**青训营后端结业项目答辩汇报文档**

**一、项目介绍**

实现极简版抖音的基础功能与互动方向

项目服务地址：本地主机 app运行环境为雷电模拟器9

github地址：https://github.com/OVOAOVO/Tiktok

**二、项目分工**

|  |  |
| --- | --- |
| 团队成员 | 主要贡献 |
| 刘浩天 | 极简版抖音项目后端的实现 |
| 徐建超 | 辅助开发 |
| 方小淋 | 辅助开发 |

**三、项目实现**

**3.1 技术选型**

**3.1.1使用redis加速点赞和关注**

点赞跟关注是一个很明显的问题，因为用户很有可能频繁的取消点赞跟关注，所以本人使用redis做了一些数据的缓存，不如是否被点赞以及用户是否被关注等

**3.1.2使用gorm建立数据库表单**

因为数据库做起来比较复杂，有一些多对多的关系要额外建立中间表，使用gorm进行自动建表

|  |
| --- |
| Go type Video struct {  Id int64 `json:"id,omitempty"`  UserInfoId int64 `json:"-"`  Author UserInfo `json:"author,omitempty" gorm:"-"` //这里应该是作者对视频的一对多的关系，而不是视频对作者，故gorm不能存他，但json需要返回它  PlayUrl string `json:"play\_url,omitempty"`  CoverUrl string `json:"cover\_url,omitempty"`  FavoriteCount int64 `json:"favorite\_count,omitempty"`  CommentCount int64 `json:"comment\_count,omitempty"`  IsFavorite bool `json:"is\_favorite,omitempty"`  Title string `json:"title,omitempty"`  Users []\*UserInfo `json:"-" gorm:"many2many:user\_favor\_videos;"`  Comments []\*Comment `json:"-"`  CreatedAt time.Time `json:"-"`  UpdatedAt time.Time `json:"-"` }  //以下为统一的数据库建表 func InitDB() {  var err error  DB, err = gorm.Open(mysql.Open(config.DBConnectString()), &gorm.Config{ //建立数据库链接  PrepareStmt: true, //缓存预编译命令  SkipDefaultTransaction: true, //禁用默认事务操作  //Logger: logger.Default.LogMode(logger.Info), //打印sql语句  })  if err != nil {  panic(err)  }  err = DB.AutoMigrate(&UserInfo{}, &Video{}, &Comment{}, &UserLogin{}) //初始化表 用户信息 视频 评论 用户登录  if err != nil {  panic(err)  } } |

**3.1.3安全防护措施**

主要有以下几点

1. 使用gorm框架进行数据库操作，防止sql注入
2. 用户的密码使用SHA1加密，过程设置在middleware包中
3. 使用JWT用户鉴权，并加入到middleware包中
4. 文件是否合法，对视频文件格式是否支持校验
5. 保证视频文件唯一，将用户id与上传视频数量拼接形成一个String进行校验
6. 所有参数在service层完成校验

**3.1.4ffmpeg视频处理**

封装ffmpeg工具类在util包下，可以根据具体参数来决定最终视频处理的操作

|  |
| --- |
| Go package util  //#include <stdlib.h> //int startCmd(const char\* cmd){ // return system(cmd); //} import "C"  import (  "errors"  "fmt"  "log"  "unsafe"   "github.com/config" )  type Video2Image struct {  InputPath string  OutputPath string  StartTime string  KeepTime string  Filter string  FrameCount int64  debug bool }  func NewVideo2Image() \*Video2Image {  return &videoChanger }  var videoChanger Video2Image  // ffmpeg的参数 const (  inputVideoPathOption = "-i"  startTimeOption = "-ss"  keepTimeOption = "-t"  videoFilterOption = "-vf"  formatToImageOption = "-f"  autoReWriteOption = "-y"  framesOption = "-frames:v" )  var (  defaultVideoSuffix = ".mp4"  defaultImageSuffix = ".jpg" )  func ChangeVideoDefaultSuffix(suffix string) { //后缀  defaultVideoSuffix = suffix }  func ChangeImageDefaultSuffix(suffix string) {  defaultImageSuffix = suffix }  func GetDefaultImageSuffix() string {  return defaultImageSuffix }  func paramJoin(s1, s2 string) string {  return fmt.Sprintf(" %s %s ", s1, s2) }  func (v \*Video2Image) Debug() {  v.debug = true }  func (v \*Video2Image) GetQueryString() (ret string, err error) {  if v.InputPath == "" || v.OutputPath == "" {  err = errors.New("输入输出路径未指定")  return  }  ret = config.Info.FfmpegPath  ret += paramJoin(inputVideoPathOption, v.InputPath)  ret += paramJoin(formatToImageOption, "image2")  if v.Filter != "" {  ret += paramJoin(videoFilterOption, v.Filter)  }  if v.StartTime != "" {  ret += paramJoin(startTimeOption, v.StartTime)  }  if v.KeepTime != "" {  ret += paramJoin(keepTimeOption, v.KeepTime)  }  if v.FrameCount != 0 {  ret += paramJoin(framesOption, fmt.Sprintf("%d", v.FrameCount))  }  ret += paramJoin(autoReWriteOption, v.OutputPath)  return }  func (v \*Video2Image) ExecCommand(cmd string) error {  if v.debug {  log.Println(cmd)  }  cCmd := C.CString(cmd)  defer C.free(unsafe.Pointer(cCmd))  status := C.startCmd(cCmd)  if status != 0 {  return errors.New("视频切截图失败")  }  return nil } |

**3.1.5配置文件单独处理**

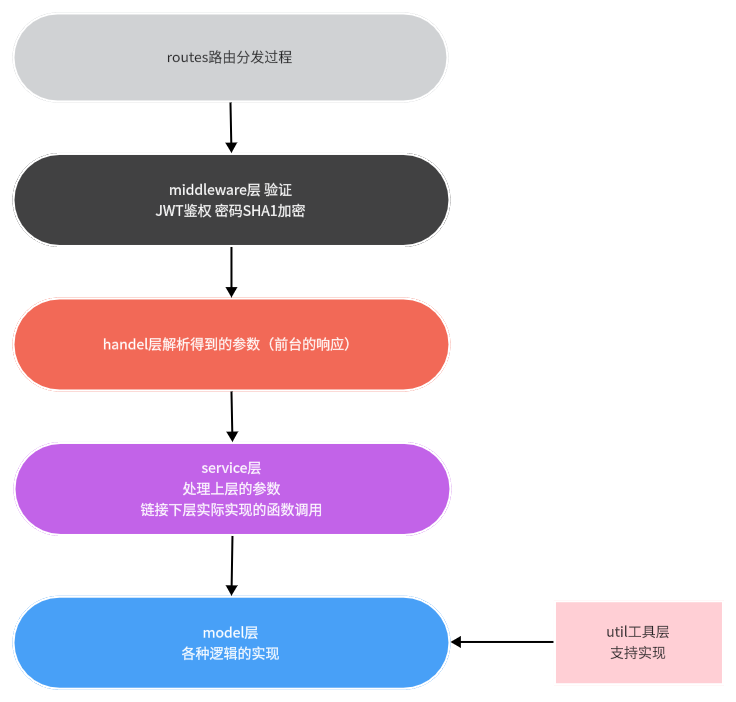
通过config.toml进行处理全部的数据库等信息配置

|  |
| --- |
| Go #关系型数据库配置 [mysql] host = "127.0.0.1" port = 3306 database = "douyin" username = "root" password = "369258147lht" charset = "utf8mb4" parse\_time = true loc = "Local"  #redis配置  [redis] host = "127.0.0.1" port = 6379 database = 0   #记录当前服务器ip和启动端口号 当前服务器ip用于生成对应的视频链接地址 [server] ip = "192.168.1.102" port = 8080  #用于保存资源的路径，用于截图工具的路径 截图工具放在lib目录 [path] ffmpeg\_path = "D:\\tiktok\\dousheng\\lib\\ffmpeg.exe" static\_source\_path = "D:\\tiktok\\dousheng\\static" |

**3.2 架构设计**

**3.2.1项目流程**

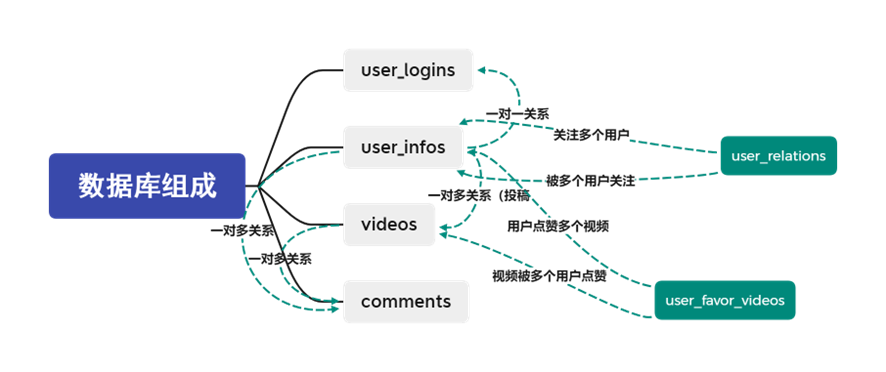
大体流程如下图所示



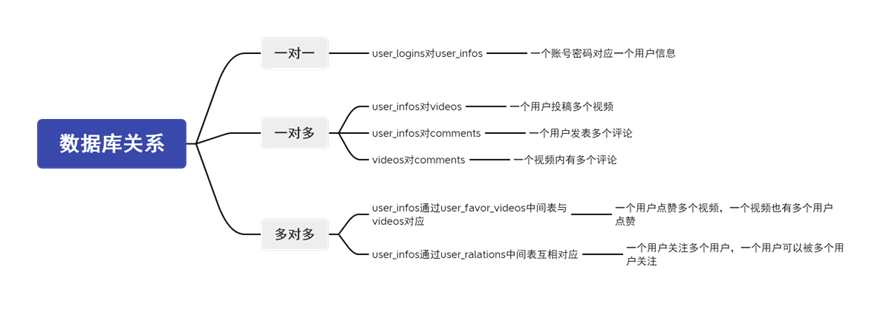
由前台获取到相应的参数handle层获取 进行middleware的验证之后再经过 servce层获取参数与调用相应的model层实现

**3.2.2数据库设计**

数据库组成部分：



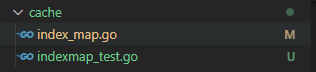
数据库相关关系说明



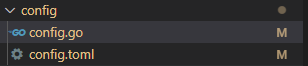
**3.3 项目代码介绍**



cache包中为redis缓存的实现,主要是存入到内存中实现相应的功能，例如：获取到相应的点赞状态后再进行更改



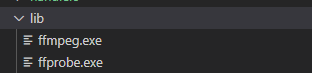
config包中为链接的配置信息与具体实现



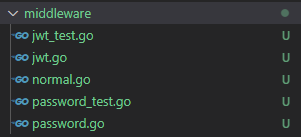
handle层中即为获取前台的相应信息得到相应的参数来调用service中的函数



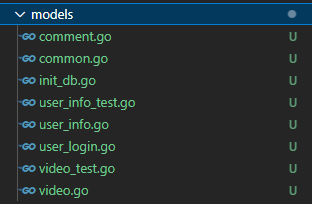
lib包中为两个ffmpeg运行文件，主要用来运行时包的调用



middleware层主要是进行一些验证如token验证以及上述过的视频上传id唯一性等



model层进行各种功能的详细实现如获取到前台的参数后进行数据库处理返回相应的值



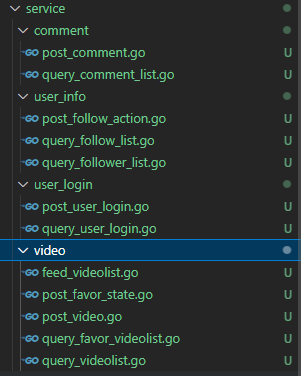
router包中主要是与极简抖音app进行关联进行相应的接口调用



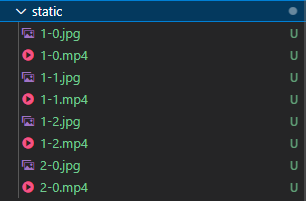
以下为调用详细代码

|  |
| --- |
| Go package router  import (  "github.com/gin-gonic/gin"  "github.com/handlers/comment"  "github.com/handlers/user\_info"  "github.com/handlers/user\_login"  "github.com/handlers/video"  "github.com/middleware"  "github.com/models" )  func InitDouyinRouter() \*gin.Engine {  models.InitDB()  r := gin.Default()   r.Static("static", "./static")   baseGroup := r.Group("/douyin")  //根据灵活性考虑是否加入JWT中间件来进行鉴权，还是在之后再做鉴权  // basic apis  baseGroup.GET("/feed/", video.FeedVideoListHandler)  baseGroup.GET("/user/", middleware.JWTMiddleWare(), user\_info.UserInfoHandler)  baseGroup.POST("/user/login/", middleware.SHAMiddleWare(), user\_login.UserLoginHandler)  baseGroup.POST("/user/register/", middleware.SHAMiddleWare(), user\_login.UserRegisterHandler)  baseGroup.POST("/publish/action/", middleware.JWTMiddleWare(), video.PublishVideoHandler)  baseGroup.GET("/publish/list/", middleware.NoAuthToGetUserId(), video.QueryVideoListHandler)   //extend 1  baseGroup.POST("/favorite/action/", middleware.JWTMiddleWare(), video.PostFavorHandler)  baseGroup.GET("/favorite/list/", middleware.NoAuthToGetUserId(), video.QueryFavorVideoListHandler)  baseGroup.POST("/comment/action/", middleware.JWTMiddleWare(), comment.PostCommentHandler)  baseGroup.GET("/comment/list/", middleware.JWTMiddleWare(), comment.QueryCommentListHandler)   //extend 2  baseGroup.POST("/relation/action/", middleware.JWTMiddleWare(), user\_info.PostFollowActionHandler)  baseGroup.GET("/relation/follow/list/", middleware.NoAuthToGetUserId(), user\_info.QueryFollowListHandler)  baseGroup.GET("/relation/follower/list/", middleware.NoAuthToGetUserId(), user\_info.QueryFollowerHandler)  return r } |

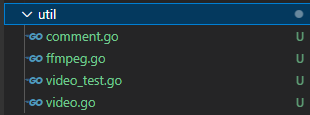
service层起到获取上述handle传入的参数调用相应接口的功能



static中主要用来保存上传的视频与视频截取的第一帧



util包中主要是实现的工具类 ffmpeg视频转换工具的使用以及 评论的时间转换与评论框的填充 还有视频的第一帧详细截取



最后main.go函数执行调用router

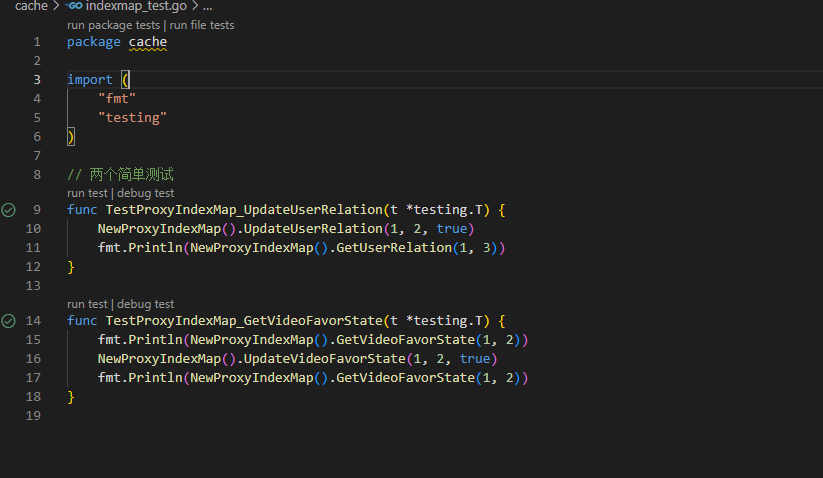


|  |
| --- |
| Go package main  import (  "fmt"   "github.com/config"  "github.com/router" )  func main() {  r := router.InitDouyinRouter()  err := r.Run(fmt.Sprintf(":%d", config.Info.Port)) // listen and serve on 0.0.0.0:8080 (for windows "localhost:8080")  if err != nil {  return  } } |

**四、测试结果**

测试类是导入go本身自带的测试类testing自动生成修改后进行测试

各测试结果均可以通过，部分测试类结果如下



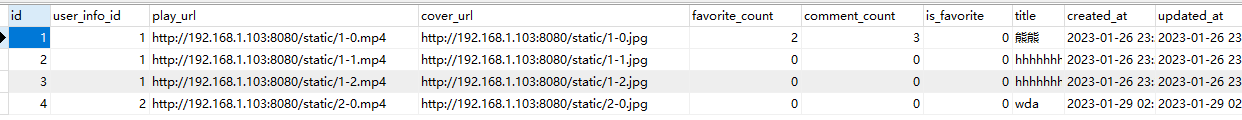
左侧标识为两个测试通过

**五、Demo 演示视频**

**[video\_20230203\_235956\_edit.mp4]**

**六、项目总结与反思**

1. 项目再正常运行情况下运行正常，但处理异常的能力很差比如，在视频中有一个显示bug在删除掉本地视频后新导入视频第一帧不会再次调入到app中但原视频依然更新为新传入的视频
2. 在使用gorm框架时，gin的导入非常费力，虽然很大程度上源于国内网络环境问题，而且一定程度上解决建表的问题，但在第一次搭建项目的时候是否考虑一个更适合本地的框架快速开发更好？
3. 密码在加密的时候只是单纯调用了SHA1加密并未进行加盐处理
4. 由于项目是在本地运行的每次重启之后由于ip地址会发生变化，所以对应的签证会失效比如视频上传时保存本地的数据在数据库中url地址，还有前面的config文件每次要运行时都要进行修改才会功能正常



**七、补充资料**

基本上是根据这位前辈的仿照了一个做的，部分内容所有改动，只不过是走了一遍前辈走过的路罢了。

https://www.bilibili.com/video/BV1yL4y1P7vq?vd\_source=2e5ee769a787cbb847e874101c16ce84