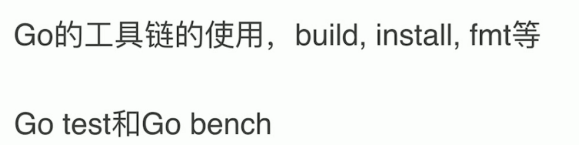
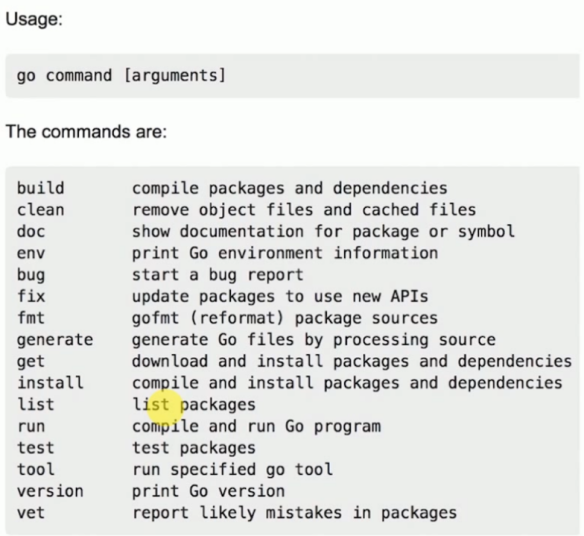
# 简介

## 入门



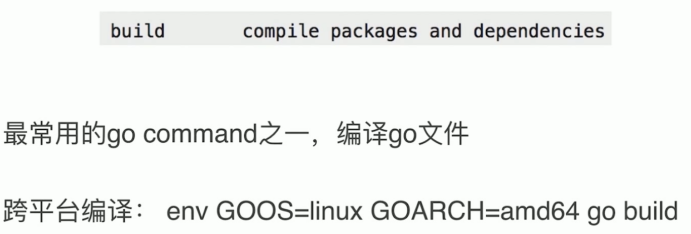
## 工具链

### 集合



### Build

编译连接



### Install

就是说这个与build的区别就是把打包文件存放至pkg里面

Go install编译安装并且打包了



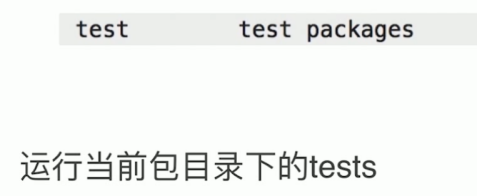
### Get



### Fmt



### Test



Go test （打印结果）/go test -v（打印出详细信息）

## test

### 样式



### 分析

go test 的作用就是执行文件名字为\_test.go的文件

不会执行普通文件名称的 \_test.go文件会调用.go文件里面的函数

### 代码



### test的写法

#### Testing包

#### 方法名Test\*\*\*

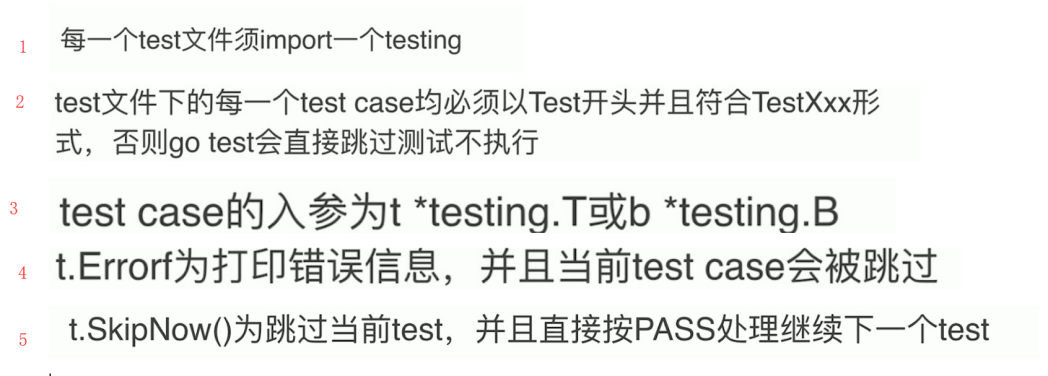
#### 参数t \*testing

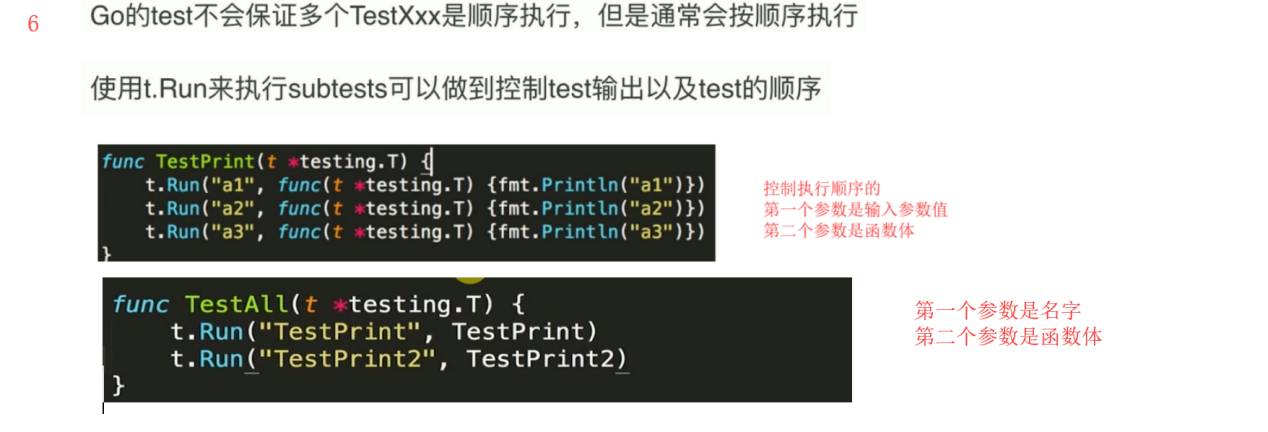
#### t.errorf

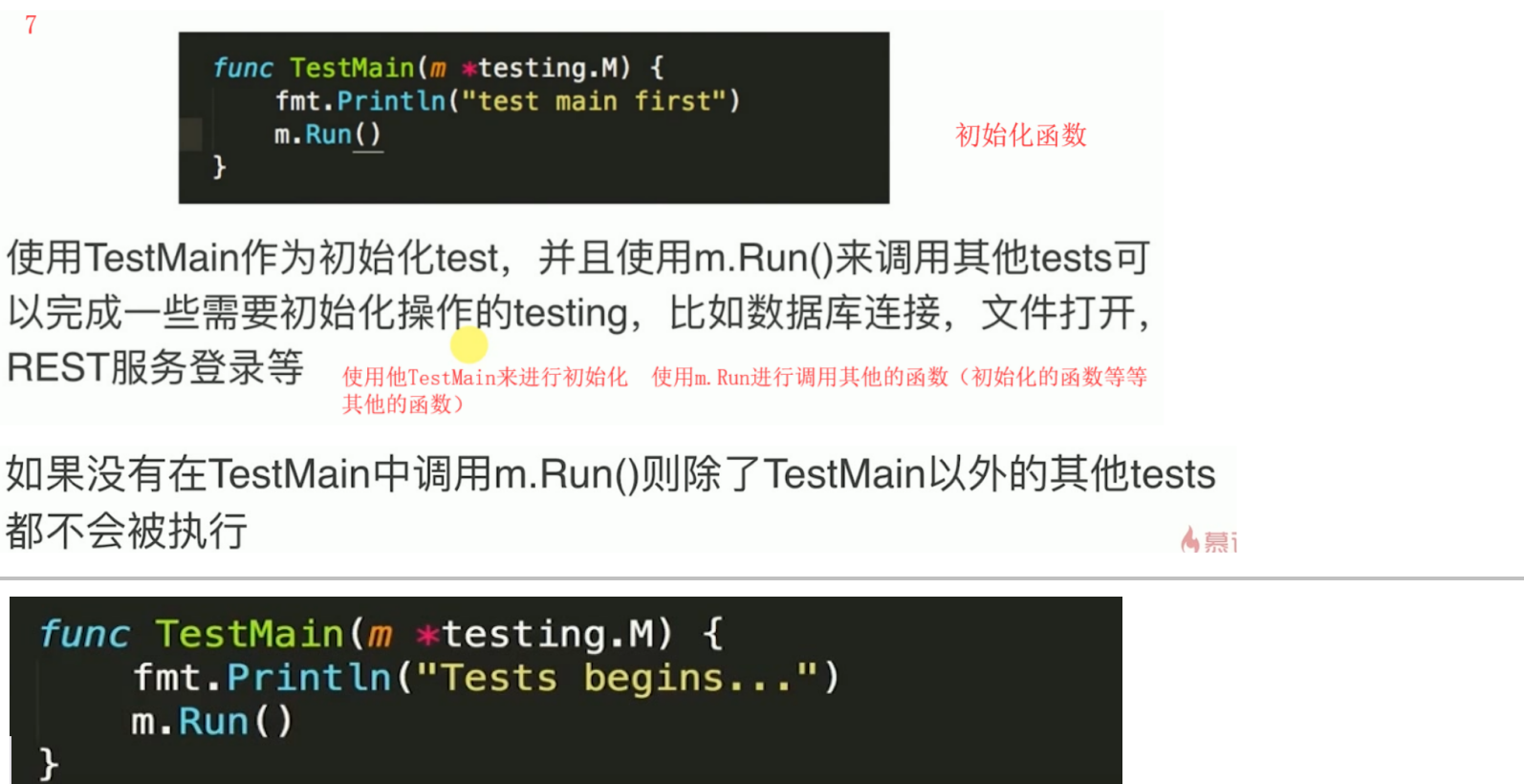
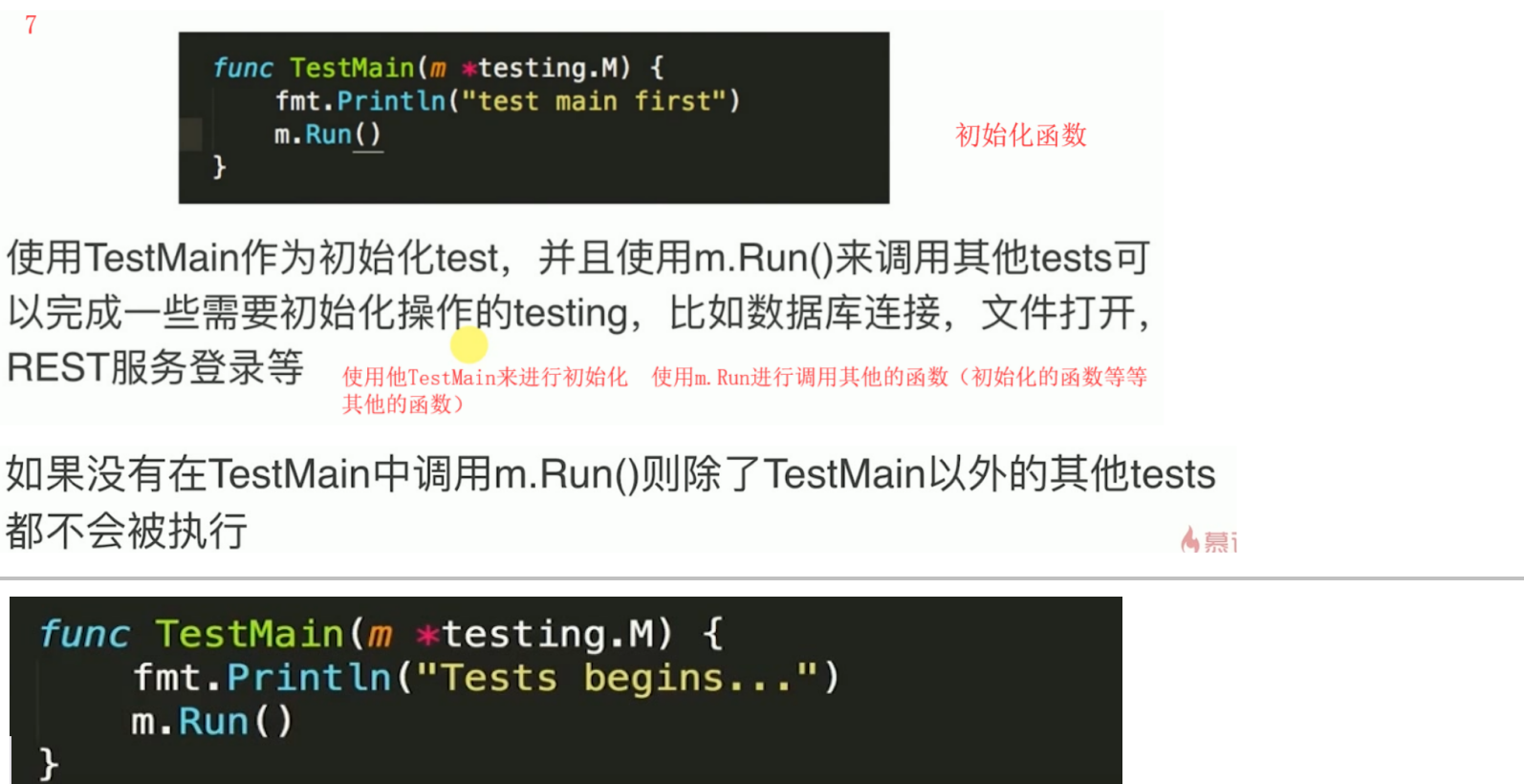
#### T.skipNow

#### T.run

#### TestMain







### benckmark的写法

#### 简介

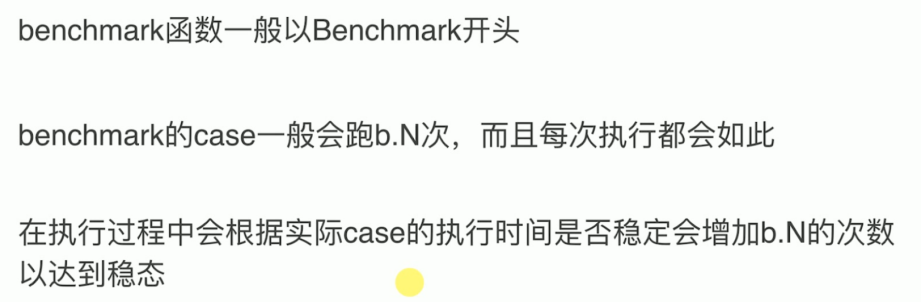
用来测试对程序的性能测试 计算每次执行的花费时间

#### 命令行

执行benchmark的函数体

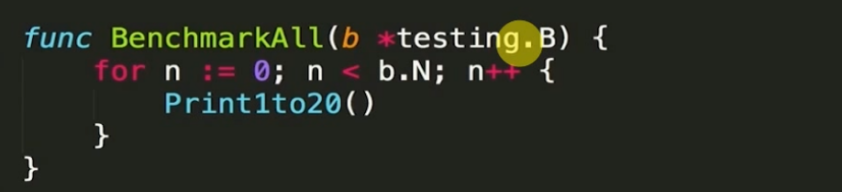
go test -bench=.

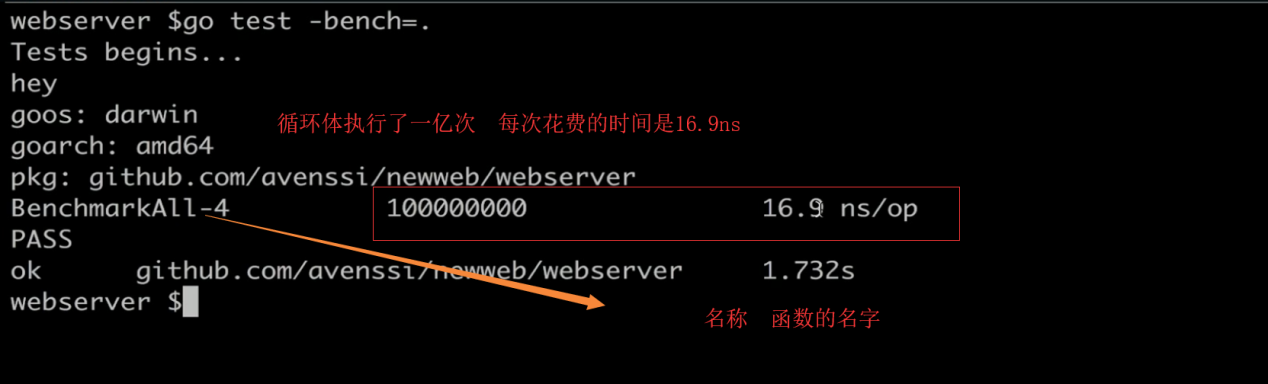
#### 说明



#### 代码

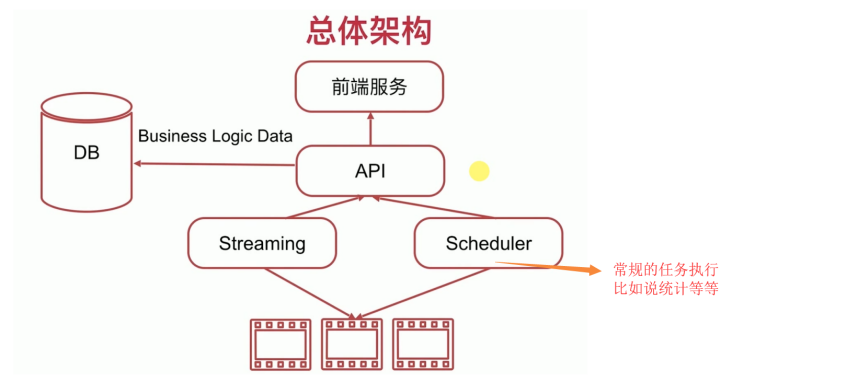
以benchmark开头 这个程序会执行b.N次 在程序执行最稳定的时候 计算出每次执行这个程序花费的时间



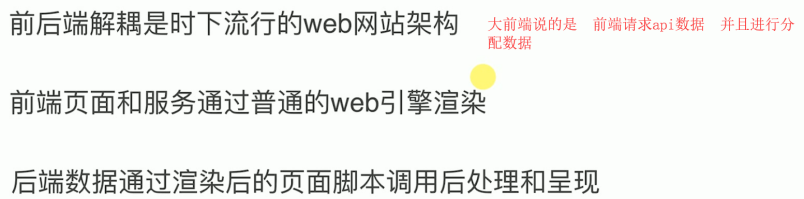


# 项目

## 架构

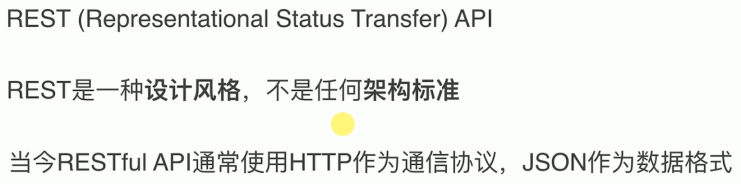


## 前后端解耦



# API

## Restful

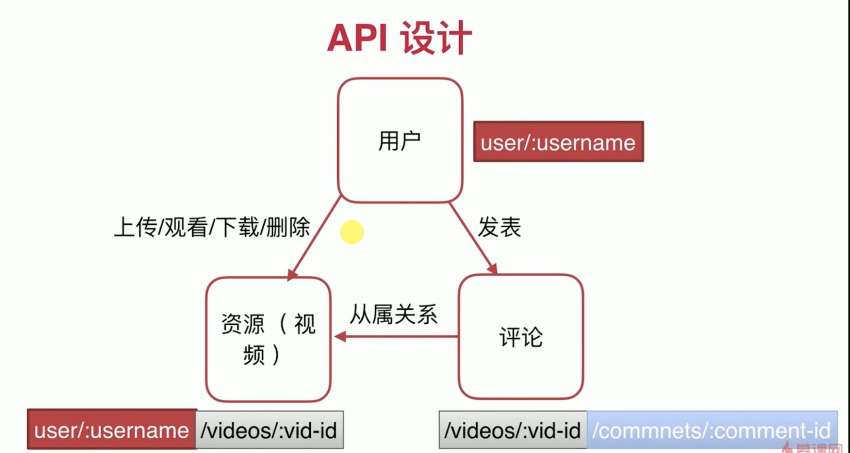


## 设计原则



## 项目思路

### 整体



### 用户



### 视频



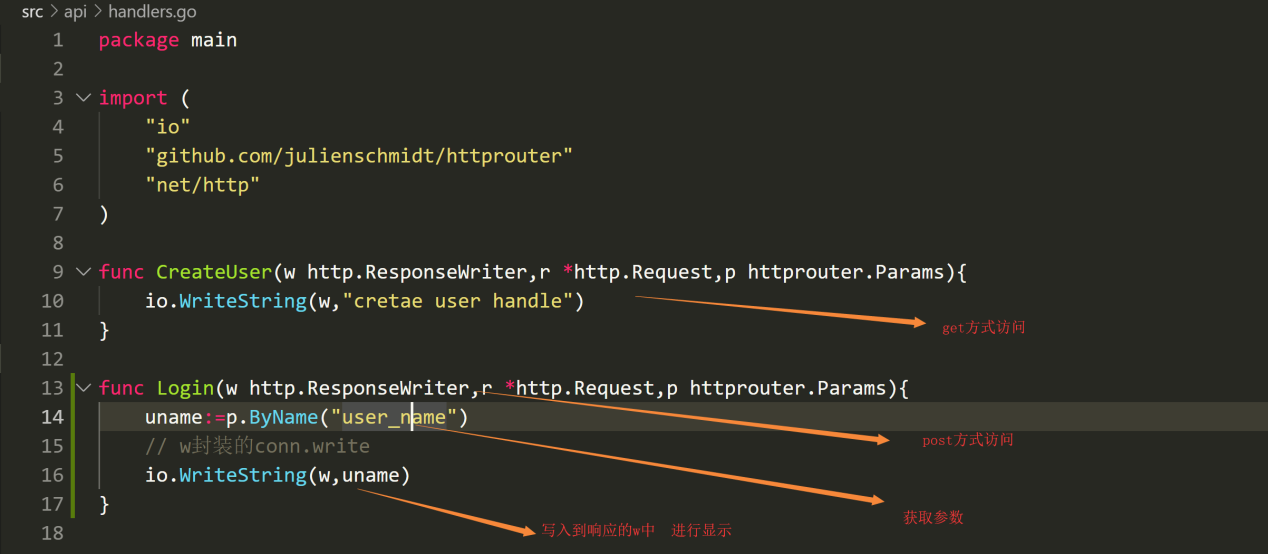
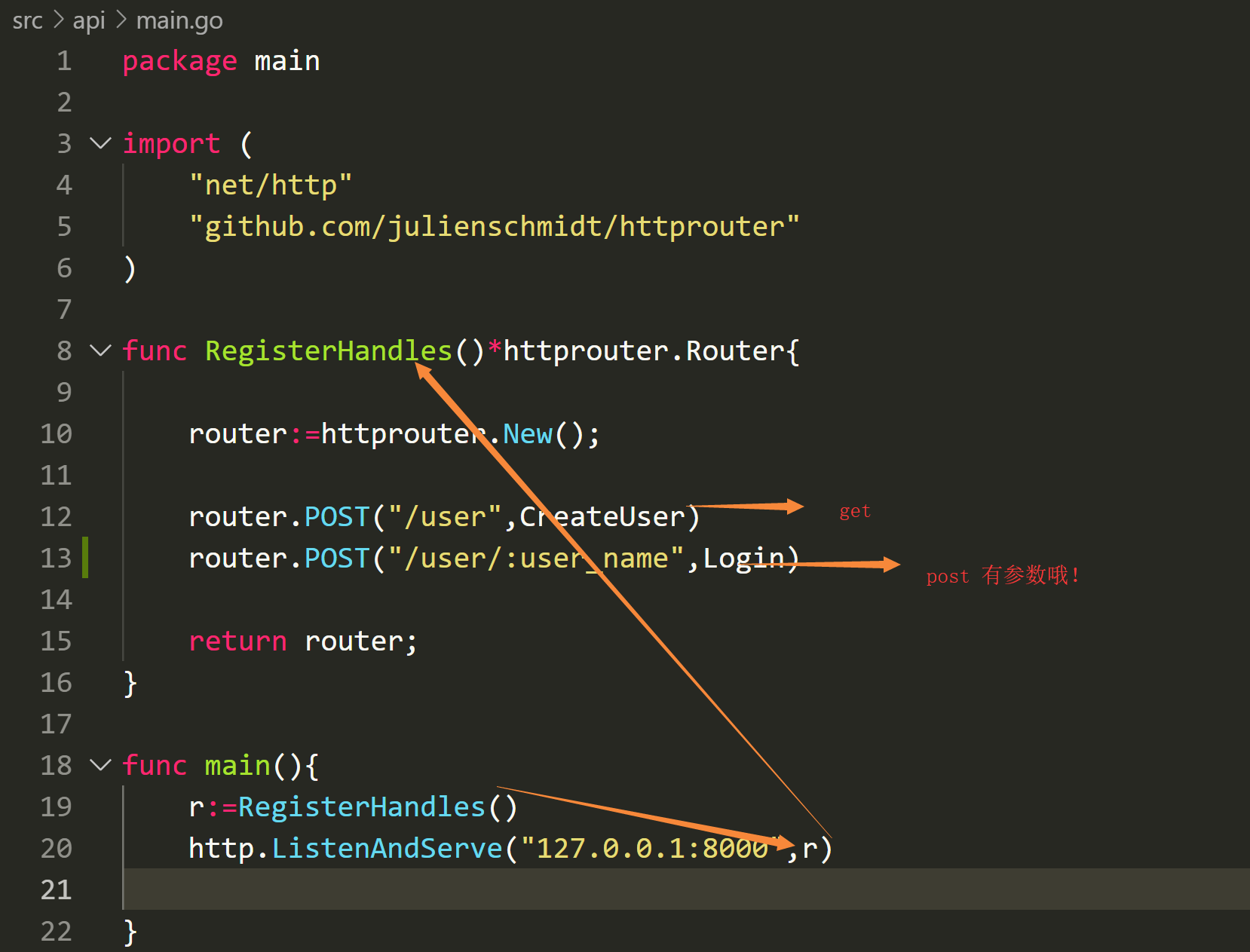
### 评论



## 项目流转

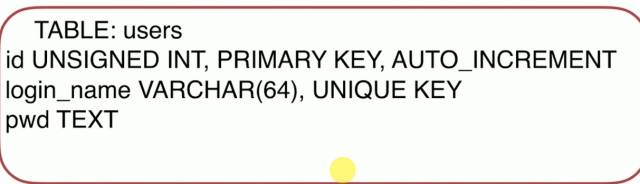
1. Main.go
2. Handle（有路径）
3. Validate (参数 请求路径校验)
4. Logic（dbops会调用消息类型的定义 数据库的操作）
5. 返回response

## get访问与post访问



# 数据库操作

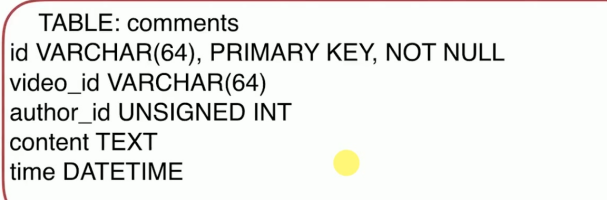
## 用户表



## 视频表



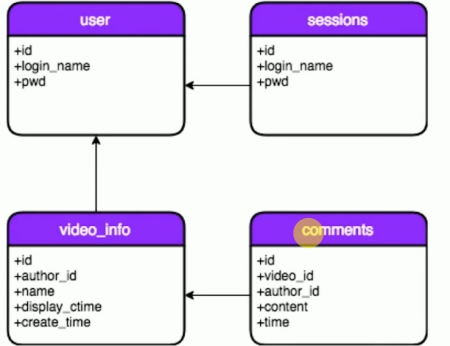
## 评论表



## Session表



## 表之间关系

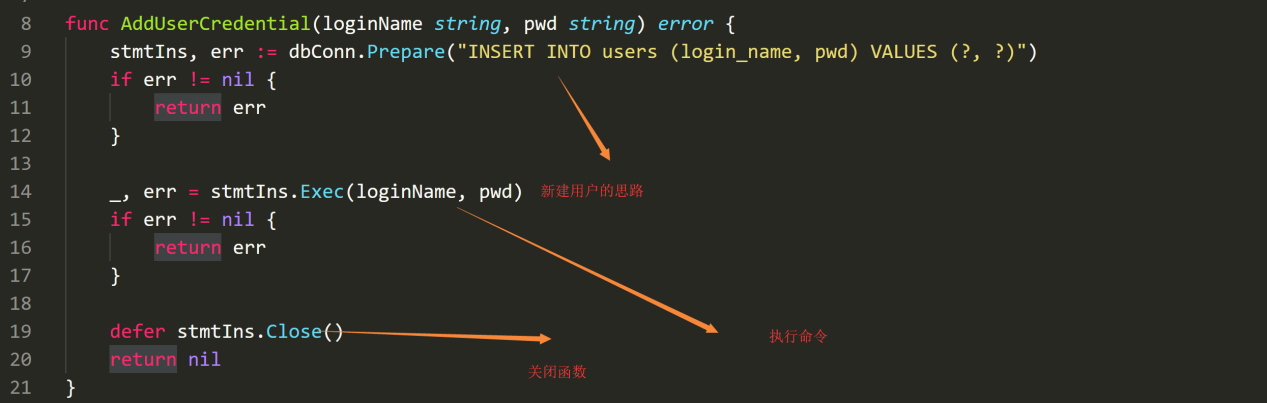


## sql初始化

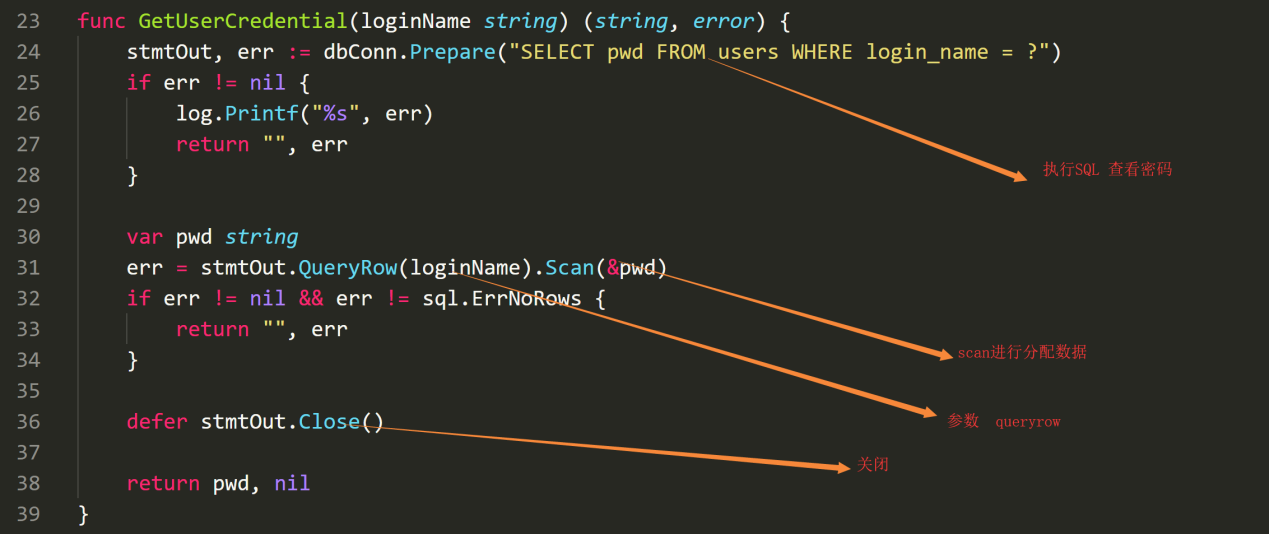


# 用户模块

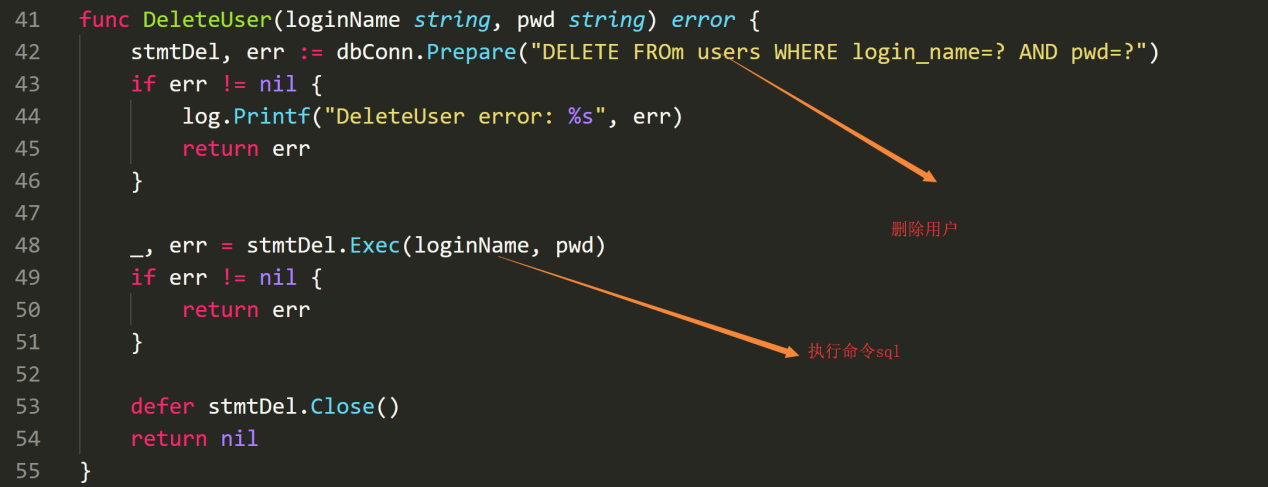
## 方法一：新建用户



## 方法二：验证用户



## 方法三：删除用户



## Test测试用户模块

### 思路

1. 连接数据库
2. 清空表格
3. 运行test文件
4. 清除db的测试数据

### run()

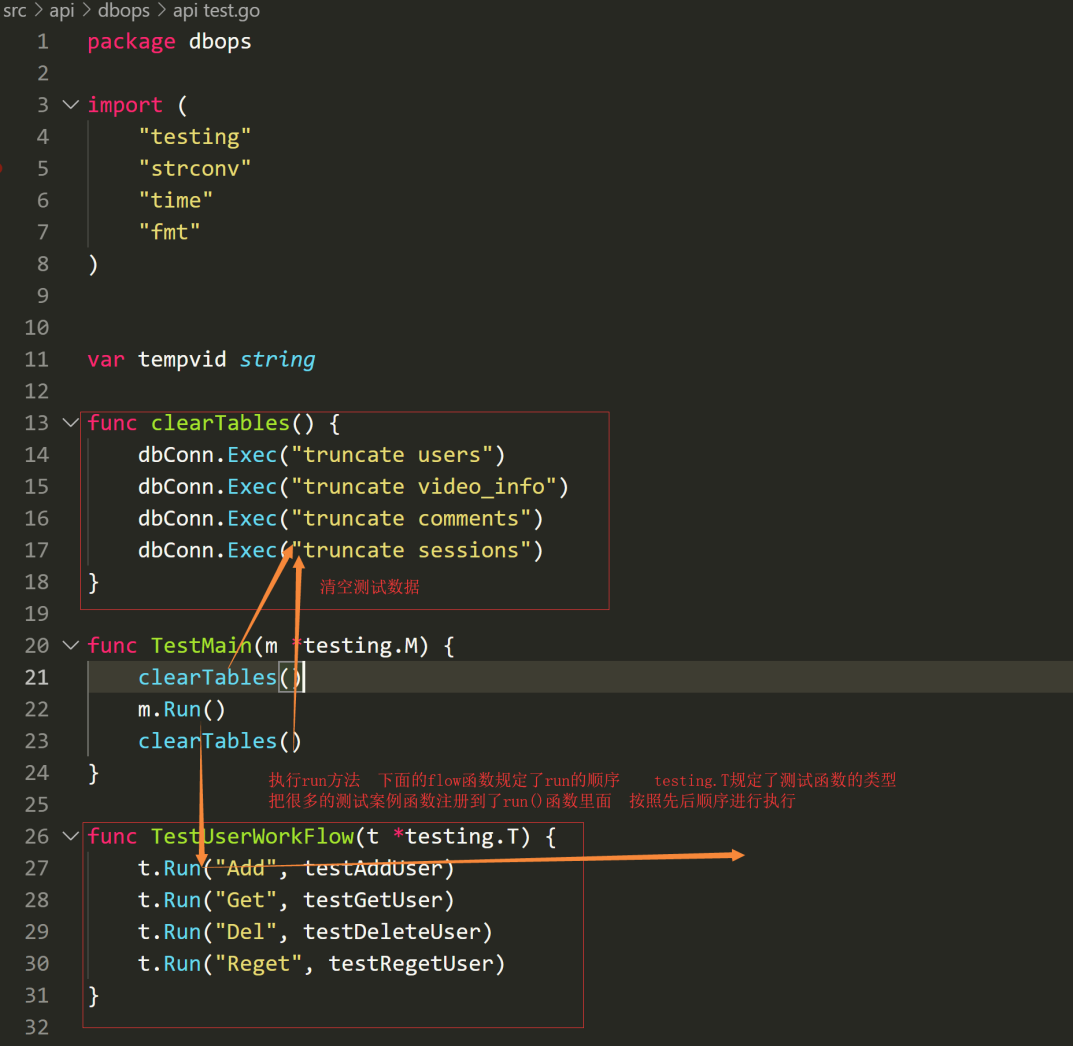
Run() flow函数把测试案例函数注册到了run()

就产生了执行顺序

### 代码

Run() flow函数把测试案例函数注册到了run()

就产生了执行顺序



# 视频模块

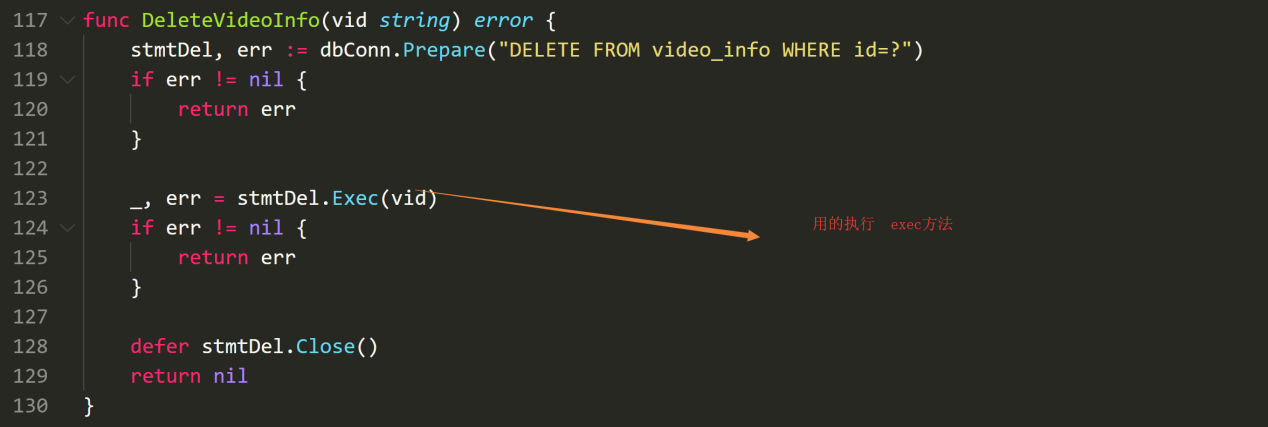
## 方法一：添加



## 方法二：查询一个



## 方法三：删除一个



## 查询数据库方法

增删改都用的exec方法

查询一条用queryrow方法

查询多条用query方法

## Test测试视频模块

func TestVideoWorkFlow(t \*testing.T) {

    clearTables()

    t.Run("PrepareUser", testAddUser)

    t.Run("AddVideo", testAddVideoInfo)

    t.Run("GetVideo", testGetVideoInfo)

    t.Run("DelVideo", testDeleteVideoInfo)

    t.Run("RegetVideo", testRegetVideoInfo)

}

func testAddVideoInfo(t \*testing.T) {

    vi, err := AddNewVideo(1, "my-video")

    if err != nil {

        t.Errorf("Error of AddVideoInfo: %v", err)

    }

    tempvid = vi.Id

}

func testGetVideoInfo(t \*testing.T) {

    \_, err := GetVideoInfo(tempvid)

    if err != nil {

        t.Errorf("Error of GetVideoInfo: %v", err)

    }

}

func testDeleteVideoInfo(t \*testing.T) {

    err := DeleteVideoInfo(tempvid)

    if err != nil {

        t.Errorf("Error of DeleteVideoInfo: %v", err)

    }

}

func testRegetVideoInfo(t \*testing.T) {

    vi, err := GetVideoInfo(tempvid)

    if err != nil || vi != nil{

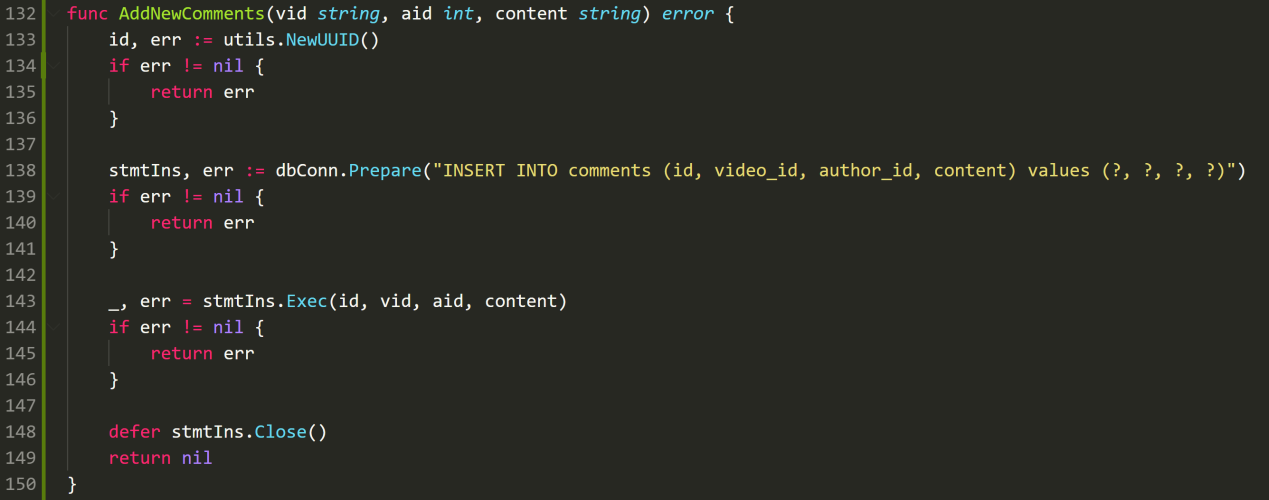
        t.Errorf("Error of RegetVideoInfo: %v", err)

    }

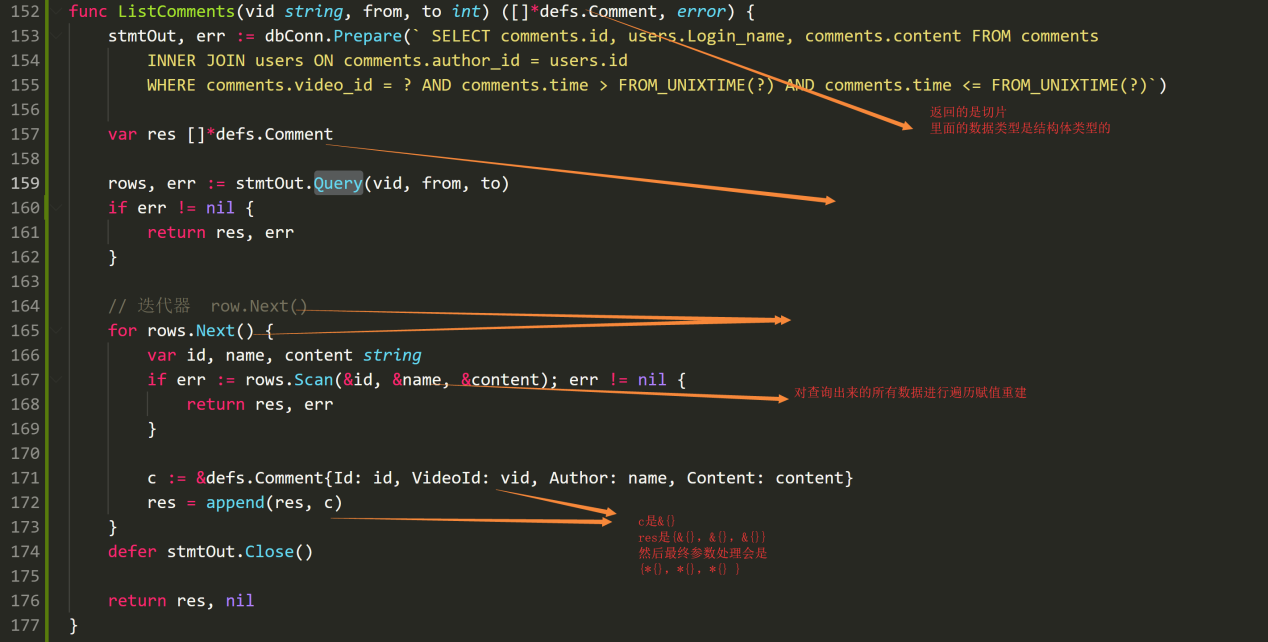
}

# 评论模块

## 方法一：添加



## 方法二：列出评论



## Test测试评论模块

// 测试评论

func TestComments(t \*testing.T) {

    clearTables()

    t.Run("AddUser", testAddUser)

    t.Run("AddCommnets", testAddComments)

    t.Run("ListComments", testListComments)

}

func testAddComments(t \*testing.T) {

    vid := "12345"

    aid := 1

    content := "I like this video"

    err := AddNewComments(vid, aid, content)

    if err != nil {

        t.Errorf("Error of AddComments: %v", err)

    }

}

func testListComments(t \*testing.T) {

    vid := "12345"

    from := 1514764800

    // atoi字符串转化为整型

    // 将整数转换为字符串形式

    // 第一个是数字

    // 第二个是进制

    to, \_ := strconv.Atoi(strconv.FormatInt(time.Now().UnixNano()/1000000000, 10)

    res, err := ListComments(vid, from, to)

    if err != nil {

        t.Errorf("Error of ListComments: %v", err)

    }

    for i, ele := range res {

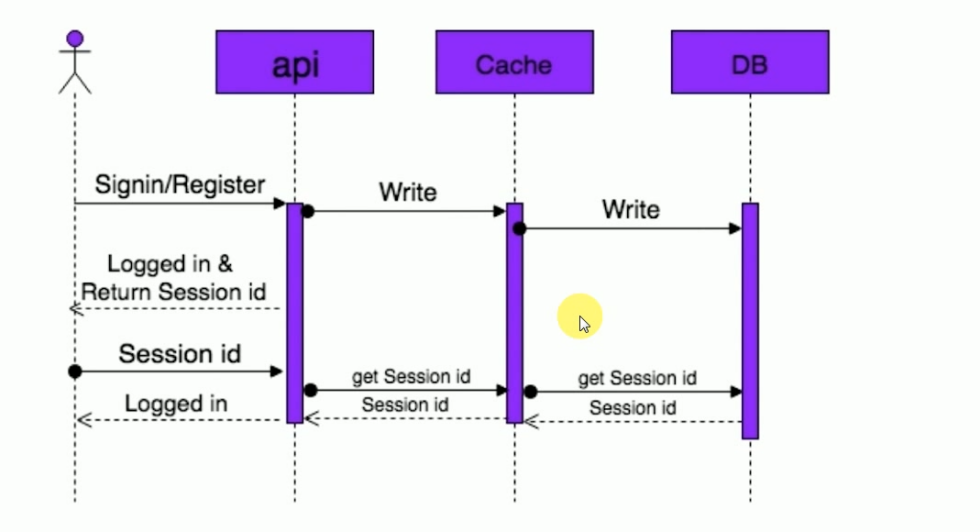
        fmt.Printf("comment: %d, %v \n", i, ele)

    }

}

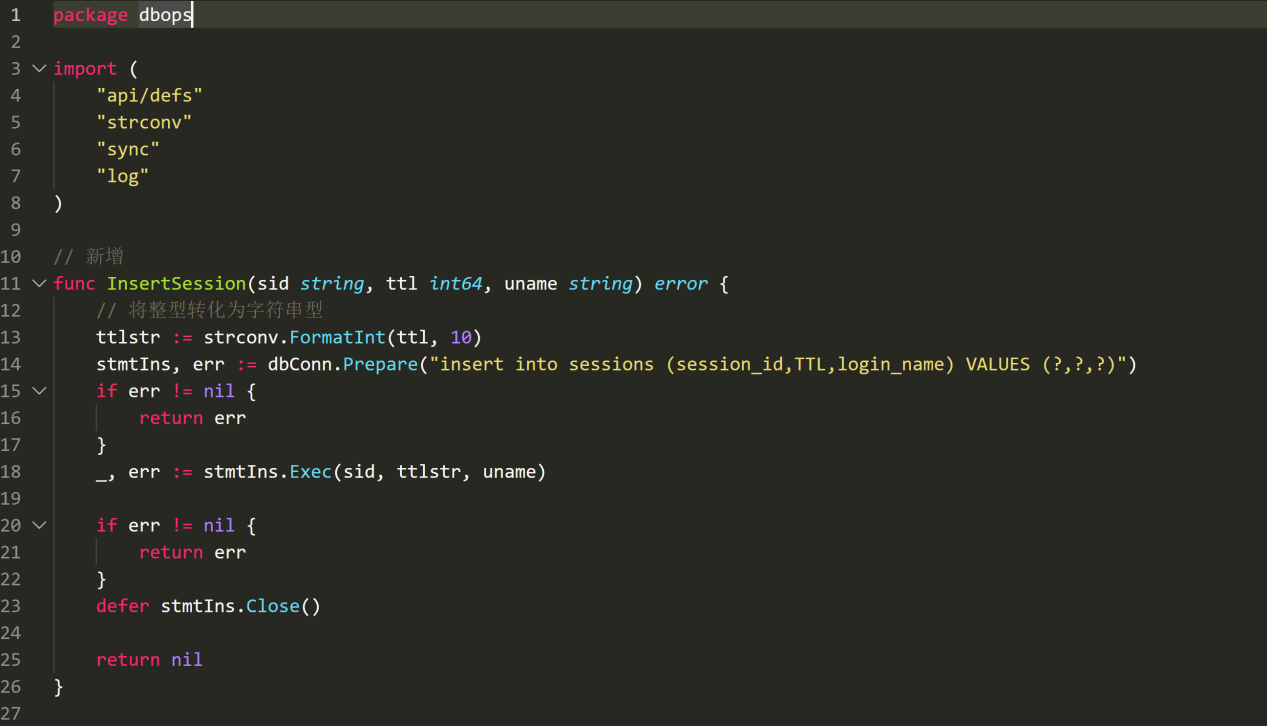
# Session

## 简介

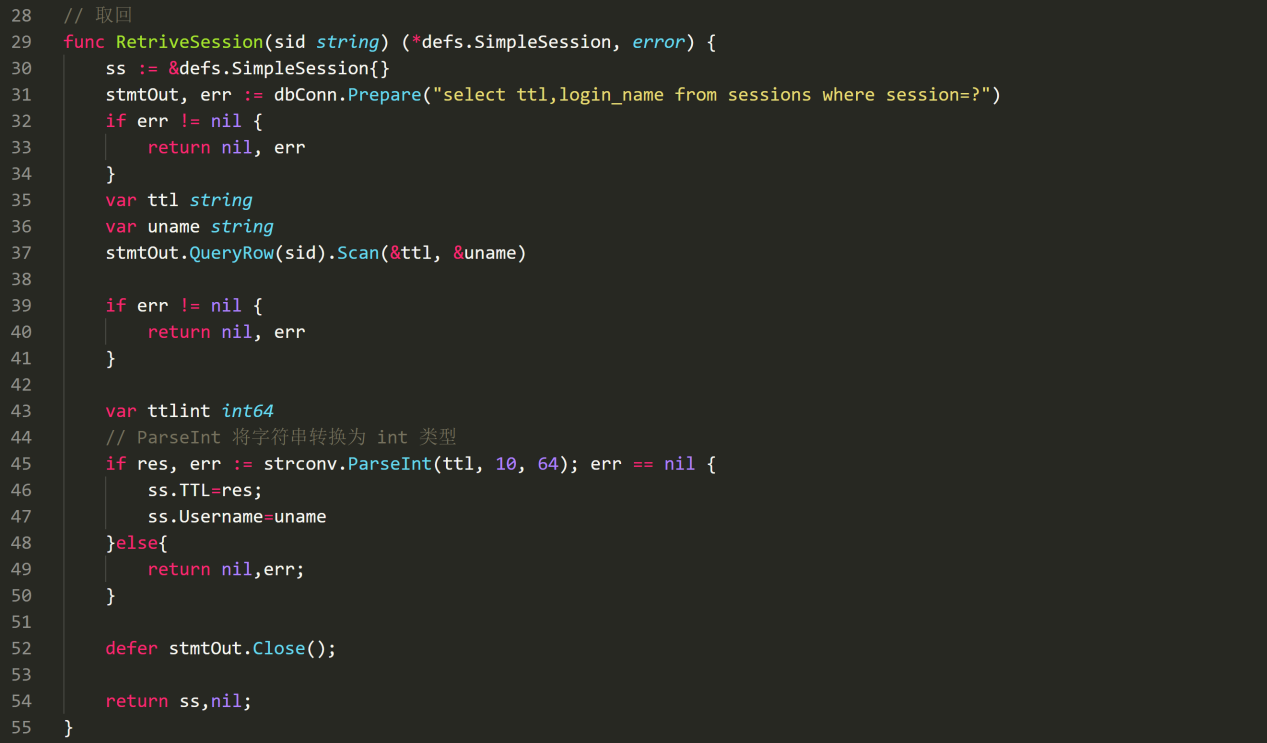


## DB

### 方法一：数据库存储session



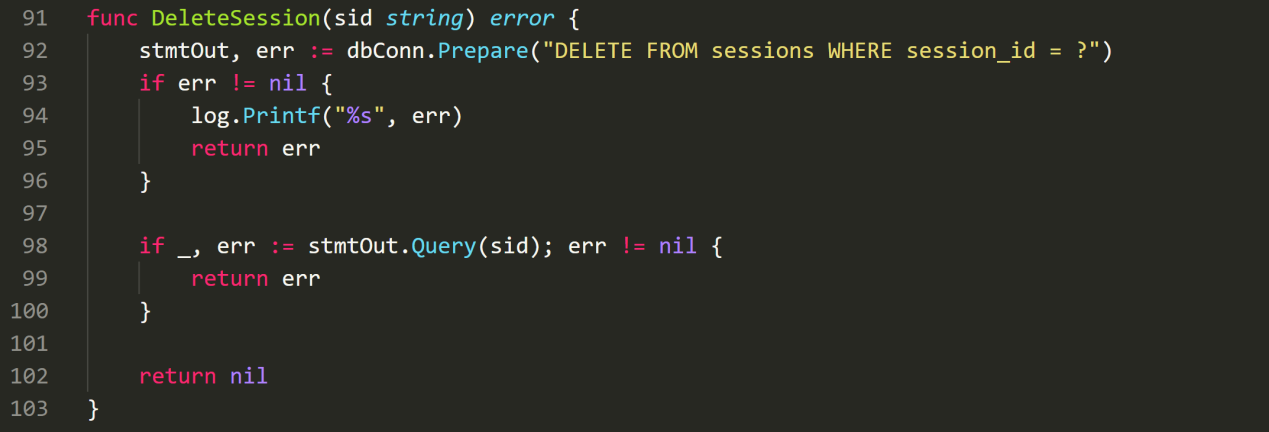
### 方法二：取回session



### 方法三：取回所有session



### 方法四：删除session

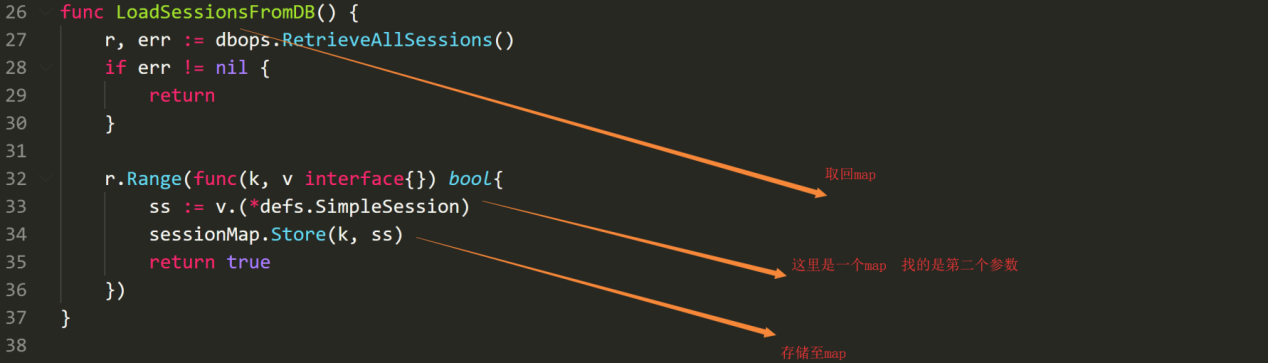


## 缓存

### 方法一：初始化



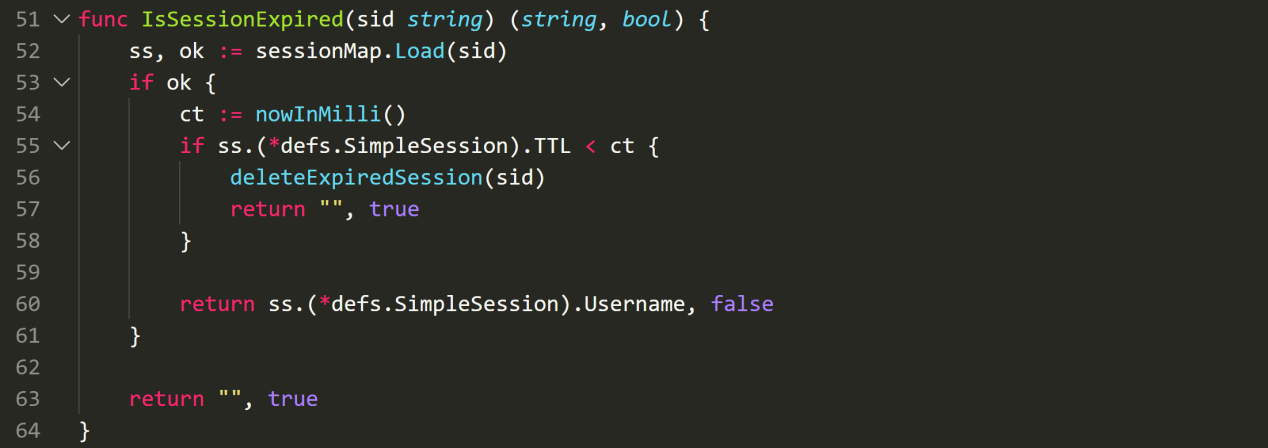
### 方法二：从DB加载至缓存



### 方法三：产生sessionid存DB与缓存



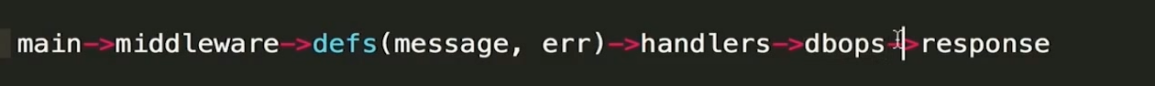
### 方法四：判断过期



# 项目流程

## 流程

1. Main.go进入
2. 执行middleware进行校验鉴权 检验访问的权限等等
3. 进行执行defs 定义数据类型错误类型消息体
4. Handle定义路由等
5. 执行dbops.逻辑处理
6. Resopnse响应处理



## Ducktype

https://blog.csdn.net/fujian9544/article/details/100098714

# MiddleWare与Handler

## 简介

Middleware是校验数据分配权限

http.router是一个类struct的方法

它其实是实现了http的handle接口的serverHTTP方法

我们要做

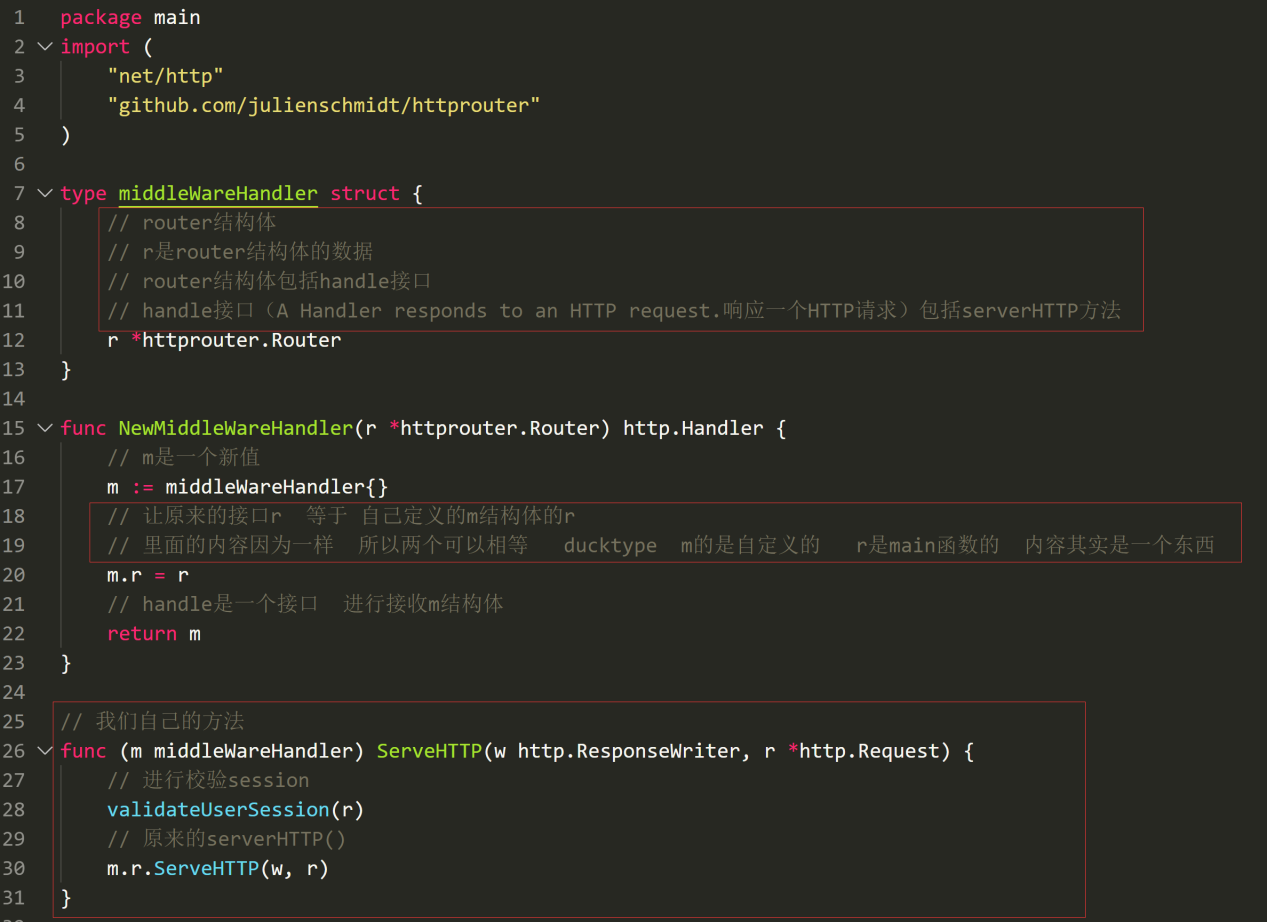
修改重写serverHTTP方法

就是修改handle接口interface

然后http.router就会变化

这样就实现了在http.Router前面进行执行我们想要的操作

## 重写serverHTTP



# Streaming

## 简介

播放视频

上传视频

流控部分

## Bucket token

多个协程进行访问的时候

使用共享通道进行同步协程之间的信息

