广东职业技术学院

毕业综合实践报告

题 目: 基于Go语言实现的网络聊天系统

类 型:

专 业: 物联网应用技术

班 级: B物联网212

学生姓名: 陈文成

指导教师: 邓立

完成时间: 2024.3.3-2024.5.24

**摘 要**

随着互联网的普及，网络聊天室已成为人们日常交流的重要工具。网络聊天室在人群中具有不可忽视的重要性。

在当今社会，人们的生活节奏加快，面对面的交流变得越来越少。网络聊天室为人们提供了一个跨越地域、时间限制的交流平台，让人们能够随时随地与他人保持联系。无论是亲朋好友之间的日常问候，还是工作团队中的协作沟通，网络聊天室都发挥着不可或缺的作用。它不仅方便了人们的沟通，还促进了信息的传播和共享。因此，网络聊天室在人群中的重要性不容忽视，它已经成为现代生活中不可或缺的一部分。

本文介绍了在VSCode环境下开发一个网络聊天系统的过程。实现了用户的远程通讯的基本功能，例如用户的注册，用户的登陆,用户于用户文本聊天等。

目录

[第一章 系统需求分析 1](#_Toc162894873)

[1.1系统目标 1](#_Toc162894874)

[1.2系统可行性分析 1](#_Toc162894875)

[1.3方案论证 3](#_Toc162894876)

[第二章 程序的设计与实现 4](#_Toc162894877)

[2.1相关技术环境介绍 4](#_Toc162894878)

[2.2架构图 5](#_Toc162894879)

[2.3架构介绍 5](#_Toc162894880)

[2.4关键问题 6](#_Toc162894881)

[2.5程序功能的实现与描述 7](#_Toc162894882)

[2.5.1网页服务如何搭建 7](#_Toc162894883)

[2.5.2终端与用户服务的连接 8](#_Toc162894884)

[2.5.3用户服务与网页服务的连接 9](#_Toc162894885)

[2.5.4网页服务与GRPC服务的连接 12](#_Toc162894886)

[2.5.5用户服务中redis数据库的初始化与连接 14](#_Toc162894887)

[2.5.6用户服务redis手机查询用户的实现 15](#_Toc162894888)

[2.5.7网页服务中redis数据库的连接 16](#_Toc162894889)

[2.5.8 GRPC服务中mysql数据库的初始化与连接 17](#_Toc162894890)

[2.5.9 GRPC服务的Id查询用户的实现 20](#_Toc162894891)

[2.5.10 GRPC服务的手机查询用户的实现 21](#_Toc162894892)

[2.5.11密码加密与验证的实现与描述 22](#_Toc162894893)

[2.5.12 GRPC服务的创建用户的实现 24](#_Toc162894894)

[2.5.13短信功能的实现与描述 27](#_Toc162894895)

[2.5.14登录时验证码的实现 29](#_Toc162894896)

[2.5.15列出用户列表的实现 30](#_Toc162894897)

[2.5.16用户注册的实现 32](#_Toc162894898)

[2.5.17用户登陆的实现 34](#_Toc162894899)

[2.5.18用户聊天的实现 36](#_Toc162894900)

[2.6程序服务描述 38](#_Toc162894901)

[2.7客户端服务 39](#_Toc162894902)

[2.8聊天室服务端 39](#_Toc162894903)

[2.9网页服务端 40](#_Toc162894904)

[2.10 GRPC服务端 40](#_Toc162894905)

[2.11程序流程图 41](#_Toc162894906)

[2.12数据结构表 41](#_Toc162894907)

[第三章 用户使用说明与调试结果 42](#_Toc162894908)

[3.1使用说明 42](#_Toc162894909)

[3.2调试结果 43](#_Toc162894910)

[3.3优缺点分析 52](#_Toc162894911)

[第五章 致谢 53](#_Toc162894912)

[总 结 54](#_Toc162894913)

[参考文献 56](#_Toc162894914)

# 系统需求分析

## 1.1系统目标

我们开发的系统应该具有以下的目标:

1. 客户端与服务器的通讯稳定;
2. 添加功能简单方便;
3. 原始数据修改简单方便;
4. 简单的数据保存;

## 1.2系统可行性分析

1、技术可行性分析

由于本系统管理的对象单一,都是聊天室的用户,所以涉及的计算过程不是很复杂。现在学校用于学生管理的微机都是Pentium（R）4 CPU以上的机器，在存储量、速度方面都能满足各方面的要求。在技术难度方面,由于有指导老师的指导和相关参考文献，因此完全可以实现。

2、开发工具可行性分析

VSCode是一款非常流行的代码编辑器，它具有强大的功能和易用性，非常适合开发人员使用。对于开发工具的可行性分析，以下是一些关键因素：

(1)易用性：VSCode提供了丰富的插件和扩展，可以满足各种开发需求。对于初学者来说，它提供了友好的界面和直观的操作方式，使得学习编程变得更加容易。

(2)集成开发环境(IDE)功能：VSCode提供了完整的集成开发环境功能，包括代码高亮、自动补全、调试器、版本控制等。这些功能可以帮助开发人员更高效地编写代码。

(3)跨平台支持：VSCode支持Windows、Mac和Linux等多个平台，这意味着开发人员无需为每个平台安装不同的编辑器。

(4)性能：VSCode的性能在许多方面都得到了优化，包括启动速度、编辑速度和插件加载速度等。这对于需要频繁切换文件和进行大量编辑的开发人员来说非常重要。

(5)社区支持：VSCode拥有庞大的社区，用户可以从中获取帮助和支持。此外，VSCode的插件生态系统也非常丰富，这意味着开发人员可以找到满足自己需求的插件。

综上所述，VSCode是一款非常适合开发人员使用的工具。它具有易用性、全面的IDE功能、跨平台支持、优化的性能和庞大的社区支持等特点。因此，对于大多数开发人员来说，使用VSCode进行开发是完全可行的。

## 1.3方案论证

采用微服务架构,每个服务都是单独部署在服务器上,有单独的数据库。服务于服务之间使用网络进行交互。相对于单体服务来说易于扩展添加更多的功能或服务,但是微服务的成本较高.微服务开始部署比较复杂,单体服务开始部署比较简单。但是微服务部署后的开发难易程度不变,单体服务部署后的开发将会越来越困难且繁琐。

数据库的选择。Mysql是关系型数据库, 其数据存储在磁盘上，读写速度相对较慢。 Redis是一个内存数据库，其读写速度非常快。因为内存是磁盘的高速缓存。所以当存储使用频率较高、键值对的、数据小的数据时使用Redis。Mysql支持的数据结构类型较全,存储比较大的数据时使用。即存储用户的信息使用Mysq,存储手机号与验证码的键值对使用Redis。

# 程序的设计与实现

## 2.1相关技术环境介绍

* MySQL[1];数据库;
* Redis[2];非关系性数据库;
* WebSocket[3]; 一种网络通信协议，它允许在单个TCP连接上进行全双工通信;
* 微服务[4]; 微服务是一种软件架构风格，它将应用程序拆分成一系列小的、独立的服务;
* Gin[5];一个用Go语言编写的高性能HTTP Web框架。它提供了快速构建Web应用程序的能力，并且具有许多优秀的特性;
* ProtoBuf[6]; 是Google公司开发的一种数据序列化协议（类似于XML、JSON、YAML等）。它可以用于结构化数据串行化，或者说序列化。使用其内置命令自动生成proto对应Go语言的版本文件;
* GRPC; grpc是一个高性能、开源和通用的 RPC 框架，面向移动和 HTTP/2 设计。中文描述是远程过程调用，它可以实现客户端像调用本地服务（方法）一样调用服务器的服务(方法);
* Gorm; 是一个用Go语言编写的开源对象关系映射（ORM）库。它提供了一种方便、高效的方式来在Go应用程序中操作数据库。通过使用GORM，开发者可以使用Go语言的结构体（struct）来映射数据库中的表，并使用GORM提供的方法来进行数据库操作;
* 短信服务;使用阿里云的API短信服务;

系统开发平台:操作系统macOS 12.7.3 、Web服务器(Gin) 、VSCode 、go1.19.4 darwin/amd64、MySQL Ver 8.0.20;

系统运行环境:本系统可以安装到Linux、windows等系统上；可以使用的Web服务器；客户可以使用的浏览器等；

## 2.2架构图

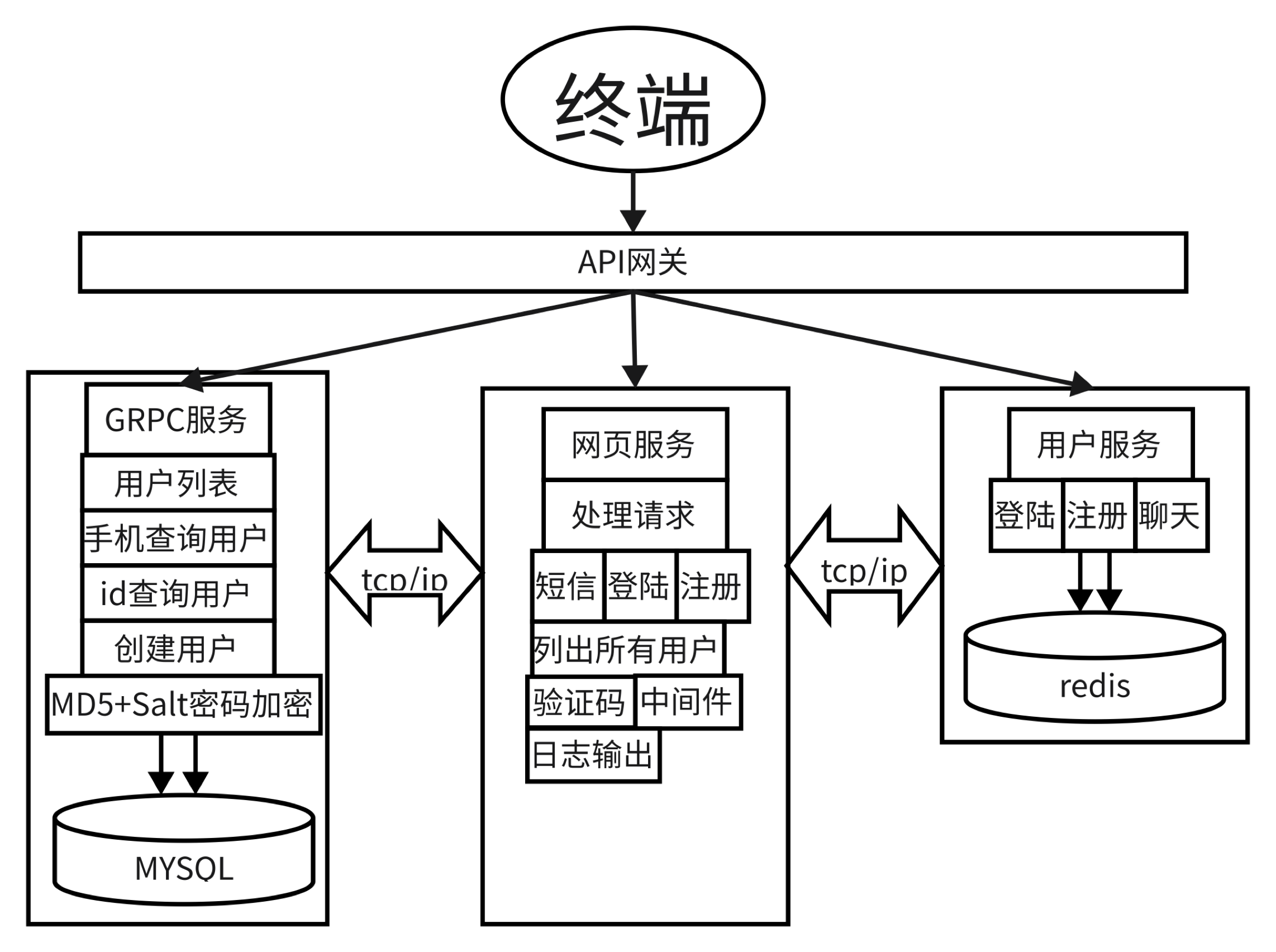


图2-1 架构图

## 2.3架构介绍

总体架构使用微服务(分布式)。微服务是一种软件架构风格，它将应用程序拆分成一系列小的、独立的服务。

## 2.4关键问题

1. 客户端与服务端的关系,web服务与GRPC服务的关系:

* 客户端是界面和数据结果的呈现,服务端是实现具体功能的呈现;
* 客户端和服务端使用Socket连接,底层是TCP/IP方式;
* Web服务相当于在客户端的层面与GRPC服务之间加了一层“界面”;
* 客户端向web服务发送请求,web服务连接GRPC服务将结果返回给web服务最后返回到客户端,客户端连接服务器,进行数据的发送与接收,从而实现聊天;

1. 前端界面如何实现:

前端界面使用Go语言内置的包fmt,实现向窗口打印输出文字,从而形成基础的界面;

1. 数据如何保存:

数据的保存使用mysql和redis数据库。mysql主要保存用户的数据,redis主要保存手机与验证码的键值对;

1. 如何对数据库进行操作:

使用GORM对Mysql进行数据操作。使用Go语言的结构体（struct）来映射数据库中的表，并使用GORM提供的方法来进行数据库操作;

1. 服务器如何实现对每一个用户的监听:

Go语言特点goroutine, go + 函数 实现对改函数起了一个协程(相当于轻量级的线程) 。使用for循环嵌套监听,每登录一个用户的连接,就起一个goroutine来维护;

## 2.5程序功能的实现与描述

## 2.5.1网页服务如何搭建

使用Gin框架实现,gin能快速搭建web服务。先进行注册路由,将域名与方法进行绑定注册进路由,然后使用Router.Run(ip:port)进行的监听,每当有相关的域名的请求过来时,相应注册的路由的域名就会响应改请求,具体函数的方法就会响应改请求。方法里面的返回请求都使用了JSON格式,当使用POST请求时请记得在请求响应头部中添加:Content-Type: application/json

本次web服务注册进路由的域名如下:

* <http://localhost:8021/u/v1/user/list> 用户列表
* [http://localhost:8021/u/v1/user/register](http://localhost:8021/u/v1/user/register/) 注册
* [http://localhost:8021/u/v1/user/pwd\_login](http://localhost:8021/u/v1/user/pwd_login%20//) 登陆
* [http://localhost:8021/u/v1/base/send\_sms](http://localhost:8021/u/v1/base/send_sms%20//) 发送短信
* [http://localhost:8021/u/v1/base/captcha](http://localhost:8021/u/v1/base/captcha%20//) 发送登陆验证码

通过路由注册的方式实现网页请求的功能。

具体注册路由和监听代码如下:

1. BaseRouter.GET("captcha", api.GetCaptcha)  //设置URL 获取验证码, 将方法注册进路由
2. BaseRouter.POST("send\_sms", api.SendSms) //设置URL 获取手机短信, 将方法注册进路由
3. { //设置URL以及中间件
4. UserRouter.GET("list",api.GetUserList)  输出所有用户,使用中间件并将方法注册进路由
5. UserRouter.POST("pwd\_login", api.PassWordLogin)  //登录逻辑, 将方法注册进路由
6. UserRouter.POST("register", api.Register)  //注册逻辑, 将方法注册进路由
7. }

## 2.5.2终端与用户服务的连接

每当有用户登陆时会对用户服务进行连接。如果出现错误会导致程序直接抛出异常,停止程序。

1.用户服务的监听:

使用go语言内置包net的net.Listen(tcp, ip:port)实现。

2.终端的连接:

使用go语言内置包net的net.Dial(tcp, ip:port)实现。

具体代码如下:

1. conn, err := net.Dial("tcp", "127.0.0.1:8889")
2. listen, err := net.Listen("tcp", "0.0.0.0:8889")

## 2.5.3用户服务与网页服务的连接

当用户服务进行登陆或者注册时向网页服务发出http请求时,网页服务就会对其进行响应。

通过GetChaptcha()实现获取验证码功能。

通过PostSms()实现发送手机验证功能。

用户服务具体发送http请求的方法如下:

1. // 通过post请求获取  // 获取数字验证码  // 返回值字符串
2. func GetChaptcha() (string, string) {
3. //实例一个request对象并配置请求方法,
4. req,\_ := http.NewRequest("GET", "http://localhost:8021/u/v1/base/captcha", nil)
5. rsp, \_ := http.DefaultClient.Do(req)   //使用默认client发送请求
6. //读取网页返回body的内容
7. body, \_ := io.ReadAll(rsp.Body)
8. //实例map对象接收body内容, interface{}可以接受任意类型的内容
9. rspJson := map[string]interface{}{}
10. //把body的内容写入rspjson
11. json.Unmarshal(body, &rspJson)
12. //返回string 并对interface{}断言为string
13. **return** rspJson["answer"].(string), rspJson["id"].(string)
14. }
15. // 发送手机短信,通过网页请求获取
16. func PostSms(userMobile string) error {
17. //发送信息的实例
18. var sendSmsInfo **struct** {
19. Mobile string
20. Type   uint
21. }
22. sendSmsInfo.Mobile = userMobile   //赋值手机号
23. sendSmsInfo.Type = uint(1)  //赋值type
24. data, \_ := json.Marshal(&sendSmsInfo)  //将实例进行json序列化
25. //实例一个request对象并配置请求方法,
26. req, \_ := http.NewRequest("POST", "http://127.0.0.1:8021/u/v1/base/send\_sms",
27. bytes.NewBuffer(data))
28. req.Header.Add("Content-Type", "application/json")  //Header添加Content-Type
29. rsp, \_ := http.DefaultClient.Do(req)  //使用默认client发送请求
30. //读取网页返回body的内容
31. body, \_ := io.ReadAll(rsp.Body)
32. defer rsp.Body.Close()
33. **return** nil
34. }
35. // 发送网页注册请求
36. //4.创建一个RegisterMes 结构体 并赋值
37. var registerMes message.RegisterMes
38. registerMes.UserMobile = userMobile
39. registerMes.UserPwd = userPwd
40. registerMes.Code = code
41. //将passWordLoginForm json化
42. data, \_ := json.Marshal(&isterMes)
43. //实例一个req的Request对象并配置好请求格式和发送的内容
44. req, \_ := http.NewRequest("POST", "http://localhost:8021/u/v1/user/register",
45. bytes.NewBuffer(data))
46. //请求使用 application/json
47. req.Header.Add("Content-Type", "application/json")
48. //发送Post请求进行登陆
49. res, \_ := http.DefaultClient.Do(req)
50. //读取body的内容
51. body, \_ := io.ReadAll(res.Body)
52. //请求状态不是OK 就登陆失败的逻辑
53. defer res.Body.Close()

## 2.5.4网页服务与GRPC服务的连接

网页服务成功响应注册,域名下注册的路由的函数方法就会创建对GRPC服务的连接。

GRPC服务下注册的方法如下:

1. 获取用户列表

GetUserList(context.Context, \*PageInfo) (\*UserListResponse, error)

1. 手机号查询用户

GetUserByMobile(context.Context, \*MobileRequest) (\*UserInfoResponse, error)

1. Id查询用户

GetUserById(context.Context, \*IdRequest) (\*UserInfoResponse, error)

1. 创建用户

CreateUser(context.Context, \*CreatUserInfo) (\*UserInfoResponse, error)

1. 检查密码是否正确

CheckPassword(context.Context, \*PassWordCheckInfo) (\*CheckResponse, error)

具体的代码如下:

1. //拨号连接用户grpc服务
2. userConn, err := grpc.Dial(fmt.Sprintf("%s:%d",
3. global.ServerConfig.UserSrvInfo.Host,
4. global.ServerConfig.UserSrvInfo.Port),
5. grpc.WithTransportCredentials(insecure.NewCredentials()))
6. }
7. defer userConn.Close()
8. //生成grpc的client并调用接口
9. userSrvClient := proto.NewUserClient(userConn)

连接成功后使用具体远程调用的方法代码如下:

1. //查询用户是否存在
2. userRsp, err := userSrvClient.GetUserByMobile(global.Ctx, &proto.MobileRequest{
3. Mobile: passWordLoginForm.Mobile,
4. })
5. //创建用户
6. user, err := userSrvClient.CreateUser(global.Ctx, &proto.CreatUserInfo{
7. NickName: registerForm.Mobile,
8. PassWord: registerForm.PassWord,
9. Mobile:   registerForm.Mobile,
10. })
11. //密码是否正确
12. passRsp, \_ := userSrvClient.CheckPassword(global.Ctx, &proto.PassWordCheckInfo{
13. Password:          passWordLoginForm.PassWord,
14. EncryptedPassword: userRsp.PassWord,
15. })

## 2.5.5用户服务中redis数据库的初始化与连接

1.初始化redis连接池并连接, 连接池的工作原理是预先创建一组数据库连接，并将它们保存在一个池中。当需要访问数据库时，它会从连接池中获取一个空闲的连接，而不是创建一个新的连接。当应用程序完成对数据库的操作后，它会将连接返回到连接池中，而不是关闭它。这样，连接就可以被其他需要访问数据库的操作重用, 接池可以减少创建和关闭连接的开销。

初始化redis连接池和连接代码如下:

1. func initPool(address string, maxIdle, maxActice **int**, idleTimeout time.Duration) {
2. pool = &redis.Pool{
3. MaxIdle:     maxIdle,     // 最大空闲链接数
4. MaxActive:   maxActice,   //表示和数据库的最大链接数， 0 表示没有限制
5. IdleTimeout: idleTimeout, //最大空闲时间,
6. Dial: func() (redis.Conn, error) { //初始化链接的代码， 连接到redis
7. **return** redis.Dial("tcp", address)
8. },
9. }
10. }

## 2.5.6用户服务redis手机查询用户的实现

当用户注册或者登陆时会使用到此方法判定手机号是否已经在数据库中存在。

参数是终端与服务连接的对象和手机号。手机号不存在返回空对象和错误。否则将查询到json化的res对象,将其反序列化成user并返回。

通过getUserByMobile实现手机查询用户功能。

手机查询用户的实现的代码如下:

1. // 1.根据用户ID返回一个User实例 + err
2. func (u \*UserDao) getUserByMobile(conn redis.Conn, mobile string) (user \*User, err error) {
3. res, err := redis.String(conn.Do("HGet", "users", mobile))  通过给定的id去ridis查询这个用户
4. user = &User{}
5. err = json.Unmarshal([]byte(res), user) 这里我们需要把res反序列化成一个user实例
6. **return**
7. }

## 2.5.7网页服务中redis数据库的连接

发送手机短信功能成功时,网页服务将连接redis保存手机号与短信验证码的唯一性在redis。确保短信的验证码与用户的验证码是唯一的。

通过NewClient()方法实现连接redis。

网页服务连接redis的实现代码如下:

1. //连接ridis 保存验证码
2. rdb := redis.NewClient(&redis.Options{
3. Addr: fmt.Sprintf("%s:%d",
4. global.ServerConfig.RedisInfo.Host, global.ServerConfig.RedisInfo.Port),
5. })

## 2.5.8 GRPC服务中mysql数据库的初始化与连接

1.用户列表、手机查询用户、Id查询用户、创建用户、更新用户等功能时会使用连接数据库的全局变量db。

Mysql数据库连接的实现的代码如下:

1. // init函数启动时自动运行
2. func init() {
3. //连接数据库的格式user:pwd@tcp(locahost:port)/tableName?charset=utf8mb4&parseTime=True&loc=Local
4. dsn := "root:00000000@tcp(127.0.0.1:3306)/chatroom\_user\_srv?charset=utf8mb4&parseTime=True&loc=Local"
5. //创建logger实例, 用来配置终端的输出样式
6. newLogger := logger.New(
7. log.New(os.Stdout, "\r\n", log.LstdFlags), // io writer
8. logger.Config{
9. SlowThreshold:             time.Second, // Slow SQL threshold
10. LogLevel:                  logger.Info, // Log level
11. IgnoreRecordNotFoundError: **true**,Ignore ErrRecordNotFound error for logger
12. ParameterizedQueries:      **true**,// Don't include params in the SQL log
13. Colorful:                  **true**,        // Disable color
14. },
15. )
16. var err error
17. //创建db连接mysql的实例, 以及加入logger的配置
18. DB, err = gorm.Open(mysql.Open(dsn), &gorm.Config{
19. NamingStrategy: schema.NamingStrategy{
20. SingularTable: **true**, //自动生成的名字是否需要复数
21. },
22. Logger: newLogger,
23. })
24. }

2.GRPC服务使用第三包gorm连接数据库.gorm屏蔽sql细节。可以自动对struct(对象)与数据库中的 Table进行字段与属性的映射,不用直接SQL编码。

初始化mysql数据库,第一次使用时需要创建Table,gorm可以实现将go语言的struct转化成mysql中的表,struct中字段里的tag与mysql中创建表时的数据类型、是否唯一、是否可以为空、注释等对应。

Mysql数据库的初始化实现的代码如下:

1. type BaseModel **struct** {  // 表的基本结构
2. ID        int32          `gorm:"primarykey"`
3. CreatedAt time.Time      `gorm:"column:add\_time"`    //创建的时间
4. UpdatedAt time.Time      `goem:"column:update\_time"` //更新的时间
5. IsDelete  **bool**           //是否物理删除
6. }
7. type User **struct** {  // 用户表结构
8. BaseModel   //索引;唯一;数据类型char(11);不为空;
9. Mobile   string     `gorm:"index:idx\_mobile;unique;type:varchar(11);not null"`
10. Password string     `gorm:"not null;type:varchar(100)"`  不为空
11. NickName string     `gorm:"type:varchar(20)"`  数据类型为char
12. Brithday \*time.Time `gorm:"type:datetime"`  数据类型为mysql的datatime
13. Gender   string     `gorm:"default:male;column:gender;type:varchar(6) comment 'female为女性 male为男性'"`  //默认值:男;mysql里的列名gender;数据类型char(6);后面注释;
14. Role     **int**        `gorm:"column:role;default:1;type:int comment '2表示普通用户, 1表示管理员'"`  // mysql里的列名role;默认值1;注释: 2表示普通用户, 1表示管理员;
15. }

连接的代码最后添加上 db.AutoMigrate(&model.User{})即可。

## 2.5.9 GRPC服务的Id查询用户的实现

使用gorm内置函数db.First()查询,返回的数据为0说明mysql里没有记录。

传入记录的数据类型和需要匹配的id。

实现的代码如下:

1. // 初始化User实例
2. var user model.User
3. //db调用查询函数,获取所有用户并赋值给users
4. result := global.DB.First(&user, req.Id)

## 2.5.10 GRPC服务的手机查询用户的实现

Gorm内置函数。使用where条件查询,传入具体具体实例的字段进行匹配。

具体代码实现如下;

1. // 初始化User实例
2. var user model.User
3. //db调用条件查询函数,获取所有用户并赋值给users
4. result := global.DB.Where(&model.User{Mobile: req.Mobile}).First(&user)

## 2.5.11密码加密与验证的实现与描述

MD5（Message Digest Algorithm 5）是一种被广泛使用的密码哈希函数。不可被解密。一个数值生成固定长度对应的哈希值。一种字符对应一种哈希值,字符出现微小的变化都会导致哈希值的变化很大,利用其雪崩特性的特点。

使用GO语言的第三方库go-password-encoder实现。注册时需要将用户输入的密码加密,登陆时需要用户输入的密码进行验证。

* 加密: 加密时加入salt和原密码一起MD5加密生成encodePassword,将salt和encodePassword连接成一段字符串.用作我们验证密码的约定。

实现加密的代码如下:

1. //配置md5加密的设置 盐值长度16 迭代次数100 密码哈希长度
2. options := &password.Options{SaltLen: 16, Iterations: 100, KeyLen: 32,
3. HashFunction: sha512.New}
4. salt, encodedPwd := password.Encode(req.PassWord, options)
5. user.Password = fmt.Sprintf("$pbkdf2-sha512$%s$%s", salt, encodedPwd)

* 验证: 调用数据库的密码加密后的哈希值与输入的密码进行验证。得到比较改密码与salt加密后是否和数据库里也就是原来的密码加密后的哈希值匹配。若匹配,则说明密码正确,否则密码错误。

实现验证的代码如下:

1. // 检查密码是否正确
2. func (s \*UserServer) CheckPassword(ctx context.Context, req \*proto.PassWordCheckInfo) (\*proto.CheckResponse, error) {
3. //获取之前的加密设置
4. options := &password.Options{SaltLen: 16, Iterations: 100, KeyLen: 32, HashFunction: sha512.New}
5. // 将字符串通过自己里面的"$"分割成字符数组
6. passwordInfo := strings.Split(req.EncryptedPassword, "$")
7. //取出盐值 密码的哈希key ,验证密码是否匹配
8. check := password.Verify(req.Password, passwordInfo[2], passwordInfo[3], options)
9. **return** &proto.CheckResponse{Success: check}, nil
10. }

## 2.5.12 GRPC服务的创建用户的实现

使用GRPC服务的手机查询用户。

用户注册时使用。先查询是否存在该用户,存在就将错误则通过grpc返回给web服务,web服务以JSON类型将错误返回给终端。否则将web服务连接进来并将CreatUserInfo struct数据类型的参数对新的User实例赋值,密码使用密码加密的实现与描述,接着使用gorm内置的db.Create()对mysql的添加记录操作,最后返回UserInfoResponse struct数据类型(具体的用户信息)给web服务。

具体实现代码如下:

1. type UserInfoResponse **struct** {
2. protobuf里的数据类型varint,占位1,转换时自动id,版本规则使用proto3
3. Id       int32  `protobuf:"varint,1,opt,name=id,proto3" json:"id,omitempty"`
4. PassWord string `protobuf:"bytes,2,opt,name=passWord,proto3" json:"passWord,omitempty"`   protobuf里的数据类型bytes,占位2,转换时自动passWord,版本规则使用proto3
5. Mobile   string `protobuf:"bytes,3,opt,name=mobile,proto3" json:"mobile,omitempty"`   protobuf里的数据类型bytes,占位3,转换时自动mobile,版本规则使用proto3
6. NickName string `protobuf:"bytes,4,opt,name=nickName,proto3" json:"nickName,omitempty"`   protobuf里的数据类型bytes,占位4,转换时自动nickName,版本规则使用proto3
7. BirthDay uint64 `protobuf:"varint,5,opt,name=birthDay,proto3" json:"birthDay,omitempty"`   protobuf里的数据类型varint,占位5,转换时自动birthDay,版本规则使用proto3
8. Gender   string `protobuf:"bytes,6,opt,name=gender,proto3" json:"gender,omitempty"`   protobuf里的数据类型bytes,占位6,转换时自动gender,版本规则使用proto3
9. Role     int32  `protobuf:"varint,7,opt,name=role,proto3" json:"role,omitempty"`   protobuf里的数据类型varint,占位7,转换时自动role,版本规则使用proto3
10. }
11. type CreatUserInfo **struct** {
12. NickName string `protobuf:"bytes,1,opt,name=nickName,proto3" json:"nickName,omitempty"`  protobuf里的数据类型bytes,占位1,转换时自动nickName,版本规则使用proto3
13. PassWord string `protobuf:"bytes,2,opt,name=passWord,proto3" json:"passWord,omitempty"`  protobuf里的数据类型bytes,占位2,转换时自动passWord,版本规则使用proto3
14. Mobile   string `protobuf:"bytes,3,opt,name=mobile,proto3" json:"mobile,omitempty"`  protobuf里的数据类型bytes,占位3,转换时自动mobile,版本规则使用proto3
15. }
16. // 创建用户
17. func (s \*UserServer) CreateUser(ctx context.Context, req \*proto.CreatUserInfo) (\*proto.UserInfoResponse, error) {
18. // 初始化User实例
19. var user model.User
20. //db调用查询函数,获取所有用户并赋值给users
21. result := global.DB.Where(&model.User{Mobile: req.Mobile}).First(&user)
22. //返回数据为1代表已有该用户存在
23. //把网页服务器端请求的数据赋值
24. user.NickName = req.NickName
25. user.Mobile = req.Mobile
26. user.Password = req.PassWord
27. //密码加密
28. //配置md5加密的设置 盐值长度16 迭代次数100 密码哈希长度
29. options := &password.Options{SaltLen: 16, Iterations: 100, KeyLen: 32, HashFunction: sha512.New}
30. salt, encodedPwd := password.Encode(req.PassWord, options)
31. user.Password = fmt.Sprintf("$pbkdf2-sha512$%s$%s", salt, encodedPwd)
32. //db调用创建函数
33. result = global.DB.Create(&user)
34. userInfoRep := ModelToResponse(user)
35. **return** userInfoRep, nil
36. }

## 2.5.13短信功能的实现与描述

使用阿里云的短信服务Go版本的第三方库实现。登陆阿里云官网,登陆账号设置accessKeyId 和 accessKeySecret 的值。需要在阿里云的网站里填写手机号码才能发送短信,因为申请到的模版时测试阶段的。具体界面如下图:

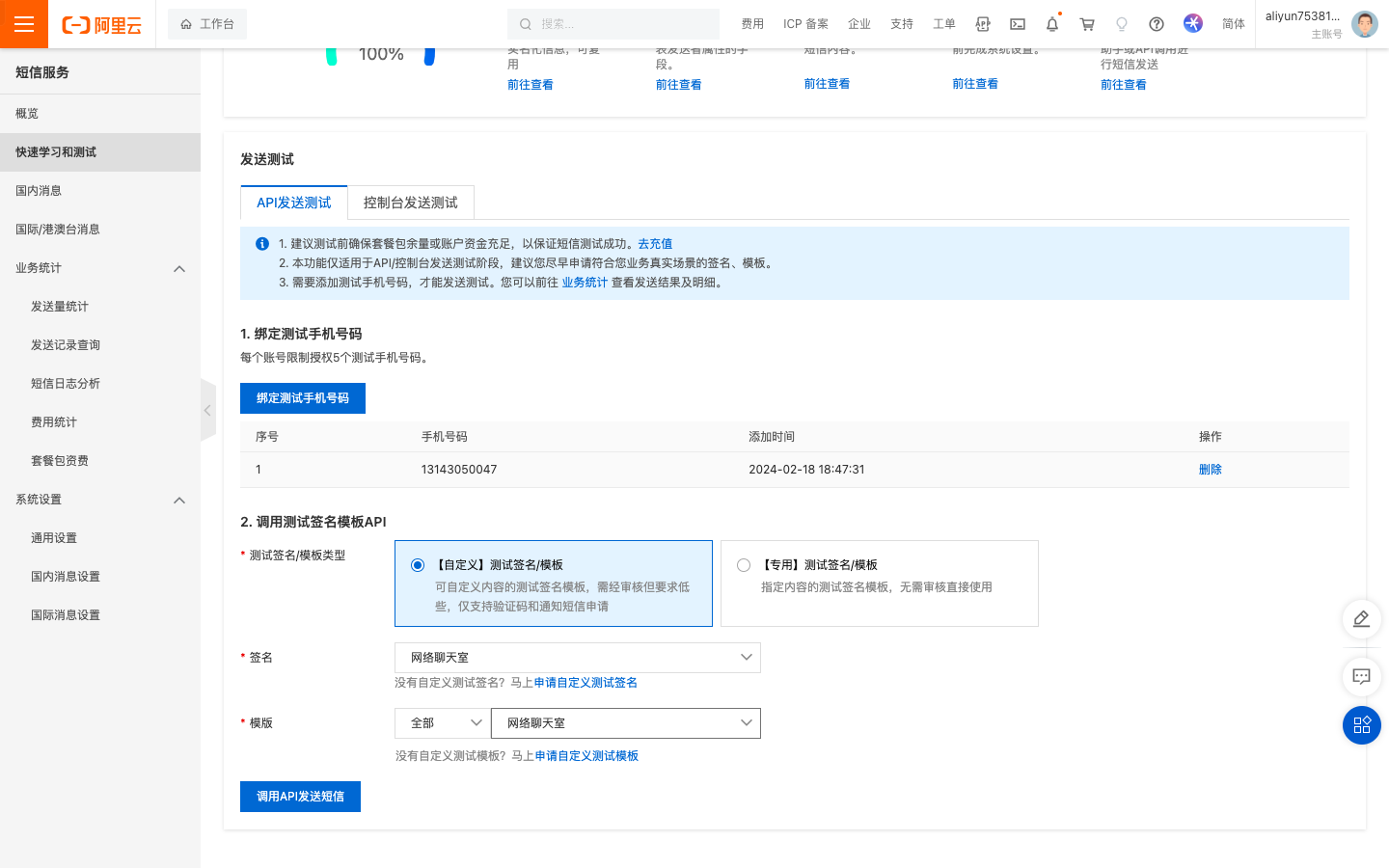


图2-2 阿里云短信服务网站图

具体实现代码如下:

1. func CreateClient(accessKeyId \*string, accessKeySecret \*string) (\_result \*dysmsapi20170525.Client, \_err error) {
2. config := &openapi.Config{
3. AccessKeyId:     accessKeyId,
4. AccessKeySecret: accessKeySecret,
5. Endpoint:        tea.String("dysmsapi.aliyuncs.com"),
6. }
7. // Endpoint 请参考 https://api.aliyun.com/product/Dysmsapi
8. \_result, \_err = dysmsapi20170525.NewClient(config)
9. **return** \_result, \_err
10. }  //实例化一个客户端，从 &dysmsapi.Client 类生成对象 client
11. client, err := CreateClient(tea.String(global.ServerConfig.AliSmsInfo.ApiKey),
12. tea.String(global.ServerConfig.AliSmsInfo.ApiSecret))
13. rand.Seed(time.Now().UnixNano()) //每次重置初始值
14. smsCode := fmt.Sprintf("%04d", rand.Intn(10000)) //10000内的随机数,不够4位数补0
15. smsCodeFormat := fmt.Sprintf("{\"code\":%s}", smsCode)
16. sendSmsRequest := &dysmsapi20170525.SendSmsRequest{  创建对应 API 的 Request
17. SignName:     tea.String("网络聊天室"),
18. TemplateCode: tea.String("SMS\_465375633"),
19. PhoneNumbers: tea.String(sendSmsForm.Mobile),
20. }
21. sendSmsRequest.TemplateParam = tea.String(smsCodeFormat)  //设置动态随机验证码
22. runtime := &util.RuntimeOptions{}
23. runtime.SetReadTimeout(10000)   //  读取超时
24. runtime.SetConnectTimeout(5000) // 连接超时
25. runtime.SetAutoretry(**true**)      // 是否自动重试

## 2.5.14登录时验证码的实现

登录时需要输入账号和密码,最后输入验证码才能点击登陆的需求。防止机器人重复登陆,浪费服务器的资源。使用Go语言的第三方库base64Captcha实现。

具体实现代码如下:

1. var store = base64Captcha.DefaultMemStore  // 实例一个base64编码的store
2. func GetCaptcha(ctx \*gin.Context) {  // 获取验证码
3. // driver := base64Captcha.NewDriverDigit(64, 240, 5, 0.7, 80)
4. driver := base64Captcha.DefaultDriverDigit使用默认的设置 长 宽 高 扭曲因子 数字验证码
5. c := base64Captcha.NewCaptcha(driver, store)  // 实例一个验证码对象
6. //生成一个验证码 返回 id b64s编码的图像 数字验证码
7. id, b64s, answer, err := c.Generate()
8. **if** err != nil {
9. ctx.JSON(http.StatusInternalServerError, gin.H{
10. "msg": err.Error(),
11. })
12. **return**
13. }
14. ctx.JSON(http.StatusOK, gin.H{      //响应返回
15. "id":     id,
16. "b64s":   b64s,
17. "answer": answer,
18. })
19. }

## 2.5.15列出用户列表的实现

管理员用户登录,用户在管理员界面使用。

首先向域名<http://localhost:8021/u/v1/user/list>发出GET的http请求,web服务响应请求,流程控制进入注册路由的方法func GetUserList(ctx \*gin.Context),建立对GRPC服务的连接,若连接失败,web服务返回JSON化的错误数据给终端。否则,流程控制进入GRPC服务注册的方法GetUserList里,具体逻辑先使用全局变量db(连接mysql的对象)调用内置Find函数查询mysql里所有的记录,若返回的result出现错误,逻辑就将错误返回给web服务,web服务返回给终端。否则将用户数组的用户类型使用protobuf化的数据返回给web服务,web服务接收自动反序列化。

最后将数据Json化返回给客户端,客户端将数据进行反序列化通过for循环打印在终端上。

具体实现的代码如下:

1. // 输出网页服务器里所有用户的信息
2. func (smsProc \*SmsProcess) GetUserList(userMobile string) {
3. //实例一个request对象并配置请求方法,
5. req, \_ := http.NewRequest("GET", "http://localhost:8021/u/v1/user/list", nil)
6. //Header添加Content-Type
7. req.Header.Add("Content-Type", "application/json")
8. req.Header.Add("x-token", global.MobileAndToken[userMobile])
9. //使用默认client发送请求
10. rsp, \_ := http.DefaultClient.Do(req)
11. //读取网页返回body的内容
12. body, \_ := io.ReadAll(rsp.Body)
13. //创建用户信息的实例
14. var users []**struct** {
15. Id     **int**    `json:"id"`     //数据库的id
16. Name   string `json:"name"`   //昵称
17. Mobile string `json:"mobile"` //手机号
18. }
19. //把body的内容写入rspjson
20. json.Unmarshal(body, &users)
21. //打印用户信息
22. fmt.Println("用户列表如下:")
23. **for** \_, user := range users {
24. fmt.Println("id:", user.Id)
25. fmt.Println("mobile:", user.Mobile)
26. fmt.Println()
27. }
28. }

## 2.5.16用户注册的实现

用户在终端注册.在终端输入手机号和密码后。

首先先对用户服务,进行连接.若连接失败,打印错误。否则,进行发送短信的确认,若确认则向web服务发送POST的http请求[http://localhost:8021/u/v1/base/send\_sms](http://localhost:8021/u/v1/base/send_sms%20//), 流程控制进入注册路由的方法func SendSms(ctx \*gin.Context),表单验证不通过就返回具体的错误,例如手机号不等于11位、手机号格式错误等.否则进行短信服务,紧接着把生成的验证码于手机号的键值对保存在redis里。

用户在手机上收到短信并在终端上输入验证码,后向web服务发送POST的http请求[http://localhost:8021/u/v1/user/register](http://localhost:8021/u/v1/user/register/), 流程控制进入注册路由的方法Register(c \*gin.Context),表单验证不通过就返回具体的错误,例如密码太短,密码太长等。web服务连接redis数据库对用户表单里的验证码进行匹配,若失败,则提示错误,将错误返回给终端,否则web服务对GRPC服务进行连接,使用GRPC上注册的方法CreateUser进行用户的创建.创建完后数据返回给web服务,web服务响应回终端。

终端紧接着对用户服务传输用户终端输入的数据,使用Json的协议。具体流程如下:

* 1. 发送消息的数据类型是一个struct,字段data保存json化后的字符串,字段type保存具体发送信息的类型,例如注册的信息类型、登陆的信息类型、聊天的信息类型、用户上线的信息类型、用户下线的信息类型等;
  2. 将发送的具体消息json化后获取其长度并赋值data(一层序列化), 接着把struct JSON序列化(二层序列化) ;
  3. 先发送长度给服务器.服务器接收到长度后将长度保存,接着阻塞等待读取;
  4. 接着将序列化的struct发送给服务器,服务器将其反序列化获得struct;
  5. 根据struct里的type数据类型进行登录逻辑、聊天逻辑、用户上线等逻辑;

用户服务接收到信息将第二层信息反序列化得到有用户信息的struct,type的值是RegisterMesType,switch选择注册的逻辑,将第一层信息反序列化得到实际用户输入的手机号和密码,用户服务从连接池里抽取一个连接出来连接redis的对象,查询redis里面是否存在,存在则提示用户已存在并返回终端,否则将手机号和密码等信息再次json序列化,保存进redis。

最后将保存成功的信息json化发送回客户端,客户端接收到后反序列化得到改请求的状态。

通过ReadPkg()方法实现客户端向服务端接收数据的功能。

通过WritePkg()方法实现客户端向服务端发送数据的功能。

客户端发送和服务器接收信息的具体代码如下:

1. // 最终获得将对方发过来的信息保存在相应结构体中
2. func (trans \*Transfer) ReadPkg() (mes message.Message, err error) {
3. fmt.Println("读取客户端的数据。。。")
4. //conn.Read() 在conn没有被关闭的情况下 ，才会阻塞
5. //如果客户端关闭了 conn 则 ， 就不会阻塞
6. \_, err = trans.Conn.Read(trans.Buf[:4]) // 返回值int是读了多少字节(Len),然后把读的数据赋给buf[0:4]; 读不到会一直等
7. //根据buf[:4] 转成一个uint32类型
8. pkgLen := binary.BigEndian.Uint32(trans.Buf[0:4]) //把buf的值转换为uint32
9. //根据pkgLen 读取消息内容
10. n, err := trans.Conn.Read(trans.Buf[:pkgLen]) //先获取切片的长度让后根据长度读的数据赋值给切片
11. //把pkgLen 反序列化成 -> message.Message
12. err = json.Unmarshal(trans.Buf[:pkgLen], &mes) //返回值的时候已经定义了; 把buf里读取的内容反序列化后的值赋给mes指向的结构体
13. **return**
14. }
15. func (trans \*Transfer) WritePkg(date []byte) (err error) {
16. pkgLen := uint32(len(date))    //先发送一个长度给对方
17. binary.BigEndian.PutUint32(trans.Buf[0:4], pkgLen) //把pkgLen的值转成[]byte类型的序列 //相当于 buf[0:4] = []byte(pkgLen)
18. //发送长度
19. n, err := trans.Conn.Write(trans.Buf[0:4]) //发的是buf存储pkgLen值的切片
20. n, err = trans.Conn.Write(date) // 发送date  发的是buf存储pkgLen值的切片
21. **return**
22. }

## 2.5.17用户登陆的实现

用户输入手机号和密码登录,最后输入验证码,回车登录.终端连接到用户服务。

向web服务发送POST的http请求[http://localhost:8021/u/v1/user/pwd\_login](http://localhost:8021/u/v1/user/pwd_login%20//), 流程控制进入注册路由的方法PassWordLogin(c \*gin.Context),表单验证不通过就返回具体的错误,例如手机号不等于11位、手机号格式错误等。否则web服务对GRPC服务进行连接,使用GRPC上注册的方法GetUserByMobile进行用户手机号查询。若用不存在则返回错误的数据给终端。否则进行登录时密码的验证。若验证失败代表手机号于密码不正确。

将用户的手机号和密码进行struct化进行两层的json序列化发送给用户服务, 用户服务接收到信息将第二层信息反序列化得到有用户信息的struct,type的值是LoginResMes选择登录的逻辑,将第一层信息反序列化得到实际用户输入的手机号和密码,用户服务从连接池里抽取一个连接出来连接redis的对象,查询redis里面是否存在。若用不存在则返回错误的数据给终端,否则提示其他在线用户我上线了。

最后将保存成功的信息json化发送回客户端,客户端接收到后反序列化得到改请求的状态。

通过ServiceProcessMes()方法实现确认消息类型为进行下一步的逻辑做铺垫。

用户信息对消息类型逻辑选择的实现代码如下:

1. // 功能： 根据客户端发送的消息的种类不同决定调用哪个函数来处理
2. func (proc \*Processor) ServiceProcessMes(mes \*message.Message) (err error) {
3. **switch** mes.Type {
4. **case** message.LoginMesType:
5. //处理登陆的逻辑
6. up := &process2.UserProcess{
7. Conn: proc.Conn,
8. }
9. err = up.ServerProcessLogin(mes)
10. **case** message.RegisterMesType:
11. //处理注册
12. up := &process2.UserProcess{
13. Conn: proc.Conn,
14. }
15. err = up.ServerProcessRegister(mes)
16. **case** message.SmsMesType:
17. smsProcess := &process2.SmsProcess{}
18. smsProcess.SendGropMes(mes)
19. **case** message.ExitMesType:
20. userStatusProc := &process2.UserStatusProc{}
21. userStatusProc.UserExitFunc(mes)
22. **default**:
23. fmt.Println("消息类型不存在, 无法处理!")
24. }
25. }

## 2.5.18用户聊天的实现

用户登陆成功,选择聊天.使用for循环实现每输入一次字符串回车发送。使用goroutine实现客户端对服务端端消息的阻塞接收。

用户的聊天内容字符串进行struct化进行两层的json序列化发送给用户服务, 用户服务接收到信息将第二层信息反序列化得到有用户信息的struct,type的值是SmsMes选择群发消息的逻辑,将第一层信息反序列化得到实际用户聊天内容字符串。

使用for循环对所有在线用户的连接进行遍历,将手机号和自己相同的过滤,也就是跳过将消息发给自己。用户服务将聊天内容进行两层json序列化发送给用户(终端), 用户(终端) 接收到信息将第二层信息反序列化得到有用户信息的struct,type的值是SmsMes选择群发消息的逻辑,将接收到的信息进行打印。

用户服务实现群发消息的代码如下:

1. **for** mobile, up := range userMgr.onlineUsers {
2. //这时我们要过滤掉自己， 就是不发给自己
3. **if** mobile == smsMes.UserMobile {
4. **continue**
5. }
6. smsProc.SendMesToEachOnlineUser(date, up.Conn)
7. }

用户(终端)实现接收群发消息的逻辑和打印信息代码如下:

1. //如果读取到了消息， 又是下一步处理逻辑
2. //根据消息的类型进行不同的逻辑
3. **switch** mes.Type {
4. // 有人上线了
5. **case** message.NotifyUserStatusMesType:
6. //1. 取出.NotifyUserStatusMes
7. var notifyUserStatusMes message.NotifyUserStatusMes
8. json.Unmarshal([]byte(mes.Date), ¬ifyUserStatusMes)
9. //2. 把这个用户信息， 状态保存到客户端map[string]User中
10. updateUserStatus(¬ifyUserStatusMes, mobile)
11. // 有人群发消息
12. **case** message.SmsMesType:
13. outputGroupMes(&mes)
14. //有人下线了
15. **case** message.ExitMesType:
16. //1. 取出exitMes
17. var exitMes message.ExitMes
18. json.Unmarshal([]byte(mes.Date), &exitMes)
19. //2. 把这个用户信息， 状态删除到客户端map[string]User中
20. UserExitFunc(exitMes.Mobile)
21. **default**:
22. fmt.Println("服务器返回了未知的消息类型")
23. }
24. //显示信息
25. info := fmt.Sprintf("用户%s:\t%s \n:%s", smsMes.UserName, smsMes.UserMobile, smsMes.Content)
26. fmt.Println(info)
27. fmt.Println()

## 2.6程序服务描述

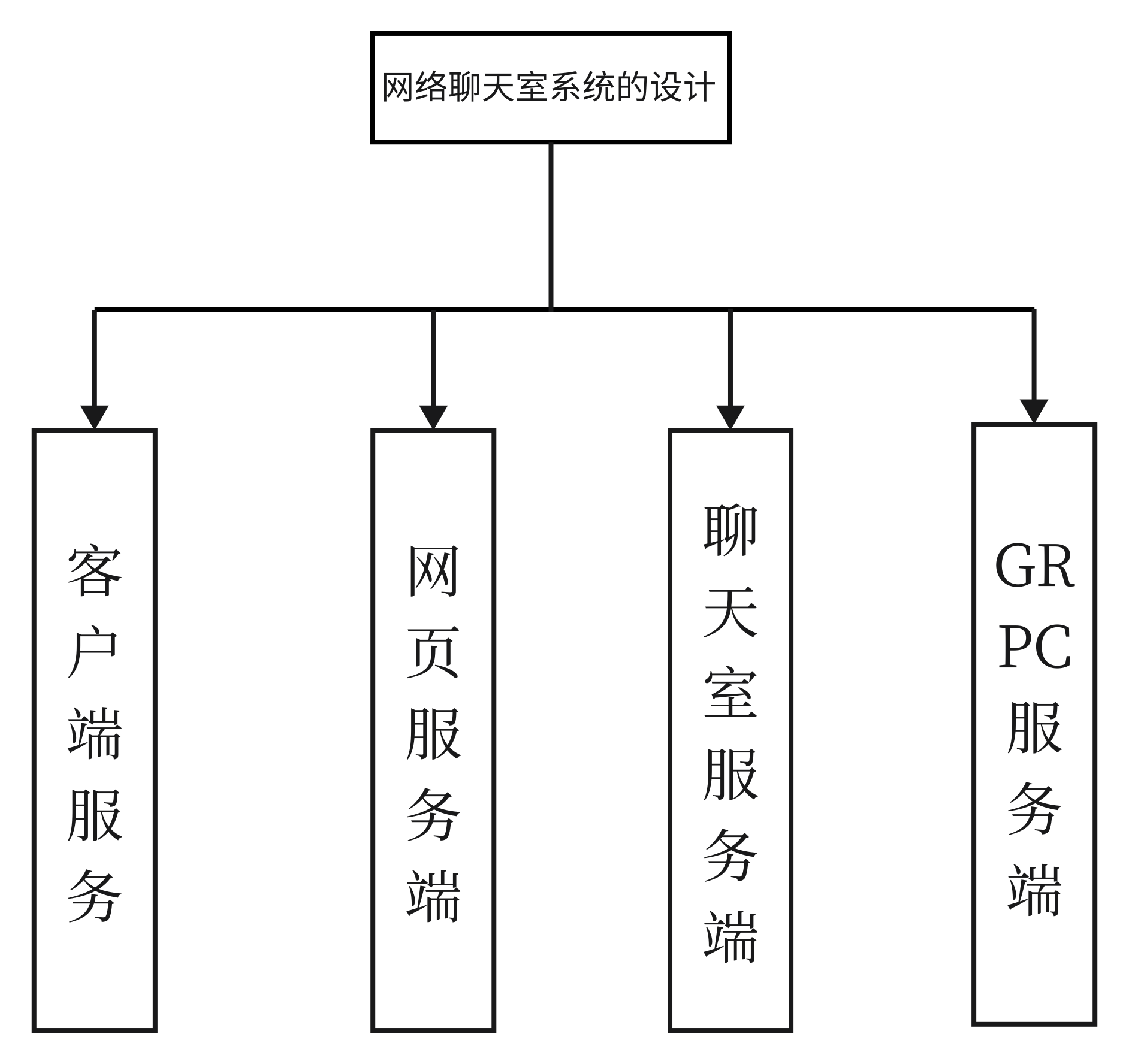


图2-3 功能服务图

## 2.7客户端服务

客户端服务是该网络聊天室系统的客户端,主要负责发送消息给聊天室服务端以及对聊天室服务器消息的接收,还有对网页服务端的发送请求和接收返回的数据,是普通用户能够触及到的,该客户端服务提供了友好的界面给用户使用,其中能够注册和登录。

当选择登陆时,客户端服务向网页服务端发送http请求,先是获取验证码,再然后诗对登陆信息的校验。同理的,注册时也是发起http请求,先校验手机号码的格式是否通过,再根据手机号码发送短信,手机号码短信匹配再继续后面的逻辑。

用户登陆成功后能够在该聊天室服务端中的所有用户发送信息,也就是与他们进行通讯,实现聊天功能。用户也可以查看自己当前账号的信息。当用户选择下线时,聊天室服务器能够处理该用户下线的通知。总的来说,客户端服务依赖于其他三个服务,也是用户用到的主要服务之一。

## 2.8聊天室服务端

聊天室服务端主要负责接收客户端服务的消息以及发送消息给客户端服务,该服务是普通用户不能直接触摸到的。

聊天室服务端根据接收客户端服务的不同类型的消息进行不一样的处理。比如说,当遇到客户端需要注册用户的消息类型,就会先去Redis数据库查询是否已经有该用户存在了,再考虑注册是否继续进行。同理的,遇到登录用户的消息类型也是,先去Redis数据库查询下是否已经有该用户存在了, 再考虑登录是否继续进行。

最常见的就是用户聊天信息的类型了,聊天室服务端将该消息包装好,然后向该聊天室里所有在线的用户一个一个的发送过去,保证每一个在线用户都接收到该消息。总的来说聊天室服务端主要是和客户端配合,具体的实用功能都是由其实现。

## 2.9网页服务端

网页服务端主要负责对客户端服务登录和注册的信息进行处理,客户端服务每次网页请求都会经过网页服务端接收,处理的结果也是网页服务端进行返回。

但是网页服务端其实本质上是一个GRPC的客户端,起到了中间网页夹层的作用.通过网页包装了一层,实质通过网页又发送了一个向GRPC服务端的连接,不同的网页请求会导致GRPC服务端返回多样的响应。

最后再把该响应返回给客户端服务。其中发送短信服务和验证码的服务不经过GRPC服务端,是由网页服务端内置路由完成的。

## 2.10 GRPC服务端

GRPC服务端主要负责对网页服务端的用户的相关数据处理,例如用户的手机号和昵称等信息,网页服务端的每次连接,在保证没有错误的情况下会顺利的将数据保存到GRPC服务端所连接的数据库中去,从而完成用户的登录和注册,调用数据库的方面使用了GORM,简化了数据库的增删改查操作,保证了数据的安全性和高效性。

总的来说,客户端服务到网页服务端的用户数据最终都会到该GRPC服务端去处理。

## 2.11程序流程图

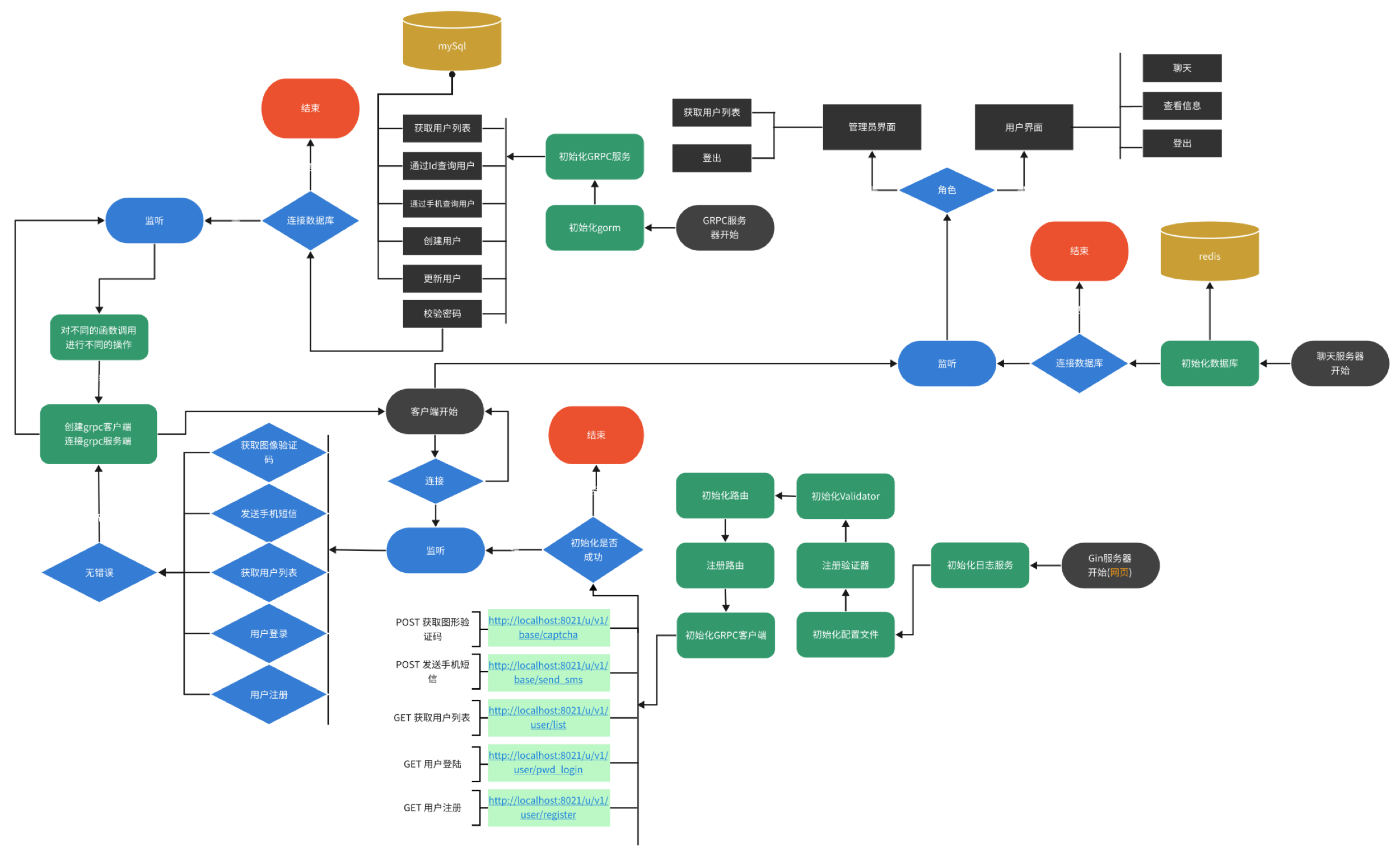


图2-4 程序流程图

## 2.12数据结构表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 成员名 | 数据类型 | 长度 | 字段含义 | 字段描述 |
| Id | char | 20 | Id | 主键 |
| Mobile | char | 11 | 手机号 | 不能为空 |
| Password | char | 100 | 密码 | 不能为空 |
| NickName | char | 20 | 昵称 | `昵称` |
| Role | char | 1 | 角色 | `管理员`,`用户`不能为空 |
| Gender | char | 6 | 性别 | `男`,`女` |

图2-5 MySQL用户表

# 第三章 用户使用说明与调试结果

## 3.1使用说明

这是一个网络聊天系统，具有登录、注册、查询、用户聊天、保存等功能，建立了一个含有姓名、性别、手机号、角色、等用户的信息。开始程序后，可按照有关提示进行相关操作：

1. 登陆聊天系统按“1”键;
2. 注册用户按“2”键;
3. 退出系统按“3”键

界面有多个,当用户登陆成功后的相关操作:

1. 显示在线用户列表按“1”键;
2. 发送消息按“2”键;
3. 查看信息按“3”键;
4. 退出系统按“4”键;

## 3.2调试结果

代码中运用了if 语句调用各子函数，数字 1、2、3、4、等四个数字代表不同的指令。

先是开始程序,显示如下:

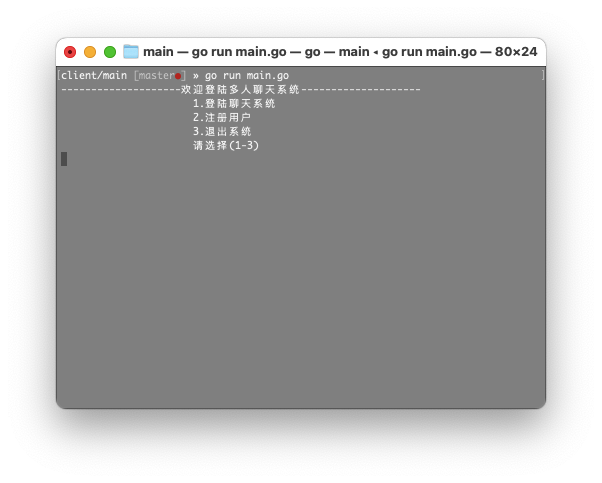


图2-4 开始程序界面

开始后按“2”键进行用户的注册,显示如下:

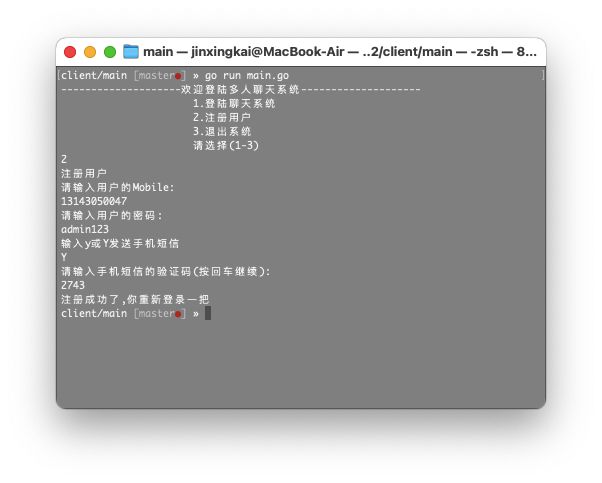


图2-5 用户注册界面

注册成功后按“1”键进行用户登录, 登陆成功后,进入主菜单。显示如下:

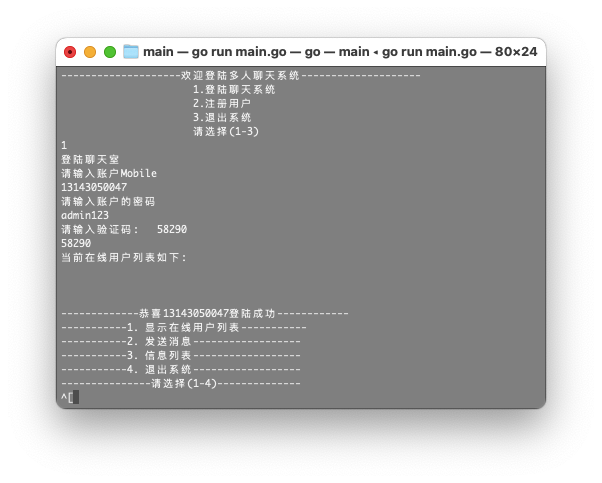


图2-6 用户登陆成功界面

若要查看当前的用户信息按“3”键。显示如下:



图2-7 查看用户信息界面

若有其他用户上线时显示如下:

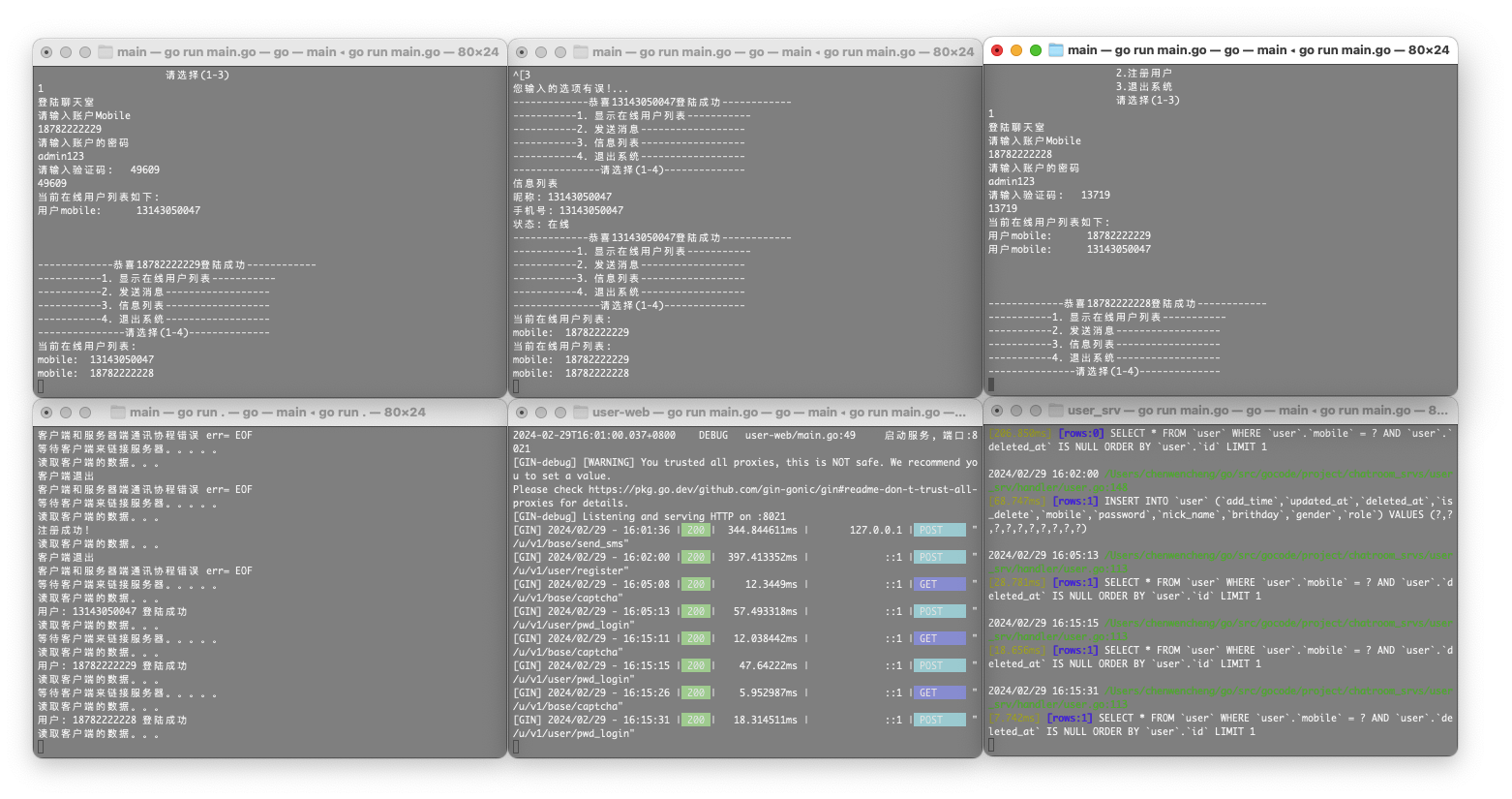


图2-8 其他用户上线界面

若要发送消息按“2”键,显示如下:

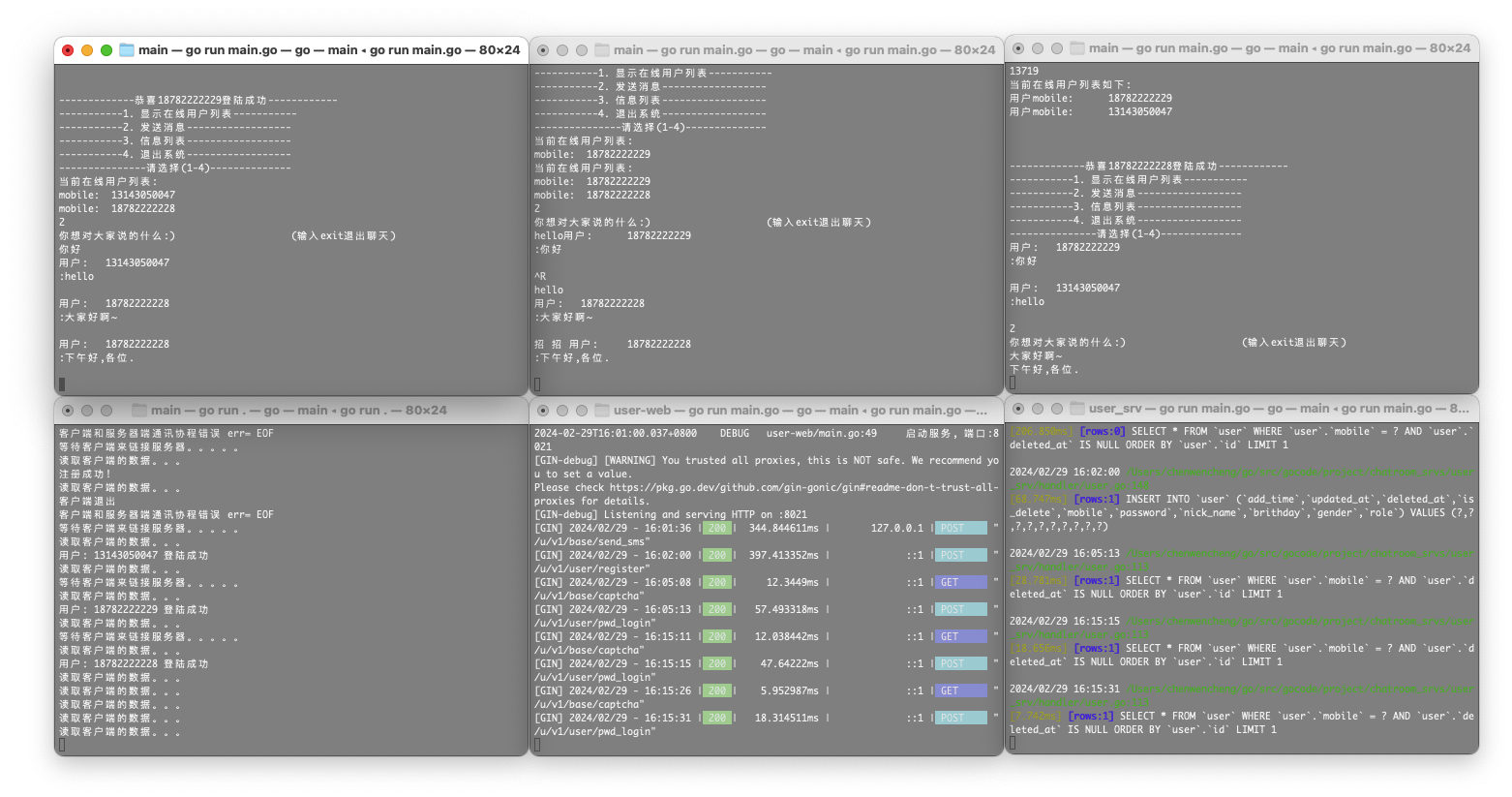


图2-9 发送消息界面

若要显示在线用户按“1”键。显示如下:

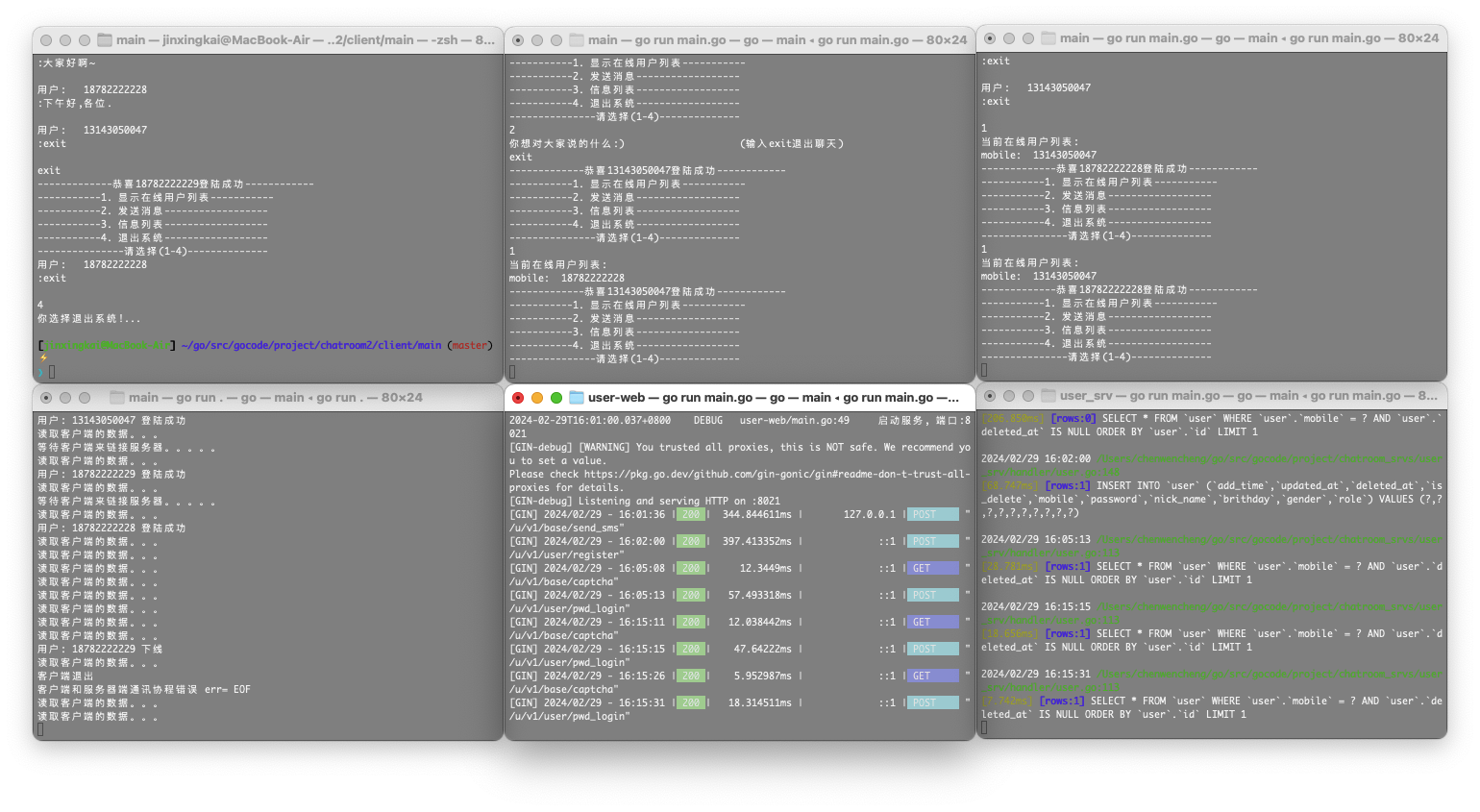


图2-10 发送消息界面

如果是管理员用户登录成功。显示如下:

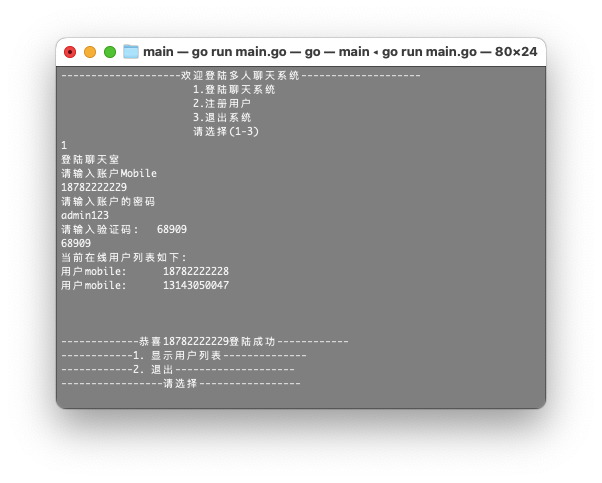


图2-11 管理员登陆成功界面

若要查看数据库所有用户按“1”,显示如下:



图2-12 查看所有用户界面

## 3.3优缺点分析

(1)优点:

* 服务端使用配置文件的方式读取初始化数据,例如服务器的ip以及端口的读取;
* 高内聚低耦合,每个服务都有自己的数据库，并专注于处理特定的业务功能;
* 服务端处对用户的每个操作有日志输出,易于维护;
* 可以随意地选择不同的指令进行操作；
* 客户端界面在终端上实现,简洁;
* 服务端处错误处理较完美;
* 操作简单方便;
* 高效性;

(2)缺点:

* 网络延迟问题,服务之间需要处理大量网络请求，可能导致网络延迟问题，影响系统性能;
* 数据库不一致,聊天室服务端的数据库和GRPC服务端的数据库是Redis和MySQL;
* 客户端的界面呈现实在终端上,在现代程序的对比下,显得不怎么美观;
* 服务器成本高;

# 第五章 致谢

首先，我要感谢我的指导老师，他在我整个毕业设计过程中给予了悉心的指导和支持。他的严谨治学态度和深厚的专业知识让我受益匪浅。同时，我也要感谢我的同学们，他们在整个过程中给予了我很多帮助和支持，让我能够顺利地完成毕业设计。

其次，我要感谢Go语言的开发者们，是他们创造了如此强大而优美的编程语言，让我能够实现自己的想法。同时，我也要感谢开源社区的贡献者们，他们的代码和经验为我提供了宝贵的参考和借鉴。

最后，我要感谢我的家人和朋友，他们一直支持着我，鼓励着我，让我能够专注于毕业设计的研究和实现。他们的支持和关爱是我前进的动力和源泉。 在即将毕业之际，我再次向所有关心、支持和帮助过我的人表示衷心的感谢！

# 总 结

本次设计我选择的题目是基于Go语言实现的网络聊天系统。因为事先的计划非常明确，所以我们在完成这个设计的过程也是比较顺利的。在系统分析和设计的部分仍然花了比较多的时间。服务设计部分因为涉及功能的实现，要从抽象的设计转化成实际的代码，因此对于某项功能的实现方法都思考得比较详尽。加上在开发系统之前，我阅读很多与本系统有相关内容的书籍以做参考，使得此次系统编写能顺利进行。由于班里每个人都需要做一份设计，所以我们经常互相交流。在交流的时候经常能够互相提出更好的建议。

经过对网络聊天室系统的分析与编写，我得到了很大的收获。自主学习的必要性更是解决问题的基础。这次设计使我重温了以前学过的各科知识，让我对Go编程有了更深刻的认识。也因为这样，如何编写代码逐渐变得明朗。每一个功能模块经过调查后后，各个功能逐渐显露出来，然后，反复探讨，验证这项功能是否有存在的必要，是否可以改进得更好。在这样的前提下，最后顺利完成了这份设计。

通过实践，我们深入了解了Go语言的并发特性和网络编程技巧，并成功地构建了一个稳定、高效的聊天室系统。该系统支持多用户同时在线聊天，具备良好的实时性和交互性。 然而，尽管我们的设计取得了一定的成果，但仍存在一些问题需要进一步研究和解决。

1. **安全性问题**：在当前的实现中，我们未对用户输入进行严格的验证和过滤，这可能导致潜在的安全风险，如XSS攻击或SQL注入攻击。未来的工作需要加强系统的安全性，对用户输入进行有效的过滤和验证。
2. **实时性优化**：虽然我们的聊天室系统已经具备一定的实时性，但在高并发或大规模用户场景下，系统的实时性可能会受到影响。未来可以研究如何进一步优化实时通信机制，提高系统的响应速度和稳定性。
3. **用户管理**：当前系统未实现用户注册、登录和权限管理等基本功能，这限制了系统的应用范围。未来的工作可以扩展用户管理模块，提供更丰富的用户交互功能。
4. **可扩展性**：随着用户数量的增加，系统的可扩展性成为一个重要的问题。如何设计一个可扩展的架构，以支持大规模用户同时在线，是未来研究的一个重要方向。
5. **错误处理和日志记录**：当前系统在错误处理和日志记录方面存在不足，这可能导致问题排查和故障恢复的困难。未来的工作需要加强错误处理和日志记录机制，提高系统的稳定性和可靠性。 综上所述，虽然我们的网络聊天室设计取得了一定的成果，但仍存在一些问题需要进一步研究和解决。通过不断优化和完善系统，我们可以构建一个更加稳定、安全、高效的网络聊天室，为用户提供更好的实时交互体验。

这次Go语言系统设计，提升了我分析问题的能力和思考问题的逻辑能力，更明白实际调研的重要性。从这次设计学到的东西，相信会对我们日后的学习工作起到很大的指导作用

# 参考文献

1. 唐汉明,翟振兴,关宝军,等.深入浅出MySQL[M].人民邮电出版社:201401.655.
2. 付磊,张益军.Redis开发与运维[M].机械工业出版社:201702.876.
3. 李代立,陈榕.WebSocket在Web实时通信领域的研究[J].电脑知识与技术,2010,6(28):7923-7925+7935.
4. 王磊.微服务架构与实践[M].电子工业出版社:201601.237.
5. 周明辉.基于Golang+Gin的技术运维系统设计与实现[J].现代电视技术,2022,(10):134-137.
6. 王斌斌.基于Protobuf的RPC系统的设计与实现[D].电子科技大学,2016.