course/2

返回:

JSON

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

从返回信息可知当前没有认证。

下边携带 JWT 令牌访问接口:

1、申请 jwt 令牌

采用密码模式申请令牌。

2、携带 jwt 令牌访问资源服务地址

Plain Text

携带 token 访问资源服务

GET http://localhost:63040/content/course/2

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOi0lsicmVzMSJdLCJ1c2VyX
25hbWUiOiJ6aGFuZ3NhbiIsInNjb3BlIjpbImFsbCJdLCJleHAiOjE2NjQzMzM0OTg
sImF1dGhvcmI0aWVzIjpbInAxIl0sImp0aSI6IjhhM2M2OTk1LWU1ZGEtNDQ1Yy05Z
DAyLTEwNDFlYzk3NTkwOSIsImNsaWVudF9pZCI6ImMxIn0.73eNDxTX5ifttGCjwc7
xrd-Sbp_mCfcIerI3lGetZto

在请求头中添加 Authorization, 内容为 Bearer 令牌, Bearer 用于通过 oauth2.0 协议访问资源。

如果携带 jwt 令牌且 jwt 正确则正常访问资源服务的内容。

如果不正确则报令牌无效的错误：

```
JSON
{
  "error": "invalid_token",
  "error_description": "Cannot convert access token to JSON"
}
```

2.4.5 测试获取用户身份

jwt 令牌中记录了用户身份信息，当客户端携带 jwt 访问资源服务，资源服务验签通过后将前两部分的内容还原即可取出用户的身份信息，并将用户身份信息放在了 SecurityContextHolder 上下文，SecurityContext 与当前线程进行绑定，方便获取用户身份。

还以查询课程接口为例，进入查询课程接口的代码中，添加获取用户身份的代码

```
Java
@ApiOperation("根据课程 id 查询课程基础信息")
@GetMapping("/course/{courseId}")
public CourseBaseInfoDto
getCourseBaseById(@PathVariable("courseId") Long courseId){
    //取出当前用户身份
    Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    System.out.println(principal);
    return courseBaseInfoService.getCourseBaseInfo(courseId);
}
```

测试时需要注意：

- 1、首先在资源服务配置中指定安全拦截机制 /course/开头的请求需要认证，即请求 /course/{courseId}接口需要携带 jwt 令牌且签证通过。
- 2、认证服务生成 jwt 令牌将用户身份信息写入令牌，目前还是将用户信息硬编码并暂放在内存中。

如下：

```
Java
@Bean
```



```

public UserDetailsService userDetailsService() {
    //这里配置用户信息,这里暂时使用这种方式将用户存储在内存中
    InMemoryUserDetailsManager manager = new
    InMemoryUserDetailsManager();

    manager.createUser(User.withUsername("zhangsan").password("123").a
    uthorities("p1").build());

    manager.createUser(User.withUsername("lisi").password("456").autho
    rities("p2").build());
    return manager;
}

```

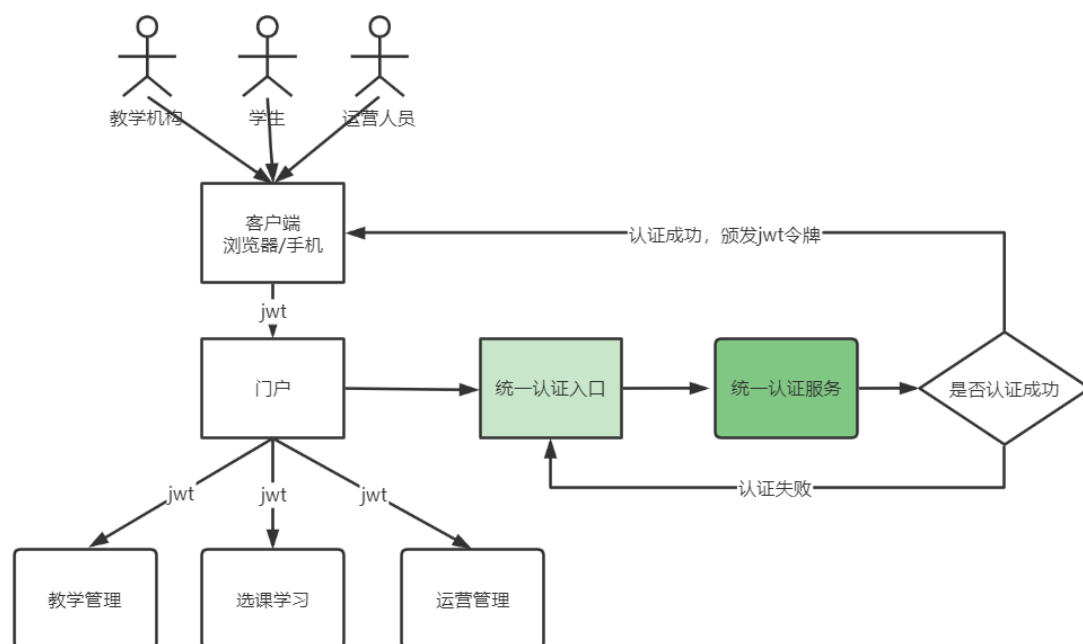
3、我们在使用密码模式生成 jwt 令牌时用的是 zhangsan 的信息，所以 jwt 令牌中存储了 zhangsan 的信息，那么在资源服务中应该取出 zhangsan 的信息才对。

清楚了以上内容，下边重启内容管理服务，跟踪取到的用户身份是否正确。

2.5 网关认证

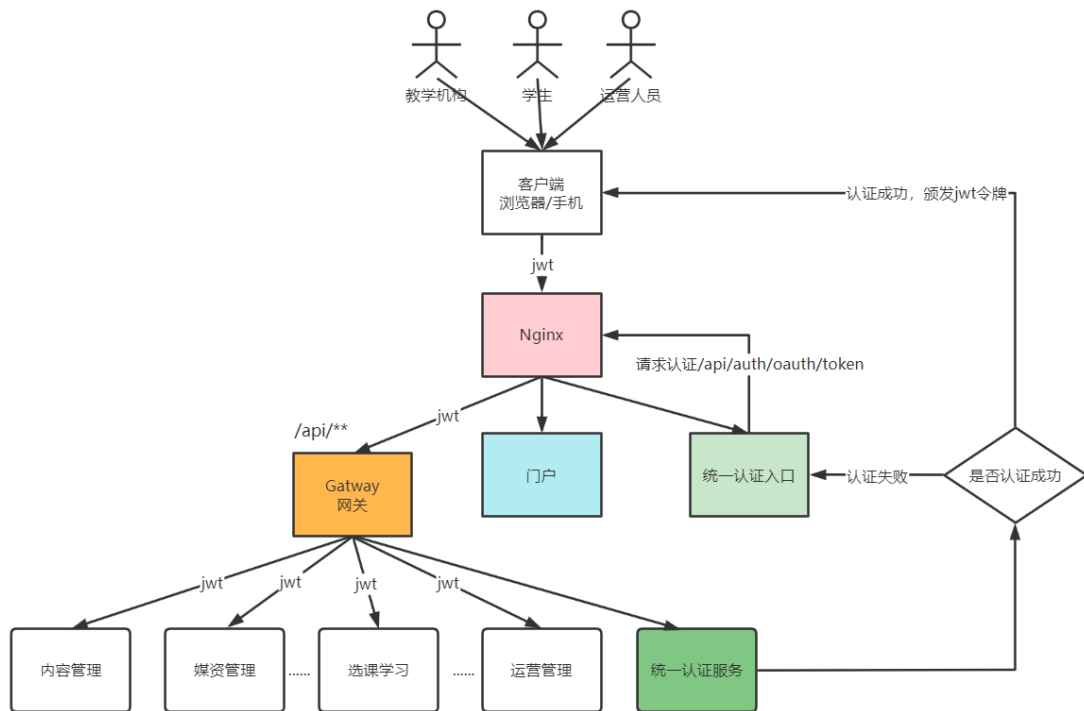
2.5.1 技术方案

到目前为止，测试通过了认证服务颁发 jwt 令牌，客户端携带 jwt 访问资源服务，资源服务对 jwt 的合法性进行验证。如下图：



仔细观察此图，遗漏了本项目架构中非常重要的组件：网关，加上网关并完善后如下

图所示：



所有访问微服务的请求都要经过网关，在网关进行用户身份的认证可以将很多非法的请求拦截到微服务以外，这叫做网关认证。

下边需要明确网关的职责：

1、网站白名单维护

针对不用认证的 URL 全部放行。

2、校验 jwt 的合法性。

除了白名单剩下的就是需要认证的请求，网关需要验证 jwt 的合法性，jwt 合法则说明用户身份合法，否则说明身份不合法则拒绝继续访问。

网关负责授权吗？

网关不负责授权，对请求的授权操作在各个微服务进行，因为微服务最清楚用户有哪些权限访问哪些接口。

2.5.2 实现网关认证

下边实现网关认证，实现以下职责：

1、网站白名单维护

针对不用认证的 URL 全部放行。

2、校验 jwt 的合法性。

除了白名单剩下的就是需要认证的请求，网关需要验证 jwt 的合法性，jwt 合法则说明用户身份合法，否则说明身份不合法则拒绝继续访问。

1、在网关工程添加依赖

XML

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
</dependency>
```

2、拷贝课程资料下网关认证配置类到网关工程的 config 包下。

3、配置白名单文件 security-whitelist.properties

内容如下（持续补充）

Plain Text

```
/auth/**=认证地址
/content/open/**=内容管理公开访问接口
/media/open/**=媒资管理公开访问接口
```

重启网关工程，进行测试

1、申请令牌

2、通过网关访问资源服务

这里访问内容管理服务

Plain Text

通过网关访问资源服务

GET http://localhost:63010/content/course/2

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOi0lsicmVzMSJdLCJ1c2VyX25hbWUiOiJ6aGFuZ3NhbiIsInNjb3BlIjpbImFsbCJdLCJleHAiOjE2NjQzNjIzMTAsImF1dGhvcml0aWVzIjpbInAxIl0sImp0aSI6Ijc2OTkwMGNiLWM1ZjItNGRiNC1hZWJmLWY1MzgxDQxZWMyZCIsImNsaWVudF9pZCI6ImMxIn0.10ITjUgYg2HCh5mDPK9EvJJqz-tIupKVfmP8yWJQIKs

当 token 正确时可以正常访问资源服务，token 验证失败返回 token 无效：

JSON

```
{
  "errorMessage": "认证令牌无效"
}
```

注意：网关鉴权功能调试通过后，由于目前还没有开发认证功能，前端请求网关的 URL 不在白名单中间时会“没有认证”的错误，暂时在白名单中添加 全部放行配置，待认证功能开发完成再屏蔽全部放行配置，

/**=全部暂时放开

/auth/**=认证地址

/content/open/**=内容管理公开访问接口

/media/open/**=媒资管理公开访问接口

由于是在网关处进行令牌校验，所以在微服务处不再校验令牌的合法性，修改内容管理服务的 ResourceServerConfig 类，屏蔽 authenticated()。

Java

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeRequests()
        // .antMatchers("/r/**", "/course/**").authenticated()//所有/r/**的请求必须认证通过
        .anyRequest().permitAll()
    ;
}
```

3 用户认证

3.1 需求分析

至此我们了解了使用 Spring Security 进行认证授权的过程，本节实现用户认证功能。

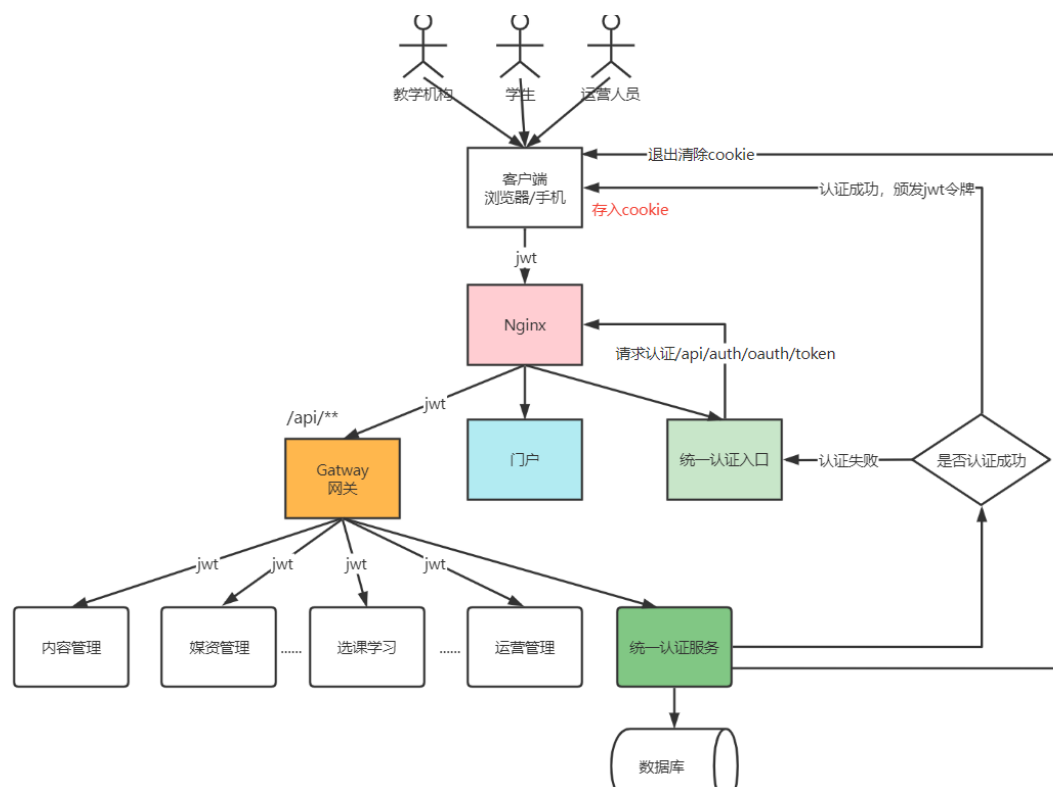
目前各大网站的认证方式非常丰富：账号密码认证、手机验证码认证、扫码登录等。

本项目也要支持多种认证试。

3.2 连接用户中心数据库

3.2.1 连接数据库认证

基于的认证流程在研究 Spring Security 过程中已经测试通过，到目前为止用户认证流程如下：



认证所需要的用户信息存储在用户中心数据库，现在需要将认证服务连接数据库查询用户信息。

在研究 Spring Security 的过程中是将用户信息硬编码，如下：

```

Java
@Bean
public UserDetailsServiceImpl userDetailsServiceImpl() {
    //这里配置用户信息,这里暂时使用这种方式将用户存储在内存中
    InMemoryUserDetailsManager manager = new
    InMemoryUserDetailsManager();

    manager.createUser(User.withUsername("zhangsan").password("123").authorities("p1").build());

    manager.createUser(User.withUsername("lisi").password("456").authorities("p2").build());

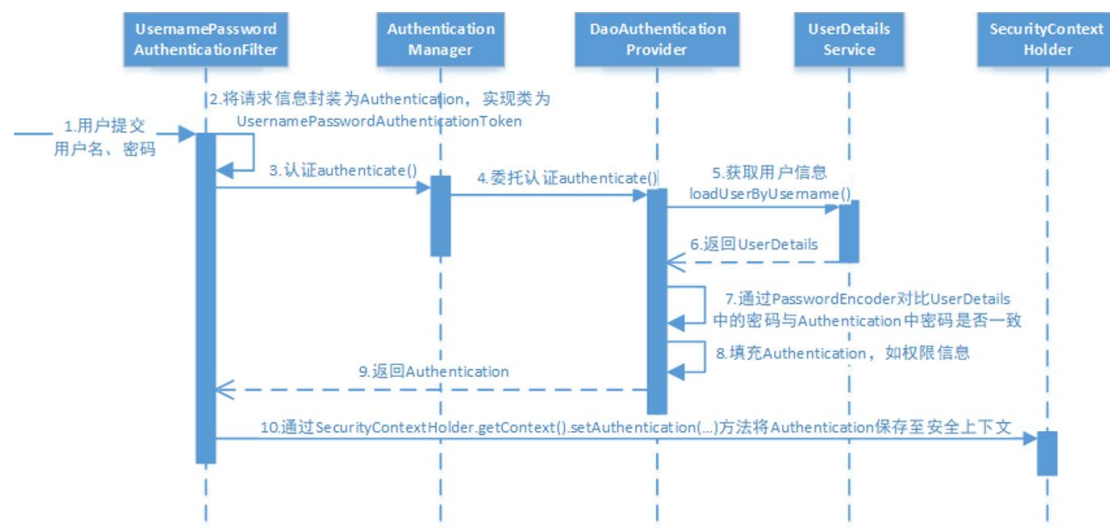
    return manager;
}

```

我们要认证服务中连接用户中心数据库查询用户信息。

如何使用 Spring Security 连接数据库认证吗？

前边学习 Spring Security 工作原理时有一张执行流程图，如下图：



用户提交账号和密码由 DaoAuthenticationProvider 调用 UserDetailsServiceImpl 的 loadUserByUsername()方法获取 UserDetails 用户信息。

查询 DaoAuthenticationProvider 的源代码如下：

```

48
49 protected final UserDetails retrieveUser(String username, UsernamePasswordAuthenticationToken authentic
50     this.prepareTimingAttackProtection();
51
52     try {
53         UserDetails loadedUser = this.getUserDetailsService().loadUserByUsername(username);
54         if (loadedUser == null) {
55             throw new InternalAuthenticationServiceException("UserDetailsService returned null, which is

```

UserDetailsService 是一个接口，如下：

Java

```
package org.springframework.security.core.userdetails;

public interface UserDetailsService {
    UserDetails loadUserByUsername(String var1) throws
    UsernameNotFoundException;
}
```

UserDetails 是用户信息接口

Java

```
public interface UserDetails extends Serializable {
    Collection<? extends GrantedAuthority> getAuthorities();

    String getPassword();

    String getUsername();

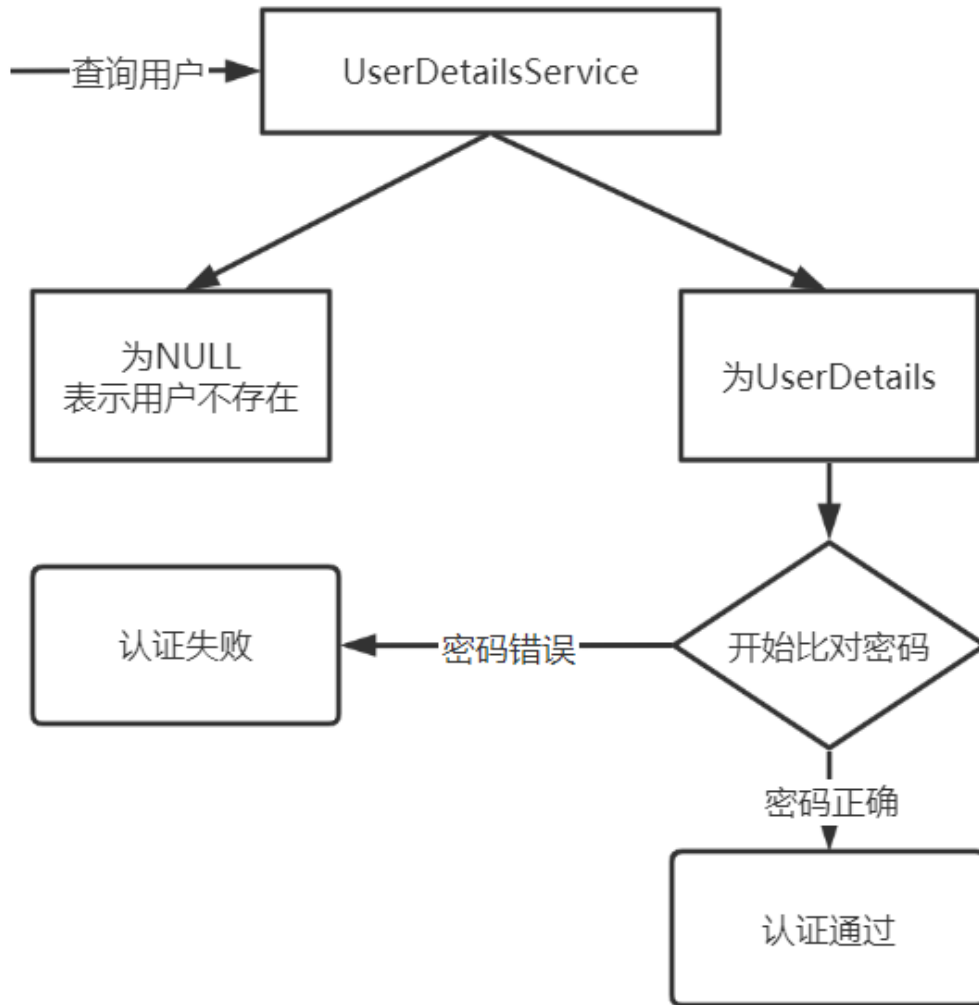
    boolean isAccountNonExpired();

    boolean isAccountNonLocked();

    boolean isCredentialsNonExpired();

    boolean isEnabled();
}
```

我们只要实现 UserDetailsService 接口查询数据库得到用户信息返回 UserDetails 类型的用户信息即可,框架调用 loadUserByUsername()方法拿到用户信息之后是如何执行的, 见下图:



首先屏蔽原来定义的 UserDetailsService。

```
Java
//配置用户信息服务
// @Bean
// public UserDetailsService userDetailsService() {
//     //这里配置用户信息,这里暂时使用这种方式将用户存储在内存中
//     InMemoryUserDetailsManager manager = new
InMemoryUserDetailsManager();
//
manager.createUser(User.withUsername("zhangsan").password("123").a
uthorities("p1").build());
//
manager.createUser(User.withUsername("lisi").password("456").autho
rities("p2").build());
//     return manager;
// }
```

下边自定义 UserDetailsService

Java

```
package com.xuecheng.ucenter.service.impl;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.ucenter.mapper.XcUserMapper;
import com.xuecheng.ucenter.model.po.XcUser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

/**
 * @author Mr.M
 * @version 1.0
 * @description TODO
 * @date 2022/9/28 18:09
 */
@Service
public class UserServiceImpl implements UserDetailsService {

    @Autowired
    XcUserMapper xcUserMapper;

    /**
     * @description 根据账号查询用户信息
     * @param s 账号
     * @return
     org.springframework.security.core.userdetails.UserDetails
     * @author Mr.M
     * @date 2022/9/28 18:30
     */
    @Override
    public UserDetails loadUserByUsername(String s) throws
    UsernameNotFoundException {

        XcUser user = xcUserMapper.selectOne(new
    LambdaQueryWrapper<XcUser>().eq(XcUser::getUsername, s));
        if(user==null){
```

```

        //返回空表示用户不存在
        return null;
    }
    //取出数据库存储的正确密码
    String password =user.getPassword();
    //用户权限,如果不加报 Cannot pass a null GrantedAuthority
collection
    String[] authorities= {"test"};
    //创建 UserDetails 对象,权限信息待实现授权功能时再向 UserDetails
中加入
    UserDetails userDetails =
User.withUsername(user.getUsername()).password(password).authoriti
es(authorities).build();

    return userDetails;
}

}

```

数据库中的密码加过密的, 用户输入的密码是明文, 我们需要修改密码格式器 PasswordEncoder, 原来使用的是 NoOpPasswordEncoder, 它是通过明文方式比较密码, 现在我们修改为 BCryptPasswordEncoder, 它是将用户输入的密码编码为 BCrypt 格式与数据库中的密码进行比对。

如下:

```

Java
@Bean
public PasswordEncoder passwordEncoder() {
//    //密码为明文方式
//    return NoOpPasswordEncoder.getInstance();
    return new BCryptPasswordEncoder();
}

```

我们通过测试代码测试 BCryptPasswordEncoder, 如下:

```

Java
public static void main(String[] args) {
    String password = "111111";
    PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    for(int i=0;i<10;i++) {
        //每个计算出的 Hash 值都不一样
        String hashPass = passwordEncoder.encode(password);
    }
}

```

```

        System.out.println(hashPass);
        //虽然每次计算的密码 Hash 值不一样但是校验是通过的
        boolean f = passwordEncoder.matches(password, hashPass);
        System.out.println(f);
    }
}

```

修改数据库中的密码为 Bcrypt 格式，并且记录明文密码，稍后申请令牌时需要。

由于修改密码编码方式还需要将客户端的密钥更改为 Bcrypt 格式。

```

Java
@Override
public void configure(ClientDetailsServiceConfigurer clients)
    throws Exception {
    clients.inMemory()// 使用 in-memory 存储
        .withClient("XcWebApp")// client_id
//        .secret("secret")//客户端密钥
        .secret(new
BCryptPasswordEncoder().encode("XcWebApp"))//客户端密钥
        .resourceIds("xuecheng-plus")//资源列表
        .authorizedGrantTypes("authorization_code",
"password","client_credentials","implicit","refresh_token")// 该
client 允许的授权类型
authorization_code,password,refresh_token,implicit,client_credenti
als
        .scopes("all")// 允许的授权范围
        .autoApprove(false)//false 跳转到授权页面
//客户端接收授权码的重定向地址
        .redirectUri("http://www.51xuecheng.cn")
    ;
}

```

现在重启认证服务。

下边使用 httpclient 进行测试：

```

Java
### 密码模式
POST
{{auth_host}}/oauth/token?client_id=XcWebApp&client_secret=XcWebAp
p&grant_type=password&username=stu1&password=111111

```

输入正确的账号和密码，申请令牌成功。

输入错误的密码，报错：

```
Java
{
  "error": "invalid_grant",
  "error_description": "用户名或密码错误"
}
```

输入错误的账号，报错：

```
Java
{
  "error": "unauthorized",
  "error_description": "UserDetailsService returned null, which is an interface contract violation"
}
```

3.2.3 扩展用户身份信息

用户表中存储了用户的账号、手机号、email，昵称、qq 等信息，UserDetails 接口只返回了 username、密码等信息，如下：

```
Java
public interface UserDetails extends Serializable {
    Collection<? extends GrantedAuthority> getAuthorities();

    String getPassword();

    String getUsername();

    boolean isAccountNonExpired();

    boolean isAccountNonLocked();

    boolean isCredentialsNonExpired();

    boolean isEnabled();
}
```

我们需要扩展用户身份的信息，在 jwt 令牌中存储用户的昵称、头像、qq 等信息。

如何扩展 Spring Security 的用户身份信息呢？

在认证阶段 DaoAuthenticationProvider 会调用 UserDetailsService 查询用户的信息，这里是可获取到齐全的用户信息的。由于 JWT 令牌中用户身份信息来源于 UserDetails，UserDetails 中仅定义了 username 为用户的身份信息，这里有两个思路：第一是可以扩展 UserDetails，使之包括更多的自定义属性，第二也可以扩展 username 的内容，比如存入 json 数据内容作为 username 的内容。相比较而言，方案二比较简单还不用破坏 UserDetails 的结构，我们采用方案二。

修改 UserServiceImpl 如下：

```
Java
package com.xuecheng.ucenter.service.impl;

import com.alibaba.fastjson.JSON;
import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.ucenter.mapper.XcUserMapper;
import com.xuecheng.ucenter.model.po.XcUser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

/**
 * @author Mr.M
 * @version 1.0
 * @description TODO
 * @date 2022/9/28 18:09
 */
@Service
public class UserServiceImpl implements UserDetailsService {

    @Autowired
    XcUserMapper xcUserMapper;

    /**
     * @description 根据账号查询用户信息
     * @param s 账号
     * @return
     */
    org.springframework.security.core.userdetails.UserDetails
```

```

    * @author Mr.M
    * @date 2022/9/28 18:30
    */
    @Override
    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {

        XcUser user = xcUserMapper.selectOne(new
LambdaQueryWrapper<XcUser>().eq(XcUser::getUsername, s));
        if(user==null){
            //返回空表示用户不存在
            return null;
        }

        //取出数据库存储的正确密码
        String password =user.getPassword();
        //用户权限,如果不加报 Cannot pass a null GrantedAuthority
collection
        String[] authorities = {"p1"};
        //为了安全在令牌中不放密码
        user.setPassword(null);
        //将 user 对象转 json
        String userString = JSON.toJSONString(user);
        //创建 UserDetails 对象
        UserDetails userDetails =
User.withUsername(userString).password(password).authorities(autho
rities).build();

        return userDetails;
    }
}

```

重启认证服务，重新生成令牌，生成成功。

我们可以使用 check_token 查询 jwt 的内容

```

JSON
###校验 jwt 令牌
POST {{auth_host}}/oauth/check_token?token=

```

响应示例如下，

JSON

```
{
  "aud": [
    "res1"
  ],
  "user_name": "{\\"birthday\\":\\"2022-09-28T19:28:46\\",\\"createTime\\":\\"2022-09-28T08:32:03\\",\\"id\\":\\"50\\",\\"name\\":\\"学生 1\\",\\"nickname\\":\\"大水牛\\",\\"password\\":\\"$2a$10$0pt7WlftbnPDTcWtp/.2Mu5CTXvohnNQhR628qq4RoKSc0dGAdEgm\\",\\"sex\\":\\"1\\",\\"status\\":\\"1\\",\\"username\\":\\"stu1\\",\\"userpic\\":\\"http://file.51xuecheng.cn/dddf\\",\\"utype\\":\\"101001\\"}",
  "scope": [
    "all"
  ],
  "active": true,
  "exp": 1664372184,
  "authorities": [
    "p1"
  ],
  "jti": "73da9f7b-bd8c-45ac-9add-46b711d11fb8",
  "client_id": "c1"
}
```

user_name 存储了用户信息的 json 格式，在资源服务中就可以取出该 json 格式的内容转为用户对象去使用。

3.2.3 资源服务获取用户身份

下边编写一个工具类在各个微服务中去使用，获取当前登录用户的对象。

在 content-api 中定义此类：

Java

```
package com.xuecheng.content.util;

import com.alibaba.fastjson.JSON;
import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.core.context.SecurityContextHolder;
```

```

import java.io.Serializable;
import java.time.LocalDateTime;

/**
 * @author Mr.M
 * @version 1.0
 * @description 获取当前用户身份工具类
 * @date 2022/10/18 18:02
 */
@Slf4j
public class SecurityUtil {

    public static XcUser getUser() {
        try {
            Object principalObj =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();

            if (principalObj instanceof String) {
                //取出用户身份信息
                String principal = principalObj.toString();
                //将 json 转成对象
                XcUser user = JSON.parseObject(principal,
XcUser.class);
                return user;
            }
        } catch (Exception e) {
            Log.error("获取当前登录用户身份出错:{}",
e.getMessage());
            e.printStackTrace();
        }

        return null;
    }

    @Data
    public static class XcUser implements Serializable {

        private static final long serialVersionUID = 1L;

        private String id;

        private String username;

```



```
private String password;

private String salt;

private String name;
private String nickname;
private String wxUnionid;
private String companyId;
/**
 * 头像
 */
private String userpic;

private String utype;

private LocalDateTime birthday;

private String sex;

private String email;

private String cellphone;

private String qq;

/**
 * 用户状态
 */
private String status;

private LocalDateTime createTime;

private LocalDateTime updateTime;

}

}
```

下边在内容管理服务中测试此工具类，以查询课程信息接口为例：

```
Java
@ApiOperation("根据课程 id 查询课程基础信息")
```

```

@GetMapping("/course/{courseId}")
public CourseBaseInfoDto
getCourseBaseById(@PathVariable("courseId") Long courseId){
    //取出当前用户身份
    // Object principal =
    SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    SecurityUtil.XcUser user = SecurityUtil.getUser();
    System.out.println(user);

    return courseBaseInfoService.getCourseBaseInfo(courseId);
}

```

重启内容管理服务：

- 1、启动认证服务、网关、内容管理服务
- 2、生成新的令牌
- 3、携带令牌访问内容管理服务的查询课程接口

```

SecurityUtil.XcUser user = SecurityUtil.getUser(); user: "SecurityUtil.XcUser
System.out.println(us user = {SecurityUtil$XcUser@15018} "SecurityUtil.XcUser(id=50, username=stu1, p
return courseBaseInfo
    > f id = "50"
    > f username = "stu1"
    > f password = null
    > f salt = null
    > f name = "学生1"
    > f nickname = "大水牛"
    > f wxUnionid = null
    > f companyId = null
    > f userpic = "http://file.xuecheng-plus.com/dddf"
    > f utype = "101001"
    > f birthday = {LocalDateTime@15034} "2022-09-28T19:28:46"
    > f sex = "1"
    > f email = null

```

3.3 支持认证方式多样

3.3.1 统一认证入口

目前各大网站的认证方式非常丰富：账号密码认证、手机验证码认证、扫码登录等。
 基于当前研究的 Spring Security 认证流程如何支持多样化的认证方式呢？

- 1、支持账号和密码认证

采用 OAuth2 协议的密码模式即可实现。

2、支持手机号加验证码认证

用户认证提交的是手机号和验证码，并不是账号和密码。

3、微信扫码认证

基于 OAuth2 协议与微信交互，学成在线网站向微信服务器申请到一个令牌，然后携带令牌去微信查询用户信息，查询成功则用户在学成在线项目认证通过。

目前我们测试通过 OAuth2 的密码模式，用户认证会提交账号和密码，由 DaoAuthenticationProvider 调用 UserDetailsService 的 loadUserByUsername() 方法获取 UserDetails 用户信息。

在前边我们自定义了 UserDetailsService 接口实现类，通过 loadUserByUsername() 方法根据账号查询用户信息。

而不同的认证方式提交的数据不一样，比如：手机加验证码方式会提交手机号和验证码，账号密码方式会提交账号、密码、验证码。

我们可以在 loadUserByUsername() 方法上作文章，将用户原来提交的账号数据改为提交 json 数据，json 数据可以扩展不同认证方式所提交的各种参数。

首先创建一个 DTO 类表示认证的参数：

```
Java
package com.xuecheng.ucenter.model.dto;

import lombok.Data;

import java.util.HashMap;
import java.util.Map;

/**
 * @author Mr.M
 * @version 1.0
 * @description 认证用户请求参数
 * @date 2022/9/29 10:56
 */
@Data
public class AuthParamsDto {

    private String username; //用户名
    private String password; //域 用于扩展
    private String cellphone; //手机号
```

```

        private String checkcode;//验证码
        private String checkcodekey;//验证码 key
        private String authType; // 认证的类型    password:用户名密码模式
        类型    sms:短信模式类型
        private Map<String, Object> payload = new HashMap<>();//附加数
        据，作为扩展，不同认证类型可拥有不同的附加数据。如认证类型为短信时包含
        smsKey : sms:3d21042d054548b08477142bbca95cfa; 所有情况下都包含
        clientId

    }

```

此时 loadUserByUsername()方法可以修改如下：

```

Java
package com.xuecheng.ucenter.service.impl;

import com.alibaba.fastjson.JSON;
import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.ucenter.mapper.XcUserMapper;
import com.xuecheng.ucenter.model.dto.AuthParamsDto;
import com.xuecheng.ucenter.model.po.XcUser;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

/**
 * @author Mr.M
 * @version 1.0
 * @description 自定义 UserDetailsService 用来对接 Spring Security
 * @date 2022/9/28 18:09
 */
@Slf4j
@Service

```

```

public class UserServiceImpl implements UserDetailsService {

    @Autowired
    XcUserMapper xcUserMapper;

    /**
     * @description 查询用户信息组成用户身份信息
     * @param s AuthParamsDto 类型的 json 数据
     * @return
     org.springframework.security.core.userdetails.UserDetails
     * @author Mr.M
     * @date 2022/9/28 18:30
     */
    @Override
    public UserDetails loadUserByUsername(String s) throws
    UsernameNotFoundException {

        AuthParamsDto authParamsDto = null;
        try {
            //将认证参数转为 AuthParamsDto 类型
            authParamsDto = JSON.parseObject(s,
AuthParamsDto.class);
        } catch (Exception e) {
            Log.info("认证请求不符合项目要求:{},s);
            throw new RuntimeException("认证请求数据格式不对");
        }
        //账号
        String username = authParamsDto.getUsername();
        XcUser user = xcUserMapper.selectOne(new
LambdaQueryWrapper<XcUser>().eq(XcUser::getUsername, username));
        if(user==null){
            //返回空表示用户不存在
            return null;
        }
        //取出数据库存储的正确密码
        String password =user.getPassword();
        //用户权限,如果不加报 Cannot pass a null GrantedAuthority
collection
        String[] authorities = {"p1"};
        //将 user 对象转 json
        String userString = JSON.toJSONString(user);
        //创建 UserDetails 对象
        UserDetails userDetails =

```

```

        User.withUsername(userString).password(password).authorities(authorities).build();

        return userDetails;
    }
}

```

原来的 DaoAuthenticationProvider 会进行密码校验，现在重新定义 DaoAuthenticationProviderCustom 类，重写类的 additionalAuthenticationChecks 方法。

```

Java
package com.xuecheng.auth.config;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;

/**
 * @description 自定义 DaoAuthenticationProvider
 * @author Mr.M
 * @date 2022/9/29 10:31
 * @version 1.0
 */
@Slf4j
@Component
public class DaoAuthenticationProviderCustom extends DaoAuthenticationProvider {

```

```

@Autowired
public void setUserDetailsService(UserDetailsService
userDetailsService) {
    super.setUserDetailsService(userDetailsService);
}

//屏蔽密码对比
protected void additionalAuthenticationChecks(UserDetails
userDetails, UsernamePasswordAuthenticationToken authentication)
throws AuthenticationException {

}

}

```

修改 WebSecurityConfig 类指定 daoAuthenticationProviderCustom

```

Java
@Autowired
DaoAuthenticationProviderCustom daoAuthenticationProviderCustom;

@Override
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
    auth.authenticationProvider(daoAuthenticationProviderCustom);
}

```

此时可以重启认证服务，测试申请令牌接口，传入的账号信息改为 json 数据，如下：

```

Plain Text
#####扩展认证请求参数后#####
###密码模式
POST
{{auth_host}}/auth/oauth/token?client_id=XcWebApp&client_secret=Xc
WebApp&grant_type=password&username={"username":"stu1","authType":
"password","password":"111111"}

```

经过测试发现 loadUserByUsername()方法可以正常接收到认证请求中的 json 数据。

有了这些认证参数我们可以定义一个认证 Service 接口去进行各种方式的认证。

定义用户信息，为了扩展性让它继承 XcUser

```
Java
@Data
public class XcUserExt extends XcUser {
}
```

定义认证 Service 接口

```
Java
package com.xuecheng.ucenter.service;

import com.xuecheng.ucenter.model.dto.AuthParamsDto;
import com.xuecheng.ucenter.model.po.XcUser;

/**
 * @description 认证 service
 * @author Mr.M
 * @date 2022/9/29 12:10
 * @version 1.0
 */
public interface AuthService {

    /**
     * @description 认证方法
     * @param authParamsDto 认证参数
     * @return com.xuecheng.ucenter.model.po.XcUser 用户信息
     * @author Mr.M
     * @date 2022/9/29 12:11
     */
    XcUserExt execute(AuthParamsDto authParamsDto);

}
```

loadUserByUsername()修改如下:

```
Java
@Slf4j
@Service
public class UserServiceImpl implements UserDetailsService {

    @Autowired
    XcUserMapper xcUserMapper;
```



```

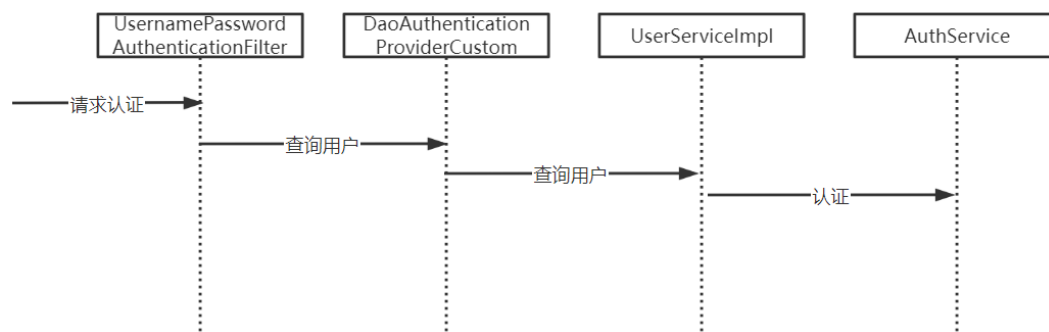
@Autowired
ApplicationContext applicationContext;

/**
 * @description 查询用户信息组成用户身份信息
 * @param s AuthParamsDto 类型的 json 数据
 * @return
org.springframework.security.core.userdetails.UserDetails
 * @author Mr.M
 * @date 2022/9/28 18:30
 */
@Override
public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {

    AuthParamsDto authParamsDto = null;
    try {
        //将认证参数转为 AuthParamsDto 类型
        authParamsDto = JSON.parseObject(s,
AuthParamsDto.class);
    } catch (Exception e) {
        Log.info("认证请求不符合项目要求:{}",s);
        throw new RuntimeException("认证请求数据格式不对");
    }
    //开始认证
    authService.execute(authParamsDto);
    .....

```

到此我们基于 Spring Security 认证流程修改为如下：



3.3.2 实现账号密码认证

上节定义了 AuthService 认证接口，下边实现该接口实现账号密码认证

Java

```
package com.xuecheng.ucenter.service.impl;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.ucenter.mapper.XcUserMapper;
import com.xuecheng.ucenter.model.dto.AuthParamsDto;
import com.xuecheng.ucenter.model.po.XcUser;
import com.xuecheng.ucenter.service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

/**
 * @description 账号密码认证
 * @author Mr.M
 * @date 2022/9/29 12:12
 * @version 1.0
 */
@Service("password_authservice")
public class PasswordAuthServiceImpl implements AuthService {

    @Autowired
    XcUserMapper xcUserMapper;

    @Autowired
    PasswordEncoder passwordEncoder;

    @Override
    public XcUserExt execute(AuthParamsDto authParamsDto) {

        //账号
        String username = authParamsDto.getUsername();
        XcUser user = xcUserMapper.selectOne(new
        LambdaQueryWrapper<XcUser>().eq(XcUser::getUsername, username));
        if(user==null){
            //返回空表示用户不存在
            throw new RuntimeException("账号不存在");
        }
    }
}
```

```

    }
    XcUserExt xcUserExt = new XcUserExt();
    BeanUtils.copyProperties(user, xcUserExt);
    //校验密码
    //取出数据库存储的正确密码
    String passwordDb = user.getPassword();
    String passwordForm = authParamsDto.getPassword();
    boolean matches = passwordEncoder.matches(passwordForm,
passwordDb);
    if(!matches){
        throw new RuntimeException("账号或密码错误");
    }
    return xcUserExt;
}
}

```

修改 UserServiceImpl 类，根据认证方式使用不同的认证 bean

```

Java
package com.xuecheng.ucenter.service.impl;

import com.alibaba.fastjson.JSON;
import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.ucenter.mapper.XcUserMapper;
import com.xuecheng.ucenter.model.dto.AuthParamsDto;
import com.xuecheng.ucenter.model.po.XcUser;
import com.xuecheng.ucenter.service.AuthService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

/**
 * @author Mr.M
 * @version 1.0
 * @description 自定义 UserDetailsService 用来对接 Spring Security

```

```

* @date 2022/9/28 18:09
*/
@Slf4j
@Service
public class UserServiceImpl implements UserDetailsService {

    @Autowired
    XcUserMapper xcUserMapper;

    @Autowired
    ApplicationContext applicationContext;

    // @Autowired
    // AuthService authService;

    /**
     * @description 查询用户信息组成用户身份信息
     * @param s AuthParamsDto 类型的 json 数据
     * @return
     org.springframework.security.core.userdetails.UserDetails
     * @author Mr.M
     * @date 2022/9/28 18:30
     */
    @Override
    public UserDetails loadUserByUsername(String s) throws
    UsernameNotFoundException {

        AuthParamsDto authParamsDto = null;
        try {
            //将认证参数转为 AuthParamsDto 类型
            authParamsDto = JSON.parseObject(s,
AuthParamsDto.class);
        } catch (Exception e) {
            Log.info("认证请求不符合项目要求:{}", s);
            throw new RuntimeException("认证请求数据格式不对");
        }

        //认证方法
        String authType = authParamsDto.getAuthType();
        AuthService authService =
applicationContext.getBean(authType +
"_authservice",AuthService.class);
        XcUserExt user = authService.execute(authParamsDto);
    }
}

```

```

        return getUserPrincipal(user);
    }

    /**
     * @description 查询用户信息
     * @param user 用户 id，主键
     * @return com.xuecheng.ucenter.model.po.XcUser 用户信息
     * @author Mr.M
     * @date 2022/9/29 12:19
     */
    public UserDetails getUserPrincipal(XcUserExt user){
        //用户权限,如果不加报 Cannot pass a null GrantedAuthority
        collection
        String[] authorities = {"p1"};
        String password = user.getPassword();
        //为了安全在令牌中不放密码
        user.setPassword(null);
        //将 user 对象转 json
        String userString = JSON.toJSONString(user);
        //创建 UserDetails 对象
        UserDetails userDetails =
        User.withUsername(userString).password(password).authorities(auth
        orities).build();
        return userDetails;
    }
}

```

重启认证服务，测试申请令牌接口。

- 1、测试账号和密码都正确的情况是否可以申请令牌成功。
- 2、测试密码错误的情况。
- 3、测试账号不存在情况。

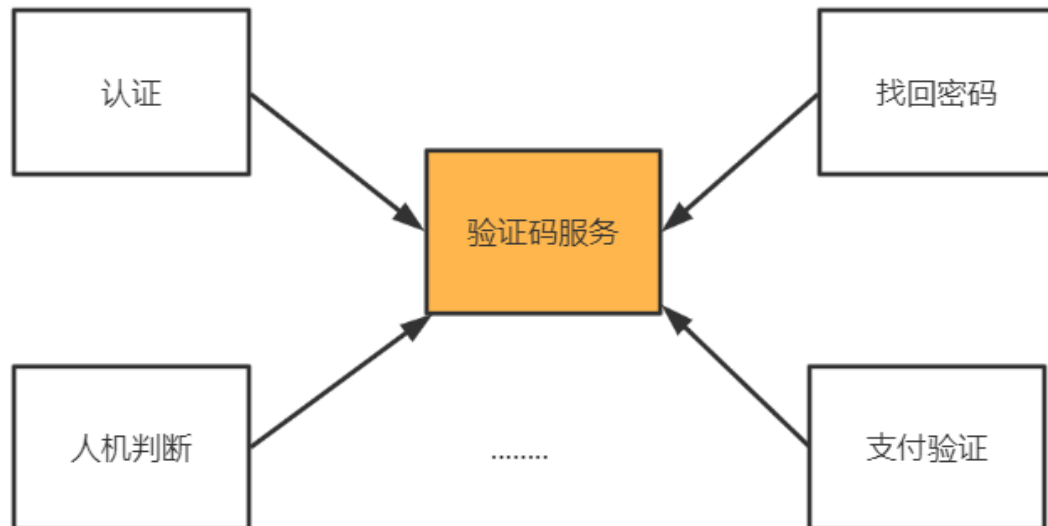
3.4 验证码服务

3.4.1 部署验证码服务工程

在认证时一般都需要输入验证码，验证码有什么用？

验证码可以防止恶性攻击，比如：XSS 跨站脚本攻击、CSRF 跨站请求伪造攻击，一些比较复杂的图形验证码可以有效的防止恶性攻击。

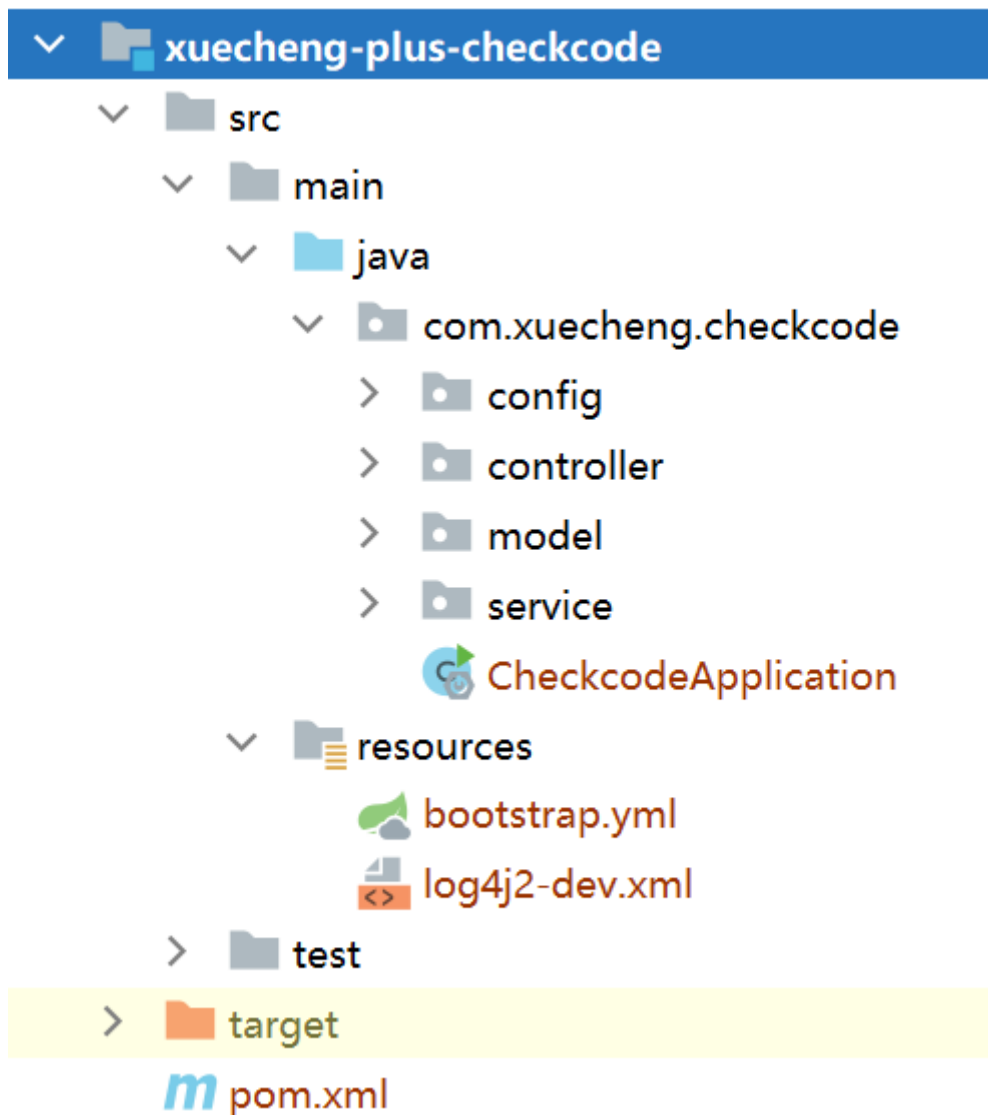
为了保护系统的安全在一些比较重要的操作都需要验证码。



验证码的类型也有很多：图片、语音、手机短信验证码等。

本项目创建单独的验证码服务为各业务提供验证码的生成、校验等服务。

拷贝课程资料目录 xuecheng-plus-checkcode 验证码服务工程到自己的工程目录。



定义 nacos 配置文件

```
YAML
server:
  servlet:
    context-path: /checkcode
  port: 63075
```

注意修改 bootstrap.yml 中的命名空间为自己定义的命名空间。

配置 redis-dev.yaml，保存 redis 服务启动

• Data ID: redis-dev.yaml

• Group: xuecheng-plus-common

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容:

```
1 spring:
2   redis:
3     host: 192.168.101.65
4     port: 6379
5     password: redis
6     database: 0
7     lettuce:
8       pool:
9         max-active: 20
10        max-idle: 10
11        min-idle: 0
12    timeout: 10000
13    # redisson:
14    # 配置文件目录
15    # config: classpath:singleServerConfig.yaml
```

内容如下:

XML

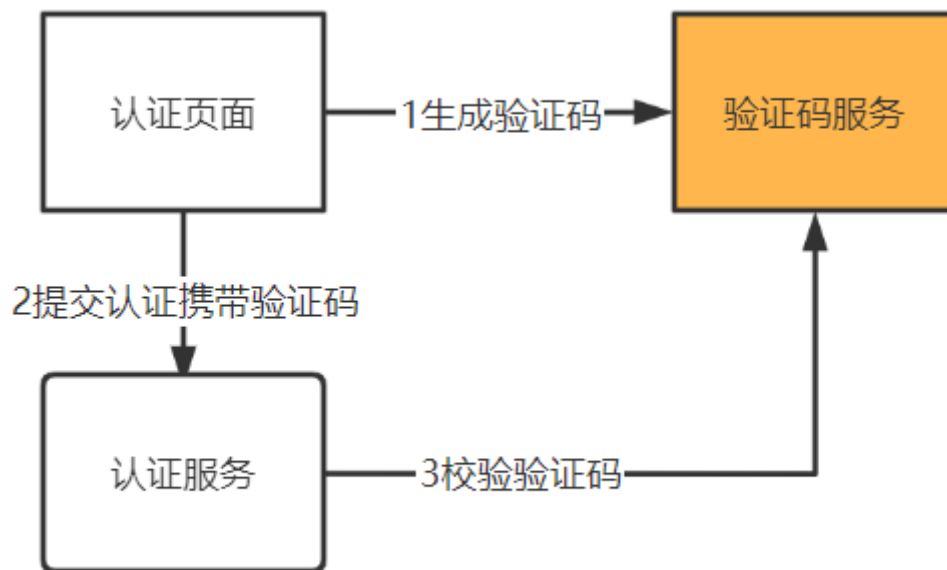
```
spring:
  redis:
    host: 192.168.101.65
    port: 6379
    password: redis
    database: 0
    lettuce:
      pool:
        max-active: 20
        max-idle: 10
        min-idle: 0
    timeout: 10000
    #redisson:
    #配置文件目录
    #config: classpath:singleServerConfig.yaml
```

3.4.2 验证码接口测试

验证码服务对外提供的接口有:

- 1、生成验证码
- 2、校验验证码。

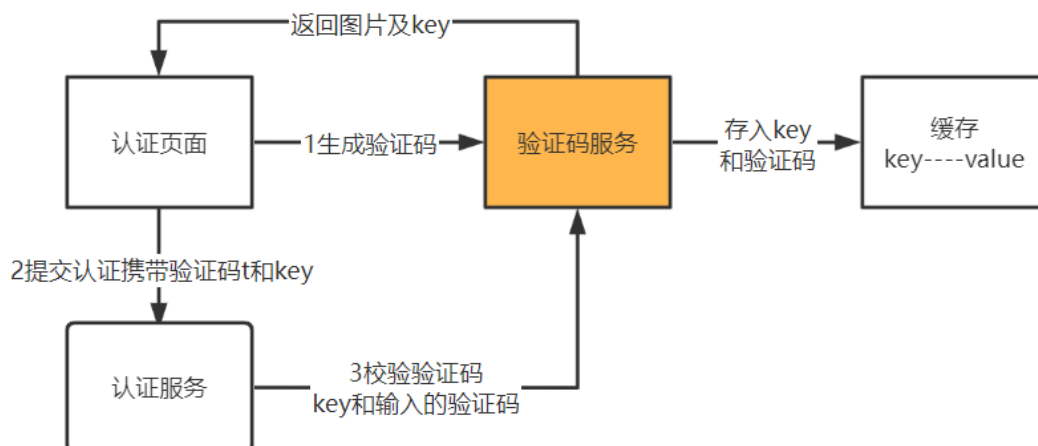
如下:



验证码服务如何生成并校验验证码？

拿图片验证码举例：

- 1、先生成一个指定位数的验证码，根据需要可能是数字、数字字母组合或文字。
- 2、根据生成的验证码生成一个图片并返回给页面
- 3、给生成的验证码分配一个 key，将 key 和验证码一同存入缓存。这个 key 和图片一同返回给页面。
- 4、用户输入验证码，连同 key 一同提交至认证服务。
- 5、认证服务拿 key 和输入的验证码请求验证码服务去校验
- 6、验证码服务根据 key 从缓存取出正确的验证码和用户输入的验证码进行比对，如果相同则校验通过，否则不通过。



根据接口分析，验证码服务接口如下：

Java

```
package com.xuecheng.checkcode.controller;

import com.xuecheng.base.model.RestResponse;
import com.xuecheng.checkcode.model.CheckCodeParamsDto;
import com.xuecheng.checkcode.model.CheckCodeResultDto;
import com.xuecheng.checkcode.service.CheckCodeService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.Map;

/**
 * @author Mr.M
 * @version 1.0
 * @description 验证码服务接口
 * @date 2022/9/29 18:39
 */
@Api(value = "验证码服务接口")
@RestController
public class CheckCodeController {

    @ApiOperation(value="生成验证信息", notes="生成验证信息")
    @PostMapping(value = "/pic")
    public CheckCodeResultDto
generatePicCheckCode(CheckCodeParamsDto checkCodeParamsDto){

    }

    @ApiOperation(value="校验", notes="校验")
    @ApiImplicitParams({
        @ApiImplicitParam(name = "name", value = "业务名称",
required = true, dataType = "String", paramType="query"),
        @ApiImplicitParam(name = "key", value = "验证 key",
required = true, dataType = "String", paramType="query"),
        @ApiImplicitParam(name = "code", value = "验证码",
required = true, dataType = "String", paramType="query")
    })
}
```

```
    })  
    @PostMapping(value = "/verify")  
    public Boolean verify(String key, String code){  
  
    }  
}
```

1、生成验证码接口

```
Plain Text  
### 申请验证码  
POST {{checkcode_host}}/checkcode/pic
```

2、校验验证码接口

根据生成验证码返回的 **key** 以及日志中输出正确的验证码去测试。

```
Bash  
### 校验验证码  
POST  
{{checkcode_host}}/checkcode/verify?key=checkcode4506b95bddbe46cdb  
0d56810b747db1b&code=70d1
```

3.5 账号密码认证

3.5.1 需求分析

到目前为止账号和密码认证所需要的技术、组件都已开发完毕，下边实现账号密码认证，输出如下图：

[登录](#)
[注册](#)

账户名

登录密码

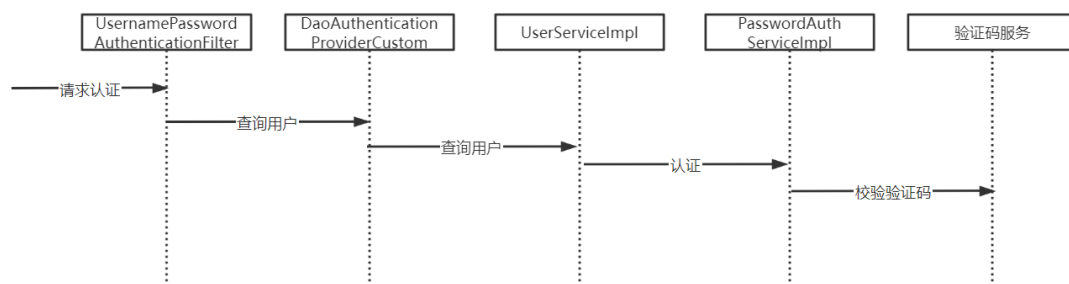
验证码



☐ 自动登录
 [忘记密码](#)

登录

执行流程如下：



3.5.2 账号密码认证开发

1、在认证服务定义远程调用验证码服务的接口

```

Java
package com.xuecheng.ucenter.feignclient;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.PostMapping;

```

```
import org.springframework.web.bind.annotation.RequestParam;

/**
 * @description
 * @author Mr.M
 * @date 2022/9/20 20:29
 * @version 1.0
 */
@FeignClient(value = "checkcode", fallbackFactory =
CheckCodeClientFactory.class)
@RequestMapping("/checkcode")
public interface CheckCodeClient {

    @PostMapping(value = "/verify")
    public Boolean verify(@RequestParam("key") String
key, @RequestParam("code") String code);

}
```

CheckCodeClientFactory:

```
Java
@Slf4j
@Component
public class CheckCodeClientFactory implements
FallbackFactory<CheckCodeClient> {
    @Override
    public CheckCodeClient create(Throwable throwable) {
        return new CheckCodeClient() {

            @Override
            public Boolean verify(String key, String code) {
                log.debug("调用验证码服务熔断异常:{}",
throwable.getMessage());
                return null;
            }
        };
    }
}
```

启动类添加:

```
Java
@EnableFeignClients(basePackages={"com.xuecheng.*.feignclient"})
```

配置文件引入 feign-dev.yaml

Java

```
- data-id: feign-${spring.profiles.active}.yaml
  group: xuecheng-plus-common
  refresh: true
```

2、完善 PasswordAuthServiceImpl

Java

```
package com.xuecheng.ucenter.service.impl;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.xuecheng.ucenter.feignclient.CheckCodeClient;
import com.xuecheng.ucenter.mapper.XcUserMapper;
import com.xuecheng.ucenter.model.dto.AuthParamsDto;
import com.xuecheng.ucenter.model.po.XcUser;
import com.xuecheng.ucenter.service.AuthService;
import org.apache.commons.lang.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

/**
 * @description 账号密码认证
 * @author Mr.M
 * @date 2022/9/29 12:12
 * @version 1.0
 */
@Service("password_authservice")
public class PasswordAuthServiceImpl implements AuthService {

    @Autowired
    XcUserMapper xcUserMapper;

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    CheckCodeClient checkCodeClient;
```

```

@Override
public XcUser execute(AuthParamsDto authParamsDto) {

    //校验验证码
    String checkcode = authParamsDto.getCheckcode();
    String checkcodekey = authParamsDto.getCheckcodekey();

    if(StringUtils.isBlank(checkcodekey) ||
    StringUtils.isBlank(checkcode)){
        throw new RuntimeException("验证码为空");
    }

    Boolean verify = checkCodeClient.verify(checkcodekey,
    checkcode);
    if(!verify){
        throw new RuntimeException("验证码输入错误");
    }
    //账号
    String username = authParamsDto.getUsername();
    XcUser user = xcUserMapper.selectOne(new
    LambdaQueryWrapper<XcUser>().eq(XcUser::getUsername, username));
    if(user==null){
        //返回空表示用户不存在
        throw new RuntimeException("账号不存在");
    }
    //校验密码
    //取出数据库存储的正确密码
    String passwordDb =user.getPassword();
    String passwordForm = authParamsDto.getPassword();
    boolean matches = passwordEncoder.matches(passwordForm,
    passwordDb);
    if(!matches){
        throw new RuntimeException("账号或密码错误");
    }
    return user;
}
}

```

小技巧：目前账号密码方式添加了验证码校验，为了后期获取令牌方便可以重新定义一个不需要验证码校验的认证类 `AuthService`，`AuthService` 中去掉验证码的校验，方便生成令牌。

3.5.3 账号密码认证测试

1、使用浏览器访问 <http://www.51xuecheng.cn/sign.html>



2、首先测试验证码，分别输入正确的验证码和错误的验证码进行测试

3、输入正确的账号密码和错误的账号密码进行测试

登录成功将 jwt 令牌存储 cookie.

4、测试自动登录

勾选自动登录 cookie 生成时间为 30 天，不勾选自动登录关闭浏览器窗口后自动删除 cookie。

4 微信扫码登录

4.1 接入规范

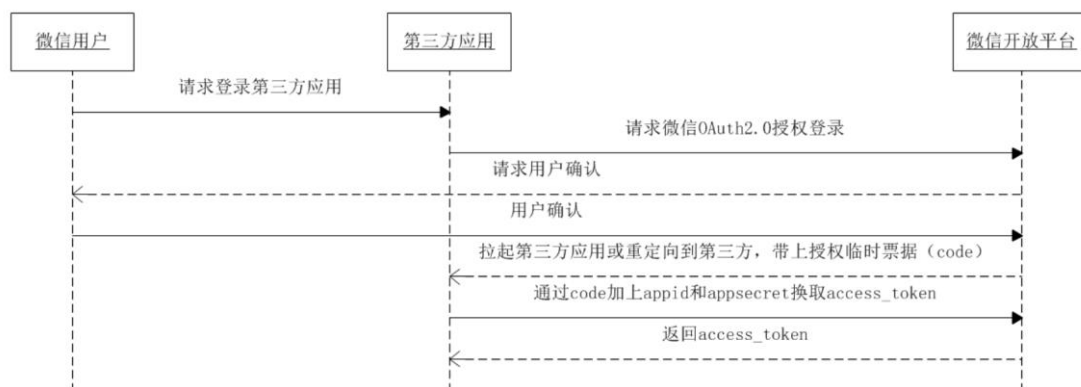
4.1.1 接入流程

微信扫码登录基于 OAuth2 协议的授权码模式，

接口文档：

https://developers.weixin.qq.com/doc/oplatform/Website_App/WeChat_Login/Wechat_Login.html

流程如下：



第三方应用获取 `access_token` 令牌后即可请求微信获取用户的信息，成功获取到用户的信息表示用户在第三方应用认证成功。

4.1.2 请求获取授权码

第三方使用网站应用授权登录前请注意已获取相应网页授权作用域

（`scope=snsapi_login`），则可以通过在 PC 端打开以下链接：

https://open.weixin.qq.com/connect/qrcode?appid=APPID&redirect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE#wechat_redirect 若提示“该链接无法访问”，请检查参数是否填写错误，如 `redirect_uri` 的域名与审核时填写的授权域名不一致或 `scope` 不为 `snsapi_login`。

参数说明

参数	是否必须	说明
appid	是	应用唯一标识
redirect_uri	是	请使用 <code>urlencode</code> 对链接进行处理
response_type	是	填code
scope	是	应用授权作用域，拥有多个作用域用逗号（,）分隔，网页应用目前仅填写 <code>snsapi_login</code>
state	否	用于保持请求和回调的状态，授权请求后原样带回给第三方。该参数可用于防止 csrf 攻击（跨站请求伪造攻击），建议第三方带上该参数，可设置为简单的随机数加 session 进行校验
lang	否	界面语言，支持cn（中文简体）与en（英文），默认为cn

返回说明

用户允许授权后，将会重定向到 `redirect_uri` 的网址上，并且带上 `code` 和 `state` 参数

Plain Text

```
redirect_uri?code=CODE&state=STATE
```

若用户禁止授权，则不会发生重定向。

登录一号店网站应用 <https://test.yhd.com/wechat/login.do> 打开后，一号店会生成 state 参数，跳转到

https://open.weixin.qq.com/connect/qrcode?appid=wxdbc5610cc59c1631&redirect_uri=https%3A%2F%2Fpassport.yhd.com%2Fwechat%2Fcallback.do&response_type=code&scope=snsapi_login&state=3d6be0a4035d839573b04816624a415e#wechat_redirect 微信用户使用微信扫码二维码并且确认登录后，PC 端会跳转到

<https://test.yhd.com/wechat/callback.do?code=CODE&state=3d6be0a4035d839573b04816624a415e> 为了满足网站更定制化的需求，我们还提供了第二种获取 code 的方式，支持网站将微信登录二维码内嵌到自己页面中，用户使用微信扫码授权后通过 JS 将 code 返回给网站。JS 微信登录主要用途：网站希望用户在网站内就能完成登录，无需跳转到微信域下登录后再返回，提升微信登录的流畅性与成功率。网站内嵌二维码微信登录 JS 实现办法：

步骤 1：在页面中先引入如下 JS 文件（支持 https）：

Plain Text

```
http://res.wx.qq.com/connect/zh_CN/htmledition/js/wxLogin.js
```

步骤 2：在需要使用微信登录的地方实例以下 JS 对象：

Plain Text

```
var obj = new WxLogin({
  self_redirect:true,
  id:"login_container",
  appid: "",
  scope: "",
  redirect_uri: "",
  state: "",
  style: "",
  href: ""
});
```

参数	是否必须	说明
self_redirect	否	true: 手机点击确认登录后可以在 iframe 内跳转到 redirect_uri, false: 手机点击确认登录后可以在 top window 跳转到 redirect_uri。默认为 false。
id	是	第三方页面显示二维码的容器id
appid	是	应用唯一标识, 在微信开放平台提交应用审核通过后获得
scope	是	应用授权作用域, 拥有多个作用域用逗号 (,) 分隔, 网页应用目前仅填写 snsapi_login即可
redirect_uri	是	重定向地址, 需要进行UrlEncode
state	否	用于保持请求和回调的状态, 授权请求后原样带回给第三方。该参数可用于防止 csrf 攻击 (跨站请求伪造攻击), 建议第三方带上该参数, 可设置为简单的随机数加 session 进行校验
style	否	提供"black"、"white"可选, 默认为黑色文字描述。详见文档底部FAQ
href	否	自定义样式链接, 第三方可根据实际需求覆盖默认样式。详见文档底部FAQ

4.1.3 通过 code 获取 access_token

通过 code 获取 access_token

Plain Text
https://api.weixin.qq.com/sns/oauth2/access_token?appid=APPID&secret=SECRET&code=CODE&grant_type=authorization_code

参数	是否必须	说明
appid	是	应用唯一标识, 在微信开放平台提交应用审核通过后获得
secret	是	应用密钥AppSecret, 在微信开放平台提交应用审核通过后获得
code	是	填写第一步获取的 code 参数
grant_type	是	填authorization_code

返回说明

正确的返回：

```
Plain Text
{
  "access_token":"ACCESS_TOKEN",
  "expires_in":7200,
  "refresh_token":"REFRESH_TOKEN",
  "openid":"OPENID",
  "scope":"SCOPE",
  "unionid": "o6_bmasdasdsad6_2sgVt7hMZOPfL"
}
```

参数说明

参数	说明
access_token	接口调用凭证
expires_in	access_token接口调用凭证超时时间，单位（秒）
refresh_token	用户刷新access_token
openid	授权用户唯一标识
scope	用户授权的作用域，使用逗号（,）分隔
unionid	当且仅当该网站应用已获得该用户的userinfo授权时，才会出现该字段。

错误返回样例：

```
Plain Text
{"errcode":40029,"errmsg":"invalid code"}
```

4.1.4 通过 access_token 调用接口

获取 access_token 后，进行接口调用，有以下前提：

```
Plain Text
access_token 有效且未超时；
微信用户已授权给第三方应用帐号相应接口作用域（scope）。
```

对于接口作用域（scope），能调用的接口有以下：

授权作用域（scope）	接口	接口说明
snsapi_base	/sns/oauth2/access_token	通过 code 换取access_token、refresh_token和已授权scope
snsapi_base	/sns/oauth2/refresh_token	刷新或续期access_token使用
snsapi_base	/sns/auth	检查access_token有效性
snsapi_userinfo	/sns/userinfo	获取用户个人信息

其中 snsapi_base 属于基础接口，若应用已拥有其它 scope 权限，则默认拥有 snsapi_base 的权限。使用 snsapi_base 可以让移动端网页授权绕过跳转授权登录页请求用户授权的动作，直接跳转第三方网页带上授权临时票据（code），但会使得用户已授权作用域（scope）仅为 snsapi_base，从而导致无法获取到需要用户授权才允许获得的数据和基础功能。接口调用方法可查阅《[微信授权关系接口调用指南](#)》

获取用户信息接口文档：

https://developers.weixin.qq.com/doc/oplatform/Website_App/WeChat_Login/Authorized_Interface_Calling_UnionID.html

接口地址

Plain Text

http 请求方式：GET

https://api.weixin.qq.com/sns/userinfo?access_token=ACCESS_TOKEN&openid=OPENID

如下：

请求说明

http请求方式: GET
https://api.weixin.qq.com/sns/userinfo?access_token=ACCESS_TOKEN&openid=OPENID

参数说明

参数	是否必须	说明
access_token	是	调用凭证
openid	是	普通用户的标识, 对当前开发者帐号唯一
lang	否	国家地区语言版本, zh_CN 简体, zh_TW 繁体, en 英语, 默认为en

响应:

```
Java
{
  "openid": "OPENID",
  "nickname": "NICKNAME",
  "sex": 1,
  "province": "PROVINCE",
  "city": "CITY",
  "country": "COUNTRY",
  "headimgurl":
  "https://thirdwx.qlogo.cn/mmopen/g3MonUZtNHkdmzicIlibx6iaFqAc56vxl
  SUfpb6n5WKSYVY0ChQKkiaJSgQ1dZuTOgvLLrhJbERQQ4eMsv84eavHiaiceqxibJx
  CfHe/0",
  "privilege": [
    "PRIVILEGE1",
    "PRIVILEGE2"
  ],
  "unionid": " o6_bmasdasdsad6_2sgVt7hMZOPfL"
}
```

说明如下:

Plain Text

参数	说明
openid	普通用户的标识, 对当前开发者帐号唯一
nickname	普通用户昵称
sex	普通用户性别, 1 为男性, 2 为女性

province	普通用户个人资料填写的省份
city	普通用户个人资料填写的城市
country	国家，如中国为 CN
headimgurl	用户头像，最后一个数值代表正方形头像大小（有 0、46、64、96、132 数值可选，0 代表 640*640 正方形头像），用户没有头像时该项为空
privilege	用户特权信息，json 数组，如微信沃卡用户为（chinaunicom）
unionid	用户统一标识。针对一个微信开放平台帐号下的应用，同一用户的 unionid 是唯一的。

4.2 准备开发环境

4.2.1 添加应用

1、注册微信开放平台

<https://open.weixin.qq.com/>

2、添加应用

进入网站应用，添加应用



3、添加应用需要指定一个外网域名作为微信回调域名

审核通过后，生成 app 密钥。

最终获取 appId 和 AppSecret

4.2.2 内网穿透

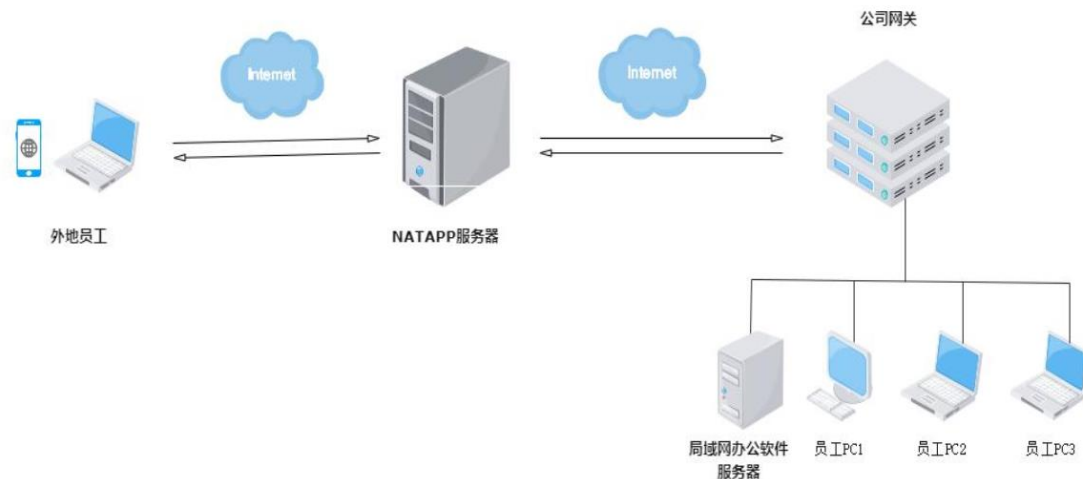
我们的开发环境在局域网，微信回调指向一个公网域名。

如何让微信回调请求至我们的开发计算机上呢？

可以使用内网穿透技术，什么是内网穿透？

内网穿透简单来说就是将内网外网通过隧道打通,让内网的数据让外网可以获取。比如常用的办公室软件等，一般在办公室或家里，通过拨号上网，这样办公软件只有在本

地的局域网之内才能访问，那么问题来了，如果是手机上，或者公司外地的办公人员，如何访问到办公软件呢？这就需要内网穿透工具了。开启隧道之后，网穿透工具会分配一个专属域名/端口,办公软件就已经在公网上了,在外地的办公人员可以在任何地方愉快的访问办公软件了~~



1、在内网穿透服务器上开通隧道，配置外网域名，配置穿透内网的端口即本地电脑上的端口。

本地地址:

127.0.0.1

默认127.0.0.1 可改为其他内网地址

本地端口:

63070

映射到本地的端口 如127.0.0.1:8080 则输入8080

这里我们配置认证服务端口，最终实现通过外网域名访问本地认证服务。

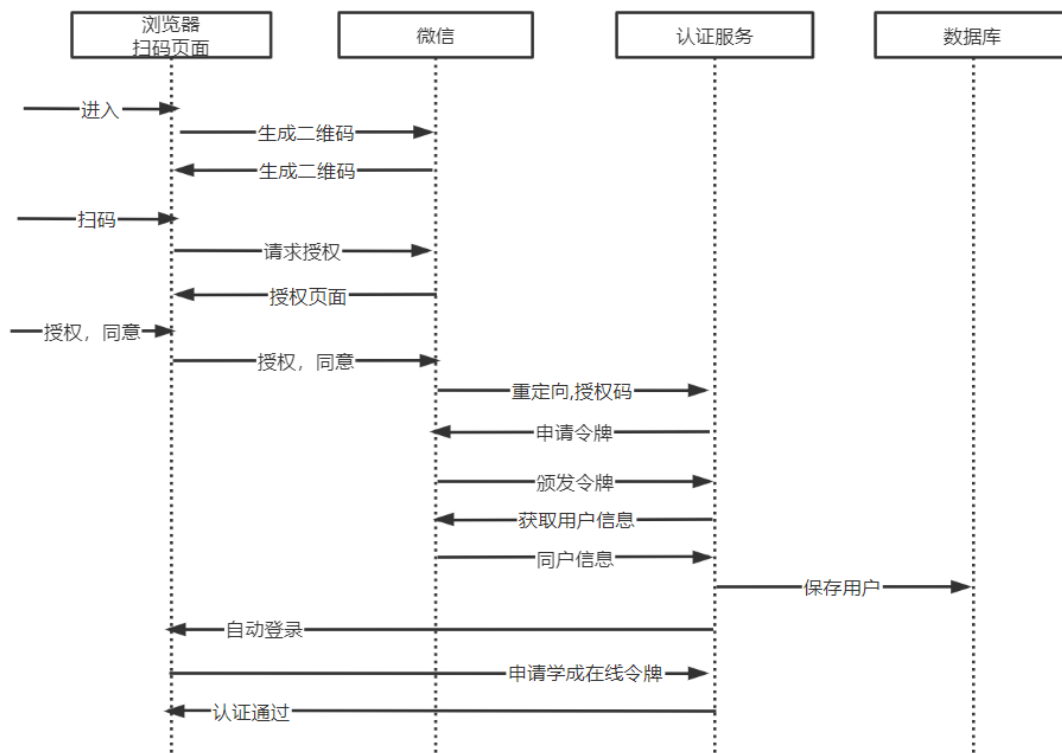
2、在本地电脑上安装内网穿透的工具，工具上配置内网穿透服务器隧道 token。

市面上做内网穿透的商家很多，需要时可以查阅资料了解下。

4.3 接入微信登录

4.3.1 接入分析

根据 OAuth2 协议授权码流程，结合本项目自身特点，分析接入微信扫码登录的流程如下：



本项目认证服务需要做哪些事？

- 1、需要定义接口接收微信下发的授权码。
- 2、收到授权码调用微信接口申请令牌。
- 3、申请到令牌调用微信获取用户信息
- 4、获取用户信息成功将其写入本项目用户中心数据库。
- 5、最后重定向到浏览器自动登录。

4.3.2 定义接口

参考接口规范中“请求获取授权码”定义接收微信下发的授权码接口，

定义 WxLoginController 类，如下：

```

Java
@Slf4j
@Controller
public class WxLoginController {

    @RequestMapping("/wxLogin")
    public String wxLogin(String code, String state) throws
    IOException {
        log.debug("微信扫码回调,code:{},state:{},code,state);
  
```

数据库

//请求微信申请令牌，拿到令牌查询用户信息，将用户信息写入本项目数据库

```
XcUser xcUser = new XcUser();
//暂时硬编写，目的是调试环境
xcUser.setUsername("t1");
if(xcUser==null){
    return "redirect:http://www.51xuecheng.cn/error.html";
}
String username = xcUser.getUsername();
return
"redirect:http://www.51xuecheng.cn/sign.html?username="+username+
&authType=wx";
}
```

定义微信认证的 service

```
Java
package com.xuecheng.ucenter.service.impl;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.UUID;

/**
 * @author Mr.M
 * @version 1.0
 * @description 微信扫码认证
 * @date 2022/9/29 12:12
 */
@Slf4j
@Service("wx_authservice")
public class WxAuthServiceImpl implements AuthService {

    @Autowired
    XcUserMapper xcUserMapper;

    @Override
    public XcUserExt execute(AuthParamsDto authParamsDto) {

        //账号
        String username = authParamsDto.getUsername();
```

```
XcUser user = xcUserMapper.selectOne(new
LambdaQueryWrapper<XcUser>().eq(XcUser::getUsername, username));
if(user==null){
    //返回空表示用户不存在
    throw new RuntimeException("账号不存在");
}
XcUserExt xcUserExt = new XcUserExt();
BeanUtils.copyProperties(user,xcUserExt);
return xcUserExt;
}
}
```

4.3.3 接口环境测试

接口定义好下边进行测试下，主要目的是测试接口调度的环境。

- 1、启动内网穿透工具
- 2、在/wxLogin 接口中打断点
- 3、打开前端微信扫码页面



点击微信图标打开二维码

[扫码登录](#)

[注册](#)

微信登录



用户扫码，确认授权

此时正常进入 /wxLogin 方法，最后跳转到

<http://www.51xuecheng.cn/sign.html?username=t1&authType=wx>。

4.3.4 接入微信认证

接下来请求微信申请令牌。

1、使用 restTemplate 请求微信，配置 RestTemplate bean

在启动类配置 restTemplate

```
Java
@Bean
RestTemplate restTemplate(){
    RestTemplate restTemplate = new RestTemplate(new
OkHttpClientHttpRequestFactory());
    return  restTemplate;
}
```

2、定义与微信认证的 service 接口：

```
Java
/**
 * @author Mr.M
 * @version 1.0
 * @description 微信认证接口
 * @date 2023/2/21 22:15
 */
```

```
public interface WxAuthService {

    public XcUser wxAuth(String code);

}
```

3、下边在 controller 中调用 wxAuth 接口：

```
Java
@Slf4j
@Controller
public class WxLoginController {

    @Autowired
    WxAuthService wxAuthService;

    @RequestMapping("/wxLogin")
    public String wxLogin(String code, String state) throws
IOException {
        log.debug("微信扫码回调,code:{},state:{},code,state);
        //请求微信申请令牌，拿到令牌查询用户信息，将用户信息写入本项目数
        据库
        XcUser xcUser = wxAuthService.wxAuth(code);
        if(xcUser==null){
            return "redirect:http://www.51xuecheng.cn/error.html";
        }
        String username = xcUser.getUsername();
        return
        "redirect:http://www.51xuecheng.cn/sign.html?username="+username+"
        &authType=wx";
    }
}
```

4、在 WxAuthService 的 wxAuth 方法中实现申请令牌、查询用户信息等内容。

```
Java

@Slf4j
@Service("wx_authservice")
public class WxAuthServiceImpl implements AuthService,
WxAuthService {
    @Autowired
    XcUserMapper xcUserMapper;
```

```

@Autowired
RestTemplate restTemplate;

@Value("${weixin.appid}")
String appid;
@Value("${weixin.secret}")
String secret;

public XcUser wxAuth(String code){

    //收到 code 调用微信接口申请 access_token
    Map<String, String> access_token_map = getAccess_token(code);
    if(access_token_map==null){
        return null;
    }
    System.out.println(access_token_map);
    String openid = access_token_map.get("openid");
    String access_token = access_token_map.get("access_token");
    //拿 access_token 查询用户信息
    Map<String, String> userinfo = getUserinfo(access_token,
openid);
    if(userinfo==null){
        return null;
    }
    //添加用户到数据库
    XcUser xcUser = null;

    return xcUser;
}

/**
 * 申请访问令牌,响应示例
 * {
 * "access_token":"ACCESS_TOKEN",
 * "expires_in":7200,
 * "refresh_token":"REFRESH_TOKEN",
 * "openid":"OPENID",
 * "scope":"SCOPE",
 * "unionid": "o6_bmasdasdsad6_2sgVt7hMZOPfL"
 * }
 */
private Map<String,String> getAccess_token(String code) {

    String wxUrl_template =

```

```

"https://api.weixin.qq.com/sns/oauth2/access_token?appid=%s&secret=%s&code=%s&grant_type=authorization_code";
    //请求微信地址
    String wxUrl = String.format(wxUrl_template, appid, secret, code);

    Log.info("调用微信接口申请 access_token, url:{}", wxUrl);

    ResponseEntity<String> exchange = restTemplate.exchange(wxUrl,
        HttpMethod.POST, null, String.class);

    String result = exchange.getBody();
    Log.info("调用微信接口申请 access_token: 返回值:{}", result);
    Map<String,String> resultMap = JSON.parseObject(result,
        Map.class);

    return resultMap;
}

/**获取用户信息，示例如下：
{
  "openid":"OPENID",
  "nickname":"NICKNAME",
  "sex":1,
  "province":"PROVINCE",
  "city":"CITY",
  "country":"COUNTRY",
  "headimgurl":
"https://thirdwx.qlogo.cn/mmopen/g3MonUZtNHkdmzicIlibx6iaFqAc56vxl
SUfpb6n5WKSyVY0ChQKkiaJSgQ1dZuTOgvLLrhJbERQQ4eMsv84eavHiaiceqxibJx
CfHe/0",
  "privilege":[
    "PRIVILEGE1",
    "PRIVILEGE2"
  ],
  "unionid": " o6_bmasdasdsad6_2sgVt7hMZOPfL"
}
*/
private Map<String,String> getUserinfo(String access_token,String
openid) {

    String wxUrl_template =
"https://api.weixin.qq.com/sns/userinfo?access_token=%s&openid=%s"
;

```

```

        //请求微信地址
        String wxUrl = String.format(wxUrl_template,
access_token,openid);

        Log.info("调用微信接口申请 access_token, url:{})", wxUrl);

        ResponseEntity<String> exchange = restTemplate.exchange(wxUrl,
HttpMethod.POST, null, String.class);

        //防止乱码进行转码
        String result = new
String(exchange.getBody().getBytes(StandardCharsets.ISO_8859_1),St
andardCharsets.UTF_8);
        Log.info("调用微信接口申请 access_token: 返回值:{})", result);
        Map<String,String> resultMap = JSON.parseObject(result,
Map.class);

        return resultMap;
    }
    ....

```

测试获取用户信息

- 1、在获取用户信息处打断点
- 2、进入 <http://www.51xuecheng.cn/wxsign.html>
- 3、手机扫码授权

4.3.5 保存用户信息

向数据库保存用户信息，如果用户不存在将其保存在数据库。

```

Java
@Autowired
XcUserRoleMapper xcUserRoleMapper;

@Transactional
public XcUser addWxUser(Map userInfo_map){
    String unionid = userInfo_map.get("unionid").toString();
    //根据 unionid 查询数据库
    XcUser xcUser = xcUserMapper.selectOne(new

```



```

LambdaQueryWrapper<XcUser>().eq(XcUser::getWxUnionid, unionid));
    if(xcUser!=null){
        return xcUser;
    }
    String userId = UUID.randomUUID().toString();
    xcUser = new XcUser();
    xcUser.setId(userId);
    xcUser.setWxUnionid(unionid);
    //记录从微信得到的昵称
    xcUser.setNickname(userInfo_map.get("nickname").toString());
    xcUser.setUserpic(userInfo_map.get("headimgurl").toString());
    xcUser.setName(userInfo_map.get("nickname").toString());
    xcUser.setUsername(unionid);
    xcUser.setPassword(unionid);
    xcUser.setUtype("101001");//学生类型
    xcUser.setStatus("1");//用户状态
    xcUser.setCreateTime(LocalDateTime.now());
    xcUserMapper.insert(xcUser);
    XcUserRole xcUserRole = new XcUserRole();
    xcUserRole.setId(UUID.randomUUID().toString());
    xcUserRole.setUserId(userId);
    xcUserRole.setRoleId("17");//学生角色
    xcUserRoleMapper.insert(xcUserRole);
    return xcUser;
}

```

调用保存用户信息

Java

```

@Autowired
WxAuthServiceImp1 currentProxy;

public XcUser wxAuth(String code){

    //收到 code 调用微信接口申请 access_token
    Map<String, String> access_token_map = getAccess_token(code);
    if(access_token_map==null){
        return null;
    }
    System.out.println(access_token_map);
    String openid = access_token_map.get("openid");
    String access_token = access_token_map.get("access_token");
    //拿 access_token 查询用户信息

```

```
    Map<String, String> userinfo = getUserinfo(access_token,
openid);
    if(userinfo==null){
        return null;
    }
    //将用户信息保存到数据库
    XcUser xcUser = currentProxy.addWxUser(userinfo);
    return xcUser;
}
```

测试保存用户信息

- 1、在保存用户信息处打断点
- 2、进入 <http://www.51xuecheng.cn/wxsign.html>
- 3、手机扫码授权
- 4、自动跳转到登录页面，提交认证成功。

5 用户授权

5.1 RBAC

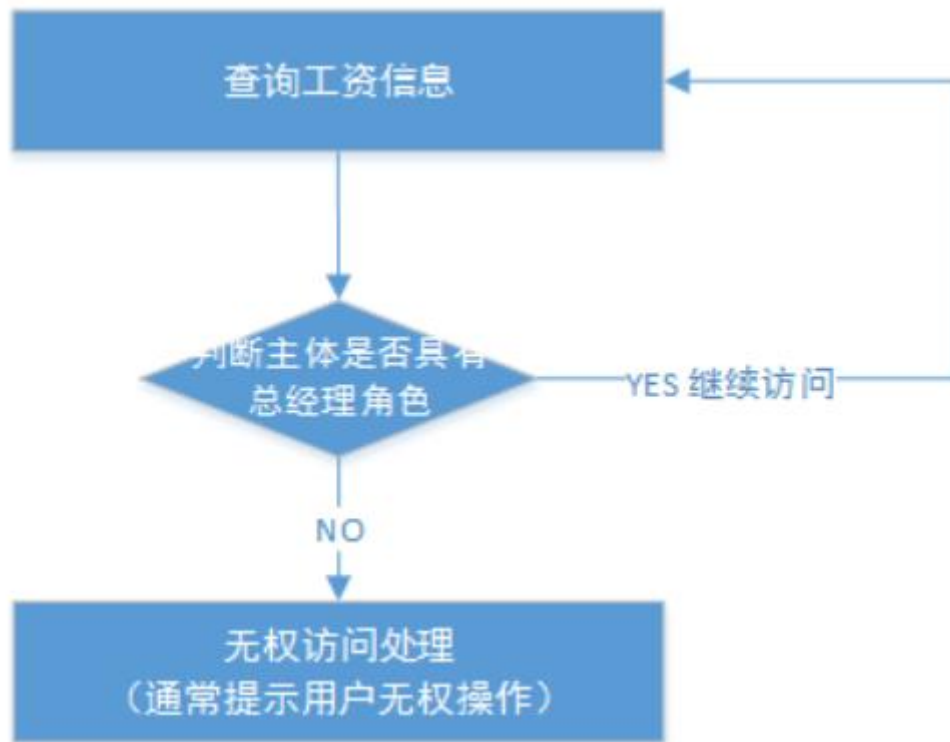
如何实现授权？业界通常基于 RBAC 实现授权。

RBAC 分为两种方式：

基于角色的访问控制（Role-Based Access Control）

基于资源的访问控制（Resource-Based Access Control）

角色的访问控制（Role-Based Access Control）是按角色进行授权，比如：主体的角色为总经理可以查询企业运营报表，查询员工工资信息等，访问控制流程如下：



根据上图中的判断逻辑，授权代码可表示如下：

```
Java
if(主体.hasRole("总经理角色 id")){
    查询工资
}
```

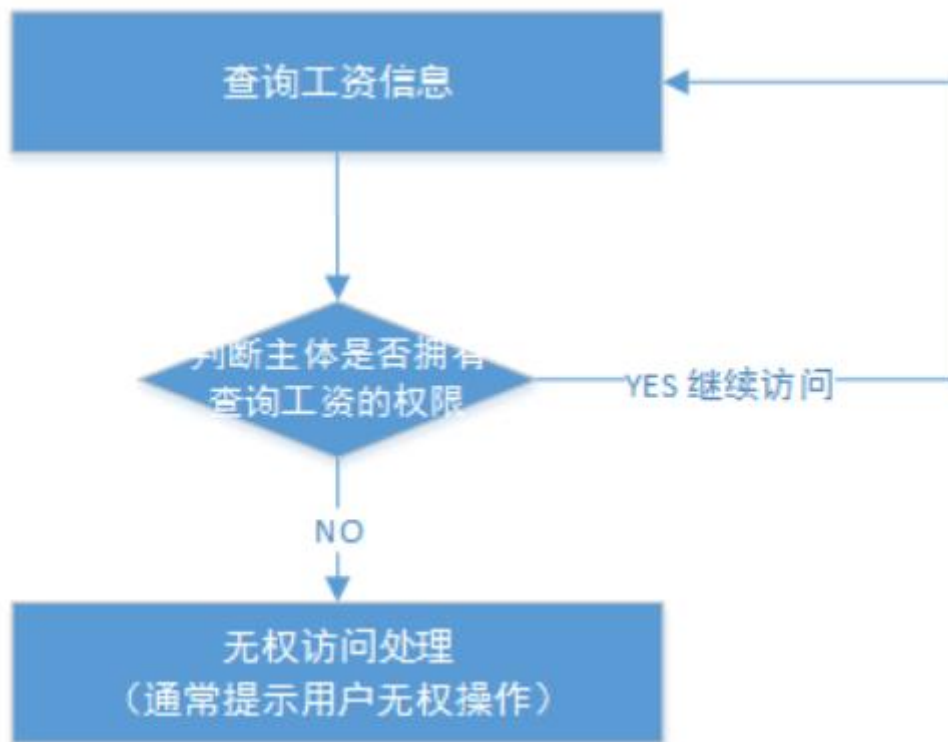
如果上图中查询工资所需要的角色变化为总经理和部门经理，此时就需要修改判断逻辑为“判断用户的角色是否是总经理或部门经理”，修改代码如下：

```
Java
if(主体.hasRole("总经理角色 id") || 主体.hasRole("部门经理角色 id")){
    查询工资
}
```

根据上边的例子发现，当需要修改角色的权限时就需要修改授权的相关代码，系统可扩展性差。

基于资源的访问控制（Resource-Based Access

Control）是按资源（或权限）进行授权，比如：用户必须具有查询工资权限才可以查询员工工资信息等，访问控制流程如下：



根据上图中的判断，授权代码可以表示为：

```
Java
if(主体.hasPermission("查询工资权限标识")){
    查询工资
}
```

优点：系统设计时定义好查询工资的权限标识，即使查询工资所需要的角色变化为总经理和部门经理也不需要修改授权代码，系统可扩展性强。

5.2 资源服务授权流程

本项目在资源服务内部进行授权，基于资源的授权模式，因为接口在资源服务，通过在接口处添加授权注解实现授权。

1、首先配置 nginx 代理

```
Java
http {
    server_names_hash_bucket_size 64;
    ...

    #前端开发服务
```

```

upstream uidevserver{
    server 127.0.0.1:8601 weight=10;
}
server {
    listen      80;
    server_name teacher.51xuecheng.cn;
    #charset koi8-r;
    ssi on;
    ssi_silent_errors on;
    #access_log logs/host.access.log main;
    #location / {
        # alias D:/itcast2022/xc_edu3.0/code_1/dist/;
        # index index.html index.htm;
    #}
    location / {
        proxy_pass http://uidevserver;
    }

    location /api/ {
        proxy_pass http://gatewayserver/;
    }
}

```

加载 nginx 配置。

2、在资源服务集成 Spring Security

在需要授权的接口处使用@PreAuthorize("hasAuthority('权限标识符')")进行控制

下边代码指定/course/list 接口需要拥有 xc_teachmanager_course_list 权限。

```

34 @ApiOperation("课程查询接口")
35 @PreAuthorize("hasAuthority('xc_teachmanager_course_list')")//拥有课程列表查询的权限方可访问
36 @PostMapping("/course/list")
37 public PageResult<CourseBase> list(PageParams pageParams, @RequestBody QueryCourseParamsDto queryCourseParams)
38     return courseBaseInfoService.queryCourseBaseList(pageParams,queryCourseParams);

```

设置了@PreAuthorize 表示执行此方法需要授权，如果当前用户请求接口没有权限则抛出异常

org.springframework.security.access.AccessDeniedException: 不允许访问

3、在统一异常处理处解析此异常信息

```

Java
@ResponseBody

```

```

@ExceptionHandler(Exception.class)
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
public RestErrorResponse exception(Exception e) {

    Log.error("【系统异常】{}",e.getMessage(),e);
    e.printStackTrace();
    if(e.getMessage().equals("不允许访问")){
        return new RestErrorResponse("没有操作此功能的权限");
    }
    return new
RestErrorResponse(CommonError.UNKOWN_ERROR.getErrMsg());

}

```

4、重启资源服务进行测试

使用教学机构用户登录系统

这里使用 t1 用户登录，账号:t1、密码：111111

登录成功，点击“教学机构”



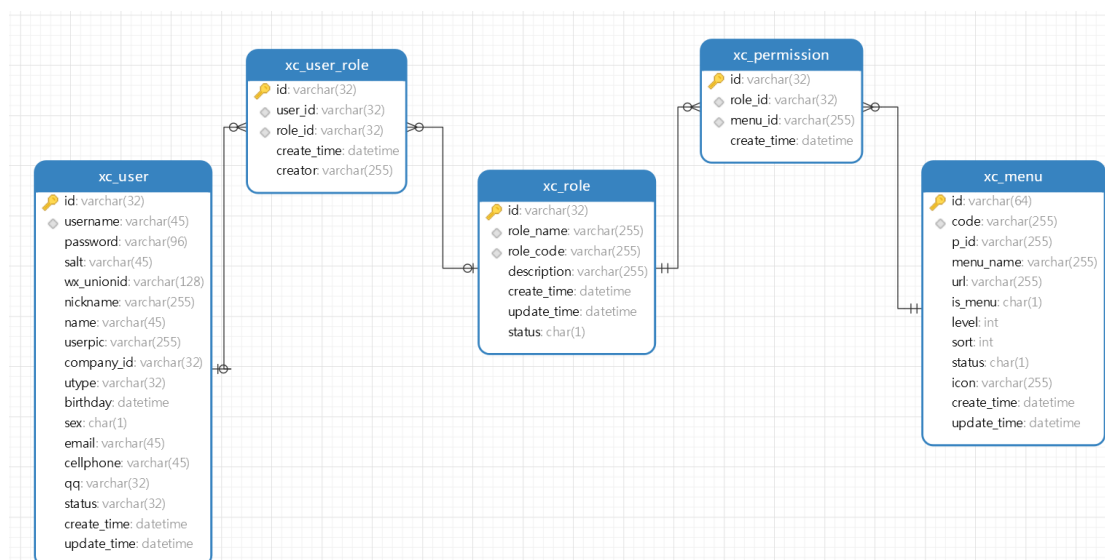
当用户没有权限时页面提示：没有操作此功能的权限。



5.3 授权相关的数据模型

如何给用户分配权限呢？

首先要学习数据模型，本项目授权相关的数据表如下：



说明如下：

xc_user：用户表，存储了系统用户信息，用户类型包括：学生、老师、管理员等

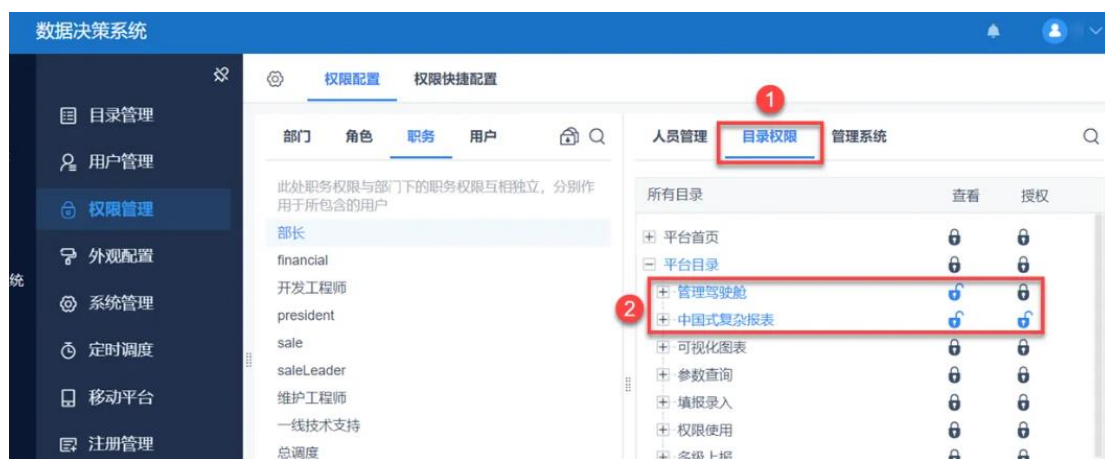
xc_role：角色表，存储了系统的角色信息，学生、老师、教学管理员、系统管理员等。

xc_user_role：用户角色表，一个用户可拥有多个角色，一个角色可被多个用户所拥有

xc_menu:模块表，记录了菜单及菜单下的权限

xc_permission:角色权限表，一个角色可拥有多个权限，一个权限可被多个角色所拥有

本项目教学阶段不再实现权限定义及用户权限分配的功能，权限分配的界面原型如下图所示：



本项目要求掌握基于权限数据模型（5张数据表），要求在数据库中操作完成给用户分配权限、查询用户权限等需求。

1、查询用户所拥有的权限

步骤：

查询用户的 id

查询用户所拥有的角色

查询用户所拥有的权限

例子：

```
SELECT * FROM xc_menu WHERE id IN(  
    SELECT menu_id FROM xc_permission WHERE role_id IN(  
        SELECT role_id FROM xc_user_role WHERE user_id = '49'  
    )  
)
```

2、给用户分配权限

1) 添加权限

查询用户的 id

查询权限的 id

查询用户的角色，如果没有角色需要先给用户指定角色

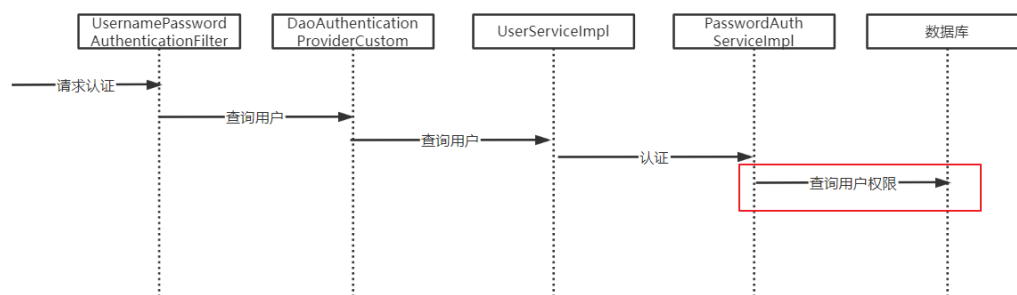
向角色权限表添加记录

2) 删除用户权限

本项目是基于角色分配权限，如果要删除用户的权限可以给用户换角色，那么新角色下的权限就是用户的权限；如果不换用户的角色可以删除角色下的权限即删除角色权限关系表相应记录，这样操作是将角色下的权限删除，属于该角色的用户都将删除此权限。

5.4 查询用户权限

使用 Spring Security 进行授权，首先在生成 jwt 前会查询用户的权限，如下图：



接下来需要修改 UserServiceImpl 和 PasswordAuthServiceImpl 从数据库查询用户的权限。

1、定义 mapper 接口

Java

```
public interface XcMenuMapper extends BaseMapper<XcMenu> {
    @Select("SELECT      * FROM xc_menu WHERE id IN (SELECT menu_id
FROM xc_permission WHERE role_id IN ( SELECT role_id FROM
xc_user_role WHERE user_id = #{userId} ))")
    List<XcMenu> selectPermissionByUserId(@Param("userId") String
userId);
}
```

2、修改 PasswordAuthServiceImpl

修改 UserServiceImpl 类的 getUserPrincipal 方法，查询权限信息

Java

//查询用户身份

```
public UserDetails getUserPrincipal(XcUserExt user){
    String password = user.getPassword();
    //查询用户权限
    List<XcMenu> xcMenus =
menuMapper.selectPermissionByUserId(user.getId());
    List<String> permissions = new ArrayList<>();
    if(xcMenus.size()<=0){
        //用户权限,如果不加则报 Cannot pass a null GrantedAuthority
collection
        permissions.add("p1");
    }else{
        xcMenus.forEach(menu->{
            permissions.add(menu.getCode());
        });
    }
    //将用户权限放在 XcUserExt 中
    user.setPermissions(permissions);

    //为了安全在令牌中不放密码
    user.setPassword(null);
    //将 user 对象转 json
    String userString = JSON.toJSONString(user);
    String[] authorities = permissions.toArray(new String[0]);
    UserDetails userDetails =
User.withUsername(userString).password(password).authorities(autho
rities).build();
    return userDetails;
}
```

```
}
```

5.5 授权测试

以上实现了认证时从数据库查询用户的权限，下边进行用户授权测试。

重启认证服务，使用内容管理课程列表查询为例，代码如下：

```
@ApiOperation("课程查询接口")
@PreAuthorize("hasAuthority('xc_teachmanager_course_list')")//拥有课程列表查询的权限方可访问
@PostMapping("/course/list")
public PageResult<CourseBase> list(PageParams pageParams, @RequestBody QueryCoursePara
    return courseBaseInfoService.queryCourseBaseList(pageParams,queryCourseParams);
```

用户拥有 xc_teachmanager_course_list 权限方可访问课程查询接口。

以用户 stu1 为例，当它没有此权限时页面报“没有此操作的权限”错误



将 xc_teachmanager_course_list 权限分配给用户。

- 1) 首先找到当前用户的角色
- 2) 找到 xc_teachmanager_course_list 权限的主键
- 3) 在角色权限关系表中添加记录

分配完权限需要重新登录

由于用户分配了 xc_teachmanager_course_list 权限，用户具有访问课程查询接口的权限。

5.6 细粒度授权

5.6.1 什么是细粒度授权

什么是细粒度授权？

细粒度授权也叫数据范围授权，即不同的用户所拥有的操作权限相同，但是能够操作的数据范围是不一样的。一个例子：用户 A 和用户 B 都是教学机构，他们都拥有“我的课程”权限，但是两个用户所查询到的数据是不一样的。

本项目有哪些细粒度授权？

比如：

我的课程，教学机构只允许查询本教学机构下的课程信息。

我的选课，学生只允许查询自己所选课。

如何实现细粒度授权？

细粒度授权涉及到不同的业务逻辑，通常在 `service` 层实现，根据不同的用户进行校验，根据不同的参数查询不同的数据或操作不同的数据。

5.6.2 教学机构细粒度授权

教学机构在维护课程时只允许维护本机构的课程，教学机构细粒度授权过程如下：

- 1) 获取当前登录的用户身份
- 2) 得到用户所属教育机构的 `Id`
- 3) 查询该教学机构下的课程信息

最终实现了用户只允许查询自己机构的课程信息。

根据公司 `Id` 查询课程，流程如下：

- 1) 教学机构用户登录系统，从用户身份中取出所属机构的 `id`
在用户表中设计了 `company_id` 字段存储该用户所属的机构 `id`.
- 2) 接口层取出当前登录用户的身份，取出机构 `id`
- 3) 将机构 `id` 传入 `service` 方法。
- 4) `service` 方法将机构 `id` 传入 `Dao` 方法，最终查询出本机构的课程信息。

代码实现如下：

```
Java
@ApiOperation("课程查询接口")
@PreAuthorize("hasAuthority('xc_teachmanager_course_list')")//拥有
课程列表查询的权限方可访问
@PostMapping("/course/list")
public PageResult<CourseBase> list(PageParams pageParams,
@RequestBody QueryCourseParamsDto queryCourseParams){
    //取出用户身份
```

```
XcUser user = SecurityUtil.getUser();
//机构 id
String companyId = user.getCompanyId();
return
courseBaseInfoService.queryCourseBaseList(Long.parseLong(companyId)
),pageParams,queryCourseParams);
}
```

Service 方法如下：

```
Java
@Override
public PageResult<CourseBase> queryCourseBaseList(Long
companyId,PageParams pageParams, QueryCourseParamsDto
queryCourseParamsDto) {

    //构建查询条件对象
    LambdaQueryWrapper<CourseBase> queryWrapper = new
    LambdaQueryWrapper<>();
    //机构 id
    queryWrapper.eq(CourseBase::getCompanyId,companyId);
    ....
}
```

5.6.3 教学机构细粒度授权测试

使用一个教学机构的用户登录项目，并且此用户具有查询课程的权限。

手机修改数据库指定用户归属到一个机构中，涉及以下数据表：

xc_company 为机构表

xc_company_user 为机构用户关系表

xc_user 表中有 company_id 字段。

我们准备了 t1 用户作为此次测试的用户，使用此用户登录系统：

提前在查询课程列表接口处打上断点。

经过测试可以正常取出用户所属的机构 id

```

@ApiOperation("课程查询接口")
@PreAuthorize("hasAuthority('xc_teachmanager_course_list')")//拥有课程列表查询的权限方可访问
@PostMapping("/course/list")
public PageResult<CourseBase> list(PageParams pageParams, @RequestBody QueryCourseParamsDto query
    //取出用户身份
    XcUser user = SecurityUtil.getUser(); user: "XcUser(id=52, username=t1, password=null, salt
    //机构id
    String companyId = user.getCompanyId(); user: "XcUser(id=52, username=t1, password=null, sa
    return courseUserService.queryCourseBaseList(Long.parseLong(companyId),pageParams,queryCo
}
+ "1232141425"

```

跟踪持久层日志发现已将机构 id 传入 dao 方法，拼装 sql 语句，查询本机构的课程信息

```

[http-nio-63040-exec-1][BaseJdbcLogger.java:137] - ==> Preparing:
SELECT COUNT(*) FROM course_base WHERE (company_id = ?)|
2022-10-01 08:14:32,426 DEBUG
[http-nio-63040-exec-1][BaseJdbcLogger.java:137] - ==> Parameters:
1232141425(Long)

```

6 实战

6.1 找回密码(实战)

需求：忘记密码需要找回，可以通过手机号找回密码，通过邮箱找回密码以及人工通道。

界面访问地址：<http://www.51xuecheng.cn/findpassword.html>

找回密码

填写手机号或电子邮箱找回密码，两个都填写优先使用手机找回

手机号码

电子邮箱

验证码

密码

确认密码

接口：

手机验证码：/api/checkcode/phone?param1=手机号

邮箱验证码：/api/checkcode/phone?param1=电子邮箱地址

找回密码：/api/auth/findpassword

请求：

JSON

```
{
  cellphone:'',
  email:'',
  checkcodekey:'',
  checkcode:'',
  confirmpwd:'',
  password:''
}
```

响应：

200: 找回成功

其它：找回失败，失败原因使用统一异常处理返回的信息格式

执行流程

- 1、校验验证码，不一致则抛出异常
- 2、判断两次密码是否一致，不一致则抛出异常
- 3、根据手机号和邮箱查询用户
- 4、如果找到用户更新为新密码

6.2 注册(实战)

需求：为学生提供注册入口，通过此入口注册的用户为学生用户。

界面访问地址：<http://www.51xuecheng.cn/register.html>



欢迎注册学成在线

已经有账号，[去登录](#)

手机号码

验证码

账号

昵称

邮箱

密码

确认密码

☒ 同意协议并注册 [《学成在线注册协议》](#)

接口：

手机验证码：</api/checkcode/phone?param1=手机号>

注册：</api/auth/register>

请求：

JSON

{

```
    cellphone:'',
    username:'',
    email:'',
    nickname:'',
    password:'',
    confirmpwd:'',
    checkcodekey:'',
    checkcode:''
}
```

响应：

200: 注册成功

其它：注册失败，失败原因使用统一异常处理返回的信息格式

执行流程：

- 1、校验验证码，如果不一致则抛出异常
- 2、校验两次密码是否一致，如果不一致则抛出异常
- 3、校验用户是否存在，如果存在则抛出异常
- 4、向用户表、用户角色关系表添加数据。角色为学生角色。