河 南 科 技 学 院

2020 届本科毕业论文（设计）

基于 SpringCloud 微服务架构的直播平台的设计与实现

学生姓名：高元明

所在学院：信息工程学院

所学专业：信息工程

导师姓名：潘灿林

完成时间：2020 年 5 月

摘 要

本文的主题是设计与实现一个基于SpringCloud+Golang微服务框架的分布式直播平台，主要聚焦于短视频与直播流媒体服务。系统由服务端，后台管理端和微信小程序端三部分组成，推拉直播流的协议分别为RTMP协议和HLS协议，直播服务器为livego。后端服务由Eureka注册中心,Gateway网关,和各种Gloang实现的微服务模块组成，Golang具有高性能、高并发、轻量等特点，服务端之间的相互调用基于HTTP协议，耦合性相对于RPC、gRPC协议较低。后台管理端使用Vue.js 框架，ui框架为Ant Design，用户端的微信小程序基于Vant-UI设计。用户端的所有API请求都需要经过Gateway网关统一拦截、鉴权、代理转发，网关在启动时从Eureka同步所有服务实例，同时提供负载均衡能力。

关键词: 流媒体直播，微服务，SpringCloud，Golang

ABSTRACT

The theme of this paper is to design and implement a distributed live platform based on spring cloud + golang micro service framework, which mainly focuses on short video and live streaming media services. The system consists of three parts: server, background management and wechat applet. The protocol of push-pull live stream is RTMP protocol and HLS protocol, and the live server is livego. The back-end service is composed of Eureka registry, gateway gateway and various micro service modules implemented by gloang. Golang has the characteristics of high performance, high concurrency and light weight. The mutual calls between servers are based on HTTP protocol, and the coupling is lower than RPC and grpc protocols. The background management end uses vue.js framework, the UI framework is ant design, and the wechat applet of the user end is designed based on vant UI. All API requests of client end need to be intercepted, authenticated and forwarded by gateway gateway. Gateway synchronizes all service instances from Eureka when it starts, and provides load balancing capability.

Keywords: Streaming Media Live; Micro Service; Spring Cloud; Golang

目　录

1 绪论

1.1 课题研究背景

现如今流媒体短视频行业发展迅猛，人们越来越趋向于对短视频和直播类型的娱乐消费。抖音，快手，微视等软件的快速崛起也证明了直播和短视频类型软件的地位在日渐上升。在全球市场上 TikTok 走出国门，迅速在全球市场积攒了大量用户。并且国家网信办网站在 3 月 22 日发布消息：国家互联网信息办公室、工业和信息化部、公安部、国家市场监督管理总局四部门联合发布《常见类型移动互联网应用程序必要个人信息范围规定》，自 2021 年 5 月 1 日起执行。进一步证明了国家方面对此类移动互联网软件重视，要求无需个人信息即可使用基本功能。因此基于微信小程序端的视频直播内容的 App，有着充分的优势，前期用户的积累无需绑定他们的隐私信息，例如手机、姓名、邮箱等等。平台用户的唯一标识为调用微信 wx.login 接口返回的 OpenID，通过微信的的开放接口获取，无需注册等繁琐的步骤，从这种角度上来说，所有的微信用户都可以扫码一键成为平台的用户，解决前期用户难注册、难留下的问题。

1.2 国内外研究现状

近年来随着互联网技术、5G、人工智能等的发展，还有互联网头部企业的涌入，为短视频媒体直播的发展提供了充分的条件，更加方便了内容的输出和用户的交互体验，直播+短视频的内容方式成为了当下的趋势。腾讯、百度、阿里相应推出了自己旗下的短视频直播 APP。根据艾媒咨数据中心的统计，预测中国在 2020 年结束，在线直播的用户数量将会到达 5.26 亿。国家与地方都制定了相应的政策予以支持，例如上海市经信委发布《上海市促进在线新经济发展行动方案》，鼓励直播电子商务，积极推动了行业的发展。在国外市场 TikTok、Kwai、Vmate、都有了自己的一席之地，未来全球在短视频直播的领域中，需求还会进一步的扩大。

1.3 系统实现目标

本次实现的目标为，设计实现一个后台管理、微信小程序、服务提供者三个主要模块的分布式直播系统。后台管理主要功能有视频拍摄的背景音乐管、视频内容的审核、直播资质的审核、直播间的管理，账号管理、邮件发送。微信小程序端，短视频方面提供拍摄、观看、上传三部分功能，用户可以根据自己的喜好关注和喜欢相应的视频和内容创作者，直播间主要有拉流观看和弹幕功能，用户

提出申请后，得到管理员审核通过后方可进行直播。服务提供者有注册中心、配置中心、网关、账户服务、内容上传服务、邮件服务、直播间服务、推拉流服务。因为推拉流需要的并发流量较大，所以为了减少网关压力，除了直播间的推拉流所有 HTTP 服务都通过网关访问。

2 微服务架构设计

2.1 微服务架构简介

微服务是由面向服务的架构 SOA 演变而来，是一种云原生的架构方法，整个应用程序由多个松耦合的服务组成，通常由业务模型划分出最小颗粒度的服务。传统的开发模式下应用程序高度耦合，一处功能的修改往往需要整个模块的重构，部分并发量高的接口可能会拖垮整个程序，虽然后期可以通过 Nginx 反向代理负载均衡，依然无法从根本上解决上述问题。而在微服务架构下每个服务有自己单独的数据库，它们可以互不依赖的单独部署上线，极大的方便了整个应用程序的迭代更新，减少了后期的运维成本。服务之间通过轻量级的协议 RPC 远程调用协议或者基于 HTTP 协议的 RESTFUL API 通信。但是微服务架构依然客观存着错误难以定位，开发周期较长，成本开销大等问题。在微服务架构中，一个服务的崩溃可能会导致所有后续服务的超时，产生雪崩效应使整个应用程序不可用，通过熔断器可以有效的减少这种情况发生的概率，对响应超时的服务熔断、隔离，从而使整个应用程序高可用。

2.2 SpringCloud 简介

SpringCloud 是目前 Java 方向较为活跃的微服务框架，集合了常用的微服务组件，它帮助开发人员快速构建出一个微服务架构的应用程序，使开发人员聚焦于业务领域。SpringCloud 主要由五个组件组成，的每个组件单元由 SpringBoot Application 组成。五大组件如下图 1 所示。

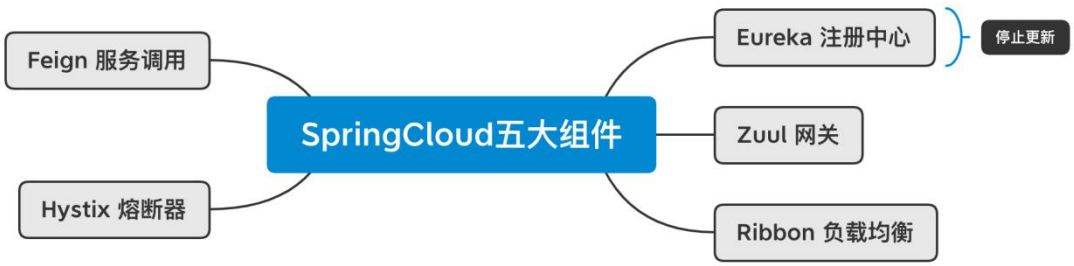

图 2-1 SpringCloud 五大组件图

2.3 开发环境简介

后端的开发环境为：Java 编辑器为 IntelliJ IDEA，是一款智能的 Java 编辑器，具有强大 DEBUG 断点调试功能，减少不必要的重复工作。Java 开发工具包 Oracle JDK11，依赖管理工具使用的是 Maven，关系型数据库 MySQL8.0，使用非关系型数据库 Redis 做外部缓存,提高系统的并发能力。多媒体处理工具 FFmpeg，对用户拍摄的 MP4 视频与 MP3 音频编码合成。推拉流服务器使用由 Go 语言编写的开源服务器 Livego，与 Nginx 服务器性能接近，并且配置相对容易，可以跨平台编译，运行非常方便。

前端开发环境为：Node.js 基于 Chrome V8 引擎的 JavasSript，版本为 14..16.1LTS 长期支持版、Vue2 和微信开发者工具。

3 系统概要设计

3.1 功能分析

项目主要有两种账户角色：管理员和用户，分别对应不相同的功能模块。管理员端的登录凭证为邮箱，有密码和邮箱验证码两种方式，登录后可进入管理员的网站页面，未登录的请求页面全部拦截重定向到登录页面。登录后的主要主要操作有系统监控方面，监控显示当前系统的各个服务实例健康状态。视频管理主要是对用户拍摄上传的视频进行内容审核，审核通过才可以在小程序端展示给所有用户，对审核通过的视频也可以撤销审核进行二次检查。合拍音乐分为上传和管理两部分，上传音乐需要填写对应的表单和上传音乐封面海报。直播间管理功能是对用户申请开设直播的审核，和对正在直播的直播间的内容审核。账户管理则为添加修改管理员账号，系统的功能概要图 3 如下所示：

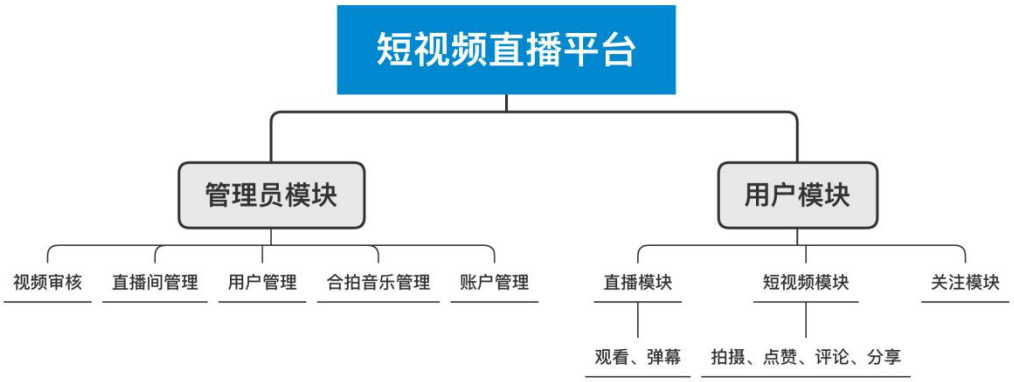

图 3-1 系统功能分析图

3.2 服务划分

根据系统的功能模块将系统抽象划分为邮件服务、上传服务、账户服务、直播间服务，服务 Service 间使用自定义的 Http Client 通信。每个服务只负责自己的独立部分，统一使用 Go WEB 框架 iris 对外提供 RESTFUL API 服务。邮件服务负责发送邮件和验证码两个主要功能。上传服务有视频上传、音频上传、图片上传、文件下载、合成 MP3 和 MP4。账户服务有管理员和用户两个主要模块，主要由账户主体的信息获取、登录、授权等。直播间服务主要为直播间的检索和直播服务器的推拉流。

3.3 系统架构实现

系统架构为 SpringCloud+Go 框架下的微服务架构，使用注册中心动态服务发现的模式。成功启动后等待被注册中心发现同步后，Gateway 网关会从 Eureka 注册中心拉取服务名和每个服务实例的 IP 与端口号，映射为自定义的路由，前端通过自定义的路由访问，统一进行鉴权。数据层的服务有 MySQL 和 Redis 两部分，使用 Redis Cache 的模式减轻数据库的压力。直播服务器流量较大所以不通过网关，并且拉流属于公共行为，推流需要密匙，所以拉流和推流直接开放接口。系统架构图如下图 4 所示：

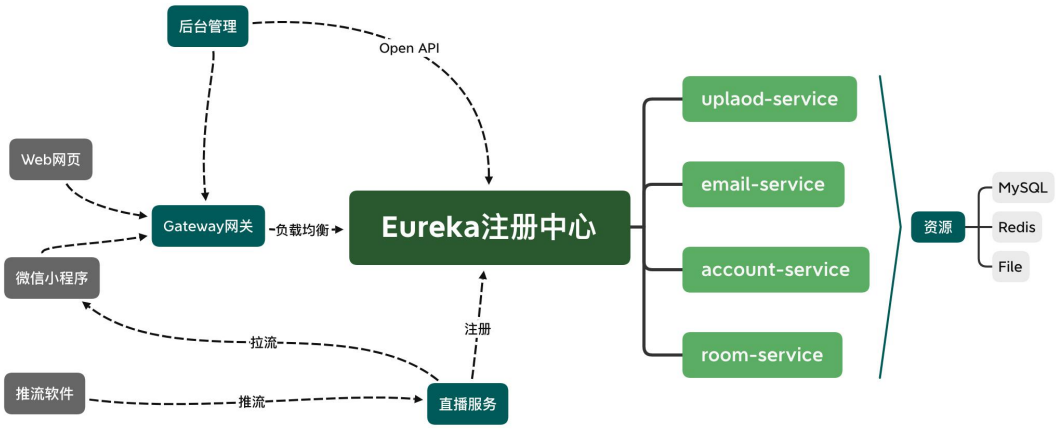

图 3-2 系统架构图

4 数据库设计

4.1 数据库需求分析

数据库的模型设计中，大部分的表结构之间存在相互关联，使用外建或者级连的方式虽然方便了业务模块的编写，但是对数据库端的并发性能影响较大，一个数据的操作可能使关联的多个表进行修改操作，外建和级连的数据库模型设计

更适合在单机情况下，并不适用于分布式体系下。因此表的关联全部封装在应用层，在各个 Service 的中处理关联逻辑。

首先用户信息的保存需要一张用户表，由于用户端只存在小程序端，主键 id 设置为微信的 openId。视频的上传观看需要视频表，用户在拍摄视频时可选择相应的音乐合拍，所以需要音乐表相关联，并且需要存入是否使用音乐的字段，用户在对视频进行点赞操作时首先需要判断用户是否对同一视频点过赞，不能单独在视频表中存入点赞数量统计，可通过视频点赞表与视频和用户两者关联。用户评论操作时需要评论表，评论表要存入视频主键 id。直播功能主要负责存储直播间的基本信息和拉流的地址。

4.2 表结构设计

（1）用户表 t_user 主要存放小程序端用户的基础信息，主键设置为微信的 openId，因为部分用户可能只进行简单的浏览，不需要获取个人信息，因此将用户的身份信息字段默认为空，在用户允许获取身份信息时，在后端执行 update 操作，更新用户表。

表 1 用户信息表 —— t_user

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
nickname	varchar (50)	DEFAULT NULL	用户的微信昵称
address	varchar (255)	DEFAULT NULL	用户的地址
has_info	tinyint	DEFAULT `0`	是否通过微信接口获取过用户信息
avatar_url	varchar (255)	DEFAULT NULL	用户头像地址
gender	varchar (1)	DEFAULT NULL	用户性别
follow_count	int	DEFAULT `0`	关注的用户数
fans_count	int	DEFAULT `0`	粉丝数量

（2）管理员信息表 t_admin 主要存放后台管理端管理员的基础信息，主键设置为随机生成的 uuid，整张表只包含主键、邮箱、密码三个字段，并且全部不可为空值。邮箱字段前端进行判断，必须要真实存在，主要用于管理员登录时接收邮箱验证码。

表 2 管理员信息表 —— t_admin

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
email	varchar (50)	NOT NULL	管理员账号邮箱
password	varchar (50)	NOT NULL	管理员账号密码

（3）短视频点赞关联表 t_video_like 主要存放用户对短视频点赞操作的记录，首先防止同一用户对一个视频多次点赞，其次用来统计视频的喜欢人数，表中的 user_id 对应用户的 id，video_id 对应视频的 id

表 3 点赞关联表 —— t_video_like

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
user_id	varchar (50)	NOT NULL	点赞用户 id
video_id	varchar (50)	NOT NULL	视频 id

（4）短视频的所有相关信息都存在 t_video 表中，因为系统中使用了为微服务架构,在查询所有短视频的场景下,每个视频信息都要查询相关的用户数据，但是用户信息不在 upload-service 服务的范围内，只能调用 account-service 服务，这样一来，假设每次服务调用耗时 100ms，获取 10 个视频就可能耗时 1s，这显然是不实际的。因此将前端需要展示的用户信息也存放在视频表中。

表 4 视频信息表 —— t_video

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
user_id	varchar (50)	NOT NULL	上传用户 id
user_nickname	varchar (50)	NOT NULL	上传用户昵称
user_avatar	varchar (50)	NOT NULL	上传用户头像
music_id	varchar (50)	DEFAULT NULL	合拍音乐 id
title	varchar (255)	DEFAULT NULL	视频标题
like_count	int	NOT NULL	点赞数

comment_count	int	NOT NULL	评论数
share_count	int	NOT NULL	分享数
poster	varchar (255)	NOT NULL	视频封面图
url	varchar (255)	NOT NULL	视频地址
use_music	tinyint	NOT NULL	是否使用音乐
music_name	varchar (50)	DEFAULT NULL	音乐名称
music_author	varchar (50)	DEFAULT NULL	音乐作者
music_poster	varchar (50)	DEFAULT NULL	音乐海报
music_url	varchar (50)	DEFAULT NULL	音乐地址
status	varchar (50)	NOT NULL	审核状态（审核 未审核）

（5）用户关注表 t_follow 类似于 t_video_like，主要关联用户的关注关系，from_user_id 为关注用户的 id，to_user_id 为被关注用户的 id。

表 5 关注信息表 —— t_follow

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
from_user_id	varchar (50)	NOT NULL	谁关注的
to_user_id	varchar (50)	NOT NULL	关注的谁

（6）短视频的评论存储在 t_comment 表，将部分前台需要展示的用户信息，冗余在其中，原因与上述视频表相同，同一个用户可对同一视频进行多次评论，与点赞模块不同。

表 6 关注信息表 —— t_comment

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
video_id	varchar (50)	NOT NULL	评论的视频 id
user_id	varchar (50)	NOT NULL	评论用户的 id

user_nickname	varchar (255)	NOT NULL	用户昵称
user_avatar	varchar (255)	NOT NULL	用户头像 url
content	varchar (500)	NOT NULL	评论内容
create_at	varchar (50)	NOT NULL	评论时间

（7）表 t_music 为用户在拍摄视频时的合拍音乐，只允许管理员端添加，表中的所有字段都不允许为空。

表 7 音乐信息表 —— t_music

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
name	varchar (255)	NOT NULL	音乐名字
author	varchar (255)	NOT NULL	作者
url	varchar (255)	NOT NULL	url 地址
poster	varchar (255)	NOT NULL	封面海报 url

（8）表 t_room 主要用于存放所有的直播间信息，为了避免其他用户的恶意推流，设置 token 字段，存放直播间的推流令牌，令牌生成使用随机的 uuid，在线人数保存用户下播后的最后在线人数。直播间的状态分为下播、正在直播、封禁、直播异常四种状态，四种状态的修改都由管理员操作。直播间的 url 地址为拉流地址，微信小程序端可根据 url 地址实时的观看直播内容。

表 8 直播间信息表 —— t_room

字段名	类型	是否为空	描述
id	varchar (50)	NOT NULL	主键 uuid
user_id	varchar (50)	NOT NULL	用户 id
token	varchar (255)	NOT NULL	推流令牌
count	varchar (255)	NOT NULL	在线人数
title	varchar (255)	NOT NULL	直播间标题
url	varchar (255)	NOT NULL	客户端拉流地址

status	int	NOT NULL	直播间状态
user_nickname	varchar (255)	NOT NULL	主播昵称
user_avatar	varchar (255)	NOT NULL	主播头像 url
user_gender	varchar (1)	NOT NULL	主播性别

5 系统设计与实现

5.1 微服务系统搭建

5.1.1 Eureka 环境搭建

Eureka 的运行需要依赖于 SpringBoot Application，在 maven 项目的 pom.xml 文件中添加依赖 spring-cloud-starter-netflix-eureka-server，版本为于父工程的 spring-boot-starter-parent 2.4.5 版本一致，在启动类 EurekaApplication 上添加@EnableEurekaServer 注解和@SpringBootApplication 注解启动注册中心，添加配置 defaultZone 为 http://localhost:8761/eureka/，应用程序启动成功后，打开浏览器访问控制台检查注册中心是否启动成功，Eureka 控制台地址为 http://localhost:8761，如下图 5 所示：

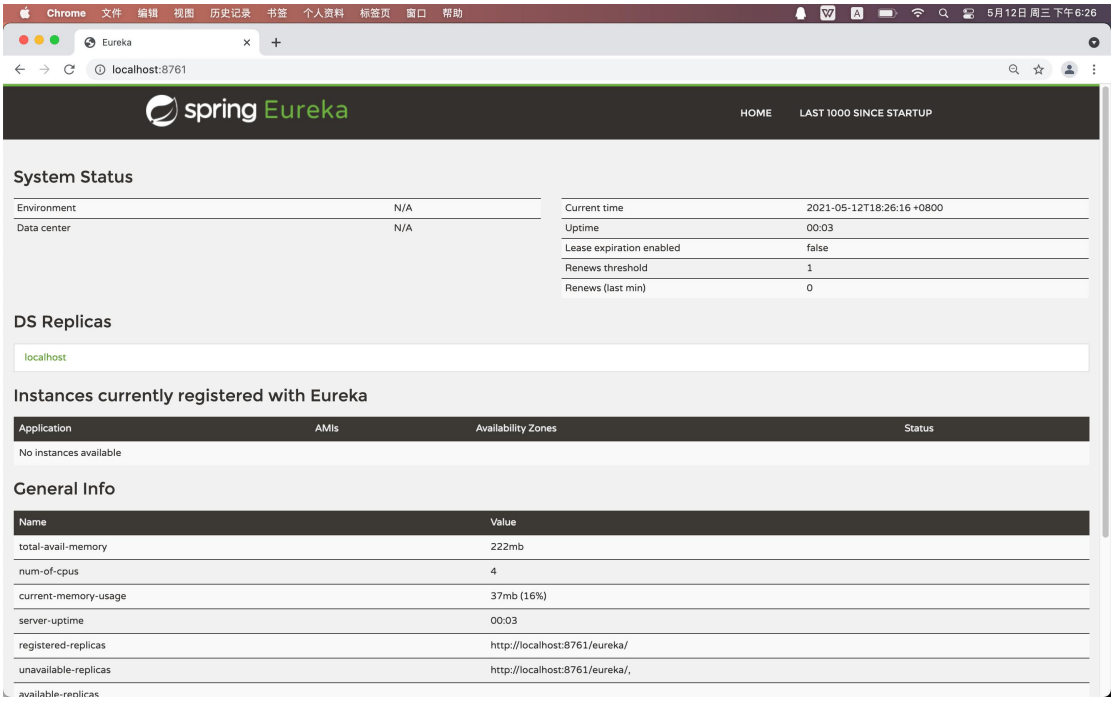

图 5-1 Eureka 控制台图

5.1.2 Gateway 网关搭建

项目中使用的网关为 SpringCloud Gateway 版本为 2.2.5 RELEASE，需要与 JDK 版本不冲突，网关的启动也需要依赖于@SpringBootApplication 启动，注入注解 @EnableDiscoveryClient，使网关也同步到 Eureka 注册中心，以此从配置中心拉取服务列表。

网关使用全局过滤器的方式获取每次的请求上下文，自定义的过滤器 GlobalResponseFilter 需要实现 GlobalFilter 和 Ordered 两个接口。网关通过配置文件 application.yaml 配置四个服务的映射地址，分别为 account-service、room-service、email-service、upload-service。网关的部分接口需要 web 端的请求，需要配置浏览器的跨域 CORS 策略，否则会导致 vue 端 axios 的请求失败，网关的配置文件如下图 6 所示：

```yaml
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowed-originPatterns: "*"
            allowed-methods: "*"
            allowed-headers: "*"
            allow-credentials: true
      discovery:
        locator:
          # 启用服务发现自动路由
          enabled: true
      routes:

        #用户服务路由
        - id: routes-account-service
          uri: lb://account-service
          predicates:
            - Path=/account-service/**

        #上传服务路由
        - id: routes-upload-service
          uri: lb://upload-service
          predicates:
```

图 5-2 Gateway application.yaml 配置图

5.1.3 服务注册

系统的服务实例通过 Eureka Open API 注册到 Eureka，使用开源的第三方库 eureka-client，实现的原理是新建 goroutine 定时请求 Eureka 的 REST API。首先注册应用的地址为 POST http://localhost:8761/eureka/apps/{appId}，Response 的编码方式为 JSON，注册成功后返回状态码 Status 204。服务注册成功后需要定时向 Eureka Server 发送心跳，发送超时 Eureka 会注销掉当前应用实例，发送心跳的地址为 PUT http://localhost:8761/eureka/apps/{appId}/{instanceId}，如果

请求成功返回 HTTP Status 200，如果当前 Eureka Server 缓存中不存在当前实例，则返回 HTTP Status 404。服务注册实现如下图 5-3 所示：

```go
package client

import ...

var eurekaClient *eureka.Client

func RegisterToEureka(eurekaAddr string, port int, appName string) {
    // create eureka client
    eurekaClient = eureka.NewClient(&eureka.Config{
        DefaultZone:           eurekaAddr,
        App:                   appName,
        Port:                  port,
        RenewalIntervalInSecs: 10,
        DurationInSecs:        30,
    })
    // start client, register、heartbeat、refresh
    eurekaClient.Start()
}
```

图 5-3 服务注册实现图

5.1.4 服务调用

Golang 服务调用其他服务的方式基于 HTTP 协议实现，服务调用函数 callServices 需要传入四个参数 serviceName（服务名）、method（请求的 HTTP 方法）、reqUrl（请求路由）、body（请求携带的数据），返回的两个参数本别为 byte 切片和 error 类型。服务调用首先从 Eureka 获取所有的应用实例，忽略大小写匹配服务，如果服务实例切片长度为 0 返回错误“服务不存在”，然后生成随机数索引，实现负载均衡，通过索引获取切片中的服务实例，实例 Instances 属性 HomePageURL 存放实例的 IP 地址，拼接出最终的请求路径，使用 golang 标准库 net/http 新建 http client 发起请求，使用 ioutil.ReadAll 读出响应数据给调用的客户端。

```go
package client

import ...

// callServices 服务调用
func callServices(serviceName string, method string, reqUrl string, body io.Reader) ([]byte, error) {
    var instances []eureka.Instance
    //获取服务实例
    for _, application := range eurekaClient.Applications.Applications {
        //eureka api返回的服务名为大写
        if strings.EqualFold(serviceName, application.Name) {
            instances = application.Instances
        }
    }
    if len(instances) == 0 { return nil, errors.New("The " + serviceName + " instance does not exist in eureka ") }
    //生成随机数 随机访问该服务下的所有实例
    serviceInstance := instances[doBalance(len(instances))]
    //此次请求的最终路径
    requestAddress := serviceInstance.HomePageURL + reqUrl
    //新建http request
    request, err := http.NewRequest(method, requestAddress, body)
    if err != nil { return nil, err }
    //新建http client
    httpClient := new(http.Client)
    response, err := httpClient.Do(request)
    if err != nil { return nil, err }
    bytes, err := ioutil.ReadAll(response.Body)
    if err != nil { return nil, err }
    return bytes, nil
}

func doBalance(len int) int {
    return rand.New(rand.NewSource(time.Now().UnixNano())).Intn(len)
}
```

图 5-4 服务调用实现图

5.2 账户服务

账户服务对外提供用户、管理员、关注信息 RESTFUL API 接口。用户接口主要有三个分别为根据 id 查询用户、更新用户信息、用户登录，管理员接口有密码登录、验证码登录、查询管理员，关注接口主要由添加关注、根据用户 id 获取关注列表。

5.2.1 用户登录

小程序端的用户登录流程为：小程序开始初始化时，会首先进入根目录下 app.js 中的 onLaunch 方法，调用微信 API “wx.login”。在调用成功的钩子函数中，发送微信服务器返回的 code 到 account-service 的登录接口，后台发起 HTTP 请求微信的 https://api.weixin.qq.com/sns/jscode2session 接口，需要携带三个参数分别为 appid、appSecret、code，接口相应内容中包含用户的 openid。openid 为微信用户的唯一标识，不同的微信账号对应不同的 openid。获取 openid 之后，查询数据库是否存在主键为此 openid 的用户，如果存在则接口返回用户信息给小程序端，否则插入一条用户数据，主键为 openid，同时插入字段 has_info 为 false，标志用户未统一授权获取微信的昵称、头像、地址等基本信息，在用户点击个人页面时，使用微信的 wx.getUserInfo 接口弹框提醒用户是否同意小程序获取个人信息，用户同意则根据 openid 执行 upadte 操作，同步用户的个人信息，并且 has_info 字段设置为 true，避免二次弹窗提升用户体验。用户登录的代码实现如下图 8 所示：

```java
/**
 * 用户登录实现
 * @param code 微信code
 * @return 用户信息
 * @throws IOException 异常
 */
@Override
public Account login(String code) throws IOException {
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
            .url("https://api.weixin.qq.com/sns/jscode2session?appid="
                    + appId + "&secret="
                    + appSecret + "&js_code="
                    + code + "&grant_type=authorization_code")
            .build();
    Response response = client.newCall(request).execute();
    if (response.isSuccessful()) {
        LoginDto dto = JSON.parseObject(Objects.requireNonNull(response.body()).byteStream(), LoginDto.class);
        //查询是否有当前用户
        Optional<Account> optional = accountRepository.findById(dto.getOpenId());
        if (optional.isPresent()) {
            return optional.get();
        } else {
            Account account = new Account();
            account.setId(dto.getOpenId());
            return accountRepository.save(account);
        }
    }
    return null;
}
```

图 5-4 用户登录代码实现图

5.2.2 管理员登录

管理员提供密码和验证码两种登录方式，密码登录的流程较为简单，核心步骤为根据表单判断邮箱和密码是否相同，根据错误类型抛出异常。邮箱验证码登录方式需要调用邮箱服务，服务调用使用 Feign 客户端的模式。前端首先要请求管理员验证码登录接口，为了避免发送无用验证码，需要查询是否存在该邮箱的管理员账号，不存在则返回身份验证失败异常，存在则调用 EmailFegin 的 checkEmailCode 方法。管理员登录 web 界面如下图 9 所示：

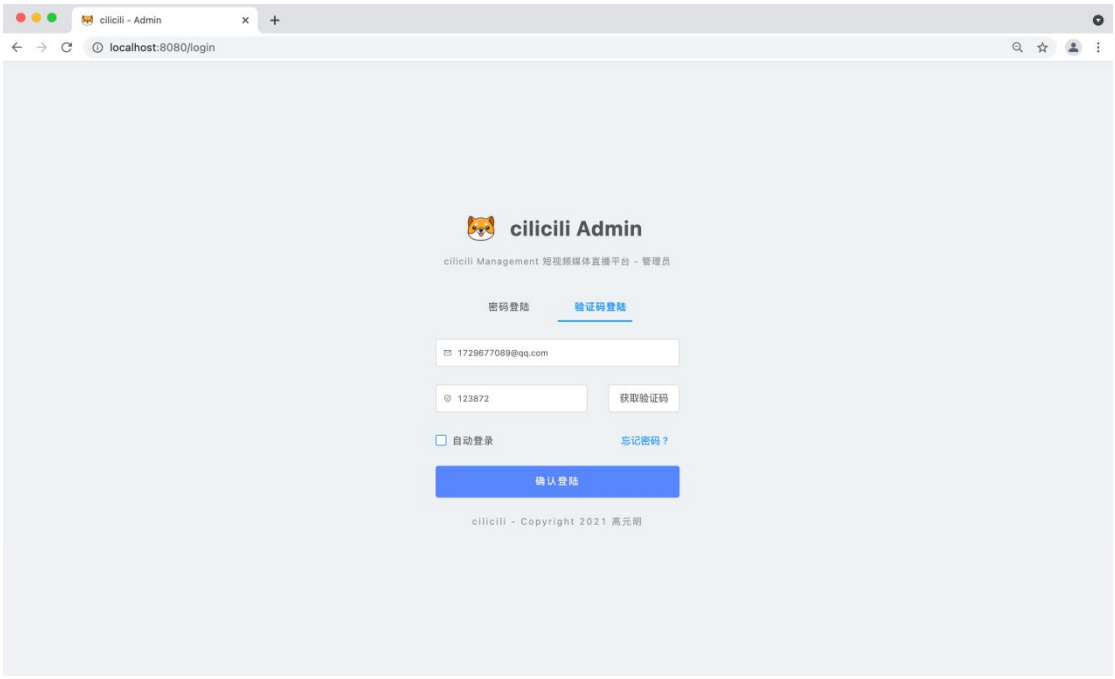

图 5-5 管理员登录代码实现图

5.3 邮箱服务

邮箱服务主要接口有发送一般邮件、发送验证码邮件、检验验证码三个。发送邮箱使用 smtp 协议，引入 Jar 包“spring-boot-starter-mail”，在配置文件 application.yaml 中配置发件人的邮箱和授权码。

验证码为 6 位随机数字，验证码存储的方案有 Redis 存储，MySQL 存储，内部缓存的方式。验证码不需要持久化，并且验证码的调用仅有管理员端，并发量小，使用 Redis 反而会提高复杂度舍本逐末，经过比较分析最终决定采用内部缓存的方式，使用 Java 原生的数据结构 ConcurrentHashMap<String, Code>哈希表存储验证码，既有一定的并发行也能保证线程安全。哈希表的 Key 为 String 类型的邮箱，key 的唯一性刚好满足了验证码与邮箱需要一一对应的要求，不会出现一个邮箱却存在两个或多个不同验证码的问题。Code 为验证码类，Code 拥有

两个属性 value 和 exp，分别对应验证码的值和过期时间戳。验证码功能的代码实现步骤如下图 10 所示：

```
/**
 * 发送验证码
 * @param to 收件人邮箱
 * @throws Exception 异常
 */
@Override
public void sendCode(String to) throws Exception {
    Code code = codeCache.get(to);
    if (code != null) {
        //验证码存在且未过期
        if (code.getExp() > System.currentTimeMillis())
            throw new Exception("请勿频繁发送验证码");
    }
    SimpleMailMessage m = new SimpleMailMessage();
    m.setFrom(from);
    m.setTo(to);
    m.setSubject("cilicili 验证码");
    String v = randomCode();
    m.setText("你的验证码为:  " + v + "  十分钟内有效");
    sender.send(m);
    codeCache.put(to, new Code(v,  exp: System.currentTimeMillis() + EXP));
}

@Override
public void checkCode(String email, String value) throws Exception {
    Code code = codeCache.get(email);
    if (code == null) throw new Exception("验证码错误，请仔细检查邮箱");
    if (code.getExp() < System.currentTimeMillis()) throw new Exception("验证码已过期，请重新发送");
    if (!code.getValue().equals(value)) throw new Exception("验证码错误，请仔细检查邮箱");
    //登录成功删除code
    codeCache.remove(email);
}
```

图 5-6 验证码功能代码实现图

5.4 上传服务

5.4.1 视频录制

（1）登录状态下的用户，进入“我的”页面，点击“发个视频”按钮，路由跳转到拍摄视频页。

（2）发布页面加载，进入生命周期函数 onLoad，通过微信 wx.getSetting 接口查看当前用户是否打开摄像头权限。

（3）如果未授予小程序摄像头权限，由于微信的隐私性要求必须使用按钮的绑定事件方式引导用户打开设置页面，无法通过页面加载的回调函数执行 wx.openSetting 方法，因此以弹出 model 框提示的方式引导用户是否进入设置页来开启摄像头权限，在 model 的确认按钮绑定是时间打开权限设置页面，如果用户点击 model 对话框的取消则退出视频拍摄页面。下次重新进入该界面依然重复上述步骤。

（4）获取到用户的摄像头权限后，<camera/>标签实时显示摄像头拍摄到的画面，使用 wx.createCameraContext 方法新建一个 cameraContext，拍摄按钮绑定事件调用 camearContext 的 startRecord 方法开始录像。

（5）用户点击结束后使用 stopRecord 方法结束录像，方法内的 success 回方法返回一个 response 对象，response 中包含本次录制过程的视频，视频第一

帧 jpg 图片的微信临时文件路径，使用 wx.uploadFile 方法将视频 MP4 文件和封面 jpg 格式的图片上传至 upload-service。

（6）临时文件上传成功后，跳转路由到发布视频的表单页，输入视频的标题，点击发布视频后，弹出对话框提示用户再次确认视频标题，点击确认后保存视频信息到 t_video 表。小程序端的页面实现如下图 11 所示：

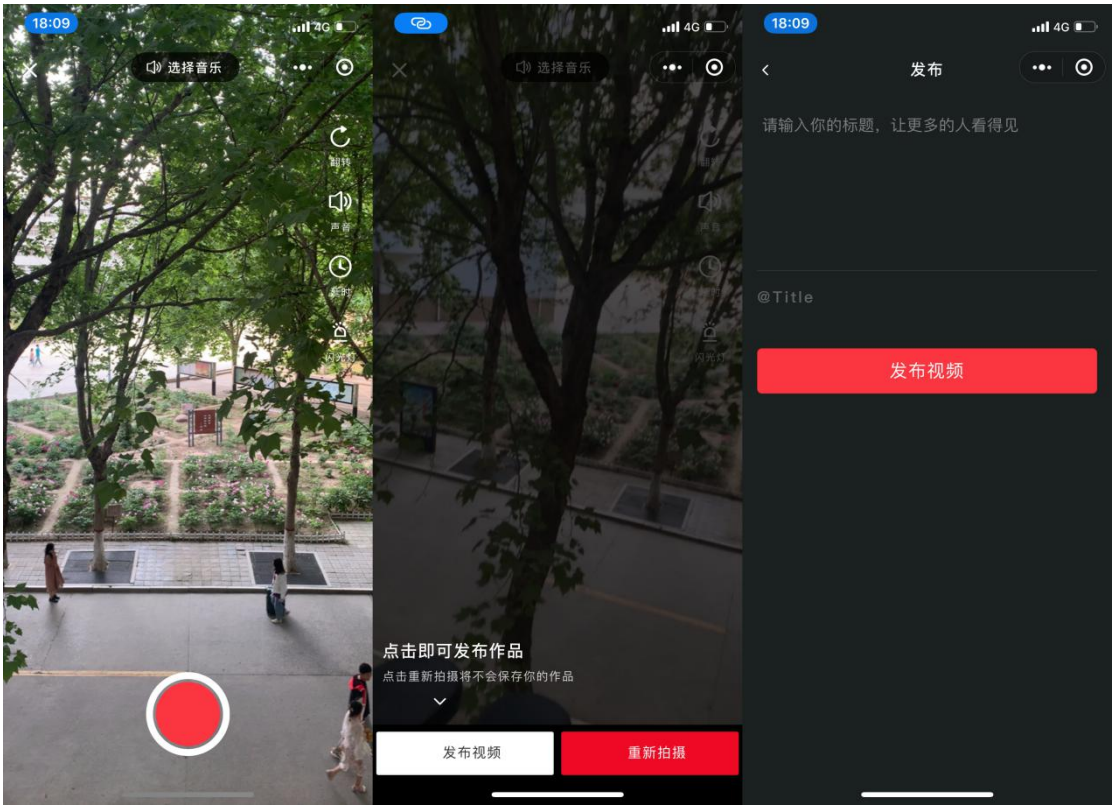

图 5-7 视频录制小程序界面实现图

5.4.2 视频合成

用户可在小程序端进行视频的录制上传，可选择视频的背景音乐，如果选择背景音乐将会扣除原视频的声音，并与选择的音乐合成。视频合成在后端部分完成，通过 ffmpeg 多媒体工具，在 Java 中调用 Runtime 包执行 Command 命令，ffmpeg 合成 MP4 和 MP3 代码实现如下图 12 所示：

```
//mp3合成mp4
if (video.isUseMusic()) {
    String id = UUID.randomUUID().toString();
    String mp3Path = temp + "/" + video.getMusicUrl();
    String mp4Path = temp + "/" + video.getUrl();
    String newPath = temp + "/" + id + ".mp4";
    String command = ffmpeg + " -i " + mp4Path + " -i " + mp3Path
            + " -c:v copy -shortest -c:a aac -strict experimental -map 0:v:0 -map 1:a:0 " + newPath;
    Process exec = Runtime.getRuntime().exec(command);
    exec.waitFor();
    video.setUrl(id + ".mp4");
}
```

图 5-8 ffmpeg 合成视频实现图

5.5 直播间服务

直播间的主要实现功能为客户端拉流和发送弹幕，用户需首先提出直播申请，等待管理员的审核通过。审核结果会以邮件的方式发送给申请用户，用户在拥有直播资质时,在 PC 端使用推流软件 OBS 推送直播流到 livego 直播服务器中，直播推流的密匙以动态的方式生成。用户每次开始直播时，服务器返回一个随机生成的 uuid，生成的 uuid 为本次直播的推流令牌。微信小程序的直播直播模块界面设计如下图 13 所示：

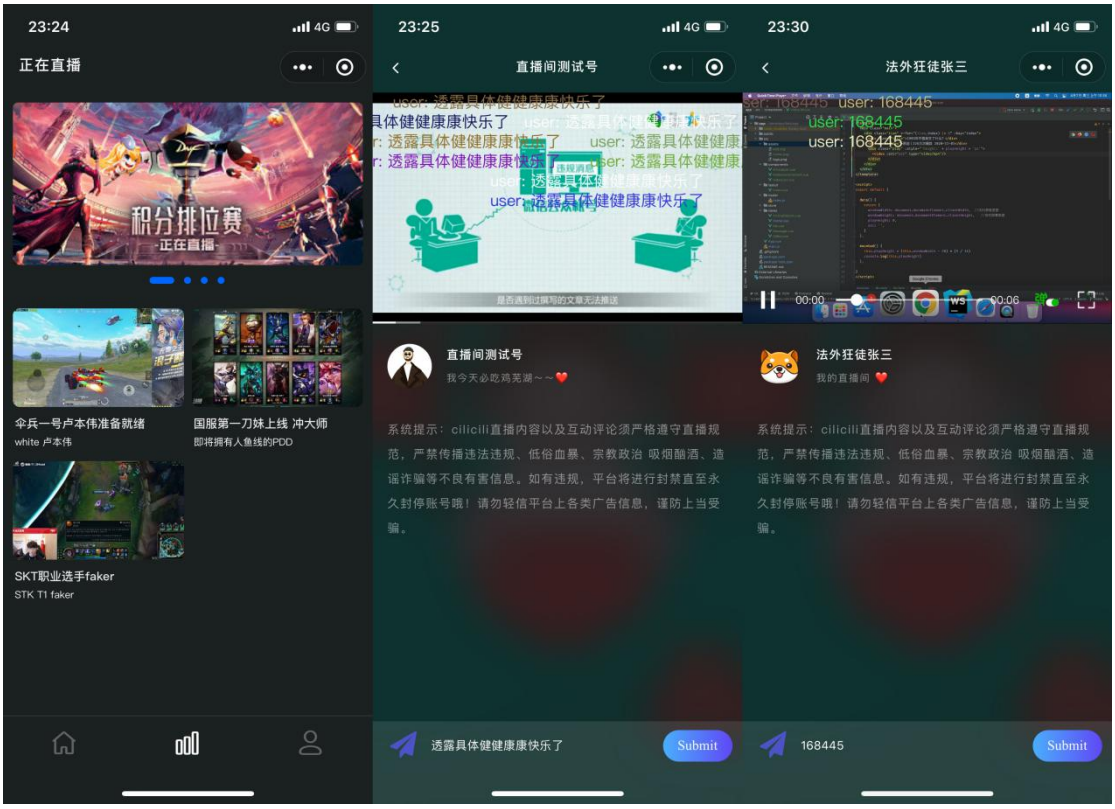

图 5-9 直播功能模块界面图

直播间的弹幕发送以 WebSocket 的方式发送，WebSocket 是定义在 TCP 协议之上的全双工通信协议，WebSocket 可与服务器保持长连接，允许客户端端主动发送消息，进行双向数据传输。

在用户进入直播间的时候，使用微信官方提供的 API wx.connectWebSocket 新建一个连接，根据直播间的 id 连接到对应的服务端。如果连接失败，在回调函数 wx.onSocketError 中返回给用户错误消息提示，引导用户重新进入直播间。弹幕发送的方法绑定在用户端“submit”按钮的事件上，判断输入值不为空后，使用 socket 向服务端发送消息内容，服务端根据房间 id 找到所有在同一房间 id 下的 socket clients，将消息转发出去。其他在同房间的微信小程序客户端使用 wx.onSocketMessage 来接受服务端的消息，接收成功后推送到 video 组件。

6 系统调试与优化

6.1 调试环境简介

首先在小程序权限管理端加入小程序的测试者，使用真机扫码调试，微信小程序基础库版本为 2.16.1，同时测试 iOS 和 Android 两种手机操作系统下的表现，在微信开发工具中实时预览真机控制台输出的日志信息。手机端与服务器端需要连接在同一 Wifi 网络下进行通信。

6.2 遇到的问题与解决

本次调试过程遇到的主要问题与解决方法如下：

（1）使用微信原生的播放组件 live-player 时，无法加载直播画面。经过阅读官方文档发现，该组件即使在本地测试不上线发布的条件下，依然需要有企业资质的 appid 才能调用接口。在阅读文档的同时，观察到 video 组件在 iOS 和 Android 下同时支持 m3u8 格式，因此意味着 video 组件可支持 HLS 协议下的直播流播放，经过真机测试验证，该方案可成功播放出直播流画面。

（2）短视频模块使用 swiper 组件，上下滑动为切换视频，水平滑动为切换推荐列表和关注列表。此方法下水平方向有两个 swiper-item，每个 item 中嵌套一个 swiper，导致整个视频页面滑动卡顿掉帧较为严重。因为小程序本身并非原生的 App，它运行在 Chrom v8 引擎上，性能与原生 App 差距较大，将决方法为推荐视频和关注视频单独出两个页面，一个页面仅包含一个 swiper 组件，允许垂直滑动禁止水平滑动，真机滑动性能提升明显，手机系统刷新率为 60 FPS 的条件下，上下快速滑动可保证在 53 FPS ~ 60 FPS 之间。优化前后小程序性能数据对比如下图 14 所示：

图 6-1 性能优化对比图

（3）在上传视频选择其中一首音乐后，后端上传服务报空指针异常，但是选择其他音乐时则无异常。证明异常的出现不在于后端，通过一点点调试，小程序端控制台打印数据，发现在使用歌曲作者名为“IU&李智恩”时，控制台出现运行错误，错误出现在 JSON.parse 方法上，带有特殊字符“&”会导致小程序端 js 代码解析 json 数据异常。解决方法为在管理员端上传音乐时，把 name 和 author 字段中的特殊字符过滤掉。

（4）在播放与 MP3 合成的视频时，手机端播放一段时间后视频有声音但是，画面静止，单独使用视频的 URL 在浏览器打开时，情况依然如此，发现视频长度与拍摄的时长不符的问题，原因在于 ffmpeg 默认以较长时间的 mp3 作为最终合成 mp4 的视频时长，导致原视频的音频被剪掉、并且小程序 video 组件的 controls 属性 controls 为 false，看不到播放条造成视频暂停的假象。在 ffmpeg 中加入命令“-shortest”属性解决问题。

7 结论

经过一段时间的努力，本次毕业设计“基于 SpringCloud 微服务架构的直播平台的设计与实现”已经完成，在此过程中认识到软件的设计首先要指定好方案，不能盲目的开始，良好的设计才能让之后的实现效率更高，但也不可过度设计。在代码实现的过程中，不放过一个问题，不能以得过且过的态度，每一个小的问题都可能导致日后一个难以寻找的 BUG，要脚踏实地一步一个脚印。在用户的交互界面中要注意多使用 Loading 动画，尽量避免页面闪动的情况，在部分点击事件中加入手机震动反馈、从用户的角度出发提升用户体验。在后端的实现中第一要务是保证数据的正确性，其次是保证性能，需要 io 操作耗时较长的地方尽量使用异步的方式，代码的编写最好有详细的注释。在数据库的设计中要注意字段的命名规范，统一使用下划线的方式分割，遵守数据库的设计范式。

参考文献

[1] Sam Newman.微服务设计[M].北京：人民邮电出版社.2016

[2] Chris Richardson.微服务架构设计模式[M].北京：机械工业出版社.2019

[3] 张峰.应用 SpringBoot 改变 web 开发模式[J].科技创新与应用.2017(23):193-194

[4] 王磊.微服务架构于实践[M].北京：电子工业出版社.2016

[5] 陈岩.轻量级响应式框架 Vue.js 应用分析[J].中国管理信息化.2018(3):181-183

[6] 朱二华.基于 Vue.js 的 Web 前端应用研究[J].科技与创新.2017(20):119-121

[7] 刘建宏.MySQL 数据库优化于集群[J].专题技术.2017(07):47

[8] 方宇荣.网络直播平台的现状于发展研究[D].北京：北京印刷学院.2017

[9] 孟艳华,罗仲伟,廖佳秋.网络直播内容价值感知与顾客契合[J].中国流通经济.2020(09):56-66

[10] 田志友,周元敏,田雨.微信小程序的媒体价值[J].新媒体研究.2018(01):47-49

[11] 吴高阳.基于 NIO 的 Java 高性能网络技术的研究与应用[D].西安：西安电子科技大学.2015

[12] 郭晶晶,刘光尧,汪磊,李志刚.FFmpeg 在视频图像处理中的应用[J].刑事技术.2020(03):234-247

[13] 李皓,杨成,刘剑波.一种基于 HLS 的安全直播方案[J].中国传媒大学学报.2018(04):42-47

[14] Biradar S M , Shekhar R , Reddy A P . Build Minimal Docker Container Using Golang[C]// 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). 2019.

[15] Wang C , Sun H , Xu Y , et al. Go-Sanitizer: Bug-Oriented Assertion Generation for Golang[C]// 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2020.

致谢

时光如梭光阴似箭，转眼间大学四年的时光就要结束了，在这四年中要感谢我的导师和同学们，老师们的淳淳教诲和同学们的关心照顾都帮助我成长了许多，导师潘灿林老师在大一的时候与我们交流了大学四年的规划问题和毕业设计的相关问题，认真谦虚的回答了每个人的问题，问了我们每个人的大学计划、职业规划等等，让我在接下来的四年有了一个自己的学习目标，对未来的规划有了清晰的认识。与同学们的交流相处也是我这四年来的大学生活充满着乐趣，在学习中遇到难懂的问题我们互相交流讨论，使原本枯燥的学习也变得生动起来。四年的大学生活让我更加的明白了学习是一辈子的事，做任何事情都要踏踏实实一步一个脚印。