

昌吉学院  
本科毕业论文（设计）

论文（设计）分类号：  
密 级：



昌吉学院

CHANGJI UNIVERSITY

## 基于JAVA和Flutter的校园社交APP

二级学院 信息工程学院  
学科门类 工 学  
专业 计算机科学与技术  
班级 信息B1902  
学号 1945829064  
姓名 姜刚刚  
指导教师 孟建峰  
教师职称 未评级

二〇二三年五月十日

## **毕业论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果或作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：

年   月   日

## **毕业论文版权使用授权书**

本毕业论文作者完全了解学院有关保存、使用毕业论文的规定，同意学院保留并向有关毕业论文管理部门或机构递交论文的复印件和电子版，允许论文被查阅和借阅。本人授权本学院及以上级别优秀毕业论文评选机构将本毕业论文的全部或部分内容编入有关数据库以资检索，可以采用复印、缩印或扫描等复制手段保存和汇编本毕业论文。

声明人签名：

导师签名：

年   月   日

年   月   日

## 摘要

二十世纪初，随着移动互联网的发展，人们生活中越来越依赖手机软件，而其中的重要代表无疑就是社交软件了。因为在社交软件上只要有网络，即便在天涯海角也能变成近在咫尺。而说起社交软件，大家基本都了熟于耳，其中最常用的就是微信和QQ。虽然现在社交App众多，但在细分领域，比如校园，却没有一款App能有效的提升同学之间的沟通效率，这主要是因为校园相较于社会社交模式更加复杂，使得App研发成本及使用成本过高，所以如何开发一款提示同学之间、学生与老师直接的沟通效率的App是本论文所研究的重点。

本项目将基于java和flutter开发一款校园定制的社交app，后端将采用当前主流的SpringCloud Alibaba微服务架构进行开发，所使用的阿里巴巴微服务组件主要有Feign、Nacos、Gateway、Hystrix等。前端将采用Flutter框架开发，因为Flutter是一个跨平台的架构，可以将项目打包成 android平台app，也可以将项目打包成ios平台的app，降低项目的开发难度。

后端服务部署将使用docker进行部署，微服务可以很好的与docker进行结合，每一个微服务都进行容器化编排，然后使用docker-compose进行统一部署，docker还可以更好的对服务进行监控以及管理。

通过以上技术对项目进行开发，将极大的降低项目的开发难度，提高项目的开发效率，可以更好对项目实施开发。

**关键词：** SpringCloud； Flutter； Docker

## Campus design APP based on JAVA and Flutter

### Abstract

At the beginning of the twentieth century, with the development of the mobile Internet, people rely more and more on mobile phone software in their lives, and an important representative of it is undoubtedly social software. Because as long as there is an Internet on social software, even in the ends of the earth, it can become close at hand. When it comes to social software, everyone is basically familiar with it, and the most commonly used are WeChat and QQ. Although there are many social apps now, in the subdivision field, such as campus, there is no App that can effectively improve the communication efficiency between students, mainly because the campus is more complex than the social social model, making the cost of App development and use too high, so how to develop an App that prompts the efficiency of direct communication between students, students and teachers is the focus of this paper.

This project will develop a campus customized social app based on Java and Flutter, and the backend will be developed using the current mainstream SpringCloud Alibaba microservice architecture, and the Alibaba microservice components used mainly include Feign, Nacos, Gateway, Hystrix, etc. The front-end will be developed using the Flutter framework, because Flutter is a cross-platform architecture, which can package the project into an Android platform app, or package the project into an iOS platform app, reducing the development difficulty of the project.

Back-end service deployment will use docker for deployment, microservices can be well combined with docker, each microservice is containerized orchestration, and then use docker-composite for unified deployment, docker can also better monitor and manage services.

The development of the project through the above technology will greatly reduce the difficulty of project development, improve the development efficiency of the project, and better implement the development of the project

**Key Words:** SpringCloud; Flutter; Docker

## 目 录

摘要 .....	I
Abstract .....	II
第一章 绪论 .....	1
1.1 开发背景 .....	1
1.2 国内外研究现状 .....	1
1.3 研究内容 .....	2
第二章 前期准备及系统分析 .....	3
2.1 前期准备 .....	3
2.2 系统需求分析 .....	3
2.3 系统开发分析 .....	3
第三章 系统设计 .....	5
3.1 系统模块设计 .....	5
3.2 数据库设计 .....	7
第四章 系统功能实现 .....	16
4.1 系统开发环境配置 .....	16
4.2 项目结构搭建 .....	16
4.3 Nacos 及项目配置bootstrap加载方法实现 .....	18
4.4 系统异常处理机制 .....	20
4.5 Hystrix 服务熔断与降级机制 .....	21
4.6 Feign 服务远程调用 .....	23
4.7 用户登录 .....	24
4.8 用户权限校验 .....	27
4.9 服务网关及配置 .....	30
4.10 查询学生课表 .....	32
4.11 朋友圈功能实现 .....	34
4.12 文件上传下载功能实现 .....	38
4.13 即时聊天 .....	40
第五章 前端界面设计 .....	43
5.1 登录界面 .....	44
5.2 首页界面 .....	44
5.3 好友界面 .....	45
5.4 校园圈界面 .....	45
第六章 服务部署 .....	47

6.1 Dockerfile .....	47
6.2 DockerCompose .....	48
6.3 部署脚本 .....	49
第七章 接口测试 .....	52
7.1 测试案例 .....	52
7.2 测试结果汇总 .....	53
结    论 .....	55
参考文献 .....	56
致    谢 .....	57
注    解 .....	58

## 第一章 绪论

### 1.1 开发背景

近年来，社交软件已经成为人们日常生活中必不可少的一部分，社交app有很多优点，例如可以让我们即使距离很远也可以随时交流，可以让我们打破地理位置的限制，让我们交到更多的朋友，可以在网上发表自己的意见等等，但是任何事物都有其反面，社交app也不可能避免的有着一些无法避免的缺点，例如因为交流成本的下降以及网络匿名的特质，导致电信诈骗横行，尤其是对即将进入社会的大学生来说，阅历的不足导致更加容易受骗、而且更容易被网络攻击，进而造成心理疾病等等。

通过对以上问题的分析以及总结，造成这种问题的原因很大一部分是因为网络匿名的特质，造成网络监管难，进而使电信诈骗以及恶意攻击，传播虚假信息的成本变动极低。所以要解决这种问题要首先实现社交平台app的实名制，像微信及qq这类社交平台，虽然实现了实名制，但是由于其面向大众的原因，以及要保护用户隐私的造成实质上添加好友后仍然无法判断对方用户身份，实质上与匿名没有什么区别，且暂时没有更好的办法去解决。

所以本app不同于微信和qq，最核心的功能是指在保护在校大学生网上信息交流的安全，所以不是面向所有大众，而是只面向特定用户（大学生），用户信息与学籍绑定，只有在校大学生才可以注册使用，这样就避免了用户匿名问题，以及使网络监管变得容易，提升网络诈骗的成本。

所以本APP是一款面向在校大学生，旨在保护用户信息安全，降低网络诈骗、网络攻击、传播虚假信息的概率的社交app。

### 1.2 国内外研究现状

#### (1) 国外研究现状：

通过数据调研，国外在校大学生目前使用的社交app如下：

①Facebook：Facebook的主要目标是让人们能够与家人、朋友和同事保持联系，分享生活中的各种内容，包括照片、视频、状态更新和链接。

②Twitter：Twitter的主要特点主要有实时性和公开性。用户可以即时发布推文，将信息迅速传播给他们的关注者。通过关注其他用户，用户可以接收其发布的推文，并可以通过回复、转发和点赞等方式与他们进行互动。

③Instagram：Instagram用户可以拍摄或选择照片和视频，然后应用各种滤镜和编辑工具进行美化和修改。他们可以在自己的个人资料页面上发布这些内容，并通过关注其他用户、点赞、评论和私信等方式与他人互动。

(2) 国内研究现状:

通过数据调研，国内在校大学生目前使用的社交app如下：

①微信：微信提供了多种功能，例如即时消息传递、语音和视频通话、朋友圈动态分享、公众号订阅、支付和转账等。用户可以通过手机号码注册微信账号，并添加其他用户为好友。

②QQ：QQ提供了多种通讯方式，包括文本消息、表情符号、图片、语音消息和视频通话。用户可以创建群组聊天，与多个好友进行群聊，并进行语音或视频会议。

③微博：微博用户可以关注其他用户，以接收其发布的微博，并可以通过评论、转发和点赞等方式与他们进行互动。微博还提供了话题标签功能，用户可以使用特定的标签将自己的微博与相关话题关联起来，并浏览其他人关于同一话题的微博。

(3) 总结目前的现状

经过资料发现，国内外都有着成熟可靠的社交app，而且拥有着大量的平台用户，但是基本没有针对校园使用情况的优化开发，基本都是面向全面的社交软件。综上所述，关于校园社交app的研究意义重大，能更好的针对学生的特殊情况做针对性的开发，从而起到保护学生的作用。

### 1.3 研究内容

基于上述的研究背景，社交软件app的研究内容可以涵盖以下几个方面：

(1) 社交软件app的用户身份认证研究：探讨用户在社交软件app上的身份信息，如何认证用户身份是否为在校大学生，防止无关人员注册使用。

(2) 基于学生身份的特点功能开发：研究基于用户学生身份的社交app特殊功能的开发，例如学生课程表、校园圈等。

(3) 用户好友系统的设计及开发：研究用户好友系统架构的设计及开发，包括添加好友，搜索好友等

(4) 社交软件app的技术架构和创新：研究社交软件app的技术特点和架构设计，探讨如何使用当前成熟的软件技术开发高性能的系统。

## 第二章 前期准备及系统分析

### 2.1 前期准备

#### (1) 相关开发工具:

代码编辑器: Visual Studio Code、IntelliJ IDEA、Android Studio  
Linux服务器连接工具: SecureCRT、Transmit、Iterm  
数据库管理工具: Navicat、Medis  
Api接口管理工具: PostMan  
抓包工具: Wireshark  
代码版本管理工具: git、gitKraken

#### (2) 相关软件安装:

Linux服务器: Centos7  
数据库: Mysql 8.3、Redis  
微服务容器管理: Docker  
反向代理服务器: Nginx  
微服务注册中心: Nacos  
Java包管理工具: Maven

### 2.2 系统需求分析

本系统相关功能分析如下:

- (1) 用户管理: 包括用户身份认证, 用户信息获取等。
- (2) 内容管理: 包括文图片、朋友圈等内容的上传、存储和管理。
- (3) 学生数据同步: 同步学生数据包括课本, 班级等。
- (3) 社交功能: 包括好友关系管理、私信互动、评论回复等。
- (4) 推荐算法: 根据用户行为数据、内容数据等, 提供内容推荐等功能。
- (5) 消息提醒: 包括新消息通知、好友请求通知等。
- (6) 性能优化: 保证 app 在大量用户同时在线时仍然保持流畅的操作体验

### 2.3 系统开发分析

本系统采用当前主流的SpringCloud微服务开发框架, 可以达到解耦、灵活性、可扩展性、可维护性等目标。微服务架构可以说是如何将功能分解成一系列服务的一种架构模式<sup>[1]</sup>。

服务开发主要从以下几个方面:

- (1) 服务设计: 需要考虑如何划分服务、如何定义接口、如何选择数据存储等方面的问题, 本系统数据库采用Mysql+Redis组合方式, “Redis集群+MySQL集群+MyCAT分库分表组件”的数据库层面应对高并发请求的解决方案<sup>[2]</sup>。
- (2) 服务开发: 需要选择合适的编程语言、框架和工具, 编写服务代码, 实现服务的基本功能。
- (3) 服务优化: 使用算法与数据结构进行代码优化, 系统开发对关键数据进行数据结构与算法优化, 数据结构不只是用于组织数据, 它还极大地影响着代码的运行速度。因为数据结构不同, 程序的运行速度可能相差多个数量级<sup>[3]</sup>。
- (4) 服务部署: 需要使用容器化技术如Docker来进行部署, 编写dockerfile文件, 使用docker-compose进行统一部署。
- (5) 服务测试: 需要进行单元测试、接口测试, 确保服务接口的可靠性。

## 第三章 系统设计

### 3.1 系统模块设计

该系统采用SpringCloud Alibaba组件开发，包括远程调用组件Feign、注册中心Nacos、服务熔断与降级Hystrix、网关Gateway等。基于SpringBoot开发，Spring Boot是Spring技术的集大成者，它带来全新的自动化配置解决方案，因此一经出现就受到了极大的关注和好评，成为Java领域的焦点之一<sup>[4]</sup>。

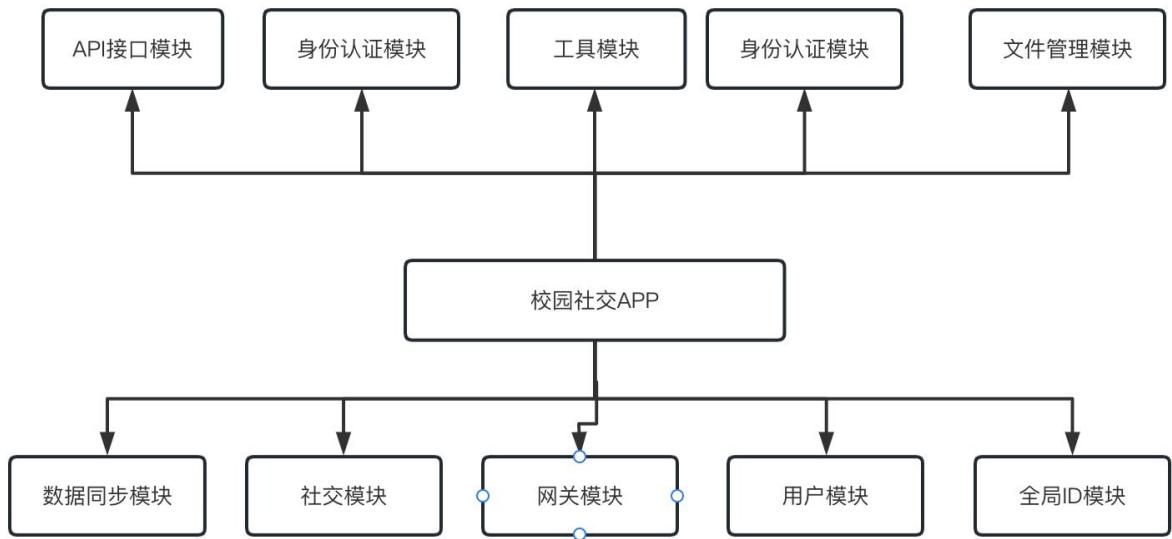


图3.1 系统设计

(1) api模块(api):

负责feign微服务远程调用的api接口统一开发，统一管理，方便工程api维护及开发，防止开发混乱。

(2) 用户认证模块(auth):

实现用户登录时身份认证，与教务系统进行对接认证，登录成功后返回用户token，并对token的超时进行刷新，以及用户退出登录等功能。

(3) 工具模块 (common) :

负责实现开发过程中所需要的工具类，所有模块继承工具模块，属于顶层模块，包括以下几个方面：

①核心工具 (core) : 开发过程中所需的各种常数枚举、自定义异常类例如 NotLoginException(未登录异常), NotRoleException(无权限异常)等、全局ThreadLocal，存储当前线程变量中的 用户id、用户名称、Token等信息、全局统一返回类 ServerResponseEntity封装三个返回字段，code（返回状态码），message（返回信息），

data（返回数据）、以及常用工具类，如UUID工具类，dataUtils（日期工具类），JwtUtils，SpringUtils（spring工具类方便在非spring管理环境中获取bean）等。

②日志工具（log）：自定义接口访问日志注解@Log，使用Spring Aop拦截注解方法，添加方法访问前后日志，日志字段有操作模块（title）、请求方法（method）、访问人员学号（account）、请求url（operationUrl）、错误信息（errMsg），错误信息只在接口报错时有效，否则为null，Logservice接口定义日志操作方法，有打印日志接口，存储日志接口等。

③Redis工具(redis)：配置Redis序列化工具FastJson2JsonRedisSerializer使用StringRedisSerializer来序列化和反序列化redis的key值，Hash的key也采用StringRedisSerializer的序列化方式，自定义redis服务类，开发缓存基本的对象，Integer、String、实体类、获得缓存的基本对象、获得缓存的基本对象列表等方法。

④数据库工具（database）：配置orm层框架MyBatis，标注mapper层位置，方便后续开发，MyBatis是一款在持久层使用的SQL映射框架，可以将SQL语句单独写在XML配置文件中，或者使用带有注解的Mapper映射类来完成数据库记录到Java实体的映射<sup>[5]</sup>。

#### （4）网关模块（gateway）：

配置swagger统一访问接口，配置系统各模块统一访问接口，用于管理和转发微服务之间的请求，它可以理解为微服务的入口，所有的请求都需要通过网关进行路由和处理。

微服务网关的主要作用包括：

①路由：根据请求的url地址，将请求转发到对应的服务器上。

②负载均衡：根据负载均衡策略将请求转发到不同的微服务实例上，提高系统的可靠性。

③限流熔断：防止流量过载导致系统崩溃，提供限流、熔断等机制，确保系统稳定运行。

#### （5）全局id模块（leaf）：

leaf是一款分布式ID生成器，它可以为分布式系统生成全局唯一ID。

Leaf的主要特点包括：

①可定制：用户可以根据自己的需求定制ID生成规则，包括ID位数、时间戳位数、节点号位数等。

②高性能：Leaf支持多种ID生成模式，生成全局唯一id。

③高可用：Leaf采用了分布式部署架构，具有高可用性和容错性，并支持故障自动切换和自动恢复。

④开源免费：Leaf是美团点评开源的项目，用户可以免费使用和修改，也可以参与项目开发和维护。

(6) 用户管理模块 (user) :

进行用户管理功能，包括用户信息查询、保存用户信息、通过学号查询用户、通过用户id查询用户等。

(7) 官网模块 (website) :

实现验证用户账号密码同教务管理系统是否相同、查询教务系统个人课表、查询其他班教务系统课表等。

(8) 社交模块 (social) :

开发用户交互核心功能，包括发送，保存，删除，获取，点赞，评论朋友圈功能、实习聊天功能等。

(9) 文件模块 (file) :

底层采用分布式对象存储服务软件Minio,minio具有一下特点，

①高可用性：MinIO 支持数据复制和多副本机制，可以在节点故障时自动切换，保证数据的可靠性和可用性。

②高性能：MinIO 基于 Go 语言编写，具有高性能和低资源消耗的特点，可以满足高并发和大规模数据存储的需求。

## 3.2 数据库设计

因为系统采用微服务的开发框架，在微服务架构中，数据库也需要进行微服务化的设计，以适应微服务架构的特点和要求。传统的单体应用程序往往使用单一的数据库，而在微服务架构中，每个微服务都有自己的数据存储，数据存储可以分散在不同的数据源中，这样可以避免数据的紧耦合和数据交叉污染。

数据库采用Mysql，MySQL的架构特点使其可以被应用在很多场景中。尽管它并不完美，但足够灵活，从小型的个人网站到大型的企业应用它都可以工作得很好。为了最大限度地使用MySQL，你需要了解它的设计，以便能够用其所长，避其所短<sup>[6]</sup>。

(1) 各模块E-R图表如下：

①auth模块:包括五个数据表,auth\_account(统一账户信息表)、sys\_auth\_role (用户和角色关联表) 、sys\_menu (菜单表) 、sys\_role (用户角色表) 、sys\_role\_menu (角色和菜单关联表) :

## 基于 JAVA 和 Flutter 的校园社交 APP

---



图4.1 auth模块

②全局id管理模块，包括一个id管理数据表leaf\_alloc:



图4.2 id生成模块

③社交模块，包括一下三个数据表， friend\_circle\_comment (朋友圈评论表) 、 friend\_circle\_like (朋友圈点赞表) 、 friend\_circle\_message (朋友圈表) :

<b>chat_messages</b>	<b>friend_circle_comment</b>	<b>friend_circle_message</b>
<b>id</b> 极大整数 from_user_account 变长字符串 to_user_account 极大整数 content 变长字符串 state 小整数 create_time 日期时间 update_time 日期时间	<b>comment_id</b> 极大整数 friend_circle_id 极大整数 account 变长字符串 content 变长字符串 like_count 大整数 root_comment_id 极大整数 to_comment_id 极大整数 create_time 日期时间 update_time 日期时间	<b>id</b> 极大整数 user_id 极大整数 account 变长字符串 content 变长字符串 picture 变长字符串 liked_count 大整数 is_delete 小整数 create_time 日期时间 update_time 日期时间
<b>user_relationship</b>	<b>friend_circle_like</b>	
<b>id</b> 极大整数 user_account 变长字符串 friend_account 极大整数 rel_state 小整数 alias_user 变长字符串 alias_friend 变长字符串 create_time 日期时间 update_time 日期时间	<b>id</b> 极大整数 friend_circle_id 极大整数 user_id 极大整数 status 小整数 create_time 时间戳 update_time 时间戳	

图4.3 社交模块

④ 用户管理模块，包括一下一张表users（用户表）：

<b>users</b>
<b>user_id</b> 极大整数 account 变长字符串 institute 变长字符串 specialty 变长字符串 classes 变长字符串 real_name 变长字符串 nick_name 变长字符串 picture 变长字符串 create_time 日期时间 update_time 日期时间 status 小整数

图4.4 用户模块

⑤ 官网对接模块，包括一下两张数据表，courses（课程表）、institute\_info(系院

表):

courses	
id	极大整数
classes	变长字符串
semester	变长字符串
weekly	变长字符串
institute	变长字符串
specialty	变长字符串
course_info	长文本
status	大整数
create_time	日期时间
update_time	日期时间

institute_info	
id	变长字符串
institute	变长字符串

图4.5 社交模块

⑥文件管理模块，包括一下一张表，sys\_file\_info（文件信息表）：

sys_file_info	
id	极大整数
data_type	小整数
file_name	变长字符串
ext	变长字符串
size	极大整数
file_md5	变长字符串
path	变长字符串
parent_dir_id	极大整数
create_time	日期时间
update_time	日期时间
status	小整数

图4.6 文件模块

(2) 数据库及数据表设计:

数据库: cjxycld\_auth

表4.1 auth\_account

名称	数据类型	长度	允许空值	主键	说明
uid	bigint	20	N	Y	
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	更新时间
account	varchar	30	N	N	学号
password	varchar	64	N	N	密码
create_ip	varchar	15	N	N	创建ip
status	tinyint	4	N	N	状态
user_id	bigint	20	N	N	id

表4.2 sys\_role

名称	数据类型	长度	允许空值	主键	说明
role_id	bigint	20	N	Y	id
status	tinyint	4	N	N	状态
role_name	varchar	30	N	N	名称
role_key	varchar	30	N	N	角色权限字符串
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	更新时间

表4.3 sys\_role\_menu

名称	数据类型	长度	允许空值	主键	说明
role_id	bigint	20	N	Y	角色ID
menu_id	bigint	20	N	Y	菜单ID

数据库: cjxyccloud\_leaf

表4.4 leaf\_alloc

名称	数据类型	长度	允许空值	主键	说明
biz_tag	varchar	128	N	Y	区分业务
max_id	bigint	20	N	N	最大值
step	int	10	N	N	号段长度
update_time	timestamp		N	N	更新时间
description	varchar	256	Y	N	描述
random_step	int	10	N	N	每次getid时随机增加的

数据库: cjxyccloud\_social

表4.5 friend\_circle\_comment

名称	数据类型	长度	允许空值	主键	说明
comment_id	bigint	20	N	Y	主键
friend_circle_id	bigint	20	Y	N	朋友圈id
user_id	bigint	20	N	N	用户id
content	varchar	1000	Y	N	评论内容
like_count	int	10	Y	N	点赞数
root_comment_id	bigint	20	Y	N	顶级评论id
to_comment_id	bigint	20	Y	N	回复目标评论id
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	修改时间

昌吉学院 2023 届本科毕业论文 (设计)

表4.6 friend\_circle\_like

名称	数据类型	长度	允许空值	主键	说明
id	bigint	20	N	Y	主键
friend_circle_id	bigint	20	N	N	被点赞的朋友圈id
user_id	bigint	20	N	N	点赞的用户id
status	bit	1	Y	N	点赞状态
create_time	timestamp		N	N	创建时间
update_time	timestamp		N	N	修改时间

表4.7 friend\_circle\_message

名称	数据类型	长度	允许空值	主键	说明
id	bigint	20	N	Y	主键
user_id	bigint	20	N	N	用户id
content	varchar	500	Y	N	
picture	varchar	200	Y	N	图片
location	varbinary	100	Y	N	位置
liked_count	int	10	Y	N	点赞数
is_delete	bit	1	Y	N	是否删除
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	更新时间

数据库: cjxycld\_user

表4.8 user

名称	数据类型	长度	允许空值	主键	说明
user_id	bigint	20	N	Y	ID

## 基于 JAVA 和 Flutter 的校园社交 APP

---

account	varchar	32	N	N	学号
institute	varchar	32	Y	N	院校
specialty	varchar	32	Y	N	专业
classes	varchar	32	Y	N	班级
real_name	varchar	32	Y	N	真实姓名
nick_name	varchar	32	Y	N	用户昵称
picture	varchar	32	Y	N	头像图片路径
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	修改时间
status	tinyint	4	N	N	状态

---

数据库: cjxyccloud\_website

表4.9 courses

---

名称	数据类型	长度	允许空值	主键	说明
id	bigint	20	N	Y	ID
classes	varchar	16	N	N	班级
semester	varchar	16	N	N	学期
weekly	varchar	16	N	N	周次
institute	varchar	32	N	N	院校
specialty	varchar	32	Y	N	专业
course_info	text	65535	N	N	json课程表
status	int	10	N	N	状态
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	修改时间

---

数据库: sys\_minio

表4.10 sys\_file\_info

名称	数据类型	长度	允许空值	主键	说明
id	bigint	20	N	Y	ID
data_type	tinyint	4	N	N	DIR:目录
file_name	varchar	255	N	N	文件名
ext	varchar	32	Y	N	文件后缀
size	bigint	20	Y	N	文件大小
file_md5	varchar	32	Y	N	文件md5
path	varchar	255	N	N	文件路径
parent_dir_id	bigint	20	N	N	父目录
create_time	datetime		N	N	创建时间
update_time	datetime		N	N	修改时间
status	tinyint	4	N	N	状态

## 第四章 系统功能实现

此章将介绍项目系统从配置到项目搭建，再到系统核心功能的实现与开发过程。需要综合考虑用户需求、系统设计、功能实现、测试和维护等多个方面，以构建出一个高效、可靠、易用的系统。

### 4.1 系统开发环境配置

- (1) 配置JDK使用JDK1.8版本， JDK环境变量路径为/Users/mac/Library/Java/corretto-1.8.0\_282-1/Contents/Home。
- (2) 配置Maven包管理工具， 使用版本为3.8.1， maven路径为/usr/local/maven-3.8.1
- (3) 配置Dart， 使用brew install dart安装dart配置dart环境变量路径为 /Users/mazaiting/Library/flutter/bin/cache/dart-sdk/bin
- (4) 配置git，并在github创建git仓库， 拉取到本地， git仓库地址为 <https://github.com/xiaowenhaohub/cjxy-cloud.git>
- (5) 在Centos 7服务器上yum install docker安装docker， 使用dockerfile编写容器， 使用docker-compose一键部署nacos、mysql8、minio

### 4.2 项目结构搭建

系统使用maven进行模块化开发，共分为九个模块，以及项目部署模块(docker)，数据库模块(db)，测试模块等，使用maven创建各系统模块如下：

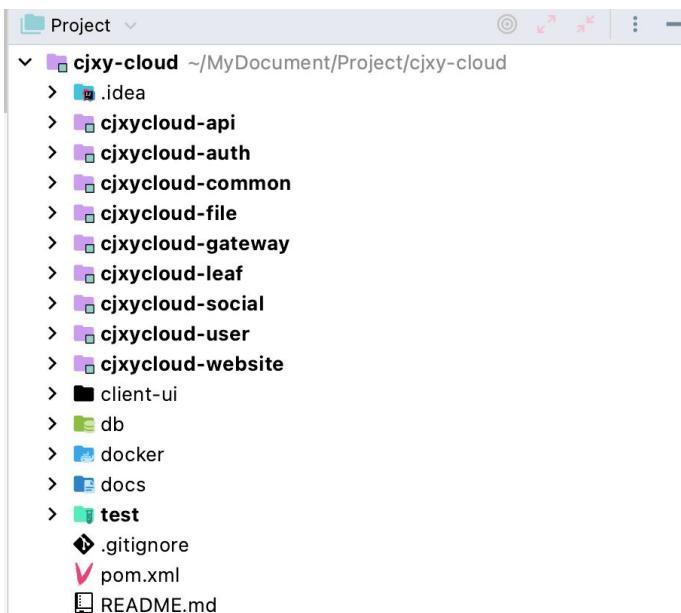


图5.1 项目结构

(1) 系统各模块:

项目各模块命名如下

```
<!-- 模块 -->
<modules>
    <module>cjxyccloud-common</module>
    <module>cjxyccloud-gateway</module>
    <module>cjxyccloud-auth</module>
    <module>test</module>
    <module>cjxyccloud-api</module>
    <module>cjxyccloud-website</module>
    <module>cjxyccloud-leaf</module>
    <module>cjxyccloud-user</module>
    <module>cjxyccloud-file</module>
    <module>cjxyccloud-social</module>
</modules>
```

(2) 系统所需依赖及版本如下:

包含SpringBoot、SpringCloud、Seata、Nacos、Hytrix、Swagger、Log4j、Mybatis等基础依赖版本

```
<!-- 依赖版本管理 -->
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven-compiler-plugin.version>3.8.1</maven-compiler-plugin.version>
    <maven-resources-plugin.version>3.1.0</maven-resources-
plugin.version>
    <java.version>1.8</java.version>
    <!-- 核心依赖 -->
    <spring-boot.version>2.4.2</spring-boot.version>
    <spring-cloud.version>2020.0.2</spring-cloud.version>
    <spring-cloud-alibaba.version>2021.1</spring-cloud-alibaba.version>
    <alibaba.seata.version>1.5.1</alibaba.seata.version>
    <!-- 及时更新 -->
    <seata.version>1.4.2</seata.version>
    <nacos.version>2.0.1</nacos.version>
    <hytrix.version>2.2.6.RELEASE</hytrix.version>
    <swagger.bootstrap.ui>1.9.6</swagger.bootstrap.ui>
    <orika.version>1.5.4</orika.version>
    <jsoup.version>1.13.1</jsoup.version>
    <knife4j.version>3.0.2</knife4j.version>
```

```
<swagger2.version>2.9.2</swagger2.version>
<swagger-bootstrap.version>1.9.6</swagger-bootstrap.version>
<hutool.version>5.5.8</hutool.version>
<mybatis-spring-boot-starter.version>2.1.4</mybatis-spring-boot-
starter.version>
<log4j.version>2.17.1</log4j.version>
<druid.version>1.2.11</druid.version>
<dynamic-ds.version>3.5.1</dynamic-ds.version>
<pagehelper.version>1.4.3</pagehelper.version>
</properties>
```

### 4.3 Nacos及项目配置bootstrap加载方法实现

在传统的spring boot项目开发过程中，需要在resource目录下application.yml配置一些开发环境的参数，例如项目port端口、log4g日志、数据库连接池druid、数据库url、数据库账号密码等参数，在applicatio.yml的参数会在spring项目启动后被加载。

单体架构下，一个项目只有一个application.yml文件，且会在项目创建初就配置完成之后在开发过程中进行小量的修改，无需太大修改，维护压力小，但是在微服务架构中，每一各微服务都会有自己的一个application.yml文件，需要对每一各项目的配置文件进行维护将是一个巨大的开支，所以，在微服务开发过程中，需要对项目的application.yml配置文件进行单独的维护，而nacos注册中心就支持微服务配置的统一管理，所以项目中我们不在需要application.yml而是需要bootstrap.yml配置文件。

Bootstrap.yml是Spring Boot应用程序中的一个配置文件，它用于在应用程序启动时加载配置。与application.yml或application.properties不同，Bootstrap.yml可以在应用程序启动之前加载，因此可以用于加载与应用程序相关的任何配置，例如连接数据库所需的配置、密钥等等。

Bootstrap.yml是在应用程序启动之前加载的，因此它不能依赖于应用程序中的任何Bean或配置。它通常用于加载应用程序中的一些基本配置，例如配置与Spring Cloud Config Server集成的连接、加载安全配置等等。

使用Bootstrap.yml可以将应用程序配置与代码分离，使得应用程序更容易维护和部署。一下是该项目中的统一bootstrap.yml文件配置格式：

```

server:
  port: 4234
spring:
  application:
    name: @artifactId@
  profiles:
    active: @profiles.active@
cloud:
  nacos:
    discovery:
      server-addr: ${@nacos.addr@:120.46.217.24}:${NACOS_PORT:8848}
      #           server-addr: ${NACOS_HOST:aliyun.xiaowenhao}:${NACOS_PORT:8848}
      username: nacos
      password: jganggang@!123
    config:
      server-addr: ${spring.cloud.nacos.discovery.server-addr}
      file-extension: yml
      namespace: @nacos.namespace@
      shared-configs:
        - application-${spring.profiles.active}.${spring.cloud.nacos.config.file-extension}
      username: ${spring.cloud.nacos.discovery.username}
      password: ${spring.cloud.nacos.discovery.password}

```

图5.2 项目配置

该文件配置了微服务项目端口，nacos注册中心地址，账号及密码，nacos配置中心中项目对应的配置文件，采用微服务名加dev表示开发的命名方式。

	Data ID ↗	Group ↗	归属应用: ↗	操作
<input type="checkbox"/>	application-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>
<input type="checkbox"/>	cjxyccloud-gateway-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>
<input type="checkbox"/>	cjxyccloud-auth-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>
<input type="checkbox"/>	cjxyccloud-user-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>
<input type="checkbox"/>	cjxyccloud-leaf-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>
<input type="checkbox"/>	cjxyccloud-website-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>
<input type="checkbox"/>	cjxyccloud-social-dev.yml	DEFAULT_GROUP		<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">编辑</a>   <a href="#">删除</a>   <a href="#">更多</a>

图5.3 nacos配置

## 4.4 系统异常处理机制

系统在运行期间都不可避免的会出现一些异常情况，一些异常是我们可以控制的异常，一些异常是我们无法控制的异常，为了避免异常情况对系统的影响，我们需要对异常进行统一的处理，而spring就提供了这样一种全局的异常处理机制。

Spring框架提供了强大的异常处理机制，可以帮助开发人员更好地处理应用程序中可能出现的异常情况。Spring异常处理机制主要包括以下几个方面：

`@ExceptionHandler`注解：使用这个注解，我们可以为Controller中的方法定义异常处理方法。当控制器方法抛出指定类型的异常时，Spring框架会自动调用与之匹配的异常处理方法。

`@ControllerAdvice`注解：使用这个注解，我们可以定义全局异常处理器，处理应用程序中所有Controller方法抛出的异常。

Spring Boot的ErrorController：Spring Boot提供了一个默认的ErrorController，用于处理应用程序中发生的所有异常。我们可以自定义ErrorController，以提供自定义的异常处理机制。

Spring Boot的Whitelabel Error Page：当应用程序中发生未处理的异常时，Spring Boot会显示默认的Whitelabel Error Page。我们可以自定义Whitelabel Error Page，以提供自定义的异常处理机制。

开发全局异常处理器如下：

```

/*
@RestControllerAdvice
@Slf4j
public class GlobalExceptionHandler {

    /**
     * token异常
     * @param e
     * @param <T>
     * @return
     */
    @ExceptionHandler(NotLoginException.class)
    public <T> ServerResponseEntity<T> handleNotLoginException(NotLoginException e) {
        String message = e.getMessage();
        log.error(message);
        return ServerResponseEntity.showFailMsg(message);
    }

    /**
     * 处理参数校验异常 --Json 转换异常
     * @param req
     * @param e
     * @return
     */
    @ExceptionHandler(value = HttpMessageNotReadableException.class)
    public <T> ServerResponseEntity<T> exceptionHandler(HttpServletRequest req, HttpMessageNotReadableException e) {
        log.error("参数校验异常--json转换异常：" , e);
        return ServerResponseEntity.showFailMsg("参数校验异常--json转换异常：" + req.getRequestURI());
    }
}

```

图5.4 异常处理

自定义异常如下：

包括未登录异常、无权限异常、基础异常、服务异常、参数异常等，该类都继承于。

RuntimeException运行时异常。

当系统抛出我们的自定义异常时，全局异常处理器就会捕捉该异常，获取异常信息，然后将异常包装成统一返回类ServerResponseEntity进行返回。如下所示：

```
/*
 * 请求方式不支持
 */
@ExceptionHandler(HttpServletRequestMethodNotSupportedException.class)
public <T> ServerResponseEntity<T> handleHttpRequestMethodNotSupported(HttpServletRequestMethodNotSupportedException e, HttpServletRequest request) {
    String requestURI = request.getRequestURI();
    String message = e.getMessage();
    log.error("请求地址'{}',不支持'{}'请求", requestURI, message);
    return ServerResponseEntity.showFailMsg(message);
}
```

图5.5 异常处理

## 4.5 Hystrix服务熔断与降级机制

在微服务架构中，一个应用往往由多个服务组成，这些服务之间相互依赖，依赖关系错综复杂。

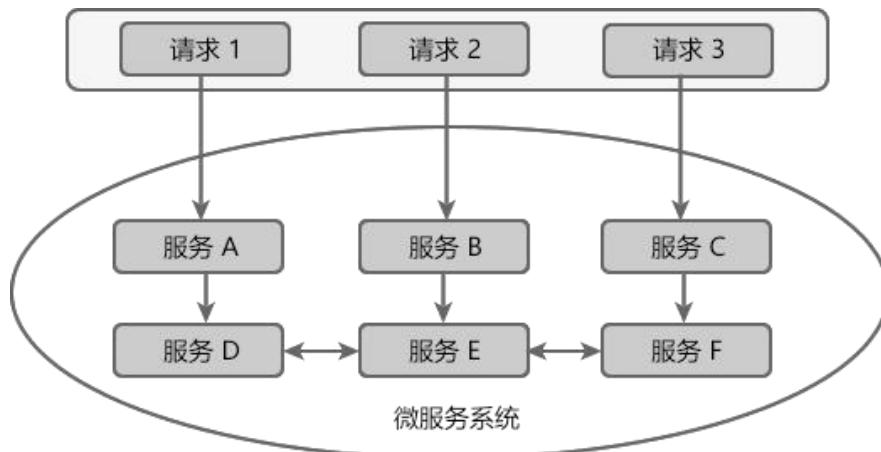


图5.6 微服务架构

通常情况下，一个用户请求往往需要多个服务配合才能完成。如图 1 所示，在所有服务都处于可用状态时，请求 1 需要调用 A、D、E、F 四个服务才能完成，请求 2 需要调用 B、E、D 三个服务才能完成，请求 3 需要调用服务 C、F、E、D 四个服务才能完成。

当服务 E 发生故障或网络延迟时，会出现以下情况：

即使其他所有服务都可用，由于服务 E 的不可用，那么用户请求 1、2、3 都会处于阻塞状态，等待服务 E 的响应。在高并发的场景下，会导致整个服务器的线程资源在短时间内迅速消耗殆尽。

所有依赖于服务 E 的其他服务，例如服务 B、D 以及 F 也都会处于线程阻塞状态，等待服务 E 的响应，导致这些服务的不可用。

所有依赖服务 B、D 和 F 的服务，例如服务 A 和服务 C 也会处于线程阻塞状态，以等待服务 D 和服务 F 的响应，导致服务 A 和服务 C 也不可用。

从以上过程可以看出，当微服务系统的一个服务出现故障时，故障会沿着服务的调用链路在系统中疯狂蔓延，最终导致整个微服务系统的瘫痪，这就是“雪崩效应”。为了防止此类事件的发生，微服务架构引入了“熔断器”的一系列服务容错和保护机制。

系统熔断机制如下所示：

```

@Component
@Slf4j
public class WebsiteFeignClientFallback implements FallbackFactory<WebsiteFeignClient> {

    @Override
    public WebsiteFeignClient create(Throwable throwable) {
        log.error("官网服务调用失败:{}" , throwable.getMessage());

        return new WebsiteFeignClient() {
            @Override
            public ServerResponseEntity<AuthAccountVO> getByAccountAndPassword(String account, String password) {
                return ServerResponseEntity.showFailMsg("查询用户失败:" + throwable.getMessage());
            }

            @Override
            public ServerResponseEntity<Object> testFallback(String username) {
                return ServerResponseEntity.showFailMsg("测试失败:" + throwable.getMessage());
            }
        };
    }
}

```

图5.7 失败处理

当服务调用失败或超时时就会触发该熔断机制，返回调用失败实例，以避免系统的雪崩效应，可以降低故障服务对其他服务和系统的影响，从而提高系统的可用性和稳定性。

## 4.6 Feign服务远程调用

在微服务框架中，各服务直接需要相互调用来完成某个业务需求，而在服务调用的过程中有许多种网络传输协议如rpc, http等，本系统采用http协议进行服务远程调用，使用spring cloud远程调用组件feign来开发远程调用接口。

使用Feign可以避免自己手动开发HTTP请求的代码，它会在运行时自动生成http调用，让服务间通信变得更加简单和直观。

在开发过程中，为了降低远程调用接口的维护难度，为远程调用接口独立开发一个api模块，包命名结构如下：

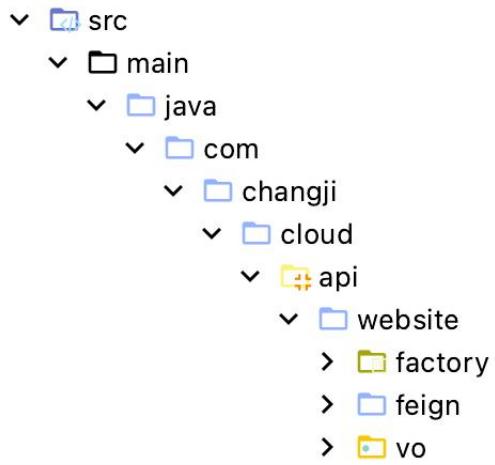


图5.8 包结构

并在feign包定义远程调用的接口规范：

```

    /**
     * @ Author      : 小问号。
     * @ Date        : Created in 16:10 2022/8/20
     * @ Modified By:
     */

    @FeignClient(value = "cjxycloud-website", contextId = "websiteService", fallbackFactory = WebsiteFeignClientFallback.class)
    public interface WebsiteFeignClient {

        /**
         * 根据用户名和密码从教务管理系统验证用户，验证失败返回null
         * @return
         */
        @PostMapping(value = "/feign/website/getByUserNameAndPassword")
        ServerResponseEntity<AuthAccountVO> getByAccountAndPassword(@RequestParam("account") String account, @RequestParam("password") String password);

        @PostMapping(value = "/feign/website/test")
        ServerResponseEntity<Object> testFallback(@RequestParam("username") String username);
    }
  
```

图5.9 接口规范

通过@PostMapping定义远程调用的url路径，并通过@RequestParam规定该接口所需要传入的参数，以及通过ServerResponseEntity<AuthAccountVO>的形式定义该接口返回数据类型。

## 4.7 用户登录

用户登录系统是系统开发当中必备的，旨在确保只有授权用户可以访问系统中的敏感信息和功能。

本系统具有用户登录功能，但不具有用户注册功能，所有的用户都将使用教务系统相同，通过调用教务系统登录功能来验证用户账号密码。

(1) 登录接口所需参数: AuthenticationDTO

```

@NotBlank(message = "学号不能为空")
@ApiModelProperty(value = "学号", required = true)
private String account;

@NotBlank(message = "密码不能为空")
@ApiModelProperty(value = "密码", required = true)
private String password;

public String getAccount() { return account; }

public void setAccount(String account) { this.account = account; }

public String getPassword() { return password; }

public void setPassword(String password) { this.password = password; }

@Override
public String toString() {
    return "AuthenticationDTO{" +
        "account='" + account + '\'' +
        ", password='" + password + '\'' +
        '}';
}

```

图5.10 用户封装

登录参数需要account学号，password密码，且通过@NotBlank注解规定参数不能为空，否则抛出参数异常。

(2) 登录接口实现:

```

//从数据库查询用户
AuthAccount authAccount = authAccountMapper.queryByAccount(account);
if (authAccount != null && SecurityUtils.matchesPassword(password, authAccount.getPassword())) {
    //查询用户权限列表
    List<String> permission = authAccountMapper.queryUserMenuByUid(authAccount.getUid());
    return toLoginUser(authAccount, permission);
}
//从教务管理系统查询用户 远程调用website模块
ServerResponseEntity<AuthAccountVO> byUserNameAndPassword = websiteFeignClient.getByAccountAndPassword(account, password);
if (!byUserNameAndPassword.isSuccess()) {
    throw new ServiceException(byUserNameAndPassword.getMsg());
}
AuthAccountVO authAccountVO = byUserNameAndPassword.getData();
authAccount = mapperFacade.map(authAccountVO, AuthAccount.class);
authAccount.setStatus(1);
//更新密码
if (authAccount.getUid() != null) {
    authAccountMapper.edit(authAccount);
    //获取角色权限
    List<String> permission = authAccountMapper.queryUserMenuByUid(authAccount.getUid());
    return toLoginUser(authAccount, permission);
}
// 从leaf获取uid
ServerResponseEntity<Long> uidResponseEntity = segmentFeignClient.getSegmentId(LeafKeyEnum.AUTH_UID_KEY.value());
ServerResponseEntity<Long> userIdResponseEntity = segmentFeignClient.getSegmentId(LeafKeyEnum.USER_ID_KEY.value());
if (!uidResponseEntity.isSuccess() || !userIdResponseEntity.isSuccess()) {
    throw new ServiceException("Leaf获取id失败");
}
Long userId = userIdResponseEntity.getData();
authAccount.setUid(uidResponseEntity.getData());
authAccount.setUserId(userId);
authAccount.setCreateIp(IpUtils.getIpAddr(ServletUtils.getRequest()));
//保存用户到auth数据库
authAccountMapper.save(authAccount);
//保存用户角色到数据库
authAccountMapper.saveAuthRole(uidResponseEntity.getData(), AuthRoleEnum.COMMON_ROLE.value());
//同步到user模块

```

图5.11 登录实现

该方法传入用户学号及密码，首先从本地数据库查询是否有该用户，如果有且校验密码正确，则查询该用户权限，并返回用户信息。

如果用户不存在，或者密码错误，则调用website服务的getByAccountAndPassword接口校验用户信息，如果是密码错误则更新密码，如果用户不存在则从leaf服务获取全局唯一id，然后将用户保存到auth数据库，并将用户信息同步到user服务数据库中，最后返回用户信息。

### (3) 返回参数:

当用户账号验证通过后，会将用户信息封装为一个Loginuser对象，包含用户ID、用户名、用户学号、登录ip、用户权限、用户角色等，如下：

```
public class LoginUser implements Serializable {
    private static final long serialVersionUID = 1L;

    /**
     * 用户唯一标识
     */
    private String token;

    /**
     * 用户名id
     */
    private Long userId;

    /**
     * 学号
     */
    private String account;

    /**
     * 登录时间
     */
    private Long loginTime;

    /**
     * 过期时间
     */
    private Long expireTime;

    /**
     * 登录IP地址
     */
    private String ipaddr;

    /**
     * 权限列表
     */
}
```

图5.12 登录对象

随后将随机生成一个uuid，并对uuid进行jwt加密，最后uuid作为key将loginuser对象存入redis缓存：

```
    /**
     * 刷新令牌有效期
     * 存入redis
     * @param loginUser 登录信息
     */
    public void refreshToken(LoginUser loginUser) {
        loginUser.setLoginTime(System.currentTimeMillis());
        loginUser.setExpireTime(loginUser.getLoginTime() + expireTime * MILLIS_MINUTE);
        String tokenKey = getTokenKey(loginUser.getToken());
        redisService.setCacheObject(tokenKey, loginUser, expireTime, TimeUnit.MINUTES);
    }
}
```

图5.13 刷新令牌

返回一个map，存入uuid密文作为Token，以及登录超时时间

```

    /**
     * 创建令牌
     */
    public Map<String, Object> createToken(LoginUser loginUser) {
        String token = IdUtils.fastUUID();
        loginUser.setToken(token);
        loginUser.setIpAddr(IpUtils.getIpAddr(ServletUtils.getRequest()));
        refreshToken(loginUser);
        // Jwt存储信息
        Map<String, Object> claimsMap = new HashMap<>();
        claimsMap.put(SecurityConstants.USER_KEY, token);

        Map<String, Object> rspMap = new HashMap<>();
        rspMap.put("access_token", JwtUtils.createToken(claimsMap));
        rspMap.put("expires_in", expireTime);
        return rspMap;
    }

    /**

```

图5.14 创建令牌

## 4.8 用户权限校验

在系统接口的权限校验中，有众多成熟可靠的权限校验框架，但是本系统不采用第三方认证框架，而是通过自定义注解，通过spring的aop来自定义实现权限校验方法，在工具包下独立开发一个权限校验模，如下：

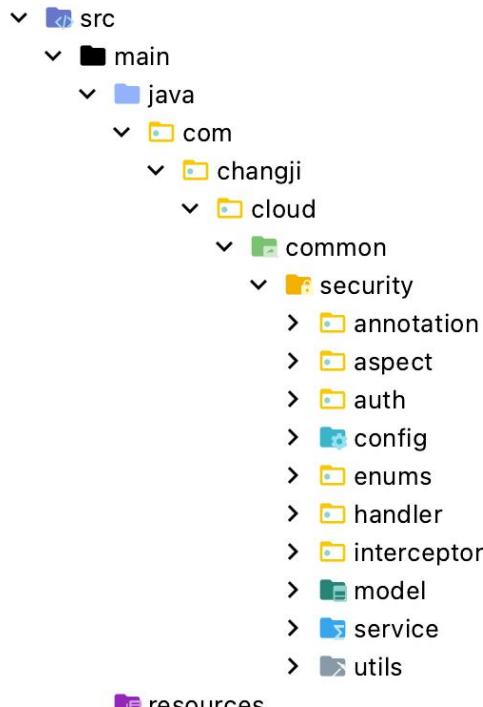


图5.15 校验模块

具体实现方式如下：

```
public class HeaderInterceptor implements AsyncHandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        if (!(handler instanceof HandlerMethod)) {
            return true;
        }

        SecurityContextHolder.setPassword(ServletUtils.getHeader(request, SecurityConstants.AUTH_PASSWORD));
        //从当前线程获取 token
        String token = SecurityUtils getToken();
        if (StringUtils.isNotEmpty(token)) {
            //从redis获取用户信息
            LoginUser loginUser = AuthUtil.getLoginUser(token);
            if (StringUtil.isNotNull(loginUser)) {
                //验证过期时间
                AuthUtil.verifyLoginUserExpire(loginUser);
                //存入 thread_local
                SecurityContextHolder.set(SecurityConstants.LOGIN_USER, loginUser);
            }
        }
        return true;
    }
}
```

图5.16 获取用户信息

(1) 用户在访问接口的过中携带用户Token，在调用目标接口之前，首先会执行 HeaderInterceptor类拦截方法，获取用户token，并解析token信息，获取redis key，再从 redis中获取到对应的loginuser对象，SecurityContextHolder当中定义了一个 TransmittableThreadLocal<sup>i</sup>，将用户信息存入ThreadLocal中，方便后续调用。

```
public class SecurityContextHolder {
    private static final TransmittableThreadLocal<Map<String, Object>> THREAD_LOCAL = new TransmittableThreadLocal<>();

    public static void set(String key, Object value) {
        Map<String, Object> map = getLocalMap();
        map.put(key, value == null ? StringUtils.EMPTY : value);
    }
}
```

图5.17 threadlocal

在需要权限校验的接口添加我们的自定义权限注解<sup>ii</sup>@RequiresPermissions并定义接口所需要的权限如 (system:course:query) ，

```
@PostMapping("course/get")
@RequiresPermissions("system:course:query")
@ApiOperation("查询我的")
@Log(title = "查询我的课表")
public ServerResponseEntity<List<List<Lesson>>> getAccountCourse(@Validated @RequestBody QueryCourseDTO query
    List<List<Lesson>> courseList = courseService.getMyCourseList(queryCourseDTO);
    return ServerResponseEntity.success(courseList);
}
```

图5.18 查询课表

当用户访问的接口带有限权校验注解，则会通过spring的AOP<sup>iii</sup>拦截，在执行接口逻辑之前，首先执行我们的权限校验逻辑。

```

@Around("pointcut()")
public Object around(ProceedingJoinPoint joinPoint) throws Throwable {
    MethodSignature signature = (MethodSignature) joinPoint.getSignature();
    checkMethodAnnotation(signature.getMethod());
    try {
        // 执行原有逻辑
        Object obj = joinPoint.proceed();
        return obj;
    }
    catch (Throwable e){
        throw e;
    }
}

/**
 * 检查方法的注解 根据权限注解检查用户权限
 * @param method
 */
public void checkMethodAnnotation(Method method) {

    RequiresPermissions requiresPermissions = method.getAnnotation(RequiresPermissions.class);
    if (requiresPermissions != null) {
        AuthUtil.checkPermissions(requiresPermissions);
    }
}

```

图5.19 权限切片

权限校验过程中，通过比较接口所需权限和loginuser对象中的用户权限，来确定用户是否具有该接口的访问权限，如果校验通过，则执行接口逻辑，如果验证未通过，则抛出异常：NotPermissionException

```
/*
 * 验证用户是否含有指令权限，必须全部拥有
 * @param permissions
 */
public void checkPermissionAnd(String... permissions) {
    //当前用户权限列表
    List<String> permissionList = getPermissionList();

    for (String permission : permissions) {
        if (!hasPermission(permissionList, permission)){
            throw new NotPermissionException(permission);
        }
    }
}

public void checkPermissionOr(String... permissions) {
    List<String> permissionList = getPermissionList();
    for (String permission : permissions) {
        if (hasPermission(permissionList, permission)) {
            return;
        }
    }
    if (permissions.length > 0) {
        throw new NotPermissionException(permissions);
    }
}
```

图5.20 权限校验

## 4.9 服务网关及配置

开发网关模块，网关有多种框架，包括Zuul、Spring Cloud Gateway、Kong等，采用 Spring Cloud Gateway框架，微服务架构中的服务通常会被拆分成多个小型的、自治的服务，每个服务都有自己的数据存储、业务逻辑和API接口。

这种架构使得服务更加容易维护、部署和扩展，但也带来了一些问题，例如服务的发现、服务的路由、负载均衡、安全认证等等。

网关配置如下：

```

routes:
  - id: auth-service
    uri: lb://cjxyccloud-auth
    predicates:
      - Path=/auth/**
    filters:
      - StripPrefix=1

  - id: website-service
    uri: lb://cjxyccloud-website
    predicates:
      - Path=/website/**
    filters:
      - StripPrefix=1

  - id: user-service
    uri: lb://cjxyccloud-user
    predicates:
      - Path=/user/**
    filters:
      - StripPrefix=1

  - id: social-service
    uri: lb://cjxyccloud-social
    predicates:
      - Path=/social/**
    filters:
      - StripPrefix=1

```

图5.21 网关配置

router表示各服务路由

- ①id: 服务名称，表示一个路由规则。
- ②uri: 在服务注册中心找对应服务名的服务，服务集群自动负载均衡
- ③predicates: 路由断言，将相对应的路由匹配到服务。
- ④filters: 过滤掉url中指定顺序的字符

路由异常请求进行处理:

```

public Mono<Void> handle(ServerWebExchange exchange, Throwable ex) {
    ServerHttpResponse response = exchange.getResponse();

    if (exchange.getResponse().isCommitted()) {
        return Mono.error(ex);
    }

    String msg;

    if (ex instanceof NotFoundException) {
        msg = "服务未找到";
    } else if (ex instanceof ResponseStatusException) {
        ResponseStatusException responseStatusException = (ResponseStatusException) ex;
        msg = responseStatusException.getMessage();
    } else {
        msg = "内部服务器错误";
    }

    log.error("[网关异常处理]请求路径:{}，异常信息:{}" , exchange.getRequest().getPath(), msg);

    return ServletUtils.webFluxResponseWriter(response, msg);
}

```

图5.22 网关异常

Swagger是一种用于构建、文档化和测试RESTful Web服务的开源工具。它可以根据API的注解生成接口文档。降低开发人员的工作量，同时也可以提高API的可读性、可测试性和可重用性，配置如下：

```
@Autowired
public SwaggerHandler(SwaggerResourcesProvider swaggerResources) { this.swaggerResources = swaggerResources; }

@GetMapping(@RequestMapping("/swagger-resources/configuration/security"))
public Mono<ResponseEntity<SecurityConfiguration>> securityConfiguration() {
    return Mono.just(new ResponseEntity<SecurityConfiguration>(
        Optional.ofNullable(securityConfiguration).orElse(SecurityConfigurationBuilder.builder().build()), HttpStatus.OK));
}

@GetMapping(@RequestMapping("/swagger-resources/configuration/ui"))
public Mono<ResponseEntity<UiConfiguration>> uiConfiguration() {
    return Mono.just(new ResponseEntity<UiConfiguration>(
        Optional.ofNullable(uiConfiguration).orElse(UiConfigurationBuilder.builder().build()), HttpStatus.OK));
}

@GetMapping(@RequestMapping("/swagger-resources"))
public Mono<ResponseEntity> swaggerResources() {
    return Mono.just((new ResponseEntity<SwaggerResources>(swaggerResources.get(), HttpStatus.OK)));
}
```

图5.22 swagger

## 4.10 查询学生课表

该方法实现查询学生的学校课表，通过调用教务管理系统的查询课表接口获取课表数据，并将查询到的数据同步到本地数据库。

请求路径为/course/get，需要用户权限system:course:query

```
@Autowired
private CourseService courseService;

@PostMapping(@RequestMapping("/course/get"))
@RequiresPermissions("system:course:query")
@ApiOperation("查询我的")
@Log(title = "查询我的课表")
public ServerResponseEntity<List<List<Lesson>>> getAccountCourse(@Validated @RequestBody QueryCourseDTO queryCourseDTO) {
    List<List<Lesson>> courseList = courseService.getMyCourseList(queryCourseDTO);
    return ServerResponseEntity.success(courseList);
}

@PostMapping(@RequestMapping("/course/query"))
@RequiresPermissions("system:course:query")
@ApiOperation("查询课表")
@Log(title = "查询课表")
public ServerResponseEntity<List<List<Lesson>>> queryCourseList(@Validated @RequestBody QueryCourseDTO queryCourseDTO) {
    List<List<Lesson>> courseList = courseService.queryCourseList(queryCourseDTO);
    return ServerResponseEntity.success(courseList);
}
```

图5.23 课表接口

(1) 接口所需参数为QueryCourseDTO，周次、学期、院校、班级，并通过@NotBlank注解规定参数不能为空。

```

    @ApiModelProperty("周次")
    @NotBlank(message = "周次不能为空")
    private String weekly;

    @NotBlank(message = "学期不能为空")
    @ApiModelProperty("学期")
    private String semester;

    @NotBlank(message = "院校不能为空")
    @ApiModelProperty("院校")
    private String institute;

    @NotBlank(message = "班级不能为空")
    @ApiModelProperty("班级")
    private String classes;

```

图5.24 课表封装

(2) 方法具体实现,首先通过传入的参数从本地数据库尝试查询课本信息,若查询到,则直接返回,若没有,从thread\_local获取用户名和密码,封装教务管理系统的post请求参数,发起http请求,从教务管理系统获取到课本信息,将其保存到本地数据库,并返回。

由于从教务系统获取的数据为一个html文件,所以需要对其进行数据处理,使用jsoup<sup>iv</sup>工具解析html文件,封装成Courses格式。

```

//获取教务管理系统cookie
CookieStore cookieStore = cookieService.getCookie(user);
String url = HttpConstants.MY_COURSE_URL.value();
List<NameValuePair> list = new ArrayList<>();
list.add(new BasicNameValuePair(name: "cj0701id", value: ""));
list.add(new BasicNameValuePair(name: "zc", queryCourseDTO.getWeekly())); //周次
list.add(new BasicNameValuePair(name: "demo", value: ""));
list.add(new BasicNameValuePair(name: "xnxq01id", queryCourseDTO.getSemester())); //学年学期
list.add(new BasicNameValuePair(name: "sffD", value: ""));
list.add(new BasicNameValuePair(name: "wkbkc", value: "1")); //无课表课程
list.add(new BasicNameValuePair(name: "kbjcmsid", value: "A63DB2E690D94F43945978F87DBE5140")); //默认节次模式

//发起http请求 从官网查询数据
CloseableHttpResponse response = HttpClientUtils.getResponse(url, cookieStore, list, context: null);
String context = BufferUtil.inputToString(response);
//解析 html
List<List<Lesson>> courseList = JsoupUtil.getMyCourseList(context);

//保存到本地数据库
String courseInfo = JSONObject.toJSONString(courseList);
Courses courses = mapperFacade.map(queryCourseDTO, Courses.class);
courses.setCourseInfo(courseInfo);
courses.setStatus(1);
coursesMapper.save(courses);

```

图5.25 解析课表

## 4.11 朋友圈功能实现

朋友圈功能包括保存朋友圈、删除朋友圈、获取朋友圈、点赞朋友圈等，该接口全部需要"common:social:friendCircle"权限，映射url路径以social开头。

```

@PostMapping(@RequestMapping("/social/saveFriendCircle"))
@RequiresPermissions("common:social:friendCircle")
@ApiOperation("保存朋友圈")
@Log(title = "保存朋友圈")
public ServerResponseEntity<Void> saveFriendCircleMessage(@Validated @RequestBody FriendCircleDTO friendCircleDTO) {
    friendCircleService.saveFriendCircleMessage(friendCircleDTO);
    return ServerResponseEntity.success();
}

@DeleteMapping(@RequestMapping("/social/deleteFriendCircle/{friendCircleId}"))
@RequiresPermissions("common:social:friendCircle")
@ApiOperation("删除朋友圈")
@Log(title = "删除朋友圈")
public ServerResponseEntity<Void> deleteFriendCircleMessage(@PathVariable("friendCircleId") Long friendCircleId) {
    friendCircleService.deleteFriendCircleMessage(friendCircleId);
    return ServerResponseEntity.success();
}

@PostMapping(@RequestMapping("/social/getFriendCircle/{userId}"))
@RequiresPermissions("common:social:friendCircle")
@ApiOperation("获取朋友圈")
@Log(title = "获取朋友圈")
public ServerResponseEntity<List<FriendCircleMessageVO>> getFriendCircleMessageMapper(@PathVariable("userId") Long
    List<FriendCircleMessageVO> friendCircleMessageList = friendCircleService.getFriendCircleList(userId, page);
    return ServerResponseEntity.success(friendCircleMessageList);
}

@GetMapping(@RequestMapping("/social/likedFriendCircle/{friendCircleId}"))
@RequiresPermissions("common:social:friendCircle")
@ApiOperation("点赞朋友圈")
@Log(title = "点赞朋友圈")
public ServerResponseEntity<Void> likedFriendCircle(@PathVariable("friendCircleId") Long friendCircleId) {
    friendCircleService.likedFriendCircle(friendCircleId);
    return ServerResponseEntity.success();
}

```

图5.26 朋友圈接口

(1) 保存朋友圈，朋友圈内容包括图片和正文，通过loginUser对象获取用户id，将该朋友圈dto转化为FriendCircleMessage对象，直接存入数据库即可。

```

@Override
@Transactional
public void saveFriendCircleMessage(FriendCircleDTO friendCircleDTO) {
    FriendCircleMessage friendCircleMessage = mapperFacade.map(friendCircleDTO, FriendCircleMessage.class);
    friendCircleMessage.setUserId(SecurityUtils.getLoginUser().getUserId());
    friendCircleMessageMapper.save(friendCircleMessage);
}

```

图5.27 保存朋友圈

(2) 删除朋友圈，传入参数朋友圈id，从loginuser对象获取用户id，通过userid和朋友圈id删除该朋友圈，若删除成功则返回成功，若失败，则抛出异常。

```
@Override  
@Transactional  
public void deleteFriendCircleMessage(Long friendCircleId) {  
    int i = friendCircleMessageMapper.deleteByFriendCircleIdAndUserId(friendCircleId, SecurityUtils.getLoginUser().getUserId());  
    if (i == 0) {  
        throw new ServiceException("非法删除");  
    }  
}
```

图5.28 删除朋友圈

(3) 点赞朋友圈，朋友圈点赞功能需要redis缓存辅助实现，首先判定是否存在该朋友圈，然后判定用户是否已经给该朋友圈点赞，若已经点击则取消点赞，若未点赞，则为点赞操作，将信息更新到redis缓存中。

```
public void likedFriendCircle(Long friendCircleId) {  
    Long userId = SecurityUtils.getLoginUser().getUserId();  
  
    //判断是否存在该朋友圈信息  
    FriendCircleMessage friendCircleMessageById = getFriendCircleMessageById(friendCircleId);  
    if (Objects.isNull(friendCircleMessageById)) {  
        throw new ServiceException("没有该朋友圈信息");  
    }  
    //判断是否已经点赞过 已经点赞过则为取消点赞  
    if (isLiked(friendCircleId, userId)) {  
        throw new ServiceException("您已点赞过该朋友圈");  
        likedRedisService.unlikeFromRedis(friendCircleId, userId);  
        likedRedisService.decrementLikedCount(friendCircleId);  
        return;  
    }  
  
    // 点赞  
    //缓存到redis  
    likedRedisService.saveLiked2Redis(friendCircleId, userId);  
    likedRedisService.incrementLikedCount(friendCircleId);
```

图5.29 点赞朋友圈

添加定时任务，每过一定时间，将redis点赞信息同步到数据库

```

@.Autowired
LikedService likedService;

private SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss");

@Override
protected void executeInternal(JobExecutionContext jobExecutionContext) throws JobExecutionException {
    log.info("LikeTask----- {}", sdf.format(new Date()));

    //将 Redis 里的点赞信息同步到数据库
    likedService.transLikedFromRedis2DB();
    likedService.transLikedCountFromRedis2DB();
}

```

图5.30 定时任务

(4) 获取朋友圈信息，传入用户id及分页信息，从数据库及redis查询信息，并进行汇总，转换为vo对象，最后返回。

```

Long myUserId= SecurityUtils.getLoginUser().getUserId();
if (userId == 0) {
    userId = null;
}
//获取最新的朋友圈
Integer friendCount = friendCircleMessageMapper.getFriendCircleMessageCount();
Integer num = friendCount - page.getPageNum();
Integer startIndex;
Integer pageSize;
if (num >= page.getPageNum()) {
    startIndex = friendCount - page.getPageNum() - page.getPageSize();
    pageSize = page.getPageSize();
} else if (num < page.getPageSize() && num >= 0) {
    startIndex = 0;
    pageSize = page.getPageSize();
} else {
    return null;
}
//分页
PageHelper.startPage(startIndex, pageSize);
List<FriendCircleMessage> list = friendCircleMessageMapper.getFriendCircle(userId);
PageInfo<FriendCircleMessage> pageInfo = new PageInfo<>(list);
List<FriendCircleMessage> friendCircleMessageList = pageInfo.getList();
List<FriendCircleMessageVO> friendCircleMessageVOList = new ArrayList<>();
//转换对象
for (FriendCircleMessage friendCircleMessage : friendCircleMessageList) {
    FriendCircleMessageVO friendCircleVO = mapperFacade.map(friendCircleMessage, FriendCircleMessageVO.class)
    //汇总点赞数 redis + mysql
    friendCircleVO.setLikedCount(friendCircleVO.getLikedCount() + likedRedisService.getOneLikedCountFromRedis);
    //判定是否点赞过
    friendCircleVO.setLiked(isLiked(friendCircleVO.getId(), myUserId));
    //设置用户信息 feign 调用用户模块查询
    ServerResponseEntity<UserFriendCircleVO> responseEntity = userFeignClient.queryUserDetailById(friendCircleVO.getId());
    if (!responseEntity.isSuccess()) {

```

图5.31 获取朋友圈

(5) 获取朋友圈评论，设置默认分页大小10页，从数据库查询朋友圈评论信息，并查询每条评论的用户信息，进行封装返回。

```

@Override
public List<CommentVO> getCommentList(GetCommentDTO commentDTO, Page page) {
    if (page.getPageNum() == null) {
        page.setPageNum(0);
        page.setPageSize(10);
    }
    List<FriendCircleComment> comments = commentMapper.getComments(commentDTO.getFriendCircleId(), commentDTO.getRootCommentId(), page);
    List<CommentVO> commentVOS = new ArrayList<>();
    comments.forEach(comment -> {
        CommentVO commentVO = mapperFacade.map(comment, CommentVO.class);
        ServerResponseEntity<UserFriendCircleVO> responseEntity = userFeignClient.queryUserDetailByAccount(commentVO.getAccount());
        if (!responseEntity.isSuccess()) {
            throw new ServiceException("查询用户信息失败");
        }
        UserFriendCircleVO userFeignVO = responseEntity.getData();
        commentVO.setNickName(userFeignVO.getNickName());
        commentVO.setAvatar(userFeignVO.getPicture());
        if (commentVO.getRootCommentId() == null) {
            commentVO.setCommentCount(commentMapper.countByCommentId(commentVO.getCommentId()));
        } else {
            commentVO.setCommentCount(0);
        }
        commentVOS.add(commentVO);
    });
    return commentVOS;
}

```

图5.31 获取评论

(6) 评论朋友圈，首先判断被回复朋友圈是否存在，然后判断被回复评论的是朋友圈还是回复评论，存入数据库。

```

@Override
public boolean save(CommentDTO commentDTO) {
    if (commentDTO.getFriendCircleId() == null) {
        throw new ServiceException("朋友圈id不能为空");
    }
    FriendCircleMessage friendCircleMessageById = friendCircleService.getFriendCircleMessageById(commentDTO.getFriendCircleId());
    if (StringUtil.isNull(friendCircleMessageById)) {
        throw new ServiceException("朋友圈不存在");
    }

    FriendCircleComment comment = mapperFacade.map(commentDTO, FriendCircleComment.class);
    comment.setAccount(SecurityUtil.getAccount());
    if (StringUtil.isNull(comment.getRootCommentId()) && StringUtil.isNotNull(comment.getToCommentId())) {
        throw new ServiceException("非法评论");
    }

    // 判断是否为回复顶级评论 root不为空 toComment为空
    if (StringUtil.isNotNull(comment.getRootCommentId()) && StringUtil.isNull(comment.getToCommentId())) {
        FriendCircleComment commentMapperById = commentMapper.findById(comment.getRootCommentId(), comment.getFriendCircleId());
        if (StringUtil.isNull(commentMapperById)) {
            throw new ServiceException("顶级评论不存在");
        }
    }

    // 判断是否为回复次级评论 root不为空 toComment不为空
    if (StringUtil.isNotNull(comment.getRootCommentId()) && StringUtil.isNotNull(comment.getToCommentId())) {
        FriendCircleComment rootComment = commentMapper.findById(comment.getRootCommentId(), comment.getFriendCircleId());
        if (StringUtil.isNull(rootComment)) {
            throw new ServiceException("次级评论不存在");
        }
        FriendCircleComment toComment = commentMapper.findById(comment.getToCommentId(), comment.getFriendCircleId());
        if (StringUtil.isNull(toComment)) {
            throw new ServiceException("次级评论不存在");
        }

        if (toComment.getRootCommentId() != comment.getRootCommentId()) {
            throw new ServiceException("非法评论");
        }
    }
}

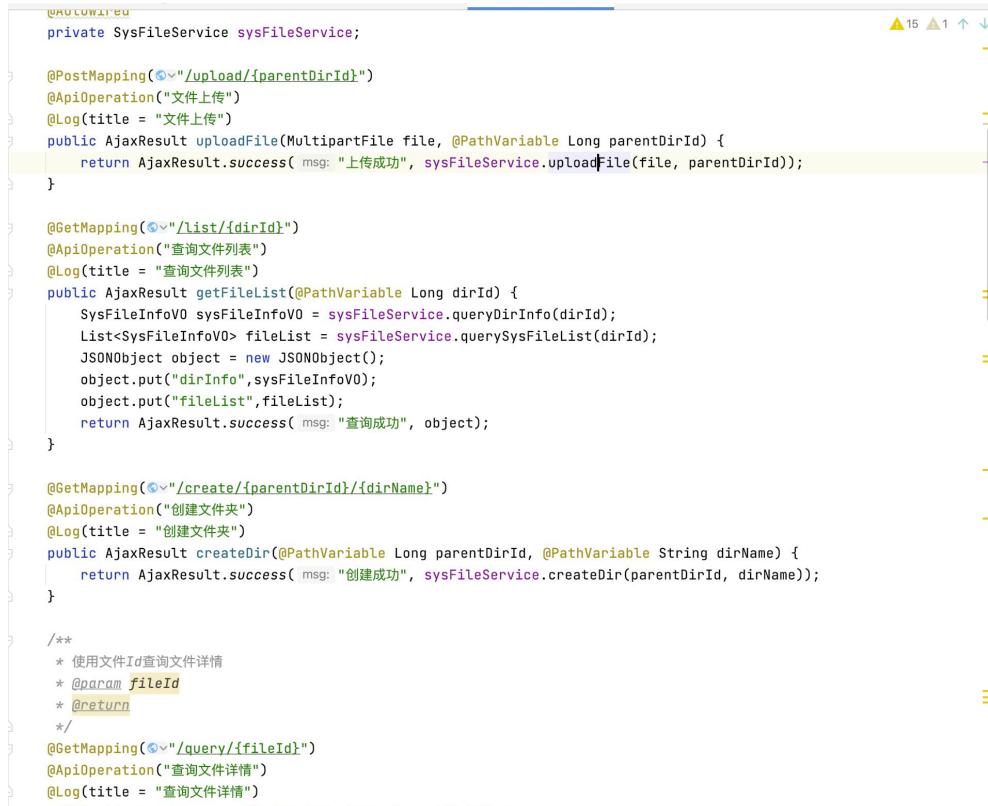
```

图5.32 评论

## 4.12 文件上传下载功能实现

实现文件的上传、下载及预览功能，使用minio对文件对象进行存储。MinIO是一个开源的分布式对象存储服务器，专为云原生应用和基础架构而设计。

具体实现如下所示：



```

private SysFileService sysFileService;

@PostMapping(@RequestMapping("/upload/{parentDirId}"))
@ApiOperation("文件上传")
@Log(title = "文件上传")
public AjaxResult uploadFile(MultipartFile file, @PathVariable Long parentDirId) {
    return AjaxResult.success(msg: "上传成功", sysFileService.uploadFile(file, parentDirId));
}

@GetMapping(@RequestMapping("/list/{dirId}"))
@ApiOperation("查询文件列表")
@Log(title = "查询文件列表")
public AjaxResult getFileList(@PathVariable Long dirId) {
    SysFileInfoVO sysFileInfoVO = sysFileService.queryDirInfo(dirId);
    List<SysFileInfoVO> fileList = sysFileService.querySysFileList(dirId);
    JSONObject object = new JSONObject();
    object.put("dirInfo", sysFileInfoVO);
    object.put("fileList", fileList);
    return AjaxResult.success(msg: "查询成功", object);
}

@GetMapping(@RequestMapping("/create/{parentDirId}/{dirName}"))
@ApiOperation("创建文件夹")
@Log(title = "创建文件夹")
public AjaxResult createDir(@PathVariable Long parentDirId, @PathVariable String dirName) {
    return AjaxResult.success(msg: "创建成功", sysFileService.createDir(parentDirId, dirName));
}

/**
 * 使用文件Id查询文件详情
 * @param fileId
 * @return
 */
@GetMapping(@RequestMapping("/query/{fileId}"))
@ApiOperation("查询文件详情")
@Log(title = "查询文件详情")

```

图5.33 文件接口

(1) 文件上传，传入文件对象，及文件夹id，根文件夹id为-1，首先获取文件名，查询文件夹信息，对文件路径进行拼接，文件夹path+文件名方式实现，检查文件是否存在，使用雪花算法生成文件id，将文件信息保存到数据库，并将文件对象上传到minio服务器。

```

@Override
@Transactional
public SysFileInfoVO uploadFile(MultipartFile file, Long parentDirId) {
    String fileName = file.getOriginalFilename();

    // 存储文件信息到数据库
    SysFileInfo sysFileInfo = sysFileInfoMapper.selectSysFileInfoById(parentDirId);
    if(Objects.isNull(sysFileInfo) || sysFileInfo.getDataType() != 1) {
        throw new ServiceException("文件夹不存在");
    }

    sysFileInfo.setPath(sysFileInfo.getPath() + "/" + fileName);

    // 检查文件是否存在
    if (!Objects.isNull(sysFileInfoMapper.selectSysFileInfoByPath(sysFileInfo.getPath()))) {
        throw new ServiceException("文件已存在");
    }
    //雪花算法生成id
    sysFileInfo.setId(SnowFlakeUtils.getInstance().nextId());
    sysFileInfo.setFileName(fileName);
    sysFileInfo.setExt(FileTypeUtils.getExtension(file));
    sysFileInfo.setData_type(DataTypeConstant.FILE);
    sysFileInfo.setParentDirId(parentDirId);
    sysFileInfo.setSize(file.getSize());
    int i = sysFileInfoMapper.insertSysFileInfo(sysFileInfo);

    // 上传到Minio
    MinioUtils.putObject(minioClient, minioConfig.getBucketName(), sysFileInfo.getPath(), file);
    SysFileInfoVO sysFileInfoVO = mapperFacade.map(sysFileInfo, SysFileInfoVO.class);
    return sysFileInfoVO;
}

```

图5.34 文件上传

(2) 文件下载，传入文件id，从minio服务器获取文件对象，从文件对象获取文件输入流，设置response对象header，最后将文件输入流copy到文件输出流返回，实现文件的下载操作。

```

@GetMapping(value = "/download/{fileId}")
@ApiOperation("下载文件")
@Log(title = "下载文件")
public void download(@PathVariable Long fileId, HttpServletRequest request, HttpServletResponse response) {
    SysFileInfoVO sysFileInfoVO = sysFileService.querySysFileInfoById(fileId);
    OutputStream outputStream = null;
    InputStream inputStream = null;
    try {
        inputStream = sysFileService.getFileInputStream(fileId);
        response.setHeader("Content-Disposition", "attachment;filename=" + sysFileInfoVO.getFileName());
        response.setHeader("Content-Type", "file");
        response.setHeader("Access-Control-Expose-Headers", "Content-Disposition" + ", Download-Type");
        response.setContentLengthLong(Long.parseLong(sysFileInfoVO.getSize()));
        // 获取输出流
        outputStream = response.getOutputStream();
        IOUtils.copy(inputStream, outputStream);
    } catch (IOException e) {
        throw new RuntimeException("文件下载失败:" + e.getMessage());
    } finally {
        try {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

图5.35 下载文件

(3) 图片预览，传入图片id，从mino服务器获取文件对象，获取文件输入流，最后将文件输入流转化为byte数组，返回给前端，实现图片的预览功能。

```

    /**
     * 图片/PDF查看
     *
     * @param fileId
     * @return
     * @throws IOException
     */
    @GetMapping(value = "/view/{fileId}", produces = MediaType.IMAGE_PNG_VALUE)
    @ApiOperation("图片/PDF查看")
    @Log(title = "图片/PDF查看")
    public ResponseEntity<byte[]> viewFilesImage(@PathVariable Long fileId) throws IOException {
        SysFileInfoVO sysFileInfoVO = sysFileService.querySysFileInfoById(fileId);
        if (!SysConstant.IMAGE_TYPE.contains(sysFileInfoVO.getExt())) {
            throw new ServiceException("非图片/PDF类型请先下载");
        }
        InputStream inputStream = sysFileService.getFileInputStream(fileId);
        return new ResponseEntity<>(FileUtils.InputStreamToByte(inputStream), HttpStatus.OK);
    }
}

```

图5.36 图片预览

## 4.13 即时聊天

即时聊天采用websocket技术，因为http是一种短链接技术，采用一问一答的方式实现通信，服务器无法主动向客户端发起通信，而websocket是一种长连接技术，服务端可以向客户端发起通信，进而实现即时聊天功能，具体开发如下：

## (1) 用户连接

```

@OnOpen
public void onOpen(Session session, @PathParam("token") String token, EndpointConfig config){
    if (StringUtils.isEmpty(token)) {
        return;
    }
    //从redis获取用户信息
    LoginUser loginUser = AuthUtil.getLoginUser(token);
    if (StringUtils.isNull(loginUser)) {
        try {
            session.getBasicRemote().sendText(s: "{code:400, message: 用户未登录}");
            session.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        log.info("websocket用户未登录");
        return;
    }
    //验证过期时间
    AuthUtil.verifyLoginUserExpire(loginUser);
    // 存入 thread_local
    SecurityContextHolder.set(SecurityConstants.LOGIN_USER, loginUser);
    if(chatService == null){
        this.chatService = SpringUtils.getBean(ChatService.class);
    }
    account = loginUser.getAccount();
    chatService.saveSession(session, loginUser.getAccount());
}

```

图5.37 用户连接

首先，用户连接时需要携带token，进行用户身份认证，认证通过后保存用户session，将用户session存储在SessionContext的ConcurrentHashMap中，方便后续使用。

## (2) 发送消息

```

@OnMessage
public void onMessage(String message) throws IOException {
    SessionContext.setAccount(account);
    ChatMessages chatMessages = ChatUtils.parseMessage(message);
    if (chatMessages.getCode() == 9999) {
        chatService.heartPacket(chatMessages);
    } else {
        chatService.sendMessageById(chatMessages);
    }
}

```

图5.38 发送消息

首先自定义消息格式如下

```
private Long id;  
private int code;  
//消息类型  
private String type;  
private String fromUserAccount;  
private String toUserAccount;  
private String content;  
private Integer state;
```

图5.39 消息格式

包括，消息id，消息代码，消息类型，消息发出者id，消息接受者id，消息主题，消息状态。

服务器接收到消息后首先进行消息解码，然后判定消息是否为心跳包，如果不是心跳包，则发送消息

```
@Override  
public void sendMessageById(ChatMessages chatMessages) {  
  
    String account = SessionUtils.getAccount();  
    log.info(account + "发送消息给:" + chatMessages.getToUserAccount());  
  
    // 验证是否为好友  
    UserRelationship userRelationship = userRelationshipMapper.selectToWayRelationship(account, chatMessages.getToUserAccount());  
    if (StringUtils.isNull(userRelationship)) {  
        log.error("不是好友关系");  
        Result.sendLocal(code: 4040, message: "不是好友关系", data: null);  
        return;  
    }  
    chatMessages.setFromUserAccount(account);  
    // 保存聊天信息  
    chatMessagesMapper.insertMessage(chatMessages);  
    Result.success(message: "发送消息", chatMessages.getContent(), chatMessages.getToUserAccount());  
}
```

图5.40 消息转发

验证消息接收者是否为好友，将消息保存进数据库，最后将消息转发给消息接收者，完成聊天动作。

## 第五章 前端界面设计

为了使软件在Android和ios上都能运行，前端采用Flutter跨平台框架开发，Flutter是一个开源的移动应用程序开发框架，由谷歌公司开发。它使用Dart编程语言，并结合了响应式编程模型、现代化的UI库和高性能的渲染引擎，可以快速构建高性能、跨平台的移动应用程序。

Flutter is a developer-friendly, open source toolkit created by Google that you can use to create applications for Android and iOS mobile devices, and now also for the web and desktop<sup>[8]</sup>.

项目结构如下所示：

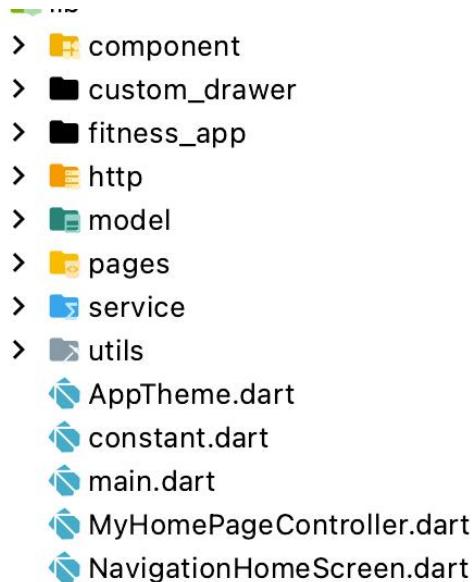


图5.1 项目结构

- (1) component包用来开发项目中会重复使用的元素，例如按钮、输入框、表单等。
- (2) http包开发与后端对接的接口，进行统一的管理，使用flutter的http包进行二次开发。
- (3) model包开发前后端交互过程中使用的实体类，方便对数据进行封装与调用。
- (4) pages包开发前端各个识图界面，例如首页、朋友圈页面、好友页面、登录页面等。
- (5) service包进行一些业务的开发，例如本地数据存储等。
- (6) utils开发一些项目中使用的工具，例如日期工具、字符串工具等。

## 5.1 登录界面

登录界面输入用户账号和密码，账号为学校教务管理系统学号，密码为教务管理系统学生密码，点击登录按钮进行登录。登录验证通过后调用获取用户信息接口，获取用户信息后登录成功。



图5.2 登录界面

## 5.2 首页界面

首页侧滑展示用户信息介绍，包括用户头像，用户昵称，以及退出登录按钮。首页展示用户学生教务系统的学生课本，通过调用学校教务系统接口生成，课本与教务系统课本保持相同。每节课都自动生成随机颜色，然后对课本进行渲染。

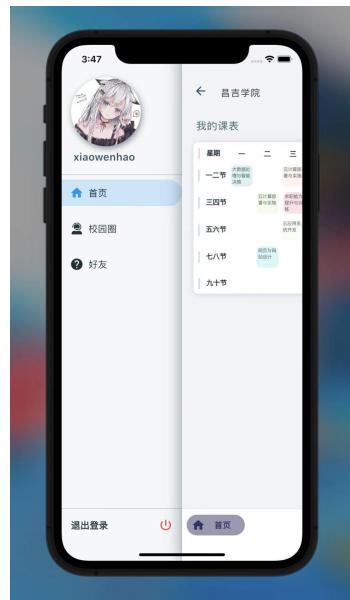


图5.3 用户界面



图5.4 首页

### 5.3 好友界面

好友页面展示好友列表，聊天界面展示聊天记录，可上拉刷新聊天记录，下拉查询历史聊天记录。

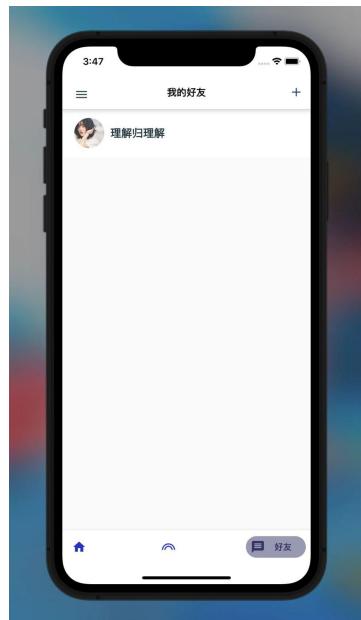


图5.5 好友页面

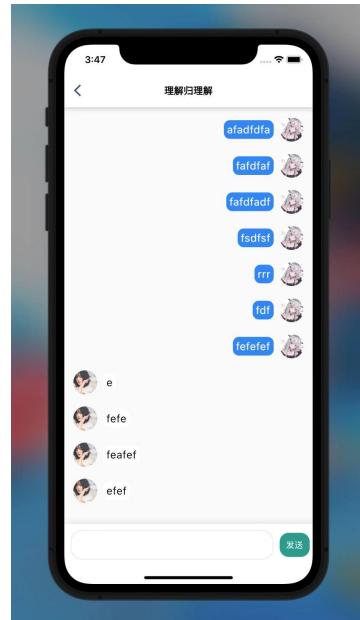


图5.6 聊天页面

### 5.4 校园圈界面

校园圈展示用户动态分享，包括校园圈文本以及图片，可下拉刷新，上拉加载。评论页面展示校园圈评论列表，并可对校园圈进行回复。

## 基于 JAVA 和 Flutter 的校园社交 APP

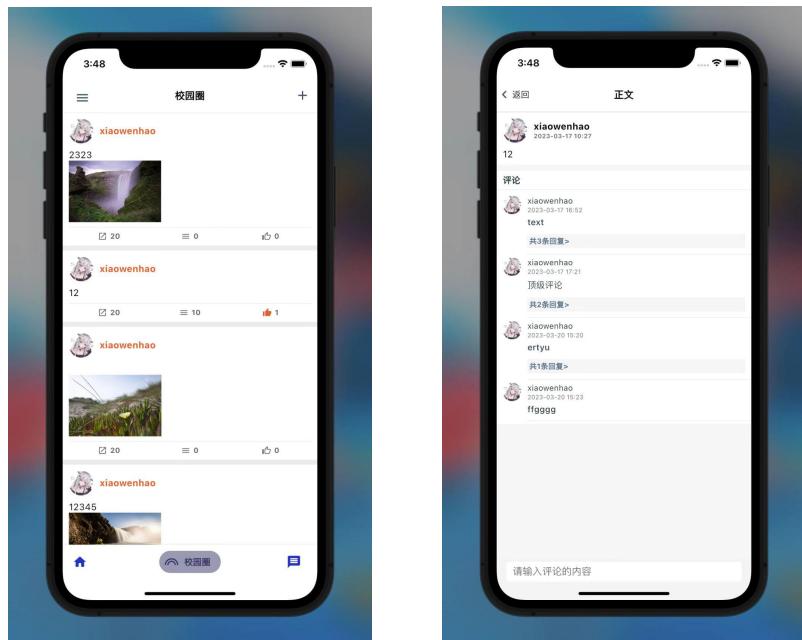


图5.7 校园圈页面

图5.8 评论页面

## 第六章 服务部署

服务进行容器话部署，容器模型其实跟虚拟机模型相似，其主要的区别在于，容器的运行不会独占操作系统。实际上，运行在相同宿主机上的容器是共享一个操作系统的，这样就能够节省大量的系统资源，如CPU、RAM以及存储<sup>[9]</sup>。

容器使用docker容器，通过每个服务单独的dockerfile与统一docker-compose.yml文件在linux服务器上实现一键话服务部署。

在linux服务器安装docker，创建对应的项目文件，Linux的文件系统提供了一种可以收纳其他文件的特殊文件，这种文件称为目录。我们不仅能把文件放到目录中，甚至还能把别的目录放到目录中。不同目录下的多个文件可以取相同的名称<sup>[10]</sup>。

Docker Compose 的主要特点包括：

- (1) 多容器管理，Docker Compose 可以一次性管理多个容器，可以定义每个容器的镜像、端口映射、数据卷、网络等。
- (2) 容器编排，Docker Compose 可以定义多个容器之间的关系和启动顺序，实现容器编排。
- (3) 可移植性，Docker Compose 使用 YAML 文件定义应用，可以轻松地在不同环境中部署相同的应用。
- (4) 简化部署，Docker Compose 可以自动化应用程序的部署过程，无需手动创建和配置容器。
- (5) 可扩展性，Docker Compose 可以在多个服务器上运行，实现分布式部署和负载均衡。

### 6.1 Dockerfile

#### (1) dockerfile文件案例（服务网关）

```
# 基础镜像 网关 gateway
FROM openjdk:8-jre
# author
MAINTAINER xiaowenhao
# 挂载目录
VOLUME /home/changji
# 创建目录
RUN mkdir -p /home/changji
# 指定路径
WORKDIR /home/changji
# 复制jar文件到路径
COPY ./jar/cjxyccloud-gateway.jar /home/changji/cjxyccloud-gateway.jar
# 启动认证服务
ENTRYPOINT ["java","-jar","cjxyccloud-gateway.jar"]
```

图6.1 dockerfile

首先确定容器基础镜像 (openjdk: 8) 。指导容器内挂载目录，创建挂载目录，然后指定项目的工作目录为容器挂载目录，最后将服务jar包复制到工作目录，编写容器启动服务命令，完成 dockerfile 的编写。

## (2) 项目服务

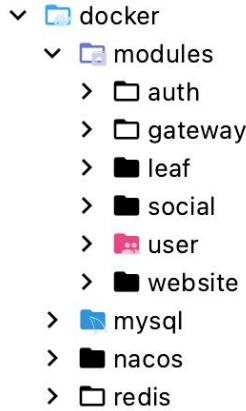


图6.2 项目服务

项目服务包括基础服务与模块服务，各模块均使用上述 dockerfile 文件编写方式进行编写，方便后续服务的一键部署。

① 基础服务：包括 mysql 数据库、nacos 注册中心、redis 数据库，基础服务为系统依赖服务，必须在模块服务前启动，否则模块服务启动将会报错，导致系统启动失败。

② 模块服务：包括 auth 身份认证服务、gateway 网关服务、leaf id 模块、social 设计模块、website 官网模块，模块服务为系统核心业务功能实现的模块。

## 6.2 DockerCompose

docker-compose 可以依照 dockerfile 文件对服务进行自动构建

docker-compose 案例（Nacos）

```
version: '3'
services:
  cjxyccloud-nacos:
    container_name: cjxyccloud-nacos
    image: nacos/nacos-server
    build:
      context: ./nacos
    environment:
      - MODE=standalone
      - JVM_XMS=256m
      - JVM_XMX=256m
    volumes:
      - ./nacos/logs/:/home/nacos/logs
      - ./nacos/conf/application.properties:/home/nacos/conf/application.properties
    ports:
      - "8848:8848"
      - "9848:9848"
      - "9849:9849"
    depends_on:
      - cjxyccloud-mysql
```

图6.3 docker-compose

docker-compose中每个service就是一个服务， container\_name指定容器名称， image 指定容器镜像， build指定容器的dockerfile文件位置， environment为容器启动环境， volumes为容器挂载目录， ports指定服务端口映射，可以映射多个端口， depends\_on 指定依赖服务， nacos需要依赖mysql服务。

### 6.3 部署脚本

项目编写部署脚本，让项目能够一键化部署，脚本文件有copy.sh和bin.sh

(1) copy.sh

```
#!/bin/sh

usage() {
    echo "Usage: sh copy.sh"
    exit 1
}

echo "begin copy sql....."
cp -rf ..\db\*.sql ./mysql\db

echo "begin copy gateway....."
cp -rf ..\cjxyccloud-gateway\target\cjxyccloud-gateway.jar ./modules/gateway/jar/cjxyccloud-gateway.jar

echo "begin copy leaf....."
cp -rf ..\cjxyccloud-leaf\target\cjxyccloud-leaf.jar ./modules/leaf/jar/cjxyccloud-leaf.jar

echo "begin copy auth....."
cp -rf ..\cjxyccloud-auth\target\cjxyccloud-auth.jar ./modules/auth/jar/cjxyccloud-auth.jar

echo "begin copy social....."
cp -rf ..\cjxyccloud-social\target\cjxyccloud-social.jar ./modules/social/jar/cjxyccloud-social.jar

echo "begin copy user....."
cp -rf ..\cjxyccloud-user\target\cjxyccloud-user.jar ./modules/user/jar/cjxyccloud-user.jar

echo "begin copy website....."
cp -rf ..\cjxyccloud-website\target\cjxyccloud-website.jar ./modules/website/jar/cjxyccloud-website.jar
```

图6.4 copy.sh

该文件使用cp指令，对java服务打包后的jar包进行复制，将jar包从target文件夹中复制到docker模块文件夹下，方便后续jar上传服务器。

## (2) bin.sh

```
#!/bin/sh

usage() {
    echo "usage: ./bin.sh [base|modules|send {发送到服务器}|sshkey {配置ssh}|stop|rm]"
}

sshkey() {
    echo "配置ssh-key....."
    ssh-copy-id root@$1
}

send() {

    echo "发送docker到$1:/app/cjxyccloud....."
    scp -r ..\docker root@$1:~/app/cjxyccloud/
}

base(){
    docker-compose up -d cjxyccloud-mysql cjxyccloud-redis cjxyccloud-nacos
}

modules(){
    docker-compose up -d cjxyccloud-gateway cjxyccloud-leaf cjxyccloud-auth cjxyccloud-user cjxyccloud-website cjxycld
```

图6.5 bin.sh

文件主要编写四个方法，第一个方法为sshkey()方法，对目标服务器配置ssh访问权限。send()方法使用scp命令，将docker文件夹发送到目标服务器。base()方法使用docker-compose up指令，启动项目基础服务。modules()方法使用docker-compose up指令，启动项目模块服务，最终启动项目所有服务。

## 第七章 接口测试

采用postman对系统接口进测试，postman是一款支持http协议的接口调试与测试工具，通过输入相应接口的地址，输入测试值，查看输出是否与预计输出相同，如果相同则测试成功，如果不同则测试失败。

系统接如下：

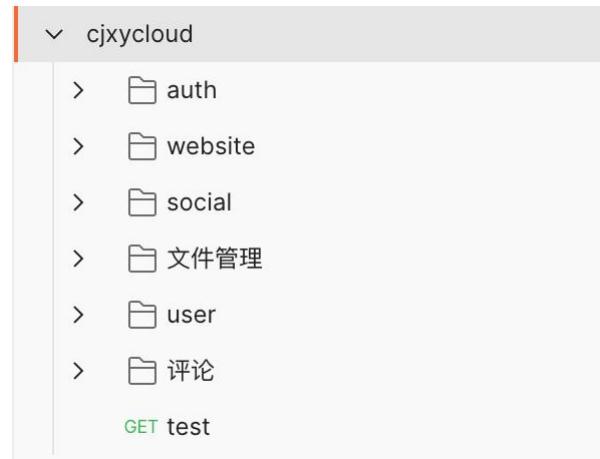


图7.1 接口

### 7.1 测试案例

登录接口测试：

```

POST {{url}}4231/login
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify
1 {"account": "1945829064",
2     "password": "(jiang.4234)"
3 }
4
    
```

图7.2 登录接口

输入测试地址 4231/login，输入测试值 account、password，然后发送进行测试，预计输出token  
查看输出结果：

```

{
  "code": 200,
  "msg": null,
  "data": {
    "access_token": "eyJhbGciOiJIUzUxMiJ9.eyJc2VtIj0KMTUxLWQwYzMtNDQxMi040WE4LWQ4YzzjY2NjNWRiZiJ9.
    BDgyb2H5pcAPyKNjtFYhOGS8TCZ8BILBND4nG1Ft0YZ-KcL2Avn1GycsRtdWHvqh35U0BLbw-R8N80bgmY0Q",
    "auth_password": "(jiang.4234)",
    "expires_in": 10080
  },
  "success": true
}

```

图7.23 输出结果

预计结果输出code200, data输出登录token, 输出与预计输出相同, 接口测试成功。

## 7.2 测试结果汇总

按照测试案例的方法进行测试, 对各个接口测试结果进行表格汇总, 结果如下

表 7.1 测试汇总

接口名称	输入	预计输出	实际输出	测试结果
用户登录	账号密码	code 200、加密token	code 200、加密token	正常
刷新Token	header携带token	code 200、加密token	code 200、加密token	正常
退出登录	header携带token	code200、data null	code200、data null	正常
我的课表	header携带token	code200、课表队列	code200、课表队列	正常
获取好友列表	header携带token	code200、好友列表	code200、好友列表	正常
添加好友申请	好友id	code200、data null	code200、data null	正常
查询好友申请列表	header携带token	code200、好友申请列表	code200、好友申请列表	正常
同意好友申请	好友id	code200、data null	code200、data null	正常
查询聊天记录	好友id	code200、聊天记录队列	code200、聊天记录队列	正常
获取校园圈	header 携带 token 、 pageNum、pageSize	code200、五条朋友圈	code200、五条朋友圈	正常
发表校园圈	正文、图片地址	code200、data null	code200、data null	正常

## 基于 JAVA 和 Flutter 的校园社交 APP

---

点赞	校园圈id	code200、data null	code200、data null	正常
文件上传	file、文件流	code200、文件存储id	code200、文件存储id	正常
获取个人信息	header携带token	code200、userinfo对象	code200、userinfo对象	正常
搜索用户	用户账号	code200、用户对象	code200、用户对象	正常
评论	校园圈id、正文、顶级评论id、次级评论id	code200、data null	code200、data null	正常
获取评论	校园圈id、顶级评论id、pageNum、pageSize	code200、顶级评论列表	code200、顶级评论列表	正常

---

经过测试比较，各个接口工作正常，总体测试通过。

## 结 论

在本篇论文中，主要探讨了项目开发的背景、目的，以及针对项目的背景对项目进行了分析以及系统设计，并对论文中的设计的功能，采用了当前主流的微服务技术进行服务功能的实现。

在开发中，后端采用JAVA微服务架构进行开发，但在开发过程中发现微服务在实际应用中需要考虑许多问题，存在着许多的问题，例如服务的设计和拆分、服务部署、服务监控等。通过对这些问题的分析与解决，提高了自己对微服务系统的理解，提高了自己的开发技术。

此外，项目采用C/S架构，前端使用Flutter架构进行开发，Flutter已经是一个很成熟的跨平台开发架构，通过对它的使用，让自己更加了解到了前端开发当中的许多问题，例如API接口对接，文件本地存储，消息更新机制等等。

通过此次项目开发，让自己更加全面的了解到了一个系统的开发流程，对于系统开发有了更多的经验，也对微服务技术有了更加深入的了解，对于我以后的工作也有很大的帮助，但因为项目技术开发经验不足，且由自己一个人开发完成，系统当中存在着许多的不足，谨请大家提出宝贵的意见。

## 参考文献

- [1] 懂超/胡炽维.SpringCloud微服务架构开发实战[M].北京: 电子工业出版社, 2018
- [2] 金华胡/胡书敏.基于Docker的Redis入门与实战[M].北京: 机械工业出版社, 2021
- [3] 杰伊 · 温格罗.数据结构与算法[M].北京:人民邮电出版社,2019
- [4] 章为忠.SpringBoot从入门到实战[M].北京: 机械工业出版社, 2021
- [5] 江荣波.Mybatis3源码深度分析[M].北京:清华大学出版社,2019
- [6] Silvia Botros / Jeremy Tinley.高性能MySQL (第4版) [M].北京: 电子工业出版社, 2022
- [7] 乔西 · 朗 / 肯尼 · 巴斯塔尼 .云原生Java[M].北京: 电子工业出版社, 2018
- [8] Simone Alessandria.Flutter Projects[M].Birmingham:Packt Publishing,2020
- [9] 奈吉尔 · 波尔顿.深入且出Docker[M].北京:人民邮电出版社,2019
- [10] 武内觉.Linux是怎样工作的[M].北京:人民邮电出版社,2022

## 致 谢

时光飞逝，大学的学习生活很快就要过去，在这四年的校园生活中，收获了很多，而这些成绩的取得是和一直关心帮助本人的人分不开的。在本篇论文中，自己要感谢许多人和组织，因为他们的支持和帮助使得这篇论文变得更加完整和精彩。

首先要感谢本人的指导教师，他们对本人进行了悉心的指导和建议，帮助自己更好的理解框架的核心概念和技术细节，并提供了对本人的论文结构和内容的有价值的反馈和建议。

并且非常感谢Spring Boot 开发团队和社区，他们不断地改进和完善这个框架，并且提供了大量的文档和示例代码，使自己能够更好地理解和掌握 Spring Boot 的技术细节和实践经验。

然后还要感谢本人的家人和朋友们，他们一直支持本人并在本人需要他们的时候给予本人力量和鼓励。感谢他们的支持和鼓励，自己才能够完成这篇论文。

最后，还要感谢学校和教育机构，他们提供了一个良好的学习环境和资源，并且提供了完成这篇论文的机会和支持。

谨以此篇论文，向所有支持和帮助过本人的人们致以最诚挚的感谢和祝福。

## 注解

<sup>i</sup> TransmittableThreadLocal是一个线程本地变量，它可以在不同线程之间传递和共享。和普通的ThreadLocal不同的是，TransmittableThreadLocal在异步调用和线程池等场景下，能够正确地传递和共享变量，避免了变量丢失或混乱的问题。

TransmittableThreadLocal是由阿里巴巴开发的，它扩展了InheritableThreadLocal并提供了更多的功能。在使用TransmittableThreadLocal时，如果需要将变量从一个线程传递到另一个线程，只需要将变量设置到TransmittableThreadLocal中，然后在另一个线程中读取该变量即可。

<sup>ii</sup> 自定义注解是一种可重复利用的元数据，可以应用于类、方法、字段等Java元素上，用于描述这些元素的特性、属性或行为等。自定义注解可以在编译时、运行时或者两者都生效的情况下，被Java虚拟机或者其他框架自动识别和使用。

<sup>iii</sup> Spring AOP是基于动态代理机制实现的一种面向切面编程的框架。它提供了一种方便的方式，可以在应用程序运行期间动态地添加或删除一些额外的行为，从而实现一些通用的功能，如事务管理、安全性检查、日志记录、性能监控等。Spring AOP通过拦截器链和代理对象来实现横向切面逻辑的调用。在AOP框架中，应用程序通常被分解为多个模块，每个模块都有自己的特定业务逻辑。当需要为这些模块添加额外的功能时，可以使用切面编程将这些功能切面化，从而对多个模块进行统一管理和维护。

<sup>iv</sup> Jsoup是一个用于处理HTML文档的Java库，它可以用于从HTML中提取数据、修改HTML内容、执行Web抓取等任务。使用Jsoup库，你可以轻松地从HTML中提取所需的数据。它提供了一组简单的API，可以根据标签、属性或文本内容来定位HTML元素，并从中提取数据。

<sup>v</sup> 雪花算法生成的ID是一个64位的整数，由时间戳、数据中心ID和机器ID组成，保证在分布式系统中不会出现重复的ID。