闭包

闭包

```go
package main

import "fmt"

// 函数内部没有办法声明带有名字的函数
//匿名函数
func main() {
    f1 := func(x, y int) {
        fmt.Println(x + y)
    }
    f1(10, 20)

    //如果知识执行一次的函数还可以写成立即执行函数
    //立即执行函数
    func(x, y int) {
        fmt.Println("Hello World")
        fmt.Println(x + y)
    }(12, 34)
}
```

闭包

```go
package main

import "fmt"

func f1(f func()) {
    f()
}

func f2(x, y int) {
    fmt.Println("This is f2")
    fmt.Println(x+y)
}

```

这种情况下，由于参数类型不同，f2无法作为参数传入f1。这时候就要用到闭包

使用闭包后的代码 Go

```go
package main

import "fmt"

func f1(f func()) {
    f()
}

func f2(x, y int) {
    fmt.Println("This is f2")
    fmt.Println(x + y)
}

func f3(f func(int, int), m, n int) {
    tmp := func() {
        f(m, n)
    }
    tmp()
}
func main() {
    f3(f2, 100, 300)
}
```

```go
package main

import "fmt"

func adder() func(int) int {
    var x = 100
    return func(y int) int {
        x += y
        return x
    }
}

func main() {
    ret := adder()
    ret2 := ret(200)
    fmt.Println(ret2)
    fmt.Printf("%T %T\n", ret, ret2)
}
```

用ret接受adder返回的变量（是个函数），然后给这个函数传参用ret2接受（返回值是整形）

```
<4 go 设置调用>
300
func(int) int int

进程 已完成，退出代码为 0
```

闭包的底层原理

1.函数可以作为返回值

2.函数内部查找变量的顺序，现在自己内部找，找不到往外层找。

闭包=函数+外部变量的引用