

# 副本 开发SpringBoot+Jwt+Vue的前后端分...

关注公众号 MarkerHub，回复【VueAdmin】可以加群讨论学习、另外还会不定时安排B站视频直播答疑！

一个spring security + jwt + vue的前后端分离项目！综合运用！

首发公众号：MarkerHub

作者：吕一明

视频讲解：<https://www.bilibili.com/video/BV1af4y1s7Wh/>

线上演示：<https://www.markerhub.com/vueadmin/>

转载请保留此应用，万分感谢！

## 1. 前言

从零开始搭建一个项目骨架，最好选择合适熟悉的技术，并且在未来易拓展，适合微服务化体系等。所以一般以Springboot作为我们的框架基础，这是离不开的了。

然后数据层，我们常用的是Mybatis，易上手，方便维护。但是单表操作比较困难，特别是添加字段或减少字段的时候，比较繁琐，所以这里我推荐使用Mybatis Plus

(<https://mp.baomidou.com/>)，为简化开发而生，只需简单配置，即可快速进行 CRUD 操作，从而节省大量时间。

作为一个项目骨架，权限也是我们不能忽略的，上一个项目vueblog我们使用了shiro，但是有些同学想学学SpringSecurity，所以这一期我们使用security作为我们的权限控制和会话控制的框架。

考虑到项目可能需要部署多台，一些需要共享的信息就保存在中间件中，Redis是现在主流的缓存中间件，也适合我们的项目。

然后因为前后端分离，所以我们使用jwt作为我们用户身份凭证，并且session我们会禁用，这样以前传统项目使用的方式我们可能就不再适合使用，这点需要注意了。

ok，我们现在就开始搭建我们的项目脚手架！

技术栈：

- SpringBoot
- mybatis plus
- spring security
- lombok
- redis

- hibernate validation
- jwt

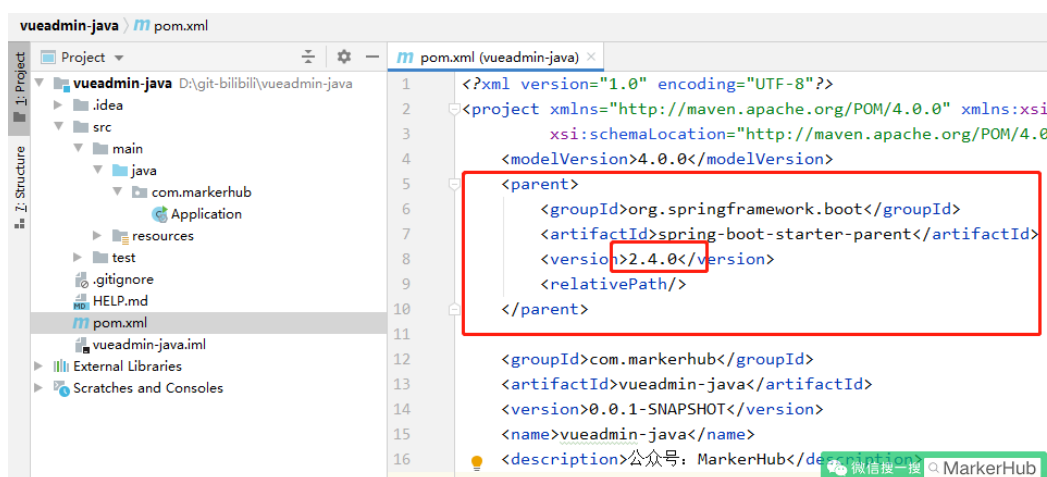
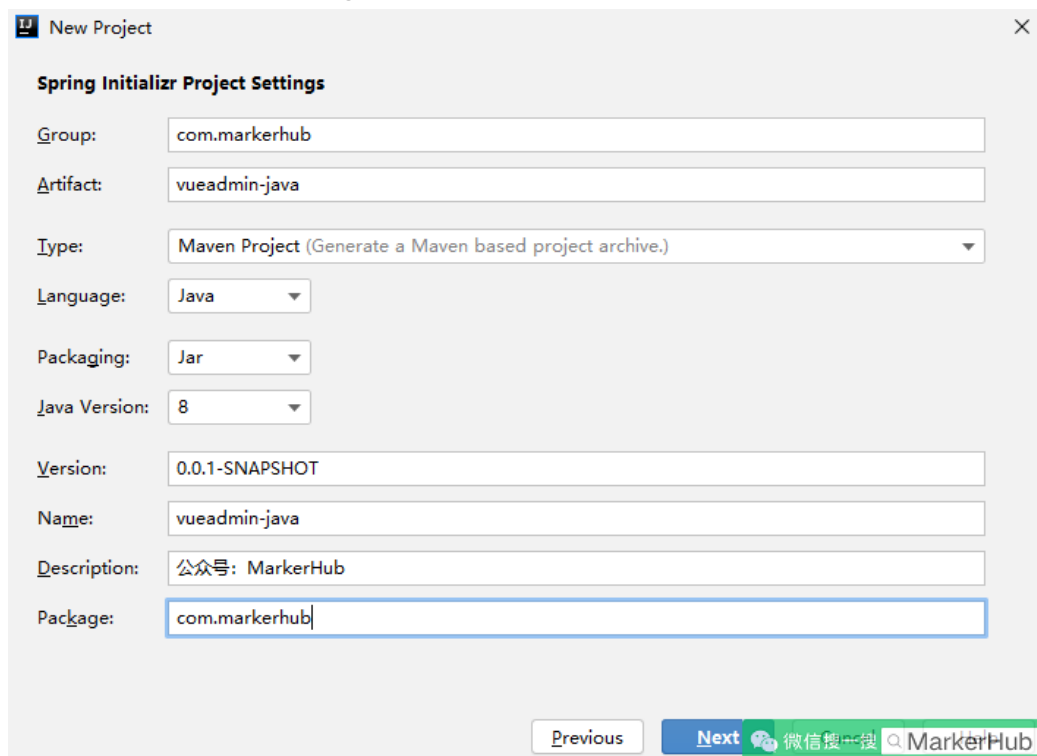
## 2. 新建springboot项目，注意版本

这里，我们使用IDEA来开发我们项目，新建步骤比较简单，我们就不截图了。

开发工具与环境：

- idea
- mysql
- jdk 8
- maven3.3.9

新建好的项目结构如下，SpringBoot版本使用的目前最新的2.4.0版本



pom的jar包导入如下：

```
1 <parent>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-parent</artifactId>
```

```

4     <version>2.4.0</version>
5     <relativePath/>
6 </parent>
7 <groupId>com.markerhub</groupId>
8 <artifactId>vueadmin-java</artifactId>
9 <version>0.0.1-SNAPSHOT</version>
10 <name>vueadmin-java</name>
11 <description>公众号：MarkerHub</description>
12 <properties>
13     <java.version>1.8</java.version>
14 </properties>
15
16 <dependencies>
17     <dependency>
18         <groupId>org.springframework.boot</groupId>
19         <artifactId>spring-boot-starter-web</artifactId>
20     </dependency>
21     <dependency>
22         <groupId>org.springframework.boot</groupId>
23         <artifactId>spring-boot-devtools</artifactId>
24         <scope>runtime</scope>
25         <optional>true</optional>
26     </dependency>
27     <dependency>
28         <groupId>org.projectlombok</groupId>
29         <artifactId>lombok</artifactId>
30         <optional>true</optional>
31     </dependency>
32 </dependencies>

```

- devtools：项目的热加载重启插件
- lombok：简化代码的工具

### 3. 整合mybatis plus，生成代码

接下来，我们来整合mybatis plus，让项目能完成基本的增删改查操作。步骤很简单：可以去官网看看：<https://mp.baomidou.com/guide/>

#### 第一步：导入jar包

pom中导入mybatis plus的jar包，因为后面会涉及到代码生成，所以我们还需要导入页面模板引擎，这里我们用的是freemarker。

```

1 <!--整合mybatis plus https://baomidou.com/-->
2 <dependency>
3     <groupId>com.baomidou</groupId>
4     <artifactId>mybatis-plus-boot-starter</artifactId>
5     <version>3.4.1</version>
6 </dependency>
7 <!--mp代码生成器-->
8 <dependency>

```

```

9      <groupId>com.baomidou</groupId>
10     <artifactId>mybatis-plus-generator</artifactId>
11     <version>3.4.1</version>
12 </dependency>
13 <dependency>
14     <groupId>org.freemarker</groupId>
15     <artifactId>freemarker</artifactId>
16     <version>2.3.30</version>
17 </dependency>
18 <dependency>
19     <groupId>mysql</groupId>
20     <artifactId>mysql-connector-java</artifactId>
21     <scope>runtime</scope>
22 </dependency>

```

## 第二步：然后去写配置文件

```

1  server:
2    port: 8081
3  # DataSource Config
4  spring:
5    datasource:
6      driver-class-name: com.mysql.cj.jdbc.Driver
7      url: jdbc:mysql://localhost:3306/vueadmin?useUnicode=true&useSSL=false&
8      username: root
9      password: admin
10  mybatis-plus:
11    mapper-locations: classpath*:/mapper/**/*.xml

```

上面除了配置数据库的信息，还配置了mybatis plus的mapper的xml文件的扫描路径，这一步不要忘记了。然后因为前段默认是8080端口了，所以后端我们设置为8081端口，防止端口冲突。

## 第三步：开启mapper接口扫描，添加分页、防全表更新插件

新建一个包：通过@MapperScan注解指定要变成实现类的接口所在的包，然后包下面的所有接口在编译之后都会生成相应的实现类。

- com.markerhub.config.MybatisPlusConfig

```

1  @Configuration
2  @MapperScan("com.markerhub.mapper")
3  public class MybatisPlusConfig {
4      /**
5       * 新的分页插件,一缓和二缓遵循mybatis的规则,
6       * 需要设置 MybatisConfiguration#useDeprecatedExecutor = false
7       * 避免缓存出现问题(该属性会在旧插件移除后一同移除)
8       */
9      @Bean
10     public MybatisPlusInterceptor mybatisPlusInterceptor() {
11         MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
12         interceptor.addInnerInterceptor(new PaginationInnerInterceptor(DbType.
13         // 防止全表更新和删除

```

```

14     interceptor.addInnerInterceptor(new BlockAttackInnerInterceptor());
15     return interceptor;
16 }
17 @Bean
18 public ConfigurationCustomizer configurationCustomizer() {
19     return configuration -> configuration.setUseDeprecatedExecutor(false);
20 }
21 }

```

上面代码中，我们给Mybatis plus添加了2个拦截器，这是根据mp官网配置的：

- PaginationInnerInterceptor：新的分页插件
- BlockAttackInnerInterceptor：防止全表更新和删除

#### 第四步：创建数据库和表

因为是后台管理系统的权限模块，所以我们需要考虑的表主要就几个：用户表、角色表、菜单权限表、以及关联的用户角色中间表、菜单角色中间表。就5个表，至于什么字段其实都听随意的，用户表里面除了用户名、密码字段必要，其他其实都听随意，然后角色和菜单我们可以参考一下其他的系统、或者自己在做项目的过程中需要的时候在添加也行，反正重新生成代码也是非常简便的事情，综合考虑，数据库名称为vueadmin，我们建表语句如下：

- vueadmin.sql
- [vueadmin-java/数据库脚本 – vueadmin.sql · MarkerHub/VueAdmin – Gitee.com](#)
- 用户名/密码 admin/111111 test/1234567

```

1  /*
2  Navicat MySQL Data Transfer
3
4  Source Server          : localhost
5  Source Server Version  : 50717
6  Source Host            : localhost:3306
7  Source Database        : vueadmin
8
9  Target Server Type     : MYSQL
10 Target Server Version  : 50717
11 File Encoding          : 65001
12
13 Date: 2021-01-23 09:41:50
14 */
15
16 SET FOREIGN_KEY_CHECKS=0;
17
18 -- -----
19 -- Table structure for sys_menu
20 -- -----
21 DROP TABLE IF EXISTS `sys_menu`;
22 CREATE TABLE `sys_menu` (
23   `id` bigint(20) NOT NULL AUTO_INCREMENT,
24   `parent_id` bigint(20) DEFAULT NULL COMMENT '父菜单ID，一级菜单为0'，
25   `name` varchar(64) NOT NULL,
26   `path` varchar(255) DEFAULT NULL COMMENT '菜单URL'，
27   `perms` varchar(255) DEFAULT NULL COMMENT '授权(多个用逗号分隔，如：user:list

```

```

28     `component` varchar(255) DEFAULT NULL,
29     `type` int(5) NOT NULL COMMENT '类型      0:目录    1:菜单    2:按钮',
30     `icon` varchar(32) DEFAULT NULL COMMENT '菜单图标',
31     `orderNum` int(11) DEFAULT NULL COMMENT '排序',
32     `created` datetime NOT NULL,
33     `updated` datetime DEFAULT NULL,
34     `statu` int(5) NOT NULL,
35     PRIMARY KEY (`id`),
36     UNIQUE KEY `name` (`name`) USING BTREE
37 ) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8;
38
39 -- -----
40 -- Table structure for sys_role
41 -- -----
42 DROP TABLE IF EXISTS `sys_role`;
43 CREATE TABLE `sys_role` (
44     `id` bigint(20) NOT NULL AUTO_INCREMENT,
45     `name` varchar(64) NOT NULL,
46     `code` varchar(64) NOT NULL,
47     `remark` varchar(64) DEFAULT NULL COMMENT '备注',
48     `created` datetime DEFAULT NULL,
49     `updated` datetime DEFAULT NULL,
50     `statu` int(5) NOT NULL,
51     PRIMARY KEY (`id`),
52     UNIQUE KEY `name` (`name`) USING BTREE,
53     UNIQUE KEY `code` (`code`) USING BTREE
54 ) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
55
56 -- -----
57 -- Table structure for sys_role_menu
58 -- -----
59 DROP TABLE IF EXISTS `sys_role_menu`;
60 CREATE TABLE `sys_role_menu` (
61     `id` bigint(20) NOT NULL AUTO_INCREMENT,
62     `role_id` bigint(20) NOT NULL,
63     `menu_id` bigint(20) NOT NULL,
64     PRIMARY KEY (`id`)
65 ) ENGINE=InnoDB AUTO_INCREMENT=102 DEFAULT CHARSET=utf8mb4;
66
67 -- -----
68 -- Table structure for sys_user
69 -- -----
70 DROP TABLE IF EXISTS `sys_user`;
71 CREATE TABLE `sys_user` (
72     `id` bigint(20) NOT NULL AUTO_INCREMENT,
73     `username` varchar(64) DEFAULT NULL,
74     `password` varchar(64) DEFAULT NULL,
75     `avatar` varchar(255) DEFAULT NULL,
76     `email` varchar(64) DEFAULT NULL,
77     `city` varchar(64) DEFAULT NULL,
78     `created` datetime DEFAULT NULL,

```

```

79     `updated` datetime DEFAULT NULL,
80     `last_login` datetime DEFAULT NULL,
81     `statu` int(5) NOT NULL,
82     PRIMARY KEY (`id`),
83     UNIQUE KEY `UK_USERNAME` (`username`) USING BTREE
84 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;
85
86 -- -----
87 -- Table structure for sys_user_role
88 -- -----
89 DROP TABLE IF EXISTS `sys_user_role`;
90 CREATE TABLE `sys_user_role` (
91     `id` bigint(20) NOT NULL AUTO_INCREMENT,
92     `user_id` bigint(20) NOT NULL,
93     `role_id` bigint(20) NOT NULL,
94     PRIMARY KEY (`id`)
95 ) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8mb4;
96

```

## 第五步：代码生成

1. 获取项目数据库所对应表和字段的信息
2. 新建一个freemarker的页面模板 – SysUser.java.ftl – \${baseEntity}
3. 提供相关需要进行渲染的动态数据 – BaseEntity、表字段、注释、baseEntity=SuperEntity
4. 使用freemarker模板引擎进行渲染! – SysUser.java

```

1  # 获取表
2  SELECT
3      *
4  FROM
5      information_schema. TABLES
6  WHERE
7      TABLE_SCHEMA = (SELECT DATABASE());
8
9  # 获取字段
10 SELECT
11     *
12 FROM
13     information_schema. COLUMNS
14 WHERE
15     TABLE_SCHEMA = (SELECT DATABASE())
16 AND TABLE_NAME = "sys_user";

```

有了数据库之后，那么现在就已经可以使用mybatis plus了，官方给我们提供了一个代码生成器，然后我写上自己的参数之后，就可以直接根据数据库表信息生成entity、service、mapper等接口和实现类。

因为代码比较长，就不贴出来了，说明一下重点：

- com.markerhub.CodeGenerator

```
// 策略配置
StrategyConfig strategy = new StrategyConfig();
strategy.setNaming(NamingStrategy.underline_to_camel);
strategy.setColumnNaming(NamingStrategy.underline_to_camel);
strategy.setSuperEntityClass("BaseEntity");
strategy.setEntityLombokModel(true);
strategy.setRestControllerStyle(true);
// 公共父类
strategy.setSuperControllerClass("BaseController");
// 写于父类中的公共字段
strategy.setSuperEntityColumns("id", "created", "updated", "statu");
strategy.setInclude(scanner(tip: "表名, 多个英文逗号分割").split(regex: ","));
strategy.setControllerMappingHyphenStyle(true);
strategy.setTablePrefix("sys_");//动态调整
mpg.setStrategy(strategy);
mpg.setTemplateEngine(new FreemarkerTemplateEngine());
mpg.execute();
```

微信搜一搜  MarkerHub

上面代码生成的过程中，我默认所有的实体类都继承BaseEntity，控制器都继承BaseController，所以在代码生成之前，最好先编写这两个基类：

- com.markerhub.entity.BaseEntity

```
1 @Data
2 public class BaseEntity implements Serializable {
3     @TableId(value = "id", type = IdType.AUTO)
4     private Long id;
5     private LocalDateTime created;
6     private LocalDateTime updated;
7     private Integer statu;
8 }
```

- com.markerhub.controller.BaseController

```
1 public class BaseController {
2     @Autowired
3     HttpServletRequest req;
4 }
```

然后我们单独运行CodeGenerator的main方法，注意调整CodeGenerator的数据库连接、账号密码啥的，然后我们输入表名称，通过逗号隔开：

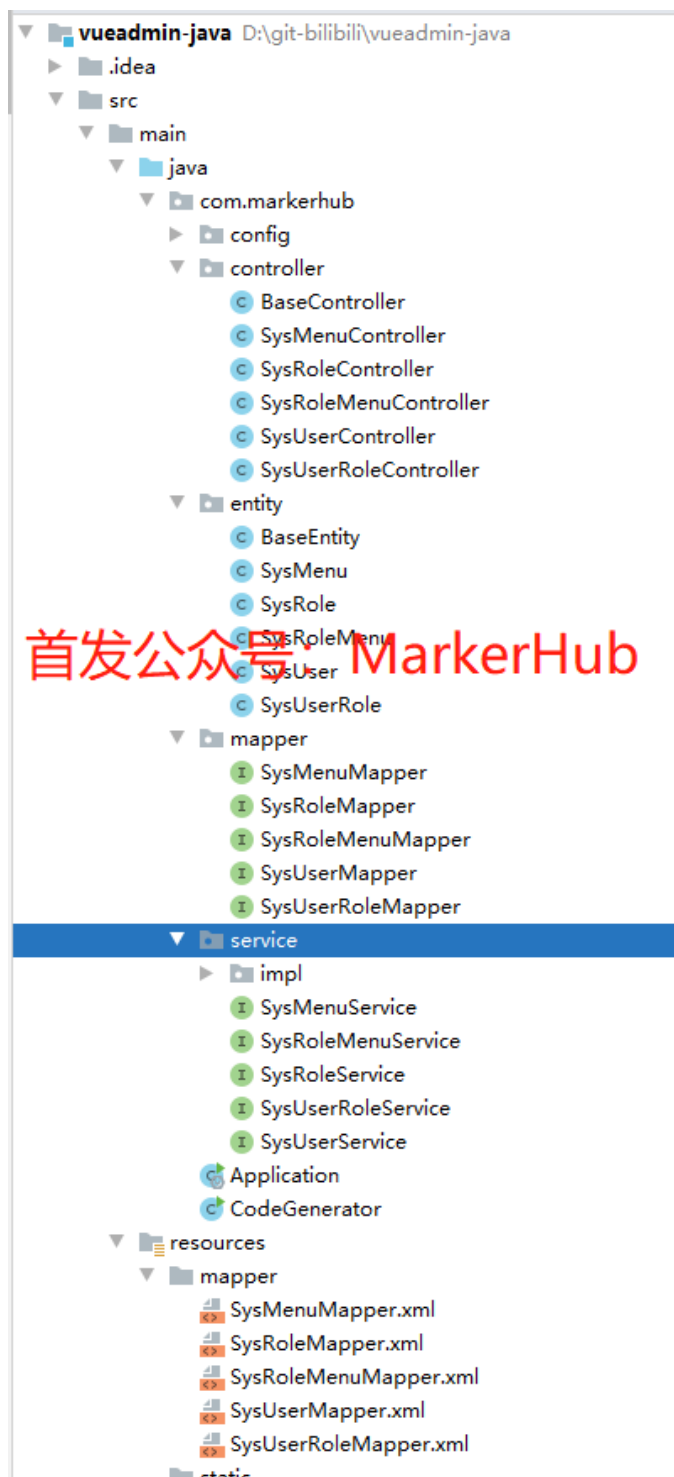
sys\_menu,sys\_role,sys\_role\_menu,sys\_user,sys\_user\_role

执行结果成功：

```
"C:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
请输入表名, 多个英文逗号分割:
sys_menu,sys_role,sys_role_menu,sys_user,sys_user_role
09:46:22.117 [main] DEBUG com.baomidou.mybatisplus.generator
```

然后我们生成了一些代码如下：





首发公众号: MarkerHub

这里有点需要注意，因为关联的用户角色中间表、菜单角色中间表我们是没有created等几个公共字段的，所以我把这两个实体继承BaseEntity去掉：

```
@Data
@EqualsAndHashCode(callSuper = true)
public class SysRoleMenu extends BaseEntity {

    private static final long serialVersionUID = 1L;

    private Long roleId;

    private Long menuId;

}
```

去掉

最后这样的：

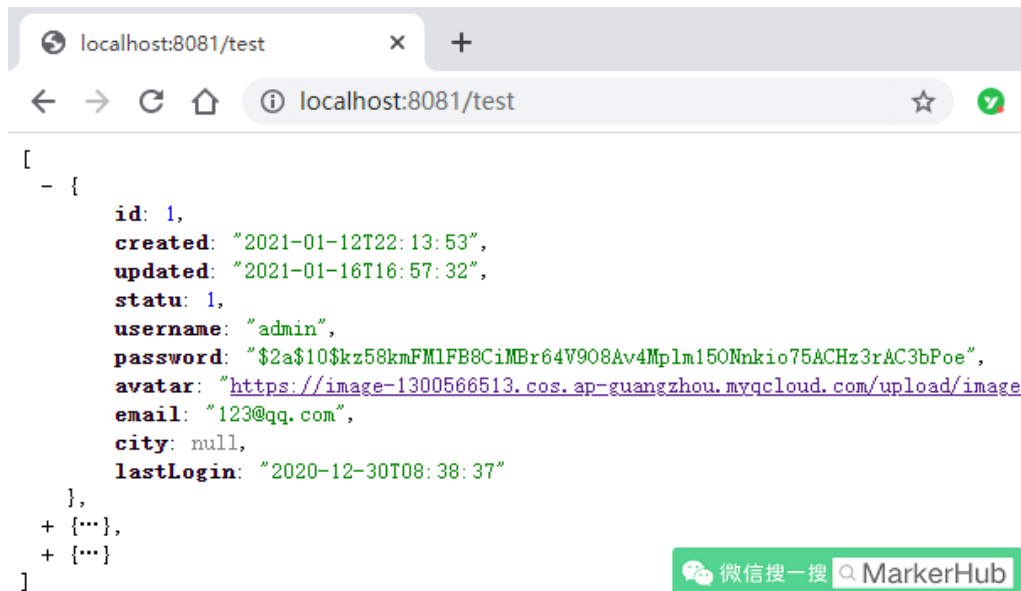
```
1 @Data
2 public class SysRoleMenu {
3     ...
4 }
```

简洁！方便！经过上面的步骤，基本上我们已经把mybatis plus框架集成到项目中了，并且也生成了基本的代码，省了好多功夫。然后我们做个简单测试：

- com.markerhub.controller.TestController


```
1 @RestController
2 public class TestController {
3     @Autowired
4     SysUserService userService;
5     @GetMapping("/test")
6     public Object test() {
7         return userService.list();
8     }
9 }
```

然后sys\_user随意添加几条数据，结果如下：



```
[
  - {
    id: 1,
    created: "2021-01-12T22:13:53",
    updated: "2021-01-16T16:57:32",
    statu: 1,
    username: "admin",
    password: "$2a$10$Kz58kmFM1FB8CiMBR64V908Av4Mplm150Nnkio75ACHz3rAC3bPoe",
    avatar: "https://image-1300566513.cos.ap-guangzhou.myqcloud.com/upload/image",
    email: "123@qq.com",
    city: null,
    lastLogin: "2020-12-30T08:38:37"
  },
  + {...},
  + {...}
]
```

ok，毛什么问题，大家不用在意密码是怎么生成的，后面我们会说到，你现在随意填写就好了。对了，好多人问我的浏览器的json数据怎么显示这么好看，这是因为我用了JSONView这个插件：

 JSONView



#### 4. 结果封装

因为是前后端分离的项目，所以我们有必要统一一个结果返回封装类，这样前后端交互的时候有个统一的标准，约定结果返回的数据是正常的或者遇到异常了。

这里我们用到了一个Result的类，这个用于我们的异步统一返回的结果封装。一般来说，结果里面有几个要素必要的

- 是否成功，可用code表示（如200表示成功，400表示异常）
- 结果消息

- 结果数据

所以可得到封装如下：

- com.markerhub.common.lang.Result

```
1 @Data
2 public class Result implements Serializable {
3
4     private int code; // 200是正常，非200表示异常
5     private String msg;
6     private Object data;
7
8     public static Result succ(Object data) {
9         return succ(200, "操作成功", data);
10    }
11    public static Result succ(int code, String msg, Object data) {
12        Result r = new Result();
13        r.setCode(code);
14        r.setMsg(msg);
15        r.setData(data);
16        return r;
17    }
18    public static Result fail(String msg) {
19        return fail(400, msg, null);
20    }
21    public static Result fail(String msg, Object data) {
22        return fail(400, msg, data);
23    }
24    public static Result fail(int code, String msg, Object data) {
25        Result r = new Result();
26        r.setCode(code);
27        r.setMsg(msg);
28        r.setData(data);
29        return r;
30    }
31 }
```

另外出了在结果封装类上的code可以提现数据是否正常，我们还可以通过http的状态码来提现访问是否遇到了异常，比如401表示无权限拒绝访问等，注意灵活使用。

## 5. 全局异常处理

有时候不可避免服务器报错的情况，如果不配置异常处理机制，就会默认返回tomcat或者nginx的5XX页面，对普通用户来说，不太友好，用户也不懂什么情况。这时候需要我们程序员设计返回一个友好简单的格式给前端。

处理办法如下：通过使用@ControllerAdvice来进行统一异常处理，@ExceptionHandler(value = RuntimeException.class)来指定捕获的Exception各个类型异常，这个异常的处理，是全局的，所有类似的异常，都会跑到这个地方处理。

步骤二、定义全局异常处理，@ControllerAdvice表示定义全局控制器异常处理，@ExceptionHandler表示针对性异常处理，可对每种异常针对性处理。

- com.markerhub.common.exception.GlobalExceptionHandler

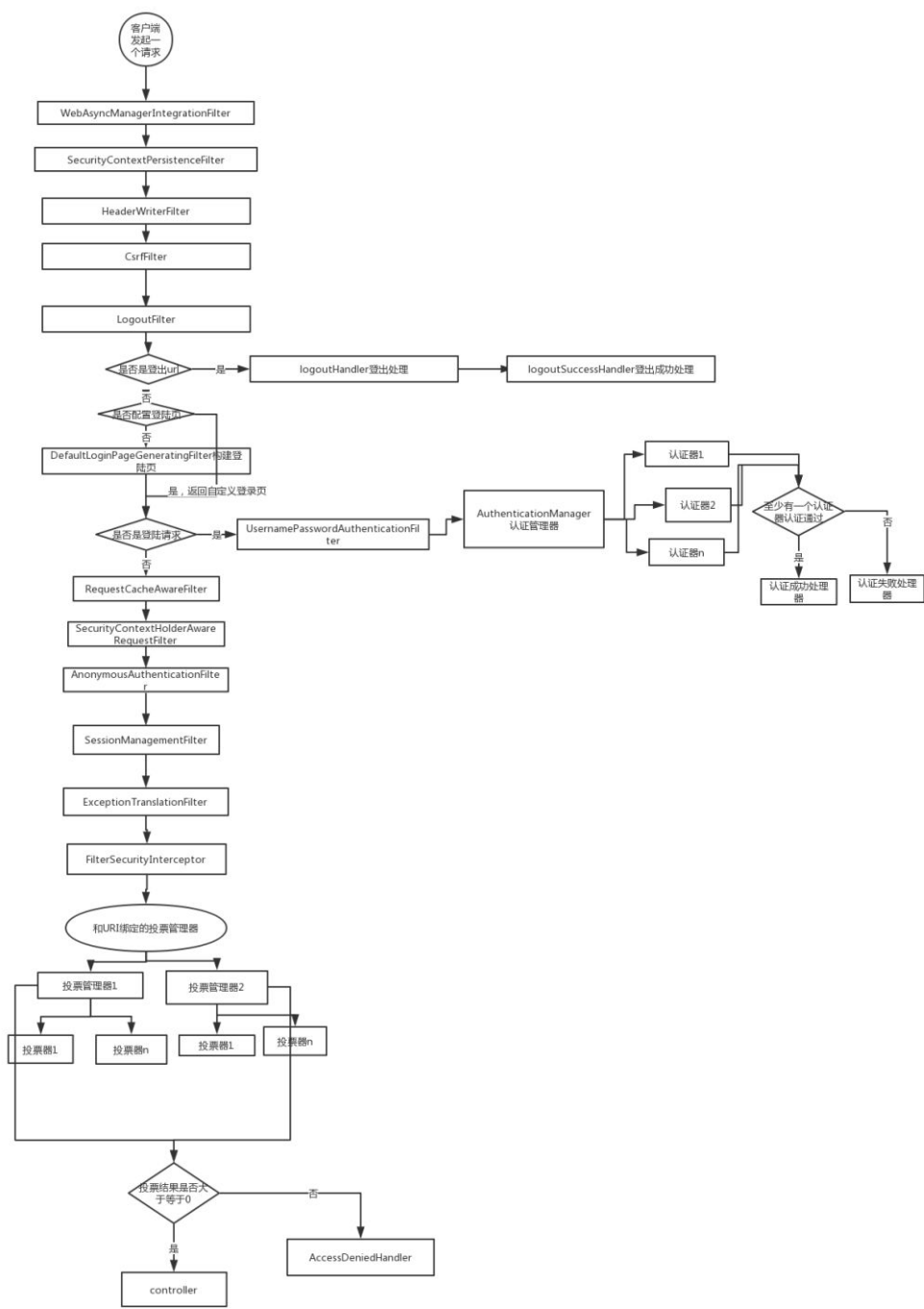
```
1  /**
2   * 全局异常处理
3   */
4  @Slf4j
5  @RestControllerAdvice
6  public class GlobalExceptionHandler {
7
8      @ResponseStatus(HttpStatus.FORBIDDEN)
9      @ExceptionHandler(value = AccessDeniedException.class)
10     public Result handler(AccessDeniedException e) {
11         log.info("security权限不足 : -----{}", e.getMessage());
12         return Result.fail("权限不足");
13     }
14
15     @ResponseStatus(HttpStatus.BAD_REQUEST)
16     @ExceptionHandler(value = MethodArgumentNotValidException.class)
17     public Result handler(MethodArgumentNotValidException e) {
18         log.info("实体校验异常 : -----{}", e.getMessage());
19         BindingResult bindingResult = e.getBindingResult();
20         ObjectError objectError = bindingResult.getAllErrors().stream().find
21             .return Result.fail(objectError.getDefaultMessage());
22     }
23
24     @ResponseStatus(HttpStatus.BAD_REQUEST)
25     @ExceptionHandler(value = IllegalArgumentException.class)
26     public Result handler(IllegalArgumentException e) {
27         log.error("Assert异常 : -----{}", e.getMessage());
28         return Result.fail(e.getMessage());
29     }
30
31     @ResponseStatus(HttpStatus.BAD_REQUEST)
32     @ExceptionHandler(value = RuntimeException.class)
33     public Result handler(RuntimeException e) {
34         log.error("运行时异常 : -----{}", e);
35         return Result.fail(e.getMessage());
36     }
37 }
```

上面我们捕捉了几个异常：

- ShiroException：shiro抛出的异常，比如没有权限，用户登录异常
- IllegalArgumentException：处理Assert的异常
- MethodArgumentNotValidException：处理实体校验的异常
- RuntimeException：捕捉其他异常

## 6. 整合Spring Security

很多人不懂spring security，觉得这个框架比shiro要难，的确，security更加复杂一点，同时功能也更加强大，我们首先来看一下security的原理，这里我们引用一张来自江南一点雨大佬画的一张原理图（<https://blog.csdn.net/u012702547/article/details/89629415>）：

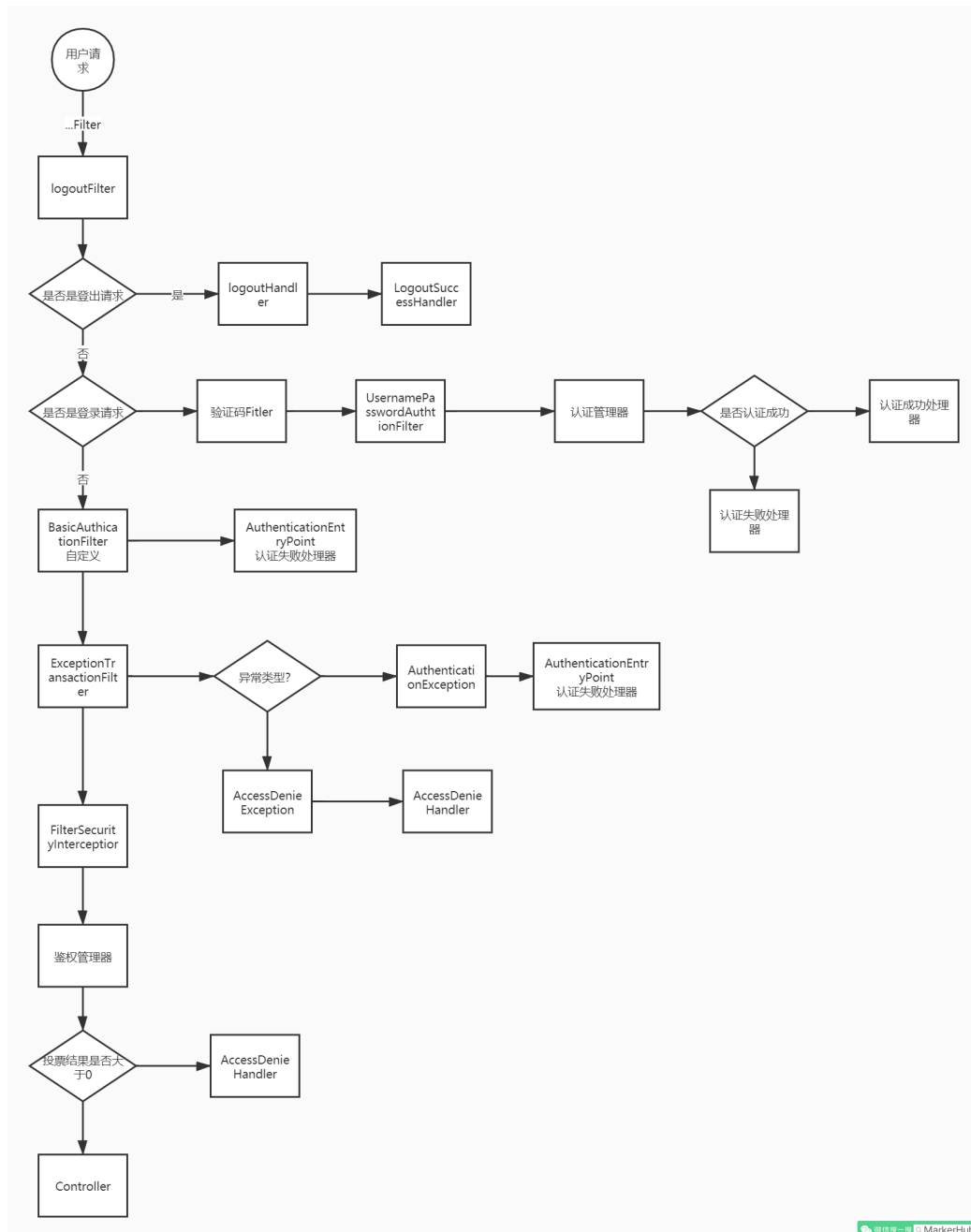


微信搜一搜 MarkerHub

(引自江南一点雨的博客)

上面这张图一定要好好看，特别清晰，毕竟security是责任链的设计模式，是一堆过滤器链的组合，如果对于这个流程都不清楚，那么你就谈不上理解security。那么针对我们现在的这个系统，我们可以自己设计一个security的认证方案，结合江南一点雨大佬的博客，我们得到这样一套流程：

<https://www.proceesson.com/view/link/606b0b5307912932d09adcb3>



#### 流程说明：

1. 客户端发起一个请求，进入 Security 过滤器链。
2. 当到 LogoutFilter 的时候判断是否是登出路径，如果是登出路径则到 logoutHandler，如果登出成功则到 logoutSuccessHandler 登出成功处理。如果不是登出路径则直接进入下一个过滤器。
3. 当到 UsernamePasswordAuthenticationFilter 的时候判断是否为登录路径，如果是，则进入该过滤器进行登录操作，如果登录失败则到 AuthenticationFailureHandler，登录失败处理器处理，如果登录成功则到 AuthenticationSuccessHandler 登录成功处理器处理，如果不是登录请求则不进入该过滤器。
4. 进入认证BasicAuthenticationFilter进行用户认证，成功的话会把认证了的结果写入到 SecurityContextHolder中SecurityContext的属性authentication上面。如果认证失败就会交给AuthenticationEntryPoint认证失败处理类，或者抛出异常被后续

ExceptionHandler过滤器处理异常，如果是AuthenticationException就交给AuthenticationEntryPoint处理，如果是AccessDeniedException异常则交给AccessDeniedHandler处理。

5. 当到 FilterSecurityInterceptor 的时候会拿到 uri ， 根据 uri 去找对应的鉴权管理器，鉴权管理器做鉴权工作，鉴权成功则到 Controller 层，否则到 AccessDeniedHandler 鉴权失败处理器处理。

Spring Security 实战干货：必须掌握的一些内置 Filter：[https://blog.csdn.net/qq\\_35067322/article/details/102690579](https://blog.csdn.net/qq_35067322/article/details/102690579)

ok，上面我们说的流程中涉及到几个组件，有些是我们需要根据实际情况来重写的。因为我们使用json数据进行前后端数据交互，并且我们返回结果也是特定封装的。我们先再总结一下我们需要了解的几个组件：

- LogoutFilter – 登出过滤器
- LogoutSuccessHandler – 登出成功之后的操作类
- UsernamePasswordAuthenticationFilter – form提交用户名密码登录认证过滤器
- AuthenticationFailureHandler – 登录失败操作类
- AuthenticationSuccessHandler – 登录成功操作类
- BasicAuthenticationFilter – Basic身份认证过滤器
- SecurityContextHolder – 安全上下文静态工具类
- AuthenticationEntryPoint – 认证失败入口
- ExceptionTranslationFilter – 异常处理过滤器
- AccessDeniedHandler – 权限不足操作类
- FilterSecurityInterceptor – 权限判断拦截器、出口

有了上面的组件，那么认证与授权两个问题我们就已经接近啦，我们现在需要做的就是去重写我们的一些关键类。

## 引入Security与jwt

首先我们导入security包，因为我们前后端交互用户凭证用的是JWT，所以我们也导入jwt的相关包，然后因为验证码的存储需要用到redis，所以引入redis。最后为了一些工具类，我们引入hutool。

- pom.xml

```
1 <!-- springboot security -->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-security</artifactId>
5 </dependency>
6 <dependency>
7     <groupId>org.springframework.boot</groupId>
8     <artifactId>spring-boot-starter-data-redis</artifactId>
9 </dependency>
10 <!-- jwt -->
11 <dependency>
12     <groupId>io.jsonwebtoken</groupId>
```

```

13     <artifactId>jjwt</artifactId>
14     <version>0.9.1</version>
15 </dependency>
16 <dependency>
17     <groupId>com.github.axet</groupId>
18     <artifactId>kaptcha</artifactId>
19     <version>0.0.9</version>
20 </dependency>
21 <!-- hutool工具类-->
22 <dependency>
23     <groupId>cn.hutool</groupId>
24     <artifactId>hutool-all</artifactId>
25     <version>5.3.3</version>
26 </dependency>
27 <dependency>
28     <groupId>org.apache.commons</groupId>
29     <artifactId>commons-lang3</artifactId>
30     <version>3.11</version>
31 </dependency>

```

启动redis，然后我们再启动项目，这时候我们再去访问<http://localhost:8081/test>，会发现系统会先判断到你未登录跳转到<http://localhost:8081/login>，因为security内置了登录页，用户名为user，密码在启动项目的时候打印在了控制台。登录完成之后我们可以正常访问接口。

因为每次启动密码都会改变，所以我们通过配置文件来配置一下默认的用户名和密码：

- application.yml

```

1 spring:
2   security:
3     user:
4       name: user
5       password: 111111

```

## 用户认证

首先我们来解决用户认证问题，分为首次登陆，和二次认证。

- 首次登录认证：用户名、密码和验证码完成登录
- 二次token认证：请求头携带Jwt进行身份认证

使用用户名密码来登录的，然后我们还想添加图片验证码，那么security给我们提供的UsernamePasswordAuthenticationFilter能使用吗？

首先security的所有过滤器都是没有图片验证码这回事的，看起来不适用了。其实这里我们可以灵活点，如果你依然想沿用自带的UsernamePasswordAuthenticationFilter，那么我们就在这过滤器之前添加一个图片验证码过滤器。当然了我们也可以通过自定义过滤器继承UsernamePasswordAuthenticationFilter，然后自己把验证码验证逻辑和认证逻辑写在一起，这也是一种解决方式。

我们这次解决方式是在UsernamePasswordAuthenticationFilter之前自定义一个图片过滤器CaptchaFilter，提前校验验证码是否正确，这样我们就可以使用



UsernamePasswordAuthenticationFilter了，然后登录正常或失败我们都可以通过对应的Handler来返回我们特定格式的封装结果数据。

## 生成验证码

首先我们先生成验证码，之前我们已经引用了google的验证码生成器，我们先来配置一下图片验证码的生成规则：

- com.markerhub.config.KaptchaConfig

```
1 @Configuration
2 public class KaptchaConfig {
3     @Bean
4     public DefaultKaptcha producer() {
5         Properties properties = new Properties();
6         properties.put("kaptcha.border", "no");
7         properties.put("kaptcha.textproducer.font.color", "black");
8         properties.put("kaptcha.textproducer.char.space", "4");
9         properties.put("kaptcha.image.height", "40");
10        properties.put("kaptcha.image.width", "120");
11        properties.put("kaptcha.textproducer.font.size", "30");
12        Config config = new Config(properties);
13        DefaultKaptcha defaultKaptcha = new DefaultKaptcha();
14        defaultKaptcha.setConfig(config);
15        return defaultKaptcha;
16    }
17 }
```

上面我定义了图片验证码的长宽字体颜色等，自己可以调整哈。

然后通过控制器提供生成验证码的方法：

- com.markerhub.controller.AuthController

```
1 @Slf4j
2 @RestController
3 public class AuthController extends BaseController{
4     @Autowired
5     private Producer producer;
6     /**
7      * 图片验证码
8      */
9     @GetMapping("/captcha")
10    public Result captcha(HttpServletRequest request, HttpServletResponse res
11        String code = producer.createText();
12        String key = UUID.randomUUID().toString();
13
14        BufferedImage image = producer.createImage(code);
15        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
16        ImageIO.write(image, "jpg", outputStream);
17        BASE64Encoder encoder = new BASE64Encoder();
18        String str = "data:image/jpeg;base64,";
19        String base64Img = str + encoder.encode(outputStream.toByteArray());
20    }
```

```

21     // 存储到redis中
22     redisUtil.hset(Const.captcha_KEY, key, code, 120);
23     log.info("验证码 -- {} - {}", key, code);
24     return Result.succ(
25         MapUtil.builder()
26             .put("token", key)
27             .put("base64Img", base64Img)
28             .build()
29     );
30 }
31 }

```

因为前后端分离，我们禁用了session，所以我们将验证码放在了redis中，使用一个随机字符串作为key，并传送到前端，前端再把随机字符串和用户输入的验证码提交上来，这样我们就可以通过随机字符串获取到保存的验证码和用户的验证码进行比较了是否正确了。

然后因为图片验证码的方式，所以我们进行了encode，把图片进行了base64编码，这样前端就可以显示图片了。

而前端的处理，我们之前是使用了mockjs进行随机生成数据的，现在后端有接口之后，我们只需要在main.js中去掉mockjs的引入即可，这样前端就可以访问后端的接口而不被mock拦截了。

## 验证码认证过滤器

图片验证码进行认证验证码是否正确。

- CaptchaFilter

```

1  /**
2   * 图片验证码校验过滤器，在登录过滤器前
3   */
4  @Slf4j
5  @Component
6  public class CaptchaFilter extends OncePerRequestFilter {
7      private final String loginUrl = "/login";
8      @Autowired
9      RedisUtil redisUtil;
10     @Autowired
11     LoginFailureHandler loginFailureHandler;
12     @Override
13     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response)
14         throws ServletException, IOException {
15         String url = request.getRequestURI();
16         if (loginUrl.equals(url) && request.getMethod().equals("POST")) {
17             log.info("获取到login链接，正在校验验证码 -- " + url);
18             try {
19                 validate(request);
20             } catch (CaptchaException e) {
21                 log.info(e.getMessage());
22                 // 交给登录失败处理器处理
23                 loginFailureHandler.onAuthenticationFailure(request, response, e);
24             }

```

```

25     }
26     filterChain.doFilter(request, response);
27 }
28 private void validate(HttpServletRequest request) {
29     String code = request.getParameter("code");
30     String token = request.getParameter("token");
31     if (StringUtils.isBlank(code) || StringUtils.isBlank(token)) {
32         throw new CaptchaException("验证码不能为空");
33     }
34     if(!code.equals(redisUtil.hget(Const.captcha_KEY, token))) {
35         throw new CaptchaException("验证码不正确");
36     }
37     // 一次性使用
38     redisUtil.hdel(Const.captcha_KEY, token);
39 }
40 }

```

上面代码中，因为验证码需要存储，所以添加了RedisUtil工具类，这个工具类代码我们就不贴出来了。

- com.markerhub.util.RedisUtil

然后验证码出错的时候我们返回异常信息，这是一个认证异常，所以我们自定了一个CaptchaException：

- com.javacat.common.exception.CaptchaException

```

1 public class CaptchaException extends AuthenticationException {
2     public CaptchaException(String msg) {
3         super(msg);
4     }
5 }

```

- com.markerhub.common.lang.Const

```

1 public class Const {
2     public static final String captcha_KEY = "captcha";
3 }

```

然后认证失败的话，我们之前说过，登录失败的时候交给AuthenticationFailureHandler，所以我们自定义了LoginFailureHandler

- com.markerhub.security.LoginFailureHandler

```

1 @Component
2 public class LoginFailureHandler implements AuthenticationFailureHandler {
3     @Override
4     public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response) throws IOException {
5         response.setContentType("application/json;charset=UTF-8");
6         ServletOutputStream outputStream = response.getOutputStream();
7         Result result = Result.fail(
8             "Bad credentials".equals(exception.getMessage()) ? "用户名或密码不正确" : exception.getMessage()
9         );
10        outputStream.write(JSONUtil.toJsonStr(result).getBytes("UTF-8"));
11        outputStream.flush();
12        outputStream.close();
13    }

```

```
14 }
```

其实主要就是获取异常的消息，然后封装到Result，最后转成json返回给前端而已哈。

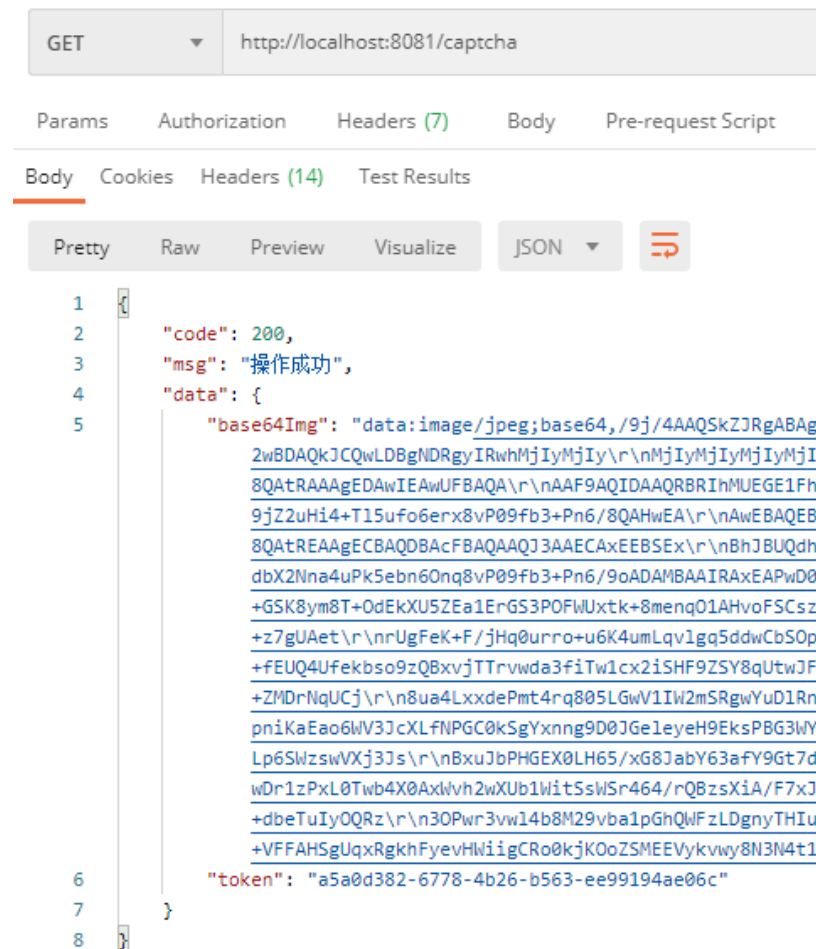
然后我们配置SecurityConfig

- com.markerhub.config.SecurityConfig

```
1  @Configuration
2  @EnableGlobalMethodSecurity(prePostEnabled = true)
3  @EnableWebSecurity
4  public class SecurityConfig extends WebSecurityConfigurerAdapter {
5
6      @Autowired
7      LoginFailureHandler loginFailureHandler;
8
9      @Autowired
10     CaptchaFilter captchaFilter;
11
12     public static final String[] URL_WHITELIST = {
13
14         "/webjars/**",
15         "/favicon.ico",
16
17         "/captcha",
18         "/login",
19         "/logout",
20     };
21
22     @Override
23     protected void configure(HttpSecurity http) throws Exception {
24         http.cors().and().csrf().disable()
25             .formLogin()
26             .failureHandler(loginFailureHandler)
27
28             .and()
29             .authorizeRequests()
30             .antMatchers(URL_WHITELIST).permitAll() //白名单
31             .anyRequest().authenticated()
32             // 不会创建 session
33             .and()
34             .sessionManagement()
35             .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
36
37             .and()
38             .addFilterBefore(captchaFilter, UsernamePasswordAuthenticationFilter)
39         ;
40     }
41 }
```

首先formLogin我们定义了表单登录提交的方式以及定义了登录失败的处理器，后面我们还要定义登录成功的处理器的。然后authorizeRequests我们除了白名单的链接之外其他请求都会被拦截。再然后就是禁用session，最后是设定验证码过滤器在登录过滤器之前。

然后我们打开前端的/login，发现出现了跨域的问题，后面我处理，我们先用postman调试接口。



可以看到，我们的随机码token和base64Img编码都是正常的。控制台上看到我们的验证是2yyxm：

```
/ : Secured GET /captcha
:roller : 验证码 -- a5a0d382-6778-4b26-b563-ee99194ae06c - f357x
:Filter : Cleared SecurityContextHolder to complete request
```

然后我们尝试登录，因为之前我们已经设置了用户名密码为user/111111，所以我们提交表单的时候再带上我们的token和验证码。

这时候我们就可以去提交表单了吗，其实还不可以，为啥？因为就算我们登录成功，security默认跳转到/链接，但是又会因为没有权限访问/，所有又会教你去登录，所以我们必须取消原先默认的登录成功之后的操作，根据我们之前分析的流程，登录成功之后会走AuthenticationSuccessHandler，因此在登录之前，我们先去自定义这个登录成功操作类：

- com.markerhub.security.LoginSuccessHandler

```
1 @Component
2 public class LoginSuccessHandler implements AuthenticationSuccessHandler {
3     @Autowired
4     JwtUtils jwtUtils;
5
6     @Override
7     public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response) throws IOException {
```

```

8         response.setContentType("application/json;charset=UTF-8");
9         ServletOutputStream outputStream = response.getOutputStream();
10
11         // 生成jwt返回
12         String jwt = jwtUtils.generateToken(authentication.getName());
13         response.setHeader(jwtUtils.getHeader(), jwt);
14
15         Result result = Result.succ("");
16         outputStream.write(JSONUtil.toJsonStr(result).getBytes("UTF-8"));
17         outputStream.flush();
18         outputStream.close();
19     }
20 }

```

登录成功之后我们利用用户名生成jwt，jwtUtils这个工具类我就不贴代码了哈，去看我们项目源码，然后把jwt作为请求头返回回去，名称就叫Authorization哈。我们需要在配置文件中配置一些jwt的一些密钥信息：

- application.yml

```

1 markerhub:
2   jwt:
3     # 加密密钥
4     secret: f4e2e52034348f86b67cde581c0f9eb5
5     # token有效时长，7天，单位秒
6     expire: 604800
7     header: Authorization

```

然后我们再security配置中加上登录成功之后的操作类：

- com.markerhub.config.SecurityConfig

```

1 @Autowired
2 LoginSuccessHandler loginSuccessHandler;
3
4 ...
5 # configure代码：
6
7 http.cors().and().csrf().disable()
8     .formLogin()
9     .failureHandler(loginFailureHandler)
10    .successHandler(loginSuccessHandler)

```

然后我们去postman的进行我们的登录测试：

► http://localhost:8080/login?username=admin&password=111111&code=aaaaa&token=b

POST http://localhost:8081/login?username=user&password=111111&code=f357x&token=a5a0d382-6778-4b26-b563-ee99194ae06c

Params Authorization Headers (8) Body Pre-request Script Tests Se

Query Params

KEY	VALUE
username	user
password	111111
code	f357x
token	a5a0d382-6778-4b26-b563-ee99194ae06c

Body Cookies Headers (15) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "操作成功",
3   "code": 200,
4   "data": ""
5 }
```

上面我们可以看到，我们已经可以登录成功了。然后去结果的请求头中查看jwt：

Body Cookies Headers (15) Test Results

Vary

Authorization

X-Content-Type-Options

X-XSS-Protection

Access-Control-Request-Headers

200 OK 844 ms 697 B Save Response

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VylwliaWF0Ij0xNjExNTgyNzU3LCJleHAiOiJlE2MTx0dC1NTd9.K7QzlpMxFF\_H3NG-q546cl\_qZj8VSvHFSsuv2onN-a2clXCmKQVXC3BeT3pXBbY-ghb7rB7Zcd-61KK1Hhza7Q

搞定，登录成功啦，验证码也正常验证了。

## 身份认证 – 1

登录成功之后前端就可以获取到了jwt的信息，前端中我们是保存在了store中，同时也保存在了localStorage中，然后每次axios请求之前，我们都会添加上我们的请求头信息，可以回顾一下：

- 前端项目的axios.js

vueadmin-vue src axios.js

```
13
14 request.interceptors.request.use( onFulfilled: config => {
15   config.headers['Authorization'] = localStorage.getItem( key: "token" )
16
17   return config
18 }
19
```

所以后端进行用户身份识别的时候，我们需要通过请求头中获取jwt，然后解析出我们的用户名，这样我们就可以知道是谁在访问我们的接口啦，然后判断用户是否有权限等操作。

那么我们自定义一个过滤器用来进行识别jwt。

- JWTAuthenticationFilter

```
1  @Slf4j
2  public class JWTAuthenticationFilter extends BasicAuthenticationFilter {
3      @Autowired
4      JwtUtils jwtUtils;
5      @Autowired
6      RedisUtil redisUtil;
7      @Autowired
8      SysUserService sysUserService;
9      public JWTAuthenticationFilter(AuthenticationManager authenticationManager) {
10         super(authenticationManager);
11     }
12     @Override
13     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
14         log.info("jwt 校验 filter");
15         String jwt = request.getHeader(jwtUtils.getHeader());
16         if (StrUtil.isBlankOrUndefined(jwt)) {
17             chain.doFilter(request, response);
18             return;
19         }
20         Claims claim = jwtUtils.getClaimByToken(jwt);
21         if (claim == null) {
22             throw new JwtException("token异常!");
23         }
24         if (jwtUtils.isTokenExpired(claim.getExpiration())) {
25             throw new JwtException("token已过期");
26         }
27         String username = claim.getSubject();
28         log.info("用户-{}, 正在登陆!", username);
29         UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken
30             = new UsernamePasswordAuthenticationToken(username, null, new TreeSet<GrantedAuthority>());
31         SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
32         chain.doFilter(request, response);
33     }
34 }
```

上面的逻辑也很简单，正如我前面说到的，获取到用户名之后我们直接把封装成 UsernamePasswordAuthenticationToken，之后交给SecurityContextHolder参数传递 authentication对象，这样后续security就能获取到当前登录的用户信息了，也就完成了用户认证。

当认证失败的时候会进入AuthenticationEntryPoint，于是我们自定义认证失败返回的数据：

- com.markerhub.security.JwtAuthenticationEntryPoint

```
1  /**
2   * 定义认证失败处理类
3   */
4  @Slf4j
5  @Component
6  public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {
7      @Override
```



```

8     public void commence(HttpServletRequest request, HttpServletResponse response)
9         throws IOException {
10         log.info("认证失败！未登录！");
11         response.setContentType("application/json;charset=UTF-8");
12         response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
13         ServletOutputStream outputStream = response.getOutputStream();
14
15         Result result = Result.fail("请先登录！");
16         outputStream.write(JSONUtil.toJsonStr(result).getBytes("UTF-8"));
17         outputStream.flush();
18         outputStream.close();
19     }
20 }

```

不过是啥原因，认证失败，我们就要求重新登录，所以返回的信息直接明了“请先登录！”哈哈。

然后我们把认证过滤器和认证失败入口配置到SecurityConfig中：

- com.markerhub.config.SecurityConfig

```

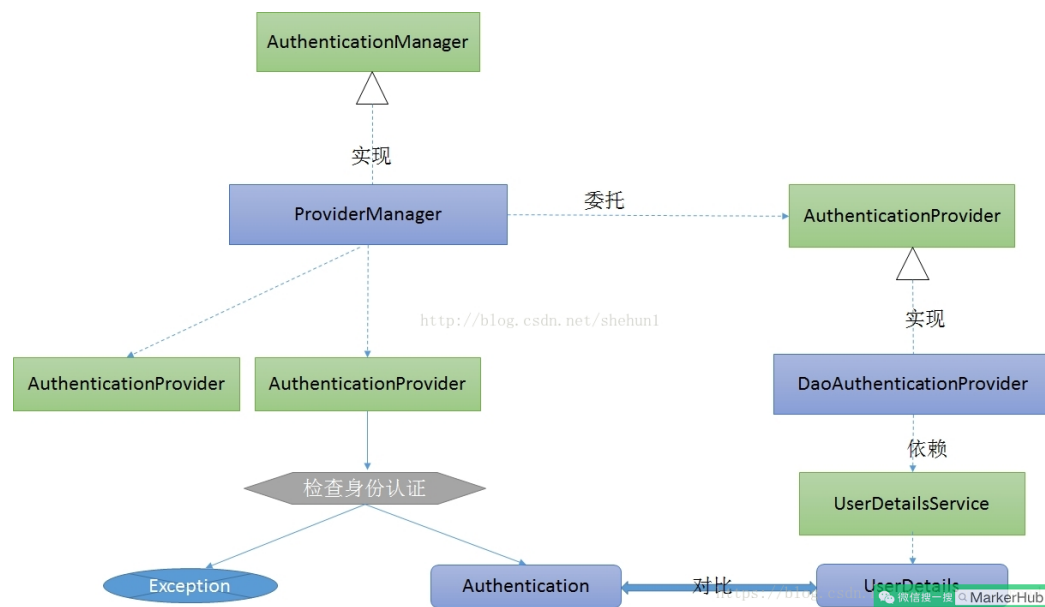
1 @Bean
2 JWTAuthenticationFilter jwtAuthenticationFilter() throws Exception {
3     JWTAuthenticationFilter filter = new JWTAuthenticationFilter(authenticationManager());
4     return filter;
5 }
6
7 .and()
8 .exceptionHandling()
9 .authenticationEntryPoint(jwtAuthenticationEntryPoint)
10
11 .and()
12 .addFilter(jwtAuthenticationFilter())
13 .addFilterBefore(captchaFilter, UsernamePasswordAuthenticationFilter.class)

```

这样携带jwt请求头我们就可以正常访问我们的接口了。

## 身份认证 – 2

之前我们的用户名密码配置在配置文件中的，而且密码也用的是明文，这明显不符合我们的要求，我们的用户必须是存储在数据库中，密码也是得经过加密的。所以我们先来解决这个问题，然后再去弄授权。



首先来插入一条用户数据，但这里有个问题，就是我们的密码怎么生成？密码怎么来的？这里我们使用Security内置了的BCryptPasswordEncoder，里面就有生成和匹配密码是否正确的方法，也就是加密和验证策略。因此我们再SecurityConfig中进行配置：

- com.markerhub.config.SecurityConfig

```

1 @Bean
2 BCryptPasswordEncoder bCryptPasswordEncoder() {
3     return new BCryptPasswordEncoder();
4 }
  
```

这样系统就会使用我们找一个新的密码策略进行匹配密码是否正常了。之前我们配置文件配置的用户名密码去掉：

- application.yml

```

1 # security:
2 #   user:
3 #     name: user
4 #     password: 111111
  
```

ok，我们先使用BCryptPasswordEncoder给我们生成一个密码，给数据库添加一条数据先，我们再TestController中注入BCryptPasswordEncoder，然后使用encode进行密码加密，对了，记得在SecurityConfig中吧/test/\*\*添加白名单哈，不然访问会提示你登录！！

- com.markerhub.controller.TestController

```

1 @Autowired
2 BCryptPasswordEncoder bCryptPasswordEncoder;
3
4 @GetMapping("/test/pass")
5 public Result passEncode() {
6     // 密码加密
7     String pass = bCryptPasswordEncoder.encode("111111");
8
9     // 密码验证
10    boolean matches = bCryptPasswordEncoder.matches("111111", pass);
11 }
  
```

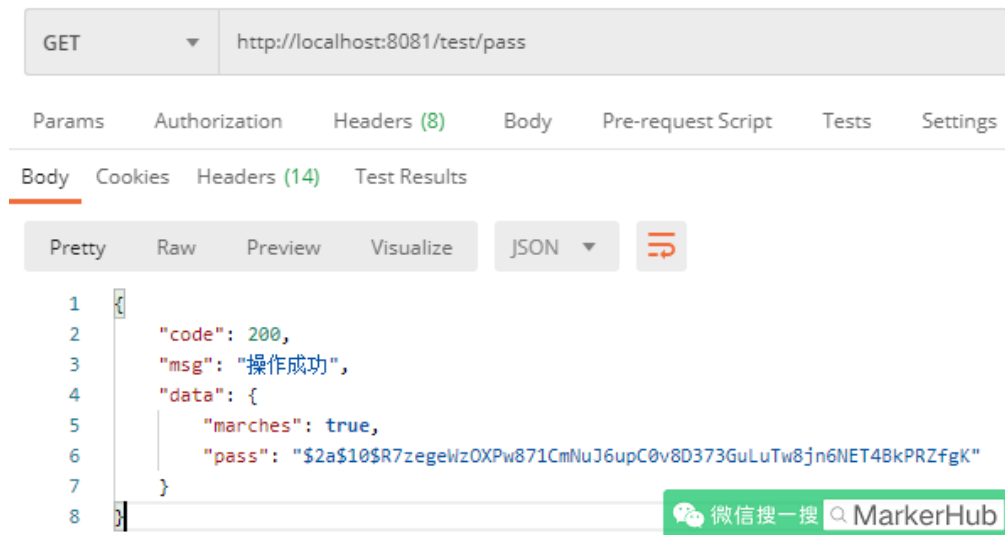
```

12     return Result.succ(MapUtil.builder()
13         .put("pass", pass)
14         .put("marches", matches)
15         .build()
16     );
17 }

```

可以看到我密码是111111，加密以及验证的结果如下：

[\\$2a\\$10\\$R7zegeWzOXPw871CmNuJ6upC0v8D373GuLuTw8jn6NET4BkPRZfgK](#)



data中的那一串字符串就是我们的密码了，可以看到marches也是true，说明密码验证也是正确的，我们添加到我们数据库sys\_user表中：

```

1 INSERT INTO `vueadmin`.`sys_user` (`id`, `username`, `password`, `avatar`, `

```

后面我们就可以使用admin/111111登录我们的系统哈。

但是先我们登录过程系统不是从我们数据库中获取数据的，因此，我们需要重新定义这个查用户数据的过程，我们需要重写UserDetailsService接口。

- com.markerhub.security.UserDetailsServiceImpl

```

1 @Slf4j
2 @Service
3 public class UserDetailsServiceImpl implements UserDetailsService {
4
5     @Autowired
6     SysUserService sysUserService;
7
8     @Override
9     public UserDetails loadUserByUsername(String username) throws UsernameNot
10         SysUser sysUser = sysUserService.getByUsername(username);
11         if (sysUser == null) {
12             throw new UsernameNotFoundException("用户名或密码不正确!");
13         }
14         return new AccountUser(sysUser.getId(), sysUser.getUsername(), sysUser
15     }

```

16 }

因为security在认证用户身份的时候会调用UserDetailsService.loadUserByUsername()方法, 因此我们重写了之后security就可以根据我们的流程去查库获取用户了。然后我们把UserDetailsServiceImpl配置到SecurityConfig中:

- com.markerhub.config.SecurityConfig

```
1 @Autowired
2 UserDetailsService userDetailsService;
3
4 @Override
5 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
6     auth.userDetailsService(userDetailsService);
7 }
```

然后上面UserDetailsService.loadUserByUsername()默认返回的UserDetails, 我们自定义了AccountUser去重写了UserDetails, 这也是为了后面我们可能会调整用户的一些数据等。

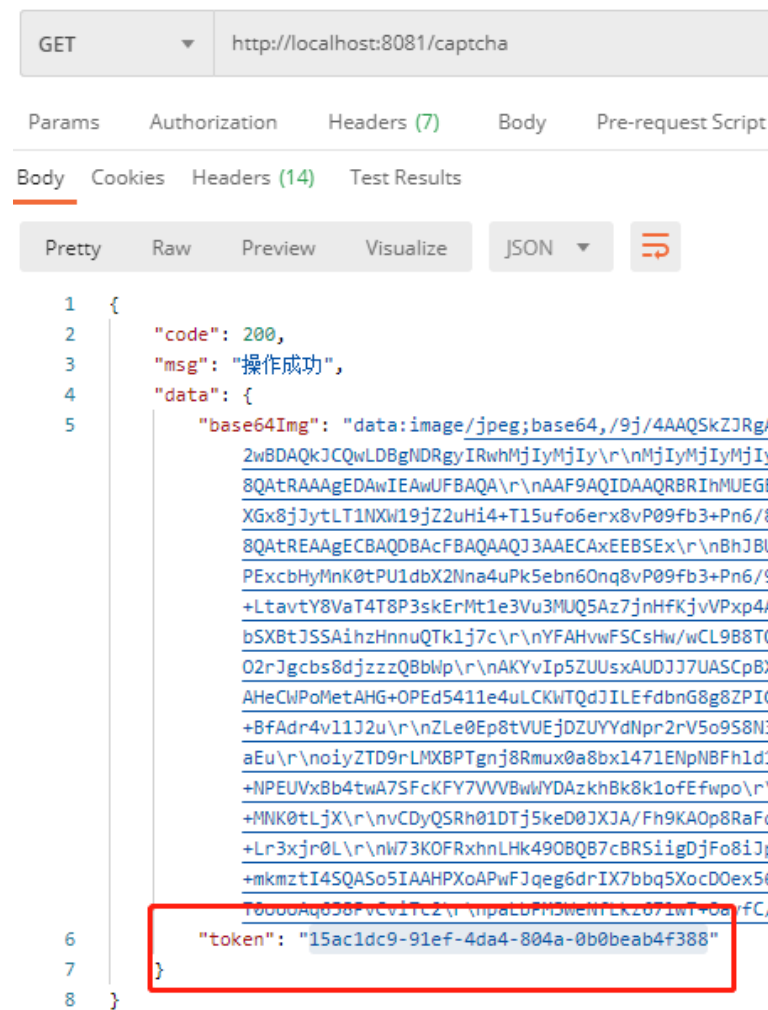
- com.markerhub.security.AccountUser

```
1 public class AccountUser implements UserDetails {
2     private Long userId;
3     private String password;
4     private final String username;
5     private final Collection<? extends GrantedAuthority> authorities;
6     private final boolean accountNonExpired;
7     private final boolean accountNonLocked;
8     private final boolean credentialsNonExpired;
9     private final boolean enabled;
10    public AccountUser(Long userId, String username, String password, Collection<? extends GrantedAuthority> authorities) {
11        this(userId, username, password, true, true, true, true, authorities);
12    }
13    public AccountUser(Long userId, String username, String password, boolean accountNonExpired, boolean accountNonLocked, boolean credentialsNonExpired, boolean enabled, Collection<? extends GrantedAuthority> authorities) {
14        Assert.isTrue(username != null && !"".equals(username) && password != null && !"".equals(password));
15        this.userId = userId;
16        this.username = username;
17        this.password = password;
18        this.enabled = enabled;
19        this.accountNonExpired = accountNonExpired;
20        this.credentialsNonExpired = credentialsNonExpired;
21        this.accountNonLocked = accountNonLocked;
22        this.authorities = authorities;
23    }
24    public Long getUserId() {
25        return userId;
26    }
27    ...
28 }
```

其实数据基本没变, 我就添加多了一个用户的id而已。

ok, 万事俱备, 我们再次尝试去登录, 看能不能登录成功。

### 1、获取验证码：



### 2、从控制台获取到对应的验证码

: Secured GET /captcha  
: 验证码 -- 15ac1dc9-91ef-4da4-804a-0b0beab4f388 - xb6x3

### 3、提交登录表单

POST http://localhost:8081/login?username=admin&password=111111&code=xb6x3

Params Authorization Headers (8) Body Pre-request Script Tests

Query Params

KEY	VALUE
username	admin
password	111111
code	xb6x3
token	15ac1dc9-91ef-4da4-804a-0b0beab4f388
Key	Value

Body Cookies Headers (15) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "操作成功",
3   "code": 200,
4   "data": ""
5 }
```

微信搜一搜 MarkerHub

4、登录成功，并在请求头中获取到了Authorization，也就是JWT。完美！！

## 解决授权

然后关于权限部分，也是security的重要功能，当用户认证成功之后，我们就知道谁在访问系统接口，这是又有一个问题，就是这个用户有没有权限来访问我们这个接口呢，要解决这个问题，我们需要知道用户有哪些权限，哪些角色，这样security才能我们做权限判断。

之前我们已经定义及几张表，用户、角色、菜单、以及一些关联表，一般当权限粒度比较细的时候，我们都通过判断用户有没有此菜单或操作的权限，而不是通过角色判断，而用户和菜单是不直接做关联的，是通过用户拥有哪些角色，然后角色拥有哪些菜单权限这样来获得的。

问题1：我们是在哪里赋予用户权限的？有两个地方：

- 1、用户登录，调用调用UserDetailsService.loadUserByUsername()方法时候可以返回用户的权限信息。
- 2、接口调用进行身份认证过滤器时候JWTAuthenticationFilter，需要返回用户权限信息

问题2：在哪里决定什么接口需要什么权限？

Security内置的权限注解：

- @PreAuthorize：方法执行前进行权限检查
- @PostAuthorize：方法执行后进行权限检查
- @Secured：类似于 @PreAuthorize

可以在Controller的方法前添加这些注解表示接口需要什么权限。

比如需要Admin角色权限：

```
1 @PreAuthorize("hasRole('admin')")
```

比如需要添加管理员的操作权限

```
1 @PreAuthorize("hasAuthority('sys:user:save')")
```

ok, 我们再来整体梳理一下授权、验证权限的流程:

1. 用户登录或者调用接口时候识别到用户, 并获取到用户的权限信息
2. 注解标识Controller中的方法需要的权限或角色
3. Security通过FilterSecurityInterceptor匹配URI和权限是否匹配
4. 有权限则可以访问接口, 当无权限的时候返回异常交给AccessDeniedHandler操作类处理

ok, 流程清晰之后我们就开始我们的编码:

- UserDetailsServiceImpl

```
1 @Override
2 public UserDetails loadUserByUsername(String username) throws UsernameNotFou
3
4     ...
5
6     return new AccountUser(sysUser.getId(), sysUser.getUsername(), sysUser.ge
7 }
8
9 public List<GrantedAuthority> getUserAuthority(Long userId) {
10     // 通过内置的工具类, 把权限字符串封装成GrantedAuthority列表
11     return AuthorityUtils.commaSeparatedStringToAuthorityList(
12         sysUserService.getUserAuthorityInfo(userId)
13     );
14 }
```

- com.markerhub.security.JWTAuthenticationFilter

```
1 SysUser sysUser = sysUserService.getByUsername(username);
2 List<GrantedAuthority> grantedAuthorities = userDetailsService.getUserAuthor
3 UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken
4     = new UsernamePasswordAuthenticationToken(username, null, grantedAutho
5
```

代码中的com.markerhub.service.impl.SysUserServiceImpl#getUserAuthorityInfo是重点:

```
1 @Slf4j
2 @Service
3 public class SysUserServiceImpl extends ServiceImpl<SysUserMapper, SysUser>
4
5     ...
6
7     @Override
8     public String getUserAuthorityInfo(Long userId) {
9         SysUser sysUser = this.getById(userId);
10         String authority = null;
11
12         if (redisUtil.hasKey("GrantedAuthority:" + sysUser.getUsername())) {
13             // 优先从缓存获取
14             authority = (String)redisUtil.get("GrantedAuthority:" + sysUser.get
15
```

```

16         } else {
17
18             List<SysRole> roles = sysRoleService.list(new QueryWrapper<SysRole>
19                 .inSql("id", "select role_id from sys_user_role where user_id = " + userId));
20             List<Long> menuIds = sysUserMapper.getNavMenuIds(userId);
21             List<SysMenu> menus = sysMenuService.listByIds(menuIds);
22
23             String roleNames = roles.stream().map(r -> "ROLE_" + r.getCode()).collect(Collectors.joining(", "));
24             String permNames = menus.stream().map(m -> m.getPerms()).collect(Collectors.joining(", "));
25
26             authority = roleNames.concat(",").concat(permNames);
27             log.info("用户ID - {} ---拥有的权限: {}", userId, authority);
28
29             redisUtil.set("GrantedAuthority:" + sysUser.getUsername(), authority);
30
31         }
32         return authority;
33     }
34 }

```

可以看到，我通过用户id分别获取到用户的角色信息和菜单信息，然后通过逗号链接起来，因为角色信息我们需要这样“ROLE\_”+角色，所以才有了上面的写法：

比如用户拥有Admin角色和添加用户权限，则最后的字符串是：**ROLE\_admin,sys:user:save**。

同时为了避免多次查库，我做了一层缓存，这里理解应该不难。

然后sysUserMapper.getNavMenuIds(userId)因为要查询数据库，具体SQL如下：

- com.markerhub.mapper.SysUserMapper#getNavMenuIds

```

1 <select id="getNavMenuIds" resultType="java.lang.Long">
2     SELECT
3         DISTINCT rm.menu_id
4     FROM
5         sys_user_role ur
6     LEFT JOIN `sys_role_menu` rm ON rm.role_id = ur.role_id
7     WHERE
8         ur.user_id = #{userId};
9 </select>

```

上面表示通过用户ID获取用户关联的菜单的id，因此需要用到两个中间表的关联了。

ok，这样我们就赋予了用户角色和操作权限了。后面我们只需要在Controller添加上具体注解表示需要的权限，Security就会自动帮我们自动完成权限校验了。

## 权限缓存

因为上面我在获取用户权限那里添加了个缓存，这时候问题来了，就是权限缓存的实时更新问题，比如当后台更新某个管理员的权限角色信息的时候如果权限缓存信息没有实时更新，就会出现操作无效的问题，那么我们现在来定义几个方法，用于清除某个用户或角色或者某个菜单的权限的方法：

- com.markerhub.service.impl.SysUserServiceImpl



```

1 // 删除某个用户的权限信息
2 @Override
3 public void clearUserAuthorityInfo(String username) {
4     redisUtil.del("GrantedAuthority:" + username);
5 }
6
7 // 删除所有与该角色关联的用户的权限信息
8 @Override
9 public void clearUserAuthorityInfoByRoleId(Long roleId) {
10     List<SysUser> sysUsers = this.list(new QueryWrapper<SysUser>()
11         .inSql("id", "select user_id from sys_user_role where role_id = " + roleId));
12 };
13 sysUsers.forEach(u -> {
14     this.clearUserAuthorityInfo(u.getUsername());
15 });
16 }
17
18 // 删除所有与该菜单关联的所有用户的权限信息
19 @Override
20 public void clearUserAuthorityInfoByMenuId(Long menuId) {
21     List<SysUser> sysUsers = sysUserMapper.listByMenuId(menuId);
22     sysUsers.forEach(u -> {
23         this.clearUserAuthorityInfo(u.getUsername());
24     });
25 }

```

上面最后一个方法查到了与菜单关联的所有用户的，具体sql如下：

- com.markerhub.mapper.SysUserMapper#listByMenuId

```

1 <select id="listByMenuId" resultType="com.javacat.entity.SysUser">
2     SELECT
3     DISTINCT
4         su.*
5     FROM
6         sys_user_role ur
7     LEFT JOIN `sys_role_menu` rm ON rm.role_id = ur.role_id
8     LEFT JOIN `sys_user` su ON su.id = ur.user_id
9     WHERE
10         rm.menu_id = #{menuId};
11 </select>

```

有了这几个方法之后，在哪里调用？这就简单了，在更新、删除角色权限、更新、删除菜单的时候调用，虽然我们现在还没写到这几个方法，后续我们再写增删改查的时候记得加上就行啦。

## 退出数据返回

jwt -username

token - 随机码 - redis

- com.markerhub.security.JwtLogoutSuccessHandler

```

1 @Component

```

```

2 public class JwtLogoutSuccessHandler implements LogoutSuccessHandler {
3     @Autowired
4     JwtUtils jwtUtils;
5     @Override
6     public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response)
7         throws IOException, ServletException {
8         if (authentication != null) {
9             new SecurityContextLogoutHandler().logout(request, response, authentication);
10        }
11        response.setContentType("application/json;charset=UTF-8");
12        response.setHeader(jwtUtils.getHeader(), "");
13        ServletOutputStream out = response.getOutputStream();
14        Result result = Result.succ("");
15        out.write(JSONUtil.toJsonStr(result).getBytes("UTF-8"));
16        out.flush();
17        out.close();
18    }
19

```

## 无权限数据返回

- com.markerhub.security.JwtAccessDeniedHandler

```

1 @Slf4j
2 @Component
3 public class JwtAccessDeniedHandler implements AccessDeniedHandler {
4     @Override
5     public void handle(HttpServletRequest request, HttpServletResponse response)
6         throws IOException, ServletException {
7         // response.sendError(HttpServletResponse.SC_FORBIDDEN, accessDeniedException.getMessage());
8         log.info("权限不够！！");
9         response.setContentType("application/json;charset=UTF-8");
10        response.setStatus(HttpServletResponse.SC_FORBIDDEN);
11        ServletOutputStream outputStream = response.getOutputStream();
12        Result result = Result.fail(accessDeniedException.getMessage());
13        outputStream.write(JSONUtil.toJsonStr(result).getBytes("UTF-8"));
14        outputStream.flush();
15        outputStream.close();
16    }
17 }

```

至此，SpringSecurity就已经完美整合到了我们的项目中来了。

## 7. 解决跨域问题

上面的调试我们都是使用的postman，如果我们和前端进行对接的时候，会出现跨域的问题，如何解决？

- com.markerhub.config.CorsConfig

```

1 @Configuration
2 public class CorsConfig implements WebMvcConfigurer {

```

```

3
4     private CorsConfiguration buildConfig() {
5         CorsConfiguration corsConfiguration = new CorsConfiguration();
6         corsConfiguration.addAllowedOrigin("*");
7         corsConfiguration.addAllowedHeader("*");
8         corsConfiguration.addAllowedMethod("*");
9         corsConfiguration.addExposedHeader("Authorization");
10        return corsConfiguration;
11    }
12
13    @Bean
14    public CorsFilter corsFilter() {
15        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
16        source.registerCorsConfiguration("/**", buildConfig());
17        return new CorsFilter(source);
18    }
19
20    @Override
21    public void addCorsMappings(CorsRegistry registry) {
22        registry.addMapping("/**")
23            .allowedOrigins("*")
24            // .allowCredentials(true)
25            .allowedMethods("GET", "POST", "DELETE", "PUT")
26            .maxAge(3600);
27    }
28 }

```

## 8. 前后端对接的问题

因为我们之前开发前端的时候，我们都是使用mockjs返回随机数据的，一般来说问题不会很大，我就怕有些同学再去掉mock之后，和后端对接却显示不出数据，这就尴尬了。这时候我建议你看我的开发视频哈。

后面因为都是接口的增删改查，难度其实不是特别大，所以大部分时候我都会直接贴代码，如果想看手把手教程，还是去看我的教学视频哈，B站搜索MarkerHub就可以啦，公众号也是叫MarkerHub。

## 9. 菜单接口开发

我们先来开发菜单的接口，因为这3个表：用户表、角色表、菜单表，才有菜单表是不需要通过其他表来获取信息的。比如用户需要关联角色，角色需要关联菜单，而菜单不需要主动关联其他表。因此菜单表的增删改查是最简单的。

再回到我们的前端项目，登录完成之后我们通过JWT获取项目的导航菜单和权限，那么接下来我们就先编写这个接口。

获取菜单导航和权限的链接是/sys/menu/nav，然后我们的菜单导航的json数据应该是这样的：

```

1 {

```

```

2     title: '角色管理',
3     icon: 'el-icon-rank',
4     path: '/sys/roles',
5     name: 'SysRoles',
6     component: 'sys/Role',
7     children: []
8 }

```

然后返回的权限数据应该是个数组：

```

1 ["sys:menu:list","sys:menu:save","sys:user:list"...]

```

注意导航菜单那里有个children，也就是子菜单，是个树形结构，因为我们的菜单可能这样：

```

1 系统管理 - 菜单管理 - 添加菜单

```

可以看到这就已经有3级了菜单了。

所以在打代码时候要注意这个关系的关联。我们的SysMenu实体类中有个parentId，但是没有children，因此我们可以在SysMenu中添加一个children，当然了其实不添加也可以，因为我们也需要一个dto，这样我们才能按照上面json数据格式返回。

我们还是来添加一个children吧：

- com.markerhub.entity.SysMenu

```

1 @Data
2 @EqualsAndHashCode(callSuper = true)
3 public class SysMenu extends BaseEntity {
4
5     ...
6
7     @TableField(exist = false)
8     private List<SysMenu> children = new ArrayList<>();
9 }

```

然后我们也先来定义一个SysMenuDto吧，知道要返回什么样的数据，我们就只需要去填充数据就好了

- com.markerhub.common.dto.SysMenuDto

```

1 @Data
2 public class SysMenuDto implements Serializable {
3     private Long id;
4     private String title;
5     private String icon;
6     private String path;
7     private String name;
8     private String component;
9     List<SysMenuDto> children = new ArrayList<>();
10 }

```

ok，我们来开始我们的编码

- com.markerhub.controller.SysMenuController#nav

```

1 /**
2  * 获取当前用户的菜单栏以及权限

```

```

3  */
4  @GetMapping("/nav")
5  public Result nav(Principal principal) {
6      String username = principal.getName();
7      SysUser sysUser = sysUserService.getByUsername(username);
8      // ROLE_Admin,sys:user:save
9      String[] authoritys = StringUtils.tokenizeToStringArray(
10         sysUserService.getUserAuthorityInfo(sysUser.getId())
11         , ",");
12     return Result.succ(
13         MapUtil.builder()
14             .put("nav", sysMenuService.getCurrentUserNav())
15             .put("authoritys", authoritys)
16             .map()
17     );
18 }

```

方法中Principal principal表示注入当前用户的信息，getName就可以获取当前用户的用户名了。sysUserService.getUserAuthorityInfo方法我们之前已经说过了，就在我们登录完成或者身份认证时候需要返回用户权限时候编写的。然后通过StringUtils.tokenizeToStringArray把字符串通过逗号分开组成数组形式。

重点在与sysMenuService.getCurrentUserNav，获取当前用户的菜单导航，

```

1  @Service
2  public class SysMenuServiceImpl extends ServiceImpl<SysMenuMapper, SysMenu>
3
4      ...
5
6      /**
7       * 获取当前用户菜单导航
8       */
9      @Override
10     public List<SysMenuDto> getCurrentUserNav() {
11         String username = (String) SecurityContextHolder.getContext().getAuthentication().getPrincipal().getName();
12
13         SysUser sysUser = sysUserService.getByUsername(username);
14
15         // 获取用户的所有菜单
16         List<Long> menuIds = sysUserMapper.getNavMenuIds(sysUser.getId());
17
18         List<SysMenu> menus = buildTreeMenu(this.listByIds(menuIds));
19         return convert(menus);
20     }
21
22     /**
23      * 把list转成树形结构的数据
24      */
25     private List<SysMenu> buildTreeMenu(List<SysMenu> menus){
26         List<SysMenu> finalMenus = new ArrayList<>();
27         for (SysMenu menu : menus) {
28

```

```

29         // 先寻找各自的孩子
30         for (SysMenu e : menus) {
31             if (e.getParentId() == menu.getId()) {
32                 menu.getChildren().add(e);
33             }
34         }
35         // 提取出父节点
36         if (menu.getParentId() == 0L) {
37             finalMenus.add(menu);
38         }
39     }
40     return finalMenus;
41 }
42
43 /**
44  * menu转menuDto
45  */
46 private List<SysMenuDto> convert(List<SysMenu> menus) {
47     List<SysMenuDto> menuDtos = new ArrayList<>();
48     menus.forEach(m -> {
49         SysMenuDto dto = new SysMenuDto();
50         dto.setId(m.getId());
51         dto.setName(m.getPerms());
52         dto.setTitle(m.getName());
53         dto.setComponent(m.getComponent());
54         dto.setIcon(m.getIcon());
55         dto.setPath(m.getPath());
56         if (m.getChildren().size() > 0) {
57             dto.setChildren(convert(m.getChildren()));
58         }
59         menuDtos.add(dto);
60     });
61     return menuDtos;
62 }
63 }

```

接口中sysUserMapper.getNavMenuIds我们之前就已经写过的了，通过用户id获取菜单的id，然后后面就是转成树形结构，buildTreeMenu方法的思想很简单，我们现实把菜单循环，让所有菜单先找到各自的子节点，然后我们在把最顶级的菜单获取出来，这样顶级下面有二级，二级也有自己的三级。最后就是convert把menu转成menuDto。这个比较简单，就不说了。

好了，导航菜单已经开发完毕，我们来写菜单管理的增删改查，因为菜单列表也是个树形接口，这次我们就不是获取当前用户的菜单列表的，而是所有菜单然后组成树形结构，一样的思想，数据不一样而已。

- com.markerhub.controller.SysMenuController

```

1 @GetMapping("/info/{id}")
2 @PreAuthorize("hasAuthority('sys:menu:list')")
3 public Result info(@PathVariable("id") Long id) {
4     return Result.succ(sysMenuService.getById(id));
5 }
6

```

```

7  @GetMapping("/list")
8  @PreAuthorize("hasAuthority('sys:menu:list')")
9  public Result list() {
10     List<SysMenu> menus = sysMenuService.tree();
11     return Result.succ(menus);
12 }
13
14 @PostMapping("/save")
15 @PreAuthorize("hasAuthority('sys:menu:save')")
16 public Result save(@Validated @RequestBody SysMenu sysMenu) {
17     sysMenu.setCreated(LocalDateTime.now());
18     sysMenu.setStatus(Const.STATUS_ON);
19     sysMenuService.save(sysMenu);
20     return Result.succ(sysMenu);
21 }
22
23 @PostMapping("/update")
24 @PreAuthorize("hasAuthority('sys:menu:update')")
25 public Result update(@Validated @RequestBody SysMenu sysMenu) {
26     sysMenu.setUpdated(LocalDateTime.now());
27     sysMenuService.updateById(sysMenu);
28     // 清除所有与该菜单相关的权限缓存
29     sysUserService.clearUserAuthorityInfoByMenuId(sysMenu.getId());
30     return Result.succ(sysMenu);
31 }
32
33 @Transactional
34 @PostMapping("/delete/{id}")
35 @PreAuthorize("hasAuthority('sys:menu:delete')")
36 public Result delete(@PathVariable Long id) {
37
38     int count = sysMenuService.count(new QueryWrapper<SysMenu>().eq("parent_
39     if (count > 0) {
40         return Result.fail("请先删除子菜单");
41     }
42
43     // 先清除所有与该菜单相关的权限缓存
44     sysUserService.clearUserAuthorityInfoByMenuId(id);
45     sysMenuService.removeById(id);
46
47     // 同步删除
48     sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().eq("menu_id",
49     return Result.succ("");
50 }

```

删除、更新菜单的时候记得调用根据菜单id清楚用户权限缓存信息的方法哈。然后每个方法前都会带有权限注解：@PreAuthorize("hasAuthority('sys:menu:delete')")，这就要求用户有特定的操作权限才能调用这个接口，sys:menu:delete这些数据不是乱写出来的，我们必须和数据库的数据保持一致才行，然后component字段，也是要和前端进行沟通，因为这个是链接到的前端的组件页面。

有了增删改查，我们就先去添加我们的所有的菜单权限数据先。效果如下：

首页

系统管理

用户管理

角色管理

菜单管理

系统工具

VueAdmin后台管理系统

admin - 欢迎语 - 注销

首页

用户管理

菜单管理

新增

名称	权限编码	图标	类型	菜单URL	菜单组件	排序号	状态	操作
系统管理	sys:manage	el-icon-s-operation	目录				正常	编辑 删除
用户管理	sys:user:list	el-icon-s-custom	菜单	/sys/users	sys/User		正常	编辑 删除
添加用户	sys:users:save		按钮				正常	编辑 删除
修改用户	sys:user:update		按钮				正常	编辑 删除
删除用户	sys:user:delete		按钮				正常	编辑 删除
分配角色	sys:user:role		按钮				正常	编辑 删除
重置密码	sys:user:repass		按钮				正常	编辑 删除
角色管理	sys:role:list	el-icon-rank	菜单	/sys/roles	sys/Role		正常	编辑 删除
添加角色	sys:roles:save		按钮				禁用	编辑 删除
修改角色	sys:role:update		按钮				正常	编辑 删除
删除角色	sys:role:delete		按钮				正常	编辑 删除
分配权限	sys:role:perm		按钮				正常	编辑 删除
菜单管理	sys:menu:list	el-icon-menu	菜单	/sys/menus	sys/Menu		正常	编辑 删除
添加菜单	sys:menus:save		按钮				正常	编辑 删除
修改菜单	sys:menu:update		按钮				正常	编辑 删除
删除菜单	sys:menu:delete		按钮				正常	编辑 删除
系统工具	sys:tools	el-icon-s-tools	目录				正常	编辑 删除
数字字典	sys:dict:list	el-icon-s-order	菜单	/sys/dicts	sys/Dict		正常	编辑 删除

首页

角色管理

用户管理

菜单管理

新增

名称	权限编码	图标
系统管理	sys:manage	el-icon-s-operation
用户管理	sys:user:list	el-icon-s-custom
添加用户	sys:users:save	
修改用户	sys:user:update	
删除用户	sys:user:delete	
分配角色	sys:user:role	
重置密码	sys:user:repass	
角色管理	sys:role:list	el-icon-rank
添加角色	sys:roles:save	
修改角色	sys:role:update	
删除角色	sys:role:delete	
分配权限	sys:role:perm	
菜单管理	sys:menu:list	el-icon-menu
添加菜单	sys:menus:save	
修改菜单	sys:menu:update	
删除菜单	sys:menu:delete	
系统工具	sys:tools	el-icon-s-tools

菜单信息

\* 上级菜单

0

\* 菜单名称

系统管理

\* 权限编码

sys:manage

图标

el-icon-s-operation

菜单URL

菜单组件

\* 类型

☒ 目录 ☐ 菜单 ☐ 按钮

\* 状态

☐ 禁用 ☒ 正常

\* 排序号

-

+

取消

确定

基本上线填好所有菜单的列表和增删改查操作权限，就ok。

## 10. 角色接口开发

角色的增删改查其实也简单，而且字段这么少，基本上吧菜单的增删改查复制过来，然后把menu改成role，在调整一下就差不多啦。然后有个角色关联菜单的操作，这个我们等下讲讲，先来看代码：

```
1 @RestController
2 @RequestMapping("/sys/role")
3 public class SysRoleController extends BaseController {
4
5     @GetMapping("/info/{id}")
```



```

6      @PreAuthorize("hasAuthority('sys:role:list')")
7      public Result info(@PathVariable Long id) {
8          SysRole role = sysRoleService.getById(id);
9          List<SysRoleMenu> roleMenus = sysRoleMenuService.list(new QueryWrapper<SysRoleMenu>()
10              .eq("role_id", id));
11          List<Long> menuIds = roleMenus.stream().map(p -> p.getMenuId()).collect(Collectors.toList());
12          role.setMenuIds(menuIds);
13          return Result.succ(role);
14      }
15
16      @GetMapping("/list")
17      @PreAuthorize("hasAuthority('sys:role:list')")
18      public Result list(String name) {
19          Page<SysRole> roles = sysRoleService.page(getPage(),
20              new QueryWrapper<SysRole>()
21                  .like(StrUtil.isNotBlank(name), "name", name)
22              );
23          return Result.succ(roles);
24      }
25
26      @PostMapping("/save")
27      @PreAuthorize("hasAuthority('sys:role:save')")
28      public Result save(@Validated @RequestBody SysRole sysRole) {
29          sysRole.setCreated(LocalDateTime.now());
30          sysRole.setStatus(Const.STATUS_ON);
31          sysRoleService.save(sysRole);
32          return Result.succ(sysRole);
33      }
34
35      @PostMapping("/update")
36      @PreAuthorize("hasAuthority('sys:role:update')")
37      public Result update(@Validated @RequestBody SysRole sysRole) {
38          sysRole.setUpdated(LocalDateTime.now());
39          sysRoleService.updateById(sysRole);
40          return Result.succ(sysRole);
41      }
42
43      @Transactional
44      @PostMapping("/delete")
45      @PreAuthorize("hasAuthority('sys:role:delete')")
46      public Result delete(@RequestBody Long[] ids){
47          sysRoleService.removeByIds(Arrays.asList(ids));
48          // 同步删除
49          sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().in("role_id", ids));
50          sysUserRoleService.remove(new QueryWrapper<SysUserRole>().in("role_id", ids));
51          return Result.succ("");
52      }
53
54      @Transactional
55      @PostMapping("/perm/{roleId}")
56      @PreAuthorize("hasAuthority('sys:role:perm')")
57      public Result perm(@PathVariable Long roleId, @RequestBody Long[] menuIds) {

```

```

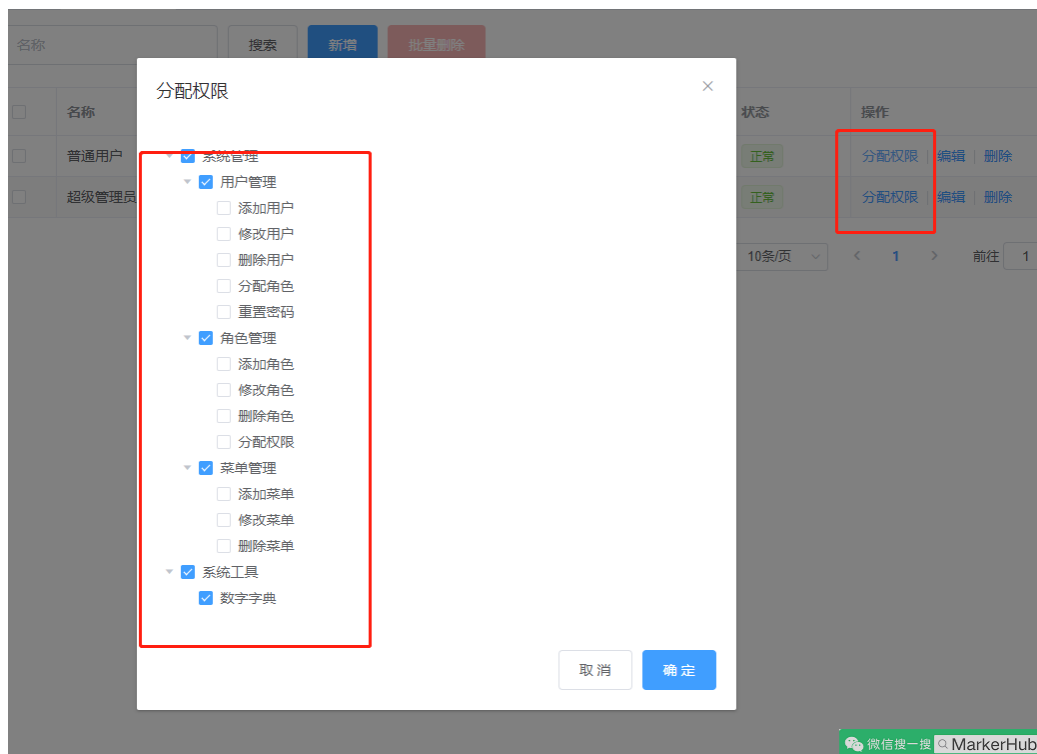
57     List<SysRoleMenu> sysRoleMenus = new ArrayList<>();
58     Arrays.stream(menuIds).forEach(menuId -> {
59         SysRoleMenu roleMenu = new SysRoleMenu();
60         roleMenu.setMenuId(menuId);
61         roleMenu.setRoleId(roleId);
62         sysRoleMenus.add(roleMenu);
63     });
64
65     sysRoleMenuService.remove(new QueryWrapper<SysRoleMenu>().eq("role_id", roleId));
66
67     sysRoleMenuService.saveBatch(sysRoleMenus);
68
69     // 清除所有用户的权限缓存信息
70     sysUserService.clearUserAuthorityInfoByRoleId(roleId);
71     return Result.succ(menuIds);
72 }
73 }

```

上面方法中：

- info方法

获取角色信息的方法，因为我们不仅仅在编辑角色时候会用到这个方法，在回显角色关联菜单的时候也需要被调用，因此我们需要把角色关联的所有的菜单的id也一并查询出来，也就是分配权限的操作。对应到前端就是这样的，点击分配权限，会弹出出所有的菜单列表，然后根据角色已经关联的菜单的id回显勾选上已经关联过的。效果如下：



然后点击保存分配权限的时候，我们需要把角色的id和所有勾选上的菜单id的数组一起传过来，所以才有了controller中的这样的写法：

```

1 @PreAuthorize("hasAuthority('sys:role:perm')")
2 public Result perm(@PathVariable Long roleId, @RequestBody Long[] menuIds) {
3
4     ...代码上面贴出来
5 }

```

可以看到，因为@RequestBody，我们知道menuIds是否安装body里面的，这个需要注意，对应到的前端写法就是这样：

```
submitPermForm(formName) {  
  var menuIds = []  
  
  menuIds = this.$refs.permTree.getCheckedKeys()  
  
  console.log(menuIds)  
  console.log(this.permForm.id)  
  
  this.$axios.post(url: "/sys/role/perm/" + this.permForm.id, menuIds).then(res => {  
    this.$message({  
      showClose: true,  
      message: '恭喜你，操作成功',  
      type: 'success',  
      onClose: () => {  
        this.resetForm(formName)  
      }  
    });  
    this.permDialogFormVisible = false  
  });  
},
```

首发公众号：MarkerHub  
更多Java项目学习

微信搜一搜 MarkerHub

ok，角色管理就讲到这里了，其他增删改查自己看下代码，不难哈。

VueAdmin后台管理系统

admin 视频讲解 网站

角色管理

名称 搜索 新增 批量删除

<input type="checkbox"/>	名称	唯一编码	描述	状态	操作
<input type="checkbox"/>	普通用户	normal	只有基本查看功能	正常	分配权限 编辑 删除
<input type="checkbox"/>	超级管理员	admin	系统默认最高权限，不可以编辑和任意修改	正常	分配权限 编辑 删除

共 2 条 10条/页 < 1 > 前往 1 页

微信搜一搜 MarkerHub

角色信息

\* 角色名称 普通用户

\* 唯一编码 normal

描述 只有基本查看功能

\* 状态 ☐ 禁用 ☒ 正常

取消 确定

共 2 条 10条

微信搜一搜 MarkerHub

## 11. 用户接口开发

用户管理里面有个用户关联角色的分配角色操作，和角色关联菜单的写法差不多的，其他增删改查也复制黏贴改改就好，哈哈。

- com.markerhub.controller.SysUserController

```
1  /**
2   * 公众号：MarkerHub
3   */
4  @RestController
5  @RequestMapping("/sys/user")
6  public class SysUserController extends BaseController {
7
8      @Autowired
9      PasswordEncoder passwordEncoder;
10
11     @GetMapping("/info/{id}")
12     @PreAuthorize("hasAuthority('sys:user:list')")
13     public Result info(@PathVariable Long id) {
14         SysUser user = sysUserService.getById(id);
15         Assert.notNull(user, "找不到该管理员！");
16         List<SysRole> roles = sysRoleService.listRolesByUserId(user.getId());
17         user.setRoles(roles);
18         return Result.succ(user);
19     }
20
21     /**
22     * 用户自己修改密码
23     *
24     */
25     @PostMapping("/updataPass")
26     public Result updataPass(@Validated @RequestBody PassDto passDto, Principal principal) {
27         SysUser sysUser = sysUserService.getByUsername(principal.getName());
28         boolean matches = passwordEncoder.matches(passDto.getCurrentPass(), sysUser.getPassword());
29         if (!matches) {
30             return Result.fail("旧密码不正确！");
31         }
32         sysUser.setPassword(passwordEncoder.encode(passDto.getPassword()));
33         sysUser.setUpdated(LocalDateTime.now());
34         sysUserService.updateById(sysUser);
35         return Result.succ(null);
36     }
37
38     /**
39     * 超级管理员重置密码
40     */
41     @PostMapping("/repass")
42     @PreAuthorize("hasAuthority('sys:user:repass')")
43     public Result repass(@RequestBody Long userId) {
44         SysUser sysUser = sysUserService.getById(userId);
45         sysUser.setPassword(passwordEncoder.encode(Const.DEFAULT_PASSWORD));
46         sysUser.setUpdated(LocalDateTime.now());
```

```
47     sysUserService.updateById(sysUser);
48     return Result.succ(null);
49 }
50
51 @GetMapping("/list")
52 @PreAuthorize("hasAuthority('sys:user:list')")
53 public Result page(String username) {
54     Page<SysUser> users = sysUserService.page(getPage(),
55         new QueryWrapper<SysUser>()
56             .like(StrUtil.isNotBlank(username), "username", username)
57     );
58     users.getRecords().forEach(u -> {
59         u.setRoles(sysRoleService.listRolesByUserId(u.getId()));
60     });
61     return Result.succ(users);
62 }
63
64 @PostMapping("/save")
65 @PreAuthorize("hasAuthority('sys:user:save')")
66 public Result save(@Validated @RequestBody SysUser sysUser) {
67     sysUser.setCreated(LocalDateTime.now());
68     sysUser.setStatu(Const.STATUS_ON);
69     // 初始默认密码
70     sysUser.setPassword(Const.DEFAULT_PASSWORD);
71     if (StrUtil.isBlank(sysUser.getPassword())) {
72         return Result.fail("密码不能为空");
73     }
74     String password = passwordEncoder.encode(sysUser.getPassword());
75     sysUser.setPassword(password);
76     // 默认头像
77     sysUser.setAvatar(Const.DEFAULT_AVATAR);
78     sysUserService.save(sysUser);
79     return Result.succ(sysUser);
80 }
81
82 @PostMapping("/update")
83 @PreAuthorize("hasAuthority('sys:user:update')")
84 public Result update(@Validated @RequestBody SysUser sysUser) {
85     sysUser.setUpdated(LocalDateTime.now());
86     if (StrUtil.isNotBlank(sysUser.getPassword())) {
87         String password = passwordEncoder.encode(sysUser.getPassword());
88         sysUser.setPassword(password);
89     }
90     sysUserService.updateById(sysUser);
91     return Result.succ(sysUser);
92 }
93
94 @PostMapping("/delete")
95 @PreAuthorize("hasAuthority('sys:user:delete')")
96 public Result delete(@RequestBody Long[] ids){
97     sysUserService.removeByIds(Arrays.asList(ids));
```

```

98     sysUserRoleService.remove(new QueryWrapper<SysUserRole>().in("user_id'"
99     return Result.succ(""));
100 }
101
102 /**
103  * 分配角色
104  * @return
105  */
106 @Transactional
107 @PostMapping("/role/{userId}")
108 @PreAuthorize("hasAuthority('sys:user:role')")
109 public Result perm(@PathVariable Long userId, @RequestBody Long[] roleIds) {
110     System.out.println(roleIds);
111     List<SysUserRole> userRoles = new ArrayList<>();
112     Arrays.stream(roleIds).forEach(roleId -> {
113         SysUserRole userRole = new SysUserRole();
114         userRole.setRoleId(roleId);
115         userRole.setUserId(userId);
116         userRoles.add(userRole);
117     });
118     sysUserRoleService.remove(new QueryWrapper<SysUserRole>().eq("user_id'"
119     sysUserRoleService.saveBatch(userRoles);
120     // 清除权限信息
121     SysUser sysUser = sysUserService.getById(userId);
122     sysUserService.clearUserAuthorityInfo(sysUser.getUsername());
123     return Result.succ(roleIds);
124 }
125 }

```

上面用到一个sysRoleService.listRolesByUserId，通过用户id获取所有关联的角色，用到了中间表，可以写sql，这里我这样写的：

- com.markerhub.service.impl.SysRoleServiceImpl#listRolesByUserId

```

1 @Override
2 public List<SysRole> listRolesByUserId(Long userId) {
3     return this.list(
4         new QueryWrapper<SysRole>()
5             .inSql("id", "select role_id from sys_user_role where user_id=" + userId)
6     );

```

userId一定要是自己数据库查出来的，千万别让前端传过来啥就直接调用这个方法，不然可能会攻击，嘿嘿嘿~最委托就是写完整的sql，而不是这样半个sql的写法。

最后效果如下：





## 12. 项目部署

部署项目其实和vueblog的部署是一样的，自己调整一下吧少年，我有写了视频和文档的：

<https://www.bilibili.com/video/BV17A411E7aE/>

## 13. 项目总结

好了，我们终于又写完了个项目，希望能让你们学到点东西，这次写的文档有点乱，多多担待，太长了，写着写着就不知道写哪了，哈哈。

原创作者：吕一明

首发公众号：MakerHub

首发B站：MarkerHub

转载请保留此声明，感谢！