函数

defer

defer意思是推迟、延迟。语法很简单，就在正常的语句前加上defer就可以了。

在某函数中使用defer语句，**会使得defer后跟的语句进行延迟处理**，当该函数即将**返回时**，或发生**panic时**，defer后语句开始执行。

同一个函数可以有多个defer语句，依次加入调用栈中（LIFO），函数返回或panic时，从栈顶依次执行defer后语句。执行的先后顺序和注册的顺序正好相反，也就是后注册的先执行。

defer后的语句必须是一个函数或方法的**调用。**

```
package main

import "fmt"

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

请思考上例执行结果是什么?

再看下面的例子，猜猜结果是什么?

```
package main

import "fmt"

func main() {
    count := 1
    fmt.Println("start")
    defer fmt.Println(count)
    count++
    defer fmt.Println(count)
    count++
    defer fmt.Println(count)
    fmt.Println("end")
}
```

结果是3 3 3吗?

结果是3 2 1。为什么？因为defer注册时就，就把其后语句的延迟执行的函数的参数准备好了，也就是**注册时计算。**

再看下面的变化，猜猜结果是什么?

```
package main
```

```
2
3  import "fmt"
4
5  func main() {
6      count := 1
7      fmt.Println("start")
8      defer func() { fmt.Println(count) }() // fmt.Println(count)
9      count++
10     defer fmt.Println(count)
11     count++
12     defer fmt.Println(count)
13     fmt.Println("end")
14 }
```

执行结果是什么？3 2 3。为什么？因为第8行注册时要确定实际参数，而这是个匿名无参函数，没法准备参数。延迟执行时，打印是才要用count，其外部作用域有一个count，目前是3。

那么下例结果又是什么？为什么？

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      count := 1
7      fmt.Println("start")
8      defer func(count int) { fmt.Println(count) }(count) //
   fmt.Println(count)
9      count++
10     defer fmt.Println(count)
11     count++
12     defer fmt.Println(count)
13     fmt.Println("end")
14 }
```