分类

类型	名称	长度	默认值	说明
pointer	指针		nil	
array	数组		0	
slice	切片		nil	引用类型
map	字典		nil	引用类型
struct	结构体			

指针

指针是一个代表着某个内存地址的值。这个内存地址往往是在内存中存储的另一个变量的值的起始位置。Go语言对指针的支持介于Java语言和C/C++语言之间，它既没有想Java语言那样取消了代码对指针的直接操作的能力，也避免了C/C++语言中由于对指针的滥用而造成的安全和可靠性问题。

基本操作

Go语言虽然保留了指针，但与其它编程语言不同的是：

- 默认值 nil，没有 NULL 常量
- 操作符 "&" 取变量地址， "*" 通过指针访问目标对象
- 不支持指针运算，不支持 "->" 运算符，直接用 "." 访问目标成员

```
func main() {
    var a int = 10                    //声明一个变量，同时
```

```
初始化
    fmt.Printf("&a = %p\n", &a) //操作符 "&" 取变量
地址

    var p *int = nil //声明一个变量p, 类型为 *int, 指
针类型
    p = &a
    fmt.Printf("p = %p\n", p)
    fmt.Printf("a = %d, *p = %d\n", a, *p)

    *p = 111 //*p操作指针所指向的内存，即为a
    fmt.Printf("a = %d, *p = %d\n", a, *p)
}
```

new 函数

表达式new(T)将创建一个T类型的匿名变量，所做的是为T类型的新值分配并清零一块内存空间，然后将这块内存空间的地址作为结果返回，而这个结果就是指向这个新的T类型值的指针值，返回的指针类型为`*T`。

```
func main() {
    var p1 *int
    p1 = new(int)                  //p1为*int 类型, 指向
匿名的int变量
    fmt.Println("*p1 = ", *p1) //*p1 =  0

    p2 := new(int) //p2为*int 类型, 指向匿名的int变量
```

```
    *p2 = 111
    fmt.Println("*p2 = ", *p2) //*p1 =  111
}
```

我们只需使用new()函数，无需担心其内存的生命周期或怎样将其删除，因为Go语言的内存管理系统会帮我们打理一切。

指针做函数参数

```
func swap01(a, b int) {
    a, b = b, a
    fmt.Printf("swap01 a = %d, b = %d\n", a, b)
}

func swap02(x, y *int) {
    *x, *y = *y, *x
}

func main() {
    a := 10
    b := 20

    //swap01(a, b) //值传递
    swap02(&a, &b) //变量地址传递
    fmt.Printf("a = %d, b = %d\n", a, b)
}
```

数组

概述

数组是指一系列同一类型数据的集合。数组中包含的每个数据被称为数组元素（element），一个数组包含的元素个数被称为数组的长度。

数组长度必须是常量，且是类型的组成部分。 [2]int 和 [3]int 是不同类型。

```
    var n int = 10
    var a [n]int  //err, non-constant array bound n
    var b [10]int //ok
```

操作数组

数组的每个元素可以通过索引下标来访问，索引下标的范围是从0开始到数组长度减1的位置。

```
    var a [10]int
    for i := 0; i < 10; i++ {
        a[i] = i + 1
        fmt.Printf("a[%d] = %d\n", i, a[i])
    }

    //range具有两个返回值，第一个返回值是元素的数组下标，第二个返回值是元素的值
```

```
    for i, v := range a {
        fmt.Println("a[", i, "]=", v)
    }
```

内置函数 len(长度) 和 cap(容量) 都返回数组长度 (元素数量):

```
a := [10]int{}
    fmt.Println(len(a), cap(a))//10 10
```

初始化:

```
a := [3]int{1, 2}           // 未初始化元素值为 0
    b := [...]int{1, 2, 3}      // 通过初始化值确定数组长度
    c := [5]int{2: 100, 4: 200} // 通过索引号初始化元素，未初始化元素值为 0
    fmt.Println(a, b, c)        //[1 2 0] [1 2 3] [0 0 100 0 200]

    //支持多维数组
    d := [4][2]int{{10, 11}, {20, 21}, {30, 31}, {40, 41}}
    e := [...][2]int{{10, 11}, {20, 21}, {30, 31}, {40, 41}} //第二维不能写"..."
    f := [4][2]int{1: {20, 21}, 3: {40, 41}}
    g := [4][2]int{1: {0: 20}, 3: {1: 41}}
```

```
    fmt.Println(d, e, f, g)
```

相同类型的数组之间可以使用 == 或 != 进行比较，但不可以使用 < 或 >，也可以相互赋值：

```
a := [3]int{1, 2, 3}
    b := [3]int{1, 2, 3}
    c := [3]int{1, 2}
    fmt.Println(a == b, b == c) //true false

    var d [3]int
    d = a
    fmt.Println(d) //[1 2 3]
```

在函数间传递数组

根据内存和性能来看，在函数间传递数组是一个开销很大的操作。在函数之间传递变量时，总是以值的方式传递的。如果这个变量是一个数组，意味着整个数组，不管有多长，都会完整复制，并传递给函数。

```
func modify(array [5]int) {
    array[0] = 10 // 试图修改数组的第一个元素
    //In modify(), array values: [10 2 3 4 5]
    fmt.Println("In modify(), array values:",
array)
}
```

```
func main() {
    array := [5]int{1, 2, 3, 4, 5} // 定义并初始化一个数组
    modify(array)                  // 传递给一个函数，并试图在函数体内修改这个数组内容
    //In main(), array values: [1 2 3 4 5]
    fmt.Println("In main(), array values:", array)
}
```

数组指针做函数参数：

```
func modify(array *[5]int) {
    (*array)[0] = 10
    //In modify(), array values: [10 2 3 4 5]
    fmt.Println("In modify(), array values:", *array)
}

func main() {
    array := [5]int{1, 2, 3, 4, 5} // 定义并初始化一个数组
    modify(&array)                 // 数组指针
    //In main(), array values: [10 2 3 4 5]
    fmt.Println("In main(), array values:", array)
}
```

难点代码示例：

```
package main

type user struct {
    name string
}

type newArray struct {
    userP *user
    len   int
    cap   int
}

func modifyArray(arr [2]newArray) {
    //打印数组元素的地址
    println("arr[0] address:", &arr[0])
    println("arr[1] address:", &arr[1])

    // 尝试修改数组元素的值
    arr[0].len = 999
    arr[0].cap = 888
    temp0 := arr[0]
    temp0.userP.name = "ninini"
    arr[1].len = 777
    arr[1].cap = 666
    temp1 := arr[1]
    temp1.userP.name = "mamama"
```

```
}
func main() {
    // 1. 声明结构体数组（长度为2，元素类型是newArray）
    var arr [2]newArray
    // 给数组元素赋值
    tempUser := user{name: "Alice"}
    tempUser2 := user{name: "Bob"}
    arr[0] = newArray{userP: &tempUser, len: 10,
cap: 20}
    arr[1] = newArray{userP: &tempUser2, len: 5,
cap: 10}

    // 打印数组元素的地址
    println("arr[0] SOURCE address:", &arr[0])
    println("arr[1] SOURCE address:", &arr[1])
    // 调用修改函数
    modifyArray(arr)
    // 打印数组元素，观察是否被修改
    println("arr[0]: len =", arr[0].len, ", cap
=", arr[0].cap, ", userP.name =",
arr[0].userP.name) // 10, 20 user.name = ninini
    println("arr[1]: len =", arr[1].len, ", cap
=", arr[1].cap, ", userP.name =",
arr[1].userP.name) // 5, 10 user.name = mamama
}
```