# **Go 语言复合类型**

## **实验简介**

这一节讲解Go语言的复合类型，包括数组、切片和映射等。

## **一、实验说明**

本课程所有源代码，可以在XfceTerminal中通过以下方式克隆到实验环境:

$ git clone http://git.shiyanlou.com/shiyanlou/Golang\_Programming

## **二. 值、指针和引用类型**

通常情况下Go语言中的变量持有相应的值。也就是说，我们可以将一个变量想象成它所持有的值来使用。其中有些例外，通道、函数、方法、映射、切片是引用变量，它们持有的都是引用，也即保存指针的变量。值在传递给函数或者方法的时候会被复制一次，对于布尔类型和数值类型来说这非常廉价，但是对于大型变量代价却非常大。而且复制传参的方式，修改值只是修改了副本，这能保证原始变量不被修改，但也一定程度上增加了修改原始值的麻烦。幸好在Go语言中有指针，使用指针时，我们每次传递给函数或者方法的只是变量的内存地址，这是非常廉价的。而且一个被指针指向的变量可以通过该指针来修改，这就很方便的在函数或者防止中通过指针修改原始变量。Go语言中的指针操作符也是使用&和\*操作符，其中&用于取地址，\*用于解引用，也就是获取指针指向的值。

使用VIM创建源文件pointer.go，输入以下源文件：

package main

import (

"fmt"

)

func swap1(x, y, p \*int) {

if \*x > \*y {

\*x, \*y = \*y, \*x

}

\*p = \*x \* \*y

}

func swap2(x, y int) (int, int, int) {

if x > y {

x, y = y, x

}

return x, y, x \* y

}

func main() {

i := 9

j := 5

product := 0

swap1(&i, &j, &product)

fmt.Println(i, j, product)

a := 64

b := 23

a, b, p := swap2(a, b)

fmt.Println(a, b, p)

}

以上源码中，我们首先创建了swap1函数，其通过指针原地的交换值，同时swap2函数通过复制的方式交换了变量的值。

运行结果如下：

$ go run pointer.go5 9 4523 64 1472

## **三. 数组和切片**

### **1. 数组**

Go语言的数组是一个定长的序列，其中的元素类型相同。多维数组可以简单地使用自身为数组的元素来创建。数组的元素使用操作符号[ ]来索引，索引从0开始，到len(array)-1结束。数组使用以下语法创建：

* [length]Type
* [N]Type{value1, value2, ..., valueN}
* [...]Type{value1, value2, ..., valueN}

如果使用了...（省略符）操作符，Go语言会为我们自动计算数组的长度。在任何情况下，一个数组的长度都是固定的并且不可修改。数组的长度可以使用len()函数获得。由于数组的长度是固定的，因此数组的长度和容量都是一样的，因此对于数组而言cap()和len()函数返回值都是一样的。数组也可以使用和切片一样的语法进行切片，只是其结果为一个切片，而非数组。同样的，数组也可以使用range进行索引访问。

### **2. 切片**

一般而言，Go语言的切片比数组更加灵活，强大而且方便。数组是按值传递的（即是传递的副本），而切片是引用类型，传递切片的成本非常小，而且是定长的。而且数组是定长的，而切片可以调整长度。创建切片的语法如下：

* make([ ]Type, length, capacity)
* make([ ]Type, length)
* [ ]Type{}
* [ ]Type{value1, value2, ..., valueN}

内置函数make()用于创建切片、映射和通道。当用于创建一个切片时，它会创建一个隐藏的初始化为零值的数组，然后返回一个引用该隐藏数组的切片。该隐藏的数组与Go语言中的所有数组一样，都是固定长度，如果使用第一种语法创建，那么其长度为切片的容量capacity；如果是第二种语法，那么其长度记为切片的长度length。一个切片的容量即为隐藏数组的长度，而其长度则为不超过该容量的任意值。另外可以通过内置的函数append()来增加切片的容量。切片可以支持以下操作：

我们练习下，使用VIM创建源文件slice\_array.go，输入以下代码：

package main

import (

"fmt"

)

func main() {

a := [...]int{1, 2, 3, 4, 5, 6, 7}

fmt.Printf("len and cap of array %v is: %d and %d\n", a, len(a), cap(a))

fmt.Printf("item in array: %v is:", a)

for \_, value := range a {

fmt.Printf("% d", value)

}

fmt.Println()

s1 := a[3:6]

fmt.Printf("len and cap of slice: %v is: %d and %d\n", s1, len(s1), cap(s1))

fmt.Printf("item in slice: %v is:", s1)

for \_, value := range s1 {

fmt.Printf("% d", value)

}

fmt.Println()

s1[0] = 456

fmt.Printf("item in array changed after changing slice: %v is:", s1)

for \_, value := range a {

fmt.Printf("% d", value)

}

fmt.Println()

s2 := make([]int, 10, 20)

s2[4] = 5

fmt.Printf("len and cap of slice: %v is: %d and %d\n", s2, len(s2), cap(s2))

fmt.Printf("item in slice %v is:", s2)

for \_, value := range s2 {

fmt.Printf("% d", value)

}

fmt.Println()

}

以上代码中，我们首先创建了一个数组，数组的长度是由Go语言自动计算出的（省略号语法），然后通过切片操作从数组a中创建了切片s1，接着我们修改了该切片的第一个位置的数值，然后发现数组a中的值也发生了变化。最后我们通过make()函数创建了一个切片，该切片的长度和容量分别为10和20，还可以发现Go语言将未初始化的项自动赋予零值。运行代码输出如下：

$ go run slice\_array.go

len and cap of array [1 2 3 4 5 6 7] is: 7 and 7

item in array: [1 2 3 4 5 6 7] is: 1 2 3 4 5 6 7

len and cap of slice: [4 5 6] is: 3 and 4

item in slice: [4 5 6] is: 4 5 6

item in array changed after changing slice: [456 5 6] is: 1 2 3 456 5 6 7

len and cap of slice: [0 0 0 0 5 0 0 0 0 0] is: 10 and 20

item in slice [0 0 0 0 5 0 0 0 0 0] is: 0 0 0 0 5 0 0 0 0 0

## **四. 映射（map）**

Go语言中的映射（map）是一种内置的数据结构，保存键=值对的无序集合，它的容量只受到机器内存的限制，类似于Python中的字典。在一个映射中所有的键都是唯一的而且必须是支持==和!=操作符的类型，大部分Go语言的基本类型都可以作为映射的键，但是切片、不能用于比较的数组、结构体（这些类型的成员或者字段不支持==和!=操作）或者基于这些的自定义类型不能作为键。但是任意类型都可以作为值。映射是引用类型，所以传递非常廉价。

Go语言中的映射可以用以下用法创建：

* make(map[KeyType]VauleType, initialCapacity)
* make(map[KeyType]ValueType)
* map[KeyType]ValueType{ }
* map[KeyType]ValueType{key1: value1, key2: value2, ..., keyN: valueN}

内置的函数make()可以用来创建切片、映射和channel（通道）。当用make()来创建一个映射时候，实际上是得到一个空映射，如果指定了容量(initialCapacity)就会预先申请足够的内存，并随着加入的项越来越多，映射会字段扩容。映射支持以下操作：

| **语法** | **含义** |
| --- | --- |
| m[k] = v | 用键k来将值赋值给映射m。如果映射m中的k已存在，则将之前的值舍弃 |
| delete(m, k) | 将键k及其相关的值从映射m中删除，如果k不存在则不执行任何操作 |
| v := m[k] | 从映射m中取得键k相对应的值并赋值给v。如果k不存在，则将映射类型的0值赋值v |
| v, found := m[k] | 从映射m中取得键k相对应的值并赋值给v， 并将found的值赋值为false。如果k不存在，则found为false |
| len(m) | 返回映射m中的项 |
| k := range m | 遍历映射m中的键 |
| k, v := range m | 同时遍历映射中的键和值 |

接下来我们练习下。使用VIM创建源文件map\_t.go输入以下代码:

package main

import (

"fmt"

)

func main() {

shiyanlou := make(map[string]string) // 与 map[string]string 相同

shiyanlou["golang"] = "docker"

shiyanlou["python"] = "flask web framework"

shiyanlou["linux"] = "sys administrator"

fmt.Print("Traverse all keys: ")

for key := range shiyanlou { // 遍历了映射的所有键

fmt.Printf("% s ", key)

}

fmt.Println()

delete(shiyanlou, "linux") // 从映射中删除键"linux"及其值

shiyanlou["golang"] = "beego web framework" // 更新键“golang"的值

v, found := shiyanlou["linux"]

fmt.Printf("Found key \"linux\" Yes or False: %t, value of key \"linux\": \"%s\"", found, v)

fmt.Println()

fmt.Println("Traverse all keys/values after changed:")

for k, v := range shiyanlou { //遍历了映射的所有键/值对

fmt.Printf("\"%s\": \"%s\"\n", k, v)

}

}

以上代码中，我们首先创建了一个映射，然后赋值了3个键/值对，然后我们遍历了映射中的所有键，使用delete()函数删除了映射中的一个键，然后再次遍历打印了映射。映射的操作都非常简单，多多练习即可。

运行结果如下：

$ go run map\_t.go

Traverse all keys: golang python linux

Found key "linux" Yes or False: false, value of key "linux": ""

Traverse all keys/values after changed:"golang": "beego web framework""python": "flask web framework"

## **作业**

请使用映射实现统计文章中每个单词出现的次数。