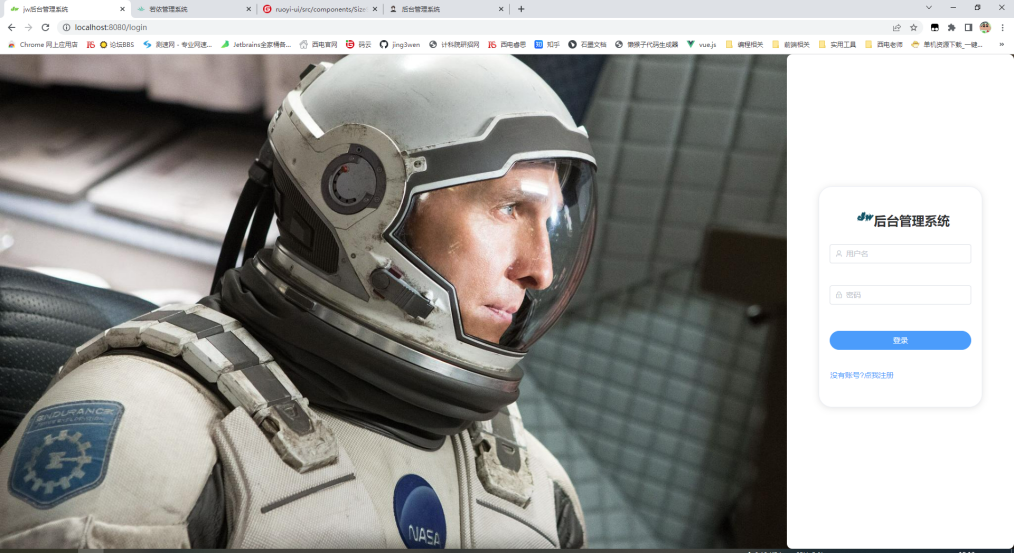
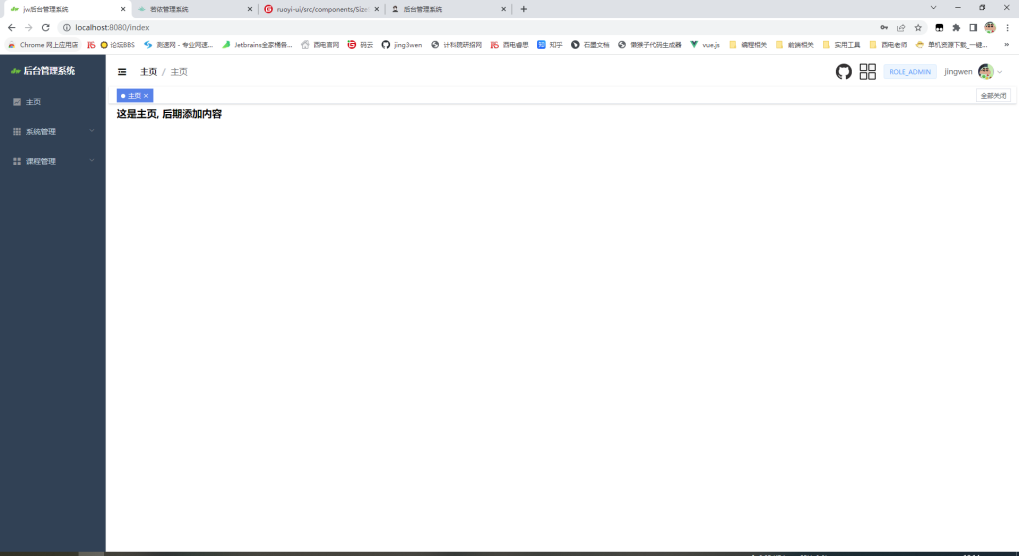
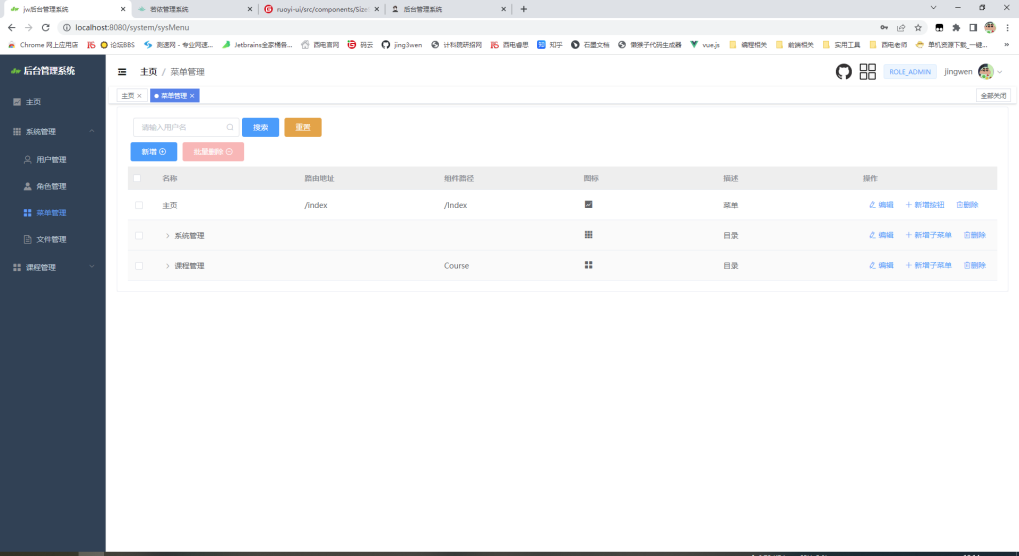
一．主体框架预览

前端主体框架已经搭建完成，现在开始后端及进一步搭建







二．数据库设计

1.用户表设计

考虑到后续可能有高并发的设计需求，在用户数据库方面的修改：①将表拆分，②设计多个相同的用户表用来存储不同的用户。

我选择第②种方法，后续的项目可能要扩展成spring cloud模块化项目，模块化项目中各个模块有自己的专属用户，所以采用多个名字不同的用户表来进行存储各自模块的用户，这样操作还能将模块内部的用户和系统用户直接拆开，组成一个新的数据库（模块数据库）

系统用户表 sys\_user utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 用户ID |
| dept\_id | INT |  | NULL | 部门ID |
| username | VARCHAR(100) | 非空 |  | 用户名（不能同名） |
| password | VARCHAR(100) |  | ‘’ | 密码 |
| nickname | VARCHAR(100) | 非空 |  | 用户昵称 |
| user\_type | VARCHAR(50) |  | ‘’ | 用户类型(模块名) |
| email | VARCHAR(50) |  | ‘’ | 邮箱 |
| phone | VARCHAR(11) |  | ‘’ | 电话号码 |
| sex | VARCHAR(5) |  | ‘’ | 性别 |
| avatar | VARCHAR(100) |  | ‘’ | 头像地址 |
| remark | VARCHAR(500) |  | NULL | 备注 |
| last\_login\_time | DATETIME |  | NULL | 最后一次登录时间 |
| status | CHAR(1) |  | ‘1’ | 状态(1正常 0停用) |
| is\_deleted | CHAR(1) |  | ‘0’ | 是否删除(0代表存在 1代表删除) |
| create\_by | VARCHAR(100) |  | ‘’ | 创建者 |
| create\_time | DATETIME |  | NULL | 创建时间 |
| update\_by | VARCHAR(100) |  | ‘’ | 更新者 |
| update\_time | DATETIME |  | NULL | 更新时间 |
|  |  |  |  |  |

此处的create\_by属性存储的为用户名，不建议用id，因为考虑到有的用户被删除了，查询会出一些问题（虽然是假删除）

角色表 sys\_role utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 角色ID |
| role\_code | VARCHAR(100) | 非空 |  | 角色标识(不能同名) |
| role\_name | VARCHAR(100) | 非空 |  | 角色名 |
| role\_sort | INT |  | ‘0’ | 显示顺序 |
| remark | VARCHAR(500) |  | NULL | 备注 |
| status | CHAR(1) |  | ‘1’ | 状态(1正常 0停用) |
| is\_deleted | CHAR(1) |  | ‘0’ | 是否删除(0代表存在 1代表删除) |
| create\_by | VARCHAR(100) |  | ‘’ | 创建者 |
| create\_time | DATETIME |  | NULL | 创建时间 |
| update\_by | VARCHAR(100) |  | ‘’ | 更新者 |
| update\_time | DATETIME |  | NULL | 更新时间 |
|  |  |  |  |  |

角色标识指的是admin，test\_user之类的

用户角色关联表 sys\_user\_role utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| user\_id | INT | 主键，非空 |  | 用户ID |
| role\_id | INT | 主键，非空 |  | 角色ID |
|  |  |  |  |  |

为什么不考虑加id作为主键，而是直接使用user\_id和role\_id两个属性共同作为主键？ 此表后期可能涉及到大量的添加删除操作，若采用id作为主键，随着写入次数变多，id值会变很大

菜单表 sys\_menu utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 菜单ID |
| parent\_id | INT |  | 0 | 父菜单ID |
| menu\_name | VARCHAR(100) | 非空 |  | 菜单名称 |
| menu\_type | VARCHAR(20) |  | ‘’ | 菜单类别（目录，菜单，按钮） |
| icon | VARCHAR(100) |  | ‘#’ | 菜单图标 |
| path | VARCHAR(300) |  | ‘’ | 路由地址 |
| component | VARCHAR(300) |  | NULL | 组件路径 |
| perms | VARCHAR(300) |  | NULL | 权限标识 |
| is\_frame | INT |  | ‘0’ | 是否为外链（0否 1是） |
| menu\_sort | INT |  | ‘0’ | 显示顺序 |
| visible | CHAR(1) |  | ‘1’ | 菜单状态（1显示 0隐藏） |
| remark | VARCHAR(500) |  | NULL | 备注 |
| is\_deleted | CHAR(1) |  | ‘0’ | 是否删除(0代表存在 1代表删除) |
| create\_by | VARCHAR(100) |  | ‘’ | 创建者 |
| create\_time | DATETIME |  | NULL | 创建时间 |
| update\_by | VARCHAR(100) |  | ‘’ | 更新者 |
| update\_time | DATETIME |  | NULL | 更新时间 |
|  |  |  |  |  |

角色菜单关联表 sys\_role\_menu utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| role\_id | INT | 主键，非空 |  | 角色ID |
| menu\_id | INT | 主键，非空 |  | 菜单ID |
|  |  |  |  |  |

系统操作日志表sys\_operation\_log utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 操作日志ID |
| opt\_module | VARCHAR(100) |  | ‘’ | 操作模块名 |
| opt\_type | VARCHAR(10) |  | ‘’ | 操作类型 |
| request\_url | VARCHAR(300) |  | ‘’ | 请求url |
| request\_method | VARCHAR(10) |  | ‘’ | 请求方式 |
| opt\_method | VARCHAR(100) |  | ‘’ | 方法名称 |
| opt\_desc | VARCHAR(200) |  | ‘’ | 操作描述 |
| request\_param | LONGTEXT |  | ‘’ | 请求参数 |
| response\_result | LONGTEXT |  | ‘’ | 响应参数 |
| status | INT |  | 0 | 操作状态（0正常 1异常） |
| error\_msg | LONGTEXT |  | ‘’ | 错误信息 |
| opt\_user | VARCHAR(100) |  | ‘’ | 操作人 |
| opt\_ip | VARCHAR(128) |  | ‘’ | 操作ip |
| opt\_location | VARCHAR(300) |  | ‘’ | 操作地址 |
| create\_time | DATETIME |  | NULL | 创建时间 |

登录日志表sys\_login\_log utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 登录日志ID |
| username | VARCHAR(100) |  | ‘’ | 登录账户 |
| login\_ip | VARCHAR(128) |  | ‘’ | 登录ip |
| login\_location | VARCHAR(300) |  | ‘’ | 操作地址 |
| login\_browser | VARCHAR(50) |  |  | 登录浏览器 |
| login\_os | VARCHAR(50) |  |  | 操作系统 |
| status | INT |  |  | 登录状态（0正常 1异常） |
| login\_msg | VARCHAR(2000) |  | NULL | 登录结果 |
| create\_time | CREATETIME |  | NULL | 创建时间 |

dl\_face\_database 人脸库 utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 人脸ID |
| parent\_id | INT |  | 0 | 父ID |
| face\_name | VARCHAR(100) |  | ‘’ | 人脸名 |
| face\_sex | VARCHAR(5) |  | ‘0’ | 人脸性别 |
| face\_type | VARCHAR(100) |  | ‘’ | 人脸类型 |
| image\_name | VARCHAR(200) |  | ‘’ | 图片名 |
| image\_address | VARCHAR(1024) |  | ‘’ | 图片地址 |
| remark | VARCHAR(500) |  | NULL | 备注 |
| status | CHAR(1) |  | ‘1’ | 状态(1正常 0停用) |
| is\_deleted | CHAR(1) |  | ‘0’ | 是否删除(0代表存在 1代表删除) |
| create\_by | VARCHAR(100) |  | ‘’ | 创建者 |
| create\_time | DATETIME |  | NULL | 创建时间 |
| update\_by | VARCHAR(100) |  | ‘’ | 更新者 |
| update\_time | DATETIME |  | NULL | 更新时间 |
|  |  |  |  |  |

dl\_face\_detect\_file 检测文件表 utf8mb4 **COLLATE** utf8mb4\_0900\_ai\_ci

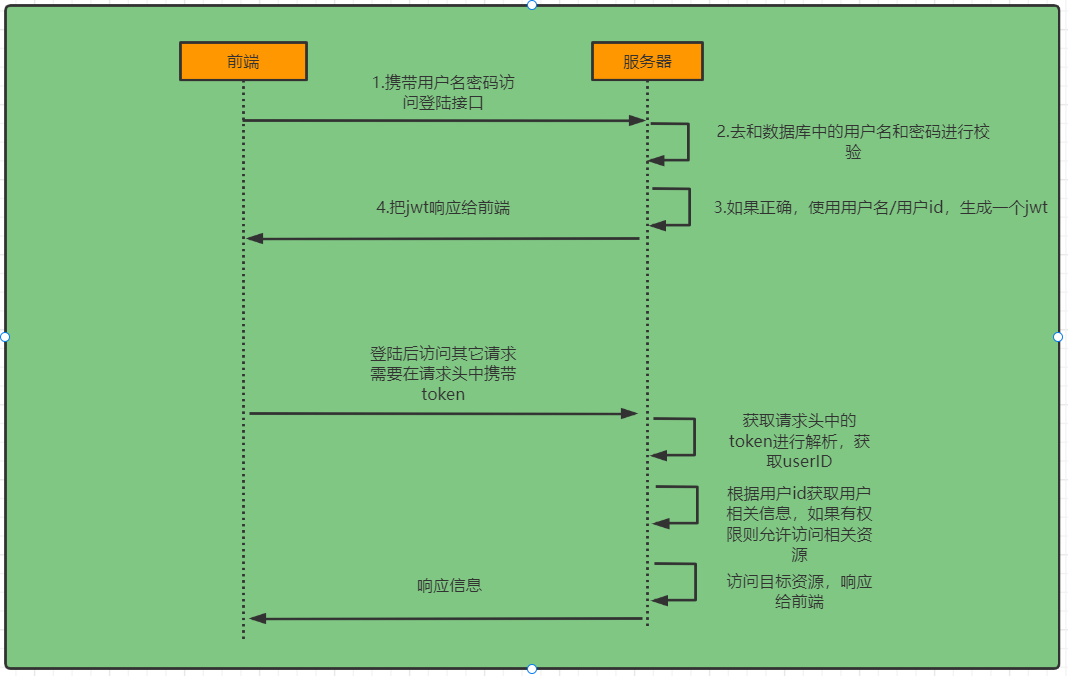
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 数据类型 | 存储参数 | 默认值 | 注备 |
| id | INT | 主键，非空，自增， |  | 检测文件ID |
| file\_name | VARCHAR(100) |  | ‘’ | 检测文件名 |
| file\_address | VARCHAR(1024) |  | ‘’ | 检测文件地址 |
| file\_type | VARCHAR(10) |  | ‘’ | 检测文件类别 |
| detect\_status | INT |  | 0 | 检测状态(0：未检测，1：检测中，2：检测完成，-1：检测失败) |
| save\_result | CHAR(1) |  | 1 | 是否保存检测结果(0：否，1：是) |
| result\_file\_address | VARCHAR(1024) |  | ‘’ | 检测结果文件地址 |
| result\_msg | VARCHAR(4096) |  | ‘’ | 检测结果描述 |
| remark | VARCHAR(500) |  | NULL | 备注 |
| is\_deleted | CHAR(1) |  | ‘0’ | 是否删除(0代表存在 1代表删除) |
| create\_by | VARCHAR(100) |  | ‘’ | 创建者 |
| create\_time | DATETIME |  | NULL | 创建时间 |
| update\_by | VARCHAR(100) |  | ‘’ | 更新者 |
| update\_time | DATETIME |  | NULL | 更新时间 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

三．系统设计

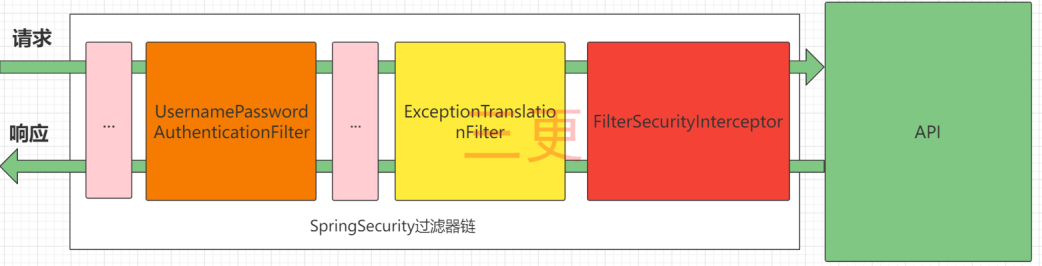
3.1.用户登录

3.1.1.原理：

基于SpringSecurity来完成登录功能



SpringSecurity过滤器链

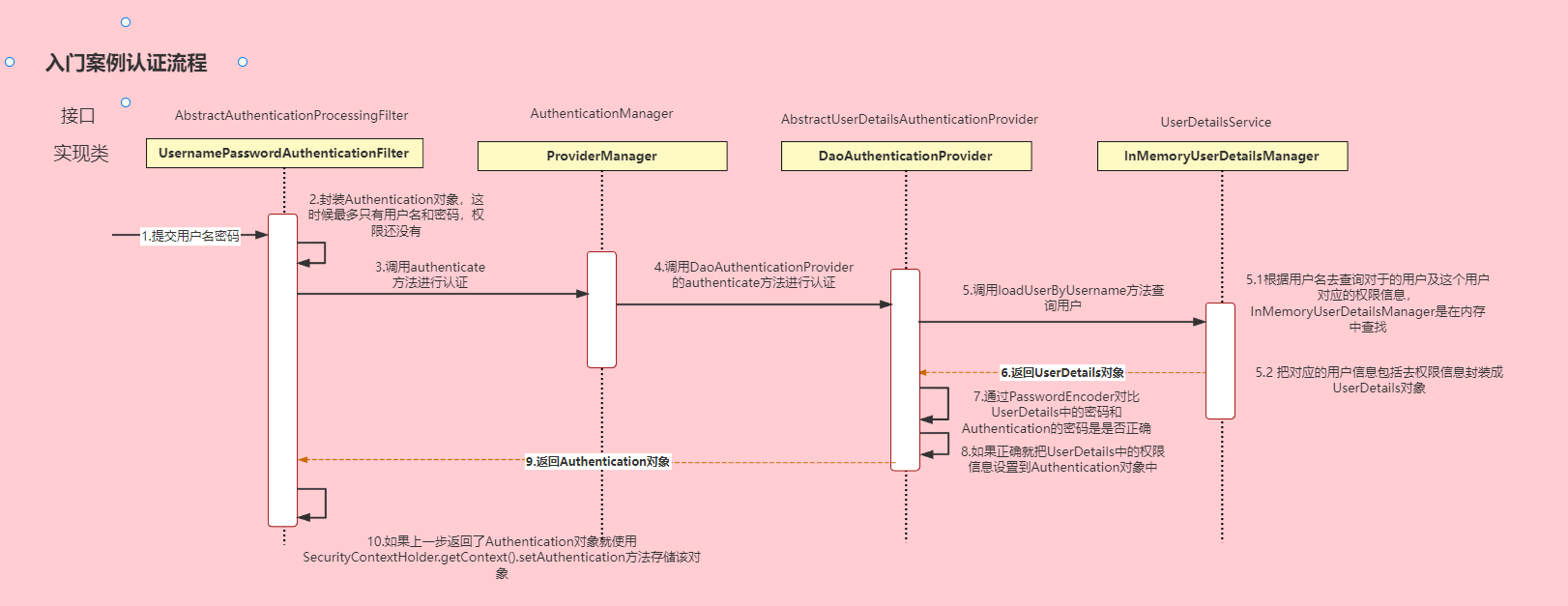


UsernamePasswordAuthenticationFilter：负责处理我们在登陆页面填写了用户名密码后的登陆请求。入门案例的认证工作主要有它负责。

ExceptionTranslationFilter：处理过滤器链中抛出的任何AccessDeniedException和AuthenticationException 。

FilterSecurityInterceptor：负责权限校验的过滤器。

认证流程详解



Authentication接口: 它的实现类，表示当前访问系统的用户，封装了用户相关信息。

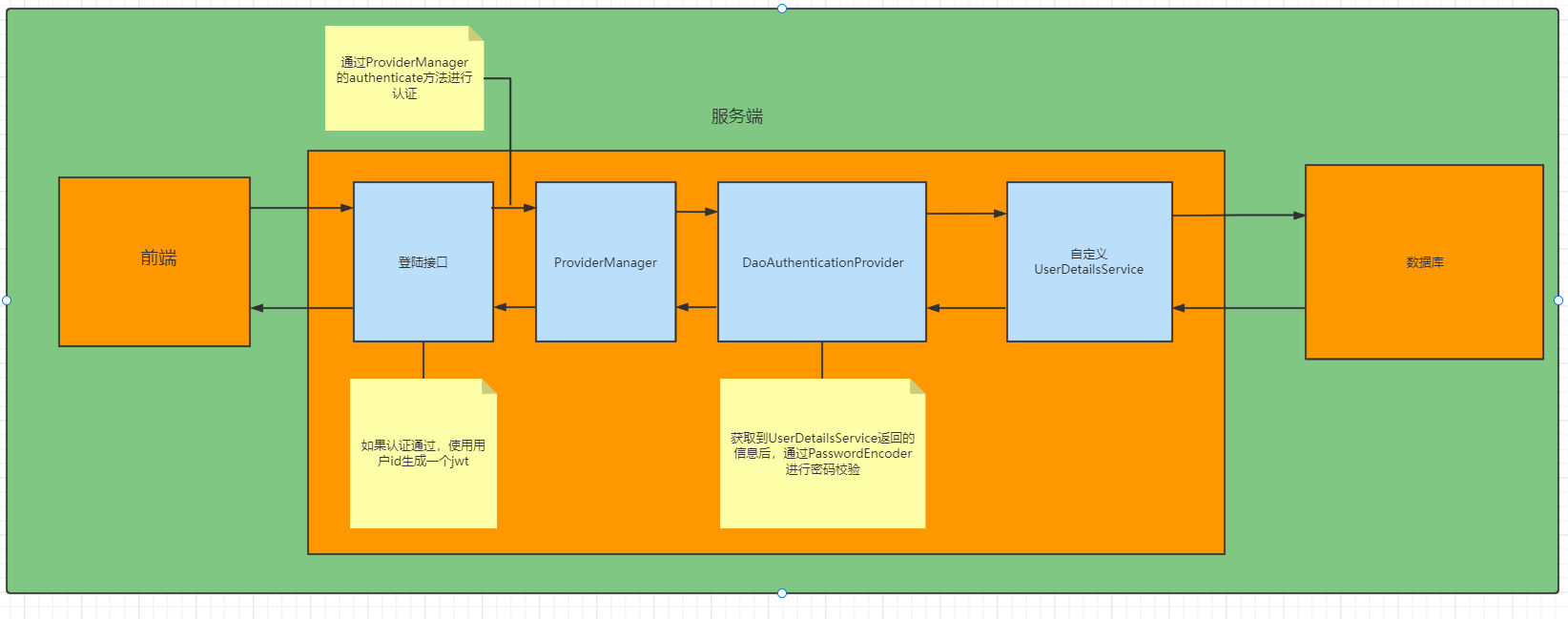
AuthenticationManager接口：定义了认证Authentication的方法

UserDetailsService接口：加载用户特定数据的核心接口。里面定义了一个根据用户名查询用户信息的方法。

UserDetails接口：提供核心用户信息。通过UserDetailsService根据用户名获取处理的用户信息要封装成UserDetails对象返回。然后将这些信息封装到Authentication对象中。

3.1.2 设计

登录流程设计



第一步：

自定义登录接口，让SpringSecurity对这个自定义登录接口放行,让用户访问这个接口的时候不用登录也能访问。

在SecurityConfig中配置把AuthenticationManager注入容器，在自定义登录接口就可以调用AuthenticationManager的authenticate方法。

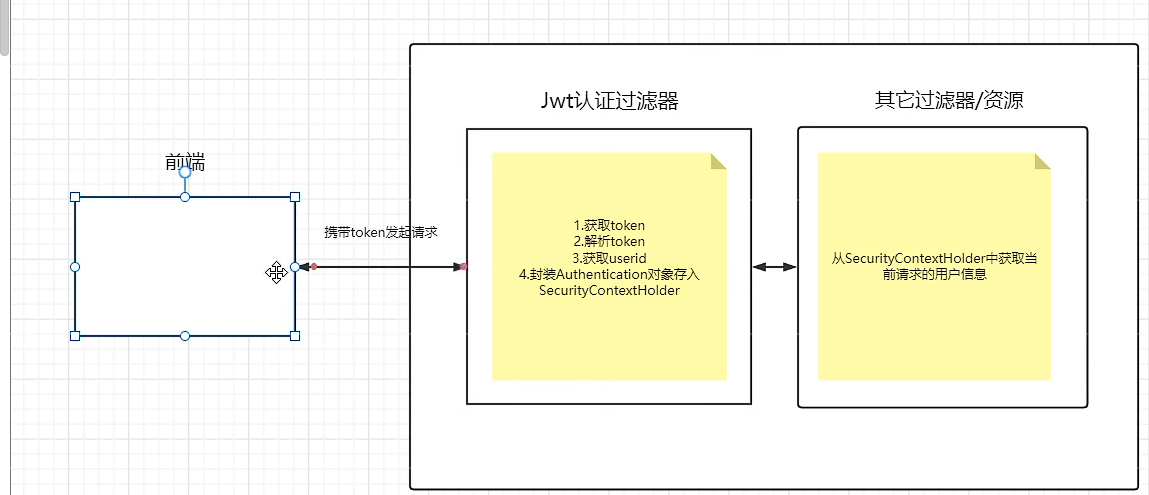
authenticate方法会调用UserDetailsService中的loadUserByUsername()方法来校验username和password。所以真正从数据库中查找用户名和密码的方法位于实现UserDetailsService的loadUserByUsername()方法

。认证成功的话要生成一个jwt，放入响应中返回。并且为了让用户下回请求时能通过jwt识别出具体的是哪个用户，我们需要把用户信息存入redis，可以把用户id作为key。

第二步：

自定义UserDetailsService，在这个实现类中去查询数据库

校验



①定义Jwt认证过滤器

我们需要自定义一个过滤器，这个过滤器会去获取请求头中的token，对token进行解析。看能否解析成功。若token解析成功，使用token的最后一段去redis中获取对应的UserDetailsVO对象，然后封装Authentication对象存入SecurityContextHolder。

3.2 退出登录

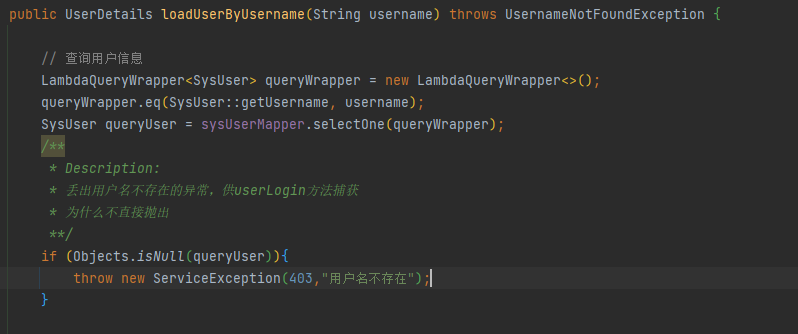
定义一个登陆接口，然后获取SecurityContextHolder中的认证信息，删除redis中对应的数据即可。

3.3 授权系统

Bug

1.异常捕获抛出 **已解决**

把用户不存在和密码错误分开来写，于是方法如下：



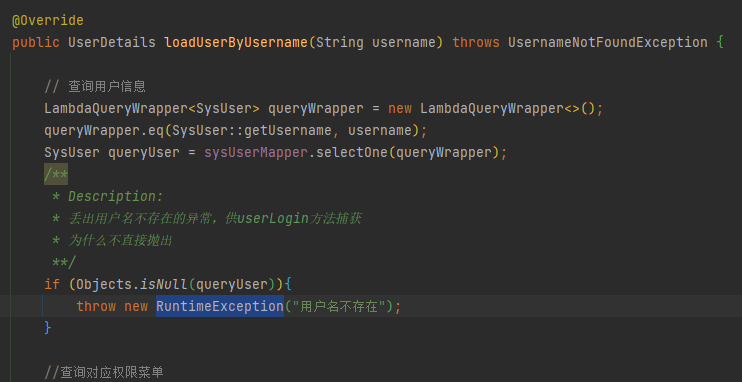


当输入不存在的用户时



发现位于loadUserByUsername方法中的我的自定义异常ServiceException并没有被拦截器拦截到，也没有被外层的ServiceException捕获，而是被最外层的Exception捕获。

**没有被拦截器拦截到这个我不知道原理，**通过控制台打印我猜测异常抛出的异常被security自带的异常InternalAuthenticationServiceException先捕获了，为了防止自定义拦截器和security自带的异常捕获器冲突出，我将loadUserByUsername中的抛出异常换成RuntimeException，不让自定义拦截器拦截到，全部丢给最外层的捕获，

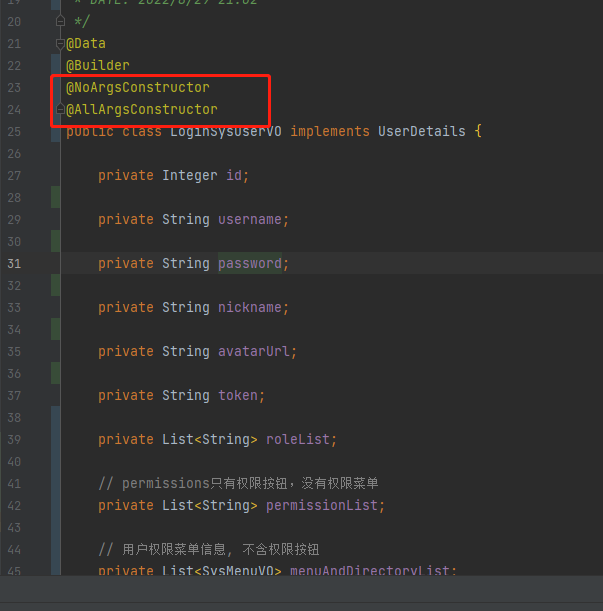




1. redis中对象解析成json时报错 已解决

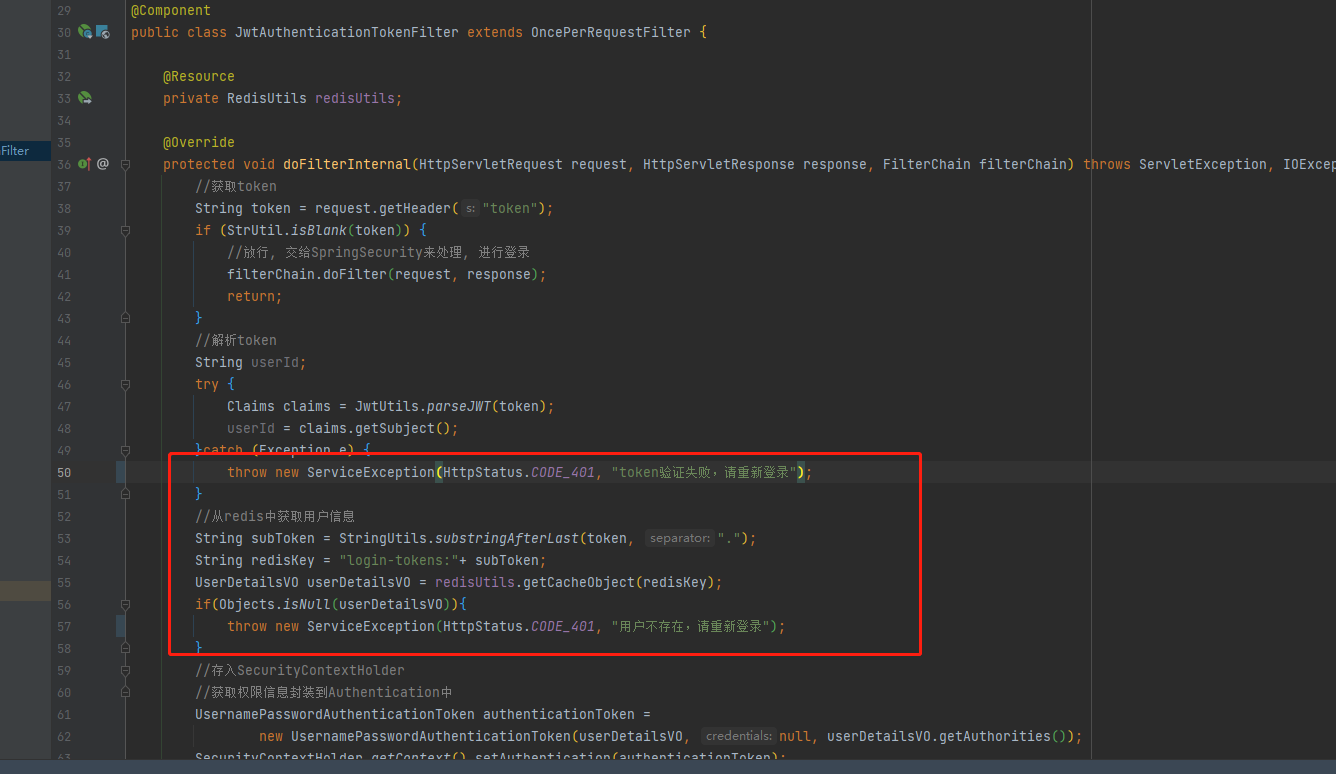
# **com.alibaba.fastjson.JSONException: default constructor not found. class**

当使用[fastjson](https://so.csdn.net/so/search?q=fastjson&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41694906/article/details/_blank)去解析对象时，对象中必须存在一个无参构造器，解决办法如下：

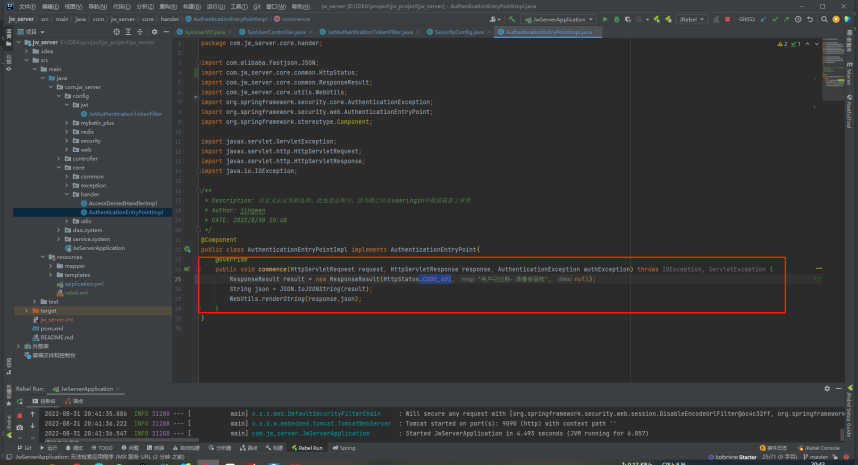


1. 自定义异常捕获器不能捕获jwt拦截器抛出的异常 已解决

jwt拦截器中抛出的异常在我自定义的异常捕获器捕获到之前，就被SpringSecurity的认证失败捕获器捕获到了



抛出的异常被SpringSecurity捕获



4.登录，认证过程中的异常捕获顺序 已解决

token过滤器 ——> springsecurity认证——>业务层异常

①token解析失败：发生在token过滤器，token过滤器在springsecurity前面，所以抛出的异常ServiceException会被springsecurity的认证失败处理器捕获，所以认证失败处理器返回的结果用户的token过期

②用户名不存在，密码错误：发生在SysUserServiceImpl中，可以被我的自定义异常拦截器捕获

5.多个认证失败请求导致多个弹窗问题 未解决

6.hutool的BeanUtil.copyProperties(A, B);可以实现带父类的复制

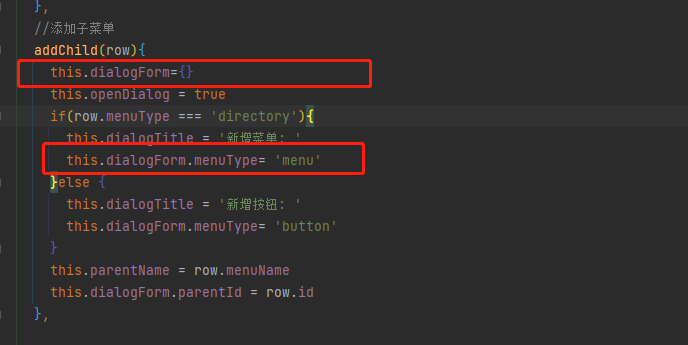
对于父子类，vue前端可以打印父子类所有属性，直接继承就完事了

7. 表单内角色多选框渲染问题 已解决->渲染问题的本质原因

若将角色多选框roleIds写到表单中去，不能实时更新，建议写到外部

<https://blog.csdn.net/weixin_45176415/article/details/122857226>

渲染问题的本质原因

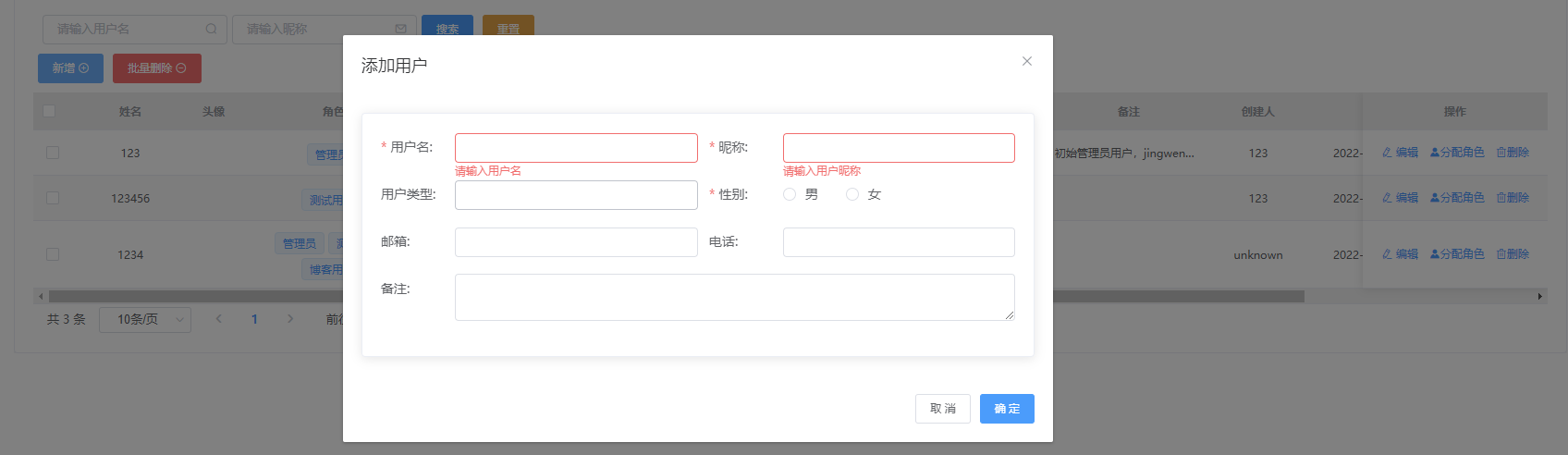


在函数内部对表单进行了一次置空，若后面对其进行操作时，则当前整个表单dialogForm={}，不含有属性menuType，可以重写一个重置表单函数，不要令整个表单直接为空

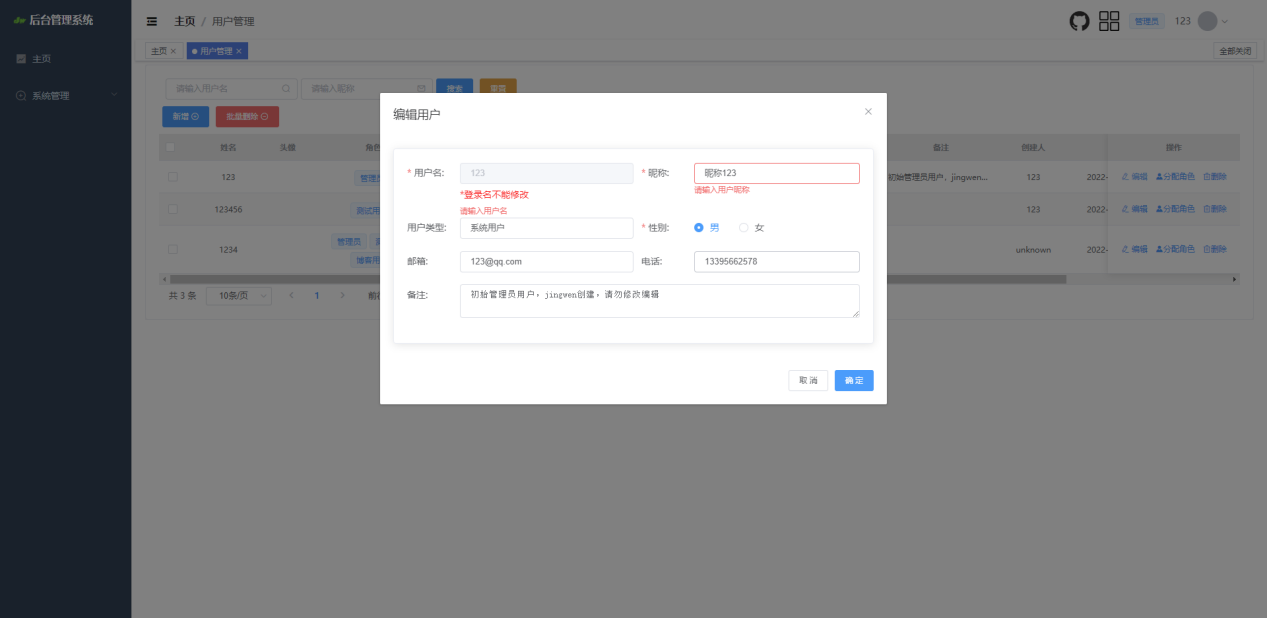
7.1 还是表单内数据渲染问题 已解决

可以通过this.$set()手动设置成响应式渲染

8.验证表单处于验证状态，点击取消后再次打开直接显示处于验证状态 已解决



点击取消后，随便点击一个编辑按钮



处于验证状态，当点击取消时需要重置验证表单的验证信息



9. ..sync可实现子组件和父组件数据双向绑定

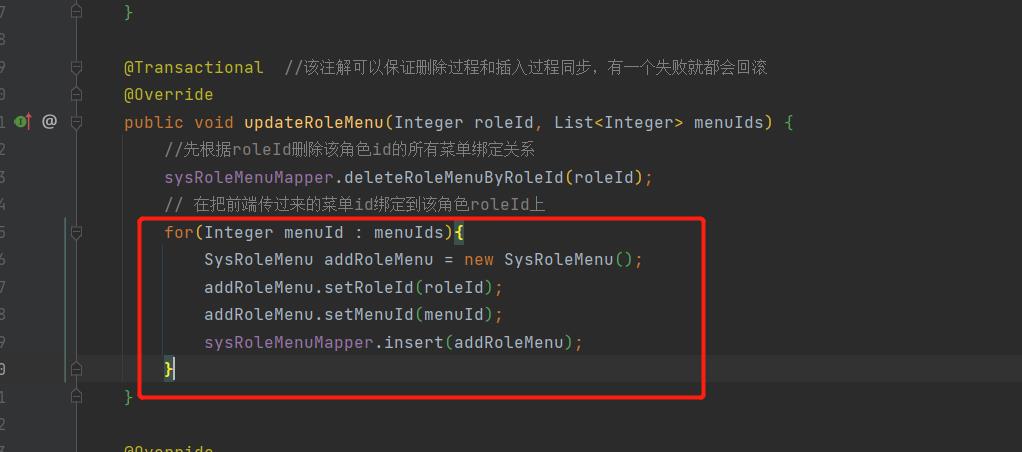
10. SpringSecurity认证的权限中有空字符

java.lang.IllegalArgumentException: A granted authority textual representation is required

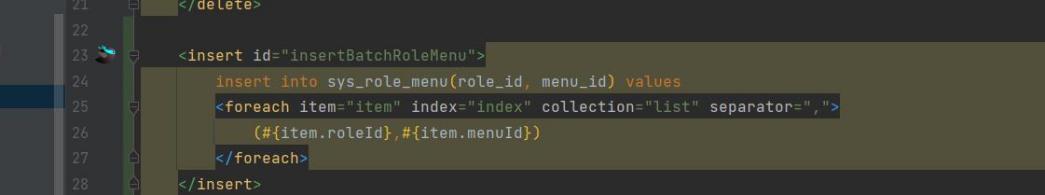
在新增按钮时，没有写入权限按钮编码

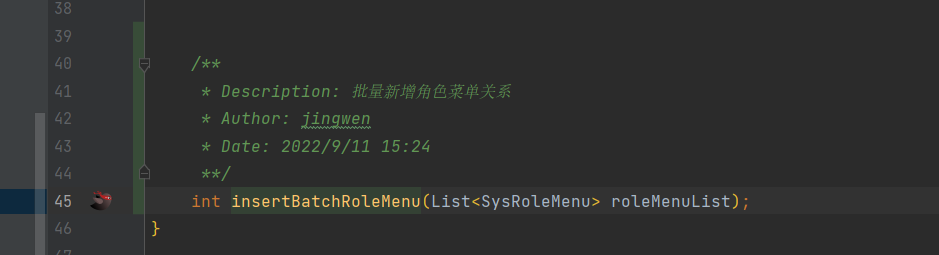
11. 批量更新

更新角色权限时发现响应太慢，检查发现数据是一条一条插入的，一次性插入比逐条插入效果要好，



自己编写批量插入语句



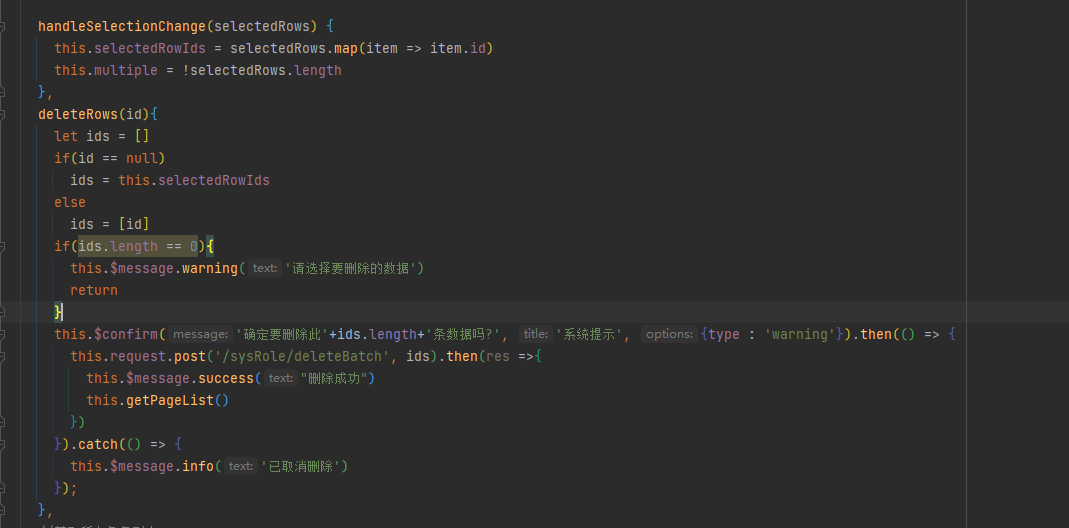




响应速度正常了

模板

删除操作模板



重置表单模板

