

Golang接口代码走查

本文汉字：3477个， 英文单词：936个

建议阅读时间：30分钟

- 写在前面
 - 日常测试过程中是否有过这样的困惑
 - 那对于以上困惑，咱们有啥方法解决
 - 讲解接口示例
- 进入正题
 - 前置条件啰嗦一下～
 - 走读代码篇
 - 篇一：go程序启动(跟具体业务逻辑无关，从整体了解项目来说 看一下没坏处)
 - 篇二：go接口走查篇之路由文件
 - 篇三：go接口走查篇之controller层
 - 篇三：go接口走查篇之service层业务逻辑
- 回到最开始我们的困惑，我的回答是：
 - 对于困惑1:
 - 对于困惑2:
 - 对于困惑3:
- 写在后面

写在前面

日常测试过程中是否有过这样的困惑

1. 唉？我明明有这个条件，为啥这条链路就走不通呢？//郁闷.....然后找到开发丢了一串CURL，吧啦吧啦告诉他报错了，你去查一下。
2. 新提测业务不知后端数据放哪儿，不清楚基本的数据“结构”，不知道怎么快速地验证不同条件下的case。（依赖开发改数据，若他没空那就要一直等了.....）
3. 不知道需求的改动范围，回归测试的内容完全依赖开发同步的信息。（很多时候开发不靠谱.....会遗漏某些关键内容）

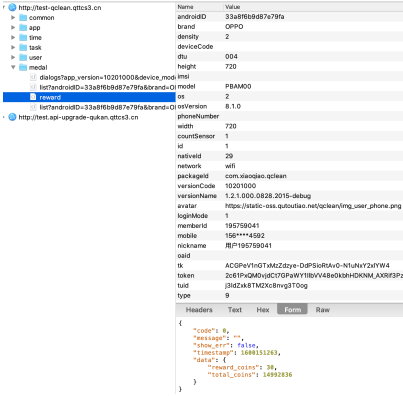
那对于以上困惑，咱们有啥方法解决

答：了解开发的代码实现逻辑，具体到每一个函数下的每一个分支干了什么事情。对代码逻辑了解越清晰，测试更加精准(看不太懂，没关系，多学多看多跟开发沟通，不仅可拉近与开发的“距离”又能学习开发知识；但是不要陷入开发代码中，毕竟写BUG也是家常便饭。

因为从软件测试本质来看，测试其实就是测代码本身，只不过我们目前习惯从UI层去验证程序的可用、稳定性。

讲解接口示例

项目	接口路由	请求方式	Charles抓包	业务场景解释	通过示例可了解

天天爱清理 (qclean)	/medal/reward	post	<div>1、接口请求信息:</div> <div></div> <div>2、接口请求关键参数: (如上图)</div> <div>id、type (对应这是哪个勋章、这个是什么类型的勋章)</div> <div>3、接口返回:</div> <div><pre>{ "code": 0, "message": "", "show_err": false, "timestamp": 1600151263, "data": { "reward_coins": 30, "total_coins": 14992836 } }</pre></div>	<div>用户在爱清理APP中完成了某个任务，比如清理手机垃圾达到500MB即可解锁成就勋章A，点击勋章→播放激励视频，完成观看视频，关闭广告会触发这个领金币接口 /medal/reward</div> <div><ol style="list-style-type: none">1. 如何从接口请求去反推业务代码逻辑~大概做了什么事儿。2. 遇到接口报业务错时，怎么去定位，在代码逻辑处，具体哪里“报错”了。3. 这个报错从业务场景角度来看是否符合正常逻辑。</div>
----------------	---------------	------	--	---

进入正文

前置条件啰嗦一下~

- 1、基本环境准备
- 本地安装一下Go相关依赖
 - 后端项目访问权限。（Reporter权限就行，提交代码Developer权限）
 - 本地建git仓库，代码克隆本地。
 - 在要测试的代码分支上操作。（这里拿master举例）
- 2、了解restful API的分层构成
- 控制器用于接受客户端请求，做一些基本参数校验： controller层（controller层下面的router.go路由文件用于收集请求并转发调用controller层代码）
 - 由controller层函数调用的具体业务逻辑处理： service层
 - 由service层调用的数据持久化： model层

走读代码篇

注：在源码中每行代码上方加入的注释内容（即//后的文字）便是解读。

篇一： go程序启动(跟具体业务逻辑无关，从整体了解项目来说 看一下没坏处)

- 1、在项目/app目录下找main.go文件(爱清理是在 src/code/api/main.go)，main函数就是启动服务的主函数。

```
package main

import (
    "flag"
    "code/api/server"
)

func main() {
    //
    env := flag.String("env", "dev", "runtime env [dev|qa|pre|prd]")

    flag.Parse()
    //

    if *env == "dev" {
        server.NewGracefulServer().Run(*env).Wait()
        return
    }

    //
    //
    //Run()
    server.NewSmoothServer().Run(*env)
}
```

2、这里是上面Run()方法的实现，可以看到这里做了创建web框架/启动服务等操作（如日志，中间件设置，路由控制）

```

func (s *SmoothServer) Run(env string) {
    onBoot(env)

    // gin web
    var engine *gin.Engine
    if graceful.IsWorker() {
        //createEngine()
        engine = s.createEngine()
    }

    //
    opt := &graceful.Option{
        ReadTimeout: 4 * time.Second,
        WriteTimeout: 4 * time.Second,
    }

    //
    if app.Env().IsDev() || app.Env().IsQa() {
        opt.ReadTimeout = time.Hour
        opt.WriteTimeout = time.Hour
    }

    //
    addr := app.Cfg("app").GetString("port")
    sv := graceful.NewServer(opt)
    sv.AddAfterWorkerShutdownHook(func(ctx context.Context) {
        onShutdown()
    })

    //
    err := sv.Register(addr, engine).Run()
    if err != nil {
        app.Log().WithError(err).Panic("smooth server run err")
    }
}

func (s *SmoothServer) createEngine() *gin.Engine {
    engine := gin.New()
    if app.Env().IsPrd() {
        gin.SetMode(gin.ReleaseMode)
    }
    //
    initMiddleware(engine)

    //
    // InitRouter()
    controller.InitRouter(engine)
    return engine
}

```

篇二：go接口走查篇之路由文件

3、下面的路由文件(src/code/api/controller/router.go)会统一收集接口请求，而后匹配调用对应业务的controller层代码，我们在这里找到/medal/reward接口，他调用了controller层(src/code/api/controller/medal.go)的Reward方法(command+鼠标单击对应方法名， 跳转进入看Reward方法)

在调用controller层代码之前，会由自定义的中间件去处理接口请求，做一些基本的反作弊、token解析、qdata加解密的操作，这也为了和具体业务逻辑处理解耦；即中间件专注于功能实现，而controller层专注业务。

我们看到的接口返回的qdata解密失败、权限错误都是中间件层做的事情。

```

func InitRouter(engine *gin.Engine) {
    //
    pprof.Register(engine)

    //
    engine.Any("/ping", Ping)
    engine.GET("/app/start", helper.Handle(App.Start))
    engine.GET("/hc", helper.Handle(App.HcCheck))

    //
    engine.GET("/app/guide", helper.Handle(App.NewGuide))

    //Decrypt()AccessToken()AccessCheat()  qdatatoken
    antiSpamService := engine.Group("/").Use(middleware.Decrypt(), middleware.AccessToken(), middleware.
AccessCheat())
    {
        //;N
        antiSpamService.POST("/medal/reward", helper.Handle(Medal.Reward))

ex := engine.Group("/example").Use(middleware.Decrypt(),middleware.AccessToken())
if app.Env().IsDev() || app.Env().IsQa() {
    ex.GET("/tk2tuid", helper.Handle(example.Tk2TUID))
    ex.GET("/test", helper.Handle(example.GetTuidTransfer))
    ex.GET("/metrics", helper.Handle(example.Metrics))
    ex.GET("/create/tk", helper.Handle(example.CreateTK))
    ex.GET("/get/user/info", helper.Handle(example.GetUserInfo))
    ex.GET("/push", helper.Handle(example.Push))
    ex.GET("/clearPush", helper.Handle(example.ClearCoinPush))
    ex.GET("/recallPush", helper.Handle(example.RecallPush))
    ex.GET("/advCoin", helper.Handle(example.GetAdvCoin))
}
if !app.Env().IsPrd() {
    ex.GET("/reporter", helper.Handle(example.Report))
    ex.POST("/ab", helper.Handle(example.Ab))
}
}
}

```

篇三：go接口走查篇之controller层

4、下面的函数就是从上面路由文件跳转过来的，这里是controller层代码，用于接受请求并做简单判断，而后调用service层代码做具体的业务逻辑处理

```

func (*_Medal) Reward(ctx *helper.Context) *helper.Response {
    //postid
    id, _ := strconv.Atoi(ctx.PostForm("id"))

    //posttypeAtoi
    category, _ := strconv.Atoi(ctx.PostForm("type"))

    //mid
    uid := ctx.GetInt("mid")

    //Reward(ctx, uid, id, category)serviceRewarderr != nilgoerr,err==nil,err !=nil
    if data, err := medal.New().Reward(ctx, uid, id, category); err != nil {

        //err
        return ctx.Output(err)
    } else {

        //;serviceGetAccountCoins(uid)
        totalCoins, _ := ucoin.GetAccountCoins(uid)

        //keytotal_coins;valuetotalCoinsservicekey-value
        data["total_coins"] = totalCoins

        //
        return ctx.Output(data)
    }
}

```

篇三：go接口走查篇之service层业务逻辑

5、这里是service层逻辑，也就是这个接口的核心业务逻辑处理（也就是常说的增删改查！）

```

JSONDecode
func JSONDecode(data []byte, val interface{}) error {
    return json.Unmarshal(data, val)
}

//
func UnLock(uid int) {
    key := cache.GetUserMedalLockkey(uid)
    redis := cache.GetCatfishRedis()
    redis.Del(key)
}

//
func Lock(uid int) bool {
    key := cache.GetUserMedalLockkey(uid)
    redis := cache.GetCatfishRedis()

    //Redissetnx https://juejin.im/post/6844903830442737671
    if sc, err := redis.SetNX(key, 1, time.Second*2).Result(); err == nil && sc {
        return true
    }
    return false
}

func GetMedal(ctx *helper.Context, id int, category int) map[string]interface{} {
    medalList := GetAllMedal(ctx)
    for _, medalCategory := range medalList {
        medals := utils.GetMapValueSliceMap(medalCategory, "list")
        for _, v := range medals {
            mid := utils.GetMapValueInt64(v, "id", 0)
            mcategory := utils.GetMapValueInt64(v, "type", 0)
            if int(mid) == id && int(mcategory) == category {
                return v
            }
        }
    }
}

```

```

    }
}
return map[string]interface{}{}
}

func GetUserMedalInfo(uid int, medal map[string]interface{}) *UserMedalInfo {
    //UserMedalInfo
    //type UserMedalInfo struct {
    //    Status    int    `json:"st"`
    //    FinishDate string `json:"f_date"`
    // }
    var res UserMedalInfo

    //redis key"user:%d:medal:lock", uid, uiduser:197793255:medal
    key := cache.GetUserMedalKey(uid)

    //id+type"%d_%d", category, id -- :11_1
    field := GetMedalIndex(medal)

    redis := cache.GetCatfishRedis()

    //redis hgethget user:197793255:medal 11_lerr == nil
    if data, err := redis.HGet(key, field).Result(); err == nil {
        _ = utils.JSONDecode([]byte(data), &res) //redisres
    }

    //
    {
        "st": "",
        "f_date": ""
    }
    return &res
}

func SetMedalStatusRewarded(uid int, userMedalInfo *UserMedalInfo, medal map[string]interface{}) error {
    userMedalInfo.Status = constant.MedalStatusRewarded
    data, _ := utils.JSONEncode(userMedalInfo)

    key := cache.GetUserMedalKey(uid)
    field := GetMedalIndex(medal)
    redis := cache.GetCatfishRedis()

    return redis.HSet(key, field, string(data)).Err()
}

// serviceReward

func (m *medal) Reward(ctx *helper.Context, uid int, id int, category int) (map[string]interface{}, error) {
    //Lockcontrollercontroller
    if !Lock(uid) {
        // ErrorMedalRewardFailed = NewErr(-40004, "", true)
        return nil, helper.ErrorMedalRewardFailed
    }

    //
    defer UnLock(uid)

    //ctx, id, categorycontrollerid, categoryidtypectx
    medal := GetMedal(ctx, id, category)

    //GetMedal(1 2AB 3id/type)
    //id+typecontroller
    if len(medal) <= 0 {
        // ErrorMedalNotExist = NewErr(-40001, "", true)
        return nil, helper.ErrorMedalNotExist
    }

    //
    userMedalInfo := GetUserMedalInfo(uid, medal)

    // MedalStatusCommon = 00- 1- 2-

```

```

    //st = 0
    if userMedalInfo.Status == constant.MedalStatusCommon {
        // ErrorMedalNotFinish = NewErr(-40002, "", true)
        return nil, helper.ErrorMedalNotFinish

        // MedalStatusRewarded = 2
    } else if userMedalInfo.Status == constant.MedalStatusRewarded {
        // ErrorMedalRewarded = NewErr(-40003, "", true)
        return nil, helper.ErrorMedalRewarded
    }

    //
    if err := SetMedalStatusRewarded(uid, userMedalInfo, medal); err == nil {
        // ~
        days, err := service.GetUserRegisterDays(uid)
        if err != nil {
            // ErrorMedalRewarded = NewErr(-40003, "", true)
            // CoinCharging
            return nil, helper.ErrorMedalRewarded
        }

        // ~
        amount, err := service.GetMedalAmountByDays(days, medal)

        // <0 NewErr(-40003, "", true)
        // ""
        if err != nil || amount <= 0 {
            return nil, helper.ErrorMedalRewarded
        }

        //
        //
        remark := utils.GetMapValueString(medal, "title") // title
        //
        tradeNo := time.Now().String()

        // memberIDID
        //
        err = ucoin.CoinCharging(uid, int(amount), ucoin.SceneMedalReward, tradeNo, remark, ctx)

        // err == nilcontroller
        if err == nil {
            return map[string]interface{}{"reward_coins": amount}, nil
        }

        // controller NewErr(-40004, "", true)
        return nil, helper.ErrorMedalRewardFailed
    }

```

最后，我们回到Charles抓到的接口响应信息，是不是有了点头绪，data["reward_coins"]是service层调用发放金币服务成功后，return（返回）给controller层的key-value，data["total_coins"]是在controller一开始就查好的用户总金币数。

```

{
  "code": 0,
  "message": "",
  "show_err": false,
  "timestamp": 1600151263,
  "data": {
    "reward_coins": 30,
    "total_coins": 14992836
  }
}

```

以上就是一个完整的接口代码走查流程。

回到最开始我们的困惑，我的回答是：

对于困惑1:

从上面完整的代码走查流程就可以知道每个接口返回的错误信息，都是开发代码里已经定义好的（错误code码+错误message）；在看到Charles抓包请求返回错误信息的时候，可以通过看源码配合服务端日志（上面这接口没有打任何log 看日志不现实）的方式去定位具体在service层 是哪个逻辑分支抛出了错误信息，此时跟开发反馈问题会更进一步。（错误信息流转过程：service层return的信息——>controller层去接收再发送——> 客户端-接收响应信息）

对于困惑2:

从上面的走查流程可以看到，这个接口对应的数据存储是放在redis的，对应的redis key已经在常量文件定义好了，读完对应的分支逻辑之后，就可以通过，修改redis key对应value的方式验证不同条件下的case了，也不需要依赖开发的修改。

对于困惑3:

通过code diff即可知道改动范围，同时可精准评估测试范围；

举例：如开发做需求的时候只改动了上面这个接口，那么我们的测试内容就是这个接口调用的每个函数（上面涉及了勋章查询、用户维度下的勋章、变更勋章领取状态、获取用户注册天数、发放金币服务）。如果每个函数每个分支都有了覆盖且测试通过，那么就可以说本次测试是完整且100%有效（case全部针对函数操作的业务去设计）；如果开发改动涉及了其它文件，我建议是先自己过一遍改动内容，然后与研发沟通确认（到底是夹带私货？是修复BUG？还是其它什么情况？总之需要明确每次改动内容），这样才能做到不漏测，也不用花费多余的时间在不必要的回归测试上。

gitlab的code diff（代码差异性比对）操作步骤参考：

- 1. 打开对应项目仓库。
- 2. 下图位置处进行操作。红框1为开发分支；红框2为主干分支。
- 3. 点击【Compare】即可知道开发本次改动内容。

举例：下面这个分支开发提交了 1564次；改动文件1000个、新增37361行代码、删除75221行代码。



文件变更1000个；新增37361行代码；删除75221行代码

Showing 1000 changed files with 37361 additions and 75221 deletions

细心的同学是不是发现这里接口没有涉及model层数据持久化？我只能说 RZ的业务太简单，在service层就操作Redis把数据输出了....不需要那么麻烦哈

写在后面

有同学可能会问？读接口代码需要掌握哪些知识点？

答：我个人的建议是~不需要系统的学习Golang（毕竟语言之间都是相通的），可以有针对性的了解下：

- 分布式锁redis setnx
- Json Unmarshal: 将json字符串解码到相应的Go结构体
- go结构体
- strconv数据类型转换
- go常规的异常处理err !=nil

单从RZ工具产品-后端接口做的事儿来说，基本流程就是：

1. 拿到请求信息，并筛选关键参数
2. 调用服务，传筛选好的参数
3. 在redis中做查、改操作
4. 调用金币发放服务发放金币

以上都是个人理解，大体可做参考，有错误还请自行更正。

严禁复制