

图片兴趣社区编码规范

项 目 名 称： 图片兴趣社区

成 员 名 单： 谈会根 张衡 曾俊铭 王鹏 辅智豪 陈浩 陈卿 赵楚涵 沈扬

导 师： 张曙

工 程 领 域： 软件程序开发

研 究 方 向： 后端开发

1 目录结构

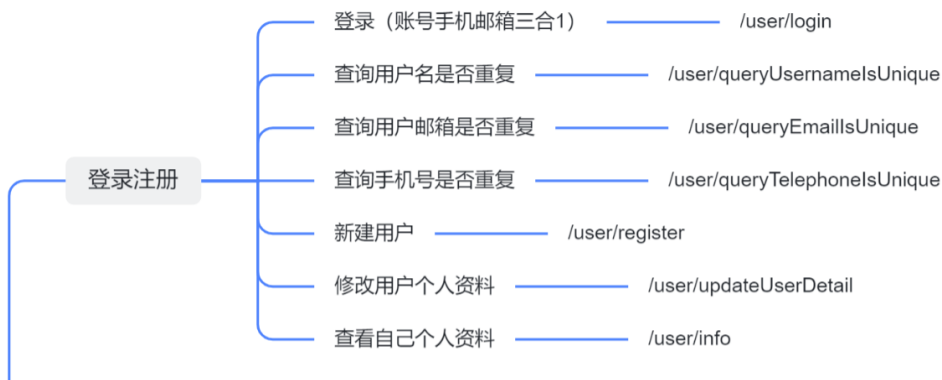
PictureCommunity	
├controller	#controller层 用于接收http请求,序列化请求和返回参数
├dao	#dao持久层 所有具体mysql和redis操作
│├firstpage	
│├post	
│└user	
├entity	#实体类
│├db	#数据库直接相关的表类
│├_request	#入参
│└_response	#回参
├global	#全局变量
├initialize	#初始化
├middleware	#中间件, 例如token鉴权
├response	#返回类型包装, 包含返回成功、失败等格式
├router	#http路由
├service	#service层
├utils	#工具类 jwt/雪花ID生产者 等
└main.go	#main

2 API文档规范

API文档是用来规范前后端接口的入参回参的文档，在前后端接口设计时起重要作用。

2.1 api路由

api路由是根据系统的模块，绑定路由和功能的对照表，在编写时首先在api路由里写清楚api的作用和url

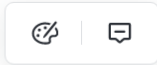



2.2 api文档

api文档需要标明api的功能、url、传参方式、传参示例、返回参数、返回示例等。

功能和url




用户登录 (谈会根)



 post: /user/login

入参表和入参方式

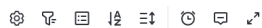
入参/xxx-form格式

<input type="checkbox"/>  参数名	 是否必填	 字段类型	参数含义	返回值说明	+
1 info	是	string	用户名 (账号名/手机/邮箱)		
2 password	是	string	密码		
3 method	是	int	1: 账号名 2: 手机 3: 邮箱		
+					

3 条记录

回参表

返回值



<input type="checkbox"/>  参数名	 是否必填	 字段类型	参数含义	返回值说明	+
<input checked="" type="checkbox"/> token	是	string	令牌	如果登录失败为空	
+					

1 条记录

返回示例

200 OK



```
{
  "code": 200,
  "data": {
    "token":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJJRCi6MzMslmV4cCI6MjI0NzgzMTQ0MiwiaXNzIjoidGhnlwibmJmljoxNjQzMDMwNDQyfQ.2YqSDm8ixvV-z1RdnlugAY23_8UXvVZDfZtJSM1W9H8"
  },
  "msg": "登录成功"
}
```



400 Bad Request

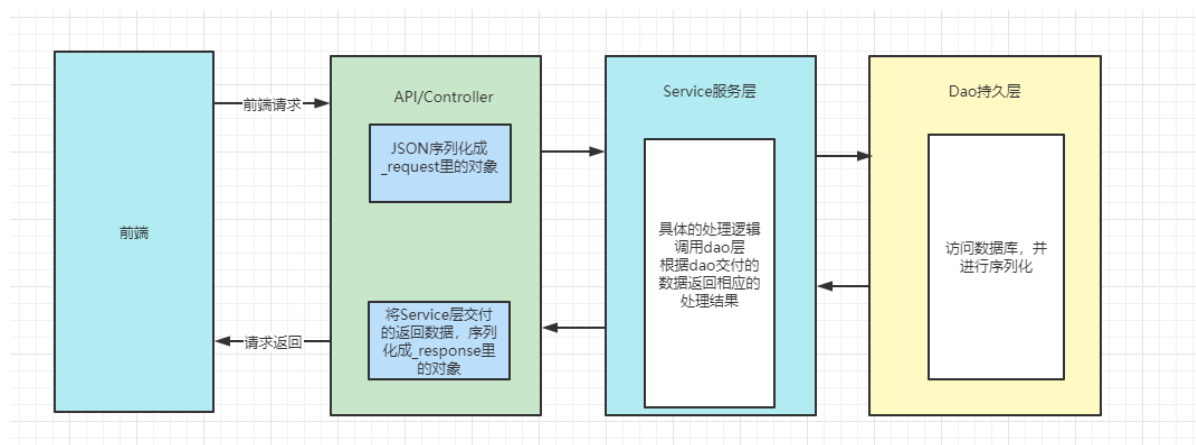


```
{
  "code": 300,
  "data": null,
  "msg": "账号不存在"
}
```



```
{
  "code": 300,
  "data": null,
  "msg": "密码错误"
}
```

3 数据处理模型



Controller 是接口入口，负责请求入参的校验，请求的返回

Service 是业务处理函数

Dao 持久层，负责和数据库打交道

4 命名规范

命名是代码规范中很重要的一部分，统一的命名规则有利于提高的代码的可读性，好的命名仅仅通过命名就可以获取到足够多的信息。

包命名：package

保持package的名字和目录保持一致，尽量采取有意义的包名，简短，有意义，尽量和标准库不要冲突。包名应该为**小写**单词，不要使用下划线或者混合大小写。

```
package demo
```

```
package main
```

文件命名

尽量采取有意义的文件名，简短，有意义，应该为**小写**单词，使用**下划线**分隔各个单词。

```
my_test.go
```

结构体命名

- 采用驼峰命名法，首字母根据访问控制大写或者小写
- struct 申明和初始化格式采用多行，例如下面：

```
// 多行申明
type User struct{
    Username string
    Email    string
}
•
// 多行初始化
u := User{
    Username: "astaxie",
    Email:    "astaxie@gmail.com",
}
```

变量命名

- 和结构体类似，变量名称一般遵循驼峰法，首字母根据访问控制原则大写或者小写，但遇到特有名词时，需要遵循以下规则：
- 如果变量为私有，且特有名词为首个单词，则使用小写，如 apiClient
- 其它情况都应当使用该名词原有的写法，如 APIClient、repoID、UserID
- 错误示例：UrlArray，应该写成 urlArray 或者 URLArray

- 若变量类型为 `bool` 类型, 则名称应以 `Has`, `Is`, `Can` 或 `Allow` 开头

```
var isExist bool
var hasConflict bool
var canManage bool
var allowGitHook bool
```

5 注释规范

包注释

每个包都应该有一个包注释, 一个位于`package`子句之前的块注释或行注释。包如果有多个go文件, 只需要出现在一个go文件中(一般是和包同名的文件)即可。包注释应该包含下面基本信息(请严格按照这个顺序, 简介, 创建人, 创建时间):

- 包的基本简介(包名, 简介)
- 创建者, 格式: 创建人: `rtx` 名
- 创建时间, 格式: 创建时间: `yyyyMMdd`

```
// util 包, 该包包含了项目共用的一些常量, 封装了项目中一些共用函数。
// 创建人: tanhuigen
// 创建时间: 2021-10-19
```

import

```
import (
    "encoding/json"
    "strings"

    "myproject/models"
    "myproject/controller"
    "myproject/utils"

    "github.com/astaxie/beego"
    "github.com/go-sql-driver/mysql"
)
```

其他注释

其他需要加注释的地方有:

- `controller` 标注对应的`restful_url`、作者、时间、函数作用等
- 对于一些关键位置的代码逻辑, 或者局部较为复杂的逻辑, 需要有相应的逻辑说明, 方便其他开发者阅读该段代码。

6 代码编写流程和规范

整体命名方式采用驼峰式，以user/login为例

1. 注册路由

现在router/router.go里注册controller

```
func SetRouter() {
    r := global.GinEngine
    user := r.Group("/user") //指定用户组
    {
        user.POST("/login", controller.LoginController)
    }
}
```

在指定的用户组里绑定路由和Controller函数。

注意：绑定函数前先思考是否需要middleware中间处理函数处理。

2. Controller

Controller 是接口入口，负责请求入参的校验，请求的返回，但注意：

- 函数名需要体现函数的作用
- 必要的地方加入注释

```
func LoginController(c *gin.Context) {
    var u _request.LoginUser

    //检查参数
    if err := c.ShouldBind(&u); err != nil {
        response.CheckFail(c, nil, "参数错误")
        return
    }
    //调用service, 并返回相应的响应
    status, message, token := service.VerifyLogin(u)
    if status {
        response.Success(c, _response.Token{Token: token}, message) //登录成功
    } else {
        response.Fail(c, nil, message) // 登录失败
    }
}
```

3. Service

Service 是具体业务处理函数

```
func VerifyLogin(param _request.LoginUser) (isValid bool, message string, token string) {
    var id int64
    var password string
```

```

var err error

message = "密码错误"
isValid = false
token = ""

switch param.Method {
case 0:
    id, password, err = user.QueryIDAndPasswordByUsername(param.Info)
case 1:
    id, password, err = user.QueryIDAndPasswordByEmail(param.Info)
case 2:
    id, password, err = user.QueryIDAndPasswordByTelephone(param.Info)
default:
    message = "格式错误"
    return
}

if err != nil {
    isValid = false
    if errors.Is(err, gorm.ErrRecordNotFound) {
        message = "账号不存在"
        return
    } else {
        message = "数据库错误"
        return
    }
}

if password == param.Password {
    isValid = true
    message = "登录成功"
    token = utils.CreateToken(id)
}
return
}

```

4.Dao

Dao 持久层，负责和数据库打交道

```

func QueryIDAndPasswordByUsername(username string) (int64, string, error) {
    var user db.User
    err := global.MysqlDB.Select("uid", "password").Where("username=?",
username).First(&user).Error
    return int64(user.UID), user.Password, err
}

func QueryIDAndPasswordByEmail(email string) (int64, string, error) {
    var user db.User
    err := global.MysqlDB.Select("uid", "password").Where("email=?",
email).First(&user).Error
    return int64(user.UID), user.Password, err
}

```

```

}

func QueryIDAndPasswordByTelephone(telephone string) (int64, string, error) {
    var user db.User
    err := global.MysqlDB.Select("uid", "password").Where("telephone=?",
telephone).First(&user).Error
    return int64(user.UID), user.Password, err
}

```

5. 返回值规范

所有的接口返回值都应该有如下几种参数

```

{
    "code": 200,
    "data": {},
    "message": "用户名非法"
}

```

其中code是http的status code, data为gin框架的interface接口, message为一个string字段

code参数几种返回

```

const (
    SuccessCode = 200
    FailCode = 300
    CheckFailCode = 400
    ServerErrorCode = 500
    Unauthorized = 600
)

```

返回时直接调用response包里的函数就行了

```

response.Success(c, _response.Token{Token: token}, message)
response.Fail(c, nil, message)

```

7 数据库实体类

数据库实体类与mysql数据一一对应, 在查看表结构时无需访问数据库, 直接查看db包下entity类即可。


```

type User struct {
    UID          uint   `gorm:"primaryKey"`
    Username     string `gorm:"unique"`
    Password     string
    Telephone    uint   `gorm:"unique"`
    Email        string `gorm:"unique"`
    Status       int8
    UpdateTime   time.Time `gorm:"autoUpdateTime:milli"`
    CreateTime   time.Time `gorm:"autoCreateTime:milli"`
}

```

8 入参回参

对于复杂的http接口，可以在_request包和response包里编写对应的对象。

- 在controller中使用shouldbind将http数据序列化成_request包中对象
- 在controller返回或者在service中将数据封装到response包对象即可

request

```

type UpdateUserDetailInfo struct {
    Nickname string `form:"nickname" json:"nickname"`
    //false为女 true为男
    Sex      bool  `form:"sex" json:"sex"`
    Birthday string `form:"birthday" json:"birthday"`
    Address  string `form:"address" json:"address"`
    Motto    string `form:"motto" json:"motto"`
    // 略缩图
    Profile string `form:"profile" json:"profile"`
    //详细头像url
    OriginProfile string `form:"origin_profile" json:"origin_profile"`
}

```

response

```

type QueryCommentBack struct {

    UID      int    `json:"userId"`
    NickName string `json:"nickName"`
    // Profile string //头像略缩图
    ChildNumber int    //子评论个数
    Content     string //内容
}
type QueryCommentBackTemp struct {

    ChildNumber int //子评论个数
    LikeNumber  int
    Content     string //内容
    UserID      uint   //评论作者id
}

```

