# 函数参数

Go语言函数可以有0或多个参数，参数需要指定**数据类型。**

声明函数时的参数列表叫做形参，调用时传递的参数叫做实参。形参就像定义在函数体内的局部变量。

Go语言是通过**传值的方式传参**的，意味着传递给函数的是拷贝后的副本，所以函数内部访问、修改的也是这个副本。

Go语言可以使用**变长参数**，有时候并不能确定参数的个数，可以使用变长参数，可以在函数定义语句的参数部分使用 `ARGS...TYPE` 的方式。这时会将 `...` 代表的参数全部保存到一个名为ARGS的slice中，注意这些参数的数据类型都是TYPE。

函数定义时，它的形参一般是有名字的，不过我们也可以定义没有形参名的函数，只有相应的形参类型，就像这样：`func f(int, int, float64)`。

没有参数的函数通常被称为 **niladic** 函数（niladic function），就像 `main.main()`。

# 按值传递（call by value)

Go 默认使用按值传递来传递参数，也就是传递参数的副本。函数接收参数副本之后，在使用变量的过程中可能对副本的值进行更改，但不会影响到原来的变量，比如 `Function(arg1)`。

**值传递：指在调用函数时将实际参数复制一份传递到函数中，这样在函数中如果对参数进行修改，将不会影响到实际参数。**

示例：

```
package main

import "fmt"

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

   fmt.Printf("交换前 a 的值为 : %d\n", a )
   fmt.Printf("交换前 b 的值为 : %d\n", b )

   /* 通过调用函数来交换值 */
   swap(a, b)

   fmt.Printf("交换后 a 的值 : %d\n", a )
   fmt.Printf("交换后 b 的值 : %d\n", b )
}

/* 定义相互交换值的函数 */
func swap(x, y int) int {
   var temp int

   temp = x /* 保存 x 的值 */
   x = y    /* 将 y 值赋给 x */
```

```
   y = temp /* 将 temp 值赋给 y*/

   return temp;
}
#结果
交换前 a 的值为 : 100
交换前 b 的值为 : 200
交换后 a 的值 : 100
交换后 b 的值 : 200
```

# 按引用传递（call by reference）

如果你希望函数可以直接修改参数的值，而不是对参数的副本进行操作，你需要将参数的地址（变量名前面添加&符号，比如 &variable）传递给函数，这就是按引用传递，比如 `Function(&arg1)`，此时传递给函数的是一个指针。如果传递给函数的是一个指针，指针的值（一个地址）会被复制，但指针的值所指向的地址上的值不会被复制；我们可以通过这个指针的值来修改这个值所指向的地址上的值（**指针也是变量类型，有自己的地址和值，通常指针的值指向一个变量的地址。所以，按引用传递也是按值传递**）。

**引用传递：是指在调用函数时将实际参数的地址传递到函数中，那么在函数中对参数所进行的修改，将影响到实际参数。**

示例:

```
package main

import "fmt"

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

   fmt.Printf("交换前，a 的值 : %d\n", a )
   fmt.Printf("交换前，b 的值 : %d\n", b )

   /* 调用 swap() 函数
   * &a 指向 a 指针，a 变量的地址
   * &b 指向 b 指针，b 变量的地址
   */
   swap(&a, &b)

   fmt.Printf("交换后，a 的值 : %d\n", a )
   fmt.Printf("交换后，b 的值 : %d\n", b )
}

func swap(x *int, y *int) {
   var temp int
   temp = *x    /* 保存 x 地址上的值 */
   *x = *y      /* 将 y 值赋给 x */
   *y = temp    /* 将 temp 值赋给 y */
}
#结果
交换前，a 的值 : 100
交换前，b 的值 : 200
```

```
交换后，a 的值 : 200
交换后，b 的值 : 100
```

> 注意1：无论是值传递，还是引用传递，传递给函数的都是变量的副本，不过，值传递是值的拷贝。引用传递是地址的拷贝，一般来说，地址拷贝更为高效。而值拷贝取决于拷贝的对象大小，对象越大，则性能越低。

> 注意2：切片（slice）、字典（map）、接口（interface）、通道（channel）、指针默认以引用的方式传递。

# 变长参数

不定参数传值 就是函数的参数不是固定的，后面的类型是固定的。（可变参数）

Golang 可变参数本质上就是 slice。只能有一个，且必须是最后一个。

在参数赋值时可以不用用一个一个的赋值，可以直接传递一个数组或者切片，特别注意的是在参数后加上“...”即可。

```
func myfunc(args ...int) {    //0个或多个参数
}

func add(a int, args…int) int {    //1个或多个参数
}

func add(a int, b int, args…int) int {    //2个或多个参数
}
```

注意：其中args是一个slice，我们可以通过arg[index]依次访问所有参数,通过len(arg)来判断传递参数的个数。

任意类型的不定参数： 就是函数的参数和每个参数的类型都不是固定的。

用interface{}传递任意类型数据是Go语言的惯例用法，而且interface{}是类型安全的。

```
func myfunc(args ...interface{}) {
}
```

示例：

```
package main

import (
    "fmt"
)

func test(s string, n ...int) string {
    var x int
    for _, i := range n {
        x += i
    }

    return fmt.Sprintf(s, x)
}
```

```
func main() {
    println(test("sum: %d", 1, 2, 3))
}
#结果
sum: 6
```

使用 slice 对象做变参时，必须展开。`（slice...）`

示例：

```
package main

import (
    "fmt"
)

func test(s string, n ...int) string {
    var x int
    for _, i := range n {
        x += i
    }

    return fmt.Sprintf(s, x)
}

func main() {
    s := []int{1, 2, 3}
    res := test("sum: %d", s...) // slice... 展开slice
    println(res)
}
#结果
sum: 6
```