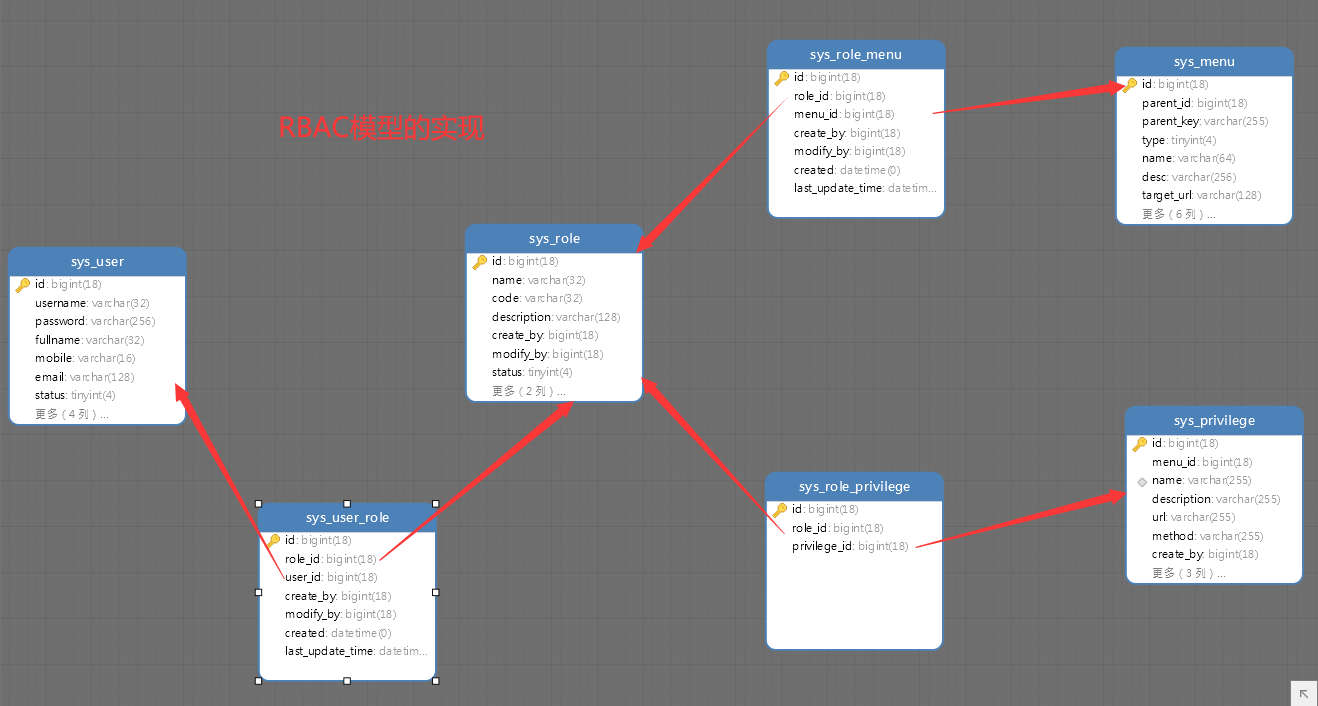
# 后台管理系统的完成

# 后台管理系统简介

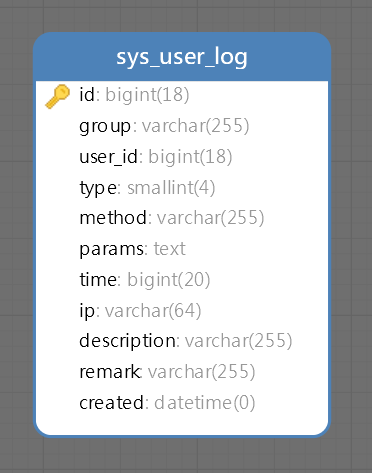
后台管理系统主要完成网站的内容管理，后台人员权限、菜单的管理，以及客服工单处理。

# 后台管理系统数据库设计

## 2.1 RBAC 模型的实现



## 2.2 操作日志记录

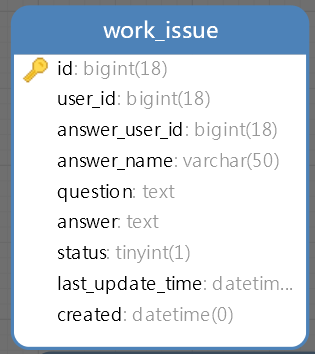


## 2.3 内容管理

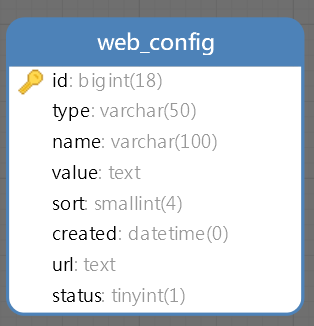
公告管理：



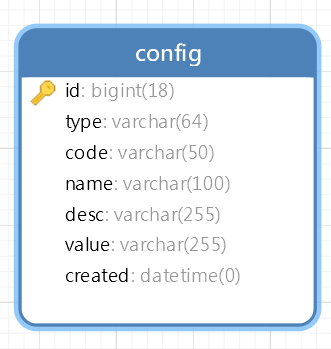
## 2.4 客服工单

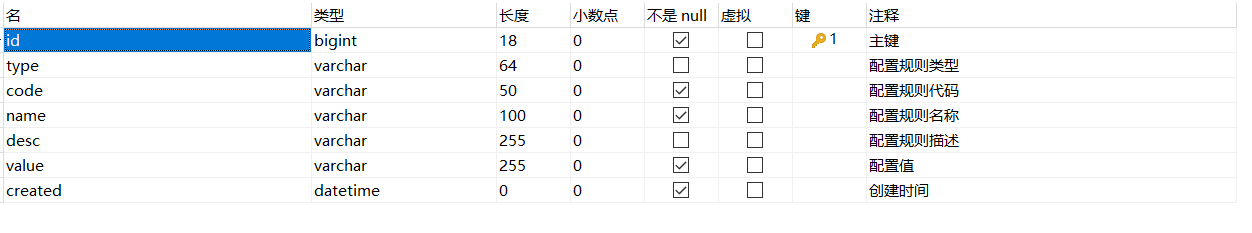


## 2.5 网站配置



## 2.6 配置项管理





# 后台管理系统API预览

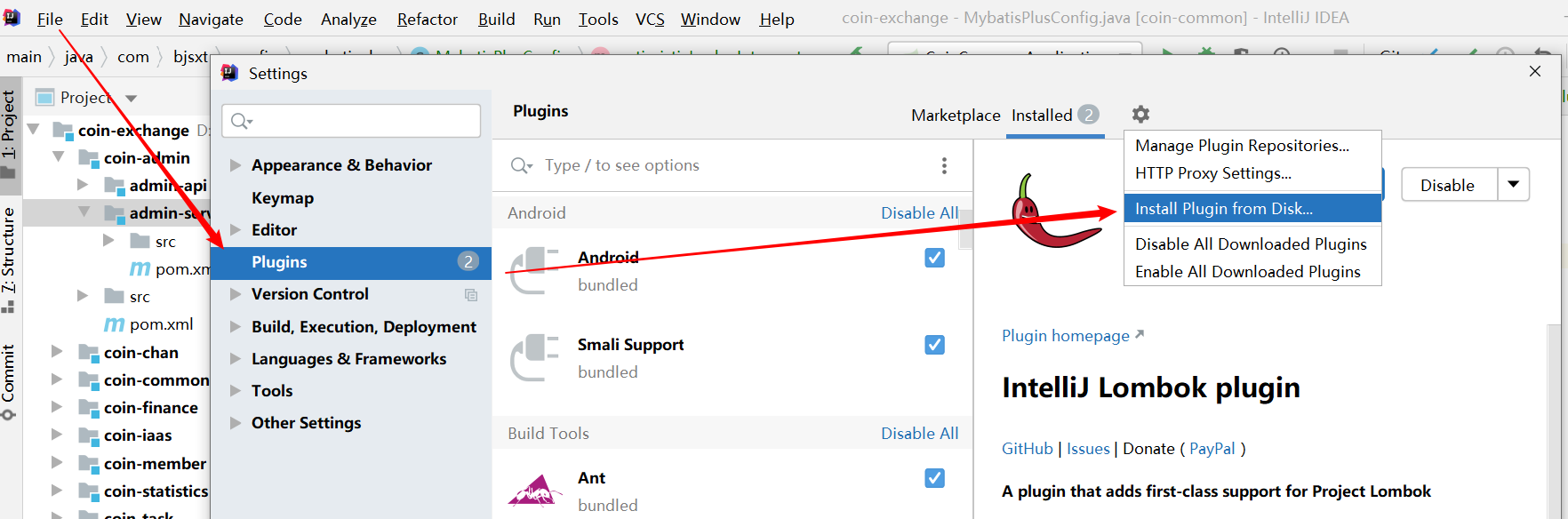
可以在YAPI里面直接搜索。

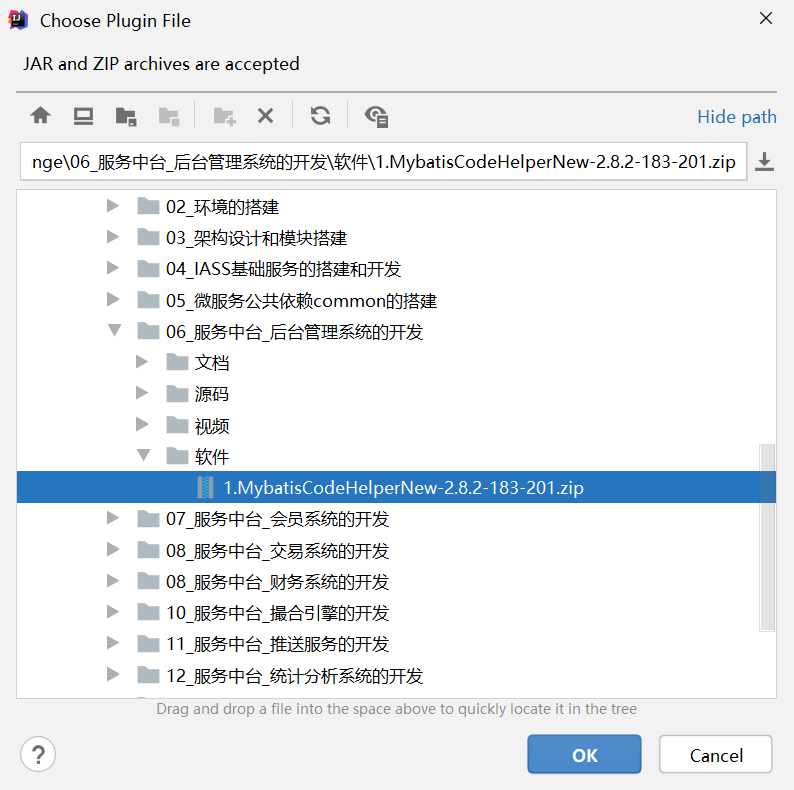


# 后台管理系统代码生成

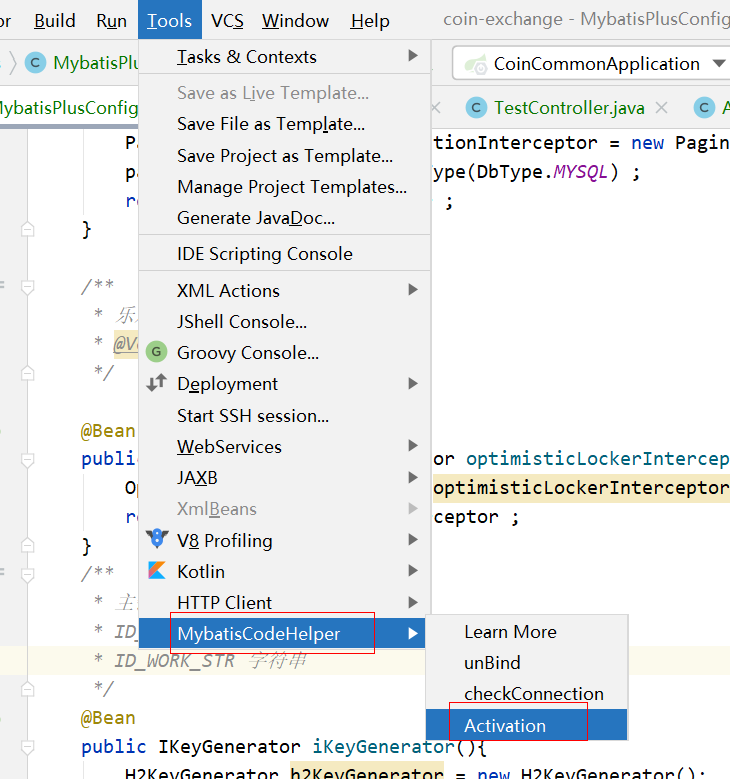
## 4.1 添加MyBatisCodeHelper-Pro代码生成器

安装插件





重启IDE,然后激活插件：

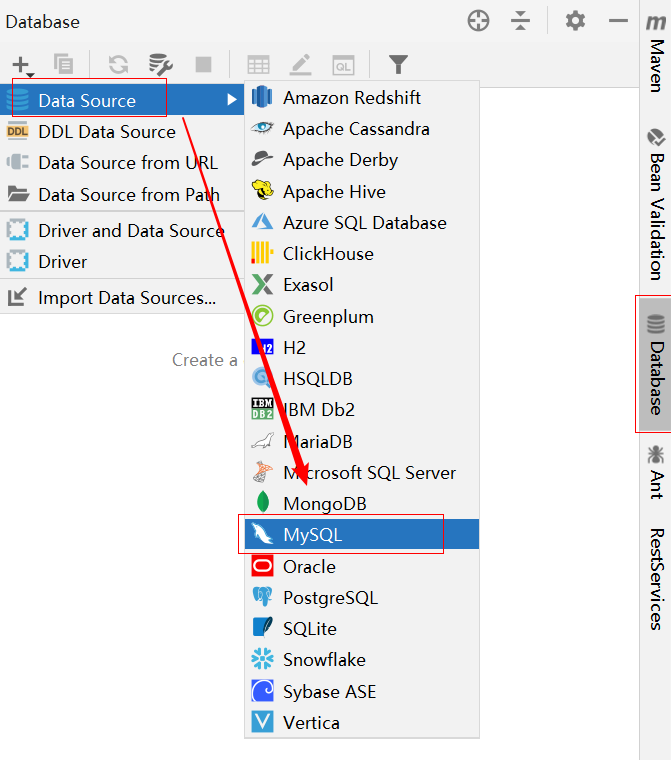


随便输入一句话：



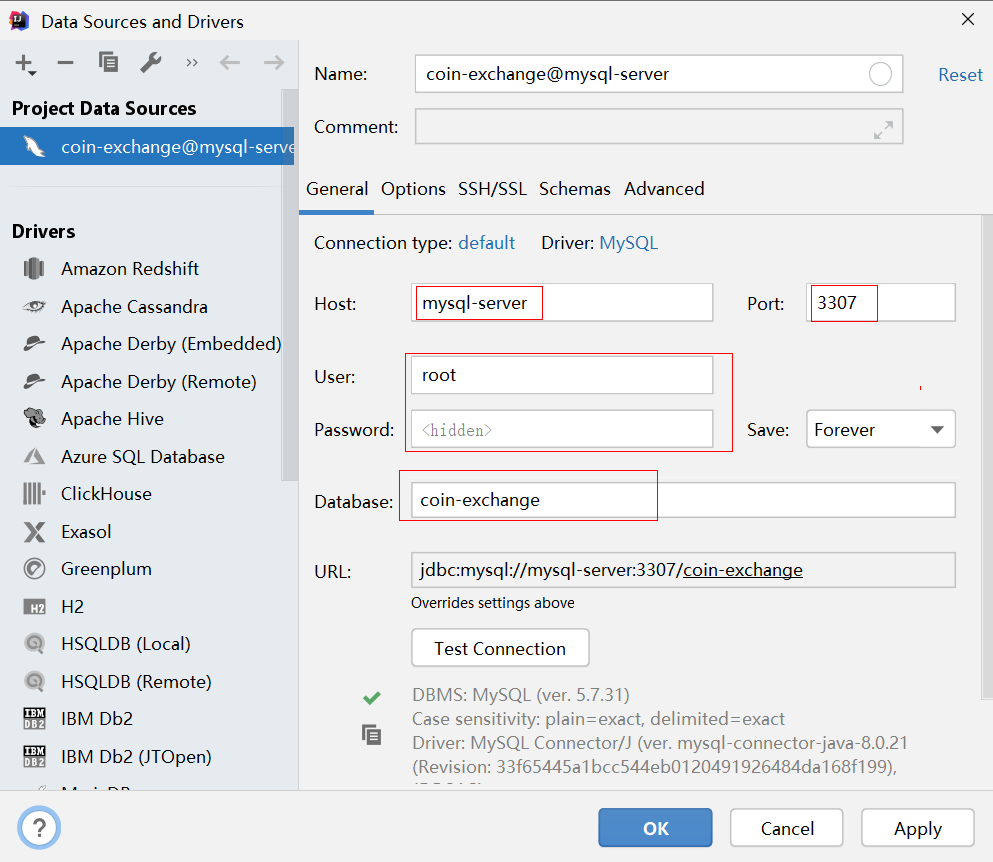
点击激活即可激活成功。

## 4.2 连接数据库

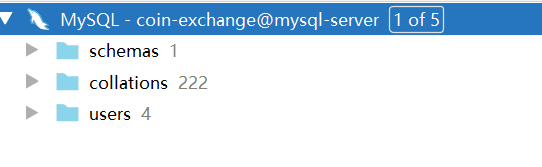


注意，第一次使用要下载驱动，时间稍微长一点。

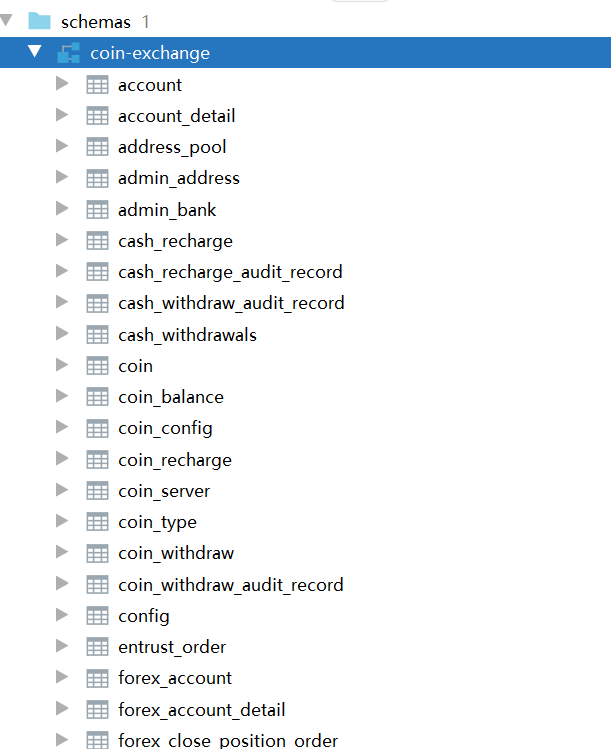
填写数据库信息：



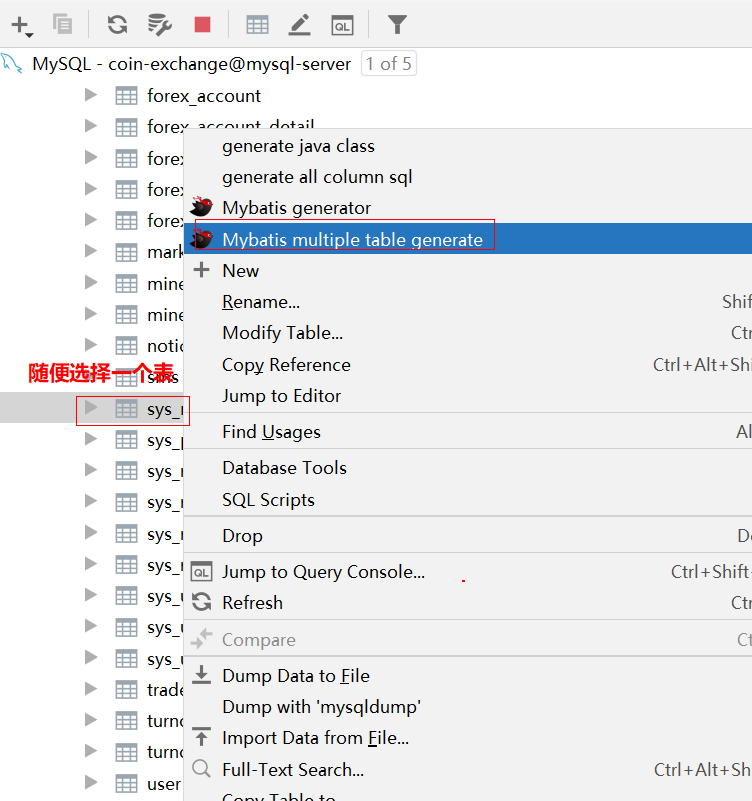
连接成功后，出现：



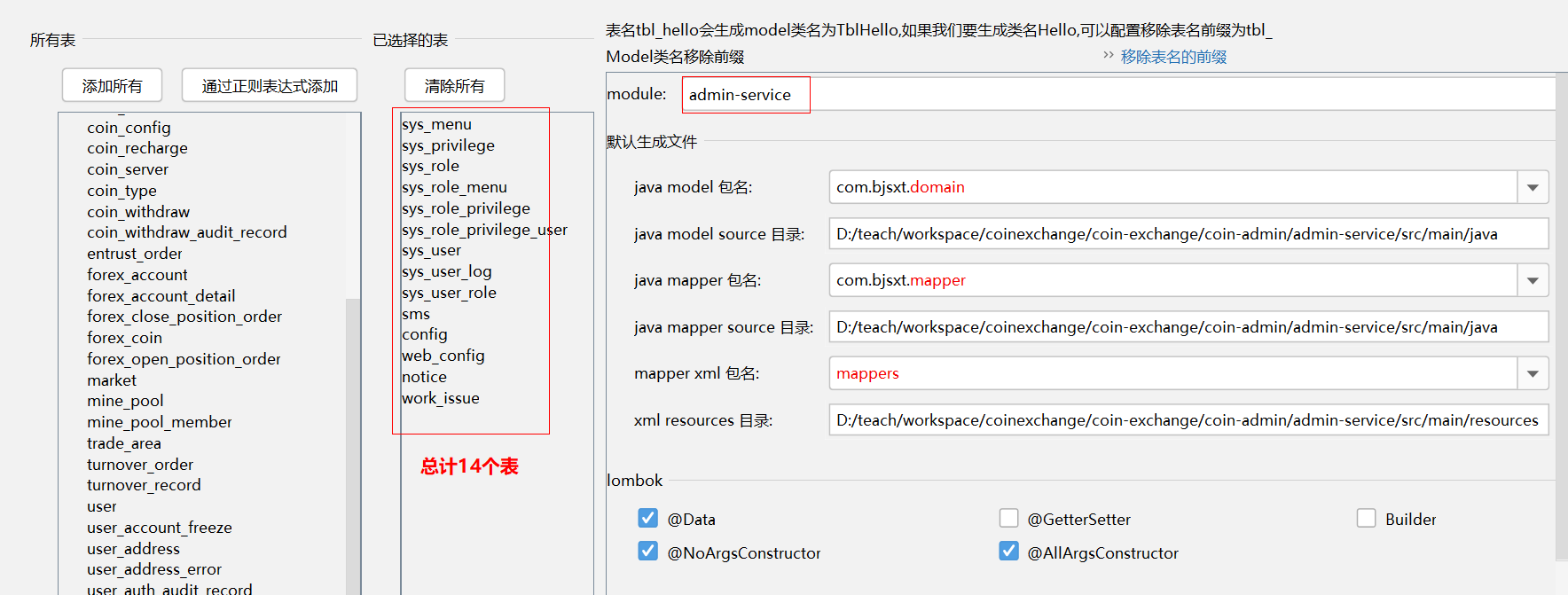
在schemas里面，我们可以看见所有的表结构：



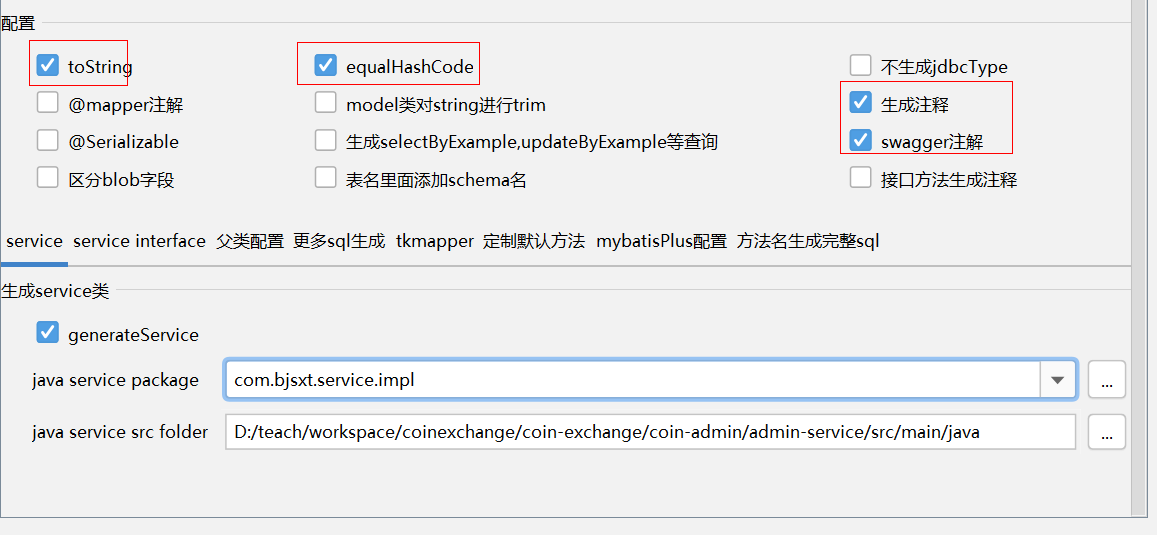
## 4.3 代码生成



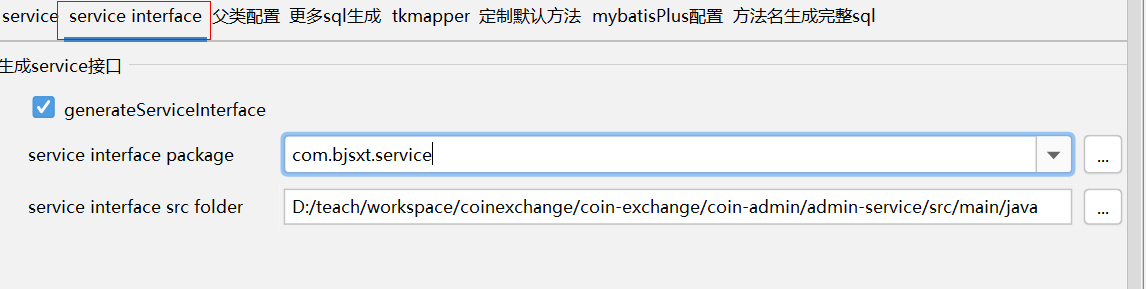
1. 配置信息-1：



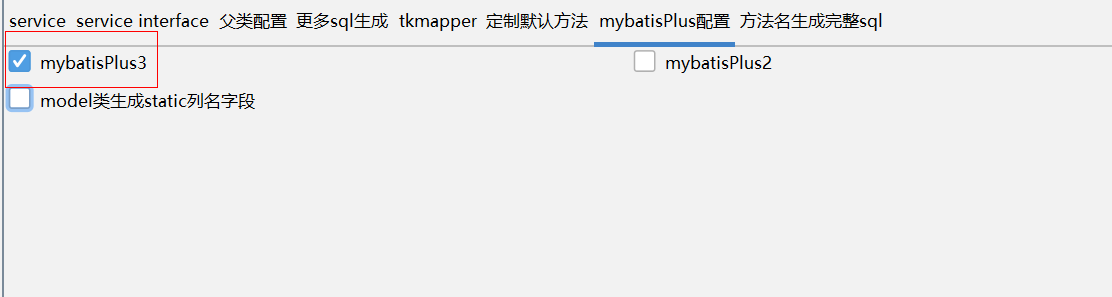
1. 配置信息-2：



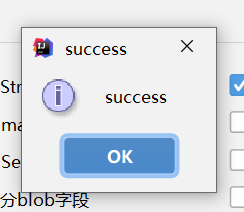
1. 配置信息-3：



1. 配置信息-4：

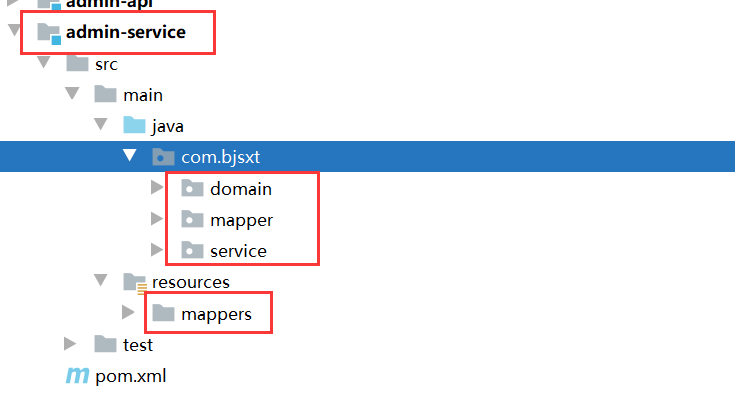


确认无误后，点击OK:



出现success 代表生成成功。

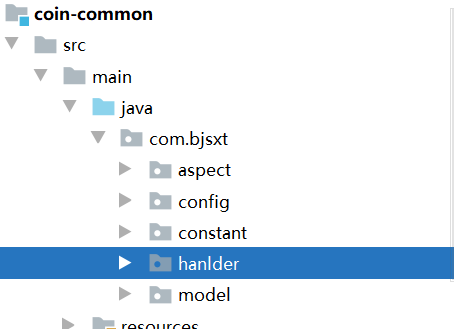
打开admin-service 这个项目，发现代码都已经生成完毕了



# 运行admin-service

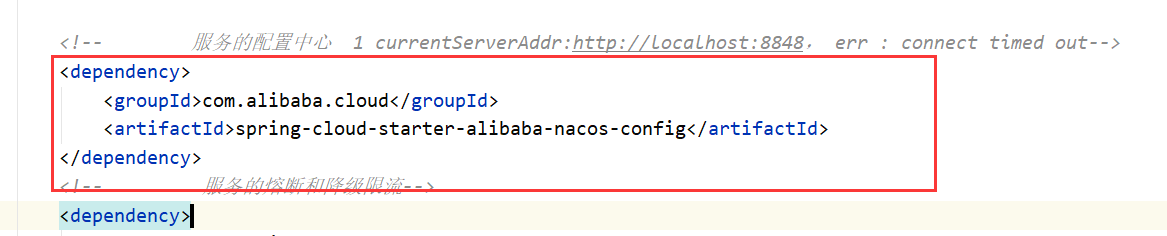
## 5.1 删除之前在coin-common里面的测试代码

删除测试时使用过的，剩下的部分为：



## 5.2 启用配置文件中心的功能

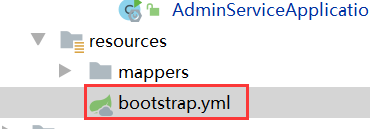
在coin-common里面的pom.xml 文件里面打开：



## 5.3 在admin-service里面添加启动类

|  |
| --- |
| */\*\*  \* admin-service 启动类  \*/* @SpringBootApplication @EnableDiscoveryClient public class AdminServiceApplication {   public static void main(String[] args) {  SpringApplication.*run*(AdminServiceApplication.class ,args) ;  } } |

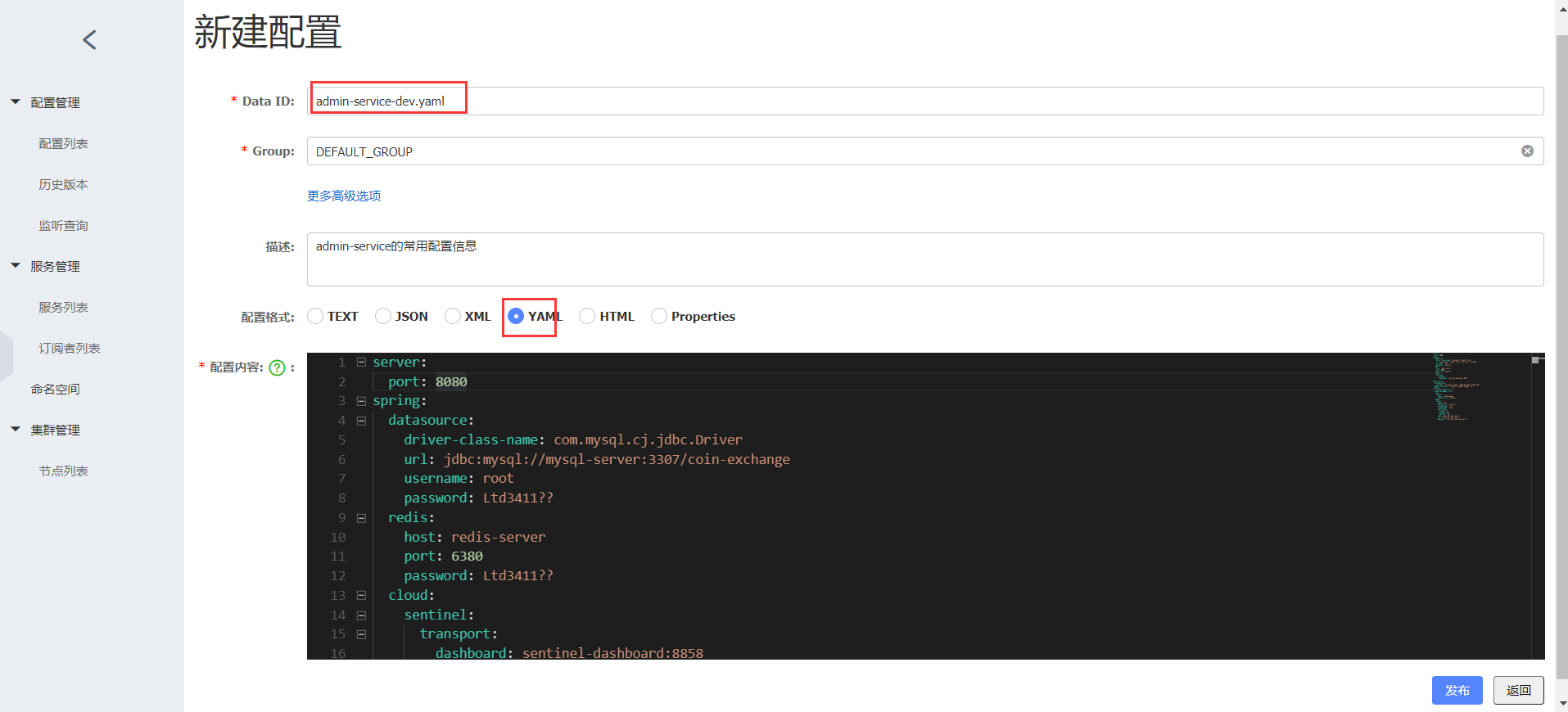
## 5.4 添加一个配置文件(bootstrap.yml)



内容为：

|  |
| --- |
| spring:  application: *# 应用名称* name: admin-service  profiles: *#切换到开发环境上面* active: dev  cloud:  nacos: *#配置Nacos的地址* server-addr: nacos-server:8848  config: *# 配置文件的扩展名* file-extension: yaml |

## 5.5 在Nacos-Dashboard 里面发布一份配置文件



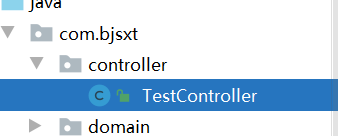
配置的内容如下：

|  |
| --- |
| server:  port: 8080  spring:  datasource:  driver-class-name: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://mysql-server:3307/coin-exchange  username: root  password: Ltd3411??  redis:  host: redis-server  port: 6380  password: Ltd3411??  cloud:  sentinel:  transport:  dashboard: sentinel-dashboard:8858  mybatis-plus:  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  mapper-locations: classpath:/mappers/\*Mapper.xml  jetcache:  statIntervalMinutes: 15  areaInCacheName: false  local:  default:  type: linkedhashmap  keyConvertor: fastjson  remote:  default:  type: redis  keyConvertor: fastjson  valueEncoder: kryo  valueDecoder: kryo  poolConfig:  minIdle: 5  maxIdle: 20  maxTotal: 50  host: ${spring.redis.host}  port: ${spring.redis.port}  password: ${spring.redis.password}  swagger2:  basePackage: com.bjsxt.controller  name: liangtiandong  url: www.liangtiandong.com  email: liangtiandong@live.com  title: 后台管理系统API接口  description: 后台管理系统API接口演示  version: 1.0  termsOfServiceUrl: www.bjsxt.com |

新建完成后点击发布即可。



## 5.6 添加一个接口测试环境

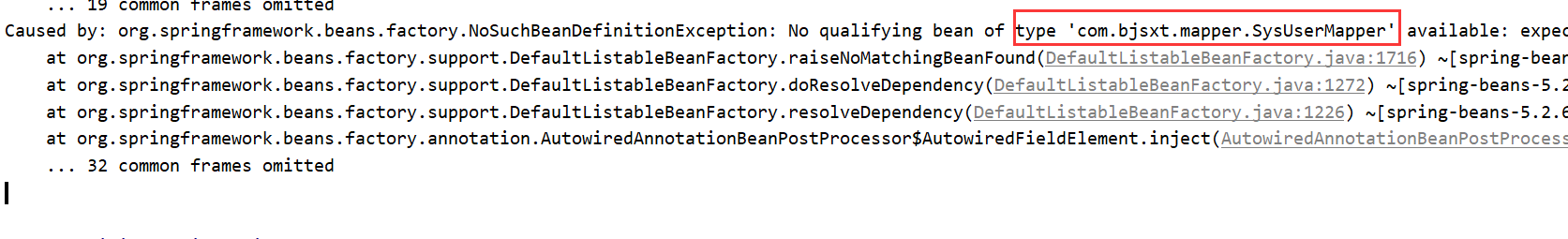


内容为：

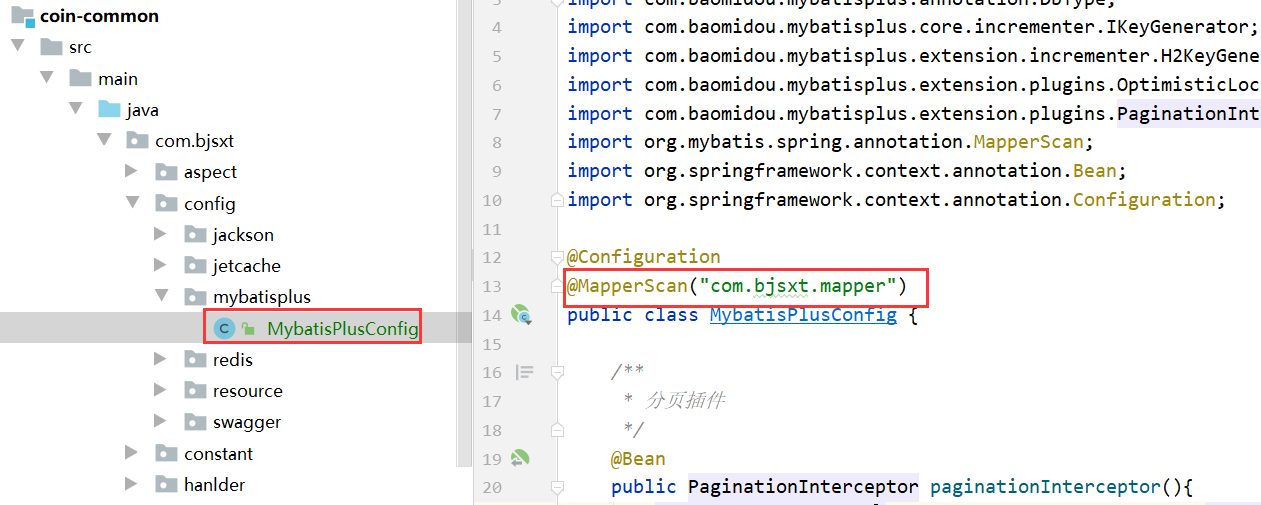
|  |
| --- |
| @RestController @Api(tags = "admin-service-test接口") public class TestController {   @Autowired  private SysUserService sysUserService ;   @GetMapping("/user/info/{id}")  @ApiOperation(value = "使用用户的id查询用户",authorizations = {@Authorization("Authorization")})  @ApiImplicitParams({  @ApiImplicitParam(name = "id",value = "用户的id值")  })  public R<SysUser> sysUser(@PathVariable("id")Long id){  SysUser sysUser = sysUserService.getById(id);  return R.*ok*(sysUser) ;  }  } |

## 5.7 启动测试

错误解析-1：

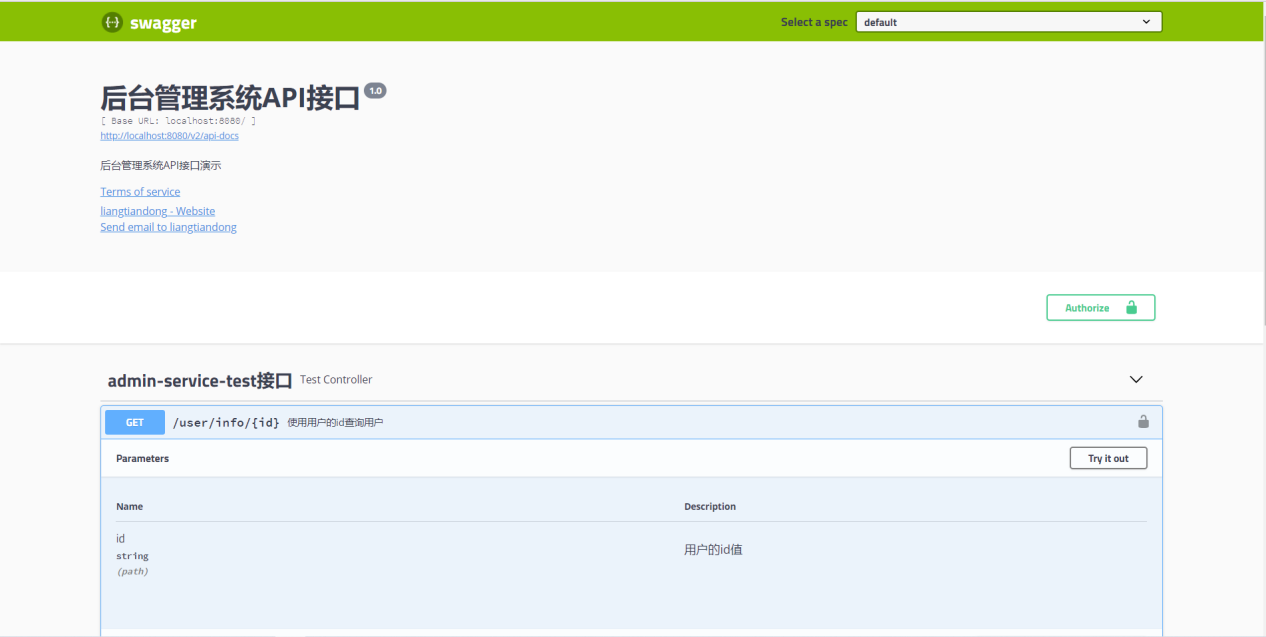


提示说IOC容器里面没有SysUserMapper该对象，原因在于我们在MybatisPlusConfig里面少添加一个注解：



添加后，就没有问题了。

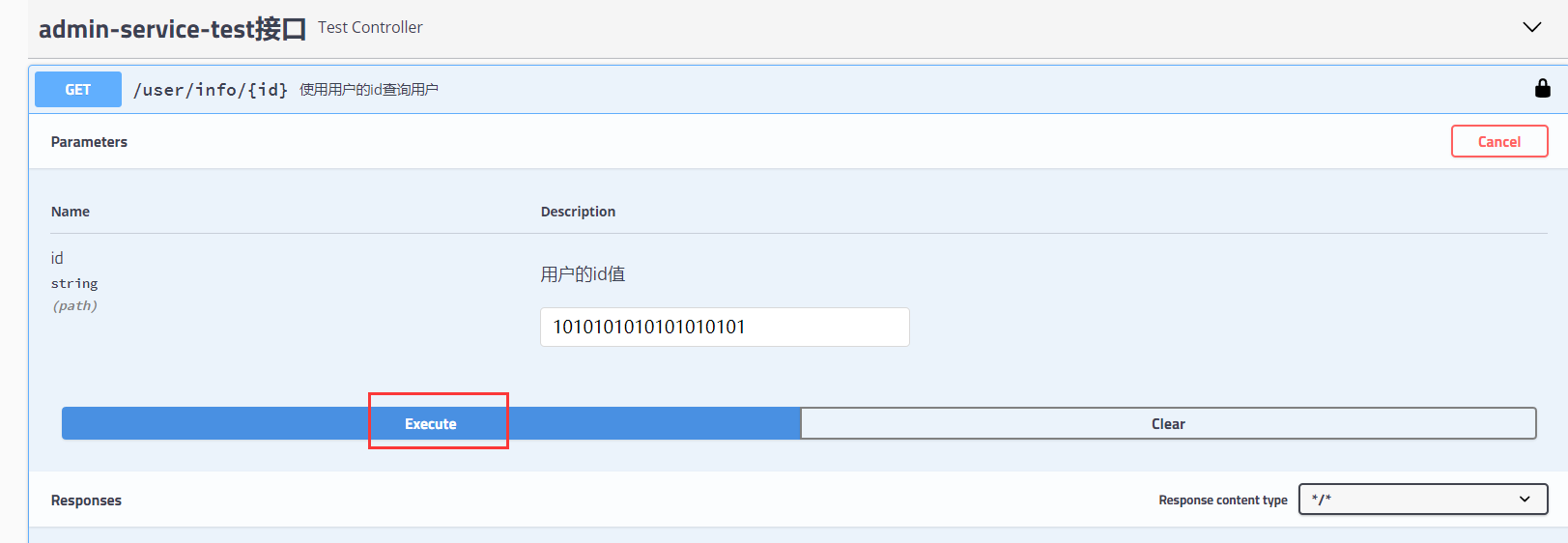
## 5.8 测试访问



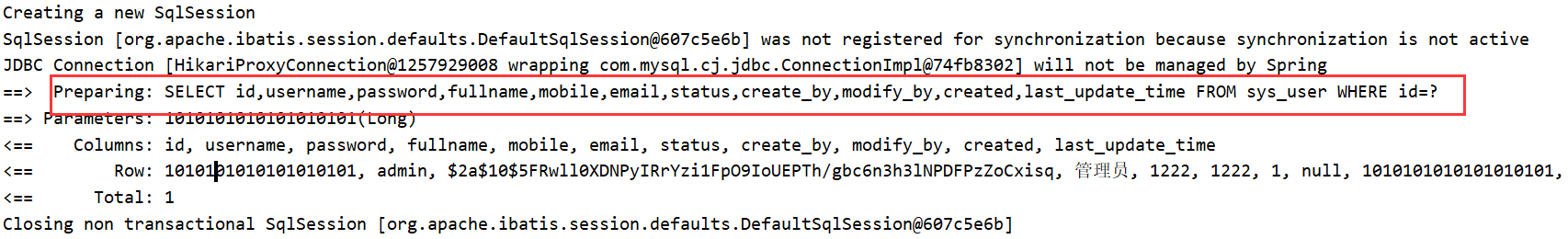
使用id为：

|  |
| --- |
| 1010101010101010101 |

访问：



后台日志-1：



后台日志-2：

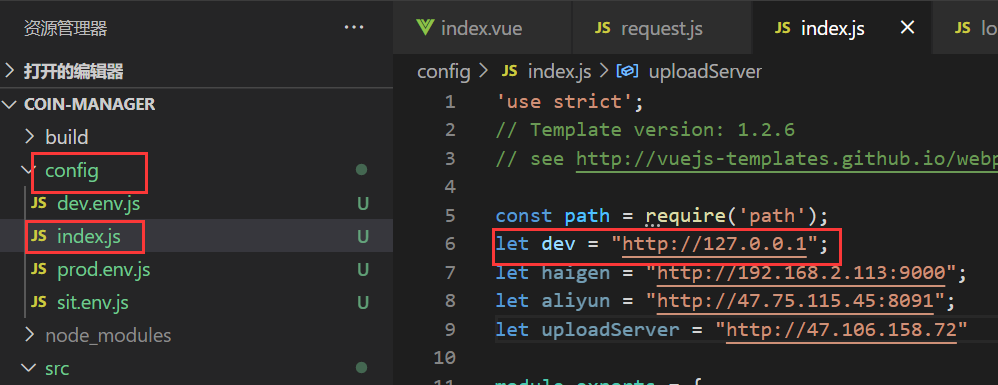


没有任何问题

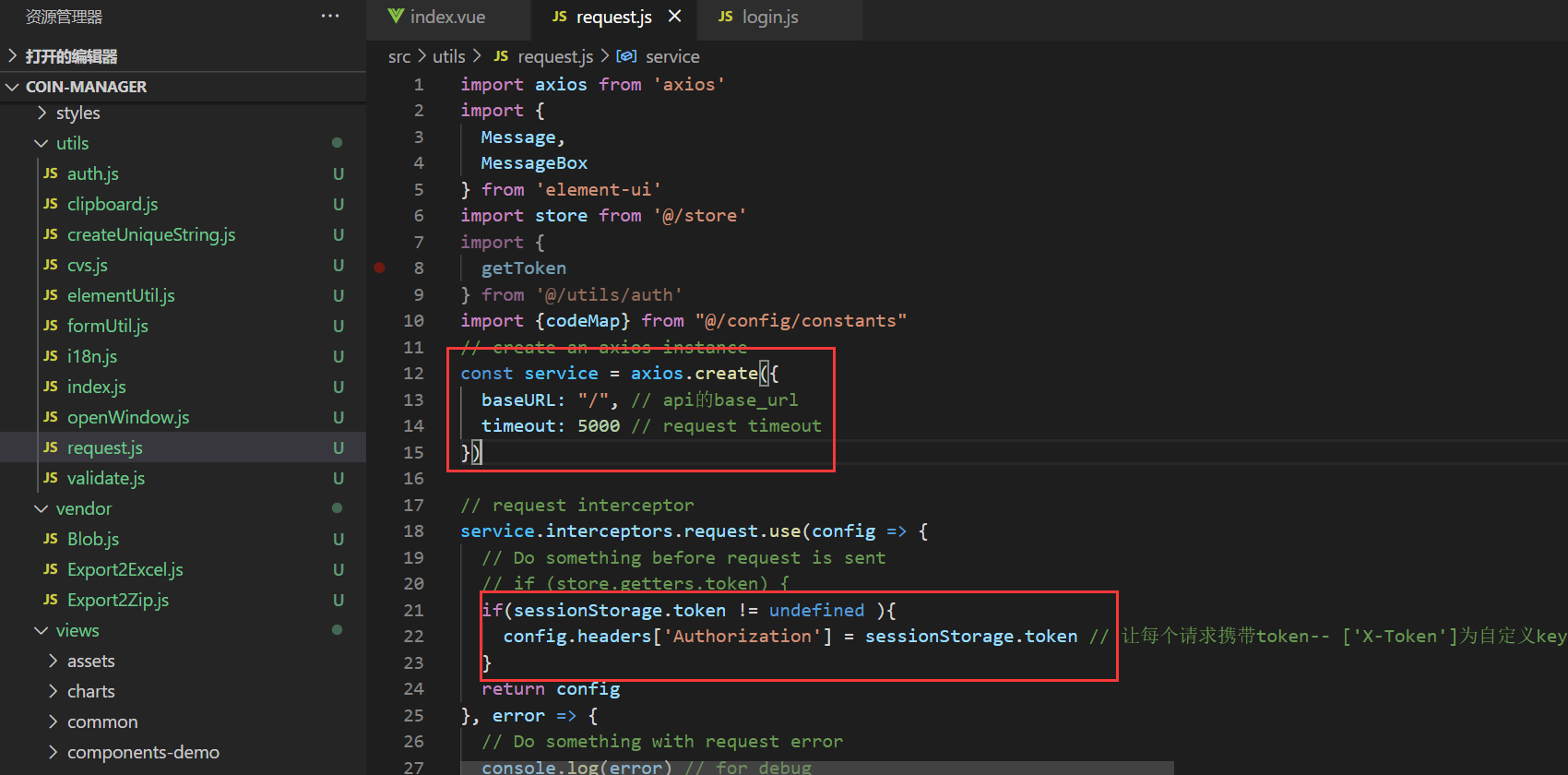
# 登录的实现

## 6.1 前端登录代码（不关注）

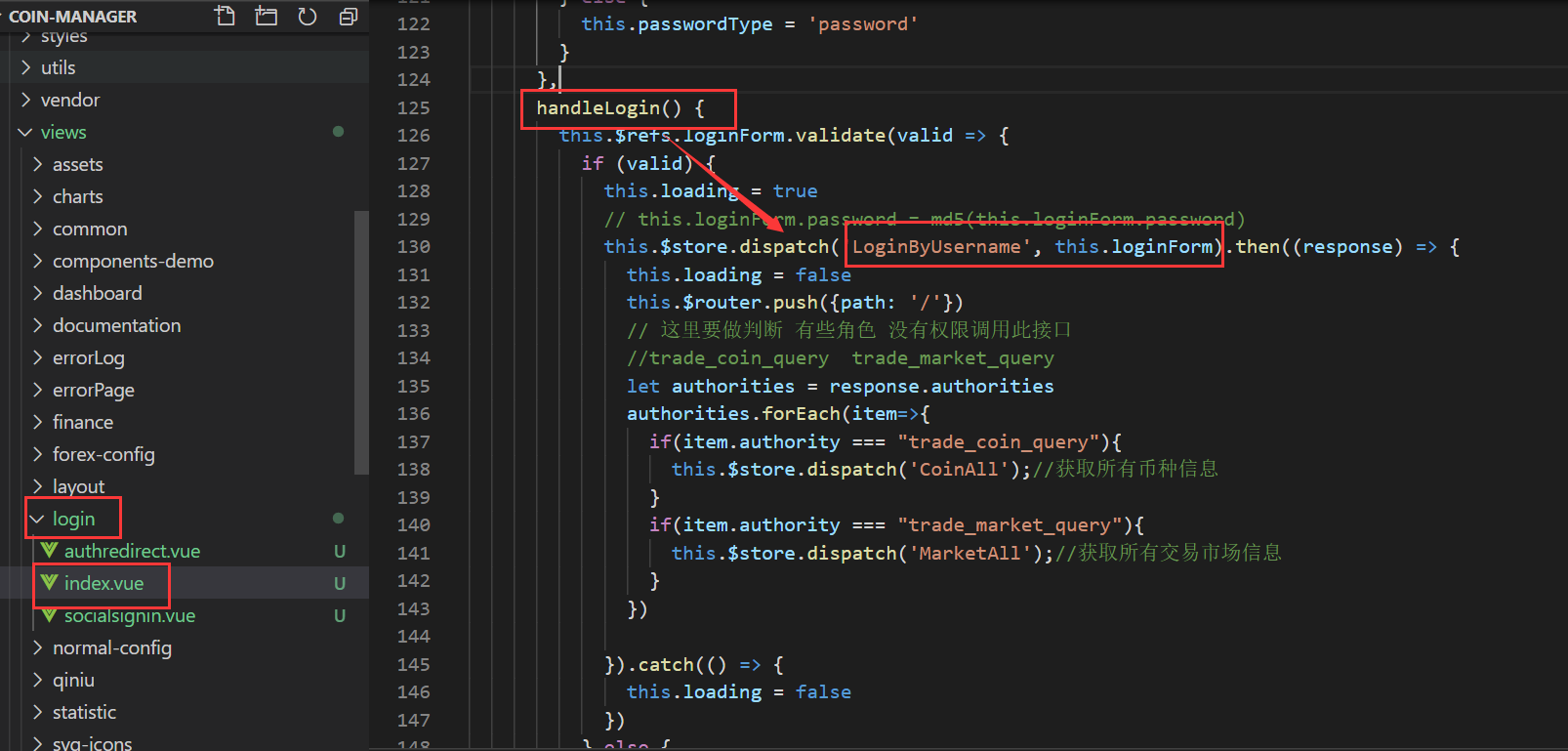
### 6.1.1 前端config



### 6.1.2 request.js



### 6.1.3请求接口



请求参数就是loginForm: 其实就是用户名和密码

|  |
| --- |
| loginForm: {            username: '',            password: ''          }, |

我们来看;LoginByUsername方法的实现：



loginApi的实现：



现在我们知道了：

请求的url: /admin/login

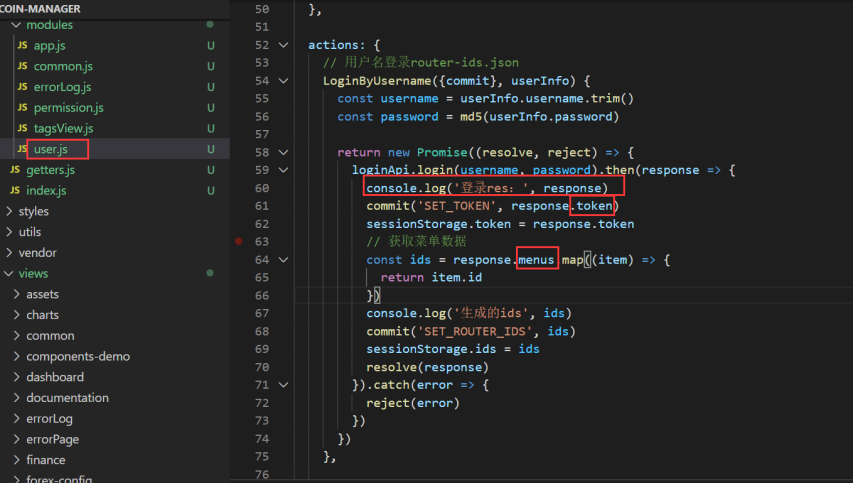
请求方式：post

请求参数：表单形式（非json）

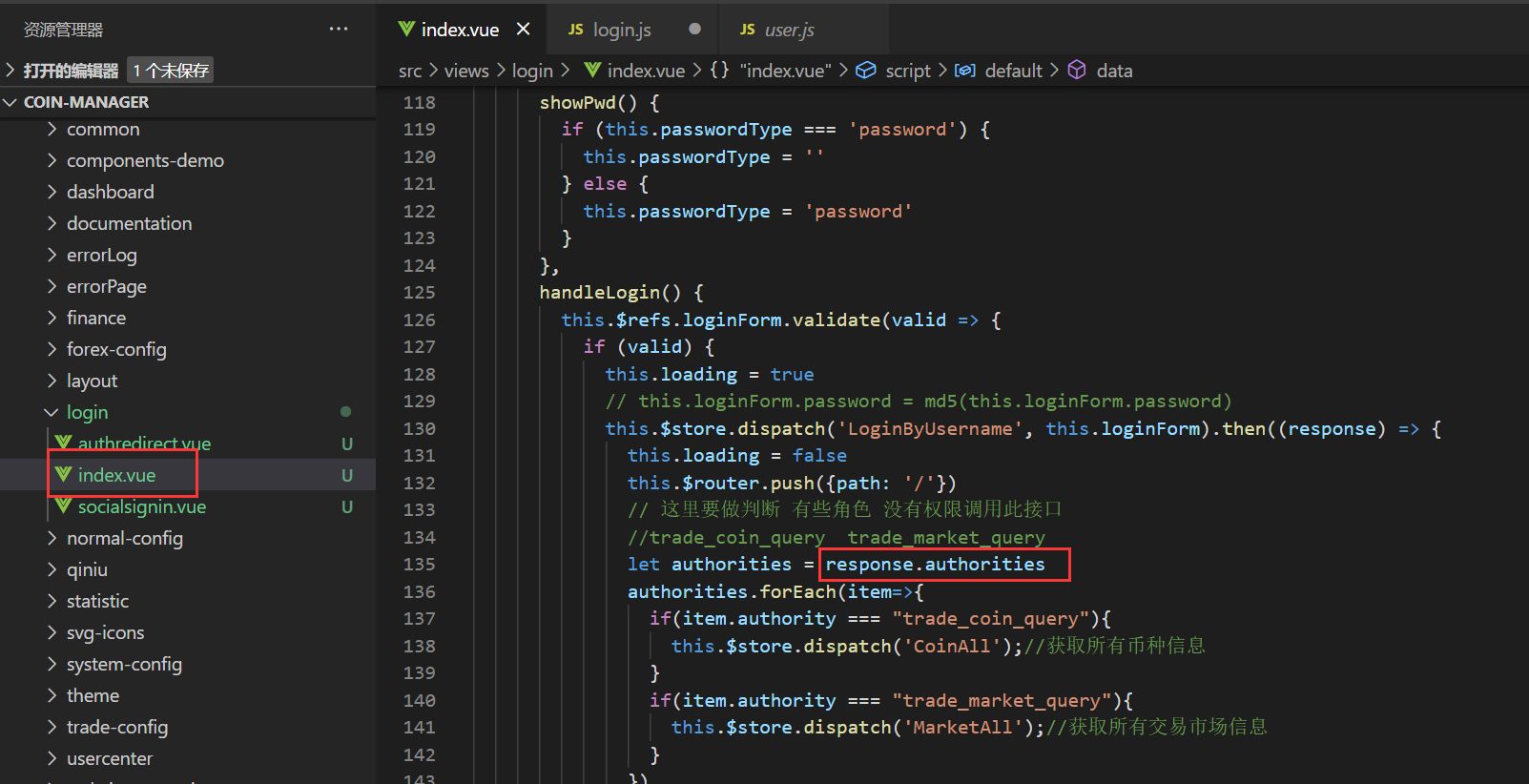
|  |
| --- |
| username=xxx  Password=xxx |

### 6.1.4 后端返回值

在store/user.js 里面：



在login/index.vue 里面



总的来说，返回值模型里面需要包含以下属性：

|  |
| --- |
| {  "token":"", 登录成功的Token  "menus":[], 返回的菜单数据，不需要树形结构  "authorities":[]，返回的权限数据  } |

## 6.2 后台代码实现

### 6.2.1 使用网关

#### 网关的配置文件：

我们知道，在前端的js里面，访问的服务器地址为：

|  |
| --- |
| http://localhost/ |

所以我们需要让网关运行在80端口。

我们使用/admin来路由到后台管理系统，因此我们还需要添加路由规则

当前台访问/admin/\*\* 时，将访问路由到**后台管理系统**里面

最后的配置文件如下：



|  |
| --- |
| server:  port: 80 spring:  application:  name: gateway-server  cloud:  nacos:  discovery:  server-addr: nacos-server:8848 *# 修改本机的host 文件* gateway:  discovery:  locator:  enabled: true  lower-case-service-id: true *# admin-service ADMIN-SERVICE /admin-service/\*\* -> 微服务（ADMIN-SERVICE）* routes:  - id: admin-service\_router  uri: lb://admin-service *# 转发到那个目的地* predicates:  - Path=/admin/\*\*  filters:  - StripPrefix=1  - id: test\_router  uri: http://www.aliyun.com  predicates:  - Path=/product  *# /test->http://www.baidu.com 网关自己内部会构建请求去访问我们要访问的地址 GET:http://www.baidu.com/test  # 通过网关来完成对2 种维度的限流： 网关维度 + API 分组* sentinel:  transport:  dashboard: sentinel-server:8858 *# sentinel-dashboard 放在ecs 里面* datasource: *# ds1.file: # file: classpath:gw-flow.json # 网关 + API分组的限流 # ruleType: gw\_flow* ds1.nacos: *#com.alibaba.csp.sentinel.datasource.nacos.NacosDataSource 使用Nacos持久化我的sentinel 数据时，需要添加nacos-datasource的依赖* serverAddr: nacos-server:8848  dataId: gw-flow  ruleType: gw\_flow *# # ds2.file: # file: classpath:api-group.json # ruleType: gw\_api\_group* ds2.nacos:  serverAddr: nacos-server:8848  dataId: api-group  ruleType: gw\_api\_group *# 我们演示了sentinel-dashboard 的规则的定义，而且规则定义好了后，我们的网关能立马的感知到（生效）（nacos无法感知），但是我们下次重启，会丢失规则。--》nacos* redis:  host: redis-server  port: 6380  password: Ltd3411?? |

#### JwtCheckFilter放行的地址：

默认放行/admin/login 了。

### 6.2.2 LoginResult

|  |
| --- |
| */\*\*  \* 登录的返回值  \*/* @Data @AllArgsConstructor @NoArgsConstructor public class LoginResult {   */\*\*  \* 登录产生的token  \*/* private String token ;   */\*\*  \* 前端的菜单数据  \*/* private List<SysMenu> menus ;   */\*\*  \* 权限数据  \*/* private List<SimpleGrantedAuthority> authorities ;  } |

### 6.2.3 SysLoginController

SysLoginController

|  |
| --- |
| */\*\*  \* 登录的控制器  \*/* @RestController @Api(tags = "登录的控制器") public class SysLoginController {   @Autowired  private SysLoginService loginService;   @PostMapping("/login")  @ApiOperation(value = "后台管理人员登录")  @ApiImplicitParams(  {  @ApiImplicitParam(name = "username", value = "用户名称"),  @ApiImplicitParam(name = "password", value = "用户的密码"),  }  )  public LoginResult login(  @RequestParam(required = true) String username, *// 用户名称* @RequestParam(required = true) String password *// 用户的密码* ) {  return loginService.login(username, password);  }  } |

### 6.2.4 SysLoginService

SysLoginService

|  |
| --- |
| */\*\*  \* 登录的接口  \*/* public interface SysLoginService {   */\*\*  \* 登录的实现  \* @param username  \* 用户名  \* @param password  \*  \* 用户的密码  \* @return  \* 登录的结果  \*/* LoginResult login(String username ,String password) ; } |

### 6.2.5 SysLoginServiceImpl

|  |
| --- |
| @Service @Slf4j public class SysLoginServiceImpl implements SysLoginService {   @Autowired  private OAuth2FeignClient oAuth2FeignClient;   @Autowired  private SysMenuService sysMenuService ;   @Value("${basic.token:Basic Y29pbi1hcGk6Y29pbi1zZWNyZXQ=}")  private String basicToken ;  */\*\*  \* 登录的实现  \*  \* @param username 用户名  \* @param password 用户的密码  \* @return 登录的结果  \*/* @Override  public LoginResult login(String username, String password) {  *log*.info("用户{}开始登录", username);  *// 1 获取token 需要远程调用authorization-server 该服务* ResponseEntity<JwtToken> tokenResponseEntity = oAuth2FeignClient.getToken("password", username, password, "admin\_type", basicToken);  if(tokenResponseEntity.getStatusCode()!= HttpStatus.*OK*){  throw new ApiException(ApiErrorCode.*FAILED*) ;  }  JwtToken jwtToken = tokenResponseEntity.getBody();  *log*.info("远程调用授权服务器成功,获取的token为{}", JSON.*toJSONString*(jwtToken,true));  String token = jwtToken.getAccessToken() ;   *// 2 查询我们的菜单数据* Jwt jwt = JwtHelper.*decode*(token);  String jwtJsonStr = jwt.getClaims();  JSONObject jwtJson = JSON.*parseObject*(jwtJsonStr);  Long userId = Long.*valueOf*(jwtJson.getString("user\_name")) ;  List<SysMenu> menus = sysMenuService.getMenusByUserId(userId);   *// 3 权限数据怎么查询 -- 不需要查询的，因为我们的jwt 里面已经包含了* JSONArray authoritiesJsonArray = jwtJson.getJSONArray("authorities");  List<SimpleGrantedAuthority> authorities = authoritiesJsonArray.stream() *// 组装我们的权限数据* .map(authorityJson->new SimpleGrantedAuthority(authorityJson.toString()))  .collect(Collectors.*toList*());  return new LoginResult(token, menus, authorities);  } } |

### 6.2.6 feign

#### JwtToken：

|  |
| --- |
| @Data public class JwtToken {  */\*\*  \* access\_token  \*/* @JsonProperty("access\_token")  private String accessToken;   */\*\*  \* token类型  \*/* @JsonProperty("token\_type")  private String tokenType;   */\*\*  \* refresh\_token  \*/* @JsonProperty("refresh\_token")  private String refreshToken;   */\*\*  \* 过期时间  \*/* @JsonProperty("expires\_in")  private Long expiresIn;    */\*\*  \* token的范围  \*/* private String scope;   */\*\*  \* 颁发的凭证  \*/* private String jti; } |

#### OAuth2FeignClient：

|  |
| --- |
| @FeignClient(value = "authorization-server") public interface OAuth2FeignClient {   @PostMapping("/oauth/token")  ResponseEntity<JwtToken> getToken(  @RequestParam("grant\_type") String grantType , *// 授权类型* @RequestParam("username") String username , *// 用户名* @RequestParam("password") String password , *// 用户的密码* @RequestParam("login\_type") String loginType, *// 登录的类型* @RequestHeader("Authorization") String basicToken *// Basic Y29pbi1hcGk6Y29pbi1zZWNyZXQ= 由第三方客户端信息加密出现的值* ) ; } |

### 6.2.7 SysMenuService

|  |
| --- |
| public interface SysMenuService extends IService<SysMenu>{   */\*\*  \* 通过用户的id 查询用户的菜单数据  \* @param userId  \* @return  \*/* List<SysMenu> getMenusByUserId(Long userId); } |

### 6.2.8 SysMenuServiceImpl

|  |
| --- |
| @Service public class SysMenuServiceImpl extends ServiceImpl<SysMenuMapper, SysMenu> implements SysMenuService{   @Autowired  private SysRoleService sysRoleService ;   @Autowired  private SysMenuMapper sysMenuMapper ;  */\*\*  \* 通过用户的id 查询用户的菜单数据  \*  \* @param userId  \* @return  \*/* @Override  public List<SysMenu> getMenusByUserId(Long userId) {  *// 1 如果该用户是超级管理员->>拥有所有的菜单* if(sysRoleService.isSuperAdmin(userId)){  return list() ; *// 查询所有* }  *// 2 如果该用户不是超级管理员->>查询角色->查询菜单* return sysMenuMapper.selectMenusByUserId(userId);  } } |

### 6.2.9 SysRoleService

|  |
| --- |
| public interface SysRoleService extends IService<SysRole>{    */\*\*  \* 判断一个用户是否为超级的管理员  \* @param userId  \* @return  \*/* boolean isSuperAdmin(Long userId); } |

### 6.2.10 SysRoleServiceImpl

|  |
| --- |
| @Service public class SysRoleServiceImpl extends ServiceImpl<SysRoleMapper, SysRole> implements SysRoleService {   @Autowired  private SysRoleMapper sysRoleMapper;   */\*\*  \* 判断一个用户是否为超级的管理员  \*  \* @param userId  \* @return  \*/* @Override  public boolean isSuperAdmin(Long userId) {  *// 当用户的角色code 为：ROLE\_ADMIN 时，该用户为超级的管理员  // 用户的id->用户的角色->该角色的Code是否为ROLE\_ADMIN* String roleCode = sysRoleMapper.getUserRoleCode(userId);  if (!StringUtils.*isEmpty*(roleCode) && roleCode.equals("ROLE\_ADMIN")) {  return true;  }  return false;  } } |

### 6.2.11 SysRoleMapper

|  |
| --- |
| public interface SysRoleMapper extends BaseMapper<SysRole> {  */\*\*  \* 获取用户角色Code的实现  \* @param userId  \* @return  \*/* String getUserRoleCode(Long userId); } |

### 6.2.12 SysRoleMapper.xml

添加的sql 语句：

|  |
| --- |
| <select id="getUserRoleCode" resultType="java.lang.String">  SELECT  r.`code`  FROM  sys\_role AS r LEFT JOIN sys\_user\_role as ur on r.id = ur.role\_id  WHERE  ur.user\_id =#{userId} </select> |

### 6.2.13 SysMenuMapper

|  |
| --- |
| public interface SysMenuMapper extends BaseMapper<SysMenu> {   */\*\*  \* 通过用户的id 查询用户的菜单数据  \* @param userId  \* @return  \*/* List<SysMenu> selectMenusByUserId(Long userId); } |

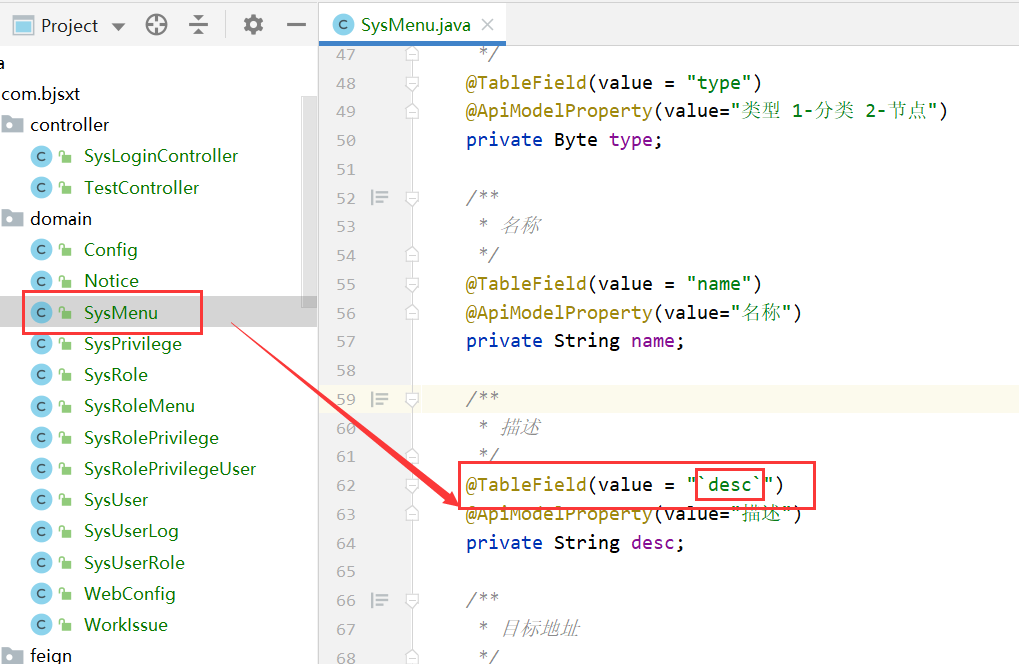
### 6.2.13 SysMenuMapper.xml

添加的sql：

|  |
| --- |
| <select id="selectMenusByUserId" resultMap="BaseResultMap">  SELECT  m.id,  m.parent\_id,  m.parent\_key,  m.`type`,  m.`name`,  m.`desc`,  m.target\_url,  m.sort,  m.`status`,  m.create\_by,  m.modify\_by,  m.created,  m.last\_update\_time  FROM sys\_menu as m  LEFT JOIN sys\_role\_menu AS rm ON rm.menu\_id = m.id  LEFT JOIN sys\_user\_role AS ur ON ur.role\_id = rm.role\_id  WHERE  ur.user\_id = #{userId} </select> |

### 6.2.14 SysMenu小细节

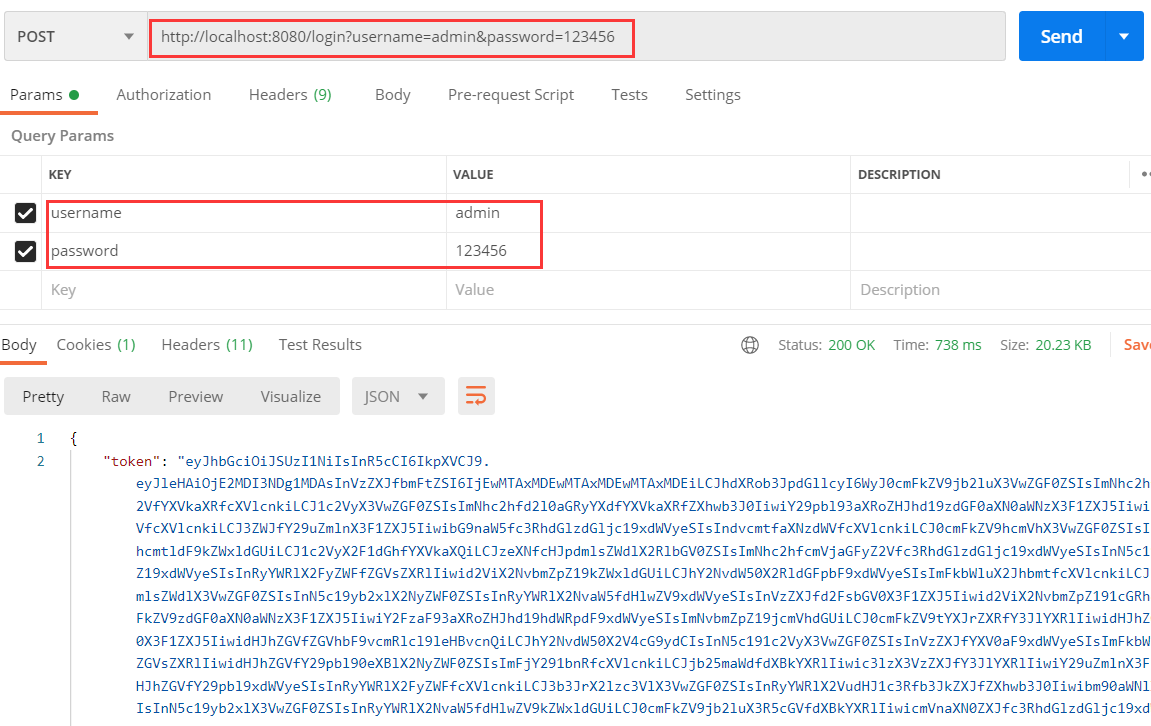
**注意：desc是Mysql 里面的关键字，必须是转义符，否则报错。**

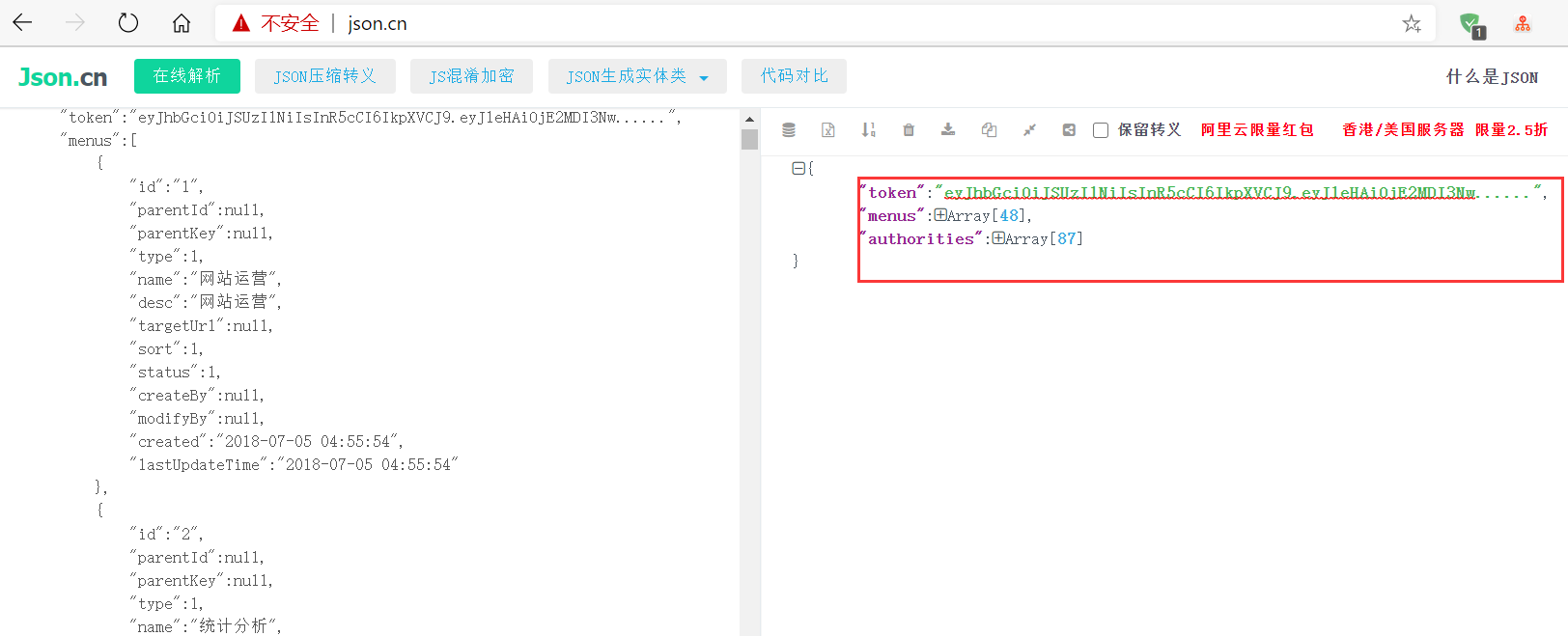


### 6.2.15 coin-common里面的ResourceServerConfig

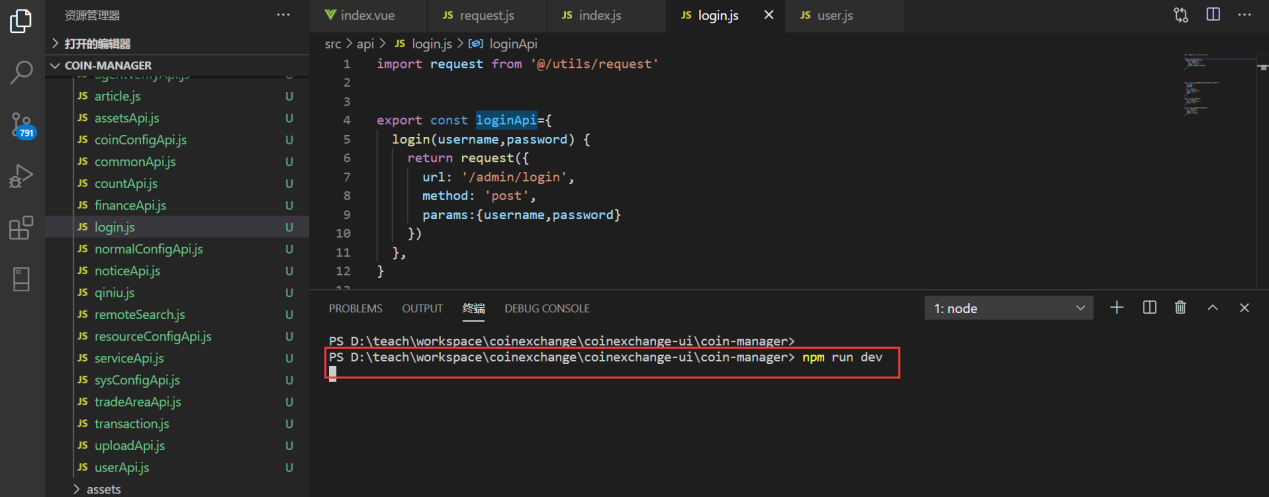


## 6.3 登录的测试





## 6.4 启动前端测试

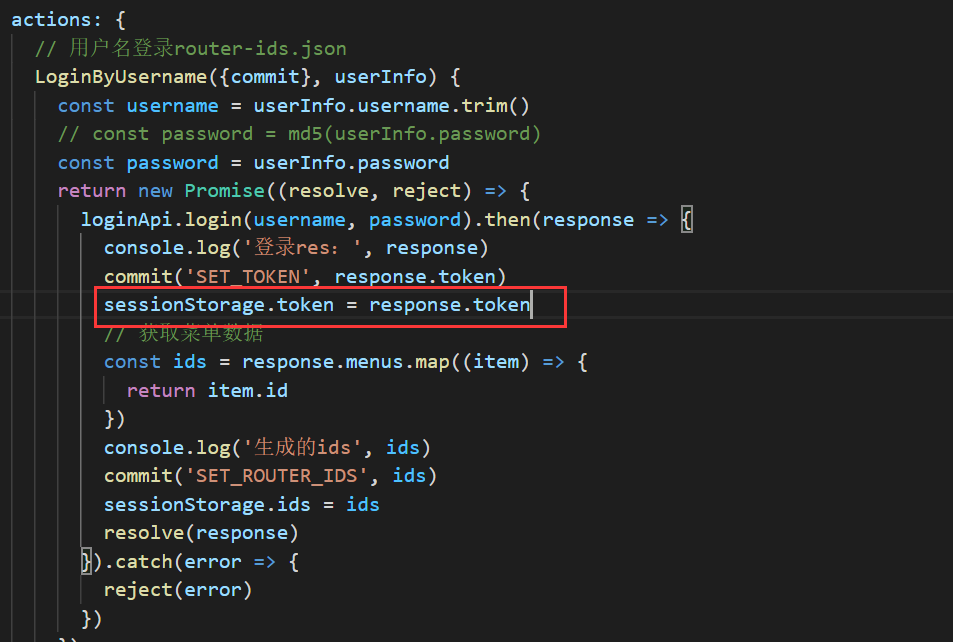


登录测试：

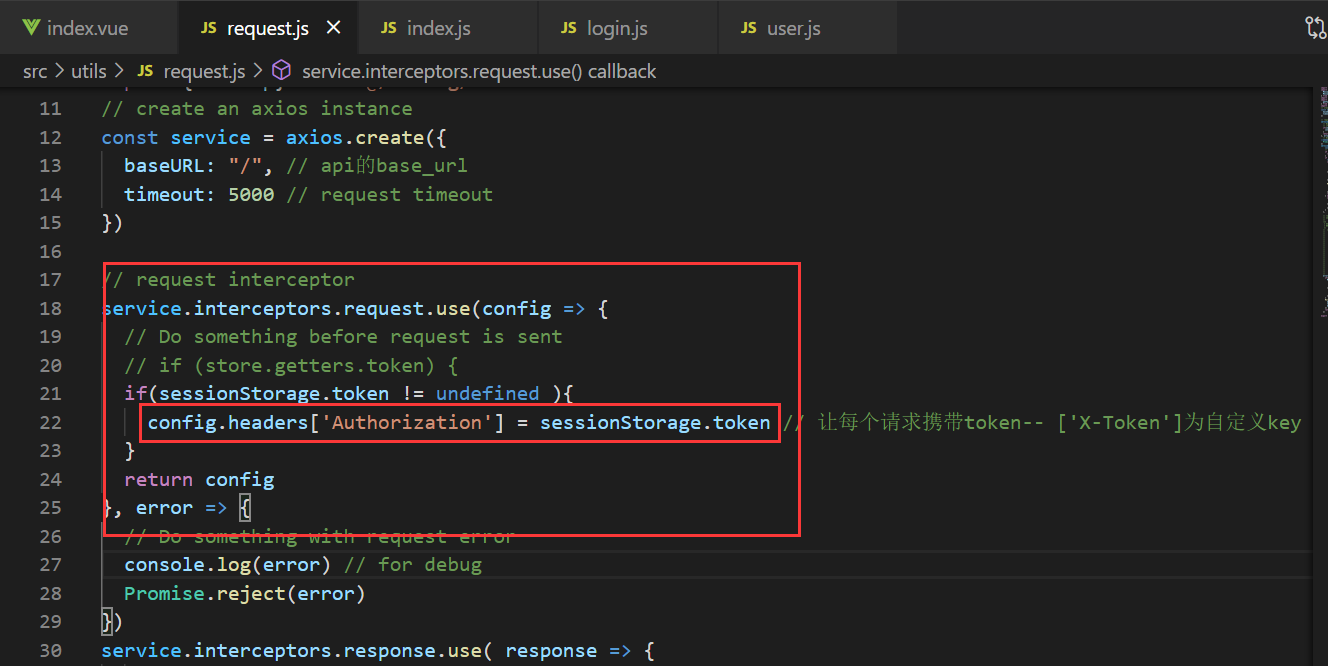


## 6.5 登录成功后数据的交互过程

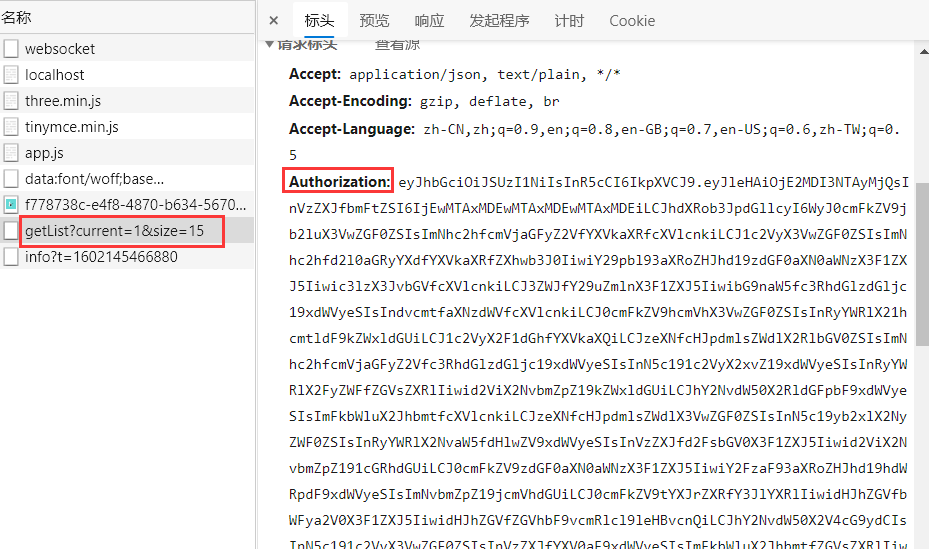
store/user.js



request.js



这样，就能保证我们登录成功后，给后端发的每一个请求都携带了Authorization这个请求头：



但是，聪明的你是否发现问题了呢?

我们的Token 里面没有:

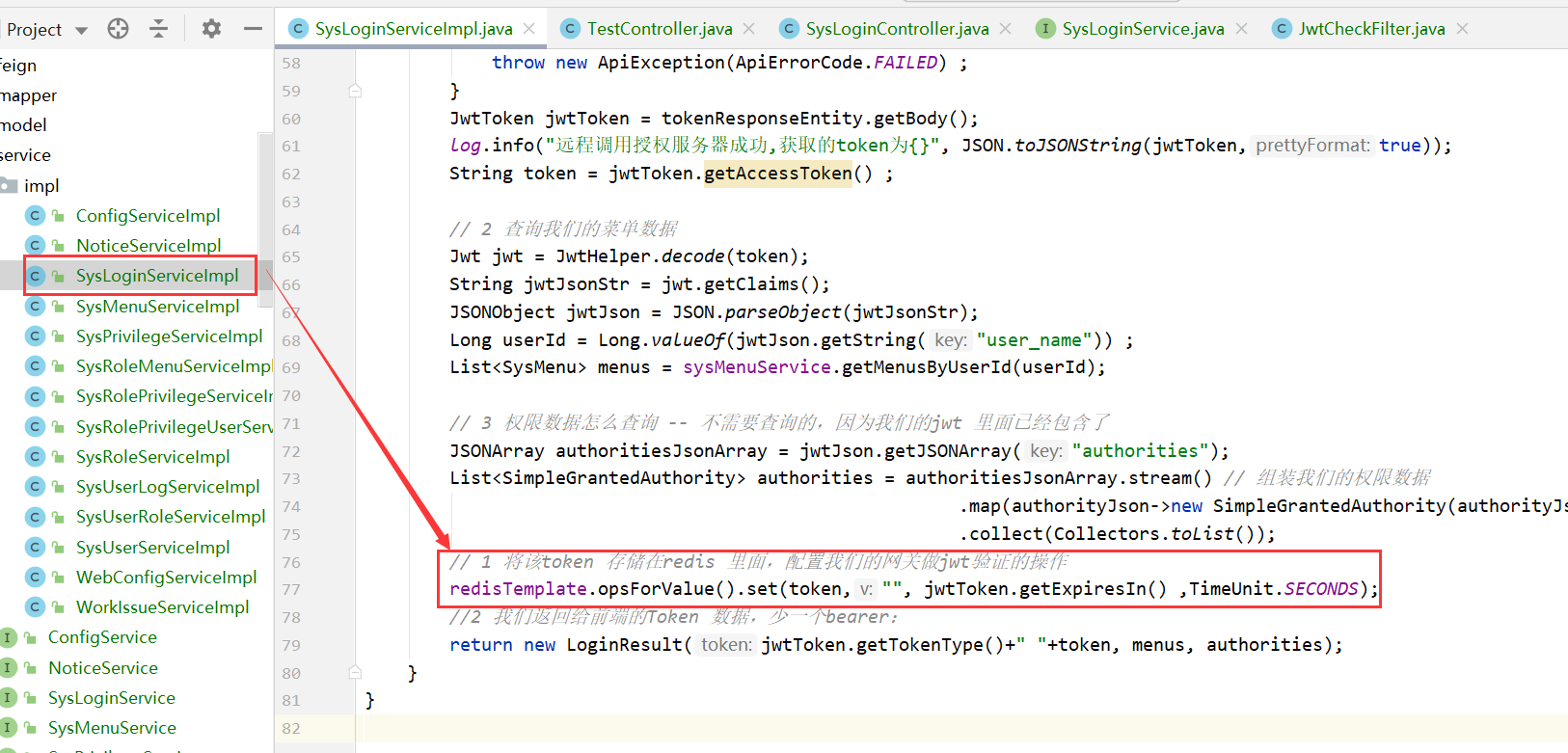
|  |
| --- |
| bearer |

## 6.6 JWT登出问题

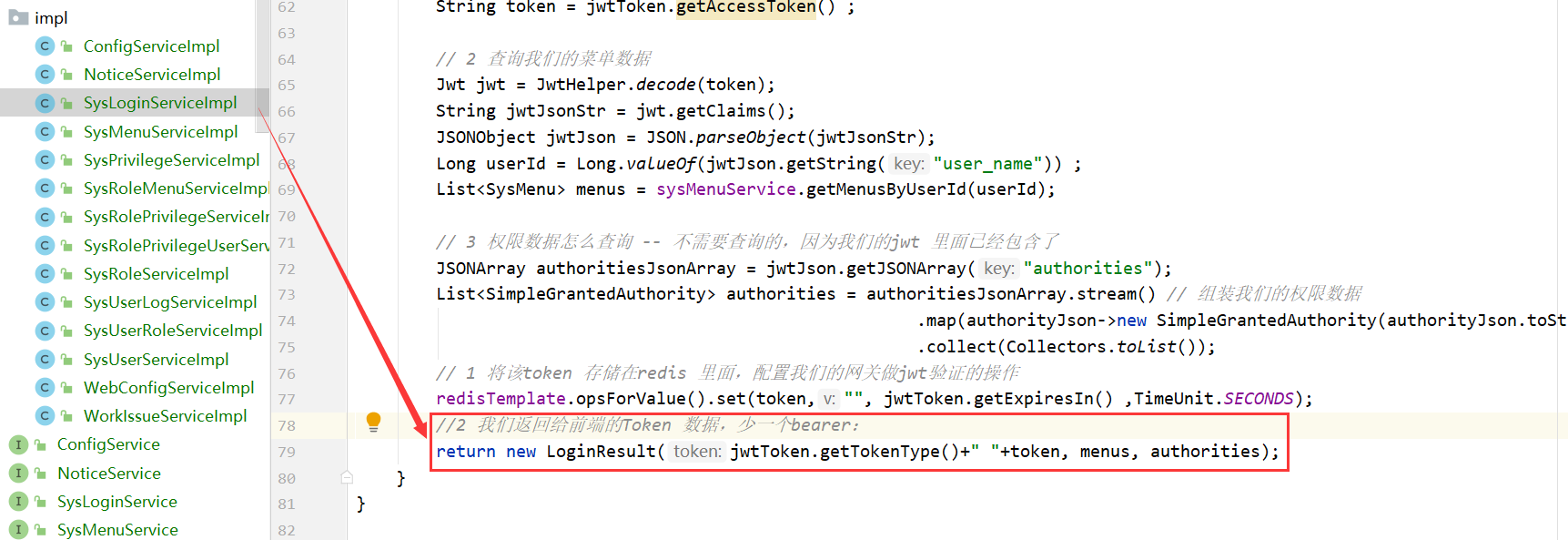
在登录成功后，没有将token保存在redis 里面，这样用户还是无法通过网关访问数据的。

## 6.7 SysLoginServiceImpl持续的改进

### 6.7.1 存储token到redis里面

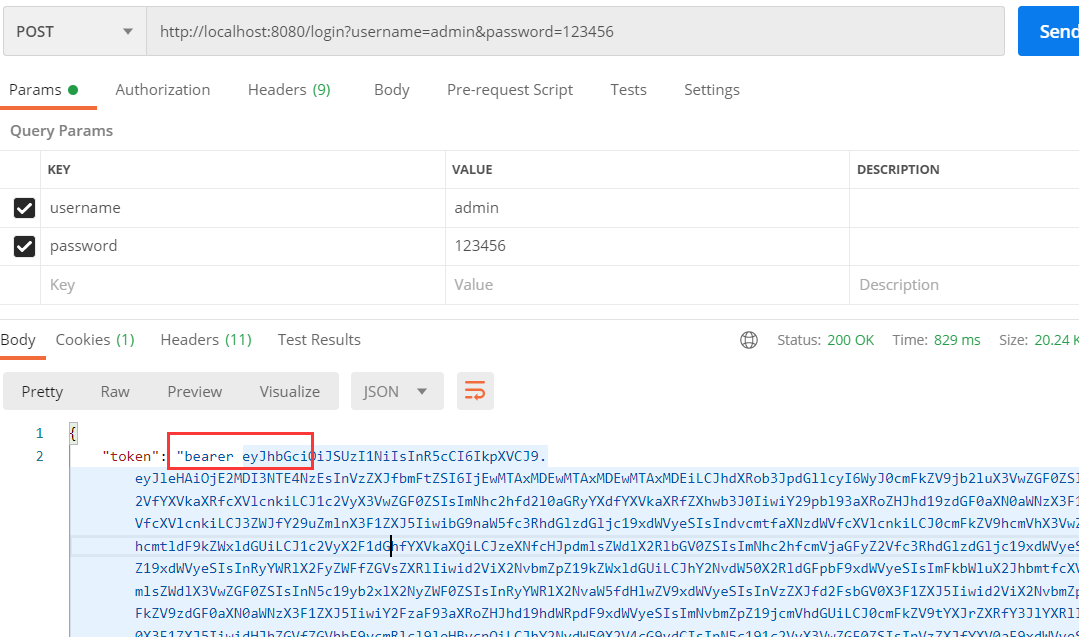


### 6.7.2 返回给前端的token添加bearer 类型

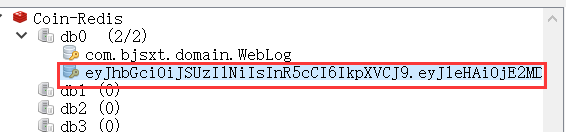


### 6.7.3 重启登录测试

登录的结果：

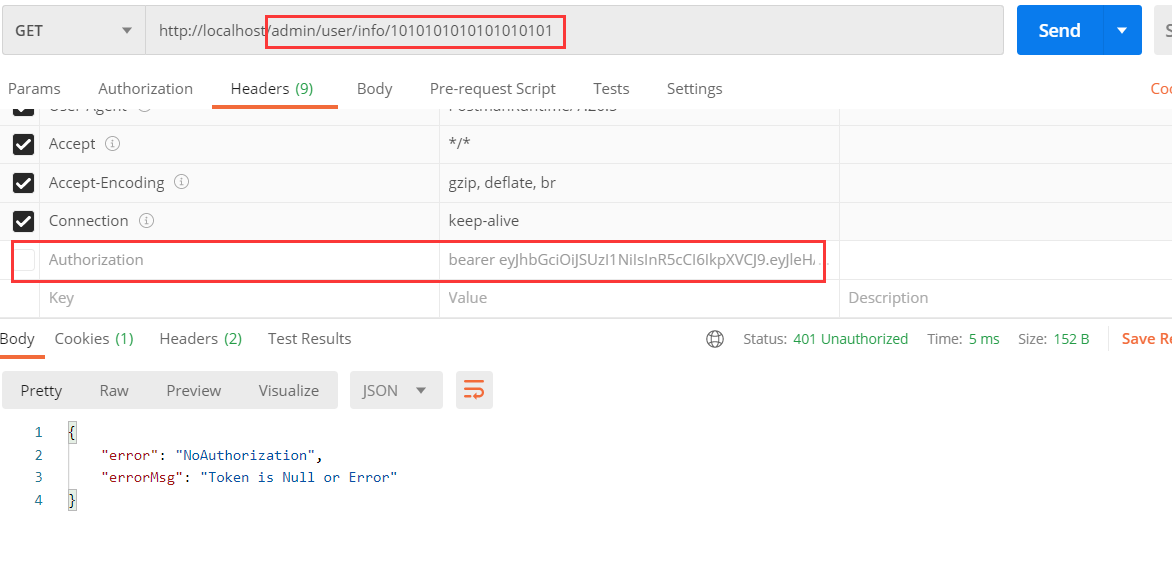


Redis：

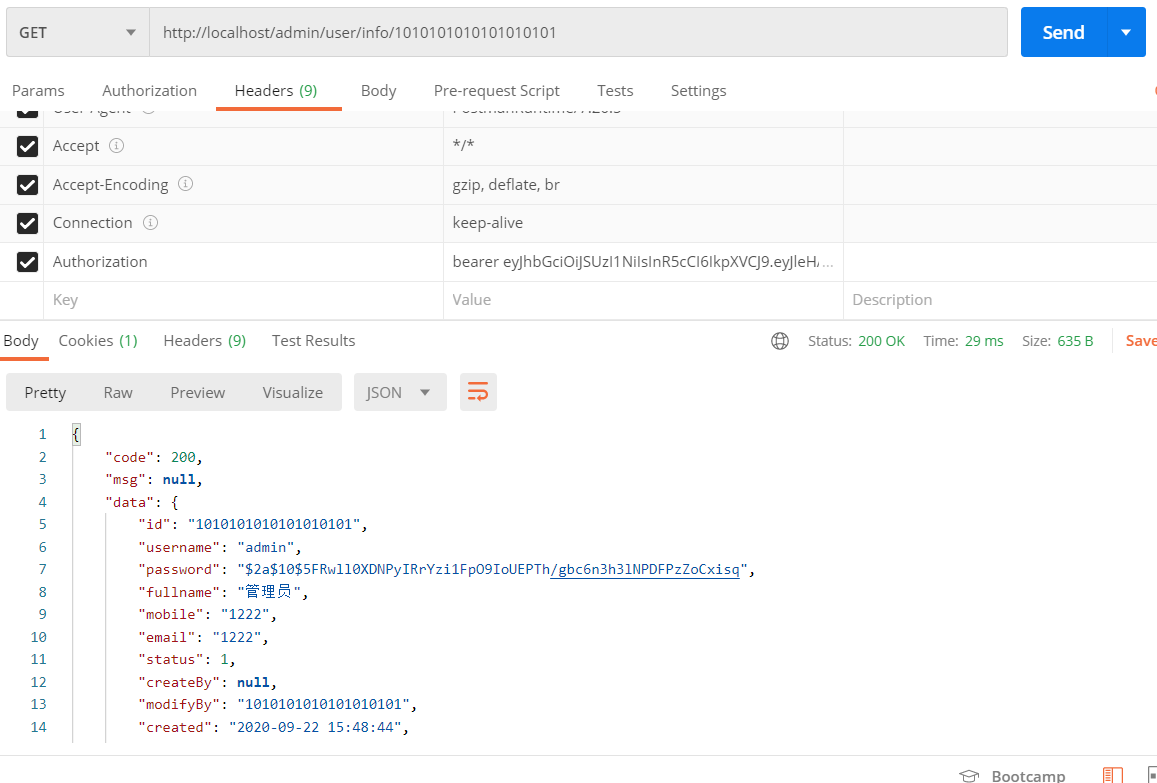


## 6.8 使用网关访问admin-service里面测试登录的接口

没有带Token时：



带Token后：



# 权限数据的CURD

## 7.1 API接口的展示



## 7.2 SysPrivilegeController

|  |
| --- |
| */\*\*  \* 权限管理的控制器  \* 1 查询  \* 2 修改  \* 3 新增  \* 4 删除  \* 权限管理的权限：  \* 查询：sys\_privilege\_query  \* 修改：sys\_privilege\_update  \* 新增：sys\_privilege\_create  \* 删除：sys\_privilege\_delete  \*/* @RestController @RequestMapping("/privileges") @Api(tags = "权限的管理") public class SysPrivilegeController {   @Autowired  private SysPrivilegeService sysPrivilegeService ;    */\*\*  \* 权限数据的分页查询  \* @param page  \* @return  \*/* @GetMapping  @ApiOperation(value = "分页查询权限数据")  @ApiImplicitParams({  @ApiImplicitParam(name = "current" ,value = "当前页") ,  @ApiImplicitParam(name = "size" ,value = "每页显示的大小") ,  })  @PreAuthorize("hasAuthority('sys\_privilege\_query')")  public R<Page<SysPrivilege>> findByPage(@ApiIgnore Page<SysPrivilege> page){   *// 查询时，我们将最近新增的、修改的数据优先展示-> 排序->lastUpdateTime* page.addOrder(OrderItem.*desc*("last\_update\_time")) ;  Page<SysPrivilege> sysPrivilegePage = sysPrivilegeService.page(page);  return R.*ok*(sysPrivilegePage) ;  }    @PostMapping  @ApiOperation(value = "新增一个权限")  @PreAuthorize("hasAuthority('sys\_privilege\_create')")  @ApiImplicitParams({  @ApiImplicitParam(name = "sysPrivilege" ,value = "sysPrivilege 的json数据")  })  public R add(@RequestBody @Validated SysPrivilege sysPrivilege){  *// 新增时，我们需要给我们的新增对象填充一些属性* String userIdStr = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal().toString();  sysPrivilege.setCreateBy(Long.*valueOf*(userIdStr));  sysPrivilege.setCreated(new Date());  sysPrivilege.setLastUpdateTime(new Date());  boolean save = sysPrivilegeService.save(sysPrivilege);  if(save){  return R.*ok*("新增成功") ;  }  return R.*fail*("新增失败") ;  }    @PatchMapping  @ApiOperation(value = "修改一个权限")  @PreAuthorize("hasAuthority('sys\_privilege\_update')")  @ApiImplicitParams({  @ApiImplicitParam(name = "sysPrivilege" ,value = "sysPrivilege 的json数据")  })  public R update(@RequestBody @Validated SysPrivilege sysPrivilege){  *// 新增时，我们需要给我们的新增对象填充一些属性* String userIdStr = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal().toString();  sysPrivilege.setModifyBy(Long.*valueOf*(userIdStr));  sysPrivilege.setLastUpdateTime(new Date());   boolean save = sysPrivilegeService.updateById(sysPrivilege);  if(save){  return R.*ok*("修改成功") ;  }  return R.*fail*("修改失败") ;  } } |

## 7.3 SysPrivilegeService

没有改动

## 7.4 SysPrivilegeServiceImpl

没有改动

## 7.5 SysPrivilege

|  |
| --- |
| */\*\*  \* 权限配置  \*/* @ApiModel(value="com-bjsxt-domain-SysPrivilege") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "sys\_privilege") public class SysPrivilege {  */\*\*  \* 主键  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="主键")  private Long id;   */\*\*  \* 所属菜单Id  \*/* @TableField(value = "menu\_id")  @ApiModelProperty(value="所属菜单Id")  **@NotNull  private Long menuId;**   */\*\*  \* 功能点名称  \*/* @TableField(value = "name")  @ApiModelProperty(value="功能点名称")  **@NotNull  private String name;**   */\*\*  \* 功能描述  \*/* @TableField(value = "description")  @ApiModelProperty(value="功能描述")  private String description;   @TableField(value = "url")  @ApiModelProperty(value="")  private String url;   @TableField(value = "method")  @ApiModelProperty(value="")  private String method;   */\*\*  \* 创建人  \*/* @TableField(value = "create\_by")  @ApiModelProperty(value="创建人")  private Long createBy;   */\*\*  \* 修改人  \*/* @TableField(value = "modify\_by")  @ApiModelProperty(value="修改人")  private Long modifyBy;   */\*\*  \* 创建时间  \*/* @TableField(value = "created")  @ApiModelProperty(value="创建时间")  private Date created;   */\*\*  \* 修改时间  \*/* @TableField(value = "last\_update\_time")  @ApiModelProperty(value="修改时间")  private Date lastUpdateTime; } |

# 自动填充

## 8.1 使用场景

当我们修改数据时，要添加一个修改人和修改时间

当我们新增数据时，要添加一个创建人、创建时间、修改时间，这些代码都是相同的，我们可以把它提出出来。

## 8.2 使用MybatisPlus的自动填充功能

### 8.2.1 添加一个AutoFillHandler（放在coin-common）

|  |
| --- |
| */\*\*  \*字段自动填充  \*/* @Component public class AutoFillHandler implements MetaObjectHandler {    */\*\*  \* 新增时填入值  \* @param metaObject  \*/* @Override  public void insertFill(MetaObject metaObject) {  Long userId = getUserId();  */\*\*  \* 3 种情况不填充  \* 1 值为null  \* 2 自动类型不匹配  \* 3 没有改字段  \*/* this.strictInsertFill(metaObject, "lastUpdateTime", Date.class, new Date());  this.strictInsertFill(metaObject, "createBy", Long.class, userId); *// 创建人的填充* this.strictInsertFill(metaObject, "created", Date.class, new Date());   }    */\*\*  \* 修改时填入值  \* @param metaObject  \*/* @Override  public void updateFill(MetaObject metaObject) {  Long userId = getUserId();  this.strictUpdateFill(metaObject, "lastUpdateTime", Date.class, new Date());  this.strictUpdateFill(metaObject, "modifyBy", Long.class, userId); *// 修改人的填充* }   */\*\*  \* 获取安全上下文里的用户对象 --- 主要是在线程里面获取改值  \* @return  \*/* private Long getUserId() {  Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  Long userId = null;  if (authentication != null) {  String principal = authentication.getPrincipal().toString();  userId = Long.*valueOf*(principal);  }  return userId;  } } |

### 8.2.2 在需要填充的字段上使用注解



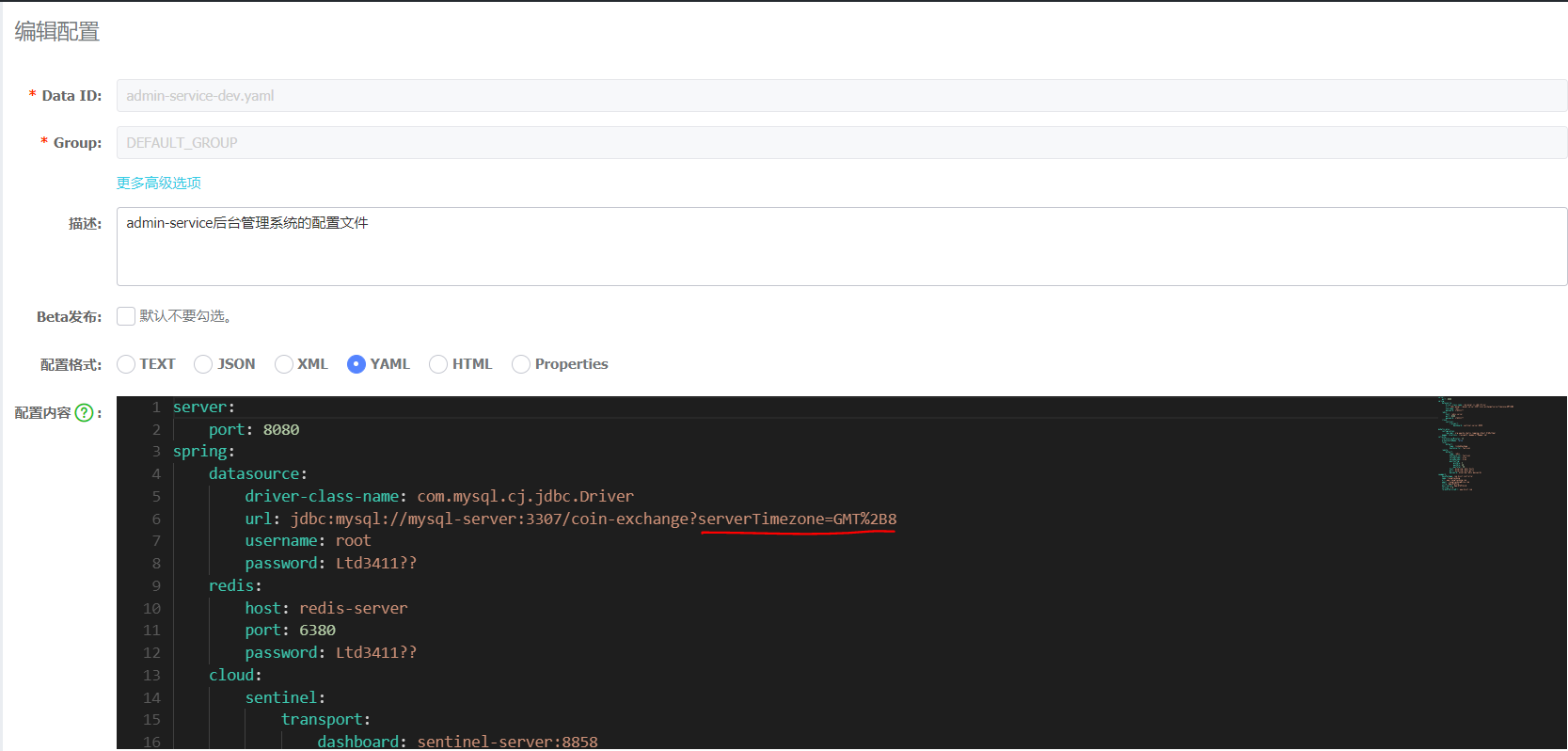
### 8.2.3 填充的判断



### 8.2.4 填充的时间问题

发现数据库新增后的时间比正常的时间少了8个小时，这是因为Mysql设置的时区问题，我们可以在连接Mysql时设置时区：

|  |
| --- |
| ?serverTimezone=GMT%2B8 |



# 角色管理

## 9.1 API接口的展示

## 9.2 SysRoleController

|  |
| --- |
| */\*\*  \* 角色管理  \*/* @RestController @RequestMapping("/roles") @Api(tags = "角色管理") public class SysRoleController {   @Autowired  private SysRoleService sysRoleService;   @GetMapping  @ApiOperation(value = "条件分页查询")  @PreAuthorize("hasAuthority('sys\_role\_query')")  @ApiImplicitParams({  @ApiImplicitParam(name = "current", value = "当前页"),  @ApiImplicitParam(name = "size", value = "每页显示的大小"),  @ApiImplicitParam(name = "name", value = "角色名称"),  })  public R<Page<SysRole>> findByPage(@ApiIgnore Page<SysRole> page, String name) {  page.addOrder(OrderItem.*desc*("last\_update\_time"));  return R.*ok*(sysRoleService.findByPage(page, name));  }    @PostMapping  @ApiOperation(value = "新增一个角色")  @PreAuthorize("hasAuthority('sys\_role\_create')")  public R add(@RequestBody @Validated SysRole sysRole) {  boolean save = sysRoleService.save(sysRole);  if (save) {  return R.*ok*();  }  return R.*fail*("新增失败");   }   @PostMapping("/delete")  @ApiOperation(value = "删除一个角色数据")  @PreAuthorize("hasAuthority('sys\_role\_delete')")  @ApiImplicitParams({  @ApiImplicitParam(name = "ids", value = "要删除角色的id的集合")  })  public R delete(@RequestBody String[] ids) {  if (ids == null || ids.length == 0) {  return R.*fail*("要删除的数据不能为null");  }  boolean b = sysRoleService.removeByIds(Arrays.*asList*(ids));  if (b) {  return R.*ok*();  }  return R.*fail*("删除失败");  }  } |

## 9.3 SysRoleService

|  |
| --- |
| public interface SysRoleService extends IService<SysRole>{    */\*\*  \* 判断一个用户是否为超级的管理员  \* @param userId  \* @return  \*/* boolean isSuperAdmin(Long userId);   */\*\*  \* 使用角色的名称模糊分页角色查询  \* @param page  \* 分页数据  \* @param name  \* 角色的名称  \* @return  \*/* Page<SysRole> findByPage(Page<SysRole> page, String name);  } |

## 9.4 SysRoleServiceImpl

|  |
| --- |
| @Service public class SysRoleServiceImpl extends ServiceImpl<SysRoleMapper, SysRole> implements SysRoleService {   @Autowired  private SysRoleMapper sysRoleMapper;     */\*\*  \* 判断一个用户是否为超级的管理员  \*  \* @param userId  \* @return  \*/* @Override  public boolean isSuperAdmin(Long userId) {  *// 当用户的角色code 为：ROLE\_ADMIN 时，该用户为超级的管理员  // 用户的id->用户的角色->该角色的Code是否为ROLE\_ADMIN* String roleCode = sysRoleMapper.getUserRoleCode(userId);  if (!StringUtils.*isEmpty*(roleCode) && roleCode.equals("ROLE\_ADMIN")) {  return true;  }  return false;  }   */\*\*  \* 使用角色的名称模糊分页角色查询  \*  \* @param page 分页数据  \* @param name 角色的名称  \* @return  \*/* @Override  public Page<SysRole> findByPage(Page<SysRole> page, String name) {  return page(page, new LambdaQueryWrapper<SysRole>().like(  !StringUtils.*isEmpty*(name),  SysRole::getName,  name  ));  } } |

## 9.5 SysRole

|  |
| --- |
| package com.bjsxt.domain;  import com.baomidou.mybatisplus.annotation.\*; import io.swagger.annotations.ApiModel; import io.swagger.annotations.ApiModelProperty; import lombok.AllArgsConstructor; import lombok.Data; import lombok.NoArgsConstructor;  import javax.validation.constraints.NotNull; import java.util.Date;  */\*\*  \* 角色  \*/* @ApiModel(value="com-bjsxt-domain-SysRole") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "sys\_role") public class SysRole {  */\*\*  \* 主键  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="主键")  private Long id;   */\*\*  \* 名称  \*/* @TableField(value = "name")  @ApiModelProperty(value="名称")  @NotNull  private String name;   */\*\*  \* 代码  \*/* @TableField(value = "code")  @ApiModelProperty(value="代码")  @NotNull  private String code;   */\*\*  \* 描述  \*/* @TableField(value = "description")  @ApiModelProperty(value="描述")  private String description;   */\*\*  \* 创建人  \*/* **@TableField(value = "create\_by",fill = FieldFill.*INSERT*)**  @ApiModelProperty(value="创建人")  private Long createBy;   */\*\*  \* 修改人  \*/* @TableField(value = "modify\_by" )   **@TableField(value = "modify\_by",fill = FieldFill.*UPDATE* )**  private Long modifyBy;   */\*\*  \* 状态0:禁用 1:启用  \*/* @TableField(value = "status")  @ApiModelProperty(value="状态0:禁用 1:启用")  private Byte status;   */\*\*  \* 创建时间  \*/* **@TableField(value = "created" ,fill = FieldFill.*INSERT*)**  @ApiModelProperty(value="创建时间")  private Date created;   */\*\*  \* 修改时间  \*/* **@TableField(value = "last\_update\_time",fill = FieldFill.*INSERT\_UPDATE*)**  @ApiModelProperty(value="修改时间")  private Date lastUpdateTime;  } |

# 角色权限分配

## 10.1 api接口的展示



## 10.2 效果展示



## 10.3 SysMenu

添加如下的代码：

|  |
| --- |
| @TableField(exist = false) @ApiModelProperty("该菜单下的所有的权限") private List<SysPrivilege> sysPrivileges ;   @TableField(exist = false) @ApiModelProperty("该菜单的子菜单") private List<SysMenu> childs ;   @TableField(exist = false) @ApiModelProperty("该菜单的唯一Key值") private String menuKey ;  */\*\*  \* 获取菜单的唯一Key凭证  \* @return  \*/* public String getMenuKey() {  if (!StringUtils.*isEmpty*(parentKey)) {  return parentKey+"."+id;  }else {  return id.toString();  } } |

## 10.4 SysPrivilege

在使用角色查询权限时,我们也会使用该字段来标记当前的角色使用有该权限,有权限为1,否则为0

|  |
| --- |
| @TableField(exist = false) @ApiModelProperty(value="当前角色是否拥有这个权限") private int own ; |

## 10.5 SysRolePrivilegeController

|  |
| --- |
| @Api(tags = "角色权限的配置") @RestController public class SysRolePrivilegeController {    @Autowired  private SysRolePrivilegeService sysRolePrivilegeService ;    @GetMapping("/roles\_privileges")  @ApiOperation(value = "查询角色的权限列表")  @ApiImplicitParams({  @ApiImplicitParam(name = "roleId" ,value = "角色的ID")  })  public R<List<SysMenu>> findSysMenuAndPrivileges(Long roleId){  List<SysMenu> sysMenus = sysRolePrivilegeService.findSysMenuAndPrivileges(roleId) ;  return R.*ok*(sysMenus) ;  }    @PostMapping("/grant\_privileges")  @ApiOperation(value = "授予角色某种权限")  @ApiImplicitParams({  @ApiImplicitParam(name = "rolePrivilegesParam" ,value = "rolePrivilegesParam json数")  })  public R grantPrivileges(@RequestBody RolePrivilegesParam rolePrivilegesParam){  boolean isOk = sysRolePrivilegeService.grantPrivileges(rolePrivilegesParam) ;  if(isOk){  return R.*ok*() ;  }  return R.*fail*("授予失败") ;  }  } |

## 10.6 SysRolePrivilegeService

|  |
| --- |
| public interface SysRolePrivilegeService extends IService<SysRolePrivilege>{   */\*\*  \* 查询角色的权限  \* @param roleId  \* @return  \*/* List<SysMenu> findSysMenuAndPrivileges(Long roleId);    */\*\*  \* 给角色授权权限  \* @param rolePrivilegesParam  \* @return  \*/* boolean grantPrivileges(RolePrivilegesParam rolePrivilegesParam); } |

## 10.7 SysRolePrivilegeServiceImpl

|  |
| --- |
| @Service public class SysRolePrivilegeServiceImpl extends ServiceImpl<SysRolePrivilegeMapper, SysRolePrivilege> implements SysRolePrivilegeService {   @Autowired  private SysMenuService sysMenuService;   @Autowired  private SysPrivilegeService sysPrivilegeService;   @Autowired  private SysRolePrivilegeService sysRolePrivilegeService;   */\*\*  \* 查询角色的权限  \*  \* @param roleId  \* @return  \*/* @Override  public List<SysMenu> findSysMenuAndPrivileges(Long roleId) {  List<SysMenu> list = sysMenuService.list(); *// 查询所有的菜单  // 我们在页面显示的是二级菜单,以及二级菜单包含的权限* if (CollectionUtils.*isEmpty*(list)) {  return Collections.*emptyList*();  }  List<SysMenu> rootMenus = list.stream()  .filter(sysMenu -> sysMenu.getParentId() == null)  .collect(Collectors.*toList*());  if (CollectionUtils.*isEmpty*(rootMenus)) {  return Collections.*emptyList*();  }  *// 查询所有的二级菜单* List<SysMenu> subMenus = new ArrayList<>();  for (SysMenu rootMenu : rootMenus) {  subMenus.addAll(getChildMenus(rootMenu.getId(), roleId, list));  }  return subMenus;  }   */\*\*  \* 查询菜单的子菜单 (递归)  \*  \* @param parentId 父菜单的ID  \* @param roleId 当前查询的角色的ID  \* @return  \*/* private List<SysMenu> getChildMenus(Long parentId, Long roleId, List<SysMenu> sources) {  List<SysMenu> childs = new ArrayList<>();  for (SysMenu source : sources) {  if (source.getParentId() == parentId) { *// 找儿子* childs.add(source);  source.setChilds(getChildMenus(source.getId(), roleId, sources)); *// 给该儿子设置儿子* List<SysPrivilege> sysPrivileges = sysPrivilegeService.getAllSysPrivilege(source.getId(), roleId);  source.setPrivileges(sysPrivileges); *// 该儿子可能包含权限* }  }  return childs;  }    */\*\*  \* 给角色授权权限  \*  \* @param rolePrivilegesParam  \* @return  \*/* @Transactional  @Override  public boolean grantPrivileges(RolePrivilegesParam rolePrivilegesParam) {  Long roleId = rolePrivilegesParam.getRoleId(); *// 角色Id  //1 先删除之前该角色的权限* sysRolePrivilegeService.remove(new LambdaQueryWrapper<SysRolePrivilege>().eq(SysRolePrivilege::getRoleId, roleId));  *// 移除之前的值成功* List<Long> privilegeIds = rolePrivilegesParam.getPrivilegeIds();  if (!CollectionUtils.*isEmpty*(privilegeIds)) {  List<SysRolePrivilege> sysRolePrivileges = new ArrayList<>();  for (Long privilegeId : privilegeIds) {  SysRolePrivilege sysRolePrivilege = new SysRolePrivilege();  sysRolePrivilege.setRoleId(rolePrivilegesParam.getRoleId());  sysRolePrivilege.setPrivilegeId(privilegeId);  sysRolePrivileges.add(sysRolePrivilege);  }  *// 2 新增新的值* boolean b = sysRolePrivilegeService.saveBatch(sysRolePrivileges);  return b;  }  *// 2 新增该角色的权限* return true ;  } } |

## 10.8 SysPrivilegeService

|  |
| --- |
| public interface SysPrivilegeService extends IService<SysPrivilege>{    */\*\*  \* 获取该菜单下面所有的权限  \* @param roleId  \* roleId 代表当前的查询的角色的ID  \* @param menuId 菜单的ID  \*  \* @return  \*/* List<SysPrivilege> getAllSysPrivilege(Long menuId ,Long roleId); } |

## 10.9 SysPrivilegeServiceImpl

|  |
| --- |
| @Service public class SysPrivilegeServiceImpl extends ServiceImpl<SysPrivilegeMapper, SysPrivilege> implements SysPrivilegeService {    @Autowired  private SysPrivilegeMapper sysPrivilegeMapper ;  */\*\*  \* 获取该菜单下面所有的权限  \*  \* @param menuId 菜单的ID  \* @param roleId roleId 代表当前的查询的角色的ID  \* @return  \*/* @Override  public List<SysPrivilege> getAllSysPrivilege(Long menuId, Long roleId) {  *// 1 查询所有的该菜单下的权限* List<SysPrivilege> sysPrivileges = list(new LambdaQueryWrapper<SysPrivilege>().eq(SysPrivilege::getMenuId, menuId));  if(CollectionUtils.*isEmpty*(sysPrivileges)){  return Collections.*emptyList*() ;  }  *// 2 当前传递的角色使用包含该权限信息也要放进去* for (SysPrivilege sysPrivilege : sysPrivileges) {  Set<Long> currentRoleSysPrivilegeIds = sysPrivilegeMapper.getPrivilegesByRoleId(roleId) ;  if (currentRoleSysPrivilegeIds.contains(sysPrivilege.getId())){  sysPrivilege.setOwn(1); *// 当前的角色是否有该权限* }   }  return sysPrivileges;  } } |

10.10 SysPrivilegeMapper

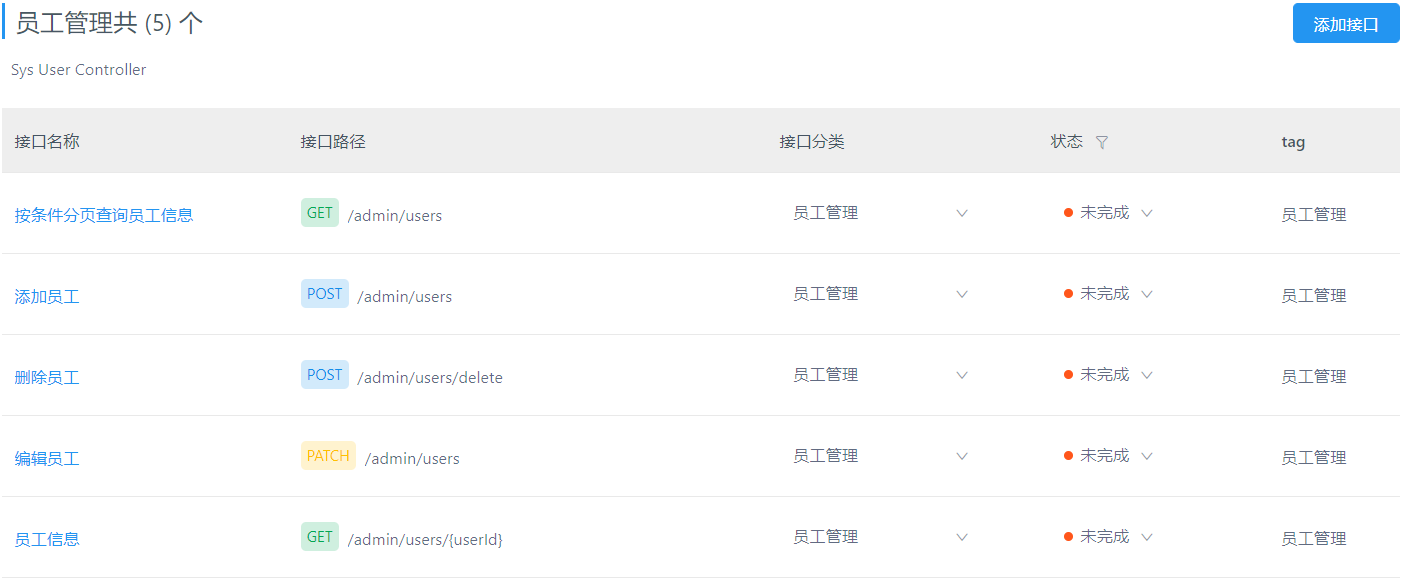
|  |
| --- |
| public interface SysPrivilegeMapper extends BaseMapper<SysPrivilege> {  */\*\*  \* 使用角色Id 查询权限  \* @param roleId  \* @return  \*/* List<SysPrivilege> selectByRoleId(Long roleId);   */\*\*  \* 使用角色的ID 查询权限的id  \* @param roleId  \* @return  \*/* Set<Long> getPrivilegesByRoleId(Long roleId); } |

## 10.11 SysPrivilegeMapper.xml

|  |
| --- |
| <select id="getPrivilegesByRoleId" resultType="java.lang.Long">   SELECT  privilege\_id  FROM  sys\_role\_privilege  WHERE role\_id = #{roleId}  </select> |

# 员工管理

## 7.1 API接口的展示



## 7.2 SysUserController

|  |
| --- |
| @Api(tags = "员工管理") @RequestMapping("/users") @RestController public class SysUserController {   @Autowired  private SysUserService sysUserService ;   @GetMapping  @ApiImplicitParams({  @ApiImplicitParam(name = "current" ,value = "当前页") ,  @ApiImplicitParam(name = "size" ,value = "每页显示的条数") ,  @ApiImplicitParam(name = "mobile" ,value = "员工的手机号码") ,  @ApiImplicitParam(name = "fullname" ,value = "员工的全名称") ,  })  @PreAuthorize("hasAuthority('sys\_user\_query')")  public R<Page<SysUser>> findByPage(@ApiIgnore Page<SysUser> page ,String mobile ,String fullname){  page.addOrder(OrderItem.*desc*("last\_update\_time")) ;  Page<SysUser> pageData = sysUserService.findByPage(page,mobile ,fullname) ;  return R.*ok*(pageData) ;  }    @PostMapping  @ApiOperation(value = "新增员工")  @ApiImplicitParams({  @ApiImplicitParam(name = "sysUser" ,value = "sysUser 的json数据")  })  @PreAuthorize("hasAuthority('sys\_user\_create')")  public R addUser(@RequestBody SysUser sysUser){  Long userId = Long.*valueOf*(SecurityContextHolder.*getContext*().getAuthentication().getPrincipal().toString());  sysUser.setCreateBy(userId);  boolean isOk = sysUserService.addUser(sysUser) ;  if(isOk){  return R.*ok*() ;  }  return R.*fail*("新增失败") ;  }     @PostMapping("/delete")  @ApiOperation(value = "删除用户")  @PreAuthorize("hasAuthority('sys\_user\_delete')")  public R deleteUser( @RequestBody Long ids[] ){  boolean b = sysUserService.removeByIds(Arrays.*asList*(ids));  if(b){  return R.*ok*() ;  }  return R.*fail*() ;  } } |

## 7.3 SysUserService

|  |
| --- |
| public interface SysUserService extends IService<SysUser>{    */\*\*  \* 分页查询员工  \* @param page  \* 分页参数  \* @param mobile  \* 员工的手机号  \* @param fullname  \* 员工的全名称  \* @return  \*/* Page<SysUser> findByPage(Page<SysUser> page, String mobile, String fullname);    */\*\*  \* 新增员工  \* @param sysUser  \* @return  \*/* boolean addUser(SysUser sysUser); } |

## 7.4 SysUserServiceImpl

|  |
| --- |
| @Service public class SysUserServiceImpl extends ServiceImpl<SysUserMapper, SysUser> implements SysUserService{    @Autowired  private SysUserRoleService sysUserRoleService ;   */\*\*  \* 分页查询员工  \*  \* @param page 分页参数  \* @param mobile 员工的手机号  \* @param fullname 员工的全名称  \* @return  \*/* @Override  public Page<SysUser> findByPage(Page<SysUser> page, String mobile, String fullname) {  Page<SysUser> pageData = page(page,  new LambdaQueryWrapper<SysUser>()  .like(!StringUtils.*isEmpty*(mobile), SysUser::getMobile, mobile)  .like(!StringUtils.*isEmpty*(fullname), SysUser::getFullname, fullname)   );  List<SysUser> records = pageData.getRecords();  if(!CollectionUtils.*isEmpty*(records)){  for (SysUser record : records) {  List<SysUserRole> userRoles = sysUserRoleService.list(new LambdaQueryWrapper<SysUserRole>().eq(SysUserRole::getUserId, record.getId()));  if(!CollectionUtils.*isEmpty*(userRoles)){  record.setRole\_strings(  userRoles.stream().  map(sysUserRole -> sysUserRole.getRoleId().toString())  .collect(Collectors.*joining*(",")));  }  }  }  return pageData;  }    */\*\*  \* 新增员工  \*  \* @param sysUser  \* @return  \*/* @Override  @Transactional  public boolean addUser(SysUser sysUser) {  *// 1 用户的密码* String password = sysUser.getPassword();  *// 用户的角色Ids* String role\_strings = sysUser.getRole\_strings();  BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();  String encode = bCryptPasswordEncoder.encode(password); *// 加密密码* sysUser.setPassword(encode); *// 设置密码* boolean save = super.save(sysUser);  if(save){  *// 给用户新增角色数据* if(!StringUtils.*isEmpty*(role\_strings)){  String[] roleIds = role\_strings.split(",");  List<SysUserRole> sysUserRoleList = new ArrayList<>(roleIds.length) ;  for (String roleId : roleIds) {  SysUserRole sysUserRole = new SysUserRole();  sysUserRole.setRoleId(Long.*valueOf*(roleId));  sysUserRole.setUserId(sysUser.getId());  sysUserRoleList.add(sysUserRole) ;  }  sysUserRoleService.saveBatch(sysUserRoleList) ;  }  }  return save;  }   @Override  public boolean removeByIds(Collection<? extends Serializable> idList) {  boolean b = super.removeByIds(idList);  sysUserRoleService.remove(new LambdaQueryWrapper<SysUserRole>().in(SysUserRole::getUserId,idList)) ;  return b;  } } |

## 7.5 SysUser

|  |
| --- |
| */\*\*  \* 平台用户  \*/* @ApiModel(value="com-bjsxt-domain-SysUser") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "sys\_user") public class SysUser {  */\*\*  \* 主键  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="主键")  private Long id;   */\*\*  \* 账号  \*/* @TableField(value = "username")  @ApiModelProperty(value="账号")  @NotBlank  private String username;   */\*\*  \* 密码  \*/* @TableField(value = "password")  @ApiModelProperty(value="密码")  @NotBlank  private String password;   */\*\*  \* 姓名  \*/* @TableField(value = "fullname")  @ApiModelProperty(value="姓名")  private String fullname;   */\*\*  \* 手机号  \*/* @TableField(value = "mobile")  @ApiModelProperty(value="手机号")  @NotBlank  private String mobile;   */\*\*  \* 邮箱  \*/* @TableField(value = "email")  @ApiModelProperty(value="邮箱")  @NotBlank  private String email;   */\*\*  \* 状态 0-无效； 1-有效；  \*/* @TableField(value = "status")  @ApiModelProperty(value="状态 0-无效； 1-有效；")  private Byte status;   */\*\*  \* 创建人  \*/* @TableField(value = "create\_by")  @ApiModelProperty(value="创建人")  private Long createBy;   */\*\*  \* 修改人  \*/* @TableField(value = "modify\_by",fill = FieldFill.*UPDATE*)  @ApiModelProperty(value="修改人")  private Long modifyBy;   */\*\*  \* 创建时间  \*/* @TableField(value = "created",fill = FieldFill.*INSERT*)  @ApiModelProperty(value="创建时间")  private Date created;   */\*\*  \* 修改时间  \*/* @TableField(value = "last\_update\_time",fill = FieldFill.*INSERT\_UPDATE*)  @ApiModelProperty(value="修改时间")  private Date lastUpdateTime;    @ApiModelProperty(value = "角色的IDS")  @TableField(exist = false)  private String role\_strings ;   } |

# 系统日志

## 12.1 API接口的展示



## 12.2系统日志的查询

|  |
| --- |
| @RestController @Api(tags = "用户的操作记录查询") @RequestMapping("/sysUserLog") public class SysUserLogController {   @Autowired  private SysUserLogService sysUserLogService ;   @GetMapping  @ApiOperation(value = "分页查询用户的操作记录")  @ApiImplicitParams({  @ApiImplicitParam(name = "page" ,value = "分页查询")  })  public R<Page<SysUserLog>> findByPage(@ApiIgnore Page<SysUserLog> page){  page.addOrder(OrderItem.*desc*("created")) ;  return R.*ok*(sysUserLogService.page(page)) ;  } } |

## 12.3系统日志的记录

|  |
| --- |
| *//@Component* @Aspect @Order(2) @Slf4j public class WebLogAdminAspect {   */\*\*  \* 雪花算法  \* 1 : 机器的id  \* 2 : 应用的id  \*/* private Snowflake snowflake = new Snowflake(1,1) ;  @Autowired  private SysUserLogService sysUserLogService ;  */\*\*  \* 日志记录：  \* 环绕通知：方法执行之前、之后  \*/   /\*\*  \* 1 定义切入点  \*/* @Pointcut("execution(\* com.bjsxt.controller.\*.\*(..))") *// controller 包里面所有类，类里面的所有方法 都有该切面* public void webLog(){}   */\*\*  \* 2 记录日志的环绕通知  \*/* @Around("webLog()")  public Object recodeWebLog(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {  Object result = null ;  WebLog webLog = new WebLog();  long start = System.*currentTimeMillis*() ;   *// 执行方法的真实调用* result = proceedingJoinPoint.proceed(proceedingJoinPoint.getArgs());   long end = System.*currentTimeMillis*() ;    webLog.setSpendTime((int)(start-end)/1000); *// 请求该接口花费的时间  // 获取当前请求的request对象* ServletRequestAttributes requestAttributes = (ServletRequestAttributes)RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = requestAttributes.getRequest();   *// 获取安全的上下文* Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();   String url = request.getRequestURL().toString();  webLog.setUri(request.getRequestURI()); *// 设置请求的uri* webLog.setUrl(url);  webLog.setBasePath(StrUtil.*removeSuffix*(url, URLUtil.*url*(url).getPath())); *// http://ip:port/* webLog.setUsername(authentication==null ? "anonymous":authentication.getPrincipal().toString()); *// 获取用户的id* webLog.setIp(request.getRemoteAddr()); *// TODO 获取ip 地址    // 获取方法* MethodSignature signature = (MethodSignature)proceedingJoinPoint.getSignature();  *// 获取类的名称* String targetClassName = proceedingJoinPoint.getTarget().getClass().getName();  Method method = signature.getMethod();  *// 因为我们会使用Swagger 这工具，我们必须在方法上面添加@ApiOperation(value="")该注解  // 获取ApiOperation* ApiOperation annotation = method.getAnnotation(ApiOperation.class);  webLog.setDescription(annotation==null ? "no desc":annotation.value());  webLog.setMethod(targetClassName+"."+method.getName()); *// com.bjsxt.controller.UserController.login()* webLog.setParameter(getMethodParameter(method,proceedingJoinPoint.getArgs())); *//{"key\_参数的名称":"value\_参数的值"}* webLog.setResult(result);   SysUserLog sysUserLog = new SysUserLog();   sysUserLog.setId(snowflake.nextId());  sysUserLog.setCreated(new Date());  sysUserLog.setDescription(webLog.getDescription());  sysUserLog.setGroup(webLog.getDescription());  sysUserLog.setUserId(Long.*valueOf*(webLog.getUsername()));  sysUserLog.setMethod(webLog.getMethod());  sysUserLog.setIp(sysUserLog.getIp());  sysUserLogService.save(sysUserLog) ;  return result ;  }   */\*\*  \* 获取方法的执行参数  \* @param method  \* @param args  \* @return  \* {"key\_参数的名称":"value\_参数的值"}  \*/* private Object getMethodParameter(Method method, Object[] args) {  Map<String, Object> methodParametersWithValues = new HashMap<>();  LocalVariableTableParameterNameDiscoverer localVariableTableParameterNameDiscoverer =  new LocalVariableTableParameterNameDiscoverer();  *// 方法的形参名称* String[] parameterNames = localVariableTableParameterNameDiscoverer.getParameterNames(method);  for (int i = 0; i <parameterNames.length ; i++) {  if(parameterNames[i].equals("password") || parameterNames[i].equals("file")){  methodParametersWithValues.put(parameterNames[i],"受限的支持类型") ;  }else{  methodParametersWithValues.put(parameterNames[i],args[i]) ;  }   }   return methodParametersWithValues ;  } } |

# 公告管理（查询、删除、状态更新）

## 13.1 API接口展示



## 13.2 NoticController

|  |
| --- |
| @RestController @RequestMapping("/notices") @Api(tags = "公告管理") public class NoticeController {   @Autowired  private NoticeService noticeService;   */\*\*  \* 分页查询  \*/* @GetMapping  @ApiOperation(value = "分页查询公告")  @PreAuthorize("hasAuthority('notice\_query')")  @ApiImplicitParams({  @ApiImplicitParam(name = "current", value = "当前页"),  @ApiImplicitParam(name = "size", value = "每页显示的条数"),  @ApiImplicitParam(name = "title", value = "公告的标题"),  @ApiImplicitParam(name = "startTime", value = "公告的创建开始时间"),  @ApiImplicitParam(name = "endTime", value = "公告的创建结束时间时间"),  })  public R<Page<Notice>> findByPage(@ApiIgnore Page<Notice> page, String title, String startTime, String endTime, Integer status) {  page.addOrder(OrderItem.*desc*("last\_update\_time"));  return R.*ok*(noticeService.findByPage(page, title, startTime, endTime, status));  }    @PostMapping("/delete")  @ApiOperation(value = "删除一个公告")  @PreAuthorize("hasAuthority('notice\_delete')")  @ApiImplicitParams({  @ApiImplicitParam(name = "ids", value = "要删除的id的集合")  })  public R delete(@RequestBody String[] ids) {  if (ids == null || ids.length == 0) {  return R.*fail*("删除时需要给id的值");  }  boolean b = noticeService.removeByIds(Arrays.*asList*(ids));  if (b) {  return R.*ok*();  }  return R.*fail*("删除失败");  }    @PostMapping("/updateStatus")  @ApiOperation(value = "启用/禁用一个公告")  @PreAuthorize("hasAuthority('notice\_update')")  @ApiImplicitParams({  @ApiImplicitParam(name = "id", value = "要启用/禁用一个公告的公告Id"),  @ApiImplicitParam(name = "status", value = "要设置的公告状态")   })  public R updateStatus(Long id, Integer status) {  Notice notice = new Notice();  notice.setId(id);  notice.setStatus(status);  boolean b = noticeService.updateById(notice); *// 局部的修改：不为null 修改* if (b) {  return R.*ok*("修改成功") ;  }  return R.*fail*("修改失败");  } } |

## 13.3 NoticeService

|  |
| --- |
| public interface NoticeService extends IService<Notice>{    */\*\*  \* 条件查询公告  \* @param page  \* 分页参数  \* @param title  \* 公告的标签  \* @param startTime  \* 公告创建的开始时间  \* @param endTime  \* 公告创建的结束时间  \* @param status  \* 公告当前的状态  \* @return  \*/* Page<Notice> findByPage(Page<Notice> page, String title, String startTime, String endTime, Integer status); } |

## 13.4 NoticeServiceImpl

|  |
| --- |
| @Service public class NoticeServiceImpl extends ServiceImpl<NoticeMapper, Notice> implements NoticeService{   */\*\*  \* 条件查询公告  \*  \* @param page 分页参数  \* @param title 公告的标签  \* @param startTime 公告创建的开始时间  \* @param endTime 公告创建的结束时间  \* @param status 公告当前的状态  \* @return  \*/* @Override  public Page<Notice> findByPage(Page<Notice> page, String title, String startTime, String endTime, Integer status) {  return page(page,new LambdaQueryWrapper<Notice>()  .like(!StringUtils.*isEmpty*(title),Notice::getTitle,title)  .between(!StringUtils.*isEmpty*(startTime)&&!StringUtils.*isEmpty*(endTime) ,Notice::getCreated,startTime,endTime+" 23:59:59")  .eq(status!=null,Notice::getStatus,status)  );  } } |

# 文件上传

## 14.1 功能和Api 接口



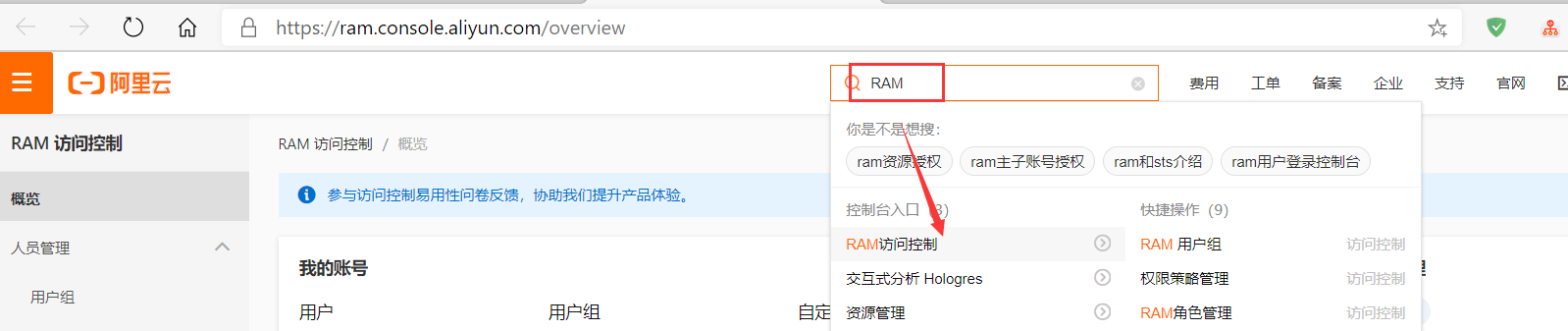

## 14.2 Spring Cloud OSS 接入准备

### 14.2.1 添加依赖

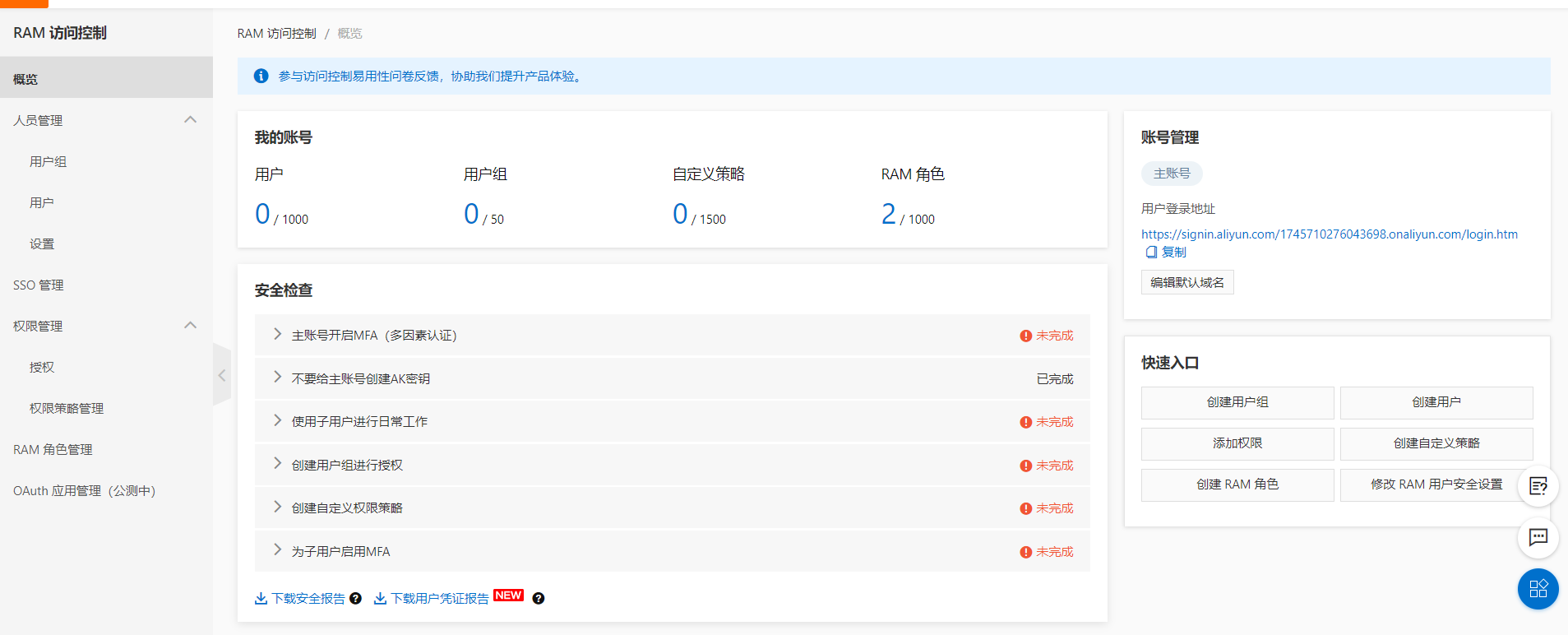
|  |
| --- |
| <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alicloud-oss</artifactId> </dependency> |

### 14.2.2 申请账号

登录阿里云官网：



#### 搜索RAM->进入RAM 访问控制。

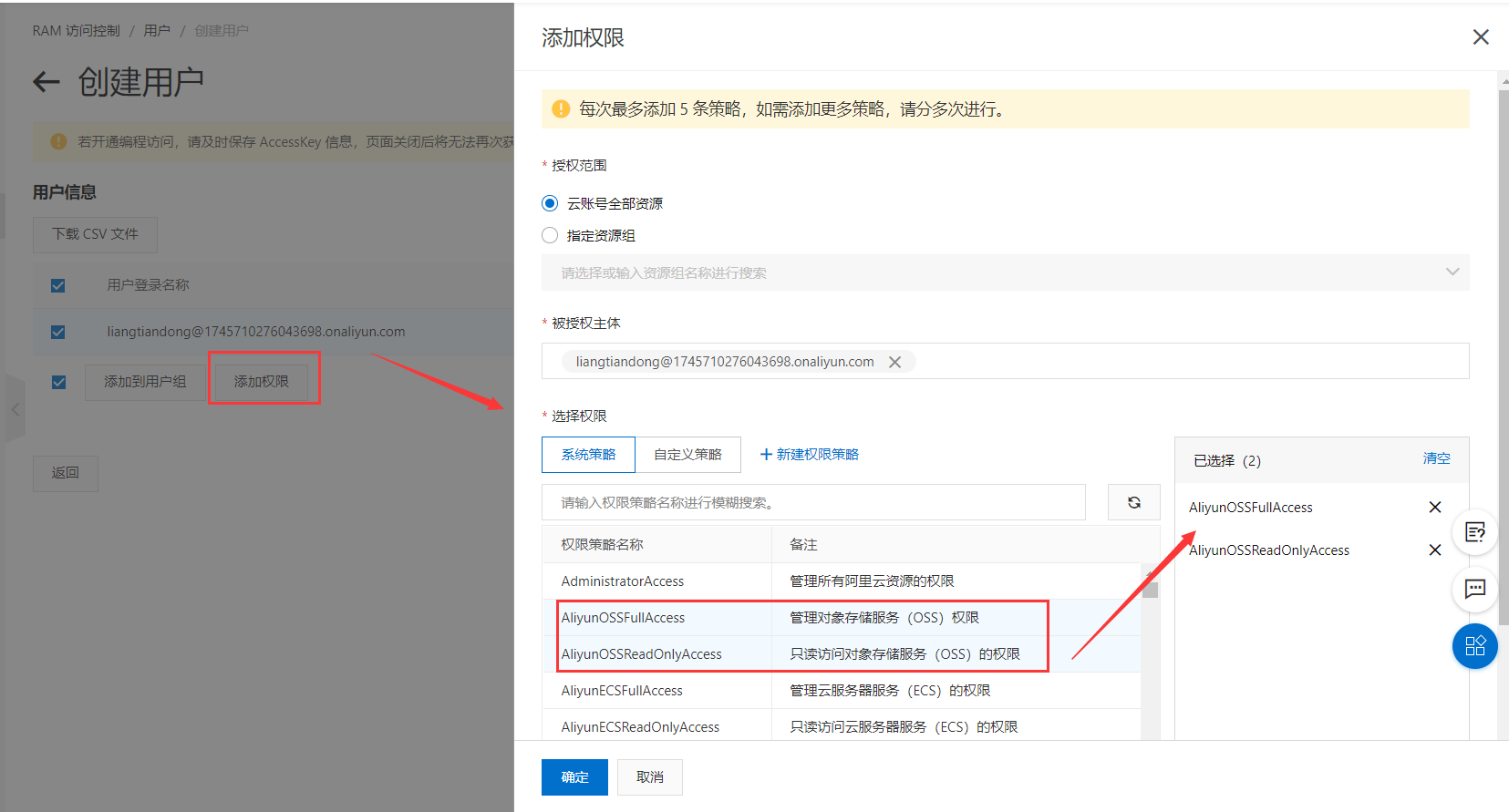


#### 创建用户：





#### 授予权限：



#### 记录你的 AccessKey ID和AccessKey Secret信息：

|  |  |
| --- | --- |
| AccessKey ID | LTAI4GEXnLoaQg6UraLEJL1v |
| AccessKey Secret | txr1DH29usTcQo5MUSsDGoVBusQQHX |

### 14.2.3 开启OSS 对象存储

#### 进入OSS控制中心：



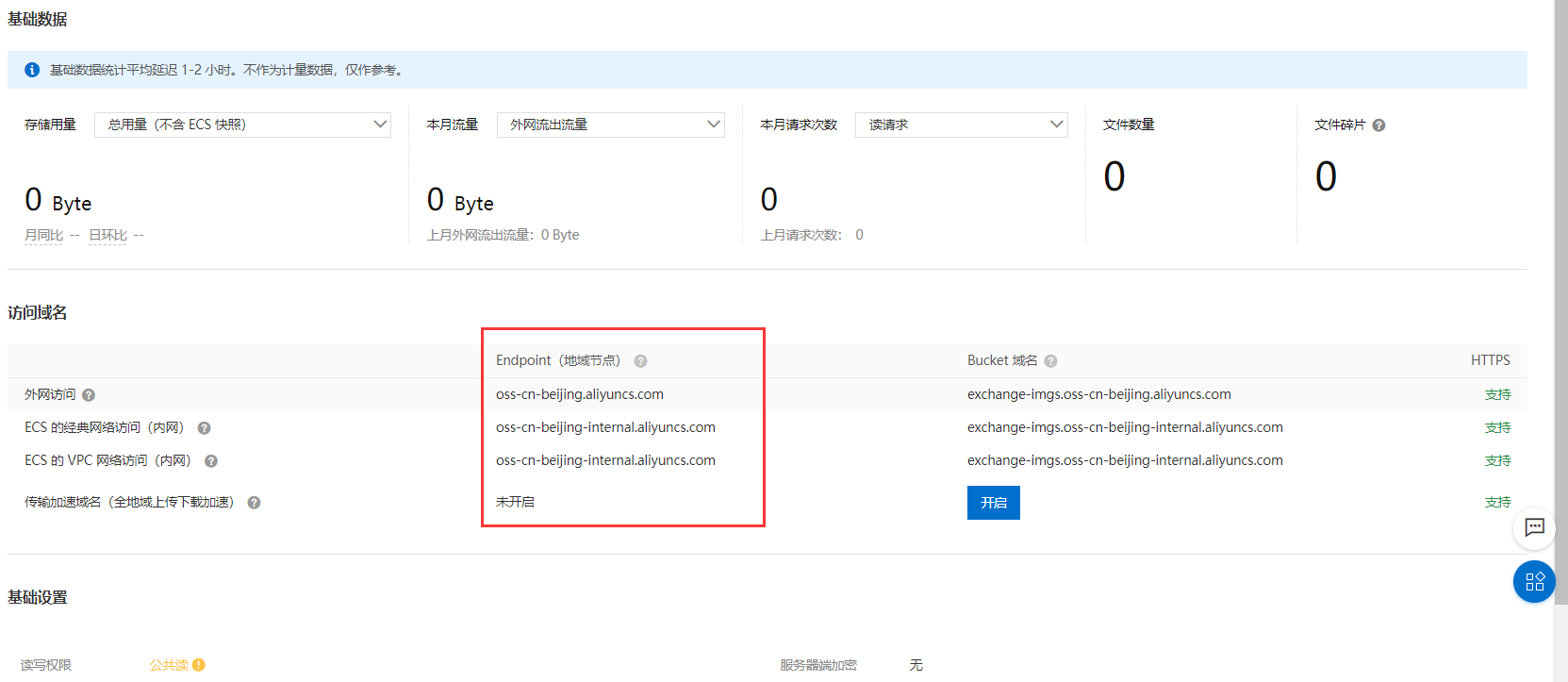
#### 创建Bucket：





信息确认无误后，点击确定：

#### 记录地域地址：



|  |
| --- |
| oss-cn-beijing.aliyuncs.com  oss-cn-beijing-internal.aliyuncs.com  oss-cn-beijing-internal.aliyuncs.com |

### 14.2.4 添加配置文件

打开Nacos，修改配置文件：





整体的配置文件：

|  |
| --- |
| server:  port: 8080  spring:  datasource:  driver-class-name: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://mysql-server:3307/coin-exchange?serverTimezone=GMT%2B8  username: root  password: Ltd3411??  redis:  host: redis-server  port: 6380  password: Ltd3411??  cloud:  sentinel:  transport:  dashboard: sentinel-server:8858  alicloud:  access-key: LTAI4GEXnLoaQg6UraLEJL1v  secret-key: txr1DH29usTcQo5MUSsDGoVBusQQHX  oss:  endpoint: oss-cn-beijing.aliyuncs.com  mybatis-plus:  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  mapper-locations: classpath:/mappers/\*Mapper.xml  jetcache:  statIntervalMinutes: 15  areaInCacheName: false  local:  default:  type: linkedhashmap  keyConvertor: fastjson  remote:  default:  type: redis  keyConvertor: fastjson  valueEncoder: kryo  valueDecoder: kryo  poolConfig:  minIdle: 5  maxIdle: 20  maxTotal: 50  host: ${spring.redis.host}  port: ${spring.redis.port}  password: ${spring.redis.password}  swagger2:  basePackage: com.bjsxt.controller  name: liangtiandong  url: www.liangtiandong.com  email: liangtiandong@live.com  title: 后台管理系统API接口  description: 后台管理系统API接口演示  version: 1.0  termsOfServiceUrl: www.bjsxt.com |

## 14.3 文件上传接口开发

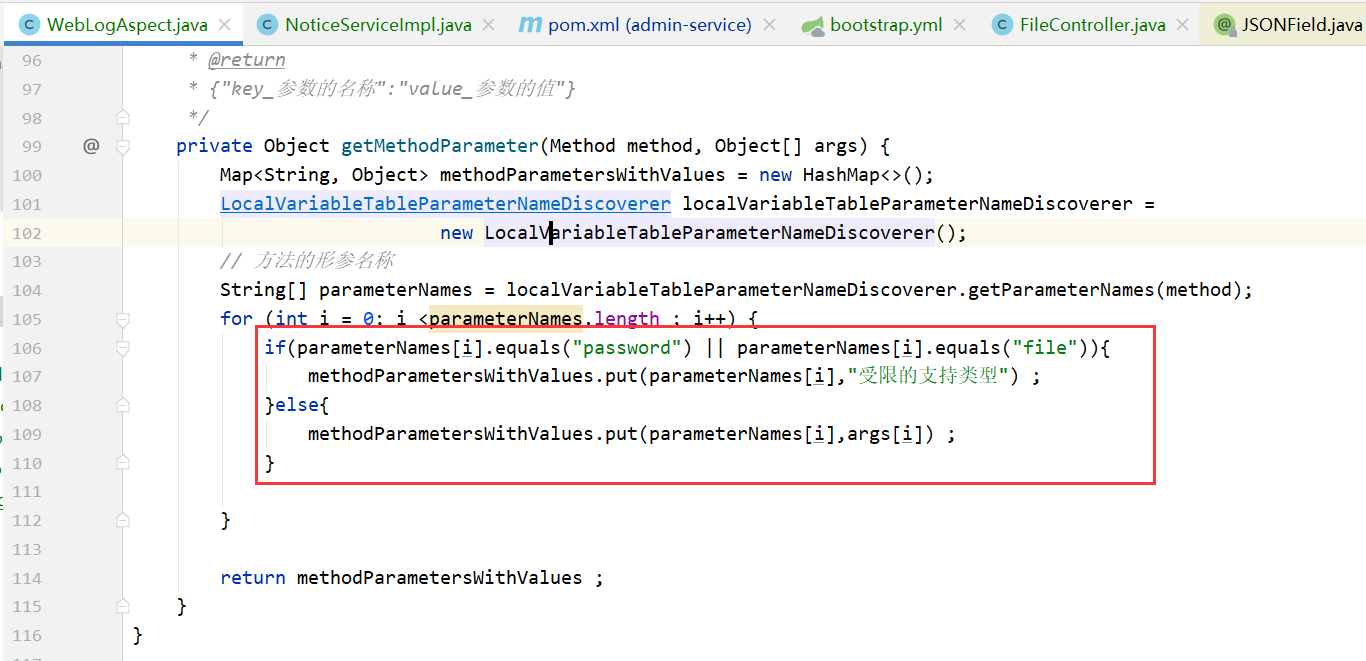
### 14.3.1 新建FileController

|  |
| --- |
| */\*\*  \* 文件上传  \*/* @RestController @Api(tags = "文件上传") public class FileController {   @Autowired  private OSS ossClient ;   @Value("${oss.bucket.name:exchange-imgs}")  private String bucketName ;   @Value("${spring.cloud.alicloud.oss.endpoint}")  private String endPoint ;    @ApiOperation(value = "文件上传")  @PostMapping("/image/AliYunImgUpload")  @ApiImplicitParams({  @ApiImplicitParam(name = "file" ,value = "你要上传的文件")  })  public R<String> fileUpload(@RequestParam("file") MultipartFile file) throws IOException {  */\*\*  \* 3 个参数：  \* 1 bucketName  \* 2 FileName  \* 3 文件的路径  \*/* String fileName = DateUtil.*today*().replaceAll("-","/")+"/"+file.getOriginalFilename() ;  ossClient.putObject(bucketName, fileName, file.getInputStream()); *// 文件上传   // 上传成功后，路径为https://exchange-imgs.oss-cn-beijing.aliyuncs.com/2020/10/10/1588586643\_meitu\_1.jpg* return R.*ok*("https://" + bucketName+ "."+ endPoint +"/"+ fileName) ;   } } |

### 14.3.2 MultipartFile 序列化问题

MultipartFile 使用fastjson 序列化会报错，因此我们需要忽略掉它：

修改WebLogAspect里面的代码：



# 公告管理（新增、修改）

## 15.1 NoticeController

|  |
| --- |
| @PostMapping @ApiOperation(value = "新增一个公告") @ApiImplicitParams({  @ApiImplicitParam(name = "notice" ,value = "notice的json数据") }) @PreAuthorize("hasAuthority('notice\_create')") public R add(@RequestBody @Validated Notice notice){  notice.setStatus(1);  boolean save = noticeService.save(notice); *//* if(save){  return R.*ok*();  }  return R.*fail*("新增公告失败") ;  }    @PatchMapping @ApiOperation(value = "修改一个公告") @ApiImplicitParams({  @ApiImplicitParam(name = "notice" ,value = "notice的json数据") }) @PreAuthorize("hasAuthority('notice\_update')") public R update(@RequestBody @Validated Notice notice){  boolean update = noticeService.updateById(notice);  if(update){  return R.*ok*();  }  return R.*fail*("修改公告失败") ;  } |

## 15.2 Notice

|  |
| --- |
| */\*\*  \* 系统资讯公告信息  \*/* @ApiModel(value="com-bjsxt-domain-Notice") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "notice") public class Notice {  @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="")  private Long id;   */\*\*  \* 标题  \*/* @TableField(value = "title")  @ApiModelProperty(value="标题")  **@NotBlank**  private String title;   */\*\*  \* 简介  \*/* @TableField(value = "description")  @ApiModelProperty(value="简介")   **@NotBlank**  private String description;   */\*\*  \* 作者  \*/* @TableField(value = "author")  @ApiModelProperty(value="作者")   **@NotBlank**  private String author;   */\*\*  \* 文章状态  \*/* @TableField(value = "status")  @ApiModelProperty(value="文章状态")  private Integer status;   */\*\*  \* 文章排序，越大越靠前  \*/* @TableField(value = "sort")  @ApiModelProperty(value="文章排序，越大越靠前")  **@NotNull**  private Integer sort;   */\*\*  \* 内容  \*/* @TableField(value = "content")  @ApiModelProperty(value="内容")   **@NotBlank**  private String content;   */\*\*  \* 最后修改时间  \*/* @TableField(value = "last\_update\_time",**fill = FieldFill.*INSERT\_UPDATE***)  @ApiModelProperty(value="最后修改时间")  private Date lastUpdateTime;   */\*\*  \* 创建日期  \*/* @TableField(value = "created" ,**fill = FieldFill.*INSERT***)  @ApiModelProperty(value="创建日期")  private Date created; } |

# 资源配置

## 16.1 资源配置的API接口



## 16.2 WebConfigController

|  |
| --- |
| @RestController @Api(tags = "webConfig的控制器") @RequestMapping("/webConfigs") public class WebConfigController {   @Autowired  private WebConfigService webConfigService ;   @GetMapping  @ApiOperation(value ="分页查询我们的webConfig")  @ApiImplicitParams({  @ApiImplicitParam(name = "name",value = "webConfig的名称") ,  @ApiImplicitParam(name = "type",value = "webConfig的类型") , *// nteger current ,Integer siz* @ApiImplicitParam(name = "current",value = "当前页") ,  @ApiImplicitParam(name = "size",value = "每页的大小") ,  })  @PreAuthorize("hasAuthority('web\_config\_query')")  public R<Page<WebConfig>> findByPage(@ApiIgnore Page<WebConfig> page , String name , String type ){  Page<WebConfig> webConfigPage = webConfigService.findByPage(page,name,type) ;  return R.*ok*(webConfigPage) ;  }    @PostMapping  @ApiOperation(value = "新增一个WebConfig")  @ApiImplicitParams({  @ApiImplicitParam(name = "webConfig" ,value = "webConfig 的json数据")  })  @PreAuthorize("hasAuthority('web\_config\_create')")  public R add(@RequestBody @Validated WebConfig webConfig){  boolean save = webConfigService.save(webConfig);  if(save){  return R.*ok*() ;  }  return R.*fail*("新增失败") ;  }    @PatchMapping  @ApiOperation(value = "修改一个WebConfig")  @ApiImplicitParams({  @ApiImplicitParam(name = "webConfig" ,value = "webConfig 的json数据")  })  @PreAuthorize("hasAuthority('web\_config\_update')")  public R update(@RequestBody @Validated WebConfig webConfig){  boolean save = webConfigService.updateById(webConfig);  if(save){  return R.*ok*() ;  }  return R.*fail*("修改失败") ;  }    @PostMapping("/delete")  @ApiOperation(value = "删除WebConfig")  @ApiImplicitParams({  @ApiImplicitParam(name = "ids" ,value = "要删除的数据的ids")  })  @PreAuthorize("hasAuthority('web\_config\_delete')")  public R update(@RequestBody String[] ids){  if(ids==null || ids.length==0){  return R.*fail*("删除时需要给删除数据的Id") ;  }  boolean b = webConfigService.removeByIds(Arrays.*asList*(ids));  if(b){  return R.*ok*() ;  }  return R.*fail*("删除失败") ;  }  } |

## 16.3 WebConfig

|  |
| --- |
| */\*\*  \* 网站配置信息  \*/* @ApiModel(value="com-bjsxt-domain-WebConfig") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "web\_config") public class WebConfig {  */\*\*  \* Id  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="Id")  private Long id;   */\*\*  \* 分组, LINK\_BANNER ,WEB\_BANNER  \*/* @TableField(value = "type")  @ApiModelProperty(value="分组, LINK\_BANNER ,WEB\_BANNER")   **@NotBlank**  private String type;   */\*\*  \* 名称  \*/* @TableField(value = "name")  @ApiModelProperty(value="名称")  **@NotBlank**  private String name;   */\*\*  \* 值  \*/* @TableField(value = "value")  @ApiModelProperty(value="值")   **@NotBlank**  private String value;   */\*\*  \* 权重  \*/* @TableField(value = "sort")  @ApiModelProperty(value="权重")  private Short sort;   */\*\*  \* 创建时间  \*/* **@TableField(value = "created" ,fill = FieldFill.*INSERT*)**  @ApiModelProperty(value="创建时间")  private Date created;   */\*\*  \* 超链接地址  \*/* @TableField(value = "url")  @ApiModelProperty(value="超链接地址")  private String url;   */\*\*  \* 是否使用 0 否 1是  \*/* @TableField(value = "status")  @ApiModelProperty(value="是否使用 0 否 1是")  private Integer status; *// Boolean 时状态在前端显示不正常* } |

# 客服工单

## 17.1 API接口的展示



## 17.2 WorkIssueController

|  |
| --- |
| @RestController @RequestMapping("/workIssues") @Api(tags = "客户工单的控制器") public class WorkIssueController {    @Autowired  private WorkIssueService workIssueService;    @GetMapping  @ApiOperation(value = "分页条件查询工单")  @ApiImplicitParams({  @ApiImplicitParam(name = "status", value = "工单当前的处理状态"),  @ApiImplicitParam(name = "startTime", value = "工单创建的起始时间"),  @ApiImplicitParam(name = "endTime", value = "工单创建的截至时间"),  @ApiImplicitParam(name = "current", value = "当前页"),  @ApiImplicitParam(name = "size", value = "每页显示的条数"),  })  @PreAuthorize("hasAuthority('work\_issue\_query')")  public R<Page<WorkIssue>> findByPage(@ApiIgnore Page<WorkIssue> page, Integer status, String startTime, String endTime) {  page.addOrder(OrderItem.*desc*("last\_update\_time"));  Page<WorkIssue> workIssuePage = workIssueService.findByPage(page, status, startTime, endTime);  return R.*ok*(workIssuePage);  }     @PatchMapping  @ApiOperation(value = "回复某个工单")  @ApiImplicitParams({  @ApiImplicitParam(name = "id", value = "工单的ID"),  @ApiImplicitParam(name = "answer", value = "工单的answer"),  })  @PreAuthorize("hasAuthority('work\_issue\_update')")  public R work\_issue\_update(Long id ,String answer) {  WorkIssue workIssue = new WorkIssue();  workIssue.setId(id);  workIssue.setAnswer(answer);  boolean updateById = workIssueService.updateById(workIssue);  if(updateById){  return R.*ok*() ;  }  return R.*fail*("回复失败") ;  } |

## 17.3 WorkIssueService

|  |
| --- |
| public interface WorkIssueService extends IService<WorkIssue>{    */\*\*  \* 条件分页查询工单列表  \* @param page  \* 分页参数  \* @param status  \* 工单的状态  \* @param startTime  \* 查询的工单创建起始时间  \* @param endTime  \* 查询的工单创建截至时间  \* @return  \*/* Page<WorkIssue> findByPage(Page<WorkIssue> page, Integer status, String startTime, String endTime); } |

## 17.4 WorkIssueServiceImpl

|  |
| --- |
| @Service public class WorkIssueServiceImpl extends ServiceImpl<WorkIssueMapper, WorkIssue> implements WorkIssueService{   */\*\*  \* 条件分页查询工单列表  \*  \* @param page 分页参数  \* @param status 工单的状态  \* @param startTime 查询的工单创建起始时间  \* @param endTime 查询的工单创建截至时间  \* @return  \*/* @Override  public Page<WorkIssue> findByPage(Page<WorkIssue> page, Integer status, String startTime, String endTime) {  return page(page, new LambdaQueryWrapper<WorkIssue>()  .eq(status!=null ,WorkIssue::getStatus,status)  .between(  !StringUtils.*isEmpty*(startTime) && !StringUtils.*isEmpty*(endTime),  WorkIssue::getCreated,  startTime,endTime+" 23:59:59" )  );  } } |

## 17.5 WorkIssue

|  |
| --- |
| */\*\*  \* 工单记录  \*/* @ApiModel(value="com-bjsxt-domain-WorkIssue") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "work\_issue") public class WorkIssue {  */\*\*  \* 主键  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="主键")  private Long id;   */\*\*  \* 用户id(提问用户id)  \*/* @TableField(value = "user\_id")  @ApiModelProperty(value="用户id(提问用户id)")  private Long userId;   */\*\*  \* 回复人id  \*/* @TableField(value = "answer\_user\_id")  @ApiModelProperty(value="回复人id")  private Long answerUserId;   */\*\*  \* 回复人名称  \*/* @TableField(value = "answer\_name")  @ApiModelProperty(value="回复人名称")  private String answerName;   */\*\*  \* 工单内容  \*/* @TableField(value = "question")  @ApiModelProperty(value="工单内容")  private String question;   */\*\*  \* 回答内容  \*/* @TableField(value = "answer")  @ApiModelProperty(value="回答内容")  private String answer;   */\*\*  \* 状态：1-待回答；2-已回答；  \*/* @TableField(value = "status")  @ApiModelProperty(value="状态：1-待回答；2-已回答；")  **private Integer status;**   */\*\*  \* 修改时间  \*/* **@TableField(value = "last\_update\_time" ,fill = FieldFill.*INSERT\_UPDATE*)**  @ApiModelProperty(value="修改时间")  private Date lastUpdateTime;   */\*\*  \* 创建时间  \*/* **@TableField(value = "created",fill = FieldFill.*INSERT*)**  @ApiModelProperty(value="创建时间")  private Date created;    **@TableField(exist = false)  @ApiModelProperty(value = "创建工单的用户名称")  public String username ="测试用户" ;    @TableField(exist = false)  @ApiModelProperty(value = "创建工单的用户真实名称")  private String realName= "测试用户" ;** } |

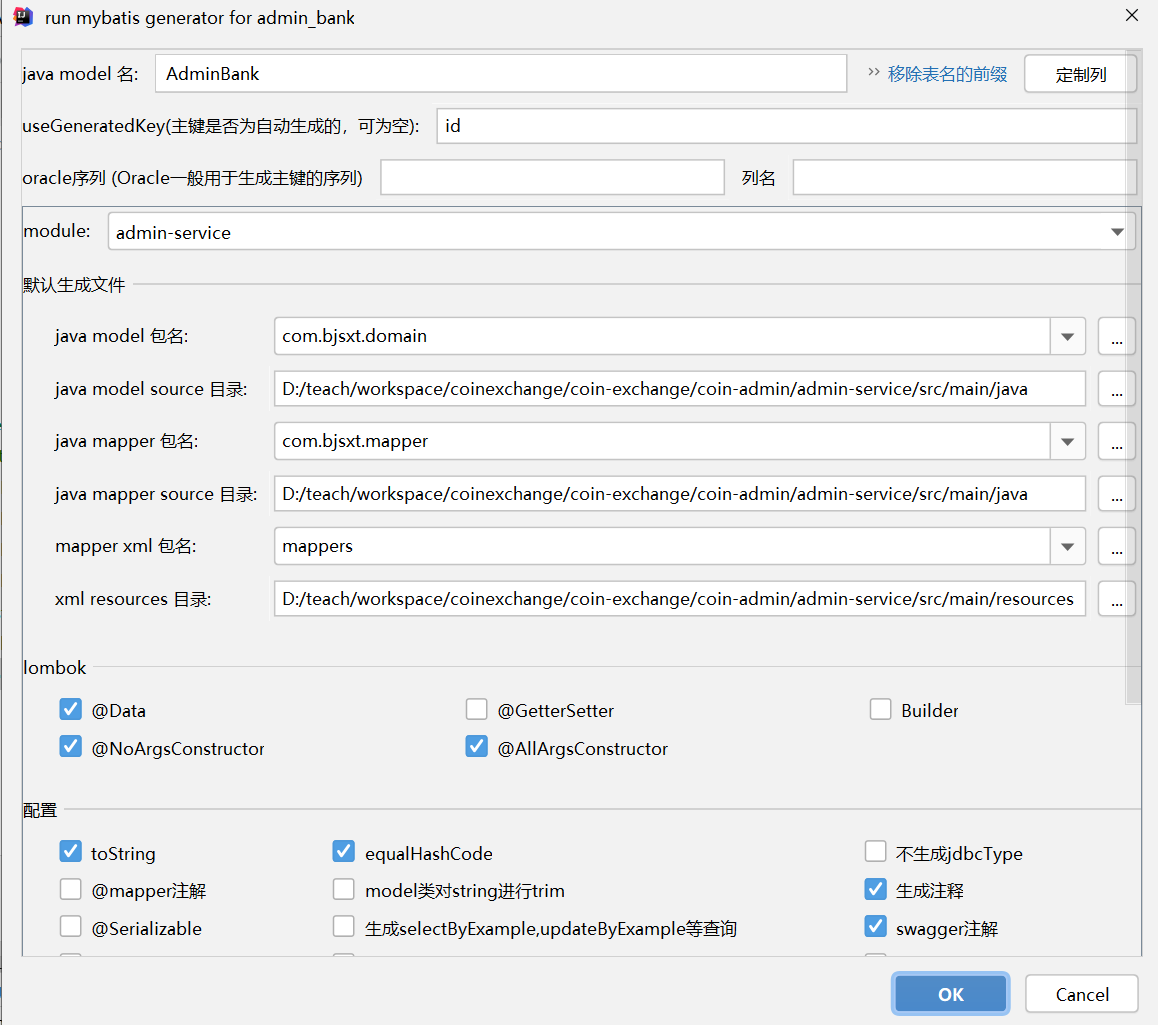
## 17.6 WorkIssue 回复时没有设置回复人



# 公司银行卡管理

## 18.1 代码的生成（复习）

保持默认就可以啦。



## 18.2 Api接口的展示



## 18.3 AdminBankController

|  |
| --- |
| @RestController @RequestMapping("/adminBanks") @Api(tags = "公司银行卡的配置") public class AdminBankController {    @Autowired  private AdminBankService adminBankService;    @GetMapping  @ApiOperation(value = "条件查询公司银行卡")  @ApiImplicitParams({  @ApiImplicitParam(name = "bankCard" ,value = "公司的银行卡") ,  @ApiImplicitParam(name = "current" ,value = "当前页") ,  @ApiImplicitParam(name = "size" ,value = "每页显示的条数")  })  @PreAuthorize("hasAuthority('admin\_bank\_query')")  public R<Page<AdminBank>> findByPage(@ApiIgnore Page<AdminBank> page , String bankCard){  Page<AdminBank> adminBankPage = adminBankService.findByPage(page, bankCard) ;  return R.*ok*(adminBankPage) ;  }    @PostMapping  @ApiOperation(value = "新增一个银行卡")  @ApiImplicitParams({  @ApiImplicitParam(name = "adminBank" ,value = "adminBank json数据")  })  @PreAuthorize("hasAuthority('admin\_bank\_create')")  public R add(@RequestBody @Validated AdminBank adminBank){  boolean save = adminBankService.save(adminBank);  if(save){  return R.*ok*() ;  }  return R.*fail*("新增失败") ;  }    @PatchMapping  @ApiOperation(value = "修改一个银行卡")  @ApiImplicitParams({  @ApiImplicitParam(name = "adminBank" ,value = "adminBank json数据")  })  @PreAuthorize("hasAuthority('admin\_bank\_update')")  public R update(@RequestBody @Validated AdminBank adminBank){  boolean updateById = adminBankService.updateById(adminBank);  if(updateById){  return R.*ok*() ;  }  return R.*fail*("修改失败") ;  }    @PostMapping("/adminUpdateBankStatus")  @ApiOperation(value = "修改银行卡的状态")  @ApiImplicitParams({  @ApiImplicitParam(name = "bankId" ,value = "要修改的银行卡的ID"),  @ApiImplicitParam(name = "status" ,value = "要修改为的状态")  })  @PreAuthorize("hasAuthority('admin\_bank\_update')")  public R changeStatus(Long bankId ,Byte status){  AdminBank adminBank = new AdminBank();  adminBank.setId(bankId);  adminBank.setStatus(status);  boolean updateById = adminBankService.updateById(adminBank);  if(updateById){  return R.*ok*() ;  }  return R.*fail*("状态修改失败") ;  }  } |

## 18.3 AdminBankServie

|  |
| --- |
| public interface AdminBankService extends IService<AdminBank>{    */\*\*  \* 条件分页查询公司银行卡  \* @param page  \* 分页参数  \* @param bankCard  \* 银行卡卡号  \* @return  \*/* Page<AdminBank> findByPage(Page<AdminBank> page, String bankCard); } |

## 18.3 AdminBankServiceImp

|  |
| --- |
| @Service public class AdminBankServiceImpl extends ServiceImpl<AdminBankMapper, AdminBank> implements AdminBankService{   */\*\*  \* 条件分页查询公司银行卡  \*  \* @param page 分页参数  \* @param bankCard 银行卡卡号  \* @return  \*/* @Override  public Page<AdminBank> findByPage(Page<AdminBank> page, String bankCard) {  return page(page,new LambdaQueryWrapper<AdminBank>()  .like(!StringUtils.*isEmpty*(bankCard),AdminBank::getBankCard ,bankCard));  } } |

## 18.3 AdminBank

|  |
| --- |
| */\*\*  \* 人民币充值卡号管理  \*/* @ApiModel(value="com-bjsxt-domain-AdminBank") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "admin\_bank") public class AdminBank {  */\*\*  \* 主键  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="主键")  private Long id;   */\*\*  \* 开户人姓名  \*/* @TableField(value = "name")  @ApiModelProperty(value="开户人姓名")   **@NotBlank**  private String name;   */\*\*  \* 开户行名称  \*/* @TableField(value = "bank\_name")  @ApiModelProperty(value="开户行名称")   **@NotBlank**  private String bankName;   */\*\*  \* 卡号  \*/* @TableField(value = "bank\_card")  @ApiModelProperty(value="卡号")  **@NotBlank**  private String bankCard;   */\*\*  \* 充值转换换币种ID  \*/* @TableField(value = "coin\_id")  @ApiModelProperty(value="充值转换换币种ID")  private Long coinId;   */\*\*  \* 币种名称  \*/* @TableField(value = "coin\_name")  @ApiModelProperty(value="币种名称")  private String coinName;   */\*\*  \* 状态：0-无效；1-有效；  \*/* @TableField(value = "status")  @ApiModelProperty(value="状态：0-无效；1-有效；")  private Byte status; |

# 参数管理

## 19.1 API接口的展示



## 19.2 ConfigController

|  |
| --- |
| @RestController @RequestMapping("/configs") @Api(tags = "后台参数配置") public class ConfigController {    @Autowired  private ConfigService configService ;    @GetMapping  @ApiOperation(value = "条件分页查询后台参数")  @ApiImplicitParams({  @ApiImplicitParam(name = "type" ,value = "后台参数类型"),  @ApiImplicitParam(name = "code" ,value = "后台参数code"),  @ApiImplicitParam(name = "name" ,value = "后台参数名称"),  @ApiImplicitParam(name = "current" ,value = "当前页"),  @ApiImplicitParam(name = "size" ,value = "每页显示的条数"),   })  @PreAuthorize("hasAuthority('config\_query')")  public R<Page<Config>> findByPage(@ApiIgnore Page<Config> page , String type , String code , String name){  Page<Config> configPage = configService.findByPage(page,type,name,code) ;  return R.*ok*(configPage) ;  }    @PostMapping  @ApiOperation(value = "新增一个参数")  @ApiImplicitParams({  @ApiImplicitParam(name = "config",value = "config 的json数据")  })  @PreAuthorize("hasAuthority('config\_create')")  public R add(@RequestBody @Validated Config config){  boolean save = configService.save(config);  if(save){  return R.*ok*() ;  }  return R.*fail*("新增失败") ;  }     @PatchMapping  @ApiOperation(value = "修改一个参数")  @ApiImplicitParams({  @ApiImplicitParam(name = "config",value = "config 的json数据")  })  @PreAuthorize("hasAuthority('config\_update')")  public R update(@RequestBody @Validated Config config){  boolean update = configService.updateById(config);  if(update){  return R.*ok*() ;  }  return R.*fail*("修改失败") ;  }  } |

## 19.3 ConfigService

|  |
| --- |
| public interface ConfigService extends IService<Config>{    */\*\*  \* 条件分页查询参数  \* @param page  \*分页参数  \* @param type  \* 类型  \* @param name  \* 参数名称  \* @param code  \* 参数Code  \* @return  \*/* Page<Config> findByPage(Page<Config> page, String type, String name, String code); } |

## 19.4 ConfigServiceImpl

|  |
| --- |
| @Service public class ConfigServiceImpl extends ServiceImpl<ConfigMapper, Config> implements ConfigService{   */\*\*  \* 条件分页查询参数  \*  \* @param page 分页参数  \* @param type 类型  \* @param name 参数名称  \* @param code 参数Code  \* @return  \*/* @Override  public Page<Config> findByPage(Page<Config> page, String type, String name, String code) {  return page(page,new LambdaQueryWrapper<Config>()  .like(!StringUtils.*isEmpty*(type),Config::getType ,type)  .like(!StringUtils.*isEmpty*(name),Config::getName ,name)  .like(!StringUtils.*isEmpty*(code),Config::getCode ,code)  );  } } |

## 19.5 Config

|  |
| --- |
| */\*\*  \* 平台配置信息  \*/* @ApiModel(value="com-bjsxt-domain-Config") @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "config") public class Config {  */\*\*  \* 主键  \*/* @TableId(value = "id", type = IdType.*AUTO*)  @ApiModelProperty(value="主键")  private Long id;   */\*\*  \* 配置规则类型  \*/* @TableField(value = "type")  @ApiModelProperty(value="配置规则类型")   **@NotBlank**  private String type;   */\*\*  \* 配置规则代码  \*/* @TableField(value = "code")  @ApiModelProperty(value="配置规则代码")   **@NotBlank**  private String code;   */\*\*  \* 配置规则名称  \*/* @TableField(value = "name")  @ApiModelProperty(value="配置规则名称")   **@NotBlank**  private String name;   */\*\*  \* 配置规则描述  \*/* **@TableField(value = "`desc`")**  @ApiModelProperty(value="配置规则描述")  private String desc;   */\*\*  \* 配置值  \*/* @TableField(value = "value")  @ApiModelProperty(value="配置值")   **@NotBlank**  private String value;   */\*\*  \* 创建时间  \*/* **@TableField(value = "created" ,fill = FieldFill.*INSERT*)**  @ApiModelProperty(value="创建时间")  private Date created; |