# 切片

作者：少林之巅

# 目录

1.切片定义

2. 切片基本操作

3. 切片传参

4. make和new的区别

5. 课后练习

# 切片定义

1. 切片是基于数组类型做的一层封装。它非常灵活，可以自动扩容。

```
var a []int
//定义一个int类型的空切片
```

# 切片定义

2. 切片初始化, a[start:end]创建一个包括从start到end-1的切片。

```
package main

import (
    "fmt"
)

func main() {
    a := [5]int{76, 77, 78, 79, 80}
    var b []int = a[1:4] //基于数组a创建一个切片, 包括元素a[1] a[2] a[3]
    fmt.Println(b)
}
```

# 切片定义

3. 切片初始化方法2。

```
package main

import (
    "fmt"
)

func main() {
    c := []int{6, 7, 8} //创建一个数组并返回一个切片
    fmt.Println(c)
}
```

# 切片基本操作

5. 数组切片的基本操作

a) arr[start:end]：包括start到end-1(包括end-1)之间的所有元素

b) arr[start:]：包括start到arr最后一个元素(包括最后一个元素)之间的所有元素

c) arr[:end]：包括0到end-1(包括end-1）之间的所有元素

d) arr[:]：包括整个数组的所有元素

# 切片基本操作

4. 切片修改

```
package main

import (
    "fmt"
)

func main() {
    //创建一个数组，其中[…]是编译器确定数组的长度,darr的长度是9
    darr := [...]int{57, 89, 90, 82, 100, 78, 67, 69, 59}
    //基于darr创建一个切片dslice,包括darr[2],darr[3],darr[4]三个元素
    dslice := darr[2:5]
    fmt.Println("array before",darr)
    for i := range dslice {
        //对于dslice中每个元素进行+1，其实修改是darr[2],darr[3],darr[4]
        dslice[i]++
    }
    fmt.Println("array after",darr)
}
```

# 切片基本操作

6. 切片修改

```
package main

import (
    "fmt"
)

func main() {
    numa := [3]int{78, 79 ,80}
    //创建一个切片，包含整个数组的所有元素
    nums1 := numa[:]
    nums2 := numa[:]
    fmt.Println("array before change 1",numa)
    nums1[0] = 100
    fmt.Println("array after modification to slice nums1", numa)
    nums2[1] = 101
    fmt.Println("array after modification to slice nums2", numa)
}
```

# 切片基本操作

7. 使用make创建切片

```
package main

import (
    "fmt"
)

func main() {
    //[]中没有长度
    i := make([]int, 5, 5)
    fmt.Println(i)
}
```

# 切片基本操作

8. 切片的长度和容量

```
package main

import (
    "fmt"
)

func main() {
    fruitarray := […]string{
            "apple", "orange", "grape",
            "mango", "water melon",
            "pine apple", "chikoo"}
    fruitslice := fruitarray[1:3]
    //长度是2, 容量is 6
    fmt.Printf("length of slice %d capacity %d",
          len(fruitslice), cap(fruitslice))
}
```

# 切片基本操作

9. 切片的再切片

```
package main

import (
    "fmt"
)
func main() {
    fruitarray := […]string{
          "apple", "orange", "grape", "mango",
          "water melon", "pine apple", "chikoo"}
    fruitslice := fruitarray[1:3]
    //长度是2, 容量是6
    fmt.Printf("length of slice %d capacity %d\n",
          len(fruitslice), cap(fruitslice))
    //再重新进行切片，不能大于数组fruitarray的长度，否则越界
    fruitslice = fruitslice[:cap(fruitslice)]
    fmt.Println("After re-slicing length is”,
         len(fruitslice), "and capacity is",cap(fruitslice))
}
```

# 切片基本操作

10.append操作

```
package main

import (
    "fmt"
)

func main() {
    cars := []string{"Ferrari", "Honda", “Ford"}
    //长度和容量都等于3
    fmt.Println("cars:", cars, "has old length",
        len(cars), "and capacity", cap(cars))
    cars = append(cars, “Toyota")
    //容量等于6
    fmt.Println("cars:", cars, "has new length",
        len(cars), "and capacity", cap(cars))
}
```

# 切片基本操作

## 11. 空切片

```
package main

import (
    "fmt"
)

func main() {
    //定义names是一个空切片，长度和容量都等于0
    //不能对空切片进行访问，否则panic
    var names []string
    if names == nil {
        fmt.Println("slice is nil going to append")
        names = append(names, "John", "Sebastian", "Vinay")
        fmt.Println("names contents:",names)
    }
}
```

# 切片基本操作

12. append一个切片

```
package main

import (
    "fmt"
)

func main() {
    veggies := []string{"potatoes","tomatoes","brinjal"}
    fruits := []string{“oranges","apples"}
    //fruits后面的3个点表示展开fruits切片成一个个元素
    food := append(veggies, fruits...)
    fmt.Println("food:",food)
}
```

# 切片传参

13. 切片传参

```
package main

import (
    "fmt"
)
//在函数内部修改numbers切片的值
func subtactOne(numbers []int) {
    for i := range numbers {
        numbers[i] -= 2
    }
}
func main() {
    nos := []int{8, 7, 6}
    fmt.Println("slice before function call", nos)
    subtactOne(nos)
    //nos修改生效了，说明切片是引用类型
    fmt.Println("slice after function call", nos)
}
```

# 切片定义

14. 切片拷贝

```
package main

import (
    "fmt"
)

func main() {
    veggies := []string{"potatoes","tomatoes","brinjal"}
    fruits := []string{“oranges","apples"}
    copy(veggies, fruits)
    fmt.Println(veggies, fruits)
}
```

# 切片遍历

15.切片遍历

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30
B := a[:]

for index, val :=  range b {
}
//和数组遍历是一样的
```

# make和new区别

16. make为内建类型slice、map和channel分配内存。

```
//初始化一个切片
s := make([]int, 10, 30)
```

17. new用于各种类型的内存分配，new返回是一个指针。

# 练习

1. 下列程序输出什么?

```
func main() {
    var sa = make ([]string,5,10);

    for i:=0;i<10;i++{
        sa=append(sa, fmt.Sprintf("%v",i))
    }
    fmt.Println(sa);
}
```

2. 使用golang标准包 “sort”对数组进行排序

3. 实现一个密码生成工具，支持以下功能：

**提示：可以用标准包 “flag”解析命令行参数**

a) 用户可以通过-l指定生成密码的长度

b) 用户可以通过-t指定生成密码的字符集，比如-t num生成全数字的密码
-t char 生成包含全英文字符的密码，-t mix包含生成数字和英文的密码，
-t advance 生成包含数字、英文以及特殊字符的密码