|  |
| --- |
| <http://blog.51cto.com/speakingbaicai/1703229> |
|  |

所谓闭包就是一个函数“捕获”了和它在同一作用域的其他常量和变量。从形式上看，在Golang中，所有的匿名函数都是闭包。闭包的创建方式和普通函数几乎一致，只有一个关键区别：闭包没有名字。

实质含义：闭包是由函数及相关引用环境组合而成的实体（即：闭包 = 函数+引用环境）。闭包在运行时可以有多个实例。不同的引用环境和相同的函数组合可以产生不同的实例。

可以推论：闭包获取捕获变量相当于引用传递，而非值传递；对于闭包函数捕获的常量和变量，无论闭包何时何处被调用，闭包都可以使用这些常量和变量，而不用关心它们表面上的作用域。

|  |
| --- |
| funcaddNumber(x int) func(int) {      fmt.Printf("x: %d, addr of x:%p\n", x, &x)      return func(y int) {          k := x + y          x = k          y = k          fmt.Printf("x: %d, addr of x:%p\n", x, &x)          fmt.Printf("y: %d, addr of y:%p\n", y, &y)      }  }    func main() {      addNum := addNumber(5)      addNum(1)      addNum(1)      addNum(1)        fmt.Println("---------------------")        addNum1 := addNumber(5)      addNum1(1)      addNum1(1)      addNum1(1)  } |
| wKioL1YfWpyhwQw6AAGAfOsAGpQ888.jpg |

即x是闭包中被捕获的变量，y只是闭包内部的局部变量，而非被捕获的变量。所以对于每一次的引用，x地址都是固定的，是同一个引用变量；y的地址则是变化的。

其次，闭包被引用两次，由此产生了两个闭包实例，即 addNum := addNumber(5) 和

AddNum1 := addNumber(5)是两个不同实例，其中引用的两个x变量也来自两个不同的实例。