Golang 中的数组

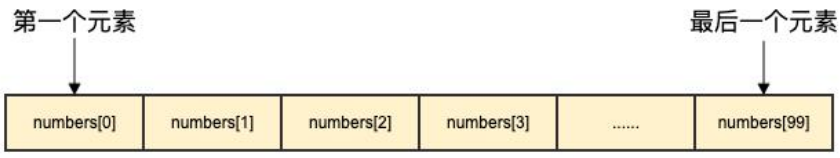
主讲教师：（大地）
合作网站：www.itying.com （IT 营）
我的专栏：https://www.itying.com/category-79-b0.html

1、Array(数组)的介绍 1
2、数组定义 2
3、数组的初始化 2
4、数组的遍历 3
5、数组是值类型 4
6、多维数组 4
7、数组练习题 6

1、Array(数组)的介绍

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)，这种类型可以是任意的原始类型，比如 int、string 等，也可以是用户自定义的类型。一个数组包含的元素个数被称为数组的长度。在 Golang 中数组是一个长度固定的数据类型，**数组的长度是类型的一部分**，也就是说 [5]int 和 [10]int 是两个不同的类型。Golang 中数组的另一个特点是占用内存的连续性，也就是说数组中的元素是被分配到连续的内存地址中的，因而索引数组元素的速度非常快。

和数组对应的类型是 Slice（切片），Slice 是可以增长和收缩的动态序列，功能也更灵活，但是想要理解 slice 工作原理的话需要先理解数组，所以本节主要为大家讲解数组的使用。



数组基本语法：

```
// 定义一个长度为3 元素类型为int 的数组a
var a [3]int
// 定义一个长度为3 元素类型为int 的数组b 并赋值
var b [3]int
b[0] = 80
b[1] = 100
b[2] = 96
```

2、数组定义

```
var 数组变量名 [元素数量]T
```

比如：var a [5]int， 数组的长度必须是常量，并且长度是数组类型的一部分。一旦定义，长度不能变。 [5]int 和[4]int 是不同的类型。

```
var a [3]int
var b [4]int
a = b //不可以这样做，因为此时a 和b 是不同的类型
```

数组可以通过下标进行访问，下标是从 0 开始，最后一个元素下标是：len-1，访问越界（下标在合法范围之外），则触发访问越界，会 panic。

3、数组的初始化

数组的初始化也有很多方式。

方法一

初始化数组时可以使用初始化列表来设置数组元素的值。

```
func main() {
    var testArray [3]int                        //数组会初始化为 int 类型的零值
    var numArray = [3]int{1, 2}                 //使用指定的初始值完成初始化
    var cityArray = [3]string{"北京", "上海", "深圳"} //使用指定的初始值完成初始化
    fmt.Println(testArray)                      //[0 0 0]
    fmt.Println(numArray)                       //[1 2 0]
```

```
    fmt.Println(cityArray)                          //[北京 上海 深圳]

}
```

方法二

按照上面的方法每次都要确保提供的初始值和数组长度一致，一般情况下我们可以让编译器根据初始值的个数自行推断数组的长度，例如：

```
func main() {
    var testArray [3]int
    var numArray = [...]int{1, 2}
    var cityArray = [...]string{"北京", "上海", "深圳"}
    fmt.Println(testArray)                                  //[0 0 0]
    fmt.Println(numArray)                                    //[1 2]
    fmt.Printf("type of numArray:%T\n", numArray)    //type of numArray:[2]int
    fmt.Println(cityArray)                                  //[北京 上海 深圳]
    fmt.Printf("type of cityArray:%T\n", cityArray) //type of cityArray:[3]string
}
```

方法三

我们还可以使用指定索引值的方式来初始化数组，例如:

```
func main() {
    a := [...]int{1: 1, 3: 5}
    fmt.Println(a)                          // [0 1 0 5]
    fmt.Printf("type of a:%T\n", a) //type of a:[4]int
}
```

4、数组的遍历

遍历数组 a 有以下两种方法：

```
func main() {
    var a = [...]string{"北京", "上海", "深圳"}
    // 方法1：for 循环遍历
    for i := 0; i < len(a); i++ {
        fmt.Println(a[i])
    }
    // 方法2：for range 遍历
```

```
    for index, value := range a {
        fmt.Println(index, value)
    }
}
```

5、数组是值类型

数组是值类型，赋值和传参会复制整个数组。因此改变副本的值，不会改变本身的值。

```
func modifyArray(x [3]int) {
    x[0] = 100
}

func modifyArray2(x [3][2]int) {
    x[2][0] = 100
}
func main() {
    a := [3]int{10, 20, 30}
    modifyArray(a) //在 modify 中修改的是 a 的副本 x
    fmt.Println(a) //[10 20 30]
    b := [3][2]int{
        {1, 1},
        {1, 1},
        {1, 1},
    }
    modifyArray2(b) //在 modify 中修改的是 b 的副本 x
    fmt.Println(b)  //[[1 1] [1 1] [1 1]]
}
```

注意：

1. 数组支持 “==“、”!=” 操作符，因为内存总是被初始化过的。
2. [n]*T 表示指针数组，*[n]T 表示数组指针、

6、多维数组

Go 语言是支持多维数组的，我们这里以二维数组为例（数组中又嵌套数组）。

```
var 数组变量名 [元素数量][元素数量]T
```

```
var variable_name [SIZE1][SIZE2]...[SIZEN] variable_type
```

	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]

二维数组的定义

```
func main() {
    a := [3][2]string{
        {"北京", "上海"},
        {"广州", "深圳"},
        {"成都", "重庆"},
    }
    fmt.Println(a) //[[北京 上海] [广州 深圳] [成都 重庆]]
    fmt.Println(a[2][1]) //支持索引取值:重庆
}
```

二维数组的遍历

```
func main() {
    a := [3][2]string{
        {"北京", "上海"},
        {"广州", "深圳"},
        {"成都", "重庆"},
    }
    for _, v1 := range a {
        for _, v2 := range v1 {
            fmt.Printf("%s\t", v2)
        }
        fmt.Println()
    }
}
```

输出：

```
北京    上海
广州    深圳
```

```
成都      重庆
```

注意： 多维数组**只有第一层**可以使用...来让编译器推导数组长度。例如：

```
//支持的写法
a := [...][2]string{
    {"北京", "上海"},
    {"广州", "深圳"},
    {"成都", "重庆"},
}
//不支持多维数组的内层使用...
b := [3][...]string{
    {"北京", "上海"},
    {"广州", "深圳"},
    {"成都", "重庆"},
}
```

7、数组练习题

1、请求出一个数组的和以及平均值。for-range

```
var intArr2 [5]int = [...]int {1, -1, 9, 90, 12}
sum := 0
for _, val := range intArr2 {
    //累计求和
    sum += val
}
//如何让平均值保留到小数.
fmt.Printf("sum=%v 平均值=%v \n\n", sum, float64(sum) / float64(len(intArr2)))
```

2、请求出一个数组的最大值，并得到对应的下标

1、声明一个数组 var intArr[5] = [...]int {1, -1, 12, 65, 11}

2、假定第一个元素就是最大值，下标就 0

3、然后从第二个元素开始循环比较，如果发现有更大，则交换

```
var intArr = [...]int{1, -1, 112, 65, 11}
maxValue := intArr[0]
maxIndex := 0
for i := 0; i < len(intArr); i++ {
    if maxValue < intArr[i] {
        maxValue = intArr[i]
```

```
            maxIndex = i
        }
    }

    fmt.Println("最大值", maxValue, "最大值索引值", maxIndex)
```

3、从数组[1, 3, 5, 7, 8]中找出和为 8 的两个元素的下标分别为(0,3)和(1,2)

```
arr := [...]int{1, 3, 5, 7, 8}
for i := 0; i < len(arr); i++ {
    for j := i + 1; j < len(arr); j++ {
        if arr[i]+arr[j] == 8 {
            fmt.Printf("(%v,%v)\n", arr[i], arr[j])
        }
    }
}
```