**Go学习笔记**

**V1.0.0**

**Author: clearuo**

Date: 2017-03-05

# 基础语法

## 基础类型和运算符

1. s:=123这种定义是int32还是int64还是其他类型？试着赋值给其给uint32或int64看是否需要强制转换

s:=123这种定义是int类型,需要强制转换才能赋值给uint32或uint64

2. 在 fmt.Printf 中使用下面的说明符来打印有关变量的相关信息：

* %+v 打印包括字段在内的实例的完整信息
* %#v 打印包括字段和限定类型名称在内的实例的完整信息
* %T 打印某个类型的完整说明

3.在 Go 语言中，&& 和 || 是具有快捷性质的运算符，当运算符左边表达式的值已经能够决定整个表达式的值的时候（&& 左边的值为 false，|| 左边的值为 true），运算符右边的表达式将不会被执行。利用这个性质，如果你有多个条件判断，应当将计算过程较为复杂的表达式放在运算符的右侧以减少不必要的运算。

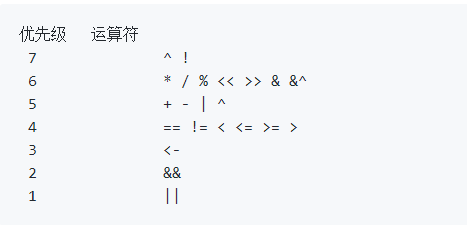
4.格式化说明

在格式化字符串里，%d 用于格式化整数（%x 和 %X 用于格式化 16 进制表示的数字），%g 用于格式化浮点型（%f 输出浮点数，%e 输出科学计数表示法），%0d 用于规定输出定长的整数，其中开头的数字 0 是必须的。

%n.mg 用于表示数字 n 并精确到小数点后 m 位，除了使用 g 之外，还可以使用 e 或者 f，例如：使用格式化字符串 %5.2e 来输出 3.4 的结果为 3.40e+00。

5.运算符与优先级

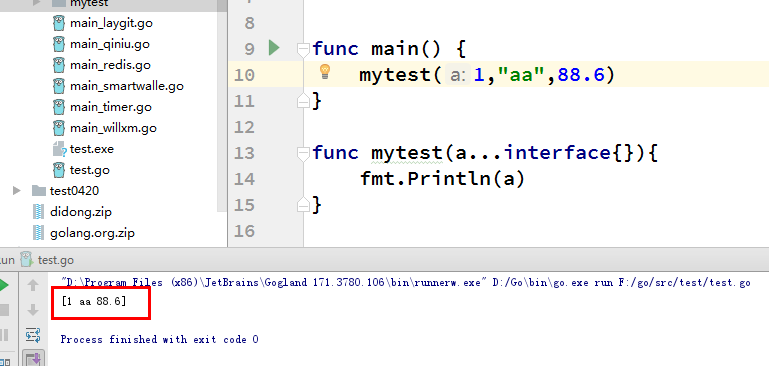
有些运算符拥有较高的优先级，二元运算符的运算方向均是从左至右。下表列出了所有运算符以及它们的优先级，由上至下代表优先级由高到低：



6.获取字符串中某个字节的地址的行为是非法的,例如:&str[i]

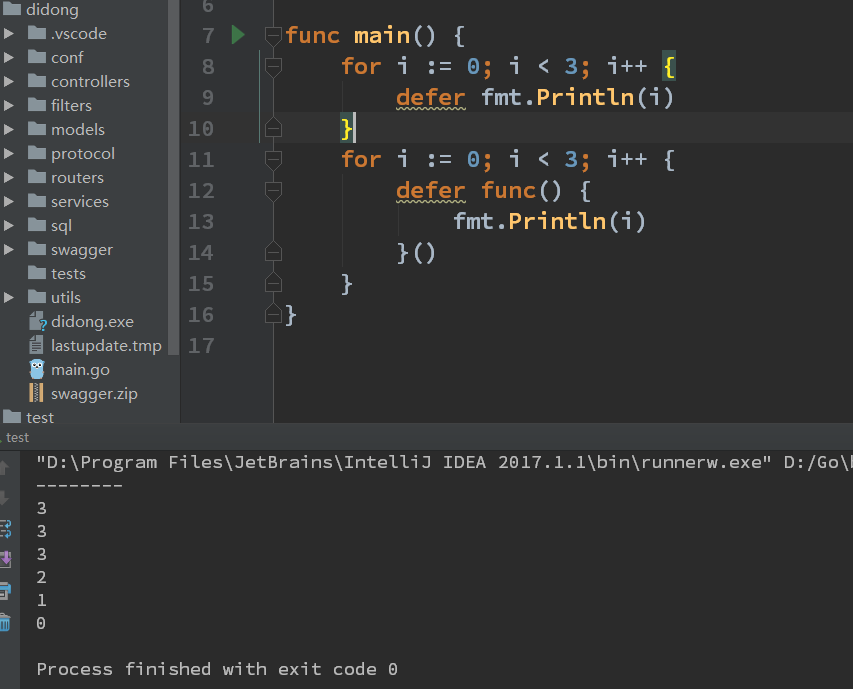
## 函数

1. func test(a…interface{})传递验证

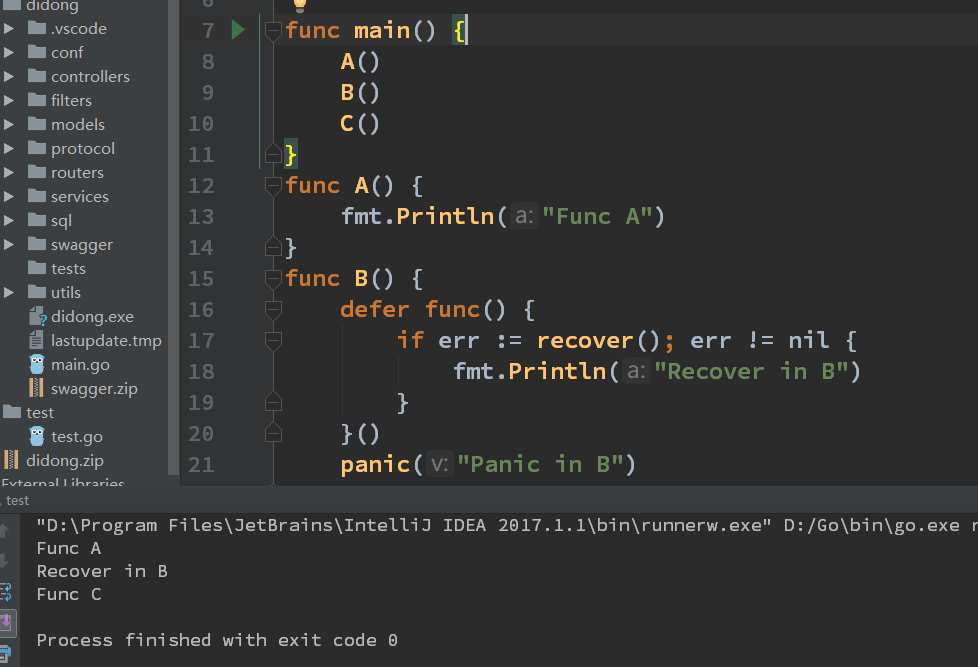


1. 闭包



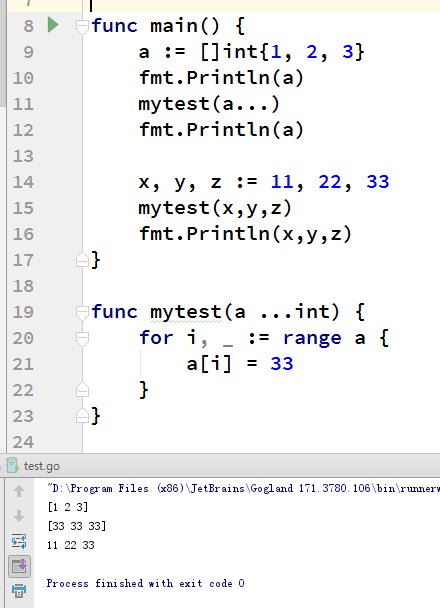


1. Panic,Recover

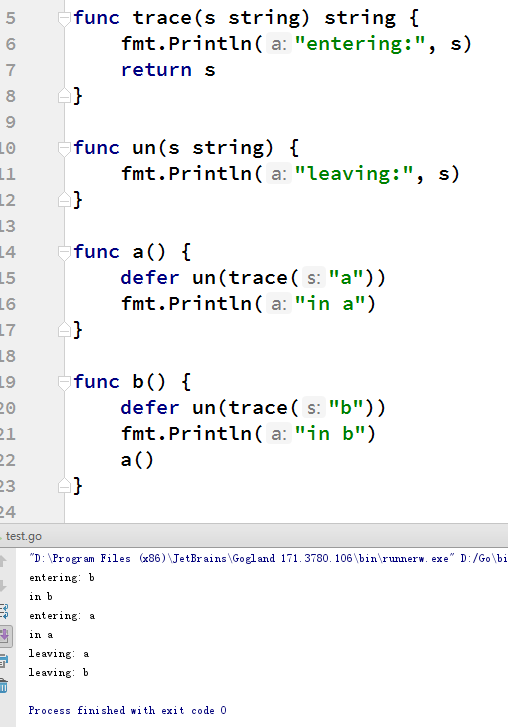




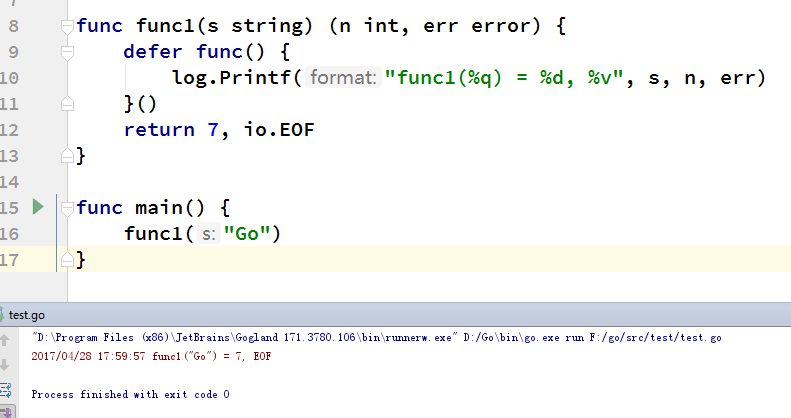
4.传递变长参数



5.使用defer实现代码追踪



6.使用defer语句来记录函数的参数与返回值



## 数组,切片

1. 由于长度也是数组类型的一部分，因此[3]int与[4]int是不同的类型，数组也就不能改变长度。数组之间的赋值是值的赋值，即当把一个数组作为参数传入函数的时候，传入的其实是该数组的副本，而不是它的指针。如果要使用指针，那么就需要用到后面介绍的slice类型了。验证函数传递

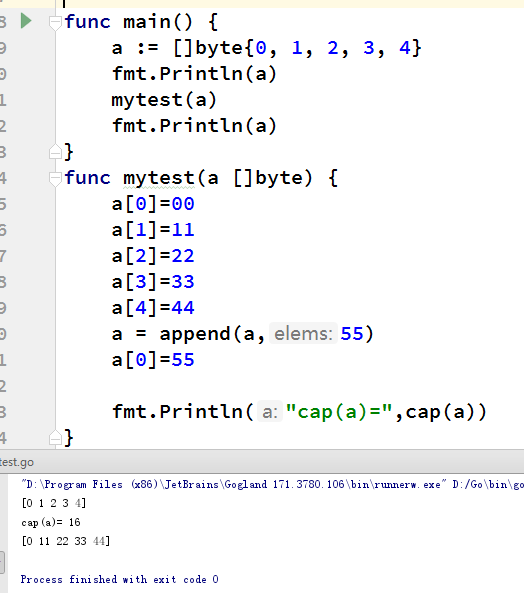
|  |
| --- |
| func main() {     arr := [...]int{1, 2, 3}     fmt.Println("all before:", arr)     test\_value(arr)     fmt.Println("value after:", arr)     test\_point(&arr)     fmt.Println("point after:", arr)     test\_slice(arr[:])     fmt.Println("slice after:", arr) } func test\_value(arr [3]int) {     arr[1] = 111     fmt.Println("test\_value:", arr) } func test\_point(arr \*[3]int) {     arr[1] = 222     fmt.Println("test\_point:", arr) } func test\_slice(arr []int) {     arr[1] = 333     fmt.Println("test\_slice:", arr) } |

2.[]interface{}这种切片是否可以存储不同数据类型

可以,因为该切片存储的类型是interface{}

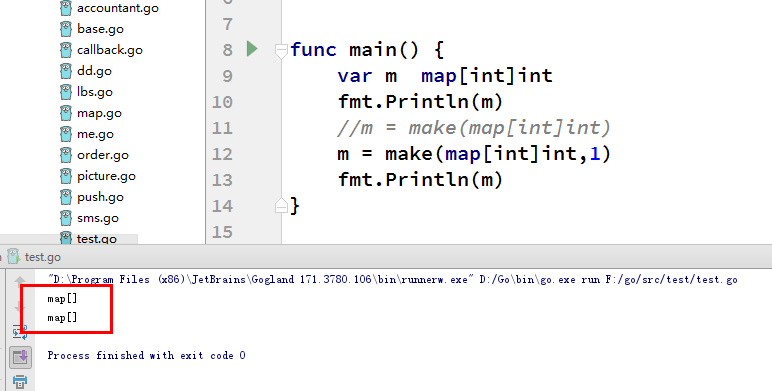
3.通过切片引用传递给函数,在函数中修改此切片后,调用者得到的切片一定是函数中修改后的内容吗?

未必,通过以此达到返回数据给调用者是不安全的,因为如果在此函数中操作切片内容大于原有切片的空间的话,将从新分配空间,此时调用者使用的还是原来数组控件,如下例子:

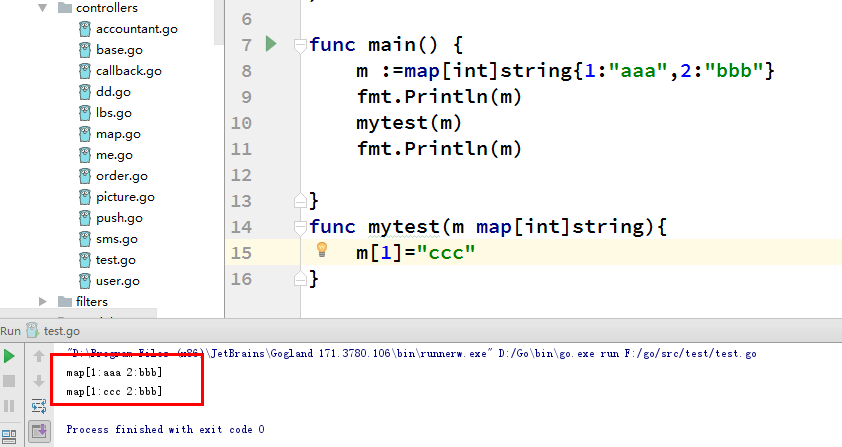


## Map

1. 未初始化的map打印出来是怎样？map[]吗？初始化后的呢？



1. Map是值类型还是引用类型?



## 指针

1.指向数组的指针和指针数组

|  |
| --- |
| func main() {     a := [...]int{9: 1}     var p \*[10]int = &a //p是指向数组的指针     fmt.Println(p)      x, y := 1, 2     b := [...]\*int{&x, &y} //b是指针数组     fmt.Println(b) } |

2. 哪些是值传递,哪些是引用传递?(例如,struct,map,slice,数组,接口,基本类型等) 什么类型的变量不需要(\*s)这样应用可以直接s,哪些些只能\*s ?

* 基本数据,如int,float,string等是值类型,基本类型如果是指针都需要\*s
* 数组是值类型，不管是指向数组本身还是指向数组本身，都可以用a[1]这种方式，指针不需要前面加\*
* 结构体是值类型,结构体无论是值还是指针,都可以直接使用s
* 切片是引用类型,
* Map是引用类型,
* Channel是引用类型

3.函数传递

* 传指针使得多个函数能操作同一个对象。
* 传指针比较轻量级 (8bytes),只是传内存地址，我们可以用指针传递体积大的结构体。如果用参数值传递的话, 在每次copy上面就会花费相对较多的系统开销（内存和时间）。所以当你要传递大的结构体的时候，用指针是一个明智的选择。
* Go语言中channel，slice，map这三种类型的实现机制类似指针，所以可以直接传递，而不用取地址后传递指针。（注：若函数需改变slice的长度，则仍需要取地址传递指针）

## 包

1. 实验多个代码文件共用同一个包，并且有init函数?

当引入包有init函数的时候,优先执行该init函数.并且当同个项目中多次引入同个包时,只执行一次init函数.

2. golang包名必须与所在文件夹同名吗?

不必须，但是同个目录中（不含子目录）的所有文件包名必须一致，通常为了方便包定位，建议包名和目录名一致，否则你import "A"，使用起来B.xxx，看上去不统一，不能一眼看出来这个B包是哪个文件的。

1. src目录下有mytest文件夹，里面有mytest.go包，引用该包的时候为什么是import "mytest" 而不是import "mytest/mytest" ？请上机实验.

import引号后面的只是包所在路径,具体使用的时候是使用该路径下的包名(一般最好和所在文件夹名称一致)来使用. 例如fmt.Println(“hello,world”) 我们调用了fmt包里面定义的函数Println。大家可以看到，这个函数是通过<pkgName>.<funcName>的方式调用的，这一点和Python十分相似。前面提到过，包名和包所在的文件夹名可以是不同的，此处的<pkgName>即为通过package <pkgName>声明的包名，而非文件夹名。

1. GOPATH目录下三个子目录,src,pkg,bin,当我们引入第三方包时,源代码都是在src目录下,如果安装该包,那么编译后的包文件会在pkg子目录下,但是为什么使用该包的时候还是需要源代码存在?而不是只要pkg子目录下的\*.a包文件就可以了?
2. 定义在包中函数内部的大写字母开头的变量，包外可以访问吗？

不行,必须是包中函数外部的公有变量,包外部才可以访问.

6.结构体中变量的首字母大小写，包外部访问性如何？

不可以直接访问;

如果该结构体变量定义在包中,并且该结构体变量名为可访问性质,那么包外可以直接访问整个结构体,但是不能访问结构体内部的变量;

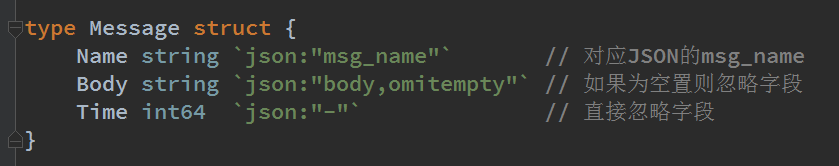
如果如果结构体变量定义在包外部,那么连这个结构体变量都不能定义

7.引入一个包的执行流程?



## 结构体

1.结构体序列化到字符串时,指定某些字段如果为空则不序列话,或者指定某字段不序列化

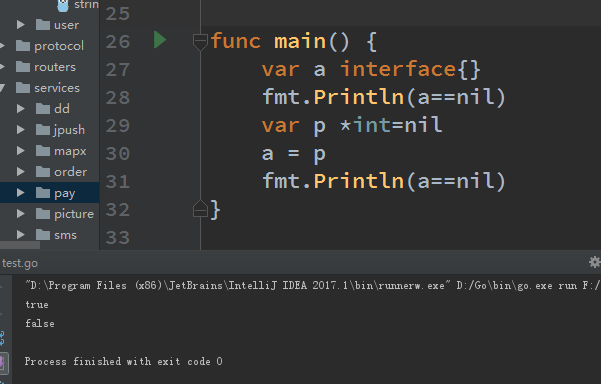


## 接口

1.接口可以匿名嵌入其它接口,或嵌入到结构体中

2.将对象赋值给接口时,会发生拷贝,而接口内部存储的是指向这个复制品的指针,既无法修改原始对象的状态,也无法获取指针

3.只有当接口存储的类型和对象都为nil时,接口才等于nil



4.接口调用不会做receiver的自动转换