课程安排，请关注**微信公众平台**或者**官方微博**

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，

欢迎来到 **字节教育** 课程的学习

字节教育官网：www.ByteEdu.Com

腾讯课堂地址：Gopher.ke.qq.Com

技术交流群 ： 221 273 219

微信公众号 ：Golang 语言社区

微信服务号 ：Golang 技术社区

目录：

第一季　Golang 语言社区-综合面试题

第一节 defer 的使用

一、公众账号：

回复关键字：客服　　获取课程助教的微信

二、核心内容讲解：

大家都知道 golang 的 defer 关键字，它可以在函数返回前执行一些操作，最常用的就是打开一个资源（例如一个文件、数据库连接等）时就用 defer 延迟关闭改资源，以免引起内存泄漏。本文主要给大家介绍了关于 golang 中 defer 的关键特性，分享出来供大家参考学习，下面话不多说，来一起看看详细的介绍：

一、defer 的作用和执行时机

go 的 defer 语句是用来延迟执行函数的，而且延迟发生在调用函数 return 之后，比如
?

```
1  func a() int {
2    defer b()
3    return 0
4  }
```

b 的执行是发生在 return 0 之后，注意 defer 的语法，关键字 defer 之后是函数的调用。

二、defer 的重要用途一：清理释放资源

由于 defer 的延迟特性，defer 常用在函数调用结束之后清理相关的资源，比如
?

```
1  f, _ := os.Open(filename)
2  defer f.Close()
```

文件资源的释放会在函数调用结束之后借助 defer 自动执行，不需要时刻记住哪里的资源需要释放，打开和释放必须相对应。

用一个例子深刻诠释一下 defer 带来的便利和简洁。

代码的主要目的是打开一个文件，然后复制内容到另一个新的文件中，没有 defer 时这样写：
?

```
1  func CopyFile(dstName, srcName string) (written int64, err error) {
2    src, err := os.Open(srcName)
3    if err != nil {
4      return
5    }
6    dst, err := os.Create(dstName)
7    if err != nil { //1
8      return
9    }
10   written, err = io.Copy(dst, src)
11   dst.Close()
12   src.Close()
13   return
14 }
```

代码在 #1 处返回之后，src 文件没有执行关闭操作，可能会导致资源不能正确释放，改用 defer 实现：
?

```
func CopyFile(dstName, srcName string) (written int64, err error) {
  src, err := os.Open(srcName)
  if err != nil {
    return
  }
  defer src.Close()
  dst, err := os.Create(dstName)
  if err != nil {
    return
  }
  defer dst.Close()
  return io.Copy(dst, src)
}
```

src 和 dst 都能及时清理和释放，无论 return 在什么地方执行。

鉴于 defer 的这种作用，defer 常用来释放数据库连接，文件打开句柄等释放资源的操作。

三、defer 的重要用途二：执行 recover

被 defer 的函数在 return 之后执行，这个时机点正好可以捕获函数抛出的 panic，因而 defer 的另一个重要用途就是执行 recover。

recover 只有在 defer 中使用才更有意义，如果在其他地方使用，由于 program 已经调用结束而提前返回而无法有效捕捉错误。

?

```
package main
import (
  "fmt"
)
func main() {
  defer func() {
    if ok := recover(); ok != nil {
      fmt.Println("recover")
    }
  }()
  panic("error")
}
```

记住 defer 要放在 panic 执行之前。

四、多个 defer 的执行顺序

defer 的作用就是把关键字之后的函数执行压入一个栈中延迟执行，多个 defer 的执行顺序是后进先出 LIFO：
?

```
1  defer func() { fmt.Println("1") }()
2  defer func() { fmt.Println("2") }()
3  defer func() { fmt.Println("3") }()
```

输出顺序是 321。

这个特性可以对一个 array 实现逆序操作。

五、被 deferred 函数的参数在 defer 时确定

这是 defer 的特点，一个函数被 defer 时，它的参数在 defer 时进行计算确定，即使 defer 之后参数发生修改，对已经 defer 的函数没有影响，什么意思？看例子：
?

```
1  func a() {
2    i := 0
3    defer fmt.Println(i)
4    i++
5    return
6  }
```

a 执行输出的是 0 而不是 1，因为 defer 时，i 的值是 0，此时被 defer 的函数参数已经进行执行计算并确定了。

再看一个例子：
?

```
1  func calc(index string, a, b int) int {
2    ret := a + b
3    fmt.Println(index, a, b, ret)
4    return ret
5  }
6  func main() {
7    a := 1
8    b := 2
9    defer calc("1", a, calc("10", a, b))
10   a = 0
11   return
12 }
```

执行代码输出
?

```
10 1 2 3

1 1 3 4
```

defer 函数的参数 第三个参数在 defer 时就已经计算完成并确定，第二个参数 a 也是如此，无论之后 a 变量是否修改都不影响。

六、被 defer 的函数可以读取和修改带名称的返回值

?

```
func c() (i int) {
  defer func() { i++ }()
  return 1
}
```

被 defer 的函数是在 return 之后执行，可以修改带名称的返回值，上面的函数 c 返回的是 2。