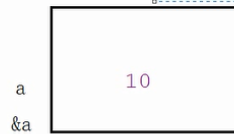


## 1. 一个变量a一般有两种含义, 一种代表内存里面的内容, 一种代表地址

```
func main() {  
    var a int = 10  
    //每个变量有2层含义: 变量的内存, 变量的地址  
    fmt.Printf("a = %d\n", a) //变量的内存  
    fmt.Printf("&a = %v\n", &a)  
}
```

a ==> 内存的内容  
&a ==> 内存外面的标号, 地址, 也叫指针



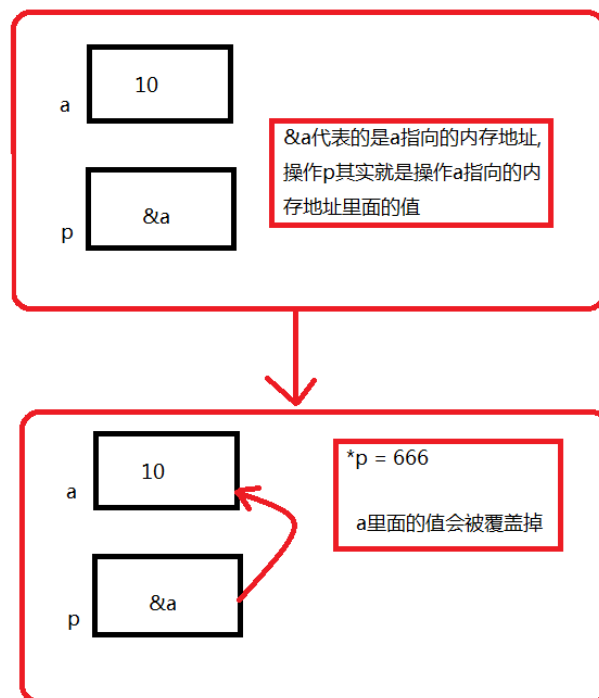
内存: =====> 教室  
&a : =====> 教室外面的门牌号

内存用来存放东西      a    =    10  
教室里面也是放桌子    教室 =    桌子

## 2. 指针的基本操作

```
func main() {  
    //定义一个指针变量  
    var p *int  
    var a = 10  
    fmt.Println("a is ", a)  
    p = &a  
    *p = 999  
    fmt.Println("after a is ", a)  
}
```

```
// 16_指针的基本操作  
package main  
  
import (  
    "fmt"  
)  
  
func main() {  
    //定义一个指针变量  
    var p *int  
    var a = 10  
    fmt.Println("a is ", a)  
    p = &a  
    *p = 999  
    fmt.Println("after a is ", a)  
}
```



## 3. 坚决不能操作没有合法指向的指针

## 4.new函数

```
1 // 17_new函数的使用
2 package main
3
4 import (
5     "fmt"
6 )
7
8 func main() {
9     //new函数可以返回一个指针的变量
10    p := new(int)
11    *p = 999
12    fmt.Println(*p is ", *p)
13 }
14
```

```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

E:\code\go\src>go run 16_指针的基本操作.go
a is 10
after a is 999

E:\code\go\src>go run 17_new函数的使用.go
*p is 999

E:\code\go\src>
```

## 5.通过指针解决值传递存在的问题

```
18_指针解决值传递存在的问题.go
1 // 18_指针解决值传递存在的问题
2 package main
3
4 import (
5     "fmt"
6 )
7
8 func swap(p1, p2 *int) {
9     *p1, *p2 = *p2, *p1
10 }
11
12 func main() {
13     a := 10
14     b := 22
15     fmt.Println("a is ", a, " b is ", b)
16     swap(&a, &b)
17     fmt.Println("a is ", a, " b is ", b)
18 }
```

```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

E:\code\go\src>go run 16_指针的基本操作.go
a is 10
after a is 999

E:\code\go\src>go run 17_new函数的使用.go
*p is 999

E:\code\go\src>go run 18_指针解决值传递存在的问题.go
a is 10 b is 22
a is 22 b is 10

E:\code\go\src>
```

6.数组定义的时候必须使用常量,操作数组元素要从0开始到len()-1结束,不对称元素,这个数字叫做下标

## 7.二维数组的定义

```
19_数组的初始化
1 // 19_数组的初始化
2 package main
3
4 import (
5     "fmt"
6 )
7
8 func main() {
9     arr := [4]int{2: 12, 3: 22}
10    fmt.Println("arr is ", arr)
11
12    //二维数组的初始化
13    secondArr := [2][3]int{1: {4, 5, 6}}
14    fmt.Println("secondArr is ", secondArr)
15 }
16
```

```
管理员: C:\Windows\system32\cmd.exe
E:\code\go\src>go run 17_new函数的使用.go
*p is 999

E:\code\go\src>go run 18_指针解决值传递存在的问题.go
a is 10 b is 22
a is 22 b is 10

E:\code\go\src>go run 19_数组的初始化.go
arr is [0 0 12 22]

E:\code\go\src>go run 19_数组的初始化.go
# command-line-arguments
.\19_数组的初始化.go:13:34: syntax error: unexpected

E:\code\go\src>go run 19_数组的初始化.go
arr is [0 0 12 22]
secondArr is [[1 2 3] [4 5 6]]

E:\code\go\src>go run 19_数组的初始化.go
arr is [0 0 12 22]
secondArr is [[0 0 0] [4 5 6]]
```

## 8.数组的比较和赋值

两个数组进行比较的前提是两个数组的类型要一样,只能使用==和!=来进行比较运算

## 9.生成随机数

```
1 // 20_随机数的产生和使用
2 package main
3
4 import (
5     "fmt"
6     "math/rand"
7     "time"
8 )
9
10 func main() {
11     //产生随机数需要用到一个包,rand
12     //在生成随机数之前需要先seed一下,如果每次传入seed的整型常量的值都一样的话,那么每次生成的
13     //随机数都是一样的,因此这里建议使用当前的系统时间来作为种子
14     rand.Seed(time.Now().UnixNano())
15     for i := 0; i < 5; i++ {
16         //newIntn(100)这里的100表示生成[0,100)之间的任意整数
17         fmt.Println("rand is ", rand.Intn(100))
18     }
19 }
20
```

```
管理员: C:\Windows\system32\cmd.exe
rand is 14
E:\code\go\src>go run 20_随机数的产生和使用.go
rand is 55
rand is 12
rand is 5
rand is 34
rand is 94
E:\code\go\src>go run 20_随机数的产生和使用.go
rand is 62
rand is 5
rand is 57
rand is 73
rand is 41
E:\code\go\src>go run 20_随机数的产生和使用.go
rand is 34
rand is 85
rand is 75
rand is 69
rand is 80
```

## 10.生成随机数和冒泡排序结合

```
    "fmt"
    "math/rand"
    "time"
)

func main() {
    //随机产生一个数组
    var arr [5]int

    rand.Seed(time.Now().UnixNano())
    for i := 0; i < len(arr); i++ {
        arr[i] = rand.Intn(100)
    }

    //接下来进行排序
    var temp int
    for i := 0; i < len(arr)-1; i++ {
        for j := 0; j < len(arr)-1-i; j++ {
            if arr[j] > arr[j+1] {
                temp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = temp
            }
        }
    }

    fmt.Println("arr is ", arr)
}
```

```
管理员: C:\Windows\system32\cmd.exe
rand is 57
rand is 73
rand is 41
E:\code\go\src>go run 20_随机数的产生和使用.go
rand is 34
rand is 85
rand is 75
rand is 69
rand is 80
E:\code\go\src>go run 21_冒泡排序.go
arr is [1 40 56 80 95]
E:\code\go\src>go run 21_冒泡排序.go
arr is [20 26 48 59 89]
E:\code\go\src>go run 21_冒泡排序.go
arr is [14 34 47 67 92]
E:\code\go\src>go run 21_冒泡排序.go
arr is [29 48 59 60 72]
E:\code\go\src>
半:
```

## 11.数组指针的传递

```
1 // 22_使用数组指针来弥补数组的值传递缺陷
2 package main
3
4 import (
5     "fmt"
6 )
7
8 func change(arr [5]int) {
9     arr[0] = 21
10    fmt.Println("arr[0] is ", arr[0])
11}
12
13 func changeP(arr *[5]int) {
14     (*arr)[0] = 33
15    fmt.Println("arr[0] is ", *arr)
16}
17
18 func main() {
19     var arr [5]int = [5]int{1, 2, 23, 12, 11}
20     changeP(&arr)
21     fmt.Println("arr[0] is ", arr[0])
22}
23
```

```
管理员: C:\Windows\system32\cmd.exe
E:\code\go\src>go run 22_使用数组指针来弥补数组的值传递缺陷.go
arr[0] is [33 2 23 12 11]
arr[0] is 33
E:\code\go\src>
```

## 12.切片和数组最主要的区别在于切片定义的时候不需要指定长度

## 13.数组和切片的区别,使用make也可以新建一个切片 arr := make([]int,5)

```

package main //必须有个main包

import "fmt"

func main() {
    array := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    // [low:high:max] 取下标从low开始的元素, len=high-low, cap=max-low
    s1 := array[:] // [0:len(array):len(array)] 不指定容量和长度一样
    fmt.Println("s1 = ", s1)
    fmt.Printf("len = %d, cap = %d\n", len(s1), cap(s1))

    // 操作某个元素, 和数组操作方式一样
    data := array[1]
    fmt.Println("data = ", data)

    s2 := array[3:6:7] // a[3], a[4], a[5]    len = 6-3=3    cap = 7-3=4
    fmt.Println("s2 = ", s2)
    fmt.Printf("len = %d, cap = %d\n", len(s2), cap(s2))

    s3 := array[:6] // 从0开始, 去6个元素, 容量也是6, 常用
    fmt.Println("s3 = ", s3)
    fmt.Printf("len = %d, cap = %d\n", len(s3), cap(s3))

    s4 := array[3:] // 从下标为3开始, 到结尾

```

#### 14. 切片和底层数组的关系, 实际的切片只有一个

```

// 23_创建一个切片
package main
import (
    "fmt"
)

func testSlice() {
    arr := []int{1, 2, 3, 8, 9, 10, 12, 23, 14, 25}
    slice1 := arr[2:5]
    slice1[2] = 777
    slice2 := slice1[2:7]
    slice2[4] = 900
    fmt.Println(arr, "\n", slice1, "\n", slice2)
}

func testSlice_01() {
    arr := [5]int{1, 2, 3, 9, 10}
    slice := arr[1:3:5]
    for i, data := range slice {
        fmt.Printf("slice[%d] is %d\n", i, data)
    }
    fmt.Printf("len is %d, cap is %d", len(slice), cap(slice))
}

func main() {
    testSlice()
}

```

```

管理员: C:\Windows\system32\cmd.exe
len is %d ,cap is %d 2 4

E:\code\go\src>go run 23_创建一个切片.go
slice[0] is 2
slice[1] is 3
len is 2 ,cap is 4
E:\code\go\src>go run 23_创建一个切片.go
[1 2 3 8 9 10 12 23 14 25] [3 8 9] [9 10 12 23 14]

E:\code\go\src>go run 23_创建一个切片.go
[1 2 3 8 9 10 12 23 14 25] [3 8 9] [9 10 12 23 14]

E:\code\go\src>go run 23_创建一个切片.go
[1 2 3 8 9 10 12 23 14 25]
[3 8 9]
[9 10 12 23 14]

E:\code\go\src>go run 23_创建一个切片.go
[1 2 3 8 777 10 12 23 900 25]
[3 8 777]
[777 10 12 23 900]

E:\code\go\src>
半:

```

#### 15. copy的使用

copy(destSlice,srcSlice),将srcSlice赋值给destSlice

#### 16. 创建一个猜数字的游戏

// 25\_猜数字的游戏

```
package main
```

```

import (
    "fmt"
    "math/rand"

```

```
"time"  
)
```

//需求描述,生成一个随机的四位数,从命令行输入以后和原来的数做对比

//生成四位随机数的函数

```
func createRandNum() (randNum int) {  
    rand.Seed(time.Now().UnixNano())  
  
    for {  
        randNum = rand.Intn(10000)  
        if randNum > 999 {  
            break  
        }  
    }  
    return  
}
```

//从命令行接收一个四位数

```
func getNumFromCmd() (inputNum int) {  
    for {  
        fmt.Scan(&inputNum)  
        //对输入的数字进行检测,如果是合法的四位数,才返回  
        if inputNum > 999 && inputNum < 10000 {  
            return  
        } else {  
            fmt.Println("输入的数字不合法,请重新输入")  
        }  
    }  
}
```

//定义一个函数,将一个数字拆分成一个切片

```
func getSliceFromNum(num int, slice []int) {  
    slice[0] = num / 1000  
    slice[1] = num / 100 % 10
```

```

slice[2] = num / 10 % 10
slice[3] = num % 10

}

func main() {
    //生成一个随机的四位数
    randNum := createRandNum()
    fmt.Println("randNum is ", randNum)

    //拿输入的数和随机生成的数按位比较
    //创建一个切片
    slice_RandNum := make([]int, 4)
    getSlicFromNum(randNum, slice_RandNum)
    fmt.Println("slice_RandNum is ", slice_RandNum)

    for {
        //从命令行接收一个四位数
        inputNum := getNumFromCmd()
        fmt.Println("输入的数字是 ", inputNum)

        //将从命令行输入的数字切分成切片
        slice_InputNum := make([]int, 4)
        getSlicFromNum(inputNum, slice_InputNum)
        fmt.Println("slice_InputNum is ", slice_InputNum)
        var n int
        //两个切片按位进行比较
        for i, data := range slice_RandNum {
            if data > slice_InputNum[i] {
                fmt.Printf("第%d位输入的有点小\n", i+1)
            } else if data < slice_InputNum[i] {
                fmt.Printf("第%d位输入的有点大\n", i+1)
            } else {
                fmt.Printf("第%d位输入正确\n", i+1)
            }
            n++
        }
    }
}

```

```

    }
}
if n == 4 {
    break
}
}
}

```

The screenshot shows a Go IDE with a file named '25\_猜数字的游戏.go'. The code implements a number guessing game where a random 4-digit number is generated, and the user has 10 attempts to guess it. Feedback is provided for each digit: '正确' (correct), '有点大' (a bit big), or '有点小' (a bit small). The terminal output shows three successful game sessions: 7080, 7090, and 7060.

```

1 // 25_猜数字的游戏
2 package main
3
4 import (
5     "fmt"
6     "math/rand"
7     "time"
8 )
9
10 //需求描述,生成一个随机的四位数,从命令行输入以后和原来的数做对比
11
12 //生成四位随机数的函数
13 func createRandNum() (randNum int) {
14     rand.Seed(time.Now().UnixNano())
15
16     for {
17         randNum = rand.Intn(10000)
18         if randNum > 999 {
19             break
20         }
21     }
22     return
23 }
24

```

```

第4位输入正确
7080
输入的数字是 7080
slice_InputNum is [7 0 8 0]
第1位输入正确
第2位输入正确
第3位输入的有点大
第4位输入正确
7090
输入的数字是 7090
slice_InputNum is [7 0 9 0]
第1位输入正确
第2位输入正确
第3位输入的有点大
第4位输入正确
7060
输入的数字是 7060
slice_InputNum is [7 0 6 0]
第1位输入正确
第2位输入正确
第3位输入正确
第4位输入正确
E:\code\go\src>
半:

```

## 17.map 字典,映射

```

func main() {
    //定义一个map
    var students map[int]string = map[int]string{1: "mike", 2: "jack", 3: "john"}
    for key, value := range students {
        fmt.Printf("key is %d,value is %s\n", key, value)
    }

    //删除某一元素
    delete(students, 2)
    fmt.Println("students is ", students)
}

```

## 18.结构体的初始化

```
//27_结构体变量的初始化
package main

import (
    "fmt"
)

type Student struct {
    age    int
    name   string
    addr   string
    score  int
}

func main() {
    //结构体变量的初始化一定要全部赋值
    s1 := Student{12, "mike", "上海", 87}
    fmt.Println("student is ", s1)

    //还可以指定某一个变量进行赋值
    s2 := Student{age: 23, score: 89}
    fmt.Println("student2 is ", s2)
}
```

```
管理员: C:\Windows\system32\cmd.exe
E:\code\go\src>go run 26_map的遍历.go
key is 1,value is mike
key is 2,value is jack
key is 3,value is john
students is map[1:mike 3:john]

E:\code\go\src>go run 27_结构体变量的初始化.go
# command-line-arguments
.\27_结构体变量的初始化.go:17:8: undefined: Studnet

E:\code\go\src>go run 27_结构体变量的初始化.go
student is {12 mike 上海 87}

E:\code\go\src>go run 27_结构体变量的初始化.go
# command-line-arguments
.\27_结构体变量的初始化.go:17:30: cannot use 87 (type int) as
ld value
.\27_结构体变量的初始化.go:17:30: too few values in struct ini

E:\code\go\src>go run 27_结构体变量的初始化.go
student is {12 mike 上海 87}
student2 is {23 89}

E:\code\go\src>
```

## 19.结构体指针变量的初始化

```
// 28_操作结构体的指针
package main

import (
    "fmt"
)

type Student struct {
    age    int
    name   string
    score  int
}

func main() {
    //定义一个结构体指针
    var s1 *Student = &Student{23, "John", 89}
    fmt.Println("s1 is ", s1)

    s2 := &Student{29, "Hellen", 90}
    fmt.Println("s2 is ", s2)
}
```

```
管理员: C:\Windows\system32\cmd.exe
student is {12 mike 上海 87}

E:\code\go\src>go run 27_结构体变量的初始化.go
# command-line-arguments
.\27_结构体变量的初始化.go:17:30: cannot use 87 (type int) as type
ld value
.\27_结构体变量的初始化.go:17:30: too few values in struct initial

E:\code\go\src>go run 27_结构体变量的初始化.go
student is {12 mike 上海 87}
student2 is {23 89}

E:\code\go\src>go run 28_操作结构体的指针.go
s1 is &{23 John 89}

E:\code\go\src>go run 28_操作结构体的指针.go
# command-line-arguments
.\28_操作结构体的指针.go:19:9: undefined: Studnet

E:\code\go\src>go run 28_操作结构体的指针.go
s1 is &{23 John 89}
*s2 is {29 Hellen 90}

E:\code\go\src>
```

## 20.go语言工程的可见性

如果想使用别的包的函数,结构体类型或者结构体成员,这些变量名称的首字母必须是大写的,不然不可见,如果首字母是小写的话,只能在同一个包下面使用