课程安排，请关注微信公众平台或者官方微博

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，我是彬哥，
欢迎大家来到 Golang 语言社区云课堂课程的学习。
社区论坛网址：www.golang.ltd
技术交流群 ： 221273219
微信公众号 ：Golang 语言社区
微信服务号 ：Golang 技术社区

第一季 Go 语言基础、进阶、提高课程

第二十一节 深入理解 slice

1、Go 语言中数组(array)和 slice 的区别

	数组(array)	Slice
长度	固定	非固定，动态增长(append())
存储的数据类型	相同类型(var vararray [10]int)	相同类型（var varslice []int）
内存模型	值类型，每次传递产生副本(内存复制一份，而不是引用源数组在内存的地址)	引用类型（底层指向数组），每个 slice 都是源数组在内存中的地址的一个引用;地址拷贝
游戏服务器开发	看游戏类型，要求不高的小游戏，像三消系列可以使用；大型游戏计算量的功能尽量少用。	用的比较多的数据结构： 1 指向原生数组的指针，而且是指定 slice 的开始位置。 2 slice 中的长度(len)，即 slice 的长度。 3 slice 已分配的存储空间

		（cap），也是 slice 开始位置到数组的最后一个位置的长度。

从底层实现的角度来看，slice 实际上仍然使用数组来管理元素，基于数组，slice 添加了一系列管理功能，可以随时动态扩充存放空间，并且可以被随意传递而不会导致管理的元素被重复复制。

2、数组（array）

Go 语言数组中每个元素是按照索引来访问的，索引从 0 到数组长度减 1.Go 语言内置函数 len 可以返回数组中的元素的个数。

<1> 数组初始化:

Var a [3]int // 3 个整型的数组，初始值为 3 个 0

Arr:=[5]int{1,2,3,4,5} // 长度为 5

Var array1 = […]int{7,8,9}// 长度不固定

Var array2 = […]int{99:-1} // 长度为 100 的数组，只有最后一个是-1，其他的都是 0

<2> 二维、多维 数组:

Var array_name [n][n]type

二维数组可以被认为具有 x 个行和 y 个列的表、包含三行四列的二维数组 a,可以如下所示：

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

实例 二维数组访问：

```
package main

import "fmt"

func main() {

   /* an array with 5 rows and 2 columns*/

   var a = [5][2]int{ {0,0}, {1,2}, {2,4}, {3,6},{4,8}}

   var i, j int
```

```
   /* output each array element's value */
   for  i = 0; i < 5; i++ {
      for j = 0; j < 2; j++ {
         fmt.Printf("a[%d][%d] = %d\n", i,j, a[i][j] )
      }
   }
}
```

3、切片（slice）

Slice 表示一个拥有相同类型元素的可变长度序列。

Var var_slice []type

创建 4 种 如下：

Make([]type,len,cap)

Make([]type,len)

[]type{}

[]type{v1,v2,v3,…,vN}

<1> 初始化

Slice1 := []int{1,2,3}

从数组中构建 slice

Array1 := [10]int{1,2,3,4,5,6,7,8,9,0}

Slice2 := Array1[2:8]

Slice3 :=make([]int,100,200) // make 函数初始化，len = 10,cap = 20

注意：

s = a[0:8] //s 指向数组 a[0]至 a[7]

s = s[2:5] //s 指向原切片的 s[2],s[3],s[4]三个元素。

//用[x:y]时总会指向 x 至 y -1 的元素

//当 x 为首个元素 或 y 为最后一个元素时可以省略不写

len 和 cap 关系

在追加元素时，如果容量 cap 不足时，cap 一般变为原来的 2 倍来实现扩容，见以下代码最后两个输出：

```
package main
import "fmt"
func print_info(my_slice[]int) {
    fmt.Println("len :",len(my_slice))
    fmt.Println("cap :",cap(my_slice))
    for i,v :=range my_slice{
        fmt.Println("element[",i,"]=",v)
    }

}

func main() {
    my_slice01:=[]int{1,2,3,4,5}
    my_slice02:=make([]int,5)
    my_slice03:=make([]int,5,6)
    my_slice04:=append(my_slice03,8,9,10)
    print_info(my_slice01)
    print_info(my_slice02)
    print_info(my_slice03)
    print_info(my_slice04)
}
```

slice 的 cap 扩容规则：

- 如果新的大小是当前大小 2 倍以上，则直接扩容为这个新的 cap；
- 否则循环以下操作：如果当前大小小于 1024，按每次 2 倍增长，否则每次按当前大小 1/4 增长。直到增长的大小超过或等于新 cap。

slice 做函数参数

注意和数组作为函数参数的区别：

```
Package mian
```

```
Import(

“fmt”

)

Func main(){

 A :=[]int{1,2,3,4,5}

Var b = a[0:3]

Var c = […]int{1,2,3,3,4,5,6}

D:=c[0:2]

sliceInfo(b)

}

Func sliceInfo(x []int){

Fmt.println(len(x),cap(x),x)

}
```

4、切片（slice）的深拷贝与浅拷贝

深浅拷贝的区别：

浅拷贝是将原始对象中的数据型字段拷贝到新对象中去，将引用型字段的“引用”复制到新对象中去，不把“引用的对象”复制进去，所以原始对象和新对象引用同一对象，新对象中的引用型字段发生变化会导致原始对象中的对应字段也发生变化。

深拷贝是在引用方面不同，深拷贝就是创建一个新的和原始字段的内容相同的字段，是两个一样大的数据段，所以两者的引用是不同的，之后的新对象中的引用型字段发生改变，不会引起原始对象中的字段发生改变。

浅拷贝 例子：

```
func main() {
    // 切片实质上是对底层匿名数组的引用
    slice := make([]int, 5, 5)
    slice1 := slice
    slice2 := slice[:]
    slice3 := slice[0:4]
    slice4 := slice[1:5]
    slice[1] = 1
    fmt.Println(slice)//[0 1 0 0 0]
    fmt.Println(slice1)//[0 1 0 0 0]
    fmt.Println(slice2)//[0 1 0 0 0]
    fmt.Println(slice3)//[0 1 0 0]
    fmt.Println(slice4)//[1 0 0 0]
  }
```

深拷贝 例子：

```
func main() {
    // 当元素数量超过容量
    // 切片会在底层申请新的数组
    slice := make([]int, 5, 5)
    slice1 := slice
    slice = append(slice, 1)
    slice[0] = 1
    fmt.Println(slice)//[1 0 0 0 0 1]
    fmt.Println(slice1)//[0 0 0 0 0]
    // copy 函数提供深拷贝功能
    // 但需要在拷贝前申请空间
    slice2 := make([]int, 4, 4)
    slice3 := make([]int, 5, 5)
    fmt.Println(copy(slice2, slice))//4
    fmt.Println(copy(slice3, slice))//5
    slice2[1] = 2
    slice3[1] = 3
    fmt.Println(slice)//[1 0 0 0 0 1]
    fmt.Println(slice2)//[1 2 0 0]
    fmt.Println(slice3)//[1 3 0 0 0]
}
```

注意：深拷贝容易引起内存泄露，最好实现自己的拷贝函数

5、切片（slice）与内存复制 memcpy 的实现方法

Go 语言原则上不支持内存的直接操作访问，但是提供了切片功能。最初我以为切片就是动态数组，实际程序设计过程中发现，切片是提供数组一个内存片段的一个合法的手段，利用切片功能，实际上我们可以自由访问数组的任何一个片段，因而可以借助 copy 函数，实现内存复制。

不同类型之间的数据复制，可以借助 unsafe 取出变量地址，类型转换为数组后，利用数组切片，实现内存复制。

```
package main
import (
    "fmt"
    "unsafe"
)
func main() {
    //数组之间的数据复制
    var a = [10]int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    var b = [10]int{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
    copy(a[5:8], b[0:])
    fmt.Println(a, b)
    //不同数据类型之间的复制
```

```
    var c uint32 = 0x04030201
    var d [4]byte
    p := unsafe.Pointer(&c)
    q := (*[4]byte)(p)
    copy(d[0:], (*q)[0:])
    fmt.Println(d)
}
```

运行结果：

```
[0 1 2 3 4 -1 -1 -1 8 9] [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
[1 2 3 4]
```