Go 语言函数

函数是基本的代码块，用于执行一个任务。我们把所有的功能单元都定义在函数中，可以重复使用。

Go 语言最少有个 main() 函数。

你可以通过函数来划分不同功能，逻辑上每个函数执行的是指定的任务。

函数声明告诉了编译器函数的名称，返回类型，和参数。

Go 语言标准库提供了多种可动用的内置的函数。例如，len() 函数可以接受不同类型参数并返回该类型的长度。如果我们传入的是字符串则返回字符串的长度，如果传入的是数组，则返回数组中包含的元素个数。

函数的主要目的是将一个需要很多行代码的复杂问题分解为一系列简单的任务（那就是函数）来解决。而且，同一个任务（函数）可以被调用多次，有助于代码重用。

Go 里面有三种类型的函数：

- 普通的带有名字的函数
- 匿名函数或者lambda函数
- 方法（Methods）

除了main()、init()函数外，其它所有类型的函数都可以有参数与返回值。函数参数、返回值以及它们的类型被统称为函数签名。

函数特点

1. go语言中有3种函数：普通函数、匿名函数(没有名称的函数)、方法(定义在struct上的函数)。
2. go语言中不允许函数重载(overload)，也就是说不允许函数同名。
3. go语言中的函数不能嵌套函数，但可以嵌套匿名函数。
4. 函数也是一种类型，一个函数可以赋值给变量，使变量也变成函数。
5. 函数可以作为一个参数传递给另一个函数。
6. 函数的返回值可以是一个函数。
7. 函数的参数可以没有名称。
8. 函数在调用的时候，如果有参数传递给函数，则先拷贝参数的副本，再将副本传递给参数。传参是个拷贝的过程
9. 函数不支持默认参数 (default parameter)。

函数声明

格式如下：

```
func function_name( [parameter list] ) [return_types] {
   函数体
}
```

- func：函数由 func 开始声明
- function_name：函数名称，参数列表和返回值类型构成了函数签名。
- parameter list：参数列表，参数就像一个占位符，当函数被调用时，你可以将值传递给参数，这个值被称为实际参数。参数列表指定的是参数类型、顺序、及参数个数。参数是可选的，也就是说函数也可以不包含参数。

- return_types：返回类型，函数返回一列值。return_types 是该列值的数据类型。有些功能不需要返回值，这种情况下 return_types 不是必须的。
- 函数体：函数定义的代码集合。

示例：求和

```
func sum(x int, y int) (ret int) {
    n := x + y
    return n
}
```

函数声明包含一个函数名，参数列表， 返回值列表和函数体。如果函数没有返回值，则返回列表可以省略。函数从第一条语句开始执行，直到执行return语句或者执行函数的最后一条语句。

函数可以没有参数或接受多个参数。

注意类型在变量名之后 。

当两个或多个连续的函数命名参数是同一类型，则除了最后一个类型之外，其他都可以省略。

函数可以返回任意数量的返回值。

使用关键字 func 定义函数，左大括号依旧不能另起一行。

```
func test(x, y int, s string) (int, string) {
    // 类型相同的相邻参数，参数类型可合并。 多返回值必须用括号。
    n := x + y
    return n, fmt.Sprintf(s, n)
}
```

> 有返回值的函数，必须有明确的终止语句，否则会引发编译错误。

函数调用

当创建函数时，你定义了函数需要做什么，通过调用该函数来执行指定任务。

调用函数，向函数传递参数，并返回值。

示例：

```
package main

import "fmt"

func main() {
   /* 定义局部变量 */
   var a int = 100
   var b int = 200
   var ret int

   /* 调用函数并返回最大值 */
   ret = max(a, b)

   fmt.Printf( "最大值是 : %d\n", ret )
}
```

```
/* 函数返回两个数的最大值 */
func max(num1, num2 int) int {
   /* 定义局部变量 */
   var result int

   if (num1 > num2) {
      result = num1
   } else {
      result = num2
   }
   return result
}
#结果
最大值是 : 200
```

函数被调用的时候，这些**实参将被复制**（简单而言）然后**传递给被调用函数。函数一般是在其他函数里面被调用的，这个其他函数被称为调用函数（calling function）**。函数能多次调用其他函数，这些被调用函数按顺序（简单而言）执行，理论上，函数调用其他函数的次数是无穷的（直到函数调用栈被耗尽）。

当函数执行到代码块最后一行（`}` 之前）或者 `return` 语句的时候会退出，其中 `return` 语句可以带有零个或多个参数；这些参数将作为返回值供调用者使用。简单的 `return` **语句也可以用来结束 for 死循环，或者结束一个协程（goroutine）**。

函数可以将其他函数调用作为它的参数，只要这个被调用函数的返回值个数、返回值类型和返回值的顺序与调用函数所需求的实参是一致的，例如：

假设 f1 需要 3 个参数 `f1(a, b, c int)`，同时 f2 返回 3 个参数 `f2(a, b int) (int, int, int)`，就可以这样调用 f1：`f1(f2(a, b))`。

函数重载（function overloading）指的是可以编写多个同名函数，只要它们拥有不同的形参与/或者不同的返回值，在 **Go 里面函数重载是不被允许的。**这将导致一个编译错误。Go 语言不支持这项特性的主要原因是函数重载需要进行多余的类型匹配影响性能；没有重载意味着只是一个简单的函数调度。所以你需要给不同的函数使用不同的名字，我们通常会根据函数的特征对函数进行命名。

如果需要申明一个在外部定义的函数，你只需要给出函数名与函数签名，不需要给出函数体：

```
func flushICache(begin, end uintptr) // implemented externally
```

函数类型

函数也可以以申明的方式被使用，作为一个函数类型，就像：

```
type binOp func(int, int) int
```

在这里，不需要函数体 `{}`。

函数是一等值（first-class value）：它们可以赋值给变量，就像 `add := binOp` 一样。

这个变量知道自己指向的函数的签名，所以给它赋一个具有不同签名的函数值是不可能的。

函数值（functions value）之间可以相互比较：如果它们引用的是相同的函数或者都是 nil 的话，则认为它们是相同的函数。函数不能在其它函数里面声明（不能嵌套），不过我们可以通过使用匿名函数来破除这个限制。

目前 Go 没有泛型（generic）的概念，也就是说它不支持那种支持多种类型的函数。不过在大部分情况下可以通过接口（interface），特别是空接口与类型选择（type switch）与/或者通过使用反射（reflection）来实现相似的功能。使用这些技术将导致代码更为复杂、性能更为低下，所以在非常注意性能的的场合，最好是为每一个类型单独创建一个函数，而且代码可读性更强。

示例：

```
package main

import "fmt"

type sum func(int, int) int

func add(a, b int) (c int) {
    c = a + b
    return c
}

func main() {
    var f sum
    f = add
    s := f(1, 2)
    fmt.Printf("s: %v\n", s)
}
#结果
s: 3
```