智能数据分析系统设计开发与实现文档

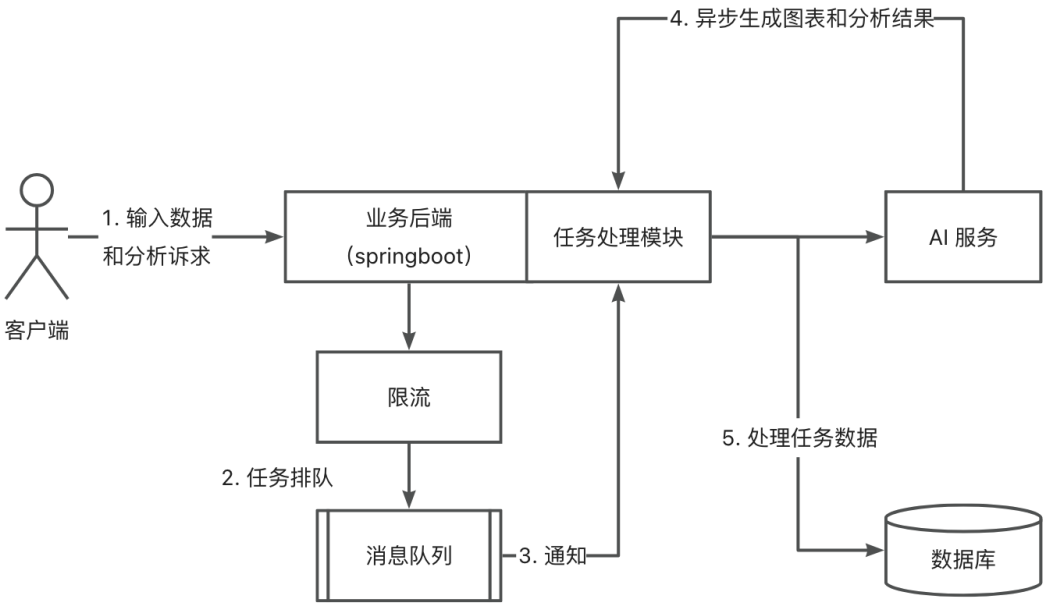
一.项目简介

1.1 项目背景

随着大数据时代的到来，数据分析变得越来越重要。智能数据分析平台旨在为用户提供一个简单易用、功能强大的数据分析工具，帮助用户从海量数据中提取有价值的信息，并通过图表直观展示，区别于传统 BI，用户只需要导入原始数据集、并输入分析诉求，就能自动生成可视化图表及分析结论，实现数据分析的降本增效。

1.2 项目特点

智能数据分析平台是一个基于Spring Boot后端和Vue.js前端的前后端分离应用，通过JWT实现身份验证，采用RBAC进行权限管理，并提供图表生成和管理功能。项目选题**新颖**，不同于泛滥的管理系统、博客、商城、本项目是结合**AIGC技术+企业BI业务场景**的综合实战，紧跟时代潮流。



二.需求分析

2.1 功能需求

2.1.1 数据分析接口

- **数据分析**：分析用户上传的数据,得出结论,并将数据可视化
- **接口限流**：使用令牌桶算法限制接口访问频率，防止滥用。
- **异步处理**：通过线程池异步处理数据分析任务，提高系统响应速度。

2.1.2 权限控制

- **基于RBAC的权限控制**：允许指定的角色访问指定的资源

2.1.3 用户管理

- **用户注册与登录**：用户可以注册新账户并登录系统。
- **用户信息管理**：用户可以查看和更新自己的信息。

2.1.4 图表管理

- **图表生成**：用户可以上传数据，系统根据分析目标生成图表。
- **图表查看与编辑**：用户可以查看和删除图表。

2.2 非功能需求

2.2.1 性能需求

- 系统应能够处理大量并发请求，响应时间不超过2秒。

2.2.2 安全需求

- 系统必须实现用户身份验证和授权，保护数据安全。

2.2.3 可用性需求

- 系统应提供友好的用户界面，确保用户能够轻松使用各项功能。

三.需求设计

3.1 技术选型

系统采用B/S架构，前端使用Vue3.js构建用户界面，后端使用Spring Boot构建RESTful API

前端

- Vue3.js
- Element Plus组件库
- ECharts可视化库

后端

- Java Spring Boot
- Java Spring Security
- MySQL 数据库
- MyBatis-Plus
- Redis + Redisson 限流
- JWT令牌校验
- Easy Excel 表格数据处理
- Swagger + Knife4j 接口文档生成
- Hutool、Apache Common Utils 等工具库

3.2 数据库设计

数据库使用MySQL，设计以下表结构：

3.2.1 用户表

- **用户ID**：唯一标识。
- **用户名**：用户昵称。
- **密码**：加密存储。
- **角色**：用户角色。

```sql
CREATE TABLE  if NOT EXISTS `user` (
                        `id` BIGINT NOT NULL  PRIMARY  KEY COMMENT '用户ID',
                        `user_account` VARCHAR(255) NOT NULL COMMENT '用户账号',
                        `user_password` VARCHAR(255) NOT NULL COMMENT '用户密码',
                        `user_name` VARCHAR(255) NOT NULL COMMENT '用户名称',
                        `user_avatar` VARCHAR(1024) DEFAULT NULL COMMENT '用户头像',
                        `user_role` CHAR(32) NOT NULL DEFAULT '用户' COMMENT '用户角色',
                        `create_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
                        `update_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
                        `delete_flag` TINYINT(1) NOT NULL DEFAULT '0' COMMENT '删除标志,0:未删除,1:已删除'
) COMMENT='用户表';
```

3.2.2 图表表

- **图表ID**：唯一标识。
- **图表名称**：用户定义的图表名称。
- **分析目标**：图表展示的数据目标。
- **图表数据**：图表的JSON配置数据。

```sql
CREATE TABLE if NOT EXISTS `chart` (
                        `id` BIGINT NOT NULL PRIMARY KEY COMMENT '图表ID',
                        `name` varchar(128) NULL COMMENT '图表名称',
                        `analysis_target` TEXT NOT NULL COMMENT '分析目标',
                        `chart_data` TEXT NOT NULL COMMENT '图标数据',
                        `chart_type` VARCHAR(255) NOT NULL COMMENT '图标类型',
                        `generated_chart_data` TEXT COMMENT '生成的图表数据',
                        `analysis_conclusion` TEXT COMMENT '生成的分析结论',
                        `user_id` BIGINT NOT NULL COMMENT '创建用户ID',
                        `state` CHAR(32) NOT NULL DEFAULT '等待中' COMMENT '图表状态,等待中,生成中,成功,失败',
                        `execute_message` TEXT COMMENT '执行信息',
                        `created_time` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
```

```
                            `updated_time` DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
                            `delete_flag` TINYINT(1) NOT NULL DEFAULT '0'
COMMENT '删除标志,0:未删除,1:已删除'
) COMMENT='图表表';
```

3.3 接口设计

使用Swagger文档化API接口。

3.3.1 用户相关接口

- `POST /user/login`：用户登录。
- `POST /user/register`：用户注册。
- `GET /user/getUserInfo`:获取用户信息
- `GET /user/page`:用户信息分页
- `POST /user`:新增用户
- `PUT /user`:更新用户信息
- `GET /user/getUserBiId/{id}`:获取用户信息
- `DELETE /user`/{ids}:删除用户

login

接口地址:`/user/login`

请求方式:`POST`

请求数据类型:`application/json`

响应数据类型:`*/*`

接口描述:用户登录

请求示例:

```
{
  "createTime": "",
  "deleteFlag": 0,
  "id": 0,
  "updateTime": "",
  "userAccount": "",
  "userAvatar": "",
  "userName": "",
  "userPassword": "",
  "userRole": ""
}
```

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
user	user	body	true	User	User
createTime			false	string(date-time)	

参数名称	参数说明	请求类型	是否必须	数据类型	schema
deleteFlag			false	integer(int32)	
id			false	integer(int64)	
updateTime			false	string(date-time)	
userAccount			false	string	
userAvatar			false	string	
userName			false	string	
userPassword			false	string	
userRole			false	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

resister

接口地址:`/user/register`

请求方式:`POST`

请求数据类型:`application/json`

响应数据类型:`*/*`

接口描述:用户注册

请求示例:

```
{
  "createTime": "",
  "deleteFlag": 0,
  "id": 0,
  "updateTime": "",
  "userAccount": "",
  "userAvatar": "",
  "userName": "",
  "userPassword": "",
  "userRole": ""
}
```

请求参数:

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
user	user	body	true	User	User
createTime			false	string(date-time)	
deleteFlag			false	integer(int32)	
id			false	integer(int64)	
updateTime			false	string(date-time)	
userAccount			false	string	
userAvatar			false	string	
userName			false	string	
userPassword			false	string	
userRole			false	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult

状态码	说明	schema
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

getUserInfo

接口地址: `/user/getUserInfo`

请求方式: `GET`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

响应状态:

状态码	说明	schema
200	OK	ResponseResult
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)

参数名称	参数说明	类型	schema
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

page

接口地址: `/user/page`

请求方式: `GET`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
currentPage	currentPage	query	false	integer(int32)	
pageSize	pageSize	query	false	integer(int32)	
userAccount	userAccount	query	false	string	
userName	userName	query	false	string	
userRole	userRole	query	false	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)

参数名称	参数说明	类型	schema
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

addUser

接口地址:`/user`

请求方式:`POST`

请求数据类型:`application/json`

响应数据类型:`*/*`

请求示例:

```
{
  "id": 0,
  "userAccount": "",
  "userAvatar": "",
  "userName": "",
  "userPassword": "",
  "userRole": ""
}
```

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
userDTO	userDTO	body	true	UserDTO	UserDTO
id			false	integer(int64)	
userAccount			false	string	
userAvatar			false	string	
userName			false	string	
userPassword			false	string	
userRole			false	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

updateUser

接口地址: `/user`

请求方式: `PUT`

请求数据类型: `application/json`

响应数据类型: `*/*`

请求示例:

```
{
  "id": 0,
  "userAccount": "",
  "userAvatar": "",
  "userName": "",
  "userPassword": "",
  "userRole": ""
}
```

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema

参数名称	参数说明	请求类型	是否必须	数据类型	schema
userDTO	userDTO	body	true	UserDTO	UserDTO
id			false	integer(int64)	
userAccount			false	string	
userAvatar			false	string	
userName			false	string	
userPassword			false	string	
userRole			false	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

getUserById

接口地址: `/user/{id}`

请求方式: `GET`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
id	id	path	true	integer(int64)	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

deleteUserById

接口地址: `/user/{ids}`

请求方式: `DELETE`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
ids	ids	path	true	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
204	No Content	
401	Unauthorized	
403	Forbidden	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

3.3.2 图表相关接口

- `POST /chart/generateChartByAI`：生成图表。
- `GET /chart/getChartById/{id}`：获取图表详情。
- `GET /chart/list`:获取图表列表
- `POST /chart/add`:新增图表
- `PUT /chart/update`:更新图表
- `DELETE /chart/{ids}`:删除图表

generateChartByAI

接口地址:`/chart/generateChartByAI`

请求方式:`POST`

请求数据类型:`multipart/form-data`

响应数据类型:`*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
analysisTarget		query	false	string	
chartData		query	false	string	

参数名称	参数说明	请求类型	是否必须	数据类型	schema
chartType		query	false	string	
isAsynchronism		query	false	boolean	
name		query	false	string	
file	file	formData	false	file	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

getChartById

接口地址: `/chart/getChartById/{id}`

请求方式: `GET`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema

参数名称	参数说明	请求类型	是否必须	数据类型	schema
id	id	path	true	integer(int64)	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

page

接口地址: `/chart/list`

请求方式: `GET`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
UserId	UserId	query	false	integer(int64)	
name	name	query	false	string	
pageNum	pageNum	query	false	integer(int32)	
pageSize	pageSize	query	false	integer(int32)	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

addChart

接口地址:`/chart/add`

请求方式:`POST`

请求数据类型:`application/json`

响应数据类型:`*/*`

请求示例:

```
{
  "analysisConclusion": "",
  "analysisTarget": "",
  "chartData": "",
  "chartType": "",
  "createdTime": "",
  "deleteFlag": 0,
  "executeMessage": "",
  "generatedChartData": "",
  "id": 0,
  "name": "",
  "state": "",
  "updatedTime": "",
  "userId": 0
```

```
15 }
```

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
chart	chart	body	true	Chart	Chart
analysisConclusion			false	string	
analysisTarget			false	string	
chartData			false	string	
chartType			false	string	
createdTime			false	string(date-time)	
deleteFlag			false	integer(int32)	
executeMessage			false	string	
generatedChartData			false	string	
id			false	integer(int64)	
name			false	string	
state			false	string	
updatedTime			false	string(date-time)	
userId			false	integer(int64)	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)

参数名称	参数说明	类型	schema
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

updateChart

接口地址:`/chart/update`

请求方式:`PUT`

请求数据类型:`application/json`

响应数据类型:`*/*`

请求示例:

```
{
  "analysisConclusion": "",
  "analysisTarget": "",
  "chartData": "",
  "chartType": "",
  "createdTime": "",
  "deleteFlag": 0,
  "executeMessage": "",
  "generatedChartData": "",
  "id": 0,
  "name": "",
  "state": "",
  "updatedTime": "",
  "userId": 0
}
```

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
chart	chart	body	true	Chart	Chart
analysisConclusion			false	string	
analysisTarget			false	string	
chartData			false	string	

参数名称	参数说明	请求类型	是否必须	数据类型	schema
chartType			false	string	
createdTime			false	string(date-time)	
deleteFlag			false	integer(int32)	
executeMessage			false	string	
generatedChartData			false	string	
id			false	integer(int64)	
name			false	string	
state			false	string	
updatedTime			false	string(date-time)	
userId			false	integer(int64)	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
201	Created	
401	Unauthorized	
403	Forbidden	
404	Not Found	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

deleteChartById

接口地址: `/chart/{ids}`

请求方式: `DELETE`

请求数据类型: `application/x-www-form-urlencoded`

响应数据类型: `*/*`

请求参数:

参数名称	参数说明	请求类型	是否必须	数据类型	schema
ids	ids	path	true	string	

响应状态:

状态码	说明	schema
200	OK	ResponseResult
204	No Content	
401	Unauthorized	
403	Forbidden	

响应参数:

参数名称	参数说明	类型	schema
code		integer(int32)	integer(int32)
data		object	
msg		string	

响应示例:

```
{
    "code": 0,
    "data": {},
    "msg": ""
}
```

3.4 安全设计

使用JWT进行用户认证，Spring Security实现权限控制。

4. 功能实现

4.1 用户管理

- **登录与注册**：用户可以通过UserController进行登录和注册，系统使用JWT进行身份验证。
- **权限控制**：基于角色的访问控制，使用Spring Security和自定义的权限注解实现。

后端接口实现

用户登录（Login）

- **接口**：`POST /user/login`
- **功能**：用户登录，验证账号和密码。
- **返回**：登录成功返回JWT令牌和用户信息，失败返回错误信息。

UserController

当客户端向`/login`路径发送POST请求并包含用户信息（如用户名和密码）时，执行登录逻辑，并返回一个包含登录结果的`ResponseResult`对象。

```
@PostMapping("/login")
public ResponseResult login(@RequestBody User user){
    return userService.login(user);
}
```

UserServce

实现了用户登录的完整流程，包括用户认证、JWT Token的生成、权限处理（待实现）、用户信息的存储和登录成功的响应。

Spring Security和JWT结合使用的用户登录实现。

1. `public ResponseResult login(User user)`：这是一个公共方法，返回类型是`ResponseResult`，参数是一个`User`对象。
2. 认证过程：
    - 创建`UsernamePasswordAuthenticationToken`对象，包含用户的账号和密码。
    - 使用`authenticationManager.authenticate(authenticationToken)`进行认证，如果认证失败，则抛出`SystemException`异常，异常中包含登录错误的枚举值。
3. 生成JWT Token：
    - 如果认证成功，从认证结果中获取`LoginUserDetails`对象，这是一个包含用户详细信息的Spring Security认证主体对象。
    - 从`LoginUserDetails`中获取用户ID，并使用`JwtUtil.createJWT(id.toString())`方法生成一个JWT Token。
4. 权限处理（待办事项）：
    - 代码中有一个`//todo:权限`注释，表示这里应该添加权限相关的处理逻辑，但目前尚未实现。
5. 存入Redis：
    - 将`LoginUserDetails`对象存储到Redis缓存中，键为`SystemConstants.LOGIN_USER_REDIS_KEY`加上用户ID。
6. 返回结果：
    - 创建一个`Map`对象来存储返回的数据。
    - 创建一个`UserInfo`对象，包含用户的账号、姓名、头像和角色信息。
    - 将JWT Token和`UserInfo`对象放入`Map`中。

- 使用`ResponseResult.okResult(map)`方法返回一个包含登录成功状态和数据的`ResponseResult`对象。

```
@Override
    public ResponseResult login(User user) {
        //认证
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(user.getUserAccount(),
user.getUserPassword());
        Authentication authentication =
authenticationManager.authenticate(authenticationToken);
        if (Objects.isNull(authentication)){
            throw new
SystemException(ResponseResult.AppHttpCodeEnum.LOGIN_ERROR);
        }
        //生成token
        LoginUserDetails principal = (LoginUserDetails)
authentication.getPrincipal();
        Long id = principal.getUser().getId();
        String jwt = JwtUtil.createJWT(id.toString());
        //todo:权限
        //存入redis

 redisCache.setCacheObject(SystemConstants.LOGIN_USER_REDIS_KEY+id,principal
);
        //返回结果
        Map<String,Object> map = new HashMap<>();
        UserInfo userInfo = new
UserInfo(principal.getUser().getUserAccount(),principal.getUser().getUserNam
e(),principal.getUser().getUserAvatar(),principal.getUser().getUserRole());
        map.put("token",jwt);
        map.put("userInfo",userInfo);
        return ResponseResult.okResult(map);
    }
```

用户注册（Register）

- **接口**：`POST /user/register`
- **功能**：用户注册，创建新用户。
- **返回**：注册成功返回成功消息，失败返回错误信息。

UserController

当客户端向`/register`路径发送POST请求并包含用户信息（如用户名、密码等）时，这个方法会被调用，执行注册逻辑，并返回一个包含注册结果的`ResponseResult`对象

```
    @PostMapping("/register")
    public ResponseResult resister(@RequestBody User user){
        return userService.register(user);
    }
```

UserServce

实现了用户注册的核心流程，包括检查账号是否存在、密码加密、生成唯一ID、保存用户信息到数据库，并返回注册成功的响应。

1. `public ResponseResult register(User user)`：这是一个公共方法，返回类型是`ResponseResult`，参数是一个`User`对象。
2. 检查用户账号是否存在：
   - 调用`userAccountExist`方法检查传入的`user`对象中的账号是否已经存在。
   - 如果账号已存在，则抛出`SystemException`异常，异常中包含用户名已存在的枚举值。
3. 密码加密：
   - 使用`passwordEncoder.encode`方法对用户密码进行加密处理。
   - 将加密后的密码设置回`user`对象的`userPassword`属性。
4. 生成ID：
   - 使用`IdUtil.getSnowflake(1, 1).nextId()`方法生成一个唯一的ID（基于Snowflake算法）。
   - 将生成的ID设置为`user`对象的`id`属性。
5. 存入数据库：
   - 调用`save`方法将`user`对象保存到数据库中。
6. 返回结果：
   - 使用`ResponseResult.okResult()`方法返回一个表示注册成功的`ResponseResult`对象。

```
    @Override
    public ResponseResult register(User user) {
        //用户账号是否存在
        if (userAccountExist(user.getUserAccount())){
            throw new
SystemException(ResponseResult.AppHttpCodeEnum.USERNAME_EXIST);
        }
        //密码加密

 user.setUserPassword(passwordEncoder.encode(user.getUserPassword()));
        //生成ID
        long id = IdUtil.getSnowflake(1, 1).nextId();
        user.setId(id);
        //存入数据库
        save(user);
        return ResponseResult.okResult();
    }
```

获取用户信息（GetUserInfo）

- **接口**：`GET /user/getUserInfo`
- **功能**：获取当前登录用户的信息。
- **返回**：用户信息。

UserController

获取当前登录用户信息的接口，只有具有“用户”角色的用户才能访问

```
    @PreAuthorize("@ps.hasRole('用户')")
    @GetMapping("/getUserInfo")
    public ResponseResult getUserInfo(){
        LoginUserDetails loginUser = SecurityUtils.getLoginUser();
        if (Objects.isNull(loginUser)){
            return
ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.NEED_LOGIN);
        }
        User user = loginUser.getUser();
        user.setUserPassword("");
        return ResponseResult.okResult(user);
    }
```

用户分页列表（UserPage）

- **接口**：`GET /user/page`
- **功能**：分页查询用户列表。
- **参数**：当前页码、页大小、用户名、账号、角色。
- **返回**：用户列表和分页信息。

Usercontroller

一个分页查询用户的接口，只有具有“管理员”角色的用户才能访问。它记录请求信息，执行分页查询，然后返回查询结果

```
    @PreAuthorize("@ps.hasRole('管理员')")
    @GetMapping("/page")
    @ResponseBody
    public ResponseResult page(Integer currentPage,Integer pageSize,String
userName,String userAccount,String userRole){
        log.info("currentPage:{},pageSize:{},userName:{},userAccount:
{}",currentPage,pageSize,userName,userAccount);
        PageVO page = userService.pageByUsernameAndUseraccount(currentPage,
pageSize, userName, userAccount,userRole);
        return ResponseResult.okResult(page);
    }
```

UserServce

实现了一个分页查询用户信息的功能，它根据用户名、账号和角色进行过滤，并将查询结果转换为视图对象，最后返回一个包含分页信息的视图对象

1. `public PageVO pageByUsernameAndUseraccount(...)`：这是一个公共方法，返回类型是`PageVO`，参数包括当前页码`currentPage`、页面大小`pageSize`以及用于过滤的用户名`userName`、账号`userAccount`和角色`userRole`。
2. 初始化用户视图对象列表：
    - 创建一个`ArrayList`来存储`UserVO`对象，这些对象将用于构建最终的分页视图对象。
3. 构建查询条件：
    - 使用`LambdaQueryWrapper`构建查询条件。这个查询包装器会根据传入的参数动态地添加查询条件。
    - `like`方法用于添加模糊匹配条件，`eq`方法用于添加精确匹配条件。

- `StringUtils.hasText`检查传入的字符串参数是否非空，只有非空时才会添加相应的查询条件。

4. 执行分页查询：
   - 使用`page`方法执行分页查询，传入分页参数和查询条件包装器。
   - `page`方法返回一个`Page<User>`对象，包含了查询结果和分页信息。
5. 转换结果并设置ID：
   - 遍历查询结果中的记录，将每个`User`对象转换为`UserVO`对象。
   - 使用`BeanCopyUtil.copyBean`方法进行对象属性的复制。
   - 将`UserVO`对象的ID设置为字符串形式，以满足视图对象的要求。
6. 构建并返回分页视图对象：
   - 创建`PageVO`对象，传入用户视图对象列表和总记录数。
   - 返回`PageVO`对象，包含了分页的用户信息和分页信息。

```
    @Override
    public PageVO pageByUsernameAndUseraccount(Integer currentPage, Integer pageSize, String userName, String userAccount, String userRole) {
        List<UserVO> userVOList = new ArrayList<>();
        LambdaQueryWrapper<User> lambdaQueryWrapper = new LambdaQueryWrapper<User>()
                .like(StringUtils.hasText(userName), User::getUserName, userName)
                .eq(StringUtils.hasText(userAccount), User::getUserAccount, userAccount)
                .eq(StringUtils.hasText(userRole), User::getUserRole, userRole);
        Page<User> page = page(new Page<User>(currentPage,pageSize), lambdaQueryWrapper);
        for (User user : page.getRecords()) {
            UserVO userVO = BeanCopyUtil.copyBean(user, UserVO.class);
            userVO.setId(user.getId().toString());
            userVOList.add(userVO);
        }
        return new PageVO(userVOList,page.getTotal());
    }
```

添加用户（AddUser）

- **接口**：`POST /user`
- **功能**：添加新用户。
- **参数**：用户信息（账号、密码、名称、头像、角色）。
- **返回**：操作结果。

UserController

一个添加新用户的接口，只有具有“管理员”角色的用户才能访问。

它接收用户信息，执行添加用户的操作，并返回操作成功的响应。

```
    @PreAuthorize("@ps.hasRole('管理员')")
    @PostMapping
    @ResponseBody
    public ResponseResult addUser(@RequestBody UserDTO userDTO){
        userService.addUser(userDTO);
        return ResponseResult.okResult();
    }
```

UserServce

实现了将用户DTO转换为用户实体、加密密码、生成唯一ID，并保存用户信息到数据库的流程

1. `public void addUser(UserDTO userDTO)`：这是一个公共方法，没有返回值（`void`），参数是一个`UserDTO`对象。
2. 复制属性：
   - 使用`BeanCopyUtil.copyBean`方法将`UserDTO`对象的属性复制到一个新的`User`对象中。这个工具方法通常用于将DTO（Data Transfer Object）对象的属性映射到实体对象。
3. 密码加密：
   - 使用`passwordEncoder.encode`方法对用户密码进行加密处理。
   - 将加密后的密码设置回`user`对象的`userPassword`属性。
4. 生成ID：
   - 使用`IdUtil.getSnowflake(1, 1).nextId()`方法生成一个唯一的ID（基于Snowflake算法）。
   - 将生成的ID设置为`user`对象的`id`属性。
5. 保存用户：
   - 调用`save`方法将`user`对象保存到数据库中。这个方法可能是一个Repository层的方法，负责实际的数据库操作。

```
    public void addUser(UserDTO userDTO) {
        User user = BeanCopyUtil.copyBean(userDTO, User.class);
        //密码加密
        user.setUserPassword(passwordEncoder.encode(user.getUserPassword()));
        //生成ID
        long id = IdUtil.getSnowflake(1, 1).nextId();
        user.setId(id);
        save(user);
    }
```

更新用户信息（UpdateUser）

- **接口**：`PUT /user`
- **功能**：更新用户信息。
- **参数**：用户信息（ID、账号、密码、名称、头像、角色）。
- **返回**：操作结果。

UserController

一个更新用户的接口，只有具有“管理员”角色的用户才能访问。

它接收用户信息，执行更新用户的操作，并返回操作成功的响应

```
    @PreAuthorize("@ps.hasRole('管理员')")
    @PutMapping
    @ResponseBody
    public ResponseResult updateUser(@RequestBody UserDTO userDTO){
        User user = BeanCopyUtil.copyBean(userDTO, User.class);
        userService.updateById(user);
        return ResponseResult.okResult();
    }
```

删除用户（DeleteUser）

- **接口**：`DELETE /user/{id}`
- **功能**：根据ID删除用户。
- **返回**：操作结果。

UserController

一个删除用户的接口，只有具有“管理员”角色的用户才能访问。

它接收用户ID列表，执行删除用户的操作，并返回操作成功的响应

```
    @PreAuthorize("@ps.hasRole('管理员')")
    @DeleteMapping("/{ids}")
    @ResponseBody
    public ResponseResult deleteUserById(@PathVariable List<Long> ids){
        userService.removeByIds(ids);
        return ResponseResult.okResult();
    }
```

完整的UserController

```
package space.anyi.BI.controller;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import space.anyi.BI.entity.LoginUserDetails;
import space.anyi.BI.entity.ResponseResult;
import space.anyi.BI.entity.User;
import space.anyi.BI.entity.dto.UserDTO;
import space.anyi.BI.entity.vo.PageVO;
import space.anyi.BI.entity.vo.UserVO;
import space.anyi.BI.service.UserService;
import space.anyi.BI.util.BeanCopyUtil;
import space.anyi.BI.util.SecurityUtils;

import javax.annotation.Resource;
import java.util.List;
import java.util.Objects;

@RequestMapping("/user")
@RestController
```

```java
public class UserController {
    private final static Logger log =
LoggerFactory.getLogger(UserController.class);
    @Resource
    private UserService userService;

    @PostMapping("/login")
    public ResponseResult login(@RequestBody User user){
        return userService.login(user);
    }

    @PostMapping("/register")
    public ResponseResult resister(@RequestBody User user){
        return userService.register(user);
    }

    @PreAuthorize("@ps.hasRole('用户')")
    @GetMapping("/getUserInfo")
    public ResponseResult getUserInfo(){
        LoginUserDetails loginUser = SecurityUtils.getLoginUser();
        if (Objects.isNull(loginUser)){
            return
ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.NEED_LOGIN);
        }
        User user = loginUser.getUser();
        user.setUserPassword("");
        System.out.println("user = " + user);
        return ResponseResult.okResult(user);
    }
    @PreAuthorize("@ps.hasRole('管理员')")
    @GetMapping("/page")
    @ResponseBody
    public ResponseResult page(Integer currentPage,Integer pageSize,String
userName,String userAccount,String userRole){
        log.info("currentPage:{},pageSize:{},userName:{},userAccount:
{}",currentPage,pageSize,userName,userAccount);
        PageVO page = userService.pageByUsernameAndUseraccount(currentPage,
pageSize, userName, userAccount,userRole);
        return ResponseResult.okResult(page);
    }
    @PreAuthorize("@ps.hasRole('用户')")
    @GetMapping("/{id}")
    @ResponseBody
    public ResponseResult getUserById(@PathVariable Long id){
        User user = userService.getById(id);
        UserVO userVO = BeanCopyUtil.copyBean(user, UserVO.class);
        userVO.setId(user.getId().toString());
        return ResponseResult.okResult(userVO);
    }

    @PreAuthorize("@ps.hasRole('管理员')")
    @DeleteMapping("/{ids}")
    @ResponseBody
    public ResponseResult deleteUserById(@PathVariable List<Long> ids){
        userService.removeByIds(ids);
```

```
        return ResponseResult.okResult();
    }

    @PreAuthorize("@ps.hasRole('管理员')")
    @PutMapping
    @ResponseBody
    public ResponseResult updateUser(@RequestBody UserDTO userDTO){
        User user = BeanCopyUtil.copyBean(userDTO, User.class);
        userService.updateById(user);
        return ResponseResult.okResult();
    }

    @PreAuthorize("@ps.hasRole('管理员')")
    @PostMapping
    @ResponseBody
    public ResponseResult addUser(@RequestBody UserDTO userDTO){
        userService.addUser(userDTO);
        return ResponseResult.okResult();
    }

}

```

完整的UserServce

```
package space.anyi.BI.service.impl;

import cn.hutool.core.util.IdUtil;
import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.util.StringUtils;
import space.anyi.BI.constant.SystemConstants;
import space.anyi.BI.entity.LoginUserDetails;
import space.anyi.BI.entity.ResponseResult;
import space.anyi.BI.entity.User;
import space.anyi.BI.entity.dto.UserDTO;
import space.anyi.BI.entity.vo.PageVO;
import space.anyi.BI.entity.vo.UserInfo;
import space.anyi.BI.entity.vo.UserVO;
import space.anyi.BI.exception.SystemException;
import space.anyi.BI.service.UserService;
import space.anyi.BI.mapper.UserMapper;
import org.springframework.stereotype.Service;
import space.anyi.BI.util.BeanCopyUtil;
import space.anyi.BI.util.JwtUtil;
import space.anyi.BI.util.RedisCache;

import javax.annotation.Resource;
```

```java
import java.util.*;

@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User>
implements UserService{
    @Resource
    private  AuthenticationManager authenticationManager;
    @Resource
    private  RedisCache redisCache;
    @Resource
    private  PasswordEncoder passwordEncoder;
    @Override
    public ResponseResult login(User user) {
        //认证
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(user.getUserAccount(),
user.getUserPassword());
        Authentication authentication =
authenticationManager.authenticate(authenticationToken);
        if (Objects.isNull(authentication)){
            throw new
SystemException(ResponseResult.AppHttpCodeEnum.LOGIN_ERROR);
        }
        //生成token
        LoginUserDetails principal = (LoginUserDetails)
authentication.getPrincipal();
        Long id = principal.getUser().getId();
        String jwt = JwtUtil.createJWT(id.toString());
        //todo:权限
        //存入redis

 redisCache.setCacheObject(SystemConstants.LOGIN_USER_REDIS_KEY+id,principa
l);
        //返回结果
        Map<String,Object> map = new HashMap<>();
        UserInfo userInfo = new
UserInfo(principal.getUser().getUserAccount(),principal.getUser().getUserNa
me(),principal.getUser().getUserAvatar(),principal.getUser().getUserRole())
;
        map.put("token",jwt);
        map.put("userInfo",userInfo);
        return ResponseResult.okResult(map);
    }

    @Override
    public void addUser(UserDTO userDTO) {
        User user = BeanCopyUtil.copyBean(userDTO, User.class);
        //密码加密

 user.setUserPassword(passwordEncoder.encode(user.getUserPassword()));
        //生成ID
        long id = IdUtil.getSnowflake(1, 1).nextId();
        user.setId(id);
        save(user);
    }
```

```
72
73      @Override
74      public PageVO pageByUsernameAndUseraccount(Integer currentPage, Integer
pageSize, String userName, String userAccount, String userRole) {
75          List<UserVO> userVOList = new ArrayList<>();
76          LambdaQueryWrapper<User> lambdaQueryWrapper = new
LambdaQueryWrapper<User>()
77                  .like(StringUtils.hasText(userName), User::getUserName,
userName)
78                  .eq(StringUtils.hasText(userAccount), User::getUserAccount,
userAccount)
79                  .eq(StringUtils.hasText(userRole), User::getUserRole,
userRole);
80          Page<User> page = page(new Page<User>(currentPage,pageSize),
lambdaQueryWrapper);
81          for (User user : page.getRecords()) {
82              UserVO userVO = BeanCopyUtil.copyBean(user, UserVO.class);
83              userVO.setId(user.getId().toString());
84              userVOList.add(userVO);
85          }
86          return new PageVO(userVOList,page.getTotal());
87      }
88
89      @Override
90      public ResponseResult register(User user) {
91          //用户账号是否存在
92          if (userAccountExist(user.getUserAccount())){
93              throw new
SystemException(ResponseResult.AppHttpCodeEnum.USERNAME_EXIST);
94          }
95          //密码加密
96
 user.setUserPassword(passwordEncoder.encode(user.getUserPassword()));
97          //生成ID
98          long id = IdUtil.getSnowflake(1, 1).nextId();
99          user.setId(id);
100         //存入数据库
101         save(user);
102         return ResponseResult.okResult();
103     }
104     /**
105      * 判断用户账号是否已被注册
106      * @param userAccount
107      * @return
108      */
109     private boolean userAccountExist(String userAccount) {
110         LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
111         queryWrapper.eq(User::getUserAccount,userAccount);
112         if (count(queryWrapper)>0) {
113             return true;
114         }
115         return false;
116     }
117 }
```

权限控制

用户管理模块中的所有操作都需要进行权限检查，确保只有具备相应权限的用户才能执行操作。例如，只有管理员可以添加、更新和删除用户。

认证

- JWTAuthenticationTokenFilter认证过滤器
    - 这个过滤器的作用是验证传入的JWT，如果验证成功，则将用户信息放入Spring Security的上下文中，允许请求继续处理。
    - 如果验证失败，则返回错误响应，阻止请求继续

```
package space.anyi.BI.filter;

import com.alibaba.fastjson.JSON;
import io.jsonwebtoken.Claims;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTo
ken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;
import space.anyi.BI.BIApplication;
import space.anyi.BI.constant.SystemConstants;
import space.anyi.BI.entity.LoginUserDetails;
import space.anyi.BI.entity.ResponseResult;
import space.anyi.BI.util.JwtUtil;
import space.anyi.BI.util.RedisCache;
import space.anyi.BI.util.WebUtils;

import javax.annotation.Resource;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Objects;

@Component
public class JWTAuthenticationTokenFilter extends OncePerRequestFilter {
    private final static Logger log =
LoggerFactory.getLogger(BIApplication.class);
    @Resource
    private RedisCache redisCache;

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, FilterChain filterChain) throws
ServletException, IOException {
```

```
        //获取token
        String uri = httpServletRequest.getRequestURI();
        String token = httpServletRequest.getHeader("token");
        log.info("token={},\n请求的uri={}",token,uri);
        if (!StringUtils.hasText(token)) {
            filterChain.doFilter(httpServletRequest, httpServletResponse);
            return;
        }
        //解析token
        Claims jwt;
        try {
            jwt = JwtUtil.parseJWT(token);
        } catch (Exception e) {
            e.printStackTrace();
            //响应前端,需要登陆
            ResponseResult errorResult = ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.NEED_LOGIN);
            WebUtils.renderString(httpServletResponse, JSON.toJSONString(errorResult));
            return;
        }
        //从redis中获取用户信息
        LoginUserDetails loginUserDetails = redisCache.getCacheObject(SystemConstants.LOGIN_USER_REDIS_KEY+jwt.getSubject());

        //已退出登陆或登陆已过期
        if(Objects.isNull(loginUserDetails)){
            //响应前端,需要登陆
            ResponseResult errorResult = ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.NEED_LOGIN);
            WebUtils.renderString(httpServletResponse, JSON.toJSONString(errorResult));
            return;
        }

        //存入SecurityContextHolder
        Authentication authenticationToken = new UsernamePasswordAuthenticationToken(loginUserDetails,null,null);

 SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        //放行
        filterChain.doFilter(httpServletRequest,httpServletResponse);
    }
}
```

- 鉴权

```
package space.anyi.BI.service.impl;

import org.springframework.stereotype.Service;
import space.anyi.BI.service.PermissionService;
import space.anyi.BI.util.SecurityUtils;

@Service("ps")
```

```
public class PermissionServiceImpl implements PermissionService {
    @Override
    public boolean hasRole(String role) {
        String userRole =
SecurityUtils.getLoginUser().getUser().getUserRole();
        if ("管理员".equals(userRole))return true;
        if (userRole.equals(role)){
            return true;
        }
        return false;
    }
}
```

- 安全配置

```
package space.anyi.BI.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.method.configuration.EnableGl
obalMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurity
ConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.security.web.access.AccessDeniedHandler;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticati
onFilter;
import space.anyi.BI.filter.JWTAuthenticationTokenFilter;

import javax.annotation.Resource;

@EnableGlobalMethodSecurity(prePostEnabled = true)
@Configuration
public class SecurityConfig  extends WebSecurityConfigurerAdapter {
    @Resource
    private JWTAuthenticationTokenFilter jwtAuthenticationTokenFilter;
    @Resource
    private  AccessDeniedHandler accessDeniedHandler;
    @Resource
    private  AuthenticationEntryPoint authenticationEntryPoint;
    /**
     * 密码校验方式
     * @return
     */
    @Bean
```

```
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    /**
     * 安全配置
     * @param http
     * @throws Exception
     */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                //关闭csrf
                .csrf().disable()
                //不通过Session获取SecurityContext
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                // 对于登录接口 允许匿名访问
                .antMatchers(HttpMethod.POST,"/user/login","/user/register").anonymous()
                .antMatchers(HttpMethod.GET,"/swagger/**","/v2/**","/v2/api-docs-ext/**","/swagger-resources/**").anonymous()
                //放行对静态资源的访问
                .antMatchers(HttpMethod.GET,"/","/*.html","/**/*.css","/**/*.js","/img/**","/fonts/**").permitAll()
                .antMatchers(HttpMethod.GET,"/user/getUserInfo").permitAll()
                .antMatchers(HttpMethod.POST,"/user/logout","/user/updatePassword").authenticated()
                .antMatchers(HttpMethod.POST,"/uploadImage").permitAll()
                // 除上面外的所有请求全部需要认证访问
                .anyRequest().authenticated();

        //关闭Security默认的退出接口
        http.logout().disable();
        http.addFilterBefore(jwtAuthenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);

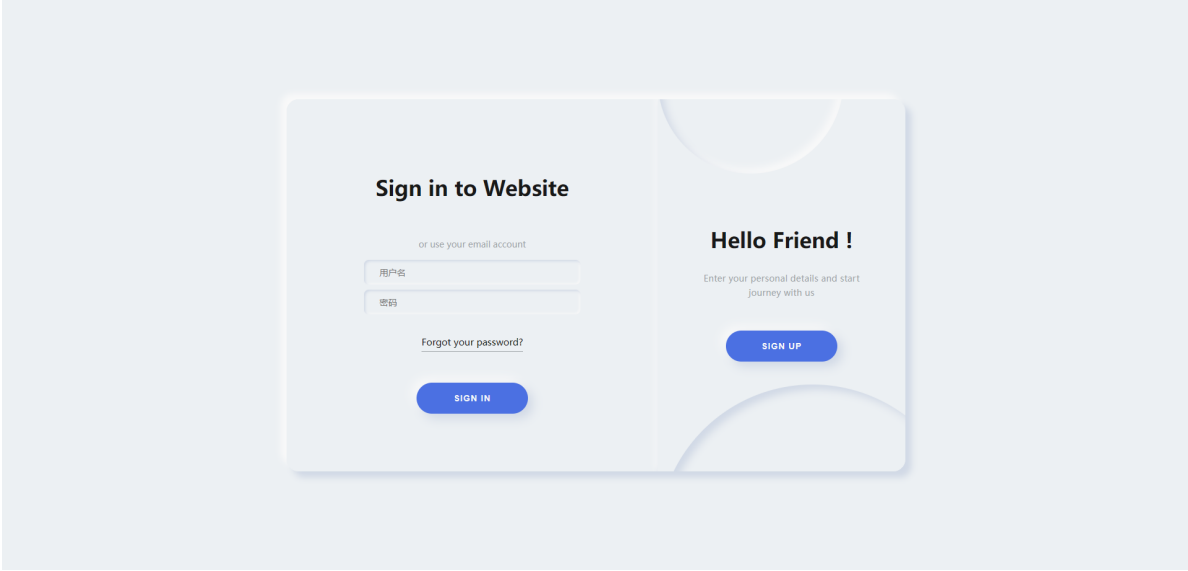
        http.exceptionHandling()
                .accessDeniedHandler(accessDeniedHandler)
                .authenticationEntryPoint(authenticationEntryPoint);
        //允许跨域
        http.cors();
    }

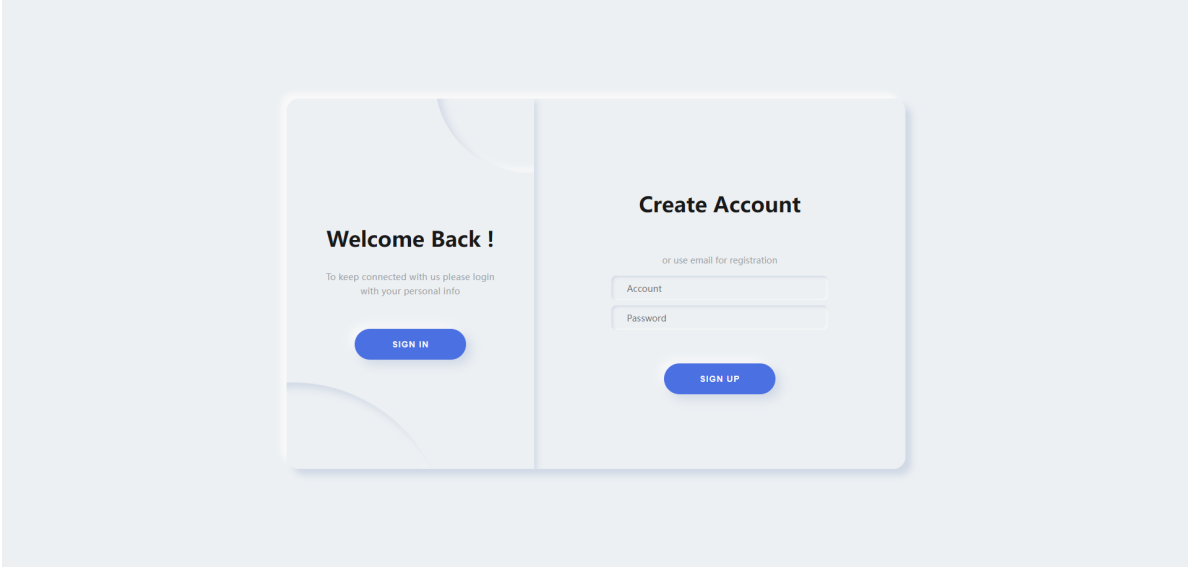

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
```

```
80 }
```

前端页面展示

登陆和注册





- **功能**:用户的登陆和注册
- 元素
  - 登陆表单

`<template>` 部分

1. `<div id="b-container" class="container b-container">`：定义了一个包含登录表单的容器。
2. `<form id="b-form" class="form" method="POST" action="">`：定义了一个登录表单，`method="POST"`表示表单提交时将使用POST请求，`action=""`属性为空，因为表单提交将通过JavaScript处理。
3. `<h2 class="form_title title">Sign in to Website</h2>` 和 `<span class="form__span">or use your email account</span>`：提供了登录表单的标题和副标题。

4. `<input v-model="loginFrom.userAccount" class="form__input" type="text" placeholder="用户名" />` 和 `<input v-model.lazy="loginFrom.userPassword" class="form__input" type="password" placeholder="密码" />`：定义了两个输入框，分别用于输入用户名和密码。`v-model` 用于创建数据的双向绑定，`v-model.lazy` 用于在 `input` 事件后更新数据。
5. `<a class="form__link">Forgot your password?</a>`：提供了一个忘记密码的链接。
6. `<button class="form__button button submit" @click.prevent="loginHandler()">SIGN IN</button>`：定义了一个提交按钮，`@click.prevent` 阻止了表单的默认提交行为，并调用了 `loginHandler` 方法。

`<script setup>` 部分

1. 导入必要的依赖：`ref`，`watch` 从 `vue`，`useMainStore` 从 `@/stores/message`，`storeToRefs` 从 `pinia`，`ElMessage` 从 `element-plus`，`login` 从 `@/common/api/user/index`，以及 `clearTokenAndUserInfo`，`saveUserInfo`，`setToken` 从 `../common/utils/auth.js`。
2. 创建 `mainStore` 实例，并使用 `storeToRefs` 来响应式地监听 `showSignup` 状态的变化。
3. 使用 `watch` 来监听 `showSignup` 的变化，并根据其值切换容器的CSS类。
4. 定义 `loginFrom` 响应式对象，用于存储表单数据。
5. 定义 `loginHandler` 方法，该方法执行登录操作：
    - 调用 `clearTokenAndUserInfo` 清除旧的认证信息。
    - 调用 `login` API方法，传入 `loginFrom.value`（包含用户名和密码）。
    - 如果登录成功（`res.code == 200`），则设置token，保存用户信息，并跳转到首页。
    - 如果登录失败，则显示错误消息。
    - 如果有错误发生，则在控制台打印错误并显示错误消息。
6. `router.push` 用于在登录成功后导航到首页。

```
<template>
  <div id="b-container" class="container b-container">
    <form id="b-form" class="form" method="POST" action="">
      <h2 class="form_title title">Sign in to Website</h2>
      <span class="form__span">or use your email account</span>
      <input v-model="loginFrom.userAccount"
class="form__input" type="text" placeholder="用户名" />
      <input v-model.lazy="loginFrom.userPassword"
class="form__input" type="password" placeholder="密码" />
      <a class="form__link">Forgot your password?</a>
      <button class="form__button button submit"
@click.prevent="loginHandler()">SIGN IN</button>
    </form>
  </div>
</template>

<script setup >
import {ref, watch} from 'vue'
import { useMainStore } from '@/stores/message'
import {storeToRefs} from "pinia";
import { ElMessage } from 'element-plus'
```

```
import {login} from "@/common/api/user/index";
import {clearTokenAndUserInfo, saveUserInfo, setToken} from "../common/utils/auth.js";
import router from "../router/index.js";

const mainStore = useMainStore()
const { showSignup } = storeToRefs(mainStore)
watch(showSignup, () => {
  const bContainer = document.querySelector('#b-container')
  bContainer.classList.toggle('is-txl')
  bContainer.classList.toggle('is-z200')
})

const loginFrom = ref({
  userAccount: '',
  userPassword: '',
})
let loginHandler = () =>{
  clearTokenAndUserInfo();
  login(loginFrom.value).then(res =>{
    if (res.code == 200){
      setToken(res.data.token);
      saveUserInfo(res.data.userInfo)

      router.push({
        name:'index',
        path:'/index'
      });
      ElMessage({
        message: '登陆成功',
        type:'success'
      })
    }else {
      ElMessage({
        message: res.msg,
        type:'error'
      })
    }

  }).catch(err =>{
    console.log(err)
    ElMessage({
      message: err,
      type:'error'
    })
  })
}
</script>
```

- 注册表单

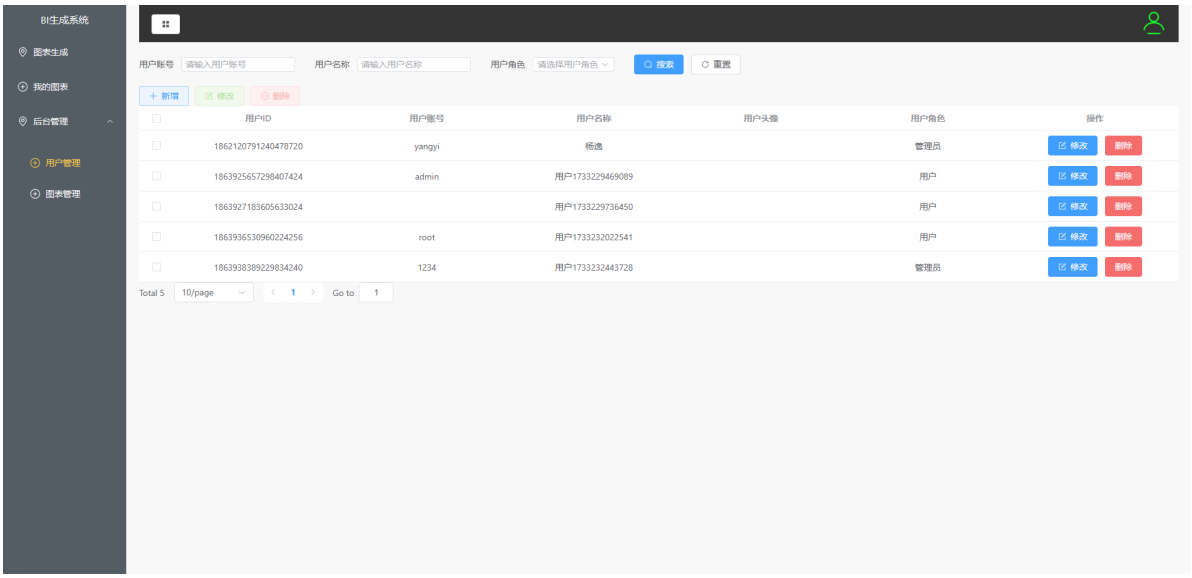
```
<template>
  <div id="a-container" class="container a-container">
    <form id="a-form" class="form" method="POST" action="">
```

```
      <h2 class="form_title title">Create Account</h2>
      <span class="form__span">or use email for registration</span>
      <input class="form__input" v-model="user.userAccount" type="text" placeholder="Account" />
      <input class="form__input" v-model.lazy="user.userPassword" type="password" placeholder="Password" />
      <button class="form__button button submit" @click.prevent="registerHandler()">SIGN UP</button>
    </form>
  </div>
</template>

<script setup >
import { useMainStore } from '@/stores/message'
import {ref, watch} from "vue";
import {storeToRefs} from "pinia";
import {ElMessage} from "element-plus";
import {userRegister} from '@/common/api/user/index'
const mainStore = useMainStore()
const { showSignup } = storeToRefs(mainStore)
watch(showSignup, () => {
  const aContainer = document.querySelector('#a-container')
  aContainer.classList.toggle('is-txl')
})
const user = ref({
  userName:'用户'+Date.now(),
  userAccount: '',
  userPassword: '',
})
function registerHandler(){
  userRegister(user.value).then(res =>{
    if (res.code == 200){
      ElMessage({
        message: '注册成功',
        type:'success'
      })
    }else {
      ElMessage({
        message: res.msg,
        type:'error'
      })
    }

  }).catch(err =>{
    console.log(err)
    ElMessage({
      message: err,
      type:'error'
    })
  })
}
</script>
```

用户列表页面



- **功能**：展示用户列表，提供搜索、分页功能。
- **元素**：
  - 用户信息表格。

```
    <el-table v-loading="loading" :data="userList" @selection-change="handleSelectionChange">
      <el-table-column type="selection" width="55" align="center" />
      <el-table-column label="用户ID" align="center" prop="id" />
      <el-table-column label="用户账号" align="center" prop="userAccount" />
      <el-table-column label="用户名称" align="center" prop="userName" />
      <el-table-column label="用户头像" align="center" prop="userAvatar" />
      <el-table-column label="用户角色" align="center" prop="userRole" />
      <el-table-column label="操作" align="center" class-name="small-padding fixed-width">
        <template #default="scope">
          <el-button
              type="primary"
              size="mini"
              @click="handleUpdate(scope.row)"
          >
            <template #icon>
              <el-icon-edit/>
            </template>
            修改</el-button>

          <el-popconfirm
              confirm-button-text="确定"
              cancel-button-text="取消"
              icon-color="#626AEF"
              title="确定删除这个用户吗?"
              @confirm="handleDelete(scope.row)"
```

```
              @cancel="cancelEvent"
            >
              <InfoFilled/>
              <template #reference>
                <el-button type="danger">删除</el-button>
              </template>
            </el-popconfirm>
          </template>
        </el-table-column>
      </el-table>
```

- 新增、编辑、删除按钮。

```
<el-row :gutter="10" class="mb8">
    <el-col :span="1.5">
      <el-button
          type="primary"
          plain
          size="mini"
          @click="handleAdd"
      >
        <template #icon>
          <el-icon><Plus /></el-icon>
        </template>
        <template #default>
          新增
        </template>
      </el-button>
    </el-col>
    <el-col :span="1.5">
      <el-button
          type="success"
          plain
          size="mini"
          :disabled="single"
          @click="handleUpdate"
      >
        <template #icon>
          <el-icon><Edit /></el-icon>
        </template>
        <template #default>
          修改
        </template>
      </el-button>
    </el-col>
    <el-col :span="1.5">
      <el-button
          type="danger"
          plain
          size="mini"
          :disabled="multiple"
          @click="handleDelete"
      >
        <template #icon>
          <el-icon><Remove /></el-icon>
```

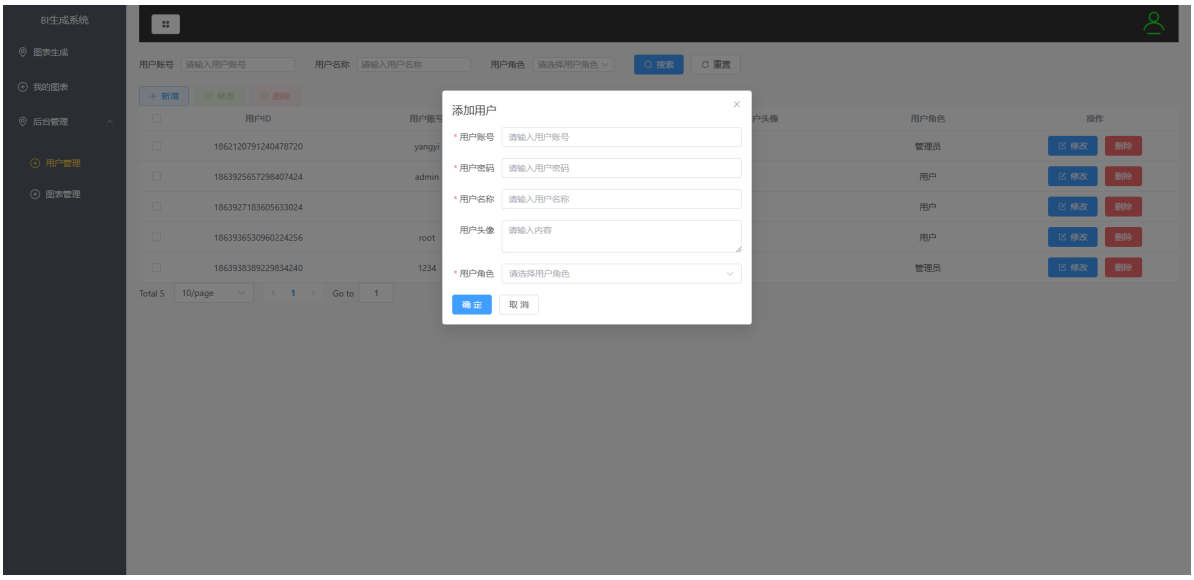
```
        </template>
        <template #default>
          删除
        </template>
      </el-button>
    </el-col>
    <right-toolbar :showSearch.sync="showSearch" @queryTable="getList"></right-toolbar>
  </el-row>
```

- 用户条件搜索,用户角色筛选。

```
<div class="app-container">
    <el-form :model="queryParams" ref="queryForm" size="small" :inline="true" v-show="showSearch" label-width="68px">
      <el-form-item label="用户账号" prop="userAccount">
        <el-input
            v-model="queryParams.userAccount"
            placeholder="请输入用户账号"
            clearable
            @keyup.enter.native="handleQuery"
        />
      </el-form-item>
      <el-form-item label="用户名称" prop="userName">
        <el-input
            v-model="queryParams.userName"
            placeholder="请输入用户名称"
            clearable
            @keyup.enter.native="handleQuery"
        />
      </el-form-item>
      <el-form-item label="用户角色" prop="userRole" style="width: 200px">
        <el-select v-model="queryParams.userRole" placeholder="请选择用户角色" clearable>
          <el-option :value="'用户'" :label="'用户'" :key="'用户'"/>
          <el-option :value="'管理员'" :label="'管理员' " :key="'管理员'"/>
        </el-select>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" size="mini" @click="handleQuery">
          <template #icon>
            <el-icon><Search /></el-icon>
          </template>
          <template #default>
            搜索
          </template>
        </el-button>
        <el-button size="mini" @click="resetQuery">
          <template #icon>
            <el-icon><RefreshRight /></el-icon>
          </template>
```

```
          <template #default>
            重置
          </template>
        </el-button>
      </el-form-item>
    </el-form>
```

用户详情页面



- **功能**：展示和编辑用户详细信息。
- **元素**：
  - 用户账号、密码、名称、头像、角色的输入框。
    -

```
<el-dialog :title="title" :model-value="open" width="500px" append-to-body>
      <el-form ref="form" :model="form" :rules="rules" label-width="80px">
        <el-form-item label="用户账号" prop="userAccount">
          <el-input v-model="form.userAccount" placeholder="请输入用户账号" />
        </el-form-item>
        <el-form-item label="用户密码" prop="userPassword">
          <el-input v-model="form.userPassword" placeholder="请输入用户密码" />
        </el-form-item>
        <el-form-item label="用户名称" prop="userName">
          <el-input v-model="form.userName" placeholder="请输入用户名称" />
        </el-form-item>
        <el-form-item label="用户头像" prop="userAvatar">
          <el-input v-model="form.userAvatar" type="textarea" placeholder="请输入内容" />
        </el-form-item>
        <el-form-item label="用户角色" prop="userRole">
          <!--<el-input v-model="form.userRole" placeholder="请输入用户角色" />-->
          <el-select v-model="form.userRole" placeholder="请选择用户角色">
            <el-option :value="'用户'" :label="'用户'"/>
```

```
            <el-option :value="'管理员'" :label="'管理员'"/>
          </el-select>
        </el-form-item>
      </el-form>
      <div slot="footer" class="dialog-footer">
        <el-button type="primary" @click="submitForm">确 定</el-button>
        <el-button @click="cancel">取 消</el-button>
      </div>
    </el-dialog>
```

- 保存和取消按钮。

完整页面代码

```
<template>
  <div class="app-container">
    <el-form :model="queryParams" ref="queryForm" size="small" :inline="true" v-show="showSearch" label-width="68px">
      <el-form-item label="用户账号" prop="userAccount">
        <el-input
            v-model="queryParams.userAccount"
            placeholder="请输入用户账号"
            clearable
            @keyup.enter.native="handleQuery"
        />
      </el-form-item>
      <el-form-item label="用户名称" prop="userName">
        <el-input
            v-model="queryParams.userName"
            placeholder="请输入用户名称"
            clearable
            @keyup.enter.native="handleQuery"
        />
      </el-form-item>
      <el-form-item label="用户角色" prop="userRole" style="width: 200px">
        <!--<el-input-->
        <!--    v-model="queryParams.userRole"-->
        <!--    placeholder="请输入用户角色"-->
        <!--    clearable-->
        <!--    @keyup.enter.native="handleQuery"-->
        <!--/>-->
        <el-select v-model="queryParams.userRole" placeholder="请选择用户角色" clearable>
          <el-option :value="'用户'" :label="'用户'" :key="'用户'"/>
          <el-option :value="'管理员'" :label="'管理员' " :key="'管理员'"/>
        </el-select>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" size="mini" @click="handleQuery">
          <template #icon>
            <el-icon><Search /></el-icon>
          </template>
```

```
          <template #default>
            搜索
          </template>
        </el-button>
        <el-button size="mini" @click="resetQuery">
          <template #icon>
            <el-icon><RefreshRight /></el-icon>
          </template>
          <template #default>
            重置
          </template>
        </el-button>
      </el-form-item>
    </el-form>

    <el-row :gutter="10" class="mb8">
      <el-col :span="1.5">
        <el-button
            type="primary"
            plain
            size="mini"
            @click="handleAdd"
        >
          <template #icon>
            <el-icon><Plus /></el-icon>
          </template>
          <template #default>
            新增
          </template>
        </el-button>
      </el-col>
      <el-col :span="1.5">
        <el-button
            type="success"
            plain
            size="mini"
            :disabled="single"
            @click="handleUpdate"
        >
          <template #icon>
            <el-icon><Edit /></el-icon>
          </template>
          <template #default>
            修改
          </template>
        </el-button>
      </el-col>
      <el-col :span="1.5">
        <el-button
            type="danger"
            plain
            size="mini"
            :disabled="multiple"
            @click="handleDelete"
        >
```

```
          <template #icon>
            <el-icon><Remove /></el-icon>
          </template>
          <template #default>
            删除
          </template>
        </el-button>
      </el-col>
      <right-toolbar :showSearch.sync="showSearch" @queryTable="getList"></right-toolbar>
    </el-row>

    <el-table v-loading="loading" :data="userList" @selection-change="handleSelectionChange">
      <el-table-column type="selection" width="55" align="center" />
      <el-table-column label="用户ID" align="center" prop="id" />
      <el-table-column label="用户账号" align="center" prop="userAccount" />
      <el-table-column label="用户名称" align="center" prop="userName" />
      <el-table-column label="用户头像" align="center" prop="userAvatar" />
      <el-table-column label="用户角色" align="center" prop="userRole" />
      <el-table-column label="操作" align="center" class-name="small-padding fixed-width">
        <template #default="scope">
          <el-button
              type="primary"
              size="mini"
              @click="handleUpdate(scope.row)"
          >
            <template #icon>
              <el-icon-edit/>
            </template>
            修改</el-button>

          <el-popconfirm
              confirm-button-text="确定"
              cancel-button-text="取消"
              icon-color="#626AEF"
              title="确定删除这个用户吗?"
              @confirm="handleDelete(scope.row)"
              @cancel="cancelEvent"
          >
            <InfoFilled/>
            <template #reference>
              <el-button type="danger">删除</el-button>
            </template>
          </el-popconfirm>
        </template>
      </el-table-column>
    </el-table>

    <el-pagination
```

```
        v-show="total>0"
        v-model:current-page="queryParams.currentPage"
        v-model:page-size="queryParams.pageSize"
        :page-sizes="[10, 15, 30]"
        layout="total, sizes, prev, pager, next, jumper"
        :total="total"
        @size-change="getList"
        @current-change="getList"
    />

    <!-- 添加或修改用户对话框 -->
    <el-dialog :title="title" :model-value="open" width="500px" append-to-body>
      <el-form ref="form" :model="form" :rules="rules" label-width="80px">
        <el-form-item label="用户账号" prop="userAccount">
          <el-input v-model="form.userAccount" placeholder="请输入用户账号" />
        </el-form-item>
        <el-form-item label="用户密码" prop="userPassword">
          <el-input v-model="form.userPassword" placeholder="请输入用户密码" />
        </el-form-item>
        <el-form-item label="用户名称" prop="userName">
          <el-input v-model="form.userName" placeholder="请输入用户名称" />
        </el-form-item>
        <el-form-item label="用户头像" prop="userAvatar">
          <el-input v-model="form.userAvatar" type="textarea" placeholder="请输入内容" />
        </el-form-item>
        <el-form-item label="用户角色" prop="userRole">
          <!--<el-input v-model="form.userRole" placeholder="请输入用户角色" />-->
          <el-select v-model="form.userRole" placeholder="请选择用户角色">
            <el-option :value="'用户'" :label="'用户'"/>
            <el-option :value="'管理员'" :label="'管理员'"/>
          </el-select>
        </el-form-item>
      </el-form>
      <div slot="footer" class="dialog-footer">
        <el-button type="primary" @click="submitForm">确 定</el-button>
        <el-button @click="cancel">取 消</el-button>
      </div>
    </el-dialog>
  </div>
</template>

<script>
import { listUser, getUser, delUser, addUser, updateUser } from "@/common/api/user";
import {ElMessage} from "element-plus";

export default {
```

```
  name: "User",
  data() {
    return {
      // 遮罩层
      loading: true,
      // 选中数组
      ids: [],
      // 非单个禁用
      single: true,
      // 非多个禁用
      multiple: true,
      // 显示搜索条件
      showSearch: true,
      // 总条数
      total: 0,
      // 用户表格数据
      userList: [],
      // 弹出层标题
      title: "",
      // 是否显示弹出层
      open: false,
      // 查询参数
      queryParams: {
        currentPage: 1,
        pageSize: 10,
        userAccount: null,
        userName: null,
        userRole: null,
      },
      // 表单参数
      form: {},
      // 表单校验
      rules: {
        userAccount: [
          { required: true, message: "用户账号不能为空", trigger: "blur" }
        ],
        userPassword: [
          { required: true, message: "用户密码不能为空", trigger: "blur" }
        ],
        userName: [
          { required: true, message: "用户名称不能为空", trigger: "blur" }
        ],
        userRole: [
          { required: true, message: "用户角色不能为空", trigger: "blur" }
        ],
      }
    };
  },
  created() {
    this.getList();
  },
```

```
  methods: {
    /** 查询用户列表 */
    getList() {
      this.loading = true;
      listUser(this.queryParams).then(response => {
        this.userList = response.data.records;
        this.total = response.data.total;
        this.loading = false;
      });
    },
    // 取消按钮
    cancel() {
      this.open = false;
      this.reset();
    },
    // 表单重置
    reset() {
      this.form = {
        id: null,
        userAccount: null,
        userPassword: null,
        userName: null,
        userAvatar: null,
        userRole: null,
        createTime: null,
        updateTime: null,
        deleteFlag: null
      };
      // this.resetForm("form");
      // this.$refs.form.resetFields();
      // this.$refs['form'].resetFields();
    },
    /** 搜索按钮操作 */
    handleQuery() {
      this.queryParams.currentPage = 1;
      this.getList();
    },
    /** 重置按钮操作 */
    resetQuery() {
      // this.resetForm("queryForm");
      // this.$refs['queryForm'].resetFields();
      this.queryParams = {
        currentPage: 1,
        pageSize: 10,
        userAccount: null,
        userName: null,
        userRole: null,
      }
      this.handleQuery();
    },
    // 多选框选中数据
    handleSelectionChange(selection) {
      this.ids = selection.map(item => item.id)
      this.single = selection.length!==1
      this.multiple = !selection.length
```

```
    },
    /** 新增按钮操作 */
    handleAdd() {
      this.reset();
      this.open = true;
      this.title = "添加用户";
    },
    /** 修改按钮操作 */
    handleUpdate(row) {
      this.reset();
      const id = row.id || this.ids
      getUser(id).then(response => {
        this.form = response.data;
        this.open = true;
        this.title = "修改用户";
      });
    },
    /** 提交按钮 */
    submitForm() {
      this.$refs["form"].validate(valid => {
        if (valid) {
          if (this.form.id != null) {
            updateUser(this.form).then(response => {
              // this.$modal.msgSuccess("修改成功");
              ElMessage({
                message: '修改成功',
                type: 'success',
              });
              this.open = false;
              this.getList();
            });
          } else {
            addUser(this.form).then(response => {
              // this.$modal.msgSuccess("新增成功");
              ElMessage({
                message: '新增成功',
                type: 'success',
              });
              this.open = false;
              this.getList();
            });
          }
        }
      });
    },
    /** 删除按钮操作 */
    handleDelete(row) {
      const ids = row.id || this.ids;
      delUser(ids).then(res => {
        if (res.code == 200){
          ElMessage({
            message: '删除成功',
            type: 'success',
          });
        }else {
```

```
          ElMessage({
            message: res.msg,
            type: 'error',
          });
        }
      }).error(err => {
        ElMessage({
            message: '删除失败',
            type: 'error',
        });
      })
      this.$modal.confirm('是否确认删除用户编号为"' + ids + '"的数据项？').then(function() {
        return delUser(ids);
      }).then(() => {
        this.getList();
        this.$modal.msgSuccess("删除成功");
      }).catch(() => {});
    },
  }
};
</script>

```

用户管理模块提供了一套完整的用户信息管理功能，包括用户登录、注册、信息查看、编辑和删除。

通过严格的权限控制，确保了系统的安全性。前端页面友好、易用，提供了良好的用户体验。

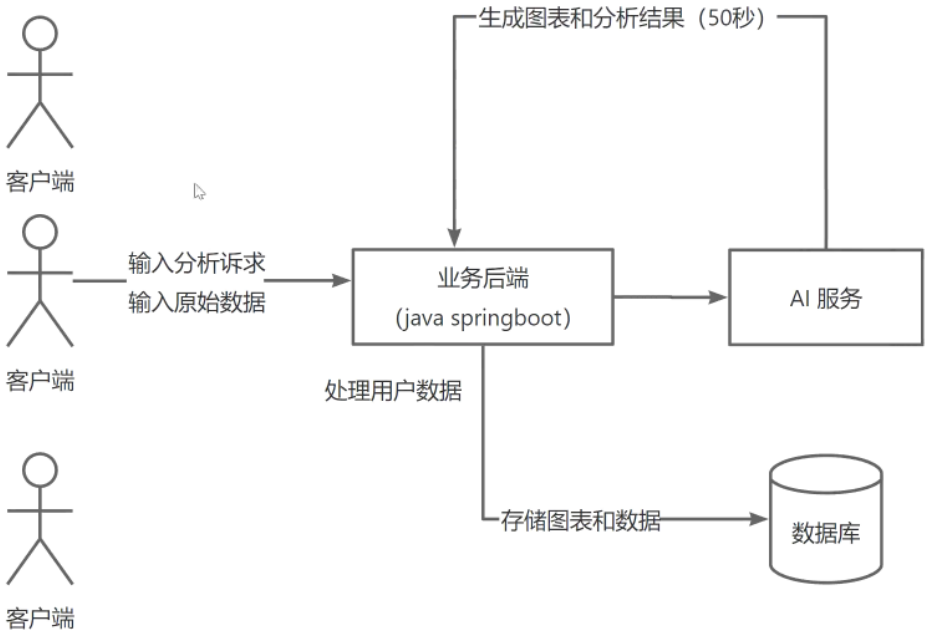
4.2 数据分析

后端

- 数据分析和图表生成模块提供了完整的数据处理和可视化功能。
- 通过令牌桶限流和线程池优化，确保了接口的稳定性和高并发处理能力
- **同步与异步**：用户可以通过ChartController上传数据文件并请求图表生成，支持同步和异步两种模式。
  - Controller
  - 一个生成图表的接口，只有具有“用户”角色的用户才能访问。
  - 它处理文件上传，执行限流检查，验证文件类型和大小，然后根据是否异步生成图表，并返回相应的结果
    - ```
      @PreAuthorize("@ps.hasRole('用户')")
      @PostMapping("/generateChartByAI")
      @ResponseBody
      public ResponseResult generateChartByAI(ChartDTO chartDTO, MultipartFile file) throws IOException {
          log.info("分析目标:{},图标名称:{},是否异步{}",chartDTO.getAnalysisTarget(),chartDTO.getName(),chartDTO.getIsAsynchronism());
          //限流判断
      ```
      ```

```
        if (!rRateLimiterHandler.accessAble("generateChartByAI_"+SecurityUtils.getUserId())){
            return ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.RATE_LIMIT_ERROR);
        }
        if (file == null || file.isEmpty()) {
            return ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.FILE_NOT_NULL);
        }
        //文件类判断
        if (!file.getContentType().equals("application/vnd.ms-excel") && !file.getContentType().equals("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")) {
            return ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.FILE_TYPE_ERROR);
        }
        //文件大小判断
        if (file.getSize()>1024*1024*2L) {
            return ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.FILE_SIZE_ERROR);
        }
        if (chartDTO.getIsAsynchronism()){
            long id = chartService.generateChartByAIAsyn(chartDTO,file);
            return ResponseResult.okResult("任务提交成功，请稍后查看结果",String.valueOf(id));
        }else {
            ChartVO vo = chartService.generateChartByAI(chartDTO,file);
            return ResponseResult.okResult(vo);
        }
    }
```

- **同步**
-



- 实现了通过AI生成图表的完整流程，包括读取和验证上传的数据、构建AI请求、发送请求、解析响应、数据可视化处理和更新数据库
    1. **初始化和数据准备**：
        - 生成一个唯一的图表ID。
        - 从 `ChartDTO` 复制数据到 `Chart` 实体对象，并设置必要的属性，如ID、用户ID和状态（初始状态为“等待中”）。
    2. **读取和转换文件**：
        - 从上传的Excel文件中读取数据，并将其转换为CSV格式的字符串数据。
    3. **数据验证**：
        - 检查CSV数据的长度，如果超过3000字符，则更新图表状态为“失败”，记录错误信息，并抛出异常。
    4. **保存初始图表状态**：
        - 将CSV数据保存到图表实体的 `chartData` 属性中，并将图表实体保存到数据库。
    5. **构建AI请求**：
        - 构建一个包含原始数据、分析目标和图表类型的请求消息，用于向AI发送请求。
    6. **发送AI请求**：
        - 使用 `AiUtil.doChat` 方法发送构建好的请求数据到AI服务，并接收响应数据。
    7. **解析AI响应**：
        - 解析AI返回的数据，提取图表生成的内容（通常是一个Echarts的option代码）和分析结论。
    8. **错误处理**：
        - 如果AI返回的数据中没有包含预期的图表生成内容（即没有找到界定图表内容的```标记），则更新图表状态为“失败”，记录错误信息，并抛出异常。
    9. **更新图表实体**：

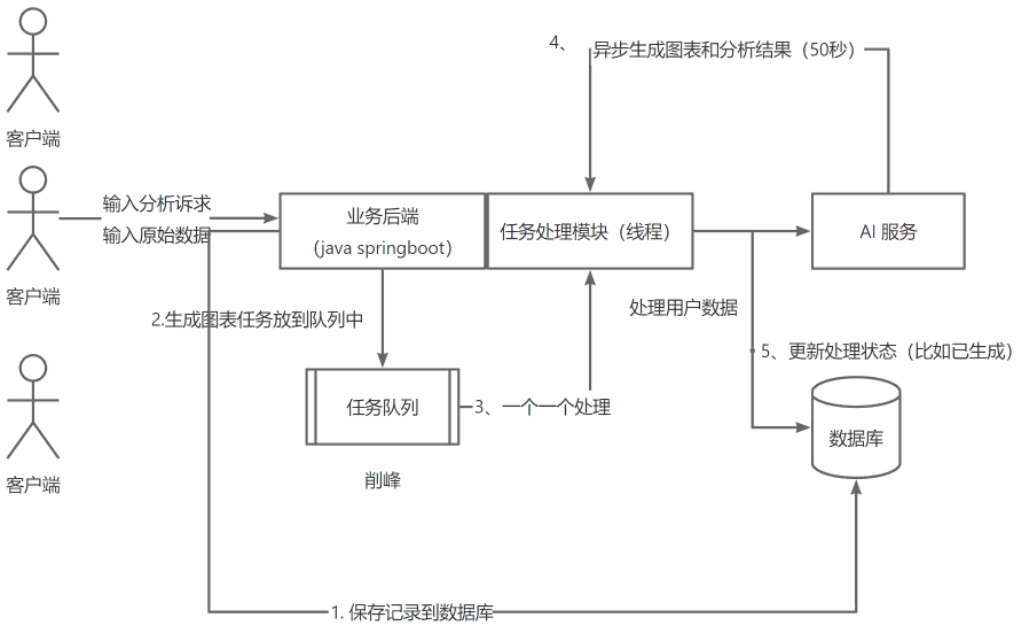
- 将提取的Echarts的option代码保存到图表实体的`generatedChartData`属性中。
- 将分析结论保存到图表实体的`analysisConclusion`属性中。
- 更新图表实体的状态为“成功”，并记录图表生成成功的信息。

10. **保存最终图表状态**：
    - 将更新后的图表实体保存到数据库。
11. **返回结果**：
    - 将图表实体转换为`ChartVO`对象，以便将生成的图表数据和结论返回给前端

```
    @Override
    public ChartVO generateChartByAI(ChartDTO chartDTO, MultipartFile file) {
        //读数据
        String csvData  = "";
        long charId = IdUtil.getSnowflake(1, 1).nextId();
        Chart chart = BeanCopyUtil.copyBean(chartDTO, Chart.class);
        chart.setId(charId);
        chart.setUserId(SecurityUtils.getUserId());
        chart.setState("等待中");
        try {
            csvData = ExcelUtils.excel2csv(file.getInputStream());
            log.info("上传的数据为:\n{}", csvData);
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (csvData.length()>3000){
            chart.setState("失败");
            chart.setExecuteMessage("数据量过大，请上传小于3000行的数据");
            chart.setChartData("");
            save(chart);
            throw new SystemException(500, "数据量过大，请上传小于3000行的数据");
        }
        chart.setChartData(csvData);
        save(chart);
        StringBuilder message = new StringBuilder("原始数据:\n");
        message.append(csvData);
        message.append("分析目标:\n");
        message.append(chartDTO.getAnalysisTarget());
        message.append("\n.使用").append(chartDTO.getChartType()).append("进行可视化分析.\n");
        chart.setState("生成中");
        chart.setExecuteMessage("AI正在生成图表");
        updateById(chart);
        //配置prompt向AI发送请求
        HttpRequestData requestData = AiUtil.createDefaultRequestData(message.toString());
        HttpResponseData responseData = AiUtil.doChat(requestData);
        //解析AI返回的数据
```

```
37          //ChartVO chartVO = BeanCopyUtil.copyBean(chartDTO,
ChartVO.class);
38          String content =
responseData.getChoices().get(0).getMessage().getContent();
39          log.info("AI返回的数据为:{}", content);
40          int index = content.indexOf("```");
41          int endIndex = content.lastIndexOf("```");
42          if (index == -1 || endIndex == -1){
43              chart.setState("失败");
44              chart.setExecuteMessage("AI生成图表失败");
45              updateById(chart);
46              throw new SystemException(500, "AI生成图表失败");
47          }
48          //数据可视化,Echarts的option代码
49          chart.setGeneratedChartData(content.substring(index+7,
endIndex).trim());
50          index = endIndex;
51          //分析结论
52
 chart.setAnalysisConclusion(content.substring(index+3).trim())
;
53
54          //更新数据库
55          chart.setState("成功");
56          chart.setExecuteMessage("图表生成成功");
57          updateById(chart);
58          ChartVO chartVO =  BeanCopyUtil.copyBean(chart,
ChartVO.class);
59          return chartVO;
60      }
```

- **异步**
-



  - 创建线程池

- 这个配置类定义了一个自定义的线程池，包括核心线程数、最大线程数、工作队列、线程工厂和拒绝策略。这个线程池可以被应用程序中的其他组件使用，以异步执行任务
  - 线程池参数配置：
    - `corePoolSize`：核心线程池的大小为1，表示线程池中始终保持的线程数量。
    - `maximumPoolSize`：最大线程池的大小为2，表示线程池中允许的最大线程数量。
    - `keepAliveTime`：非核心线程空闲存活时间设置为100秒。
    - `unit`：时间单位设置为秒。
    - `workQueue`：工作队列使用`ArrayBlockingQueue`，容量为5，用于存放待执行任务。
    - `threadFactory`：线程工厂，用于创建新线程。这里没有自定义线程名称或其他属性，直接使用默认的线程创建方式。
- ```java
  package space.anyi.BI.config;

  import org.slf4j.Logger;
  import org.slf4j.LoggerFactory;
  import org.springframework.context.annotation.Bean;
  import
  org.springframework.context.annotation.Configuration;

  import java.util.concurrent.*;

  @Configuration
  public class ThreadPoolExecutorConfig {
      private final static Logger log =
  LoggerFactory.getLogger(ThreadPoolExecutorConfig.class);
      @Bean
      public ThreadPoolExecutor getThreadPoolExecutor(){
          //核心线程数
          int corePoolSize = 1;
          //最大线程数
          int maximumPoolSize = 2;
          //非核心线程存活空闲时间
          long keepAliveTime = 100L;
          //时间单位
          TimeUnit unit = TimeUnit.SECONDS;
          //任务队列
          BlockingQueue<Runnable> workQueue = new
  ArrayBlockingQueue<>(5);
          //线程工厂
          ThreadFactory threadFactory = new ThreadFactory() {
              @Override
              public Thread newThread(Runnable r) {
                  Thread thread = new Thread(r);
                  return thread;
              }
          };
          //拒绝策略处理器
          //RejectedExecutionHandler handler= new
  ThreadPoolExecutor.CallerRunsPolicy();
  ```
  ```

```
        RejectedExecutionHandler handler= new RejectedExecutionHandler(){
            @Override
            public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
                int activeCount = executor.getActiveCount();
                int corePoolSize = executor.getCorePoolSize();
                int maximumPoolSize = executor.getMaximumPoolSize();
                int queueSize = executor.getQueue().size();
                log.warn("任务被拒绝执行,当前工作线程数:{},核心线程数:{},最大线程数:{},队列大小:{}",activeCount,corePoolSize,maximumPoolSize,queueSize);
            }
        };
        //创建线程池
        ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(corePoolSize,maximumPoolSize,keepAliveTime,unit,workQueue,threadFactory);
        return threadPoolExecutor;
    }
}

```

- 使用线程池进行异步优化
  - 实现了异步生成图表的流程，包括读取和验证上传的数据、保存初始图表状态、使用线程池异步执行AI请求、解析响应、数据可视化处理和更新数据库。通过异步处理，可以提高应用的性能，避免在生成图表时阻塞主线程。

1. `public long generateChartByAIAsyn(ChartDTO chartDTO, MultipartFile file)`：这是一个公共方法，返回类型是`long`，参数包括一个`ChartDTO`对象和一个`MultipartFile`对象，用于生成图表的异步操作。
2. 初始化图表实体：
   - 使用`BeanCopyUtil.copyBean`方法将`ChartDTO`对象的属性复制到一个新的`Chart`实体对象中。
   - 生成一个唯一的图表ID，并设置到`Chart`实体对象中。
   - 设置用户ID和图表状态为“等待中”。
3. 读取数据：
   - 尝试将上传的Excel文件转换为CSV格式的字符串数据，并记录日志。
   - 如果转换过程中发生`IOException`，则打印堆栈跟踪。
4. 数据量检查：
   - 如果CSV数据长度超过3000字符，则更新图表状态为“失败”，保存到数据库，并抛出异常。
5. 保存图表实体：
   - 将CSV数据设置为图表实体的`chartData`属性，并保存到数据库。
6. 异步处理：
   - 使用线程池`threadPoolExecutor`来异步执行图表生成逻辑。
7. 异步执行的逻辑：

- 更新图表状态为“生成中”并保存。
- 构建请求消息，包含原始数据、分析目标和图表类型。
- 发送请求到AI服务，并接收响应数据。
- 解析AI返回的数据，提取图表生成的内容和分析结论。
- 如果AI返回的数据格式不正确（没有找到预期的```标记），则更新图表状态为“失败”并抛出异常。
- 提取Echarts的option代码和分析结论，并更新图表实体的相关属性。
- 更新图表状态为“成功”并保存。

8. 返回结果：

- 返回生成的图表ID

```
    @Override
    public long generateChartByAIAsyn(ChartDTO chartDTO,
MultipartFile file) {

        Chart chart = BeanCopyUtil.copyBean(chartDTO,
Chart.class);
        long chartId = IdUtil.getSnowflake(1, 1).nextId();
        chart.setId(chartId);
        chart.setUserId(SecurityUtils.getUserId());
        chart.setState("等待中");
        //读数据
        String csvData  = "";
        try {
            csvData =
ExcelUtils.excel2csv(file.getInputStream());
            log.info("上传的数据为:\n{}", csvData);
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (csvData.length()>3000){
            chart.setChartData("");
            chart.setState("失败");
            chart.setExecuteMessage("数据量过大，请上传小于3000行的
数据");
            save(chart);
            throw new SystemException(500, "数据量过大，请上传小于
3000行的数据");
        }
        chart.setChartData(csvData);
        save(chart);
        //使用线程池优化生成图表的逻辑
        String finalCsvData = csvData;
        threadPoolExecutor.execute(()->{
            chart.setState("生成中");
           updateById(chart);

            StringBuilder message = new StringBuilder("原始数
据:\n");
            message.append(finalCsvData);
            message.append("分析目标:\n");
            message.append(chartDTO.getAnalysisTarget());
            message.append("\n.使
用").append(chartDTO.getChartType()).append("进行可视化分析.\n");
            //配置prompt向AI发送请求
```

```
            HttpRequestData requestData = AiUtil.createDefaultRequestData(message.toString());
            HttpResponseData responseData = AiUtil.doChat(requestData);
            //解析AI返回的数据
            String content = responseData.getChoices().get(0).getMessage().getContent();
            log.info("AI返回的数据为:{}", content);
            int index = content.indexOf("```");
            int endIndex = content.lastIndexOf("```");
            if (index == -1 || endIndex == -1){
                chart.setState("失败");
                chart.setExecuteMessage("AI生成图表失败");
                updateById(chart);
                throw new SystemException(500, "AI生成图表失败");
            }
            //数据可视化,Echarts的option代码

 chart.setGeneratedChartData(content.substring(index+7, endIndex).trim());
            index = endIndex;
            //分析结论

 chart.setAnalysisConclusion(content.substring(index+3).trim());
            //保存到数据库
            chart.setState("成功");
            chart.setExecuteMessage("AI生成图表成功");
            updateById(chart);
        });

        return chartId;
    }
```

- **限流**：使用Redisson实现令牌桶算法，通过RRateLimiterHandler进行接口限流。
  - 这个服务类提供了一个简单的限流功能，通过Redisson客户端实现。它允许开发者为特定的请求设置一个限流器，并且根据设定的速率来控制请求的通过。
  - `tryAcquire`方法是非阻塞的，它会立即返回一个布尔值，指示是否成功获取到令牌

```
package space.anyi.BI.handler.redisson;

import org.redisson.api.RRateLimiter;
import org.redisson.api.RateIntervalUnit;
import org.redisson.api.RateType;
import org.redisson.api.RedissonClient;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;

@Service
public class RRateLimiterHandler {
    @Resource
```

```
    private RedissonClient redissonClient;
    public boolean accessAble(String key){
        //获取限流器
        RRateLimiter rateLimiter = redissonClient.getRateLimiter(key);
        //设置限流器速率
        //rateLimiter.setRate(RateType.OVERALL,5,1, RateIntervalUnit.SECONDS);
        rateLimiter.trySetRate(RateType.OVERALL,5,1, RateIntervalUnit.SECONDS);
        //获取令牌,每次获取一个令牌,如果获取不到则返回false
        //return rateLimiter.tryAcquire(1);
        //阻塞获取令牌,每次获取一个令牌
        //rateLimiter.acquire();
        return rateLimiter.tryAcquire(1);
    }
}
```

- 调用AI进行数据分析
  - ```
    package space.anyi.BI.util;

    import cn.hutool.http.HttpResponse;
    import cn.hutool.http.HttpUtil;
    import com.fasterxml.jackson.core.JsonProcessingException;
    import com.fasterxml.jackson.databind.ObjectMapper;
    import space.anyi.BI.entity.ResponseResult;
    import space.anyi.BI.entity.xinghuo.HttpRequestData;
    import space.anyi.BI.entity.xinghuo.HttpRequestMessage;
    import space.anyi.BI.entity.xinghuo.HttpResponseData;
    import space.anyi.BI.exception.SystemException;

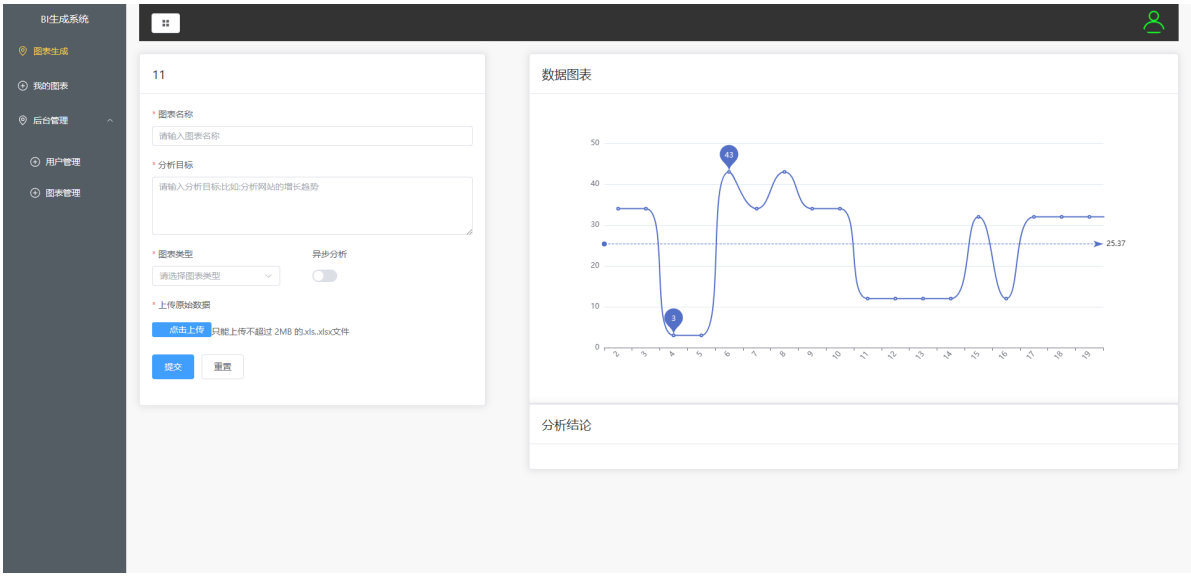
    import java.util.ArrayList;
    import java.util.List;

    public class AiUtil {
        private static final String url = "https://spark-api-open.xf-yun.com/v1/chat/completions";
        public static ObjectMapper objectMapper = new ObjectMapper();

        /**
         * 调用星火AI接口
         * @param requestData
         * @return {@code HttpResponseData }
         * @description:
         * @author: 杨逸
         * @data:2024/12/04 20:50:06
         * @since 1.0.0
         */
        public static HttpResponseData doChat(HttpRequestData requestData){
            String json = null;
            try {
                json = objectMapper.writeValueAsString(requestData);
            } catch (JsonProcessingException e) {
    ```
    ```

```
            e.printStackTrace();
        }
        HttpResponse httpResponse = HttpUtil.createPost(url)
                .header("Content-Type", "application/json")
                .header("Authorization", "Bearer token")
                .body(json)
                .execute();
        String body = httpResponse.body();
        int index = body.indexOf('0');
        HttpResponseData httpResponseData = null;
        if (index == 8){
            //调用成功
            try {
                httpResponseData = objectMapper.readValue(body,
HttpResponseData.class);
            } catch (JsonProcessingException e) {
                e.printStackTrace();
            }
        }else{
            System.err.println(body);
            throw new
SystemException(ResponseResult.AppHttpCodeEnum.SYSTEM_ERROR);
        }
        return httpResponseData;
    }

    /**
     * 创建一个简单的请求体
     * @param message
     * @return {@code HttpRequestData }
     * @description:
     * @author: 杨逸
     * @data:2024/12/04 20:54:46
     * @since 1.0.0
     */
    public static HttpRequestData createDefaultRequestData(String
message){
        HttpRequestData httpRequestData = new HttpRequestData();
        List<HttpRequestMessage> messages = new ArrayList<>();
        messages.add(HttpRequestMessage.getPromptMessage());
        messages.add(new
HttpRequestMessage(HttpRequestMessage.USER_ROLE, message));
        httpRequestData.setMessages(messages);
        return httpRequestData;
    }
}

```

前端



- 提供了一个完整的用户界面，用于上传数据、配置图表生成参数、提交表单、显示图表和分析结论

`<template>` 部分

1. 使用`el-row`和`el-col`创建布局，分为三个主要部分：表单区域、分隔符和图表显示区域。
2. 在表单区域中，使用`el-form`创建一个表单，包含图表名称、分析目标、图表类型、异步分析选项和文件上传等字段。
3. 图表类型使用`el-select`下拉选择框，异步分析使用`el-switch`开关。
4. 文件上传使用`el-upload`组件，限制上传文件大小不超过2MB，并且只能是`.xls`或`.xlsx`格式。
5. 提交和重置按钮用于提交表单或重置表单数据。
6. 图表显示区域中，使用`div`元素作为图表的容器，并在底部提供了一个按钮用于获取分析结果。
7. 分析结论部分，显示分析的结论文本。

`<script>` 部分

1. 导入了生成图表和获取图表结果的API方法，以及`ElMessage`用于显示消息提示。
2. 定义了组件的数据结构，包括表单数据、图表配置、分析结论等。
3. 定义了表单验证规则。
4. 定义了文件上传的action地址，这里使用的是一个占位符URL，实际开发中需要替换为真实的上传接口。
5. 提供了`submitForm`方法用于提交表单，根据是否异步分析调用不同的API，并处理响应结果。
6. `changeFile`方法用于更新表单中的文件数据。
7. `resetForm`方法用于重置表单。
8. `fileBeforeUpload`方法用于在文件上传前进行校验，确保文件大小和类型符合要求。
9. `chartRefresh`方法用于初始化或刷新图表。
10. `getChartResult`方法用于获取异步分析的结果，并更新图表和分析结论。

```
<template>
  <el-row>
    <el-col :span="8">
      <el-card>
        <template #header>11</template>
        <template #default>
```

```
          <el-form ref="chartForm" :model="formData" :rules="rules" size="medium" label-width="100px"
                   label-position="top">
            <el-form-item label="图表名称" prop="name">
              <el-input v-model="formData.name" placeholder="请输入图表名称" clearable :style="{width: '100%'}"></el-input>
            </el-form-item>
            <el-form-item label="分析目标" prop="analysisTarget">
              <el-input v-model="formData.analysisTarget" type="textarea" placeholder="请输入分析目标;比如:分析网站的增长趋势"
                        :autosize="{minRows: 4, maxRows: 4}" :style="{width: '100%'}"></el-input>
            </el-form-item>
            <el-row>
              <el-col :span="12">
                <el-form-item label="图表类型" prop="chartType">
                  <el-select v-model="formData.chartType" placeholder="请选择图表类型" clearable :style="{width: '80%'}">
                    <el-option v-for="(item, index) in chartTypeOptions" :key="index" :label="item.label"
                               :value="item.value" :disabled="item.disabled"></el-option>
                  </el-select>
                </el-form-item>
              </el-col>
              <el-col :span="12">
                <el-form-item label="异步分析" prop="isAsynchronism">
                  <el-switch v-model="formData.isAsynchronism" @change="console.log(formData.isAsynchronism)"/>
                </el-form-item>
              </el-col>
            </el-row>
            <el-form-item label="上传原始数据" prop="file" required>
              <el-upload ref="file" :file-list="filefileList" :action="fileAction" :auto-upload="false" limit="1"
                         :before-upload="fileBeforeUpload" @change="changeFile" accept=".xls,.xlsx">
                <el-button size="small" type="primary" icon="el-icon-upload">点击上传</el-button>
                <div slot="tip" class="el-upload__tip">只能上传不超过 2MB 的.xls,.xlsx文件</div>
              </el-upload>
            </el-form-item>
            <el-form-item size="large">
              <el-button type="primary" @click="submitForm">提交</el-button>
              <el-button @click="resetForm">重置</el-button>
            </el-form-item>
          </el-form>
        </template>
      </el-card>

    </el-col>
    <el-col :span="1">
      <!--<el-divider direction="vertical" style="height: 100vh"/>-->
    </el-col>
```

```
    <el-col :span="15">
      <el-row>
        <el-col :span="24">
          <el-card>
            <template #header>
              数据图表
            </template>
            <template #default>
              <div ref="chart" style="height:50vh"/>
            </template>
            <template #footer v-if="chartId">
              <el-button  type="primary" @click.prevent="getChartResult">获取分析结果</el-button>
            </template>
          </el-card>
        </el-col>
      </el-row>
      <el-row>
        <el-col :span="24">
          <el-card>
            <template #header>分析结论</template>
            <template #default>
              {{analysisConclusion}}
            </template>
          </el-card>
        </el-col>
      </el-row>
    </el-col>
  </el-row>

</template>

<script>
import {generateChartByAI, getChart} from "../common/api/chart/index.js";
import {ElMessage} from "element-plus";
import * as echarts from 'echarts';

export default {
  name: 'generateChart',
  components: {},
  props: [],
  data() {
    return {
      formData: {
        name: undefined,
        analysisTarget: undefined,
        chartType: undefined,
        file: null,
        isAsynchronism:false
      },
      option:{
        xAxis: {
          type: 'category',
          data: ['2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19'],
```

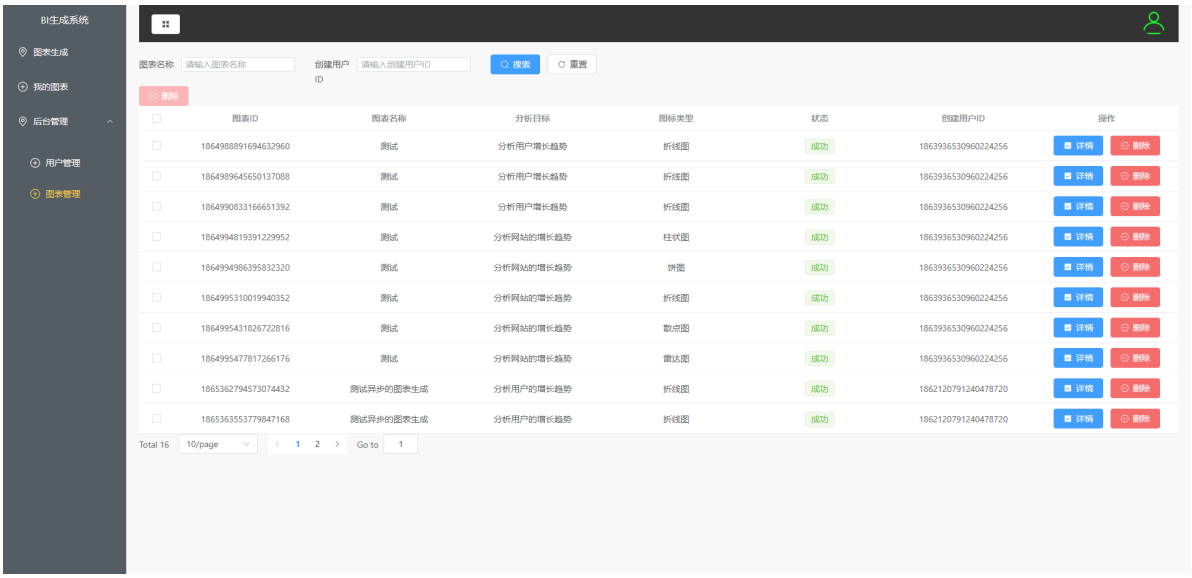
```
          boundaryGap: true,
          axisLabel:{
            interval:0,
            rotate : 50
          },
        },
        yAxis: {
          type: 'value'
        },
        series: [{
          name:'增长人数',
          type:'line',
          data:[34.0, 34.0, 3.0, 3.0, 43.0, 34.0, 43.0, 34.0, 34.0, 12.0, 12.0, 12.0, 12.0, 32.0, 12.0, 32.0, 32.0, 32.0, 32.0],
          markPoint: {
            data: [
              {type: 'max', name: '最大值'},
              {type: 'min', name: '最小值'}
            ]
          },
          markLine: {data: [
              {type: 'average', name: '平均值'}
            ]},
          smooth: true
        }]
      },
      analysisConclusion:'',
      chart:{},
      chartId:undefined,
      rules: {
        name: [{
          required: true,
          message: '请输入图表名称',
          trigger: 'blur'
        }],
        analysisTarget: [{
          required: true,
          message: '请输入分析目标;比如:分析网站的增长趋势',
          trigger: 'blur'
        }],
        chartType: [{
          required: true,
          message: '请选择图表类型',
          trigger: 'change'
        }],
      },
      fileAction: 'https://jsonplaceholder.typicode.com/posts/',
      filefileList: [],
      chartTypeOptions: [{
        "label": "折线图",
        "value": "折线图"
      }, {
        "label": "柱状图",
        "value": "柱状图"
      }, {
```

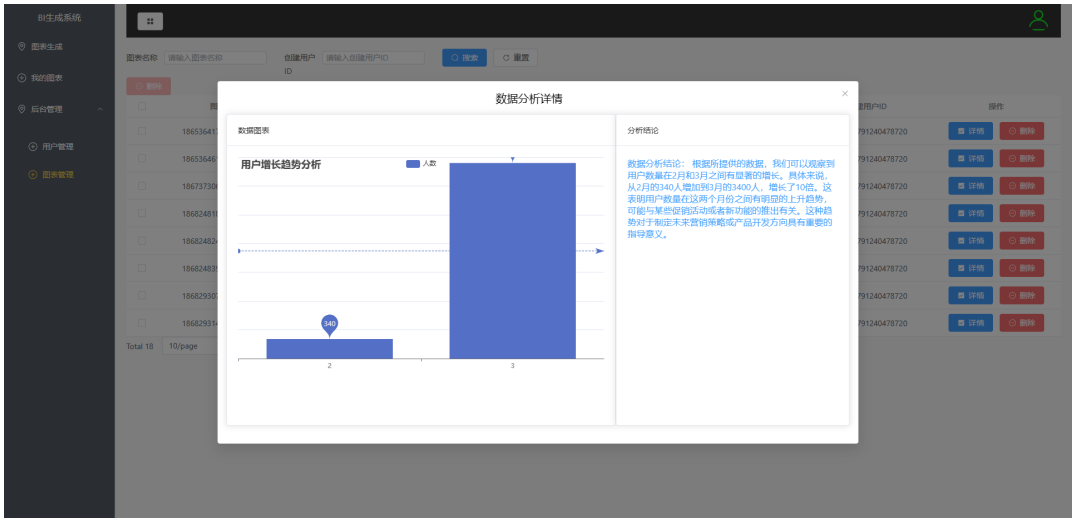
```
        "label": "饼图",
        "value": "饼图"
      }, {
        "label": "热力图",
        "value": "热力图"
      },{
        "label": "散点图",
        "value": "散点图"
      },{
        "label": "雷达图",
        "value": "雷达图"
      }],
    }
  },
  computed: {},
  watch: {},
  created() {

  },
  mounted() {
    this.chartRefresh();
  },
  methods: {
    submitForm() {
      console.log(this.formData)
      this.$refs['chartForm'].validate(valid => {
        if (!valid) return
        if (this.formData.isAsynchronism) {
          generateChartByAI(this.formData).then(res =>{
            if (res.code == 200){
              console.log(res)
              this.chartId = res.data;
              ElMessage({
                message: res.msg,
                type: 'success',
              });
            }else{
              ElMessage({
                message: res.msg,
                type: 'error',
              });
            }
          }).catch(err=>{
            ElMessage({
              message: err,
              type: 'error',
            });
          });
        } else {
          this.chartId = undefined;
          generateChartByAI(this.formData).then(res => {
          if (res.code === 200) {
            this.analysisConclusion = res.data.analysisConclusion;
            //取出配置对象的代码
            let generatedChartData = res.data.generatedChartData;
```

```
            //将配置对象的代码转换为JSON对象,不是不是标准的JSON对象,需要使用eval函数进行转换
            // let optionJson = JSON.parse(option);
            let optionJson = eval("(" + generatedChartData + ")")
            //返回格式不稳定,有可能会多封装一层option,需要判断一下
            if (optionJson.option === undefined || optionJson.option === null) {
              this.option = optionJson;
            } else {
              this.option = optionJson.option;
            }
            this.chartRefresh();
            ElMessage({
              message: res.msg,
              type: 'success',
            })
          } else {
            ElMessage({
              message: res.msg,
              type: 'error',
            });
          }
        }).catch(err => {
          ElMessage({
            message: err,
            type: 'error',
          });
        });
        }
      });
    },
    changeFile(file){
      this.formData.file = file.raw;
    },
    resetForm() {
      this.$refs['chartForm'].resetFields()
    },
    fileBeforeUpload(file) {
      let isRightSize = file.size / 1024 / 1024 < 2
      if (!isRightSize) {
        this.$message.error('文件大小超过 2MB')
      }
      let isAccept = new RegExp('.xls,.xlsx').test(file.type)
      if (!isAccept) {
        this.$message.error('应该选择.xls,.xlsx类型的文件')
      }
      return isRightSize && isAccept
    },
    chartRefresh(){
      setTimeout(()=>{
        this.chart = echarts.init(this.$refs.chart);
        this.chart.clear();
        setTimeout(()=>{
          this.chart.setOption(this.option);
        },300)
```

```
      },200)
    },
    getChartResult(){
      getChart(this.chartId).then(res =>{
        if (res.code == 200 && res.data.state == '成功'){
          this.analysisConclusion = res.data.analysisConclusion;
          //取出配置对象的代码
          let generatedChartData = res.data.generatedChartData;
          //将配置对象的代码转换为JSON对象,不是不是标准的JSON对象,需要使用eval函数进行转换
          let optionJson = eval("(" + generatedChartData + ")")
          //返回格式不稳定,有可能会多封装一层option,需要判断一下
          if (optionJson.option === undefined || optionJson.option === null) {
            this.option = optionJson;
            } else {
              this.option = optionJson.option;
              }
            this.chartRefresh();
            ElMessage({
              message: '生成图表成功',
              type: 'success',
            });
            this.chartId = undefined;
        }else if (res.code == 200){
          ElMessage({
            message: res.data.state,
            type: 'info',
          });
        }else {
          ElMessage({
            message: res.msg,
            type: 'error',
          });
        }
      }).catch(err =>{
        ElMessage({
          message: err,
          type: 'error',
        });
      });
    }
  }
}
</script>

<style scoped>
.el-upload__tip {
  line-height: 1.2;
}

</style>

```

4.3 图表管理



- **图表展示**：前端使用ECharts展示图表，用户可以查看和操作图表。
  - 

- **图表删除**:删除图表

4.4 安全认证

- **JWTAuthenticationTokenFilter**：自定义过滤器，用于验证请求中的JWT。
- **异常处理**：GlobalExceptionHandler处理认证和鉴权过程中的异常。
  - 全局异常处理器定义了两个方法，分别用于处理自定义的 `SystemException` 异常和所有其他类型的 `Exception` 异常。通过这种方式，可以统一异常处理逻辑，确保所有异常都能被恰当地记录和响应，提高应用程序的健壮性和用户体验。
    1. `@RestControllerAdvice`：这个注解表示这个类是一个全局的异常处理控制器，它会对整个应用程序范围内的异常进行捕捉和处理。
    2. `GlobalExceptionHandler` 类：定义了一个全局异常处理器。
    3. `private final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);`：创建一个日志对象，用于记录日志信息。
    4. `@ExceptionHandler(SystemException.class)`：这个注解指定了方法 `systemException` 用于处理 `SystemException` 类型的异常。
    5. `systemException(SystemException e)` 方法：处理 `SystemException` 异常。

        - `log.error("发生错误:{}",e);`：记录错误日志，包括异常堆栈跟踪。
        - `return ResponseResult.errorResult(e.getCode(),e.getMsg());`：返回一个`ResponseResult`对象，包含错误码和错误消息。
    6. `@ExceptionHandler(Exception.class)`：这个注解指定了方法`systemException`用于处理所有未被捕获的`Exception`类型的异常。
    7. `systemException(Exception e)` 方法：处理通用`Exception`异常。
        - `log.error("发生错误:{}",e);`：记录错误日志，包括异常堆栈跟踪。
        - `return ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.SYSTEM_ERROR.getCode(),e.getMessage());`：返回一个`ResponseResult`对象，包含系统错误码和异常消息。

```java
package space.anyi.BI.handler.exception;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import space.anyi.BI.entity.ResponseResult;
import space.anyi.BI.exception.SystemException;

/**
 * @ProjectName: BI
 * @FileName: GlobalExceptionHandler
 * @Author: 杨逸
 * @Data:2024/11/28 20:25
 * @Description:
 */
@RestControllerAdvice
public class GlobalExceptionHandler{
    private final Logger log = LoggerFactory.getLogger(GlobalExceptionHandler.class);
    @ExceptionHandler(SystemException.class)
    public ResponseResult systemException(SystemException e){
        log.error("发生错误:{}",e);
        return ResponseResult.errorResult(e.getCode(),e.getMsg());
    }
    @ExceptionHandler(Exception.class)
    public ResponseResult systemException(Exception e){
        log.error("发生错误:{}",e);
        return ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.SYSTEM_ERROR.getCode(),e.getMessage());
    }
}

```

- **认证异常处理**:
    - 这个类定义了认证失败后的处理逻辑，根据不同的认证异常返回不同的错误信息，并以JSON格式响应给客户端

1. `AuthenticationEntryPointImpl` 类：实现了`AuthenticationEntryPoint`接口，用于自定义处理认证失败后的行为。
2. `commence` 方法：当Spring Security认证失败时，这个方法会被调用。它接收`HttpServletRequest`和`HttpServletResponse`对象，以及一个`AuthenticationException`异常对象。
3. 异常处理：
   - `InsufficientAuthenticationException`：当认证信息不足时抛出的异常，例如，用户未提供认证信息。针对这种异常，返回一个特定的错误码`NO_OPERATOR_AUTH`。
   - `InternalAuthenticationServiceException`：当认证过程中发生内部错误时抛出的异常，例如，密码错误。针对这种异常，返回一个错误码`LOGIN_ERROR`。
   - 其他`AuthenticationException`：对于其他类型的认证异常，返回系统错误码`SYSTEM_ERROR`，并附带异常消息。
4. 响应处理：
   - 使用`WebUtils.renderString`方法将错误信息以JSON格式返回给客户端。这里使用`JSON.toJSONString(responseResult)`将`ResponseResult`对象转换为JSON字符串。
5. 异常堆栈跟踪打印：
   - `e.printStackTrace();`：打印异常堆栈跟踪.

```java
package space.anyi.BI.handler.security;

import com.alibaba.fastjson.JSON;
import org.springframework.security.authentication.InsufficientAuthenticationException;
import org.springframework.security.authentication.InternalAuthenticationServiceException;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;
import space.anyi.BI.entity.ResponseResult;
import space.anyi.BI.util.WebUtils;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * @ProjectName: BI
 * @FileName: AuthenticationEntryPointImpl
 * @Author: 杨逸
 * @Data:2024/11/28 20:18
 * @Description:
 */
@Component
public class AuthenticationEntryPointImpl implements AuthenticationEntryPoint {
    @Override
```

```
    public void commence(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AuthenticationException e) throws IOException, ServletException {
        e.printStackTrace();
        ResponseResult responseResult = null;
        //针对不同异常,返回对应的信息
        if (e instanceof InsufficientAuthenticationException) {
            responseResult = ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.NO_OPERATOR_AUTH);
        } else if (e instanceof InternalAuthenticationServiceException) {
            responseResult = ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.LOGIN_ERROR);
        } else {
            responseResult = ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.SYSTEM_ERROR.getCode(),e.getMessage());
        }
        WebUtils.renderString(httpServletResponse, JSON.toJSONString(responseResult));
    }
}

```

- **鉴权异常处理**:
  - 这个类定义了用户访问被拒绝后的处理逻辑，返回一个错误信息给前端，并以JSON格式响应
    1. `AccessDeniedHandlerImpl` 类：实现了 `AccessDeniedHandler` 接口，用于自定义处理用户访问被拒绝后的行为。
    2. `handle` 方法：当用户的访问被拒绝时，这个方法会被调用。它接收 `HttpServletRequest` 和 `HttpServletResponse` 对象，以及一个 `AccessDeniedException` 异常对象。
    3. 异常处理：
       - `e.printStackTrace();`：打印异常堆栈跟踪，这通常用于调试目的。
    4. 响应处理：
       - 创建一个 `ResponseResult` 对象，使用 `ResponseResult.errorResult` 方法，并传入错误码 `NO_OPERATOR_AUTH`，表示用户没有操作权限。
       - 使用 `WebUtils.renderString` 方法将 `ResponseResult` 对象转换为JSON字符串，并写入到 `HttpServletResponse` 中，作为HTTP响应返回给客户端。
    5. 异常信息：
       - `AccessDeniedException`：当用户尝试访问他们没有权限的资源时，Spring Security会抛出这个异常。
  -
```
package space.anyi.BI.handler.security;

import com.alibaba.fastjson.JSON;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.stereotype.Component;
```

```java
import space.anyi.BI.entity.ResponseResult;
import space.anyi.BI.util.WebUtils;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * @ProjectName: BI
 * @FileName: AccessDeniedHandlerImpl
 * @Author: 杨逸
 * @Data:2024/11/28 20:17
 * @Description:
 */
@Component
public class AccessDeniedHandlerImpl implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, AccessDeniedException e) throws IOException, ServletException {
        e.printStackTrace();
        ResponseResult responseResult = ResponseResult.errorResult(ResponseResult.AppHttpCodeEnum.NO_OPERATOR_AUTH);
        WebUtils.renderString(httpServletResponse, JSON.toJSONString(responseResult));
    }
}

```