今日总结 2019-03-20

每日一学

问题一：

Go语言中的错误是一种接口类型。接口信息中包含了原始类型和原始的值。只有当接口的类型和原始的值都为空的时候，接口的值才对应nil。其实当接口中类型为空的时候，原始值必然也是空的；反之，当接口对应的原始值为空的时候，接口对应的原始类型并不一定为空的。

```
func myFunc() error {
    var p *MyError = nil
    if fail() {
        p = ErrFail
    }
    return p
}
```

以上代码有什么问题?

讨论结果：

1. 这里即使 fail() 为false， 返回的error也!=nil；
2. return p 不是nil， 因为指针p有类型。如果调用这个函数调用方判断 if err == nil， 这个逻辑不会被执行。

知识点学习

1. recover 必须放在 defer 中才有效，否则永远返回 nil。
2. **实用工具：** GCTT | 【干货】go get 自动代理

常见坑

问题一：

可变参数是空接口类型 当参数的可变参数是空接口类型时，传入空接口的切片时需要注意参数展开的问题。例如：

```
func main() {
    var a = []interface{}{1, 2, 3}
    fmt.Println(a)
    fmt.Println(a...)
}
```

不管是否展开，编译器都无法发现错误，但是输出是不同的。实际中可能会出现“莫名”的情况。

讨论结果：

1. 展开相当于 println(1,2,3)，传入了三个interface，每个interface只装了一个int变量，不展开相当于 println([1,2,3])，传入了一个[]int参数；
2. a...是a中的三个元素1, 2, 3分别传入。

面试题

问题一：

recover 知识点 以下哪些能正常捕获异常，哪些不能?

```
//  1:
func main() {
    if r := recover(); r != nil {
        log.Fatal(r)
    }
    panic(123)
    if r := recover(); r != nil {
        log.Fatal(r)
    }
}
//  2:
func main() {
    defer func() {
        if r := MyRecover(); r != nil {
            fmt.Println(r)
        }
    }()
    panic(1)
}
func MyRecover() interface{} {
    log.Println("trace...")
    return recover()
}
//  3:
func main() {
    defer func() {
        defer func() {
            if r := recover(); r != nil {
                fmt.Println(r)
            }
        }()
    }()
    panic(1)
}
//  4:
func MyRecover() interface{} {
```

```
    return recover()
}
func main() {
    defer MyRecover()
    panic(1)
}
//  5:
func main() {
    defer recover()
    panic(1)
}
//  6:
func main() {
    defer func() {
        if r := recover(); r != nil { ... }
    }()
    panic(nil)
}
```

讨论结果：

1. 还未有可靠结果.....

问题二：

请使用 Go 实现一个函数得到两数相加结果，可用以下两种调用方式： sum(2,3)() 输出5 sum(2)(3)() 输出5 sum(2)(3)(4)() 输出9 请写出你的代码。

讨论结果：

1. demo01：

```
package main

type f func(...int) f

func fsum(i ...int) f {
    var sum int
    var fun f
    fun = func(a ...int) f {
        for _, v := range a {
            sum += v
        }
        if len(a)<=0 {
            fmt.Println(sum)
            return nil
        }
```

```
            return fun
        }

        for _, value := range i {
            sum+=value
        }
        if len(i)>1{
            fmt.Println(sum)
        }
        return fun
    }
    func main() {
        fsum(2)(3)(4)()
        fsum(2,3)()
        fsum(2)(3)()
    }
```