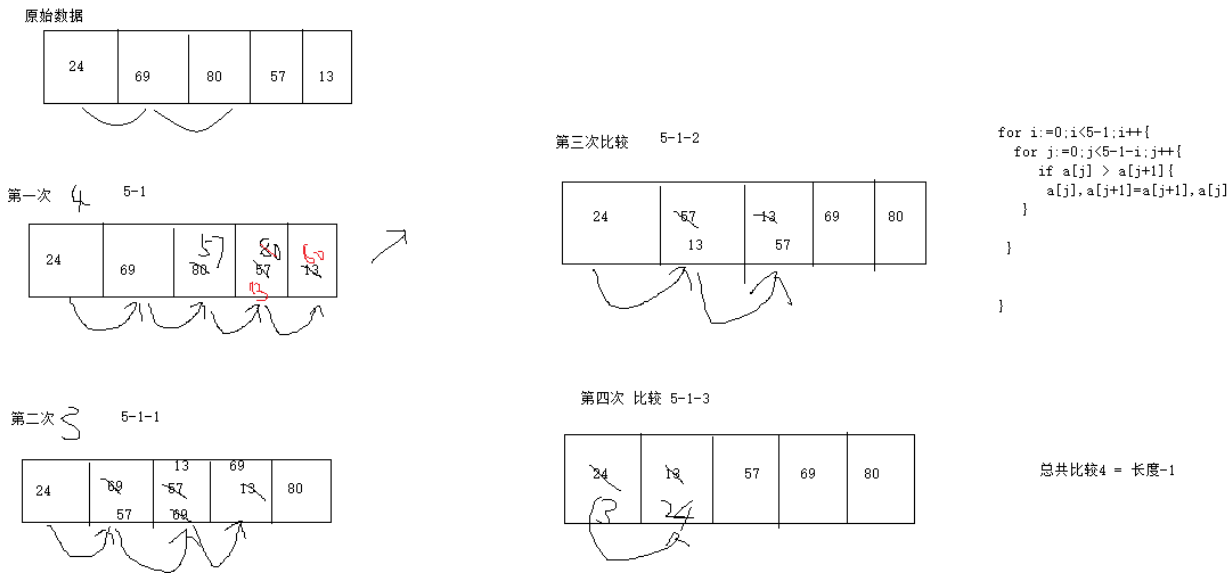
go区块链课程 主讲老师：张长志

冒泡排序



代码实战

```
package main

import "fmt"

func main()  {
    a :=[8]int{24,69,80,57,13,1,100}
    //算出a的长度
    //n :=len(a)
    //0 1 2 3
    //冒泡排序
    /*for i:=0;i<5-1;i++ {
       for j:=0;j<5-1-i;j++{
          if a[j] > a[j+1]{
             a[j],a[j+1] = a[j+1],a[j]
          }
       }
    }

    fmt.Printf("\n培训后：\n")
    for i:=0;i< 5;i++{
        fmt.Printf("%d\n",a[i])
```

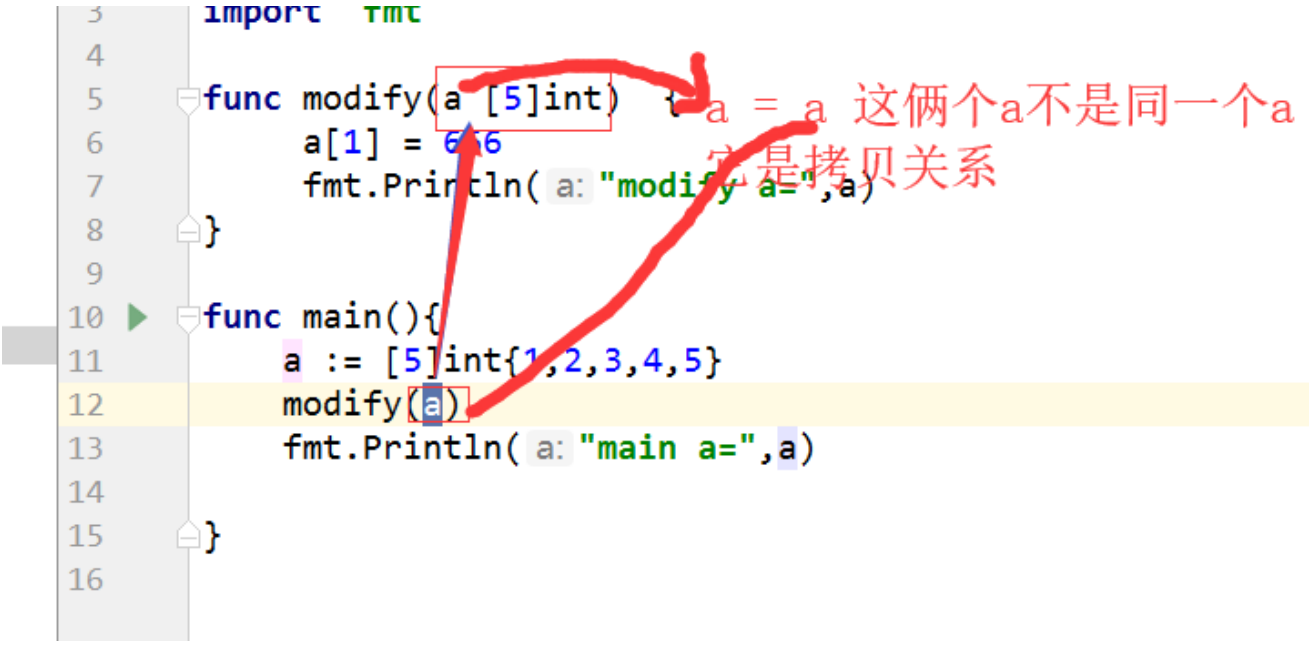
```
    }
    fmt.Println()*/

    //升序
   /* n := len(a)
    for i:=0;i<n-1;i++ {
        for j:=0;j<n-1-i;j++{
            if a[j] > a[j+1]{
                a[j],a[j+1] = a[j+1],a[j]
            }
        }
    }*/

    //降序
    n := len(a)
    for i:=0;i<n-1;i++ {
        for j:=0;j<n-1-i;j++{
            if a[j] < a[j+1]{
                a[j],a[j+1] = a[j+1],a[j]
            }
        }
    }


    fmt.Printf("\n排序后：\n")
    for i:=0;i< n;i++{
        fmt.Printf("%d\n",a[i])
    }
    fmt.Println()
}
```

数组做函数参数

数组做函数参数，它是值拷贝传递 实参数组的每个元素给形参数组拷贝一份，形参是实参的复制品

```
import "fmt"

func modify(a [5]int)  {
     a[1] = 666
     fmt.Println("modify a=",a)
}

func main(){
    a := [5]int{1,2,3,4,5}
    modify(a)
    fmt.Println("main a=",a)
```

```
}
```



数组指针做函数参数

数组 & == *p

p指向实现数组a，它是指向数组，它是数组指针 *p代表指针所指向的内存 就是实参

```
//*p =a 代表指针所指向的内存，就是实参
func modify1(p *[5]int)  {
    (*p)[0] = 666
    fmt.Println("modify *p=",*p)
}

func main(){

    a := [5]int{1,2,3,4,5}
    modify1(&a)
    fmt.Println("main:a=",a)

}
```

切片

数组的长度在定义之后无法再次修改；数组是值类型，每次传递都将产生一份副本。显然这种数据结构无法完全满足开发者的真实需求。Go语言提供了数组切片（slice）来弥补数组的不足。 切片并不是数组或数组指针，它通过内部指针和相关属性引用数组片段，以实现变长方案

[5]int{1,2,3,4,5} [6]int

```
s := a[0:3:5]  min max c
s := a[1:3:5]  min max c
```

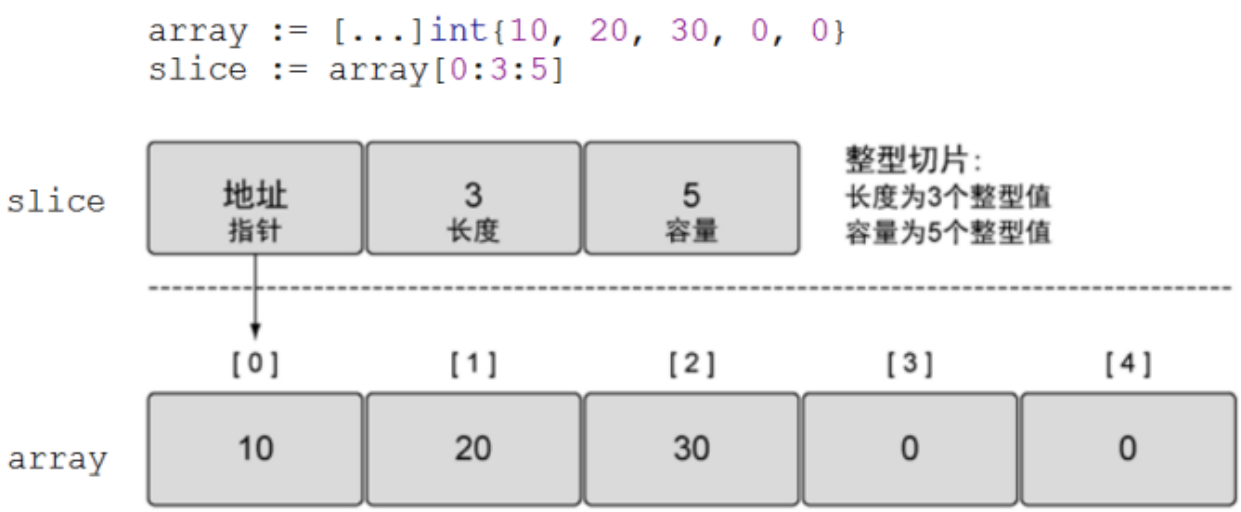
切片长度= 3-0 = 3

容量 = 5-0 =5

切片长度= 3-1 = 2

容量= 5-1=4

slice并不是真正意义上的动态数组，而是一个引用类型。slice总是指向一个底层array，slice的声明也可以像array一样，只是不需要长度。




```
import "fmt"

func main(){
    a := [5]int{1,2,3,4,5} //左包含右不包含
    s := a[0:3:5]
    fmt.Println("s=",s)
    fmt.Println("len(s)=",len(s))
    fmt.Println("cap(s)=",cap(s))
```

```
    s1 := a[1:3:5]
    fmt.Println("s1=",s1)
    fmt.Println("len(s1)=",len(s1))
    fmt.Println("cap(s1)=",cap(s1))

}
```

切片的加深理解

```
import "fmt"

func main(){
    a := [5]int{1,2,3,4,5} //左包含右不包含
    /*s := a[0:3:5]
    fmt.Println("s=",s)
    fmt.Println("len(s)=",len(s))
    fmt.Println("cap(s)=",cap(s))*/

    s1 := a[1:3:5]
    fmt.Println("s1=",s1)
    fmt.Println("len(s1)=",len(s1))
    fmt.Println("cap(s1)=",cap(s1))
    fmt.Printf("%p\n",&s1)
    fmt.Println("a=",a)

    s1 = append(s1, 1)
    fmt.Println("s1=",s1)
    fmt.Println("len(s1)=",len(s1))
    fmt.Println("cap(s1)=",cap(s1))
    fmt.Printf("%p\n",&s1)
    fmt.Println("a=",a)

    s1 = append(s1, 1)
    fmt.Println("s1=",s1)
    fmt.Println("len(s1)=",len(s1))
    fmt.Println("cap(s1)=",cap(s1))
    fmt.Printf("%p\n",&s1)
    fmt.Println("a=",a)

    s1 = append(s1, 1)
    fmt.Println("s1=",s1)
    fmt.Println("len(s1)=",len(s1))
    fmt.Println("cap(s1)=",cap(s1))
    fmt.Printf("%p\n",&s1)
    fmt.Println("a=",a)

    s1[0] = 9999
    fmt.Println("a=",a)

}
```

切片和数组的区别

1、切片是不定长的 不需要指定固定长度 可以增加

2、数组 数组【】的长度是固定的常量 ，数组不能修改长度，len和cap固定

```
package main

import (
    "fmt"

)

func main()  {
     //切片和数组区别
    //数组 数组【】的长度是固定的常量 ，数组不能修改长度，len和cap固定
     a :=[5]int{}
     fmt.Printf("len=%d,cap=%d\n",len(a),cap(a))
     //切片 []里面为空 【...】切片长度不固定 可以追加
     s := []int{} //底层引入一个数组 不能转载下 就会新引入地址 不过对切片本身来说不变
      //只要知道切片不定长就可以 arraylist
     fmt.Printf("len=%d,cap=%d\n",len(s),cap(s))

     s = append(s,1)
    fmt.Printf("len=%d,cap=%d\n",len(s),cap(s))
    s = append(s,1)
    fmt.Printf("len=%d,cap=%d\n",len(s),cap(s))
    s = append(s,1)
    fmt.Printf("len=%d,cap=%d\n",len(s),cap(s))
    s = append(s,1)
    fmt.Printf("len=%d,cap=%d\n",len(s),cap(s))
    s = append(s,1)
    fmt.Printf("len=%d,cap=%d\n",len(s),cap(s))

}
```

初始化切片的方式

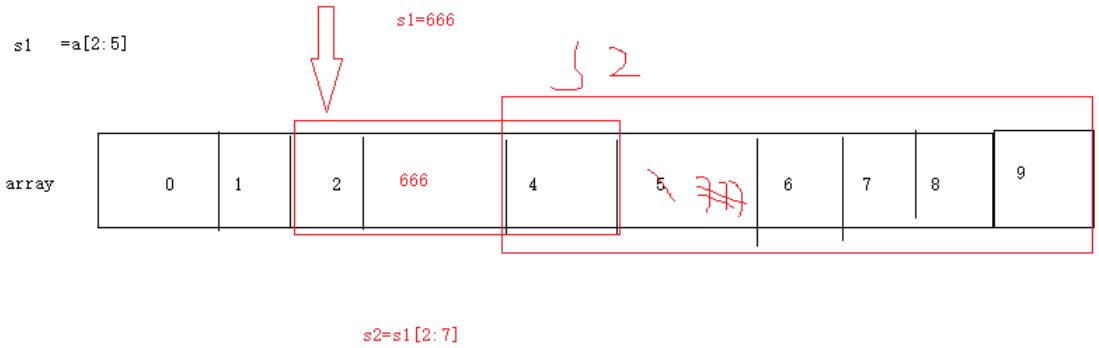
```
package main

import "fmt"

func main()  {
      //自动推导类型，同时进行初始化
     s1 := []int{1,2,3,4}
```

```
    fmt.Println("s1=",s1)

    //借助make的方式创建切片（类型 长度 容量）
    s2 := make([]int,5,10)
    fmt.Println(s2)
    fmt.Printf("len=%d,cap=%d\n",len(s2),cap(s2))

    //如果没有指定容量  容量和长度一样
   s3 := make([]int,5)
   fmt.Printf("len=%d,cap=%d\n",len(s3),cap(s3))

}
```

切片截取

```
import "fmt"

func main(){
    array :=[10]int{0,1,2,3,4,5,6,7,8,9}
    //[low:hight:max] s【1:2:4】 取下标从low开始 len =hight-low cap =max-low
    s1 := array[:]  //[0:len(array):len(array)] 容量和长度是一致
    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 cap = 7-3
    fmt.Println("s2=",s2)
    fmt.Printf("len=%d,cap=%d\n",len(s2),cap(s2))

    s3 := array[:5] //从0开始 6个元素 容量6
    fmt.Println("s3=",s3)
    fmt.Printf("len=%d,cap=%d\n",len(s3),cap(s3))

    s4 := array[3:]

    fmt.Println("s4=",s4)
    fmt.Printf("len=%d,cap=%d\n",len(s4),cap(s4))

}
```

切片和底层数组关系



对应代码

```
package main

import "fmt"

func main(){

   a := [10]int{0,1,2,3,4,5,6,7,8,9}

   s1 := a[2:5]
   s1[1]= 666
   fmt.Println("s1=",s1)
   fmt.Println("a=",a)

   s2 := s1[2:7]
   s2[1]=777
   fmt.Println("s2=",s2)
   fmt.Println("a=",a)

}
```

在切片的末尾添加元素我们用append，如果超过原理容量，通常以2倍容量扩容

```
func main()  {
    s := make([]int,0,1) // 容量1

    oldCap := cap(s)

    for i:=0;i<20;i++{
      s = append(s,i)
      if newCap:=cap(s);oldCap <newCap{
         fmt.Printf("cap :%d ====>%d\n",oldCap,newCap)
         oldCap = newCap
      }
```

```
    }

}
```

copy

```
import "fmt"

func main()  {
   srcSlice := []int{1,2}
   destSlice := []int{7,7,7,7}
   //copy(destSlice,srcSlice)
   copy(srcSlice,destSlice)
   fmt.Println("dst=",srcSlice)
}
```

切片做参数

引入类型 地址改变原内容值

```
package main

import (
   "math/rand"
   "time"
   "fmt"
)

func InitData(s []int)  {

   //设置种子
   rand.Seed(time.Now().UnixNano())

   for i:=0;i< len(s);i++{
      s[i]= rand.Intn(100) //100 以内的值
   }

}

//冒泡排序
func BubbleSort(s []int)  {
    n :=len(s)
```

```
    for i:=0;i<n-1;i++{
      for j:=0;j<n-1-i;j++{
         if s[j] > s[j+1]{
            s[j],s[j+1] = s[j+1],s[j]
         }
      }
    }
}

func main()  {
   n := 10

   //创建切片
   s :=make([]int,n)
   InitData(s)
   fmt.Println("排序前",s)
   BubbleSort(s)

   fmt.Println("排序后",s)


}
```