## Go

func test(arr \*[]string) {  
 \*arr = append(\*arr, "22222")  
}

现在明白了，普通的数组当参数传到函数里面是没用的，得用这样的方式，传进去一个数组指针才会影响到外面。 \*[]

map同理

获取map的代码

if tfPlanInfo, ok := res.TfPlanMap[tfPlanId]; ok {

 }

go中的方法和传统的类的方法不太一样，方法和类并非组织在一起，传统的oop方法和类放在一个文件里面，而go语言只要在同一个包里就可，可分散在不同文件里。go的理念就是数据和实现分离，

go的结构体和属于这个结构体的方法可以在不同的文件里，但是要在同一个包里

把一个interface{}类型的东西转成他真正的类型。

if res, ok := data.TfPlanDH.Data.(\*data.TfPlanData); ok {

if err = conn.Table("tf\_plan").Where("plan\_id =?", tfPlan.PlanId).Updates(tfPlan).Error; err != nil {

  logs.CtxWarn(ctx, "UpdateTfPlan Updates error: %s", err)

  return err

 }这种更新的时候传进来的结构体需要有id的

这种就会像java那样，你的结构体的元素不是空就会去更新，是null的就会保持原来的。

func test(arr \*[]string) (s string){

这样的话，看到了么，返回值列表既定义了一个类型，顺便也创建了这个返回的变量，代码里对s进行复制操作就可以了，但是这种返回方式必须要用小括号扩起来

if err := json.UnmarshalFromString(item.PlanContent, &tfGradeList); nil != err {

这是正确的json代码，要有err ，而且还要带上string

init 函数不需要参数并且也不返回任何值，与 main 类似，标识符 init 没有被声明所以也就不能被引用。

go func(){fmt.Println("Hello inline")}()

        //call a function as goroutine

        go printHello()

但是呢，go运行一个用户线程的时候，函数名后面必须是个（）

所以即使第一行的是一个匿名函数，后面也要加一个（）

:= 结构不能使用在函数外，函数外的每个语法块都必须以关键字开始

var (

    a int

    b string

    c []float32

    d func() bool

    e struct {

        x int

    }

)          批量定义变量

go语言很多的那流程操作都是在init里面，如果不看init单撸代码的话可能看不到

channel是一个定义比较简单的携程之间的消息队列啊，因为channel是多个的。

看到了么，函数指针的执行。

        if (eventActFun)

        {

            eventActFun();

声明时的\*eventActFun的\*只是声明作用，下面用的时候还是要用eventActFun

返回值（\*函数名）（参数列表）

是在函数名前面顶了一个\*，这个就是函数指针。

var ptr \*\*int; 指针的指针

go的携程是用户线程。

numbers = append(numbers, 2,3,4) 还可以追加切片和数组。

numbers[1:4] 这样截取的话包括索引1不包括索引4

slice1 := make([]type, len) len是切片的初始i长度，指定0就可以，我们可以往上在append。

/\* 拷贝 numbers 的内容到 numbers1 \*/

copy(numbers1,numbers) 切片复制

range就是foreach呗

**那种既不是大写也没有指定结构体的方法就是包内可以调用，那种大写的是包名调用（这种是类似与静态）。 而实现接口的方法或者结构体自己的方法就是结构体指针调用。**

**包名调用的还有就是那种文件var定义的全局变量，还有就是包下的结构体信息，其他包想创建这个包定义的结构体的时候。**



go的依赖会先去vendor里找，然后在去gopath里面找。

Go的数组会如果长度超了会扩容和java的listy差不多，关键的是lsit的扩容是list对象里面的buff变成新的，list指针不会变，而go的扩容后数组的地址都变了！！！！

## init函数

注意，init的顺序，上面的还没跑到他的init函数，你就去调用他。

## Error

go的error不像java的异常是分很多类型的，而是一个接口，一般可以返回error的时候根据error的字符串匹配返回error的类型。或者继承error定义新的error。

## 协程与channel

go语言的channel天生就是线程安全的，可以被多协程同时读写。

协同程序（coroutine）与多线程情况下的线程比较类似：有自己的堆栈，自己的局部变量，有自己的指令指针（IP，instruction pointer），但与其它协同程序共享全局变量等很多信息。

也就是操作系统执行起来肯定还是一个个线程的跑，只不过多个协程在同一个线程中被调度，一会跑跑协程a，一会跑跑协程b，一会跑跑协程c，这三个轮着来的，这样的好处就是一个协程阻塞了其他线程也可以跑？

所以线程的调度是操作系统在做，但是一个线程中协程的调度是go的源码在做，所以我们也可以一定程度上认为协程是用户线程。

线程中协程的调度是go源码在做，线程的调度是操作系统在做。

## Kite

注册中心用的consule。



过载就是服务器抗不住了，你的下游找你让你别调用他了，你就可以配置下限流。



熔断的意思和以前差不多。

限流和熔断都是保护下游不被打挂的。



集成了一些公共的告警比如cpu过高，错误日志过多，协程太多。

## Sql

SELECT \* FROM table WHERE zongbu NOT REGEXP'北京|上海|深圳|天津|香港|沈阳';

多个not like无论是or还是and都不好使，要用这个。

如果实现了结构体中的String方法，在使用fmt打印时候会调用该方法，  类似于java中的tostring

## Protobuf

protobuf比json更加的块 速度。时间效率空间效率都是json的3-5倍

repeated GradeInfo grade\_list = 5;//年级列表  在pro里面这个repeated表示的意思是list

可以向前兼容向后兼容

有代码生成机制

## Tips

接口去抖就是防止用户手抖点击多次，导致瞬间调用接口两次。

大力这里用的也不是protobuf。如果真用的是protobuf的话，那么你的http请求响应体李的内容你就看不懂了，因为那里都是乱码了

centos yum mac brew debain apt

ssh的生成是由git登陆的邮箱生成的，每输入一次生成的都不一样，但是私钥文件是在你本地的，公钥配置在远端。

设置ssh key免密登陆服务器，把自己的公钥配置在服务器的.ssh文件夹下的一个文件里。