复杂数据类型

数组

```go
package main

//数组
//存放元素的容器
//必须指明类型和容量
//长度是数组类型的一部分
import "fmt"

func main() {
    var arr [5]int
    var arr2 [3]int
    fmt.Printf("arr:%T  arr2:%T \n", arr, arr2)
}
```

数组长度是数组类型的一部分

```
>4 go 设置调用>
arr:[5]int  arr2:[3]int

进程 已完成，退出代码为 0
```

数组的初始化一般为‘0’值

```
>4 go 设置调用>
 [0 0 0 0 0] [0 0 0]

 进程 已完成，退出代码为 0
```

```go
a1 := [3]int{1, 1, 2}
fmt.Println(a1)
//方式2
a100 := [...]int{1, 2, 3, 4, 5, 2, 14, 346, 767} //根据初始化值的个数判断数组的长度
fmt.Println(a100)
//方式3
a2 := [5]int{0: 1, 4: 2} //根据索引初始化
fmt.Println(a2)
```

运行结果：

```
[1 1 2]
[1 2 3 4 5 2 14 346 767]
[1 0 0 0 2]
```

数组的遍历

```go
//方式1:通过索引
for i := 0; i < len(a100); i++ {
    fmt.Println(a100[i])
}
//方式2: for range
for index, value := range a100 {
    fmt.Println(index, value)
}
```

二维数组的定义

Go

```go
arr := [3][2]int{{1, 2}, {3, 4}, {5, 6}}
for i := 0; i <= 2; i++ {
    for j := 0; j <= 1; j++ {
        fmt.Println(arr[i][j])
    }
}
```

引子

因为数组的长度是固定的并且数组长度属于类型的一部分，所以数组有很多的局限性。例如：

```go
func arraySum(x [3]int) int{
    sum := 0
    for _, v := range x{
        sum = sum + v
    }
    return sum
}
```

这个求和函数只能接受 `[3]int` 类型，其他的都不支持。再比如，

```go
a := [3]int{1, 2, 3}
```

数组a中已经有三个元素了，我们不能再继续往数组a中添加新元素了。

切片（slice）

拥有相同数据类型的可变序列

由于切片的底层就是一个数组，所以我们可以基于数组定义切片。

```go
func main() {
    // 基于数组定义切片
    a := [5]int{55, 56, 57, 58, 59}
    b := a[1:4]                     //基于数组a创建切片，包括元素a[1],a[2],a[3]
    fmt.Println(b)                  //[56 57 58]
    fmt.Printf("type of b:%T\n", b) //type of b:[]int
}
```

还支持如下方式：

```go
c := a[1:] //[56 57 58 59]
d := a[:4] //[55 56 57]
e := a[:]  //[55 56 57 58 59]
```

```go
func main() {
    var s1 []int    //定义一个存放int类型的切片
    var s2 []string //定义一个string类型的切片
    fmt.Println(s1, s2)
    fmt.Println(s1 == nil)
    fmt.Println(s2 == nil)
    //初始化
    s1 = []int{1, 2, 3}
    s2 = []string{"沙河", "湛江", "深圳"}
    fmt.Println(s1, s2)
    fmt.Println(s1 == nil)
    fmt.Println(s2 == nil)
    fmt.Println(len(s1), cap(s1))
    fmt.Println(len(s2), cap(s2))

    //由数组等到切片
    a1 := [...]int{1, 2, 3, 4, 5, 6, 7, 8, 9}
    s3 := a1[0:4] //0,1,2,3
    fmt.Println(s3)
    fmt.Printf("%T\n", s3)
    fmt.Printf("%T\n", a1)
}
```

```
[] []
true
true
[1 2 3] [沙河 湛江 深圳]
false
false
3 3
3 3
[1 2 3 4]
[]int
[9]int
```

注意：切片的容量是底层数组的容量

1.切片指向一个底层数组

2.切片的长度就是元素个数

3.切片的容量就是底层数组从切片的第一个元素到最后一个元素的数量

```go
a1 := [...]int{1, 2, 3, 4, 5, 6, 7, 8, 9}
s3 := a1[4:] //0,1,2,3
fmt.Println(s3)
fmt.Printf("%T\n", s3)
fmt.Printf("%T\n", a1)
a1[8] = 123
fmt.Println(s3)
```

这里只将a1数组中的最后一个值修改但是打印由这个数组制作的切片，发现切片的值也发生改变，更验证了切片的底层是一个数组。

make函数构造切片

```go
func main() {
    s1 := make([]int, 5, 10) //类型，长度，容量
    fmt.Printf("%v  %v  %v\n", len(s1), cap(s1), s1)
}
```

```
<4 go 设置调用>
5  10  [0 0 0 0 0]

进程 已完成，退出代码为 0
```

切片的本质

就是一个框，框住了一块连续的内存

本质都是保存在一个底层数组中。

```go
s3 := []int{1, 3, 5}
    s4 := s3
    fmt.Println(s3, s4)
    s3[0] = 1234
    fmt.Println(s3, s4)

    //索引遍历
    for i := 0; i < len(s3); i++ {
        fmt.Println(s3[i])
    }
    //for range循环
    for index, val := range s3 {
        fmt.Println(index, val)
    }
```

切片追加元素append

```go
func main() {
    //
    s1 := []string{"北京", "上海", "深圳"}
    //调用append函数必须用原来的切片变量接受返回值
    s1 = append(s1, "广东")
    fmt.Println(s1)
    fmt.Printf("%v %v %v", s1, len(s1), cap(s1))
}
```

复制切片copy

```go
a1 := []int{1, 3, 5}
    a2 := a1
    var a3 = make([]int, 3, 5)
    copy(a3, a1)
    a1[0] = 100
    fmt.Println(a1, a2, a3)
```

这里是为a3单独开辟一个空间

```
[Running] go run "d:\goproject\num1_go_progress\main.go"
[100 3 5] [100 3 5] [1 3 5]
```

删除切片中的值

go语言中没有内置删除切片的函数

但可以手写，如下

```go
    a1 := []int{1, 3, 5}
    a1 = append(a1[:1], a1[…]...)
    fmt.Println(a1)
```