Go 语言延迟调用（defer）

关键字 defer 允许我们推迟到函数返回之前（或任意位置执行 `return` 语句之后）一刻才执行某个语句或函数（为什么要在返回之后才执行这些语句？因为 `return` 语句同样可以包含一些操作，而不是单纯地返回某个值）。

defer特性：

1. 关键字 defer 用于注册延迟调用。
2. 这些调用直到 return 前才被执。因此，可以用来做资源清理。
3. 多个defer语句，按**先进后出**的方式执行。
4. defer语句中的变量，在defer声明时就决定了。

defer用途：

1. 关闭文件句柄
2. 锁资源释放
3. 数据库连接释放

defer 是先进后出，这个很自然,后面的语句会依赖前面的资源，因此如果先前面的资源先释放了，后面的语句就没法执行了。

defer功能强大，对于资源管理非常方便，但是如果没用好，也会有陷阱

示例：

执行顺序：

```
package main

import "fmt"

func main() {
    var whatever = [5]int{1, 2, 3, 4, 5}

    for i := range whatever {
        defer fmt.Println(i)
    }
}
#结果
4
3
2
1
0
```

defer 碰上闭包：

```
package main

import "fmt"
```

```
func main() {
    var whatever = [5]int{1, 2, 3, 4, 5}
    for _, i := range whatever {
        defer func() { fmt.Println(i) }()
    }
}
#结果
5
5
5
5
5
```

函数正常执行,由于闭包用到的变量 i 在执行的时候已经变成5，所以输出全都是5。

关键字 defer 允许我们进行一些函数执行完成后的收尾工作，例如：

- 关闭文件流

```
// open a file
defer file.Close()
```

- 解锁一个加锁的资源

```
mu.Lock()
defer mu.Unlock()
```

- 打印最终报告

```
printHeader()
defer printFooter()
```

- 关闭数据库链接

```
// open a database connection
defer disconnectFromDB()
```

- 使用 defer 语句实现代码追踪

一个基础但十分实用的实现代码执行追踪的方案就是在进入和离开某个函数打印相关的消息，即可以提炼为下面两个函数：

```
func trace(s string) { fmt.Println("entering:", s) }
func untrace(s string) { fmt.Println("leaving:", s) }
```

```
package main

import "fmt"

func trace(s string)   { fmt.Println("entering:", s) }
func untrace(s string) { fmt.Println("leaving:", s) }

func a() {
```

```
    trace("a")
    defer untrace("a")
    fmt.Println("in a")
}

func b() {
    trace("b")
    defer untrace("b")
    fmt.Println("in b")
    a()
}

func main() {
    b()
}
#结果
entering: b
in b
entering: a
in a
leaving: a
leaving: b
```

- 使用 defer 语句来记录函数的参数与返回值

```
package main

import (
    "io"
    "log"
)

func func1(s string) (n int, err error) {
    defer func() {
        log.Printf("func1(%q) = %d, %v", s, n, err)
    }()
    return 7, io.EOF
}

func main() {
    func1("Go")
}
#结果
Output: 2011/10/04 10:46:11 func1("Go") = 7, EOF
```

合理使用 defer 语句能够使得代码更加简洁。