Go语言基础之数组

2017年6月19日 | Golang |11669 阅读

本文主要介绍Go语言中数组（array）及它的基本使用。

Array(数组)

数组是同一种数据类型元素的集合。 在Go语言中，数组从声明时就确定，使用时可以修改数组成员，但是数组大小不可变化。 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
```

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 `[10]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。

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。

```go
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：

```go
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：

```go
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
}
```

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)
    }
}
```

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）。

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"},
}
```

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 。

练习题

1. 求数组[1, 3, 5, 7, 8]所有元素的和
2. 找出数组中和为指定值的两个元素的下标，比如从数组[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)。