课程内容：Gin+WebSocket 项目实战IM

介绍：

Gin 是目前 golang使用最广泛的Web架构之一 。

web开发框架，适合api接口、微服务开发，相较于其他框架(iris、beego)更轻量级和更好的性能。其路由功能很强大提供分组功能，非常适合做api开发

API：https://gin-gonic.com/zh-cn/docs/

需求分析：

项目目的：

项目背景：IM对性能和体验敏感度非常高 。 大厂必备

你将获得什么：

熟悉开发流程 ，熟练相关技术栈 gin+GORM+swagger + logrus auth 等中间件，三高性能

核心功能：

发送和接受消息，文字 表情 图片 音频 ，访客，点对点，群聊 ，广播，快捷回复，撤回，心跳检测....

技术栈：

前端 后端 （webSocket ,channel/goroutine ,gin ,temlate,gorm ,sql,nosql,mq.....）

系统架构：

四层：前端，接入层，逻辑层，持久层

消息发送流程：

A > 登录> 鉴权>(游客) > 消息类型 >(群/广播) > B

（需求文档 表设置 功能设计文档.....）

环境搭建：

go version go1.17.8 windows/amd64

set GO111MODULE=on

go mod init go_exam

go mod tidy

一:引入GORM

https://pkg.go.dev/ 搜到 GORM

https://gorm.io/zh_CN/docs/ 中文API

首先 直接用快速开始的代码改

go get -u gorm.io/gorm
go get -u gorm.io/driver/mysql

用户模块struct设计

创建models 包 user_basic.go 再写一个 struct

```
package models

import (
    "time"

    "gorm.io/gorm"
)

type UserBasic struct {
    gorm.Model
    Name          string
    PassWord      string
    Phone         string
    Email         string
    Identity      string
    ClientIp      string
    ClientPort    string
    LoginTime     time.Time
    HeartbeatTime time.Time
    LoginOutTime  time.Time `gorm:"column:login_out_time"
json:"login_out_time"`
    IsLogout      bool
    DeviceInfo    string
}

func (table *UserBasic) TableName() string {
    return "user_basic"
}
```

测试 数据正确性。

让后封装 配置类 测试类 。

创建 test包 testGorm.go

```
package main

import (
    "fmt"
    "ginchat/models"

    "gorm.io/driver/mysql"
    "gorm.io/gorm"
)

func main() {
    db, err :=
gorm.Open(mysql.Open("root:1234@tcp(127.0.0.1:3307)/ginchat?
charset=utf8mb4&parseTime=True&loc=Local"), &gorm.Config{})
    if err != nil {
        panic("failed to connect database")
    }

    // 迁移 schema
    db.AutoMigrate(&models.UserBasic{})

    // Create
    user := &models.UserBasic{}
    user.Name = "申专"
    db.Create(user)

    // Read
    fmt.Println(db.First(user, 1)) // 根据整型主键查找
    //db.First(user, "code = ?", "D42") // 查找 code 字段值为 D42 的记录

    // Update - 将 product 的 price 更新为 200
    db.Model(user).Update("PassWord", "1234")
    // Update - 更新多个字段
    //db.Model(&product).Updates(Product{Price: 200, Code: "F42"}) // 仅
更新非零值字段
    //db.Model(&product).Updates(map[string]interface{}{"Price": 200,
"Code": "F42"})

    // Delete - 删除 product
    //db.Delete(&product, 1)
}
```

运行查看数据情况

二：引入Gin框架

https://pkg.go.dev/ 搜：

```
get -u github.com/gin-gonic/gin

//项目根目录 main.go
package main

import (
    "ginchat/router" //  router "ginchat/router"
)

func main() {
    r := router.Router() // router.Router()
    r.Run(":8081")        // listen and serve on 0.0.0.0:8080 (for
windows "localhost:8080")
}

//router 包 app.go
package router

import (
    "ginchat/service"

    "github.com/gin-gonic/gin"
)

func Router() *gin.Engine {
    r := gin.Default()
    r.GET("/index", service.GetIndex)

    return r
}

//service包  index.go
package service
import "github.com/gin-gonic/gin"
func GetIndex(c *gin.Context) {
    c.JSON(200, gin.H{
        "message": "welcome !!  ",
    })
}
```

三：将数据和请求关联起来

1.在main方法里面 初始化配置文件 以及 数据库

utils.InitConfig()

utils.InitMySQL()

2.再config 包里面 app.yaml

```
mysql:

  dns: root:1234@tcp(127.0.0.1:3307)/ginchat?
charset=utf8mb4&parseTime=True&loc=Local
```

3.新建 utils包 system_init.go

```
package utils

import (
    "fmt"

    "github.com/spf13/viper"
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
)

var DB *gorm.DB

func InitConfig() {
    viper.SetConfigName("app")
    viper.AddConfigPath("config")
    err := viper.ReadInConfig()
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println("config  app inited 。。。。")
}
func InitMySQL() {
    DB, _ = gorm.Open(mysql.Open(viper.GetString("mysql.dns")),
        &gorm.Config{})
    fmt.Println(" MySQL inited 。。。。")
    //user := models.UserBasic{}
    //DB.Find(&user)
    //fmt.Println(user)
}
```

4.到models包里面 user_basic.go 里面加上 下面的方法

```
func GetUserList() []*UserBasic {
    data := make([]*UserBasic, 10)
    utils.DB.Find(&data)
    for _, v := range data {
        fmt.Println(v)
    }
    return data
}
```

5.到 service包里面新建 userservice.go

```
package service

import (
    "ginchat/models"

    "github.com/gin-gonic/gin"
)

func GetUserList(c *gin.Context) {
    data := make([]*models.UserBasic, 10)
    data = models.GetUserList()

    c.JSON(200, gin.H{
        "message": data,
    })
}
```

6.到 router包的app.go 里面的Router方法加上这一行即可

```
    r.GET("/user/getUserList", service.GetUserList)
```

四：整合swagger

搜 gin-swagger

```
go get -u github.com/swaggo/swag/cmd/swag

swag init
查看是否项目多一个dosc目录

然后拉取 ：
go get -u github.com/swaggo/gin-swagger
go get -u github.com/swaggo/files
```

改造我们的router包的 app.go

```
package router

import (
    "ginchat/docs"
    "ginchat/service"

    "github.com/gin-gonic/gin"
    swaggerfiles "github.com/swaggo/files"
    ginSwagger "github.com/swaggo/gin-swagger"
)

func Router() *gin.Engine {

    r := gin.Default()
    docs.SwaggerInfo.BasePath = ""
    r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerfiles.Handler))

    r.GET("/index", service.GetIndex)
    r.GET("/user/getUserList", service.GetUserList)

    return r
}
```

然后测试 http://localhost:8081/swagger/index.html

然后再到 service层 index.go 方法上加注释

```
// GetIndex
// @Tags 首页
// @Success 200 {string} welcome
// @Router /index [get]
func GetIndex(c *gin.Context) {
    c.JSON(200, gin.H{
        "message": "welcome !!  ",
    })
}
```

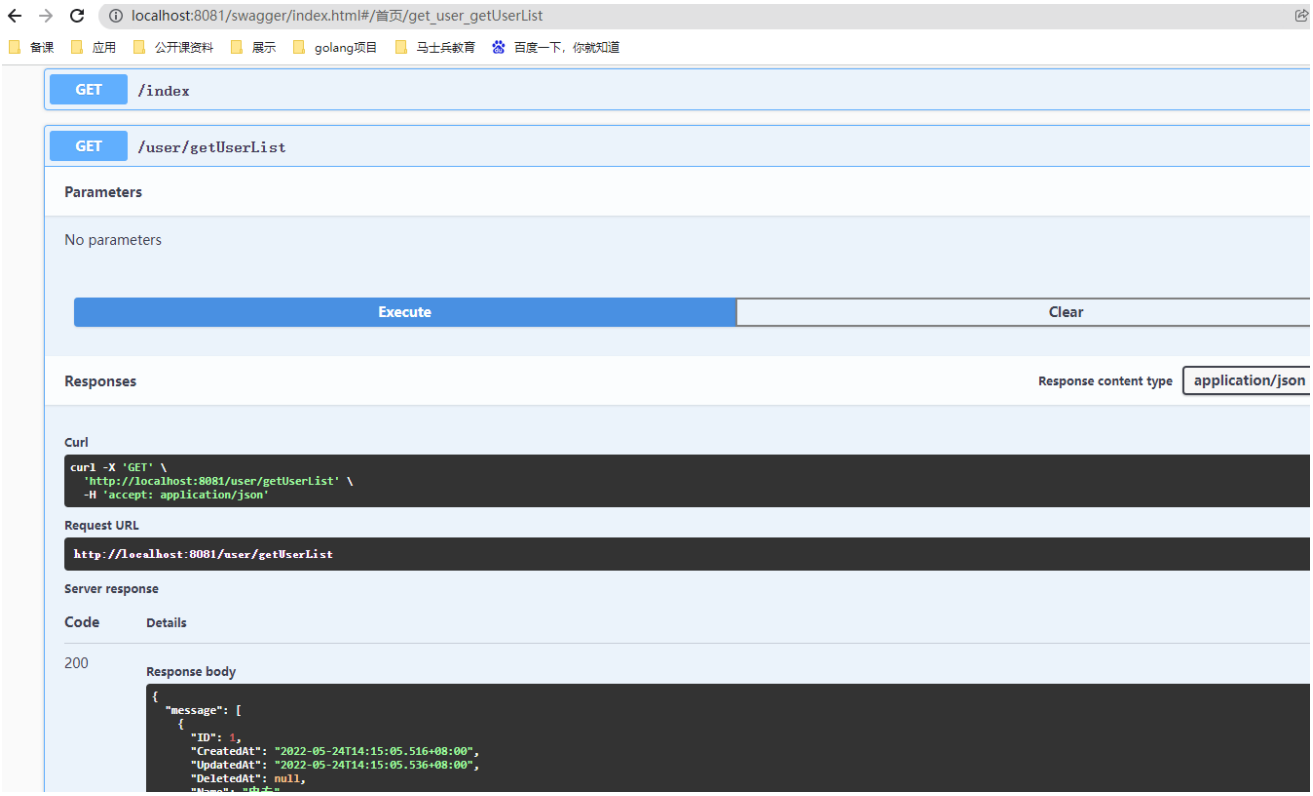
以及userservice.go

```
// GetUserList
// @Tags 首页
// @Success 200 {string} json{"code","message"}
// @Router /user/getUserList [get]
func GetUserList(c *gin.Context) {
    data := make([]*models.UserBasic, 10)
    data = models.GetUserList()

    c.JSON(200, gin.H{
        "message": data,
```

```
    })
}
```

swag init 一下 ，重启服务

重新测试 http://localhost:8081/swagger/index.html 就能看到 index 和 getUserList 的请求



五：日志打印

在数据库初始化的时候 加入自己的 logger

```
func InitMySQL() {
    //自定义日志模板 打印SQL语句
    newLogger := logger.New(
        log.New(os.Stdout, "\r\n", log.LstdFlags),
        logger.Config{
            SlowThreshold: time.Second, //慢SQL阈值
            LogLevel:      logger.Info, //级别
            Colorful:      true,        //彩色
        },
    )
    DB, _ = gorm.Open(mysql.Open(viper.GetString("mysql.dns")),
        &gorm.Config{Logger: newLogger})
    fmt.Println(" MySQL inited 。。。。")
    //user := models.UserBasic{}
    //DB.Find(&user)
    //fmt.Println(user)
}
```

功能实现

完成用户模块基本的

加入修改电话号码和邮箱 并校验

先引入

```
get github.com/asaskevich/govalidator
结构体字段后面 加检验规则
最后service govalidator.ValidatorStrut(user)
```

1.router包 app.go

```
    r.GET("/user/getUserList", service.GetUserList)
    r.GET("/user/createUser", service.CreateUser)
    r.GET("/user/deleteUser", service.DeleteUser)
    r.POST("/user/updateUser", service.UpdateUser)
```

2.service 包 userservice.go

```
// GetUserList
// @Summary 所有用户
// @Tags 用户模块
// @Success 200 {string} json{"code","message"}
// @Router /user/getUserList [get]
func GetUserList(c *gin.Context) {
    data := make([]*models.UserBasic, 10)
    data = models.GetUserList()

    c.JSON(200, gin.H{
        "message": data,
    })
}

// CreateUser
// @Summary 新增用户
// @Tags 用户模块
// @param name query string false "用户名"
// @param password query string false "密码"
// @param repassword query string false "确认密码"
// @Success 200 {string} json{"code","message"}
// @Router /user/createUser [get]
```

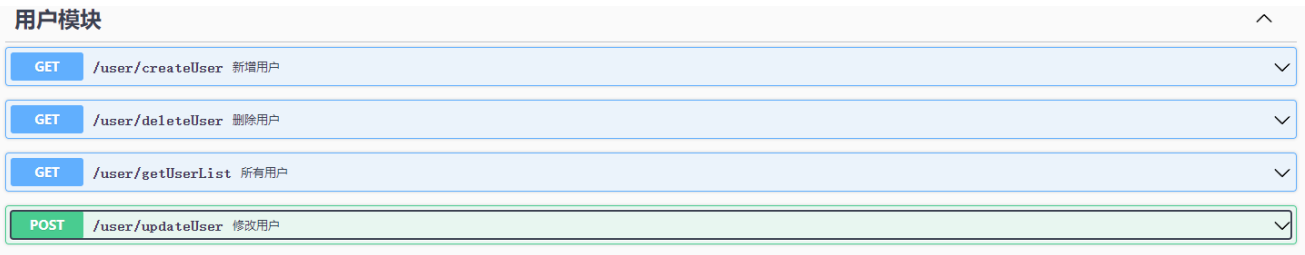
```go
func CreateUser(c *gin.Context) {
    user := models.UserBasic{}
    user.Name = c.Query("name")
    password := c.Query("password")
    repassword := c.Query("repassword")
    if password != repassword {
        c.JSON(-1, gin.H{
            "message": "两次密码不一致！",
        })
        return
    }
    user.PassWord = password
    models.CreateUser(user)
    c.JSON(200, gin.H{
        "message": "新增用户成功！",
    })
}

// DeleteUser
// @Summary 删除用户
// @Tags 用户模块
// @param id query string false "id"
// @Success 200 {string} json{"code","message"}
// @Router /user/deleteUser [get]
func DeleteUser(c *gin.Context) {
    user := models.UserBasic{}
    id, _ := strconv.Atoi(c.Query("id"))
    user.ID = uint(id)
    models.DeleteUser(user)
    c.JSON(200, gin.H{
        "message": "删除用户成功！",
    })
}

// UpdateUser
// @Summary 修改用户
// @Tags 用户模块
// @param id formData string false "id"
// @param name formData string false "name"
// @param password formData string false "password"
// @param phone formData string false "phone"
// @param email formData string false "email"
// @Success 200 {string} json{"code","message"}
// @Router /user/updateUser [post]
func UpdateUser(c *gin.Context) {
    user := models.UserBasic{}
    id, _ := strconv.Atoi(c.PostForm("id"))
    user.ID = uint(id)
    user.Name = c.PostForm("name")
```

```
    user.PassWord = c.PostForm("password")
    user.Phone = c.PostForm("phone")
    user.Email = c.PostForm("email")
    fmt.Println("update :", user)

    _, err := govalidator.ValidateStruct(user)
    if err != nil {
        fmt.Println(err)
        c.JSON(200, gin.H{
            "message": "修改参数不匹配！",
        })
    } else {
        models.UpdateUser(user)
        c.JSON(200, gin.H{
            "message": "修改用户成功！",
        })
    }

}
```

3.modesl包 user_basic.go

```
    Phone          string `valid:"matches(^1[3-9]{1}\\d{9}$)"`
    Email          string `valid:"email"`
```

4，然后测试



重复注册校验：

```
func FindUserByName(name string) UserBasic {
    user := UserBasic{}
    utils.DB.Where("name = ?", name).First(&user)
    return user
}
func FindUserByPhone(phone string) *gorm.DB {
    user := UserBasic{}
    return utils.DB.Where("Phone = ?", phone).First(&user)
}
```

```go
func FindUserByEmail(email string) *gorm.DB {
    user := UserBasic{}
    return utils.DB.Where("email = ?", email).First(&user)
}
```

再到service层 加入判断

```go
    data := models.FindUserByName(user.Name)
    if data.Name != "" {
        c.JSON(-1, gin.H{
            "message": "用户名已注册！",
        })
        return
    }
```

注册 加密操作

```go
package utils

import (
    "crypto/md5"
    "encoding/hex"
    "fmt"
    "strings"
)

//小写
func Md5Encode(data string) string {
    h := md5.New()
    h.Write([]byte(data))
    tempStr := h.Sum(nil)
    return hex.EncodeToString(tempStr)
}

//大写
func MD5Encode(data string) string {
    return strings.ToUpper(Md5Encode(data))
}

//加密
func MakePassword(plainpwd, salt string) string {
    return Md5Encode(plainpwd + salt)
}

//解密
```

```
func ValidPassword(plainpwd, salt string, password string) bool {
    md := Md5Encode(plainpwd + salt)
    fmt.Println(md + "               " + password)
    return md == password
}



service层 判断之后加入
    //user.PassWord = password
    user.PassWord = utils.MakePassword(password, salt)
    user.Salt = salt  //表更新了字段  db.AutoMigrate(&models.UserBasic{})
    fmt.Println(user.PassWord)
    models.CreateUser(user)
```

登录解密：

```
//dao层
func FindUserByNameAndPwd(name string, password string) UserBasic {
    user := UserBasic{}
    utils.DB.Where("name = ? and pass_word=?", name,
password).First(&user)
    return user
}



// GetUserList
// @Summary 所有用户
// @Tags 用户模块
// @param name query string false "用户名"
// @param password query string false "密码"
// @Success 200 {string} json{"code","message"}
// @Router /user/findUserByNameAndPwd [get]
func FindUserByNameAndPwd(c *gin.Context) {
    data := models.UserBasic{}

    name := c.Query("name")
    password := c.Query("password")
    user := models.FindUserByName(name)
    if user.Name == "" {
        c.JSON(200, gin.H{
            "message": "该用户不存在",
        })
        return
    }
```

```
    flag := utils.ValidPassword(password, user.Salt, user.PassWord)
    if !flag {
        c.JSON(200, gin.H{
            "message": "密码不正确",
        })
        return
    }
    pwd := utils.MakePassword(password, user.Salt)
    data = models.FindUserByNameAndPwd(name, pwd)

    c.JSON(200, gin.H{
        "message": data,
    })
}
```

router层 ：

```
r.POST("/user/findUserByNameAndPwd", service.FindUserByNameAndPwd)
```

token的加入对返回的结构调整。

修改登录的方法：

```
func FindUserByNameAndPwd(name string, password string) UserBasic {
    user := UserBasic{}
    utils.DB.Where("name = ? and pass_word=?", name,
password).First(&user)

    //token加密
    str := fmt.Sprintf("%d", time.Now().Unix())
    temp := utils.MD5Encode(str)
    utils.DB.Model(&user).Where("id = ?", user.ID).Update("identity",
temp)
    return user
}


// 返回的结果：
    c.JSON(200, gin.H{
            "code":    0, //  0成功   -1失败
            "message": "修改用户成功！",
            "data":    user,
        })
```

加入Redis

go get github.com/go-redis/redis

配置redis

```
redis:
  addr: "192.168.137.131:6379"
  password: ""
  DB: 0
  poolSize: 30
  minIdleConn: 30
```

然后main方法中

utils.InitRedis()

最后再 utils

```
func InitRedis() {
    Red = redis.NewClient(&redis.Options{
        Addr:         viper.GetString("redis.addr"),
        Password:     viper.GetString("redis.password"),
        DB:           viper.GetInt("redis.DB"),
        PoolSize:     viper.GetInt("redis.poolSize"),
        MinIdleConns: viper.GetInt("redis.minIdleConn"),
    })
    pong, err := Red.Ping().Result()
    if err != nil {
        fmt.Println("init redis  。。。。", err)
    } else {
        fmt.Println(" Redis inited 。。。。", pong)
    }
}
```

测试看是否正常

通过WebSocket通信

```
go get github.com/gorilla/websocket
go get github.com/go-redis/redis/v8



package utils
```

```
import (
    "context"
    "fmt"
    "log"
    "os"
    "time"

    "github.com/go-redis/redis/v8"
    "github.com/spf13/viper"
    "gorm.io/driver/mysql"
    "gorm.io/gorm"
    "gorm.io/gorm/logger"
)

var (
    DB  *gorm.DB
    Red *redis.Client
)

func InitConfig() {
    viper.SetConfigName("app")
    viper.AddConfigPath("config")
    err := viper.ReadInConfig()
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println("config  app inited 。。。。")
}

func InitMySQL() {
    //自定义日志模板 打印SQL语句
    newLogger := logger.New(
        log.New(os.Stdout, "\r\n", log.LstdFlags),
        logger.Config{
            SlowThreshold: time.Second, //慢SQL阈值
            LogLevel:      logger.Info, //级别
            Colorful:      true,        //彩色
        },
    )

    DB, _ = gorm.Open(mysql.Open(viper.GetString("mysql.dns")),
        &gorm.Config{Logger: newLogger})
    fmt.Println(" MySQL inited 。。。。")
    //user := models.UserBasic{}
    //DB.Find(&user)
    //fmt.Println(user)
}
```

```
func InitRedis() {
    Red = redis.NewClient(&redis.Options{
        Addr:         viper.GetString("redis.addr"),
        Password:     viper.GetString("redis.password"),
        DB:           viper.GetInt("redis.DB"),
        PoolSize:     viper.GetInt("redis.poolSize"),
        MinIdleConns: viper.GetInt("redis.minIdleConn"),
    })
}

const (
    PublishKey = "websocket"
)

//Publish 发布消息到Redis
func Publish(ctx context.Context, channel string, msg string) error {
    var err error
    fmt.Println("Publish 。。。。", msg)
    err = Red.Publish(ctx, channel, msg).Err()
    if err != nil {
        fmt.Println(err)
    }
    return err
}

//Subscribe 订阅Redis消息
func Subscribe(ctx context.Context, channel string) (string, error) {
    sub := Red.Subscribe(ctx, channel)
    fmt.Println("Subscribe 。。。。", ctx)
    msg, err := sub.ReceiveMessage(ctx)
    if err != nil {
        fmt.Println(err)
        return "", err
    }
    fmt.Println("Subscribe 。。。。", msg.Payload)
    return msg.Payload, err
}
```

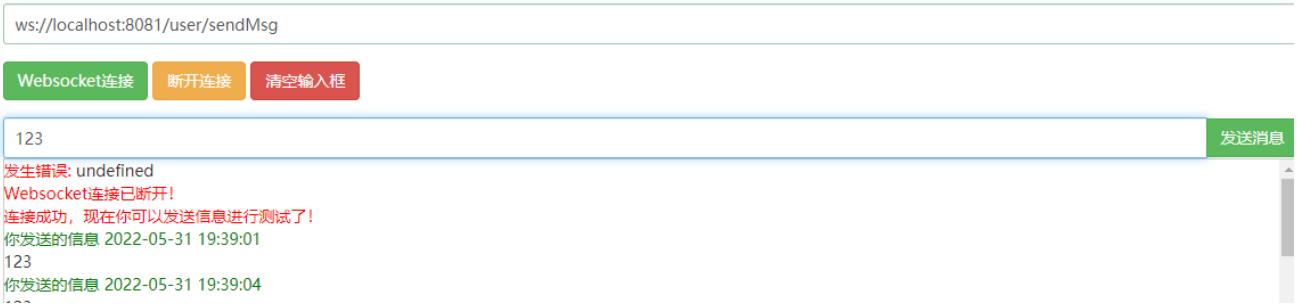
userservice.go中加入

```
//防止跨域站点伪造请求
var upGrader = websocket.Upgrader{
    CheckOrigin: func(r *http.Request) bool {
        return true
    },
```

```
}

func SendMsg(c *gin.Context) {
    ws, err := upGrader.Upgrade(c.Writer, c.Request, nil)
    if err != nil {
        fmt.Println(err)
        return
    }
    defer func(ws *websocket.Conn) {
        err = ws.Close()
        if err != nil {
            fmt.Println(err)
        }
    }(ws)
    MsgHandler(c, ws)
}
func MsgHandler(c *gin.Context, ws *websocket.Conn) {
    for {
        msg, err := utils.Subscribe(c, utils.PublishKey)
        if err != nil {
            fmt.Println(" MsgHandler 发送失败", err)
        }

        tm := time.Now().Format("2006-01-02 15:04:05")
        m := fmt.Sprintf("[ws][%s]:%s", tm, msg)
        err = ws.WriteMessage(1, []byte(m))
        if err != nil {
            log.Fatalln(err)
        }
    }
}



router层 app.go
    //发送消息
    r.GET("/user/sendMsg", service.SendMsg)
```

测试： http://www.jsons.cn/websocket/

ws://localhost:8081/user/sendMsg



设计 关系表，群信息表，消息表

```go
package models

import "gorm.io/gorm"

//消息
type Message struct {
    gorm.Model
    FormId   uint   //发送者
    TargetId uint   //接受者
    Type     string //消息类型  群聊 私聊 广播
    Media    int    //消息类型  文字 图片 音频
    Content  string //消息内容
    Pic      string
    Url      string
    Desc     string
    Amount   int //其他数字统计
}

func (table *Message) TableName() string {
    return "message"
}


package models

import "gorm.io/gorm"

//群信息
type GroupBasic struct {
```

```
    gorm.Model
    Name     string
    OwnerId uint
    Icon     string
    Type     int
    Desc     string
}

func (table *GroupBasic) TableName() string {
    return "group_basic"
}
```

```
package models

import "gorm.io/gorm"

//人员关系
type Contact struct {
    gorm.Model
    OwnerId  uint //谁的关系信息
    TargetId uint //对应的谁
    Type     int  //对应的类型  0  1  3
    Desc     string
}

func (table *Contact) TableName() string {
    return "contact"
}
```

发送消息 接受消息

需要： 发送者ID ，接受者ID ，消息类型，发送的内容，发送类型

校验token ，关系，

```
package models

import (
    "encoding/json"
    "fmt"
    "net"
    "net/http"
    "strconv"
    "sync"

    "github.com/gorilla/websocket"
    "gopkg.in/fatih/set.v0"
```

```
    "gorm.io/gorm"
)

//消息
type Message struct {
    gorm.Model
    FormId   int64  //发送者
    TargetId int64  //接受者
    Type     int    //发送类型  群聊 私聊 广播
    Media    int    //消息类型  文字 图片 音频
    Content  string //消息内容
    Pic      string
    Url      string
    Desc     string
    Amount   int //其他数字统计
}

func (table *Message) TableName() string {
    return "message"
}

type Node struct {
    Conn      *websocket.Conn
    DataQueue chan []byte
    GroupSets set.Interface
}

//映射关系
var clientMap map[int64]*Node = make(map[int64]*Node, 0)

//读写锁
var rwLocker sync.RWMutex

//  需要 ：发送者ID ，接受者ID ，消息类型，发送的内容，发送类型
func Chat(writer http.ResponseWriter, request *http.Request) {
    //1.  获取参数 并 检验 token 等合法性
    //token := query.Get("token")
    query := request.URL.Query()
    Id := query.Get("userId")
    userId, _ := strconv.ParseInt(Id, 10, 64)
    //msgType := query.Get("type")
    //targetId := query.Get("targetId")
    //  context := query.Get("context")
    isvalida := true //checkToke()  待.........
    conn, err := (&websocket.Upgrader{
        //token 校验
        CheckOrigin: func(r *http.Request) bool {
            return isvalida
        },
```

```
    }).Upgrade(writer, request, nil)
    if err != nil {
        fmt.Println(err)
        return
    }
    //2.获取conn
    node := &Node{
        Conn:      conn,
        DataQueue: make(chan []byte, 50),
        GroupSets: set.New(set.ThreadSafe),
    }
    //3. 用户关系
    //4. userid 跟 node绑定 并加锁
    rwLocker.Lock()
    clientMap[userId] = node
    rwLocker.Unlock()
    //5.完成发送逻辑
    go sendProc(node)
    //6.完成接受逻辑
    go recvProc(node)
    sendMsg(userId, []byte("欢迎进入聊天系统"))

}

func sendProc(node *Node) {
    for {
        select {
        case data := <-node.DataQueue:
            err := node.Conn.WriteMessage(websocket.TextMessage, data)
            if err != nil {
                fmt.Println(err)
                return
            }
        }
    }
}

func recvProc(node *Node) {
    for {
        _, data, err := node.Conn.ReadMessage()
        if err != nil {
            fmt.Println(err)
            return
        }
        broadMsg(data)
        fmt.Println("[ws] <<<<< ", data)
    }
}
```

```
var udpsendChan chan []byte = make(chan []byte, 1024)

func broadMsg(data []byte) {
    udpsendChan <- data
}

func init() {
    go udpSendProc()
    go udpRecvProc()
}

//完成udp数据发送协程
func udpSendProc() {
    con, err := net.DialUDP("udp", nil, &net.UDPAddr{
        IP:   net.IPv4(192, 168, 0, 255),
        Port: 3000,
    })
    defer con.Close()
    if err != nil {
        fmt.Println(err)
    }
    for {
        select {
        case data := <-udpsendChan:
            _, err := con.Write(data)
            if err != nil {
                fmt.Println(err)
                return
            }
        }
    }

}

//完成udp数据接收协程
func udpRecvProc() {
    con, err := net.ListenUDP("udp", &net.UDPAddr{
        IP:   net.IPv4zero,
        Port: 3000,
    })
    if err != nil {
        fmt.Println(err)
    }
    defer con.Close()
    for {
        var buf [512]byte
        n, err := con.Read(buf[0:])
        if err != nil {
            fmt.Println(err)
```

```
            return
        }
        dispatch(buf[0:n])
    }
}

//后端调度逻辑处理
func dispatch(data []byte) {
    msg := Message{}
    err := json.Unmarshal(data, &msg)
    if err != nil {
        fmt.Println(err)
        return
    }
    switch msg.Type {
    case 1: //私信
        sendMsg(msg.TargetId, data)
        // case 2:  //群发
        //  sendGroupMsg()
        // case 3://广播
        //  sendAllMsg()
        //case 4:
        //
    }
}

func sendMsg(userId int64, msg []byte) {
    rwLocker.RLock()
    node, ok := clientMap[userId]
    rwLocker.RUnlock()
    if ok {
        node.DataQueue <- msg
    }
}
```

集成html 登录和注册

```
//app.go  加入
    //首页
    r.GET("/", service.GetIndex)
    r.GET("/index", service.GetIndex)
    r.GET("/toRegister", service.ToRegister)


// index.go
package service
```

```
import (
    "text/template"

    "github.com/gin-gonic/gin"
)

// GetIndex
// @Tags 首页
// @Success 200 {string} welcome
// @Router /index [get]
func GetIndex(c *gin.Context) {
    ind, err := template.ParseFiles("index.html",
"views/chat/head.html")
    if err != nil {
        panic(err)
    }
    ind.Execute(c.Writer, "index")
    // c.JSON(200, gin.H{
    //  "message": "welcome !!  ",
    // })
}

func ToRegister(c *gin.Context) {
    ind, err := template.ParseFiles("views/user/register.html")
    if err != nil {
        panic(err)
    }
    ind.Execute(c.Writer, "register")
    // c.JSON(200, gin.H{
    //  "message": "welcome !!  ",
    // })
}
```

然后页面：

```
<!DOCTYPE html>
<html>

<head>
    <!--js include-->
    {{template "/chat/head.shtml"}}
</head>
<body>

<header class="mui-bar mui-bar-nav">
    <h1 class="mui-title">登录</h1>
```

```
</header>
{{.}}
<div class="mui-content" id="pageapp">
    <form id='login-form' class="mui-input-group">
        <div class="mui-input-row">
            <label>账号</label>
            <input v-model="user.name" placeholder="请输入用户名"
type="text" class="mui-input-clear mui-input" >
        </div>
        <div class="mui-input-row">
            <label>密码</label>
            <input v-model="user.password" placeholder="请输入密码"
 type="password" class="mui-input-clear mui-input" >
        </div>
    </form>
    <div class="mui-content-padded">
        <button @click="login"  type="button"  class="mui-btn mui-btn-
block mui-btn-primary">登录</button>
        <div class="link-area"><a id='reg' href="/toRegister">注册账号</a>
<span class="spliter">|</span> <a  id='forgetPassword'>忘记密码</a>
        </div>
    </div>
    <div class="mui-content-padded oauth-area">
    </div>
</div>
</body>
</html>
<script>
    var app = new Vue({
        el:"#pageapp",
        data:function(){
          return {
              user:{
                name:"",
                password:"",
              }
          }
        },
        methods:{
            login:function(){
                //检测手机号是否正确
                console.log("login")
                //检测密码是否为空

                //网络请求
                //封装了promis

 util.post("user/findUserByNameAndPwd",this.user).then(res=>{
                    console.log(res)
```

```
                    if(res.code!=0){
                        mui.toast(res.message)
                    }else{
                         //
location.replace("//127.0.0.1/demo/index.shtml")
                        mui.toast("登录成功,即将跳转")
                       //
location.replace("//127.0.0.1/demo/index.shtml")
                    }
                })
            },
        }
    })
</script>
```

以及head.html

```
{{define "/chat/head.shtml"}}
<script>
    function userId(id){
        if(typeof  id =="undefined"){
            var r = sessionStorage.getItem("userid");
            if(!r){
                return 0;
            }else{
                return parseInt(r)
            }
        }else{
            sessionStorage.setItem("userid",id);
        }
    }
    function userInfo(o){
        if(typeof  o =="undefined"){
            var r = sessionStorage.getItem("userinfo");
            if(!!r){
                return JSON.parse(r);
            }else{
                return null
            }
        }else{
            sessionStorage.setItem("userinfo",JSON.stringify(o));
        }
    }
    var url = location.href;
    var isOpen = url.indexOf("/login")>-1 || url.indexOf("/register")>-1
    if (!userId() && !isOpen){
      // location.href = "login.shtml";
```

```
    }

</script>
    <!--登录所需 -->
    <link rel="stylesheet" href="/asset/css/login.css" />
    <!--聊天所需-->
<meta name="viewport" content="width=device-width, initial-
scale=1,maximum-scale=1,user-scalable=no">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<title>IM解决方案</title>
<meta name="Description" content="马士兵教育IM通信系统">
<meta name="Keywords" content="无人售货机，小程序，推送，群聊,单聊app">
<link rel="stylesheet" href="/asset/plugins/mui/css/mui.css" />
<link rel="stylesheet" href="/asset/css/chat.css" />
<link rel="stylesheet" href="/asset/css/audio.css" />
<script src="/asset/plugins/mui/js/mui.js" ></script>
<script src="/asset/js/vue.min.js" ></script>
<script src="/asset/js/util.js" ></script>
<script>
   function post(uri,data,fn){
                var xhr = new XMLHttpRequest();
                xhr.open("POST","//"+location.host+"/"+uri, true);
                // 添加http头，发送信息至服务器时内容编码类型
                xhr.setRequestHeader("Content-Type", "application/x-www-
form-urlencoded");
                xhr.onreadystatechange = function() {
                    if (xhr.readyState == 4 && (xhr.status == 200 ||
xhr.status == 304)) {
                        fn.call(this, JSON.parse(xhr.responseText));
                    }
                };
                var _data=[];
                if(!! userId()){
                   // data["userid"] = userId();
                }
                for(var i in data){
                    _data.push( i +"=" + encodeURI(data[i]));
                }
                xhr.send(_data.join("&"));
            }
            function uploadfile(uri,dom,fn){
                var xhr = new XMLHttpRequest();
                xhr.open("POST","//"+location.host+"/"+uri, true);
                // 添加http头，发送信息至服务器时内容编码类型
                xhr.onreadystatechange = function() {
                    if (xhr.readyState == 4 && (xhr.status == 200 ||
xhr.status == 304)) {
                        fn.call(this, JSON.parse(xhr.responseText));
```

```
                    }
                };
                var _data=[];
                var formdata = new FormData();
                if(!! userId()){
                    formdata.append("userid",userId());
                }
                formdata.append("file",dom.files[0])
                xhr.send(formdata);
            }
    function uploadblob(uri,blob,filetype,fn){
        var xhr = new XMLHttpRequest();
        xhr.open("POST","//"+location.host+"/"+uri, true);
        // 添加http头，发送信息至服务器时内容编码类型
        xhr.onreadystatechange = function() {
            if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status
== 304)) {
                fn.call(this, JSON.parse(xhr.responseText));
            }
        };
        var _data=[];
        var formdata = new FormData();
        formdata.append("filetype",filetype);
        if(!! userId()){
            formdata.append("userid",userId());
        }
        formdata.append("file",blob)
        xhr.send(formdata);
    }
        function uploadaudio(uri,blob,fn){
                uploadblob(uri,blob,".mp3",fn)
         }
        function uploadvideo(uri,blob,fn){
            uploadblob(uri,blob,".mp4",fn)
        }
</script>

<style>
    .flex-container{
        display:flex;
        flex-direction:row;
        width:100%;
        padding-top: 10px;
        position: fixed;
        bottom: 0px;
        background-color: #FFFFFF;
    }
    .item-1{
        height:50px;
```

```
    height:50px;
    padding: 5px 5px 5px 5px;
}
.item-2{
            margin-right:auto;
    height:50px;
    width: 100%;
}
.txt{
    margin-right:auto;
}
.item-3{
    height:50px;
    height:50px;
    padding: 5px 5px 5px 5px;
}
.item-4{
    height:50px;
    height:50px;
    padding: 5px 5px 5px 5px;
}

 li.chat{
     justify-content: flex-start;
     align-items: flex-start;
     display: flex;

 }
 .chat.other{
     flex-direction: row;
 }
.chat.mine{
    flex-direction: row-reverse;
}
img.avatar{
    width: 64px;
    height:64px;
}
.other .avatar{
    margin-left:10px;
}
.mine .avatar{
    margin-right:10px;
}
.other span{
    border: 10px solid;
    border-color: transparent #FFFFFF transparent transparent ;
    margin-top: 10px;
}
```

```
.mine span{
    border: 10px solid;
    border-color: transparent  transparent transparent #32CD32;
    margin-top: 10px;
}
.other>.content{
    background-color: #FFFFFF;

}
.mine>.content{
    background-color: #32CD32;

}
div.content{
    min-width: 60px;
    clear: both;
    display: inline-block;
    padding: 16px 16px 16px 10px;
    margin: 0 0 20px 0;
    font: 16px/20px 'Noto Sans', sans-serif;
    border-radius: 10px;

    min-height: 64px;
}
.content>img.pic{
    width: 100%;
    margin:3px 3px 3px 3px;
}
.content>img.audio{
    width: 32px;
    color: white;
}
#panels{
    background-color: #FFFFFF;
    display: flex;
    position: fixed;
    bottom: 50px;
}
.doutures{
    flex-direction: row;
    flex-wrap: wrap;
    display: flex;
}
.doutures img{
    margin: 10px 10px 10px 10px;
}
.doutupkg{
    flex-direction: row;
    flex-wrap: wrap;
```

```
        display: flex;
    }
    .plugins{
        flex-direction: row;
        flex-wrap: wrap;
        display: flex;
    }
    .plugin{
        padding: 10px 10px 10px 20px;
        margin-left: 10px;
        margin-right: 10px;
    }
    .plugin img{
        width: 40px;
    }
    .plugin p{
        text-align: center;
        font-size: 16px;
    }
    .doutupkg img{
        width: 32px;
        height: 32px;
        margin: 5px 5px 5px 5px;
    }
    .upload{
        width: 64px;
        height: 64px;
        position: absolute;
        top: 1px;
        opacity:0;
    }
    .tagicon{
        width: 32px;
        height:32px;
    }

    .small{
        width: 32px;
        height:32px;
    }
    .middle{
        width: 64px;
        height:64px;
    }
    .large{
        width: 96px;
        height:96px;
    }
    .res image{
```

```
        width: 32px;
        height:32px;
    }
</style>
{{end}}
```

集成聊天页面 完成 发送接受消息（文本）

```
前端需要拼接     Message对象
```

需要： 发送者ID ， 接受者ID ， 消息类型1，发送类型 1,发送的内容context token

jsonStr = JSON.stringify(msg)

websocket.send(jsonStr)

recvProc协程 读取数据

发送给对应的人

websocket.onMessage

```
//app.go  加入router
    r.GET("/toChat", service.ToChat)

//index.go 加入

func ToChat(c *gin.Context) {
    ind, err := template.ParseFiles("views/chat/index.html",
        "views/chat/head.html",
        "views/chat/foot.html",
        "views/chat/tabmenu.html",
        "views/chat/concat.html",
        "views/chat/group.html",
        "views/chat/profile.html",
        "views/chat/main.html")
    if err != nil {
        panic(err)
    }
    userId, _ := strconv.Atoi(c.Query("userId"))
    token := c.Query("token")
    user := models.UserBasic{}
    user.ID = uint(userId)
    user.Identity = token
    //fmt.Println("ToChat>>>>>>>>", user)
    ind.Execute(c.Writer, user)
```

```
    // c.JSON(200, gin.H{
    //  "message": "welcome !!  ",
    // })
}

//最后页面
index.html

<!DOCTYPE html>
<html>

<head>
    <!--js include-->
    {{template "/chat/head.shtml"}}
</head>
<body>

<header class="mui-bar mui-bar-nav">
    <h1 class="mui-title">登录</h1>
</header>
{{.}}
<div class="mui-content" id="pageapp">
    <form id='login-form' class="mui-input-group">
        <div class="mui-input-row">
            <label>账号</label>
            <input v-model="user.name" placeholder="请输入用户名"
type="text" class="mui-input-clear mui-input" >
        </div>
        <div class="mui-input-row">
            <label>密码</label>
            <input v-model="user.password" placeholder="请输入密码"
 type="password" class="mui-input-clear mui-input" >
        </div>
    </form>
    <div class="mui-content-padded">
        <button @click="login"  type="button"  class="mui-btn mui-btn-
block mui-btn-primary">登录</button>
        <div class="link-area"><a id='reg' href="/toRegister">注册账号</a>
<span class="spliter">|</span> <a  id='forgetPassword'>忘记密码</a>
        </div>
    </div>
    <div class="mui-content-padded oauth-area">
    </div>
</div>
</body>
</html>
<script>
    var app = new Vue({
        el:"#pageapp",
```

```
        data:function(){
          return {
              user:{
                name:"",
                password:"",
              }
          }
        },
        methods:{
            login:function(){
                //检测手机号是否正确
                console.log("login")
                //检测密码是否为空

                //网络请求
                //封装了promis

 util.post("user/findUserByNameAndPwd",this.user).then(res=>{
                    console.log(res)
                    if(res.code!=0){
                        mui.toast(res.message)
                    }else{
                        var url = "/toChat?
userId="+res.data.ID+"&token="+res.data.Identity
                        location.href = url
                        mui.toast("登录成功,即将跳转")
                    }
                })
            },
        }
    })
</script>


//views/chat/index.html
<!DOCTYPE html>
<html>
<head>
<!--js include-->
{{template "/chat/head.shtml"}}
</head>
<body>
<!--底部菜单-->
{{template "/chat/tabmenu.shtml"}}
<header class="mui-bar mui-bar-nav">
</header>
<div class="mui-content" id="pageapp">
    <!--联系人-->
    {{template "/chat/concat.shtml"}}
```

```
    <!--群聊-->
    {{template "/chat/group.shtml"}}
    <!--个人中心-->
    {{template "/chat/profile.shtml"}}
    <!--聊天主界面-->
    {{template "/chat/main.shtml"}}

</div>
</body>
</html>
{{template "/chat/foot.shtml"}}
```

测试登录成功之后正常跳转到聊提案首页

获取好友列表：

```
//app.go
    r.POST("/searchFriends", service.SearchFriends)

//userservice.go
func SearchFriends(c *gin.Context) {
    id, _ := strconv.Atoi(c.Request.FormValue("userId"))
    users := models.SearchFriend(uint(id))

    c.JSON(200, gin.H{
        "code":    0, //  0成功   -1失败
        "message": "查询好友列表成功！",
        "data":    users,
    })

}

//contact.go
func SearchFriend(userId uint) []UserBasic {
    contacts := make([]Contact, 0)
    objIds := make([]uint64, 0)
    utils.DB.Where("owner_id = ? and type=1", userId).Find(&contacts)
    for _, v := range contacts {
        fmt.Println(" >>>>>>>>>>>>>  ", v)
        objIds = append(objIds, uint64(v.TargetId))
    }
    users := make([]UserBasic, 0)
    utils.DB.Where("id in ?", objIds).Find(&users)
    return users
```

```
}
```

调试前端页面：

关键让后端的loadFrients在前端显示

```
{{define "/chat/foot.shtml"}}
<script>

    function upload(dom){
        uploadfile("attach/upload",dom,function(res){
            if(res.code==0){
                app.sendpicmsg(res.data)
            }

        })
    }
    function userId(){
        return parseInt(util.parseQuery("userId"))
    }
    var app=new Vue(
            {
                el:"#pageapp",
                data:{
                    usermap:{},
                    friends:[],
                    communitys:[],
                    profile:{
                        avatar:"",
                        nickname:"",
                        memo:"",
                    },
                    webSocket:{},
                    win:"main",
                    txtmsg:"",
                    panelstat:"kbord",
                    txtstat:"kbord",
                    title:"",
                    doutu:{
                        config:{
                            "baseurl":"/asset/plugins/doutu/",
                            "pkgids":["mkgif","emoj"]
                        },
                        packages:[],
```

```
                choosed:{"pkgid":"emoj","assets":
[],"size":"small"}
            },
            msglist:[],

            msgcontext:{
                dstid:10,
                cmd:10,
                userid:userId()
            },
            plugins:[
                {
                    icon:"/asset/images/upload.png",
                    name:"照片",
                    id:"upload",
                    slot:"<input
accept=\"image/gif,image/jpeg,,image/png\" type=\"file\"
onchange=\"upload(this)\" class='upload' />"
                },

                {
                    icon:"/asset/images/camera.png",
                    name:"拍照",
                    id:"camera",
                    slot:"<input accept=\"image/*\"
capture=\"camera\" type=\"file\" onchange=\"upload(this)\"
class='upload' />"
                },
                {
                    icon:"/asset/images/audiocall.png",
                    name:"语音",
                    id:"audiocall"
                },
                {
                    icon:"/asset/images/videocall.png",
                    name:"视频",
                    id:"videocall"
                },
                {
                    icon:"/asset/images/redpackage.png",
                    name:"红包",
                    id:"redpackage"
                },
                {
                    icon:"/asset/images/exchange.png",
                    name:"转账",
                    id:"exchange"
                },
                {
```

```
                        icon:"/asset/images/address.png",
                        name:"地址",
                        id:"address"
                    },
                    {
                        icon:"/asset/images/person.png",
                        name:"名片",
                        id:"person"
                    }

                ],
                timer:0,
                recorder:{},
                allChunks:[],
                iscomplete:false,
                duration:0,
                showprocess:false,
            },
            created:function(){
                this.loadfriends();
                this.loadcommunitys();
                this.loaddoutures();
                var user = userInfo()
                if(!!user){
                    this.profile.avatar = user.avatar;
                    this.profile.nickname = user.nickname;
                    this.profile.memo = user.memo;
                }

                this.initwebsocket()

            },
            mounted:function(){

            },
            methods:{
                playaudio:function(url){
                    document.getElementById('audio4play').src = url;
                    document.getElementById('audio4play').play();
                },
                startrecorder:function(){
                    let audioTarget =
document.getElementById('audio');
                    var types = ["video/webm",
                        "audio/webm",
                        "video/webm\;codecs=vp8",
                        "video/webm\;codecs=daala",
                        "video/webm\;codecs=h264",
                        "audio/webm\;codecs=opus",
```

```
                        "video/mpeg"];
                    var suporttype ="";
                    for (var i in types) {
                        if(MediaRecorder.isTypeSupported(types[i])){
                            suporttype = types[i];
                        }
                    }
                    if(!suporttype){
                        mui.toast("编码不支持")
                        return ;
                    }

                    this.duration = new Date().getTime();
                    navigator.mediaDevices.getUserMedia({audio:
true, video: false})
                            .then(function(stream){
                                this.showprocess = true
                                this.recorder = new
MediaRecorder(stream);
                                audioTarget.srcObject = stream;

                                this.recorder.ondataavailable =
(event) => {
                                    console.log("ondataavailable");

 uploadblob("attach/upload",event.data,".mp3",res=>{
                                        var duration =
Math.ceil((new Date().getTime()-this.duration)/1000);

 this.sendaudiomsg(res.data,duration);
                                    })

 stream.getTracks().forEach(function (track) {
                                        track.stop();
                                    });
                                    this.showprocess = false
                                }
                                this.recorder.start();
                            }.bind(this)).
                    catch(function(err){
                        console.log(err)
                        mui.toast(err)
                        this.showprocess = false
                    }.bind(this));
                },
                stoprecorder :function() {
                    if(typeof this.recorder.stop=="function"){
                        this.recorder.stop();
                    }
```

```
                        this.showprocess = false
                        console.log("stoprecorder")

                    },
                    dispatchplugin:function(item){
                        switch (item.id) {
                            case "upload":
                            case "camera":

                                break;
                            default:
                                mui.toast("系统暂不支持,请自行扩展")
                        }
                    },
                    reset:function(){
                        this.panelstat="kbord";
                        this.txtstat="kbord";
                        this.txtmsg = "";
                    },
                    createmsgcontext:function(){
                        return
JSON.parse(JSON.stringify(this.msgcontext))
                    },
                    loaddoutures:function(){
                        var res=[];
                        var config = this.doutu.config;
                        for(var i in config.pkgids){
                            res[config.pkgids[i]]=
(config.baseurl+"/"+config.pkgids[i]+"/info.json")
                        }
                        var that = this;
                        for(var id in res){
                            //console.log("res[i]",id,res[id])
                            post(res[id],{},function(pkginfo){
                                //console.log("post
res[i]",id,res[id],pkginfo)
                                var baseurl=
config.baseurl+"/"+pkginfo.id+"/"
                                for(var j in pkginfo.assets){
                                    pkginfo.assets[j] =
baseurl+pkginfo.assets[j];
                                }
                                pkginfo.icon = baseurl + pkginfo.icon;
                                that.doutu.packages.push(pkginfo)
                                if(that.doutu.choosed.pkgid==pkginfo.id)
{

 that.doutu.choosed.assets=pkginfo.assets;
                                }
```

```
                    })
                }
            },
            showweixin:function(){
                mui.alert("请加微信号jiepool-winlion索取")
            } ,
            showmsg:function(user,msg){
                var data={

                }
                data.ismine = userId()==msg.userid;
                //console.log(data.ismine,userId(),msg.userid)
                data.user = user;
                data.msg = msg;
                this.msglist = this.msglist.concat(data)
                this.reset();
                var that =this;
                that.timer = setTimeout(function(){
                    window.scrollTo(0,
document.getElementById("convo").offsetHeight);
                    clearTimeout(that.timer)
                },100)

            },
            startrecord:function(){

            },
            sendtxtmsg:function(txt){

 //{id:1,userid:2,dstid:3,cmd:10,media:1,content:"hello"}
                var msg =this.createmsgcontext();
                msg.media=1;
                msg.content=txt;
                this.showmsg(userInfo(),msg);
                this.webSocket.send(JSON.stringify(msg))
            },
            sendpicmsg:function(picurl){

 //{id:1,userid:2,dstid:3,cmd:10,media:4,url:"http://www.baidu.com/a/log
,jpg"}
                var msg =this.createmsgcontext();
                msg.media=4;
                msg.url=picurl;
                this.showmsg(userInfo(),msg)
                this.webSocket.send(JSON.stringify(msg))
            },
            sendaudiomsg:function(url,num){
```

```
 //{id:1,userid:2,dstid:3,cmd:10,media:3,url:"http://www.a,com/dsturl.mp
3",anount:40}
                        var msg =this.createmsgcontext();
                        msg.media=3;
                        msg.url=url;
                        msg.amount = num;
                        this.showmsg(userInfo(),msg)
                        //console.log("sendaudiomsg",this.msglist);
                        this.webSocket.send(JSON.stringify(msg))
                    },
                    singlemsg:function(user){
                        //console.log(user)
                        this.win = "single";
                        this.title = "和"+user.Name+"聊天中";
                        this.msgcontext.dstid = parseInt(user.ID);
                        this.msgcontext.cmd = 1;
                    },
                    groupmsg:function(group){

                        this.win = "group";
                        this.title=group.name;
                        this.msgcontext.dstid = parseInt(group.id);
                        this.msgcontext.cmd = 11;
                    },
                    loaduserinfo:function(userid,cb){
                        userid = ""+userid;
                        var userinfo = this.usermap[userid];
                        if(!userinfo){
                            post("user/find",
{id:parseInt(userid)},function(res){
                                cb(res.data);
                                this.usermap[userid] = res.data;
                            }.bind(this))
                        }else{
                            cb(userinfo)
                        }
                    },
                    onmessage:function(data){

                        this.loaduserinfo(data.userid,function(user){
                            this.showmsg(user,data)
                        }.bind(this))

                    },
                    initwebsocket:function(){
                        var
url="ws://"+location.host+"/user/sendUserMsg?id="+userId()+"&token="
+util.parseQuery("token");
```

```
                this.webSocket=new WebSocket(url);
                //消息处理
                this.webSocket.onmessage = function(evt){
                    //{"data":"}",...}
                    if(evt.data.indexOf("}")>-1){
                        this.onmessage(JSON.parse(evt.data));
                    }else{
                        console.log("recv<=="+evt.data)
                    }
                }.bind(this)
                //关闭回调
                this.webSocket.onclose=function (evt) {
                    console.log(evt.data)
                }
                //出错回调
                this.webSocket.onerror=function (evt) {
                    console.log(evt.data)
                }
                /*{
                    this.webSocket.send()
                }*/
            },
            sendmsg:function(){

            },
            loadfriends:function(){
                var that = this;
                post("searchFriends",
{userId:userId()},function(res){
                    that.friends = res.Rows ||[];
                    var usermap = this.usermap;
                    for(var i in res.Rows){
                        var k = ""+res.Rows[i].ID
                        usermap[k]=res.Rows[i];
                    }
                    this.usermap = usermap;
                }.bind(this))
            },
            loadcommunitys:function(){
                var that = this;
                post("contact/loadcommunity",
{userid:userId()},function(res){
                    that.communitys = res.rows ||[];
                })
            },
            addfriend:function(){
                var that = this;
                //prompt
```

```
                        mui.prompt('','请输入好友ID','加好友',['取消','确
认'],function (e) {
                            if (e.index == 1) {
                                if (isNaN(e.value) || e.value <= 0) {
                                    mui.toast('格式错误');
                                }else{
                                    //mui.toast(e.value);
                                    that._addfriend(e.value)
                                }
                            }else{
                                //mui.toast('您取消了入库');
                            }
                        },'div');
                        document.querySelector('.mui-popup-input
input').type = 'number';
                    },
                    _addfriend:function(dstobj){
                        var that = this
                        post("contact/addfriend",{dstid:dstobj,userid:
userId()},function(res){
                            if(res.code==0){
                                mui.toast("添加成功");
                                that.loadfriends();
                            }else{
                                mui.toast(res.msg);
                            }
                        })
                    },
                    _joincomunity:function(dstobj){
                        var that = this;
                        post("contact/joincommunity",
{dstid:dstobj,"userid":userId()},function(res){
                            if(res.code==0){
                                mui.toast("添加成功");

                                that.loadcommunitys();
                            }else{
                                mui.toast(res.msg);
                            }
                        })
                    },
                    joincomunity:function(){
                        var that = this;
                        mui.prompt('','请输入群号','加群',['取消','确
认'],function (e) {
                            if (e.index == 1) {
                                if (isNaN(e.value) || e.value <= 0) {
                                    mui.toast('格式错误');
                                }else{
```

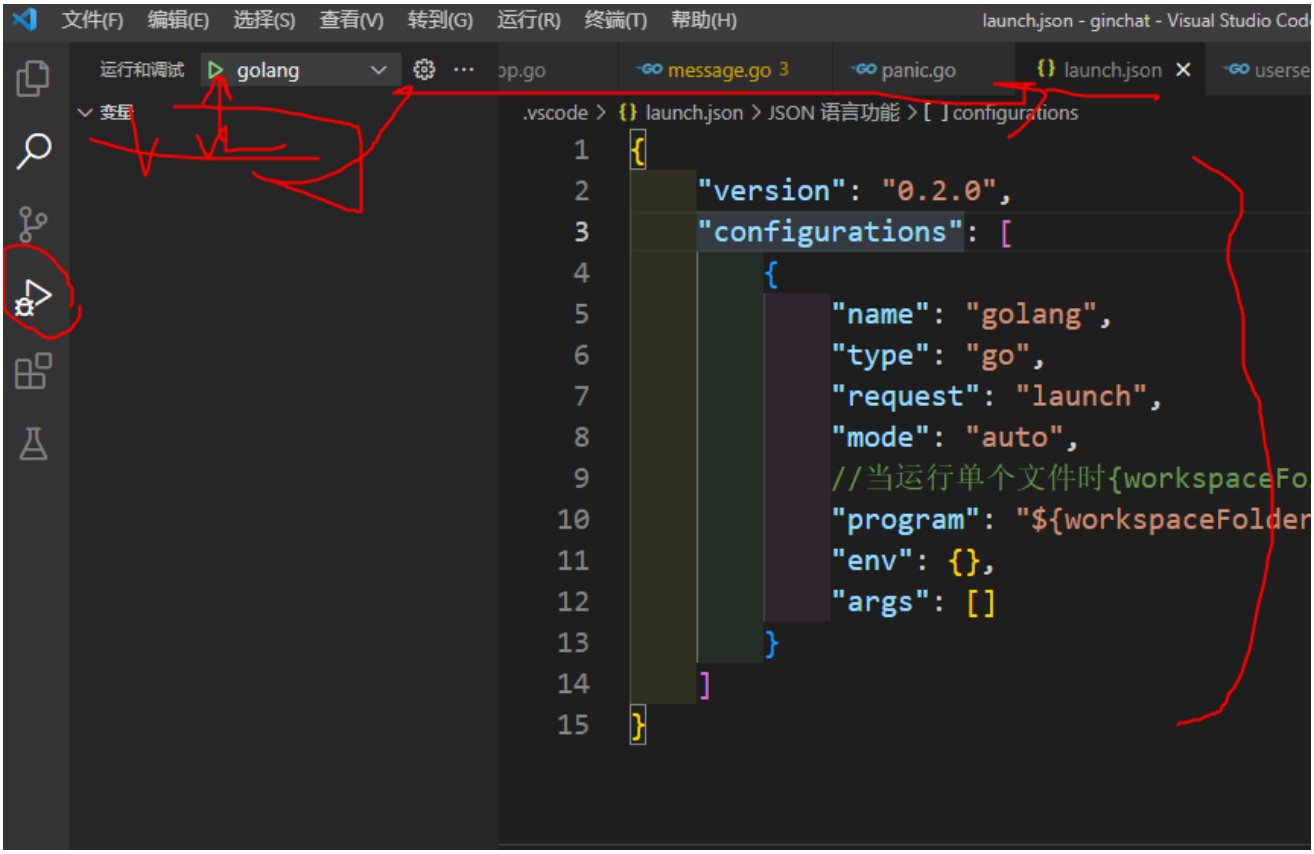
```
                                    //mui.toast(e.value);
                                    that._joincomunity(e.value)
                                }
                            }else{
                                //mui.toast('您取消了入库');
                            }
                        },'div');
                        document.querySelector('.mui-popup-input
input').type = 'number';
                    },
                    quit:function () {
                        sessionStorage.removeItem("userid")
                        sessionStorage.removeItem("userinfo")
                        location.href="login.shtml"
                    }


                },
                watch:{
                    "win":function(n,o){
                       // console.log("watch",o,n)
                        if(n!="main"){

 document.getElementById("menubar").style.display="none";
                        }else{

 document.getElementById("menubar").style.display="block";
                        }
                    }
                }
            }
    )
</script>
{{end}}
```

请求的返回改成封装的类型：

```
func SearchFriends(c *gin.Context) {
    id, _ := strconv.Atoi(c.Request.FormValue("userId"))
    users := models.SearchFriend(uint(id))

    // c.JSON(200, gin.H{
    //  "code":    0, //  0成功   -1失败
    //  "message": "查询好友列表成功！",
    //  "data":    users,
    // })
    utils.RespOKList(c.Writer, users, len(users))
}
```

调试前后端 ，首先通过页面和postman测试，再调试前端页面，然后再post到后台，确保发送正常之后调试前端显示。

开启Debug调试




```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "golang",
            "type": "go",
            "request": "launch",
            "mode": "auto",
            //当运行单个文件时{workspaceFolder}可改为{file}
            "program": "${workspaceFolder}",
            "env": {},
            "args": []
        }
    ]
}
```

调试发送和接收； 修改foot.html页面参数传递 id 》 userId

```
initwebsocket:function(){

            var url="ws://"+location.host+"/chat?
userId="+userId()+"&token=" +util.parseQuery("token");
```

修改页面main.html 得判断逻辑

```
 <li class="chat " :class="item.ismine?'mine':'other'" v-for="item in
msglist"  v-if="(item.ismine? item.msg.TargetId==msgcontext.TargetId :
item.msg.userId==msgcontext.TargetId )"  >
```

git 版本控制

https://git-scm.com/

下载安装包， 并且集成 git history插件

完成前端页面加载 表情包的　引入 vue-resource.min.js

```
通过 this.$http.get(res[id]).then( response => {

            pkginfo = response.data

               var baseurl= config.baseurl+"/"+pkginfo.id+"/"

              // console.log("post res[i]",id,res[id],pkginfo)
```

并调整显示 判断 Media=4 的时候

```
   <div v-if="item.msg.Media==3" @tap="playaudio(item.msg.url)">
                <img class="audio" src="/asset/images/audiom.png" />
                <span v-text="item.msg.amount"></span>
            </div>
```

完成 图片发送的后端代码

```
package service

import (
    "fmt"
    "ginchat/utils"
    "io"
    "math/rand"
    "os"
    "strings"
    "time"

    "github.com/gin-gonic/gin"
)

func Upload(c *gin.Context) {
    w := c.Writer
    req := c.Request
    srcFile, head, err := req.FormFile("file")
    if err != nil {
        utils.RespFail(w, err.Error())
    }
    suffix := ".png"
    ofilName := head.Filename
    tem := strings.Split(ofilName, ".")
    if len(tem) > 1 {
        suffix = "." + tem[len(tem)-1]
    }
    fileName := fmt.Sprintf("%d%04d%s", time.Now().Unix(), rand.Int31(),
suffix)
    dstFile, err := os.Create("./asset/upload/" + fileName)
    if err != nil {
        utils.RespFail(w, err.Error())
    }
    _, err = io.Copy(dstFile, srcFile)
    if err != nil {
        utils.RespFail(w, err.Error())
    }
    url := "./asset/upload/" + fileName
    utils.RespOK(w, url, "发送图片成功")
}

    r.POST("/attach/upload", service.Upload)
```

前端： 1.上传这个图片

2.发送一条消息 url 图片地址即可

```
  function upload(dom){

    uploadfile("attach/upload",dom,function(res){

        if(res.Code==0){

        app.sendpicmsg(res.Data)

        }
   })

  }

sendpicmsg:function(picurl){

 //{id:1,userid:2,dstid:3,cmd:10,media:4,url:"http://www.baidu.com/a/log
,jpg"}
                        var msg =this.createmsgcontext();
                        msg.Media=4;
                        msg.url=picurl;
                        this.showmsg(userInfo(),msg)
                        this.webSocket.send(JSON.stringify(msg))
},
```

语音发送：

recorder

```
 this.duration = new Date().getTime();
                        //video 摄像头   ，audio 音频
                        navigator.mediaDevices.getUserMedia({audio:
true, video: false})
                                    .then(function(stream){
                                        this.showprocess = true
                                        this.recorder = new
MediaRecorder(stream);
                                        audioTarget.srcObject = stream;
                                        //是否可用
                                        this.recorder.ondataavailable =
(event) => {
```

```
                                        console.log("ondataavailable");

    uploadblob("attach/upload",event.data,".mp3",res=>{
                                            var duration =
    Math.ceil((new Date().getTime()-this.duration)/1000);

    this.sendaudiomsg(res.Data,duration);
                                        })

    stream.getTracks().forEach(function (track) {
                                            track.stop();
                                        });
                                        this.showprocess = false
                                    }
                                    this.recorder.start();
                                }.bind(this)).
                            catch(function(err){
                                console.log(err)
                                mui.toast(err)
                                this.showprocess = false
                            }.bind(this));
                        },
```

群聊的功能。

原理分析

方案一： map <qunId,qunId,,,,,> 以用户基准

优点：锁的频率较低

缺点：需要轮询全部map

type Node struct {

    Conn *websocket .Conn

    DataQueue chan [] byte

   GroupSets set.Interface

}

var clientMap map[int64] *Node = make().....

方案二：

map <userId,userId,,,,,> 以群为ID

优点：查询效率会更快

缺点：发送消息需要根据用户ID获取Node，锁的频率较高

代码落地：

1.新建群　初始化groupSet

```
    2. 加入群　　刷新groupSet
    3. 分发消息（群里面的人都要收到）
```

添加好友

models

```
//查找某个用户
func FindUserByID(name string) UserBasic {
    user := UserBasic{}
    utils.DB.Where("name = ?", name).First(&user)
    return user
}
```

server层：

```
func AddFriend(c *gin.Context) {
    userId, _ := strconv.Atoi(c.Request.FormValue("userId"))
    targetId, _ := strconv.Atoi(c.Request.FormValue("targetId"))
    code := models.AddFriend(uint(userId), uint(targetId))
    // c.JSON(200, gin.H{
    //  "code":    0, //  0成功   -1失败
    //  "message": "查询好友列表成功！",
    //  "data":    users,
    // })
    if code == 0 {
        utils.RespOK(c.Writer, code, "添加成功")
    } else {
        utils.RespFail(c.Writer, "添加失败")
    }

}
```

router层：

```
    r.POST("/attach/upload", service.Upload)
```

前端：

```
 _addfriend:function(dstobj){
                        var that = this
                        post("contact/addfriend",
{targetId:dstobj,userId: userId()},function(res){
                            if(res.Code==0){
                                mui.toast("添加成功");
                                that.loadfriends();
                            }else{
                                mui.toast(res.Msg);
                            }
                        })
                    },
```

加入事务:

```
//添加好友
func AddFriend(userId uint, targetId uint) int {
    user := UserBasic{}
    if targetId != 0 {
        user = FindByID(targetId)
        fmt.Println(targetId, "          ", userId)
        if user.Salt != "" {
            tx := utils.DB.Begin()
            //事务一旦开始，不论什么异常最终都会Rollback
            defer func() {
                if r := recover(); r != nil {
                    tx.Rollback()
                }
            }()
            contact := Contact{}
            contact.OwnerId = userId
            contact.TargetId = targetId
            contact.Type = 1
            if err := utils.DB.Create(&contact).Error; err != nil {
                tx.Rollback()
                return -1
            }
            contact1 := Contact{}
            contact1.OwnerId = targetId
            contact1.TargetId = userId
            contact1.Type = 1
            if err := utils.DB.Create(&contact1).Error; err != nil {
                tx.Rollback()
                return -1
```

```
            }
            tx.Commit()
            return 0
        }
        return -1
    }
    return -1
}
```

考虑是否自己

和重复添加的问题

```
//添加好友
func AddFriend(userId uint, targetId uint) (int, string) {
    user := UserBasic{}
    if targetId != 0 {
        user = FindByID(targetId)
        fmt.Println(targetId, "          ", userId)
        if user.Salt != "" {
            if userId == user.ID {
                return -1, "不能加自己"
            }
            contact0 := Contact{}
            utils.DB.Where("owner_id =?  and target_id =? and type=1",
userId, targetId).Find(&contact0)
            if contact0.ID != 0 {
                return -1, "不能重复添加"
            }
            tx := utils.DB.Begin()
            //事务一旦开始，不论什么异常最终都会Rollback
            defer func() {
                if r := recover(); r != nil {
                    tx.Rollback()
                }
            }()
            contact := Contact{}
            contact.OwnerId = userId
            contact.TargetId = targetId
            contact.Type = 1
            if err := utils.DB.Create(&contact).Error; err != nil {
                tx.Rollback()
                return -1, "添加好友失败"
            }
            contact1 := Contact{}
            contact1.OwnerId = targetId
```

```
            contact1.TargetId = userId
            contact1.Type = 1
            if err := utils.DB.Create(&contact1).Error; err != nil {
                tx.Rollback()
                return -1, "添加好友失败"
            }
            tx.Commit()
            return 0, "添加好友成功"
        }
        return -1, "没有找到此用户"
    }
    return -1, "好友ID不能为空"
}


server层:
func AddFriend(c *gin.Context) {
    userId, _ := strconv.Atoi(c.Request.FormValue("userId"))
    targetId, _ := strconv.Atoi(c.Request.FormValue("targetId"))
    code, msg := models.AddFriend(uint(userId), uint(targetId))
    if code == 0 {
        utils.RespOK(c.Writer, code, msg)
    } else {
        utils.RespFail(c.Writer, msg)
    }
}
```

群管理

新建群

```
models层

package models

import (
    "fmt"
    "ginchat/utils"

    "gorm.io/gorm"
)

type Community struct {
    gorm.Model
    Name    string
    OwnerId uint
    Img     string
```

```
    Desc    string
}

func CreateCommunity(community Community) (int, string) {
    if len(community.Name) == 0 {
        return -1, "群名称不能为空"
    }
    if community.OwnerId == 0 {
        return -1, "请先登录"
    }
    if err := utils.DB.Create(&community).Error; err != nil {
        fmt.Println(err)
        return -1, "建群失败"
    }
    return 0, "建群成功"
}

server层 ：
func CreateCommunity(c *gin.Context) {
    ownerId, _ := strconv.Atoi(c.Request.FormValue("ownerId"))
    name := c.Request.FormValue("name")
    community := models.Community{}
    community.OwnerId = uint(ownerId)
    community.Name = name
    code, msg := models.CreateCommunity(community)
    if code == 0 {
        utils.RespOK(c.Writer, code, msg)
    } else {
        utils.RespFail(c.Writer, msg)
    }
}

router层
    //创建群
    r.POST("/contact/createCommunity", service.CreateCommunity)
```

前端新建群

websocket 单页面聊天 如果跳转到其他地方 1001错误码 。

1.先将createcom.html引入到 chat 聊天页面

index.go ToChat方法中加入

template.ParseFiles(引入模板的时候 加入 "views/chat/createcom.html",

2.index.html 引入模块页面

{{template "/chat/createcom.shtml"}}

3.profile.html 页面点击 新建群的修改

```
<li  @click="createCom" class="mui-table-view-cell">
            <a class="mui-navigate-right" >
        创建社群
      </a>
    </li>
```

4.到foot.hmtl中加入 createCom的方法

```
//新建群显示
                    createCom:function(){
                        this.win ="community"
                        //console.log("createCom")
                    },
```

5.在createcom.html 包一层

head div ....

6.先注释掉js（有很多问题），然后再到foot.html 加入 createcommunity

```
//新建群提交
                    createcommunity (){
                        console.log("createcommunity")
                    },
```

7.发现还是报错 com 就在 foot.html 中初始化

```
        com:{
                        "icon":"",
                        "cate":"",
                        "name":"",
                        "memo":"",
         },
```

8.删除createcom.html 不起效果的 样式 重新到head.html中加入

```
   .mui-content {
                padding-top: 44px;
                position: absolute;
                left: 0;
                top: 0;
                background: #fff;
                width: 100%;
                height: 100%;
        }
```

9.最后回到上一步的修复 。

```
            在createcom.html中加入 @click的点击事件
                <a class="mui-icon mui-icon-left-nav mui-pull-left"
@click="goBack()"></a>

        在foot.html中加入
            //回到聊天首页
                    goBack(){
                        this.win="main"
                    },
```

完成建群前后端联调:

```
createcom.html 的表单
<div class="mui-input-group">


            <div class="mui-input-row">
                <label>类型</label>
                <select v-model="com.cate" class="mui-input-clear mui-
input">
                    <option value="0">默认</option>
                    <option value="1">兴趣爱好</option>
```

```
                    <option value="2">行业交流</option>
                    <option value="3">生活休闲</option>
                    <option value="4">学习考试</option>
                </select>
            </div>
            <div class="mui-input-row">
                <label>名称</label>
                <input v-model="com.name" id='mobile' type="text"
class="mui-input-clear mui-input" placeholder="请输入群名称">
            </div>
            <div class="mui-input-row">
                <label>介绍</label>
                <textarea v-model="com.memo" id="textarea" rows="3"
placeholder="群描述"></textarea>
            </div>


        </div>
        <div class="mui-content-padded">
            <button @click="createcommunity" id='reg' class="mui-btn
mui-btn-block mui-btn-primary">确认</button>
        </div>

    </div>
```

foot.html中定义com对象加一个ownerId

```
com:{
                        "ownerId":"",
                        "icon":"",
                        "cate":"",
                        "name":"",
                        "memo":"",
                    },
```

并且修改 提交的方法

```
  //新建群提交
                    createcommunity (){
                        //console.log("createcommunity")
                        this.com.ownerId=  userId()
                        console.log(this.com)

 util.post("/contact/createCommunity",this.com).then(res=>{
                            console.log(res)
                            if(res.code!=0){
                                mui.toast(res.Msg)
                            }else{
```

```
                                //location.replace("localhost:8081")
                                //location.href = "/"
                                mui.toast("建群成功,即将跳转")
                                goBack()
                            }
                        })
                    },
```

群列表:

router层:

```
    //群列表
    r.POST("/contact/loadcommunity", service.LoadCommunity)
```

server层

```
//加载群列表
func LoadCommunity(c *gin.Context) {
    ownerId, _ := strconv.Atoi(c.Request.FormValue("ownerId"))
    //  name := c.Request.FormValue("name")
    data, msg := models.LoadCommunity(uint(ownerId))
    if len(data) != 0 {
        utils.RespList(c.Writer, 0, data, msg)
    } else {
        utils.RespFail(c.Writer, msg)
    }
}
```

models层:

```
func LoadCommunity(ownerId uint) ([]*Community, string) {
    data := make([]*Community, 10)
    utils.DB.Where("owner_id = ? ", ownerId).Find(&data)
    for _, v := range data {
        fmt.Println(v)
    }
    //utils.DB.Where()
    return data, "查询成功"
}
```

foot.html页面

```
func LoadCommunity(ownerId uint) ([]*Community, string) {
```

```
    data := make([]*Community, 10)
    utils.DB.Where("owner_id = ? ", ownerId).Find(&data)
    for _, v := range data {
        fmt.Println(v)
    }
    //utils.DB.Where()
    return data, "查询成功"
}
```

加入群:

```

```

点击一次 发起两次的问题（禁用高频发送）

尝试换 this.winA

尝试 换 click.once

正确解决方案:

```
在 data 定义一个 isDisable = true


_xxx的方法里面   先判断
    if(this.isDisable) {
                                this.setTimeFlag();

方法的封装 :
    setTimeFlag(){
                            this.isDisable = false;
                                    setTimeout(()=>{
                                        this.isDisable = true;
                                    },100 )
                        }
```

然后将发送消息 图片等方法也加上这个判断

loge图片的加入 ico

引入favicon.ico文件 之后 router里面加上静态资源 r.StaticFile("/favicon.ico", "asset/images/favicon.ico")

再到页面 head.html

群聊消息后端

首先新建群 的放修改 加入事务 新增群表同时新增关系表

```
func CreateCommunity(community Community) (int, string) {
    tx := utils.DB.Begin()
    //事务一旦开始，不论什么异常最终都会 Rollback
    defer func() {
        if r := recover(); r != nil {
            tx.Rollback()
        }
    }()

    if len(community.Name) == 0 {
        return -1, "群名称不能为空"
    }
    if community.OwnerId == 0 {
        return -1, "请先登录"
    }
    if err := utils.DB.Create(&community).Error; err != nil {
        fmt.Println(err)
        tx.Rollback()
        return -1, "建群失败"
    }
    contact := Contact{}
    contact.OwnerId = community.OwnerId
    contact.TargetId = community.ID
    contact.Type = 2 //群关系
    if err := utils.DB.Create(&contact).Error; err != nil {
        tx.Rollback()
        return -1, "添加群关系失败"
    }
```

```
    tx.Commit()
    return 0, "建群成功"

}
```

加入通过群找到群人员的方法

```
func SearchUserByGroupId(communityId uint) []uint {
    contacts := make([]Contact, 0)
    objIds := make([]uint, 0)
    utils.DB.Where("target_id = ? and type=2",
communityId).Find(&contacts)
    for _, v := range contacts {
        objIds = append(objIds, uint(v.OwnerId))
    }
    return objIds
}
```

最后处理消息的时候判断2 群发

```
func sendGroupMsg(targetId int64, msg []byte) {
    fmt.Println("开始群发消息")
    userIds := SearchUserByGroupId(uint(targetId))
    for i := 0; i < len(userIds); i++ {
        sendMsg(int64(userIds[i]), msg)
    }
}
```

页面的修改

```
                    groupmsg:function(group){
                        if(this.isDisable) {
                            this.setTimeFlag()
                            this.win = "group";
                            this.title=group.Name;
                            this.msgcontext.TargetId =
parseInt(group.ID);
                            this.msgcontext.Type = 2;
                        }


                                //新建群提交
                    createcommunity (){
                        //console.log("createcommunity")
                        this.com.ownerId=  userId()
                        console.log(this.com)
```

```
 util.post("/contact/createCommunity",this.com).then(res=>{
                                console.log(res)
                                if(res.Code!=0){
                                    mui.toast(res.Msg)
                                }else{
                                    //location.replace("localhost:8081")
                                    //location.href = "/"
                                    mui.toast("建群成功")
                                    this.loadcommunitys();
                                    //goBack()
                                }
                            })
                        },
```

前端显示消息：

首先加入 头像字段

router加入

r.POST("/user/find", service.FindByID)

server层

```
func FindByID(c *gin.Context) {

  userId, _ := strconv.Atoi(c.Request.FormValue("userId"))



  //  name := c.Request.FormValue("name")

  data := models.FindByID(uint(userId))

  utils.RespOK(c.Writer, data, "ok")

}  //dao层之前已写好
```

页面 foot.html的js

```
loaduserinfo:function(userid,cb){
                            userid = ""+userid;
                            console.log(">>>> "+userid)
                            var userinfo = this.usermap[userid];
                            if(!userinfo){
```

```
                                post("user/find",
{userId:parseInt(userid)},function(res){
                                    cb(res.Data);
                                    this.usermap[userid] = res.Data;
                                }.bind(this))
                            }else{
                                cb(userinfo)
                            }
                        },
                        onmessage:function(data){
                            this.loaduserinfo(data.userId,function(user){
                                if  (userId()!=data.userId ) {
                                    this.showmsg(user,data)
                                }

                            }.bind(this))

                        },
```

以及main.html的判断显示

```
        <li class="chat " :class="item.ismine?'mine':'other'" v-
for="item in msglist"
        v-if=" item.msg.Type==msgcontext.Type
            && ((item.msg.Type==1) && (item.ismine?
item.msg.TargetId==msgcontext.TargetId :
item.msg.userId==msgcontext.TargetId ))
            || ((item.msg.Type==2) && (
item.msg.TargetId==msgcontext.TargetId ))
            "  >
```

性能调优： 静态资源的分离

文件会比较多 （磁盘的IO）。 阿里云OOS (Object Storage Service) 海量，安全，低成本，高速，可靠 云存储。

OOS API：http://doc.oss.aliyuncs.com/#_Toc336676738

登录阿里云 https://oss.console.aliyun.com/bucket

参看代码：https://help.aliyun.com/document_detail/88601.htm?spm=a2c4g.11186623.0.0.1d8f2cb7wXZMVL#section-4yj-fxf-vaj

Key socket ： https://ram.console.aliyun.com/manage/ak?spm=a2c8b.12215442.top-nav.dak.18fb336aZdJlJ4

代码实现：

引入 oos的包

go get github.com/aliyun/aliyun-oss-go-sdk/oss

1.配置 四个key

1 封装一个

```
func Upload(c *gin.Context) {
    UploadOOS(c)
}
```

重新加一个 UploadOOS的方法 ，将原来Upload改成 UploadLocal

```
//上传文件到阿里云
func UploadOOS(c *gin.Context) {
    w := c.Writer
    req := c.Request
    srcFile, head, err := req.FormFile("file")
    if err != nil {
        utils.RespFail(w, err.Error())
    }
    suffix := ".png"
    ofilName := head.Filename
    tem := strings.Split(ofilName, ".")
    if len(tem) > 1 {
        suffix = "." + tem[len(tem)-1]
    }
    fileName := fmt.Sprintf("%d%04d%s", time.Now().Unix(), rand.Int31(),
suffix)
    //utils.Oos.AccessKeyId
    client, err := oss.New(viper.GetString("oos.EndPoint"),
viper.GetString("oos.AccessKeyId"),
viper.GetString("oos.AccessKeySecret"))
    if err != nil {
        fmt.Println("oos new failed : ", err)
        os.Exit(-1)
```

```
    }
    // 填写存储空间名称，例如examplebucket。
    bucket, err := client.Bucket(viper.GetString("oos.Bucket"))
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(-1)
    }
    err = bucket.PutObject(fileName, srcFile)
    if err != nil {
        fmt.Println("Error:", err)
        utils.RespFail(w, err.Error())
        os.Exit(-1)
    }
    //上传本地的逻辑
    // dstFile, err := os.Create("./asset/upload/" + fileName)
    // if err != nil {
    //  utils.RespFail(w, err.Error())
    // }
    // _, err = io.Copy(dstFile, srcFile)
    // if err != nil {
    //  utils.RespFail(w, err.Error())
    // }

    url := "http://" + viper.GetString("oos.Bucket") + "." +
viper.GetString("oos.EndPoint") + "/" + fileName
    utils.RespOK(w, url, "发送图片成功")
}
```

性能优化之 心跳检测

websocket 长连接，在用户特别多情况下。不在线的情况（做心跳检测）移除推送消息的队列。

实现：

前端：过一段时间检测一下 看是否还在。　　生命时长 6分钟

方式1：页面定时（30秒）发送一个请求 更新 生命时长。　在线用户列表 加入 NoSQL（Redis） 在加入在线用户的时候给他一个生命时长（12分钟）

方式2：只有页面有操作 才做心跳检测

后端：请求方法

更新心跳时间，判断 心跳时间是否超过最大时长 就判定为离线

后端检测下线:

```
yml配置
timeout:
  DelayHeartbeat: 3     //首次延迟多久检测
  HeartbeatHz: 6          //检测频率
  HeartbeatMaxTime: 30   //最大超时 就下线


 message.go修改Node
  type Node struct {
    Conn          *websocket.Conn //连接
    Addr          string          //客户端地址
    FirstTime     uint64          //首次连接时间
    HeartbeatTime uint64          //心跳时间
    LoginTime     uint64          //登录时间
    DataQueue     chan []byte     //消息
    GroupSets     set.Interface   //好友 / 群
}


并且在建立连接时：
    node := &Node{
        Conn:          conn,
        Addr:          conn.RemoteAddr().String(), //客户端地址
        HeartbeatTime: currentTime,                //心跳时间
        LoginTime:     currentTime,                //登录时间
        DataQueue:     make(chan []byte, 50),
        GroupSets:     set.New(set.ThreadSafe),
    }

 并加入

//更新用户心跳
func (node *Node) Heartbeat(currentTime uint64) {
    node.HeartbeatTime = currentTime
    return
}

//清理超时连接
func CleanConnection(param interface{}) (result bool) {
    result = true
    defer func() {
        if r := recover(); r != nil {
            fmt.Println("cleanConnection err", r)
        }
```

```
    }()
    fmt.Println("定时任务,清理超时连接 ", param)
    //node.IsHeartbeatTimeOut()
    currentTime := uint64(time.Now().Unix())
    for i := range clientMap {
        node := clientMap[i]
        if node.IsHeartbeatTimeOut(currentTime) {
            fmt.Println("心跳超时..... 关闭连接： ")
            node.Conn.Close()
        }
    }
    return result
}

//用户心跳是否超时
func (node *Node) IsHeartbeatTimeOut(currentTime uint64) (timeout bool)
{
    if node.HeartbeatTime+uint64(viper.GetInt("HeartbeatMaxTime")) <=
currentTime {
        fmt.Println("心跳超时。。。自动下线")
        timeout = true
    }
    return
}
```

在新建 定时任务

```
package utils

import (
    "time"
)

type TimerFunc func(interface{}) bool

/**
delay  首次延迟
tick  间隔
fun  定时执行的方法
param  方法的参数
**/
func Timer(delay, tick time.Duration, fun TimerFunc, param interface{})
{
    go func() {
        if fun == nil {
            return
        }
        t := time.NewTimer(delay)
        for {
```

```
            select {
            case <-t.C:
                if fun(param) == false {
                    return
                }
                t.Reset(tick)
            }
        }
    }()
}
```

最后在main方法启动时 调用

```
func main() {
    utils.InitConfig()
    utils.InitMySQL()
    utils.InitRedis()
    InitTimer()
    r := router.Router() // router.Router()
    r.Run(":8081")       // listen and serve on 0.0.0.0:8080 (for
windows "localhost:8080")
}
func InitTimer() {
    utils.Timer(time.Duration(viper.GetInt("DelayHeartbeat")),
time.Duration(viper.GetInt("HeartbeatHz")), models.CleanConnection, "")
}
```

前后端 联调:

```
message.go

func recvProc(node *Node) {
    for {
        _, data, err := node.Conn.ReadMessage()
        if err != nil {
            fmt.Println(err)
            return
        }
        msg := Message{}
        err = json.Unmarshal(data, &msg)
        if err != nil {
```

```
            fmt.Println(err)
        }
        //心跳检测 msg.Media == -1 || msg.Type == 3
        if msg.Type == 3 {
            currentTime := uint64(time.Now().Unix())
            node.Heartbeat(currentTime)
        } else {
            dispatch(data)
            broadMsg(data) //todo 将消息广播到局域网
            fmt.Println("[ws] recvProc <<<<< ", string(data))
        }

    }

}
```

然后将 前端心跳检测的方法 移到websocet里面

```
                heartbeat (){
                    console.log("心跳.............")
                    var msg =this.createmsgcontext();
                    msg.Media=-1; //备用
                    msg.Type=3
                    msg.Content="心跳";
                    //this.showmsg(userInfo(),msg);
                    this.webSocket.send(JSON.stringify(msg))
                }
```

在线用户缓存

key:userId value : Addr

{ Node} xxxx表示userId 过期时间

后期 可以考虑 Node 信息 。

后期 安全性 同源策略

实现:

封装一个 user_cache.go

```
package models

import (
    "context"
    "ginchat/utils"
    "time"
)

/**
设置在线用户到redis缓存
**/
func SetUserOnlineInfo(key string, val []byte, timeTTL time.Duration) {
    ctx := context.Background()
    utils.Red.Set(ctx, key, val, timeTTL)
}
```

调用在message.go 的 ：func Chat(writer http.ResponseWriter, request *http.Request) {

```
    //加入在线用户到缓存
SetUserOnlineInfo("online_"+Id, []byte(node.Addr), time.Duration(viper.GetInt("timeout.RedisOnlineTime"))*time.Hour)

}
```

顺路修改下前端掉线了还使劲心跳报错

```
  heartbeat (){
                        if (this.webSocket.readyState==1){   //失去连接 3
                            var msg =this.createmsgcontext();
                            msg.Media=-1;
                            msg.Type=3
                            msg.Content="心跳";
                            //this.showmsg(userInfo(),msg);
                            this.webSocket.send(JSON.stringify(msg))
                        }

                    }
```

发送消息根据缓存在线用户

修改配置

```
timeout:
  DelayHeartbeat: 3   #延迟心跳时间  单位秒
  HeartbeatHz: 30   #每隔多少秒心跳时间
  HeartbeatMaxTime: 30000  #最大心跳时间  ，超过此就下线
  RedisOnlineTime: 4  #缓存的在线用户时长    单位H
```

注释掉一开始测试的消息

```
//sendMsg(userId, []byte("欢迎进入聊天系统"))
```

发送的方法加入判断

```
func sendMsg(userId int64, msg []byte) {
    rwLocker.RLock()
    node, ok := clientMap[userId]
    rwLocker.RUnlock()
    jsonMsg := Message{}
    json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
    r, err := utils.Red.Get(ctx,
"online_"+strconv.Itoa(int(jsonMsg.TargetId))).Result()
    if err != nil {
        fmt.Println(err) //没有找到
    }
    if r != "" {
        if ok {
            fmt.Println("sendMsg >>> userID: ", userId, "  msg:",
string(msg))
            node.DataQueue <- msg
        }
    }
}
```

前端心跳请求改成10s：

```
setInterval(this.heartbeat,10*1000);//心跳检测的定时
```

修改上节问题：群收不到消息。

```
//jsonMsg := Message{}
  //json.Unmarshal(msg, &jsonMsg)
```

r, err := utils.Red.Get(ctx, "online_"+strconv.Itoa(int(userId)).Result()

消息的持久化:

1.首先修改Messasge实体类 加上两个

CreateTime uint64 //创建时间

ReadTime uint64 //读取时间

2.Dispatch处理消息给默认时间

//后端调度逻辑处理

func dispatch(data []byte) {

msg := Message{}

```
msg.CreateTime = uint64(time.Now().Unix())
```

err := json.Unmarshal(data, &msg)

3.发送消息排除自己

```
func sendGroupMsg(targetId int64, msg []byte) {
  fmt.Println("开始群发消息")
  userIds := SearchUserByGroupId(uint(targetId))
  for i := 0; i < len(userIds); i++ {
    //排除给自己的
    if targetId != int64(userIds[i]) {
      sendMsg(int64(userIds[i]), msg)
    }
```

```
    }

}
```

4，最后发送消息的同时持久化到redis

```
func sendMsg(userId int64, msg []byte) {
    rwLocker.RLock()
    node, ok := clientMap[userId]
    rwLocker.RUnlock()
    jsonMsg := Message{}
    json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
    targetIdStr := strconv.Itoa(int(userId))
    userIdStr := strconv.Itoa(int(jsonMsg.UserId))
    r, err := utils.Red.Get(ctx, "online_"+userIdStr).Result()
    if err != nil {
        fmt.Println(err) //没有找到
    }
    if r != "" {
        if ok {
            fmt.Println("sendMsg >>> userID: ", userId, "  msg:",
string(msg))
            node.DataQueue <- msg
        }
    }
    key := "msg_" + userIdStr + "_" + targetIdStr
    //utils.Red.ZAdd(ctx, key, &redis.Z{1, msg}) //jsonMsg
    utils.Red.Do(ctx, "zadd", key, 1, jsonMsg) //上面也OK
}
```

读取Redis缓存的消息

首先： r.POST("/user/redisMsg", service.RedisMsg)

然后service 里面

func RedisMsg(c *gin.Context) {

 userIdA, _ := strconv.Atoi(c.PostForm("userIdA"))

 userIdB, _ := strconv.Atoi(c.PostForm("userIdB"))

models.RedisMsg(int64(userIdA), int64(userIdB))

utils.RespOK(c.Writer, "ok", "")

}

接下来修改 message.go

```
func sendMsg(userId int64, msg []byte) {

    rwLocker.RLock()
    node, ok := clientMap[userId]
    rwLocker.RUnlock()
    jsonMsg := Message{}
    json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
    targetIdStr := strconv.Itoa(int(userId))
    userIdStr := strconv.Itoa(int(jsonMsg.UserId))
    r, err := utils.Red.Get(ctx, "online_"+userIdStr).Result()
    if err != nil {
        fmt.Println(err) //没有找到
    }
    if r != "" {
        if ok {
            fmt.Println("sendMsg >>> userID: ", userId, "  msg:",
string(msg))
            node.DataQueue <- msg
        }
    }
    var key string
    if userId > jsonMsg.UserId {
        key = "msg_" + userIdStr + "_" + targetIdStr
    } else {
        key = "msg_" + targetIdStr + "_" + userIdStr
    }
    res, e := utils.Red.ZAdd(ctx, key, &redis.Z{1, msg}).Result()
//jsonMsg
    //res, e := utils.Red.Do(ctx, "zadd", key, 1, jsonMsg).Result() //备
用 后续拓展 记录完整msg
    if e != nil {
        fmt.Println(e)
    }
    fmt.Println(res)
}

//需要重写此方法才能完整的msg转byte[]
func (msg Message) MarshalBinary() ([]byte, error) {
```

```
    return json.Marshal(msg)
}

//获取缓存里面的消息
func RedisMsg(userIdA int64, userIdB int64) {
    rwLocker.RLock()
    node, _ := clientMap[userIdA]
    rwLocker.RUnlock()
    //jsonMsg := Message{}
    //json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
    userIdStr := strconv.Itoa(int(userIdA))
    targetIdStr := strconv.Itoa(int(userIdB))
    var key string
    if userIdA > userIdB {
        key = "msg_" + targetIdStr + "_" + userIdStr
    } else {
        key = "msg_" + userIdStr + "_" + targetIdStr
    }
    //key = "msg_" + userIdStr + "_" + targetIdStr
    rels, err := utils.Red.ZRange(ctx, key, 0, 10).Result()
    if err != nil {
        fmt.Println(err) //没有找到
    }
    for _, val := range rels {
        fmt.Println("sendMsg >>> userID: ", userIdA, "  msg:", val)
        node.DataQueue <- []byte(val)
    }
}
```

整体调整：

新的页面 以及样式之类

index.go 引入新的静态页面 "views/chat/userinfo.html",

以及更新前端提供的样式

消息的 显示调整。.....

foot.html中

```
this.friends.map((item) => {

                if (item.ID == data.userId) {

                  // 1文字 2表情包 3图片 4音频
```

```
                if (data.Media === 1) {
                    item.memo = data.Content
                } else if (data.Media === 2) {
                    item.memo = data.Url
                } else if (data.Media === 3) {
                  item.memo = "[语音]"
                 } else if (data.Media === 4) {
                  item.memo = "[图片]"
                }
              }
            })
```

通过名称添加好友：

contact.go

```
//添加好友    自己的ID  ，  好友的ID
func AddFriend(userId uint, targetName string) (int, string) {
    //user := UserBasic{}

    if targetName != "" {
        targetUser := FindUserByName(targetName)
        //fmt.Println(targetUser, " userId        ", )
        if targetUser.Salt != "" {
            if targetUser.ID == userId {
                return -1, "不能加自己"
            }
            contact0 := Contact{}
            utils.DB.Where("owner_id =?  and target_id =? and type=1",
userId, targetUser.ID).Find(&contact0)
            if contact0.ID != 0 {
                return -1, "不能重复添加"
            }
            tx := utils.DB.Begin()
            //事务一旦开始，不论什么异常最终都会 Rollback
            defer func() {
                if r := recover(); r != nil {
                    tx.Rollback()
```

```
                }
            }()
            contact := Contact{}
            contact.OwnerId = userId
            contact.TargetId = targetUser.ID
            contact.Type = 1
            if err := utils.DB.Create(&contact).Error; err != nil {
                tx.Rollback()
                return -1, "添加好友失败"
            }
            contact1 := Contact{}
            contact1.OwnerId = targetUser.ID
            contact1.TargetId = userId
            contact1.Type = 1
            if err := utils.DB.Create(&contact1).Error; err != nil {
                tx.Rollback()
                return -1, "添加好友失败"
            }
            tx.Commit()
            return 0, "添加好友成功"
        }
        return -1, "没有找到此用户"
    }
    return -1, "好友ID不能为空"
}
```

userserver.go

```
func AddFriend(c *gin.Context) {
    userId, _ := strconv.Atoi(c.Request.FormValue("userId"))
    targetName := c.Request.FormValue("targetName")
    //targetId, _ := strconv.Atoi(c.Request.FormValue("targetId"))
    code, msg := models.AddFriend(uint(userId), targetName)
    if code == 0 {
        utils.RespOK(c.Writer, code, msg)
    } else {
        utils.RespFail(c.Writer, msg)
    }
}
```

foot.html

```
 addfriend: function () {
                    //console.log("addfriend....")
                    var that = this;
```

```
                    mui.prompt('', '请输入好友名称', '加好友', ['取消', '确
认'], function (e) {
                        if (e.index == 1) {
                            //判断数字
                            //if (isNaN(e.value) || e.value <= 0) {
                            //    mui.toast('格式错误');
                            //} else {
                            //mui.toast(e.value);
                            that._addfriend(e.value)
                            //}
                        } else {
                            //mui.toast('您取消了入库');
                        }
                    }, 'div');
                    document.querySelector('.mui-popup-input
input').type = 'text';

                },
```

并修复注册提示消息

```
 <form id='login-form' class="mui-input-group register-form">
            <div class="mui-input-row">
                <input v-model="user.name" placeholder="请输入用户名"
type="text" class="mui-input-clear mui-input">
            </div>




 methods: {
            login: function () {
                //检测密码是否为空
                console.log(this.user)
                //网络请求
                //封装了promis
                util.post("/user/createUser", this.user).then(res => {
                    console.log(res)
                    if (res.code != 0) {
                        mui.toast(res.message)
                    } else {
                        //location.replace("localhost:8081")
                        location.href = "/"
                        mui.toast("注册成功,即将跳转")
                    }
                })
            },
        }
    })
```

加入群改为通过群名称或者群号：

message.go

```
func JoinGroup(userId uint, comId string) (int, string) {
    contact := Contact{}
    contact.OwnerId = userId
    //contact.TargetId = comId
    contact.Type = 2
    community := Community{}

    utils.DB.Where("id=? or name=?", comId, comId).Find(&community)
    if community.Name == "" {
        return -1, "没有找到群"
    }
    utils.DB.Where("owner_id=? and target_id=? and type =2 ", userId,
comId).Find(&contact)
    if !contact.CreatedAt.IsZero() {
        return -1, "已加过此群"
    } else {
        contact.TargetId = community.ID
        utils.DB.Create(&contact)
        return 0, "加群成功"
    }
}
```

userserver.go :

```
//加入群 userId uint, comId uint
func JoinGroups(c *gin.Context) {
    userId, _ := strconv.Atoi(c.Request.FormValue("userId"))
    comId := c.Request.FormValue("comId")

    //  name := c.Request.FormValue("name")
    data, msg := models.JoinGroup(uint(userId), comId)
    if data == 0 {
        utils.RespOK(c.Writer, data, msg)
    } else {
        utils.RespFail(c.Writer, msg)
    }
}
```

foot.html

```
joincom: function () {
```

```
                    var that = this;
                    mui.prompt('', '请输入群号或者群名称', '加群', ['取消',
'确认'], function (e) {
                        if (e.index == 1) {
                            //    if (isNaN(e.value) || e.value <= 0) {
                            //        mui.toast('格式错误');
                            //    } else {
                            //mui.toast(e.value);
                            that._joincomunity(e.value)
                            // }
                        } else {
                            //mui.toast('您取消了入库');
                        }
                    }, 'div');
                    document.querySelector('.mui-popup-input
input').type = 'text';
                },
```

封装端口号配置参数

yml中

port:

server: ":8082"

udp: 3001

route.go

r.Run(viper.GetString("port.server"))

message.go

udpSendProc() {中 读取udp

顺序写和读取消息记录

message.go

```
func sendMsg(userId int64, msg []byte) {

    rwLocker.RLock()
    node, ok := clientMap[userId]
    rwLocker.RUnlock()
    jsonMsg := Message{}
    json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
```

```
    targetIdStr := strconv.Itoa(int(userId))
    userIdStr := strconv.Itoa(int(jsonMsg.UserId))
    jsonMsg.CreateTime = uint64(time.Now().Unix())
    r, err := utils.Red.Get(ctx, "online_"+userIdStr).Result()
    if err != nil {
        fmt.Println(err) //没有找到
    }
    if r != "" {
        if ok {
            fmt.Println("sendMsg >>> userID: ", userId, "  msg:",
string(msg))
            node.DataQueue <- msg
        }
    }
    var key string
    if userId > jsonMsg.UserId {
        key = "msg_" + userIdStr + "_" + targetIdStr
    } else {
        key = "msg_" + targetIdStr + "_" + userIdStr
    }
    res, err := utils.Red.ZRevRange(ctx, key, 0, -1).Result()
    if err != nil {
        fmt.Println(err)
    }
    score := float64(cap(res)) + 1
    ress, e := utils.Red.ZAdd(ctx, key, &redis.Z{score, msg}).Result()
//jsonMsg
    //res, e := utils.Red.Do(ctx, "zadd", key, 1, jsonMsg).Result() //备
用 后续拓展 记录完整msg
    if e != nil {
        fmt.Println(e)
    }
    fmt.Println(ress)
}


//获取缓存里面的消息
func RedisMsg(userIdA int64, userIdB int64, start int64, end int64)
[]string {
    rwLocker.RLock()
    //node, ok := clientMap[userIdA]
    rwLocker.RUnlock()
    //jsonMsg := Message{}
    //json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
    userIdStr := strconv.Itoa(int(userIdA))
    targetIdStr := strconv.Itoa(int(userIdB))
    var key string
    if userIdA > userIdB {
```

```
        key = "msg_" + targetIdStr + "_" + userIdStr
    } else {
        key = "msg_" + userIdStr + "_" + targetIdStr
    }
    //key = "msg_" + userIdStr + "_" + targetIdStr
    //rels, err := utils.Red.ZRevRange(ctx, key, 0, 10).Result()  //根据
score倒叙
    rels, err := utils.Red.ZRange(ctx, key, start, end).Result()
    if err != nil {
        fmt.Println(err) //没有找到
    }
    // 发送推送消息
    /**
    // 后台通过websoket 推送消息
    for _, val := range rels {
        fmt.Println("sendMsg >>> userID: ", userIdA, "  msg:", val)
        node.DataQueue <- []byte(val)
    }**/
    return rels
}
```

userservice.go

```
func RedisMsg(c *gin.Context) {
    userIdA, _ := strconv.Atoi(c.PostForm("userIdA"))
    userIdB, _ := strconv.Atoi(c.PostForm("userIdB"))
    start, _ := strconv.Atoi(c.PostForm("start"))
    end, _ := strconv.Atoi(c.PostForm("end"))
    res := models.RedisMsg(int64(userIdA), int64(userIdB), int64(start),
int64(end))
    utils.RespOKList(c.Writer, "ok", res)
}
```

foot.html

```
   isReadRedisMsg: [],  //是否已读取某个用户的缓存消息


    singlemsg: function (user) {
                    if (this.isDisable) {
                        //首次读取某个用户的消息记录
                        if (this.isReadRedisMsg.filter(item => item ===
user.ID).length <= 0) {
                            post("user/redisMsg", { userIdA: userId(),
userIdB: user.ID, start: 0, end: 9 }, function (res) {
                                //循环读取的消息记录 并显示
                                for (var i in res.Total) {
```

```
                                    this.showmsg(user,
JSON.parse(res.Total[i]))
                                }
                            }.bind(this))
                            this.isReadRedisMsg.push(user.ID)
                        }

                        this.setTimeFlag()
                        //console.log(user)
                        this.win = "single";
                        this.title = "和" + user.Name + "聊天中";
                        this.msgcontext.TargetId = parseInt(user.ID);
                        this.msgcontext.Type = 1;
                    }
                },
```

整体功能完善:

1，新建群功能 + 图片 描述

userservice.go

```
//新建群
func CreateCommunity(c *gin.Context) {
    ownerId, _ := strconv.Atoi(c.Request.FormValue("ownerId"))
    name := c.Request.FormValue("name")
    icon := c.Request.FormValue("icon")
    desc := c.Request.FormValue("desc")
    community := models.Community{}
    community.OwnerId = uint(ownerId)
    community.Name = name
    community.Img = icon
    community.Desc = desc
    code, msg := models.CreateCommunity(community)
    if code == 0 {
        utils.RespOK(c.Writer, code, msg)
    } else {
        utils.RespFail(c.Writer, msg)
    }
}
```

2,维护用户信息

```
func UpdateUser(c *gin.Context) {
    user := models.UserBasic{}
    id, _ := strconv.Atoi(c.PostForm("id"))
    user.ID = uint(id)
    user.Name = c.PostForm("name")
```

```
    user.PassWord = c.PostForm("password")
    user.Phone = c.PostForm("phone")
    user.Avatar = c.PostForm("icon")
    user.Email = c.PostForm("email")
    fmt.Println("update :", user)

    _, err := govalidator.ValidateStruct(user)
    if err != nil {
        fmt.Println(err)
        c.JSON(200, gin.H{
            "code":    -1, //  0成功   -1失败
            "message": "修改参数不匹配！",
            "data":    user,
        })
    } else {
        models.UpdateUser(user)
        c.JSON(200, gin.H{
            "code":    0, //  0成功   -1失败
            "message": "修改用户成功！",
            "data":    user,
        })
    }

}
```

3,缓存消息记录

```
func RedisMsg(c *gin.Context) {
    userIdA, _ := strconv.Atoi(c.PostForm("userIdA"))
    userIdB, _ := strconv.Atoi(c.PostForm("userIdB"))
    start, _ := strconv.Atoi(c.PostForm("start"))
    end, _ := strconv.Atoi(c.PostForm("end"))
    isRev, _ := strconv.ParseBool(c.PostForm("isRev"))
    res := models.RedisMsg(int64(userIdA), int64(userIdB), int64(start),
int64(end), isRev)
    utils.RespOKList(c.Writer, "ok", res)
}
```

对应的 message.go

```
//获取缓存里面的消息
func RedisMsg(userIdA int64, userIdB int64, start int64, end int64,
isRev bool) []string {
    rwLocker.RLock()
    //node, ok := clientMap[userIdA]
    rwLocker.RUnlock()
```

```
    //jsonMsg := Message{}
    //json.Unmarshal(msg, &jsonMsg)
    ctx := context.Background()
    userIdStr := strconv.Itoa(int(userIdA))
    targetIdStr := strconv.Itoa(int(userIdB))
    var key string
    if userIdA > userIdB {
        key = "msg_" + targetIdStr + "_" + userIdStr
    } else {
        key = "msg_" + userIdStr + "_" + targetIdStr
    }
    //key = "msg_" + userIdStr + "_" + targetIdStr
    //rels, err := utils.Red.ZRevRange(ctx, key, 0, 10).Result()  //根据
score倒叙

    var rels []string
    var err error
    if isRev {
        rels, err = utils.Red.ZRange(ctx, key, start, end).Result()
    } else {
        rels, err = utils.Red.ZRevRange(ctx, key, start, end).Result()
    }
    if err != nil {
        fmt.Println(err) //没有找到
    }
    // 发送推送消息
    /**
    // 后台通过websoket 推送消息
    for _, val := range rels {
        fmt.Println("sendMsg >>> userID: ", userIdA, "  msg:", val)
        node.DataQueue <- []byte(val)
    }**/
    return rels
}
```

对应 user_basic.go

```
func UpdateUser(user UserBasic) *gorm.DB {
    return utils.DB.Model(&user).Updates(UserBasic{Name: user.Name,
PassWord: user.PassWord, Phone: user.Phone, Email: user.Email, Avatar:
user.Avatar})
}
```

消息乱序及页面遮挡问题修复

main.html

```
<div v-show="win == 'single' || win == 'group'">
```

foot.html

```
  singlemsg: function (user) {
                    this.start = 0;
                    this.end = 9;
                    if (this.isDisable) {
                        //首次读取某个用户的消息记录
                        if (this.isReadRedisMsg.filter(item => item ===
user.ID).length <= 0) {
                            post("user/redisMsg", { userIdA: userId(),
userIdB: user.ID, start: this.start, end: this.end, isRev: false },
function (res) {
                                //循环读取的消息记录 并显示
                                for (var i in res.Total) {
                                    this.showmsg(user,
JSON.parse(res.Total[i]), false, true)
                                }
                            }.bind(this))
                            this.isReadRedisMsg.push(user.ID)
                        }




  //下拉获取历史消息记录
                        document.querySelector('.mui-scroll-
wrapper').addEventListener('scroll', (e) => {
                            let translate =
e.target.style?.transform?.match(/translate3d\(\d+px,\s*(\d+)px,\s*
(\d+)px\)/i);
                            if (translate && translate.length > 1) {
                                if (translate[1] > 0 && this.isLoadMore
== false) {
                                    this.isLoadMore = true;
                                    this.start = this.end + 1;
                                    this.end = this.end + 2;
                                    post("user/redisMsg", { userIdA:
userId(), userIdB: user.ID, start: this.start, end: this.end, isRev:
false }, function (res) {
                                        //循环读取的消息记录 并显示
                                        for (var i in res.Total) {
                                            this.showmsg(user,
JSON.parse(res.Total[i]), true)
```

```
                                    }
                                    setTimeout(() => {
                                        this.isLoadMore = false;
                                    }, 300);
                                }.bind(this))
                                this.isReadRedisMsg.push(user.ID)
                            }
                        }
                    })


    showmsg: function (user, msg, isReverse, isFirst) {
                    //console.log(">>>>>>>>>>>", user)
                    // console.log(">>>>>>>>>>>", msg)
                    var data = {
                    }
                    data.ismine = userId() == msg.userId;
                    //console.log(data.ismine,userId(),msg.userid)
                    data.user = user;
                    data.msg = msg;
                    console.log(this.msglist)
                    if (isReverse) {
                        this.msglist = [data].concat(this.msglist);
                    } else {
                        //首次获取消息渲染
                        if (isFirst) {
                            this.msglist = [data].concat(this.msglist);
                            //下拉获取消息渲染
                        } else {
                            this.msglist = this.msglist.concat(data)
                        }
                    }
                    this.reset();
```

打包与发布服务：

首先Windows上

go build main.go

如果需要静态资源打包：

```
rd /s/q release
md release
::go build -ldflags "-H windowsgui" -o chat.exe
go build -o chat.exe
COPY chat.exe release\
COPY favicon.ico release\favicon.ico
XCOPY asset\*.* release\asset\  /s /e
XCOPY view\*.* release\view\  /s /e
```

执行之后会生成 chat.exe 以及 release包

打包到Linux 上面：

set GOARCH=amd63

go env -w GOOS=linux

go build main.go

会得到一个 main 文件

执行这个文件。 赋予权限： chmod 777 main

然后执行这个 ./main

如果需要静态资源单独打包：

```
#!/bin/sh
rm -rf ./release
mkdir  release
go build -o chat
chmod +x ./chat
cp chat ./release/
cp favicon.ico ./release/
cp -arf ./asset ./release/
```

```
cp -arf ./view ./release/
```

当然记得改回window模式go env -w GOOS=windows

set GOARCH=amd64

然后就OK拉

docker 镜像：

mkdir /root/ginchatdockerfile

vim /root/ginchatdockerfile/Dockerfile

FROM centos:centos7

ADD ./ginchat.tgz /

WORKDIR /ginchat-v1.0

RUN chmod +x /ginchat-v1.0/main

EXPOSE 8081

CMD /ginchat-v1.0/main

-- 然后：wq退出

-- 打包

tar cvzf ginchat.tgz ginchat-v1.0

-- 移到到 docker镜像目录

mv ginchat.tgz ./../ginchatdockerfile

--创建镜像

docker build -t ginchat:v1 .

运行 镜像：

docker run -d -p 8081:8081 ginchat:v1

-- 查看日志

docker logs f50 | tail -f

git

https://git.mashibing.com/msb_47094/GinChat.git