Go 语言面试题合集

1 写出下面代码输出内容。

```
package main
import (
"fmt"
)
funcmain() {
   defer_call()
}
funcdefer_call() {
    deferfunc() {fmt.Println("打印前")}()
    deferfunc() {fmt.Println("打印中")}()
    deferfunc() {fmt.Println("打印后")}()
    panic("触发异常")
}
```

考点：defer执行顺序

解答：

defer 是后进先出。

panic 需要等defer 结束后才会向上传递。出现panic恐慌时候，会先按照defer的后入先出的顺序执行，最后才会执行panic。

```
打印后
打印中
打印前
panic: 触发异常
```

2 以下代码有什么问题，说明原因。

```
type student struct {
   Name string
    Age  int
}
funcpase_student() {
    m := make(map[string]*student)
    stus := []student{
        {Name: "zhou",Age: 24},
        {Name: "li",Age: 23},
```

```
        {Name: "wang",Age: 22},
    }    for _,stu := range stus {
        m[stu.Name] =&stu
    }
}
```

考点：foreach

解答：

这样的写法初学者经常会遇到的，很危险！ 与Java的foreach一样，都是使用副本的方式。所以m[stu.Name]=&stu实际上一致指向同一个指针， 最终该指针的值为遍历的最后一个struct的值拷贝。 就像想修改切片元素的属性

：

```
for _, stu := rangestus {
    stu.Age = stu.Age+10}
```

也是不可行的。 大家可以试试打印出来：

```
func pase_student() {
    m := make(map[string]*student)
    stus := []student{
        {Name: "zhou",Age: 24},
        {Name: "li",Age: 23},
        {Name: "wang",Age: 22},
    }
   // 错误写法
   for _,stu := range stus {
        m[stu.Name] =&stu
    }
        fork,v:=range m{
          println(k,"=>",v.Name)
    }
         // 正确
    for i:=0;i<len(stus);i++ {
       m[stus[i].Name] = &stus[i]
    }
         fork,v:=range m{
              println(k,"=>",v.Name)
    }
}
```

3 下面的代码会输出什么，并说明原因

```
func main() {
    runtime.GOMAXPROCS(1)
    wg := sync.WaitGroup{}
    wg.Add(20)    for i := 0; i < 10; i++ {
                gofunc() {
            fmt.Println("A: ", i)
           wg.Done()
        }()
    }
          for i:= 0; i < 10; i++ {
                  gofunc(i int) {
            fmt.Println("B: ", i)
           wg.Done()
        }(i)
    }
   wg.Wait()}
```

考点：go执行的随机性和闭包

解答：

谁也不知道执行后打印的顺序是什么样的，所以只能说是随机数字。 但是A:均为输出10，B:从0~9输出(顺序不定)。

第一个go func中i是外部for的一个变量，地址不变化。遍历完成后，最终i=10。 故go func执行时，i的值始终是10。第二个go func中i是函数参数，与外部for中的i完全是两个变量。 尾部(i)将发生值拷贝，go func内部指向值拷贝地址。

4 下面代码会输出什么?

```
type People struct{}func (p *People)ShowA() {
    fmt.Println("showA")
    p.ShowB()
}
func(p*People)ShowB() {
    fmt.Println("showB")
}
typeTeacher struct {
    People
}
```

```
func(t*Teacher)ShowB() {
   fmt.Println("teachershowB")
}
funcmain() {
    t := Teacher{}
    t.ShowA()
}
```

考点：go的组合继承

解答：

这是Golang的组合模式，可以实现OOP的继承。 被组合的类型People所包含的方法虽然升级成了外部类型Teacher这个组合类型的方法（一定要是匿名字段），但它们的方法(ShowA())调用时接受者并没有发生变化。 此时People类型并不知道自己会被什么类型组合，当然也就无法调用方法时去使用未知的组合者Teacher类型的功能。showAshowB

5 下面代码会触发异常吗？请详细说明

```
func main() {    runtime.GOMAXPROCS(1)
    int_chan := make(chanint, 1)
    string_chan := make(chanstring, 1)
    int_chan <- 1
    string_chan <- "hello"
    select {
            case value := <-int_chan:
      fmt.Println(value)
         casevalue := <-string_chan:
                panic(value)
    }
}
```

考点：select随机性

解答：

select会随机选择一个可用通用做收发操作。 所以代码是有肯触发异常，也有可能不会。单个chan如果无缓冲时，将会阻塞。但结合 select可以在多个chan间等待执行。

有三点原则：select 中只要有一个case能return，则立刻执行。当如果同一时间有多个case均能return则伪随机方式抽取任意一个执行。如果没有一个case能return则可以执行”default”块。

6 下面代码输出什么？

```
funccalc(indexstring, a, bint) int {
```

```
    ret := a+ b
    fmt.Println(index,a, b, ret)
   return ret
}
funcmain() {
        a := 1
    b := 2
   defer calc("1", a,calc("10", a, b))    a = 0
   defer calc("2", a,calc("20", a, b))    b = 1
}
```

考点：defer执行顺序

解答：

这道题类似第1题 需要注意到defer执行顺序和值传递 index:1肯定是最后执行的，但是index:1的第三个参数是一个函数，所以最先被调用calc("10",1,2)==>10,1,2,3 执行index:2时,与之前一样，需要先调用calc("20",0,2)==>20,0,2,2 执行到b=1时候开始调用，index:2==>calc("2",0,2)==>2,0,2,2最后执行index:1==>calc("1",1,3)==>1,1,3,410 1 2 320 0 2 22 0 2 21 1 3 4

7 请写出以下输入内容

```
funcmain() {
          s := make([]int,5)
    s = append(s,1, 2, 3)
   fmt.Println(s)}
```

考点：make默认值和append

解答：

make初始化是由默认值的哦，此处默认值为0

[00000123]

大家试试改为:

```
s := make([]int, 0)
s = append(s, 1, 2, 3)
fmt.Println(s)//[1 2 3]
```

8 下面的代码有什么问题?

```
type UserAges struct {
    ages map[string]int
    sync.Mutex
}
```

```
func(ua*UserAges)Add(name string, age int) {
   ua.Lock()
         deferua.Unlock()
    ua.ages[name] = age
}
func(ua*UserAges)Get(name string)int {
         ifage, ok := ua.ages[name]; ok {
               return age
 }
        return-1
}
```

考点：map线程安全

解答：

可能会出现

fatal error: concurrent mapreadandmapwrite.

修改一下看看效果

```
func (ua *UserAges)Get(namestring)int {
    ua.Lock()
        deferua.Unlock()
         ifage, ok := ua.ages[name]; ok {
               return age
    }
return-1
}
```

9. 下面的迭代会有什么问题?

```
func (set *threadSafeSet)Iter()<-chaninterface{} {
    ch := make(chaninterface{})
              gofunc() {
        set.RLock()
             for elem := range set.s {
          ch <- elem
        }
              close(ch)
      set.RUnlock()
    }()
```

```
    return ch
}
```

考点：chan缓存池

解答：

看到这道题，我也在猜想出题者的意图在哪里。 chan?sync.RWMutex?go?chan缓存池?迭代? 所以只能再读一次题目，就从迭代入手看看。 既然是迭代就会要求set.s全部可以遍历一次。但是chan是为缓存的，那就代表这写入一次就会阻塞。 我们把代码恢复为可以运行的方式，看看效果

```
package main
import (
        "sync"
    "fmt")//下面的迭代会有什么问题？type threadSafeSet struct {
    sync.RWMutex
    s []interface{}
}
func(set*threadSafeSet)Iter() <-chaninterface{} {
    //ch := make(chan interface{}) // 解除注释看看！
    ch := make(chaninterface{},len(set.s))
   gofunc() {
        set.RLock()
     forelem,value := range set.s {
            ch <- elem
            println("Iter:",elem,value)
       }       close(ch)
        set.RUnlock()
    }()
   return ch
}
funcmain() {
    th:=threadSafeSet{
        s:[]interface{}{"1","2"},
    }
    v:=<-th.Iter()
   fmt.Sprintf("%s%v","ch",v)
}
```

10 以下代码能编译过去吗？为什么？

```
package main
import (   "fmt")
typePeople interface {
    Speak(string) string
}
typeStduent struct{}
func(stu*Stduent)Speak(think string)(talk string) {
    ifthink == "bitch" {
        talk = "Youare a good boy"
    } else {
        talk = "hi"
    }
    return
}
funcmain() {
    var peoPeople = Stduent{}
    think := "bitch"
   fmt.Println(peo.Speak(think))
}
```

考点：golang的方法集

解答：

编译不通过！ 做错了！？ 说明你对golang的方法集还有一些疑问。 一句话：golang的方法集仅仅影响接口实现和方法表达式转化，与通过实例或者指针调用方法无关。

11 以下代码打印出来什么内容，说出为什么。

```
package main
import (   "fmt")
typePeople interface {
   Show()
}
typeStudent struct{}
func(stu*Student)Show() {
}
funclive()People {
    var stu*Student
   return stu
```

```
}
funcmain() {   if live() == nil
 {
       fmt.Println("AAAAAAA")
   } else {
      fmt.Println("BBBBBBB")
   }
}
```

考点：interface内部结构

解答：

很经典的题！ 这个考点是很多人忽略的interface内部结构。 go中的接口分为两种一种是空的接口类似这样：

```
varininterface{}
```

另一种如题目：

```
type People interface {
   Show()
}
```

他们的底层结构如下：

```
type eface struct {      //空接口
   _type *_type        //类型信息
   data  unsafe.Pointer //指向数据的指针(go语言中特殊的指针类型unsafe.Pointer类似于c语言中的void*)}
typeiface struct {      //带有方法的接口
   tab  *itab          //存储type信息还有结构实现方法的集合
   data unsafe.Pointer  //指向数据的指针(go语言中特殊的指针类型unsafe.Pointer类似于c语言中的void*)}
type_type struct {
   size       uintptr //类型大小
   ptrdata    uintptr //前缀持有所有指针的内存大小
   hash       uint32  //数据hash值
   tflag      tflag
   align      uint8   //对齐
   fieldalign uint8   //嵌入结构体时的对齐
   kind       uint8   //kind 有些枚举值kind等于0是无效的
```

```
    alg       *typeAlg //函数指针数组，类型实现的所有方法
    gcdata    *byte   str       nameOff
    ptrToThis typeOff
}type itab struct {
    inter  *interfacetype //接口类型
    _type  *_type         //结构类型
    link   *itab
    bad    int32
    inhash int32
    fun    [1]uintptr     //可变大小方法集合}
```

可以看出iface比eface 中间多了一层itab结构。 itab 存储_type信息和[]fun方法集，从上面的结构我们就可得出，因为data指向了nil 并不代表interface 是nil， 所以返回值并不为空，这里的fun(方法集)定义了接口的接收规则，在编译的过程中需要验证是否实现接口 结果：BBBBBBB

12.是否可以编译通过？如果通过，输出什么？

```
func main() {
    i := GetValue() switch i.(type) {
          caseint:
          println("int")
           casestring:
           println("string")
          caseinterface{}:
           println("interface")
           default:
             println("unknown")
    }
}
funcGetValue()int {
return1
}
```

解析考点：type

编译失败，因为type只能使用在interface

13.下面函数有什么问题?

```
func funcMui(x,y int)(sum int,error){
    returnx+y,nil
```

}
解析考点：函数返回值命名
在函数有多个返回值时，只要有一个返回值有指定命名，其他的也必须有命名。 如果返回值有有多个返回值必须加上括号； 如果只有一个返回值并且有命名也需要加上括号； 此处函数第一个返回值有sum名称，第二个未命名，所以错误。

14.是否可以编译通过？如果通过，输出什么？

```
package mainfunc main() {
    println(DeferFunc1(1)) println(DeferFunc2(1)) println(DeferFunc3(1))
}func DeferFunc1(i int)(t int) {
    t = i   deferfunc() {
        t += 3
    }() return t
}
funcDeferFunc2(i int)int {
    t := i  deferfunc() {
        t += 3
    }() return t
}
funcDeferFunc3(i int)(t int) {   deferfunc() {
        t += i
    }() return2}
```

解析考点:defer和函数返回值
需要明确一点是defer需要在函数结束前执行。 函数返回值名字会在函数起始处被初始化为对应类型的零值并且作用域为整个函数 DeferFunc1有函数返回值t作用域为整个函数，在return之前defer会被执行，所以t会被修改，返回4; DeferFunc2函数中t的作用域为函数，返回1;DeferFunc3返回3

15.是否可以编译通过？如果通过，输出什么？

```
funcmain() {    list := new([]int)
    list = append(list,1)
    fmt.Println(list)
}
```

解析考点：new
list:=make([]int,0)

16.是否可以编译通过？如果通过，输出什么？

```
package mainimport "fmt"funcmain() {
```

```
    s1 := []int{1, 2, 3}
    s2 := []int{4, 5}
    s1 = append(s1,s2)
   fmt.Println(s1)}
```

解析考点：append

append切片时候别漏了'...'

17.是否可以编译通过？如果通过，输出什么？

```
func main() {
    sn1 := struct {
       age  int
        name string
    }{age: 11,name: "qq"}
    sn2 := struct {
        age  int
        name string
    }{age: 11,name: "qq"}  if sn1== sn2 {
        fmt.Println("sn1== sn2")
    }
    sm1 := struct {
        age int
       m   map[string]string
    }{age: 11, m:map[string]string{"a": "1"}}
    sm2 := struct {
        age int
        m   map[string]string
    }{age: 11, m:map[string]string{"a": "1"}}
          if sm1 == sm2 {
        fmt.Println("sm1== sm2")
    }
}
```

解析考点:结构体比较

进行结构体比较时候，只有相同类型的结构体才可以比较，结构体是否相同不但与属性类型个数有关，还与属性顺序相关。

```
sn3:= struct {
    name string
```

```
    age  int
}
{age:11,name:"qq"}
```

sn3与sn1就不是相同的结构体了，不能比较。 还有一点需要注意的是结构体是相同的，但是结构体属性中有不可以比较的类型，如map,slice。 如果该结构属性都是可以比较的，那么就可以使用“==”进行比较操作。

可以使用reflect.DeepEqual进行比较

```
if reflect.DeepEqual(sn1, sm) {
    fmt.Println("sn1==sm")
}else {
    fmt.Println("sn1!=sm")
}
```

所以编译不通过： invalid operation: sm1 == sm2

18.是否可以编译通过？如果通过，输出什么？

```
func Foo(x interface{}) {    if x== nil {
        fmt.Println("emptyinterface")
                return
    }
    fmt.Println("non-emptyinterface")
}
       funcmain() {
          var x *int = nil
    Foo(x)
}
```

解析考点：interface内部结构

non-emptyinterface

19.是否可以编译通过？如果通过，输出什么？

```
func GetValue(m map[int]string, id int)(string, bool) {
            if _,exist := m[id]; exist {
                return"存在数据", true
    }
          returnnil, false}funcmain() {
    intmap:=map[int]string{
    1:"a",
       2:"bb",
```

```
    3:"ccc",
  }
  v,err:=GetValue(intmap,3)
  fmt.Println(v,err)
}
```

解析考点：函数返回值类型

nil 可以用作 interface、function、pointer、map、slice 和 channel 的“空值”。但是如果不特别指定的话，Go 语言不能识别类型，所以会报错。报:cannot use nil as type string in return argument.

20.是否可以编译通过？如果通过，输出什么？

```
const (
  x = iota
  y
  z = "zz"
 k
  p = iota)
funcmain()
 {
  fmt.Println(x,y,z,k,p)
}
```

解析考点：iota

结果:0 1 zz zz 4

21.编译执行下面代码会出现什么?

```
package mainvar(
  size :=1024
  max_size = size*2)
funcmain() {
  println(size,max_size)
}
```

解析考点:变量简短模式

变量简短模式限制：定义变量同时显式初始化不能提供数据类型只能在函数内部使用

结果：syntaxerror: unexpected :=

22.下面函数有什么问题?

```
package main
const cl = 100
```

```
var bl    = 123
funcmain() {
    println(&bl,bl)
   println(&cl,cl)
}
```

解析考点:常量

常量不同于变量的在运行期分配内存，常量通常会被编译器在预处理阶段直接展开，作为指令数据使用，

cannot take the address of cl

23.编译执行下面代码会出现什么?

```
package main
funcmain() {
    for i:=0;i<10;i++  {
    loop:
       println(i)
    }    gotoloop
}
```

解析考点：goto

goto不能跳转到其他函数或者内层代码

goto loop jumps intoblock starting at

24.编译执行下面代码会出现什么?

```
package main
import"fmt"
funcmain() {
typeMyInt1 int
 typeMyInt2 = int
    var i int =9
    var i1MyInt1 = i
    var i2MyInt2 = i
    fmt.Println(i1,i2)
}
```

解析考点：**Go 1.9 新特性 Type Alias **

基于一个类型创建一个新类型，称之为defintion；基于一个类型创建一个别名，称之为alias。 MyInt1为称之为defintion，虽然底层类型为int类型，但是不能直接赋值，需要强转； MyInt2称之为alias，可以直接赋值。

结果:cannot use i (typeint) astype MyInt1 in assignment

25.编译执行下面代码会出现什么?

```
package main
import"fmt"
typeUser struct {
}
typeMyUser1 User
typeMyUser2 = User
func(iMyUser1)m1(){
    fmt.Println("MyUser1.m1")
}
func(iUser)m2(){
    fmt.Println("User.m2")
}
funcmain() {
    var i1MyUser1
    var i2MyUser2
    i1.m1()
    i2.m2()
}
```

解析考点：**Go 1.9 新特性 Type Alias **

因为MyUser2完全等价于User，所以具有其所有的方法，并且其中一个新增了方法，另外一个也会有。 但是

i1.m2()

是不能执行的，因为MyUser1没有定义该方法。

结果:MyUser1.m1User.m2

26.编译执行下面代码会出现什么?

```
package main
import"fmt"
type T1 struct {
}
func(tT1)m1(){
    fmt.Println("T1.m1")
}
type T2= T1
```

```
typeMyStruct struct {
    T1
    T2
}
funcmain() {
    my:=MyStruct{}
    my.m1()
}
```

解析考点：**Go 1.9 新特性 Type Alias **

是不能正常编译的,

异常：

ambiguousselectormy.m1

结果不限于方法，字段也也一样；也不限于type alias，type defintion也是一样的，只要有重复的方法、字段，就会有这种提示，因为不知道该选择哪个。

改为:

my.T1.m1()

my.T2.m1()

type alias的定义，本质上是一样的类型，只是起了一个别名，源类型怎么用，别名类型也怎么用，保留源类型的所有方法、字段等。

27.编译执行下面代码会出现什么?

```
package main
import (
        "errors"
    "fmt")
varErrDidNotWork = errors.New("did not work")
funcDoTheThing(reallyDoItbool)(errerror) {
    ifreallyDoIt {
        result, err:= tryTheThing()
        if err!= nil || result != "it worked" {
            err = ErrDidNotWork
        }
    }    return err
}
functryTheThing()(string,error) {
    return"",ErrDidNotWork
```

```
}
funcmain() {
    fmt.Println(DoTheThing(true))
    fmt.Println(DoTheThing(false))
}
```

解析考点：变量作用域

因为 if 语句块内的 err 变量会遮罩函数作用域内的 err 变量，结果：

改为：

```
func DoTheThing(reallyDoIt bool)(errerror) {
    varresult string
    ifreallyDoIt {
        result, err =tryTheThing()
        if err!= nil || result != "it worked" {
            err = ErrDidNotWork
        }
    }    return err
}
```

28.编译执行下面代码会出现什么?

```
package main
functest() []func()
{    varfuns []func()
    fori:=0;i<2;i++  {
        funs = append(funs,func() {
                println(&i,i)
        })
    }    returnfuns
}
funcmain(){
    funs:=test()
        for_,f:=range funs{
        f()
    }
}
```

解析考点：闭包延迟求值

for循环复用局部变量i，每一次放入匿名函数的应用都是想一个变量。

结果：

0xc042046000 2

0xc042046000 2

如果想不一样可以改为：

```
func test() []func()  {
    varfuns []func()
    fori:=0;i<2;i++  {
        x:=i
       funs = append(funs,func() {
          println(&x,x)
        })
    }    returnfuns
}
```

29.编译执行下面代码会出现什么?

```
package main
functest(x int)(func(),func()) {
    returnfunc() {
       println(x)
    x+=10
    }, func() {
            println(x)
    }
}
funcmain() {
    a,b:=test(100)
    a()
    b()
}
```

解析考点：闭包引用相同变量*

结果：

100

110

30. 编译执行下面代码会出现什么?

```
package main
import (   "fmt"
```

```
    "reflect")
funcmain1() {
    deferfunc() {
      iferr:=recover();err!=nil{
          fmt.Println(err)
       }else {
          fmt.Println("fatal")
       }
    }()
    deferfunc() {
       panic("deferpanic")
    }()
    panic("panic")
}
funcmain() {
    deferfunc() {
       iferr:=recover();err!=nil{
          fmt.Println("++++")
          f:=err.(func()string)
fmt.Println(err,f(),reflect.TypeOf(err).Kind().String())
        }else {
           fmt.Println("fatal")
        }
    }()
    deferfunc() {
    panic(func()string {
          return "defer panic"
        })
    }()
    panic("panic")
}
```

解析考点：panic仅有最后一个可以被revover捕获

触发panic("panic")后顺序执行defer，但是defer中还有一个panic，所以覆盖了之前的panic("panic")
