



大批量数据的处理问题



张华

64174234@qq.com

内容


- 1.1 大批量数据处理的现实需求
- 1.2 开发一个简单的成绩排序管理系统
- 1.3 总结
- 1.4 思考

1.1大批量数据处理的现实需求

□ 在编程过程中，涉及到很多大量数据处理问题

序号	班名	学号	姓名	我的桌面	用户管理
1	1404101	140410101	刘雨星		
2	1404101	140410102	陶明明		
3	1404101	140410103	宋文杰		
4	1404101	140410104	王玉		
5	1404101	140410105	葛嘉莉		
6	1404101	140410106	王浩		
7	1404101	140410107	李志鹏		
8	1404101	140410108	林凯		
9	1404101	140410109	刘家威		
10	1404101	140410110	郭丰瑞		
11	1404101	140410111	李荣东		
12	1404101	140410112	蒋之凯		
13	1404101	140410113	吴卓		
14	1404101	140410114	张韬		
15	1404101	140410115	仇单宁		
16	1404101	140410116	岳旭东		
17	1404101	140410117	黄摆		
18	1404101	140410118	陈天阳		
19	1404101	140410119	冯钰鈰		
20	1404101	140410120	杨发淇		

分配角色	新增	删除	重置密码	批量分配	导出
登陆ID	用户编码	用户名称	所属组别		
<input type="checkbox"/> admin	admin	admin	办公室		
<input type="checkbox"/> xg-wangzm	0205002	王争明	技术开发		
<input type="checkbox"/> xg-rangh	0103001	冉高华	国际业务		
<input type="checkbox"/> xg-yus	0203002	于水	物资采购		
<input type="checkbox"/> xg-guzp	0101002	谷展鹏	营销一部		
<input type="checkbox"/> xg-lis	0202001	李胜	财务部		
<input type="checkbox"/> xg-wuck	0204003	武传坤	生产部		
<input type="checkbox"/> xg-songsh	0203005	宋珊珊	物资采购		
<input type="checkbox"/> xg-guozh	0204007	郭志慧	生产部		
<input type="checkbox"/> xg-fancm	0204005	樊昌明	生产部		
<input type="checkbox"/> xg-fanwj	0204006	范文杰	生产部		
<input type="checkbox"/> xg-zoucc	00010101	邹翠翠	营销一部		
<input type="checkbox"/> xg-zhangjh	0202004	张嘉慧	财务部		
<input type="checkbox"/> xg-biyr	0202003	毕玉荣	财务部		
<input type="checkbox"/> xg-dingwj	0205005	丁文郡	技术开发		
<input type="checkbox"/> xg-qinz	0202005	秦珍	财务部		
<input type="checkbox"/> xg-yinmh	0206005	殷美红	质量检验		
<input type="checkbox"/> xg-lin	0208001	李年	科研开发		
<input type="checkbox"/> xg-liuyh	0205006	刘永红	技术开发		
<input type="checkbox"/> xg-wangzg	0205003	王振国	技术开发		
<input type="checkbox"/> xg-wangtl	0206001	王同亮	质量检验		
<input type="checkbox"/> xg-miaohw	0201004	苗会文	办公室		



- 每种数据都具有相同的数据类型，数量也很大
- 编程时，如果数据的数量为10个，我们要分别定义10个变量吗？100个呢？1000个呢？更多呢？
- 解决方案：Go的数组与切片可以解决这些问题

1.2开发一个简单的成绩排序管理系统

- 一个班级有30个学生学习Go语言，请完成：
 - 录入每个学生的Go语言成绩
 - 对学生成绩按照从大到小的顺序进行冒泡排序
 - 输出排序后的结果
- 量化
 - `var score [30]int` //成绩数组，存放学生成绩
- 算法
 - 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
 - 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
 - 针对所有的元素重复以上的步骤，除了最后一个。
 - 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

1.2开发一个简单的成绩排序管理系统(续)

□ 实现1——非模块化

```
1 package main
2 import "fmt"
3 func main() {
4     const n int = 30           //成绩数组的长度
5     var score [n]int           //定义成绩数组，数组长度为常量n的值
6     for i := 0; i < n; i++ { //录入数据
7         fmt.Printf("input a int score to score[%v]\n", i)
8         fmt.Scanln(&score[i])
9     }
10    fmt.Println("before sort:", score) //显示录入的数据
11    for i := 0; i < n-1; i++ {          //比较n-1轮
12        for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次
13            if score[j] < score[j+1] {
14                temp := score[j]
15                score[j] = score[j+1]
16                score[j+1] = temp
17            }
18        }
19    }
20    fmt.Println("after sort:", score) //输出排序后的结果
21 }
```

1.2开发一个简单的成绩排序管理系统(续)

□ 实现2——模块化：数组名作为函数参数-代码全貌

```
1 package main
2 import "fmt"
3 func main() {
4     const n int = 30 //成绩数组的长度
5     var score [n]int //定义成绩数组，数组长度为常量n的值
6     Input(score)
7     fmt.Println("before sort:", score) //显示录入的数据
8     sort(score)
9     fmt.Println("after sort:", score) //输出排序后的结果
10 }
11 func Input(scorearr [30]int) { //录入数据,长度要和主调函数中的score一致。
12     n := len(scorearr) //计算数组的长度
13     for i := 0; i < n; i++ {
14         fmt.Printf("input a int score to scorearr[%v]\n", i)
15         fmt.Scanln(&scorearr[i])
16     }
17 }
18 func sort(scorearr [30]int) {
19     n := len(scorearr) //计算数组的长度
20     for i := 0; i < n-1; i++ { //比较n-1轮
21         for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次
22             if scorearr[j] < scorearr[j+1] {
23                 temp := scorearr[j]
24                 scorearr[j] = scorearr[j+1]
25                 scorearr[j+1] = temp
26             }
27         }
28     }
29 }
```

1.2开发一个简单的成绩排序管理系统(续)

□ 实现2——模块化：数组名作为函数参数-录入

//录入数据,长度要和主调函数中的score一致。

```
func Input(scorearr [30]int) {  
    n := len(scorearr) //计算数组的长度  
    for i := 0; i < n; i++ {  
        fmt.Printf("input a int score to scorearr[%v]\n", i)  
        fmt.Scanln(&scorearr[i])  
    }  
}
```

1.2开发一个简单的成绩排序管理系统(续)

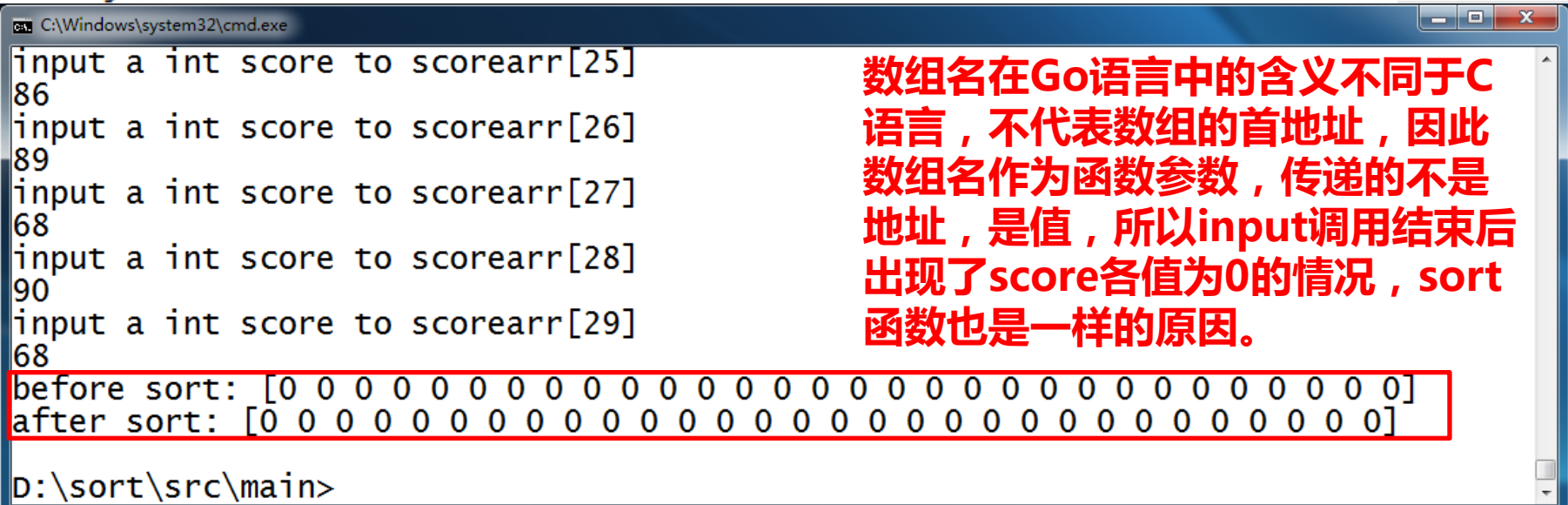
□ 实现2——模块化：数组名作为函数参数-排序

```
20 func sort(scorearr [30]int) {  
21     n := len(scorearr)           //计算数组的长度  
22     for i := 0; i < n-1; i++ { //比较n-1轮  
23         for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次  
24             if scorearr[j] < scorearr[j+1] {  
25                 temp := scorearr[j]  
26                 scorearr[j] = scorearr[j+1]  
27                 scorearr[j+1] = temp  
28             }  
29         }  
30     }  
31 }  
32
```


1.2开发一个简单的成绩排序管理系统(续)

□ 实现2——模块化：数组名作为函数参数-主函数

```
1 package main
2 import "fmt"
3 func main() {
4     const n int = 30 //成绩数组的长度
5     var score [n]int //定义成绩数组，数组长度为常量n的值
6     Input(score)
7     fmt.Println("before sort:", score) //显示录入的数据
8     sort(score)
9     fmt.Println("after sort:", score) //输出排序后的结果
10 }
```



input a int score to scorearr[25]
86
input a int score to scorearr[26]
89
input a int score to scorearr[27]
68
input a int score to scorearr[28]
90
input a int score to scorearr[29]
68
before sort: [0 0]
after sort: [0 0]
D:\sort\src\main>

数组名在Go语言中的含义不同于C语言，不代表数组的首地址，因此数组名作为函数参数，传递的不是地址，是值，所以input调用结束后出现了score各值为0的情况，sort函数也是一样的原因。

1.2开发一个简单的成绩排序管理系统(续)

□ 实现3——数组作为函数参数和返回值-录入

//参数与返回值的长度要和主调函数中的score一致。

```
func Input(scorearr [30]int) [30]int {  
    n := len(scorearr) //计算数组的长度  
    for i := 0; i < n; i++ {  
        fmt.Printf("input a int score to scorearr[%v]\n", i)  
        fmt.Scanln(&scorearr[i])  
    }  
    return scorearr  
}
```

1.2开发一个简单的成绩排序管理系统(续)

□ 实现3——数组作为函数参数和返回值-排序

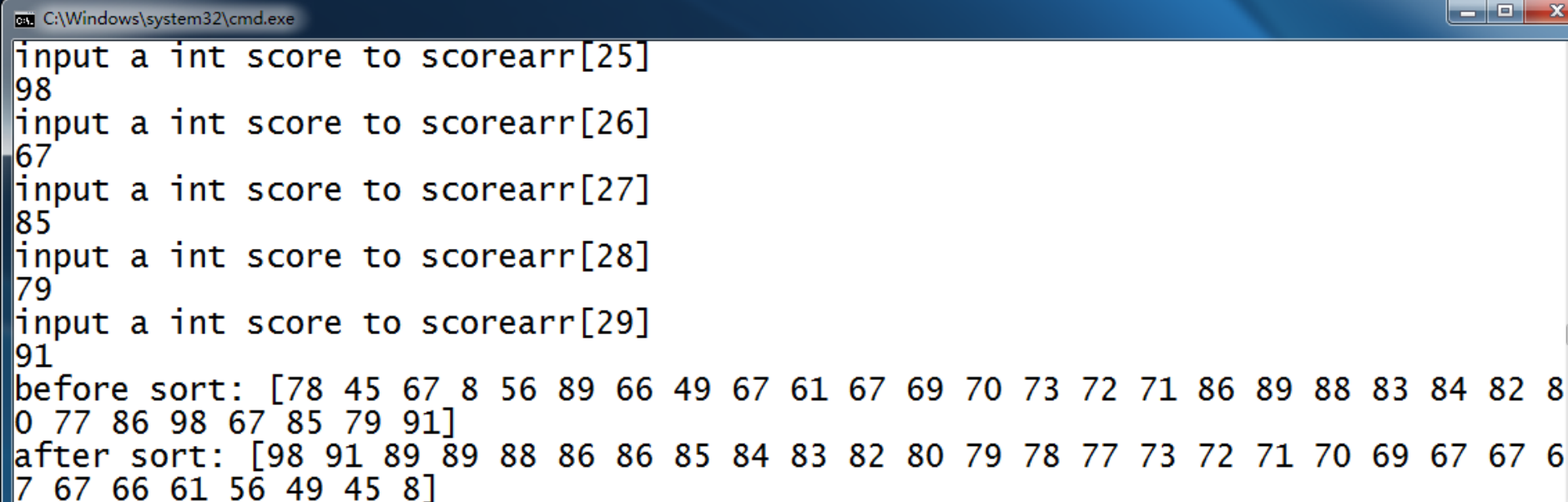
//参数与返回值的长度要和主调函数中的score一致。

```
func sort(scorearr [30]int) [30]int {  
    n := len(scorearr)           //计算数组的长度  
    for i := 0; i < n-1; i++ { //比较n-1轮  
        for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次  
            if scorearr[j] < scorearr[j+1] {  
                temp := scorearr[j]  
                scorearr[j] = scorearr[j+1]  
                scorearr[j+1] = temp  
            }  
        }  
    }  
    return scorearr  
}
```

1.2开发一个简单的成绩排序管理系统(续)

□ 实现3——数组作为函数参数和返回值-主函数

```
1 package main
2 import "fmt"
3 func main() {
4     const n int = 30           //成绩数组的长度
5     var score [n]int           //定义成绩数组，数组长度为常量n的值
6     score = Input(score)       //以score作为参数，结果再返回给score
7     fmt.Println("before sort:", score) //显示录入的数据
8     score = sort(score)        //以score作为参数，结果再返回给score
9     fmt.Println("after sort:", score) //输出排序后的结果
10 }
```



```
C:\Windows\system32\cmd.exe
input a int score to scorearr[25]
98
input a int score to scorearr[26]
67
input a int score to scorearr[27]
85
input a int score to scorearr[28]
79
input a int score to scorearr[29]
91
before sort: [78 45 67 8 56 89 66 49 67 61 67 69 70 73 72 71 86 89 88 83 84 82 8
0 77 86 98 67 85 79 91]
after sort: [98 91 89 89 88 86 86 85 84 83 82 80 79 78 77 73 72 71 70 69 67 67 6
7 67 66 61 56 49 45 8]
```

1.2开发一个简单的成绩排序管理系统(续)

□ 实现3——数组作为函数参数和返回值-代码全貌

```
1 package main
2 import "fmt"
3 func main() {
4     const n int = 30 //成绩数组的长度
5     var score [n]int //定义成绩数组，数组长度为常量n的值
6     score = Input(score) //以score作为参数，结果再返回给score
7     fmt.Println("before sort:", score) //显示录入的数据
8     score = sort(score) //以score作为参数，结果再返回给score
9     fmt.Println("after sort:", score) //输出排序后的结果
10 }
11 //参数与返回值的长度要和主调函数中的score一致。
12 func Input(scorearr [30]int) [30]int {
13     n := len(scorearr) //计算数组的长度
14     for i := 0; i < n; i++ {
15         fmt.Printf("input a int score to scorearr [%v]\n", i)
16         fmt.Scanln(&scorearr[i])
17     }
18     return scorearr
19 }
20 //参数与返回值的长度要和主调函数中的score一致。
21 func sort(scorearr [30]int) [30]int {
22     n := len(scorearr) //计算数组的长度
23     for i := 0; i < n-1; i++ { //比较n-1轮
24         for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次
25             if scorearr[j] < scorearr[j+1] {
26                 temp := scorearr[j]
27                 scorearr[j] = scorearr[j+1]
28                 scorearr[j+1] = temp
29             }
30         }
31     }
32     return scorearr
33 }
```

数组作为函数参数和返回值，在定义函数的时候，参数数组和返回数组必须和主调函数中的实参数组的类型及长度保持一致，同时也不能动态增减元素。有无更好的方案？——采用切片的方式来解决

1.2开发一个简单的成绩排序管理系统(续)

□ 实现4——采用切片作为数据结构-代码全貌

```
1 package main
2 import "fmt"
3 func main() {
4     var scoreSlice = make([]int, 0, 5) //定义成绩切片, 元素个数为0, 容量为5
5     scoreSlice = Input(scoreSlice)    //录入
6     fmt.Println("before sort:", scoreSlice) //显示录入的数据
7     Sort(scoreSlice)                  //排序
8     fmt.Println("after sort:", scoreSlice) //输出排序后的结果
9 }
10 func Input(scoresl []int) []int {
11     n := 0 //学生数量, 初始为0
12     temp := 0 //存储成绩的临时变量, 辅助append用
13     fmt.Println("input the num of students")
14     fmt.Scanln(&n) //录入元素的个数
15     for i := 0; i < n; i++ {
16         fmt.Printf("input a int score to scoresl[%v]\n", i)
17         fmt.Scanln(&temp) //录入当前成绩到给temp
18         scoresl = append(scoresl, temp) //将在切片的最后追加元素temp
19     }
20     return scoresl //当元素个数超过切片容量的时候, 就会空间再分配, 一定要返回新地址
21 }
22 func Sort(scoresl []int) {
23     n := len(scoresl) //计算切片元素的个数
24     for i := 0; i < n-1; i++ { //比较n-1轮
25         for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次
26             if scoresl[j] < scoresl[j+1] {
27                 temp := scoresl[j]
28                 scoresl[j] = scoresl[j+1]
29                 scoresl[j+1] = temp
30             }
31         }
32     }
33 }
```

1.2开发一个简单的成绩排序管理系统(续)

□ 实现4——采用切片作为数据结构-主函数

```
1 package main
2 import "fmt"
3 func main() {
4     var scoreSlice = make([]int, 0, 5)           //定义成绩切片，元素个数为0，容量为5
5     scoreSlice = Input(scoreSlice)              //录入
6     fmt.Println("before sort:", scoreSlice)     //显示录入的数据
7     Sort(scoreSlice)                            //排序
8     fmt.Println("after sort:", scoreSlice)      //输出排序后的结果
9 }
```

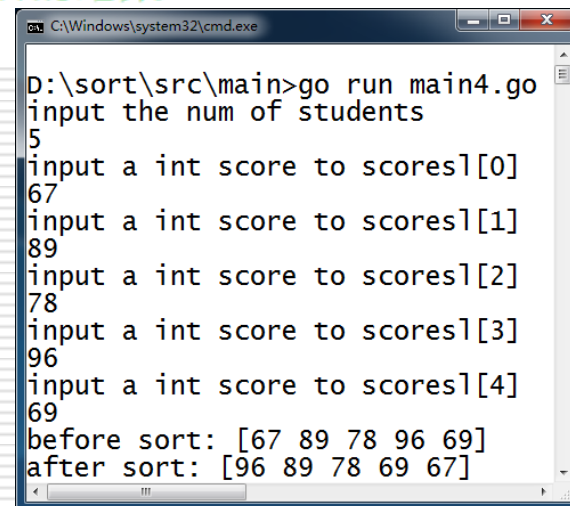
■ 使用内置函数make直接创建切片

□ var scoreSlice = make([]int, 0, 5)

■ 定义切片

□ var scoreSlice = make([]int, 0, 5){1,2,3,4,5}

■ 定义并初始化



```
C:\Windows\system32\cmd.exe
D:\sort\src\main>go run main4.go
input the num of students
5
input a int score to scores[0]
67
input a int score to scores[1]
89
input a int score to scores[2]
78
input a int score to scores[3]
96
input a int score to scores[4]
69
before sort: [67 89 78 96 69]
after sort: [96 89 78 69 67]
```


1.2开发一个简单的成绩排序管理系统(续)

□ 实现4——采用切片作为数据结构-录入

```
10 func Input(scores1 []int) []int {
11     n := 0    //学生数量, 初始为0
12     temp := 0 //存储成绩的临时变量, 辅助append用
13     fmt.Println("input the num of students")
14     fmt.Scanln(&n) //录入元素的个数
15     for i := 0; i < n; i++ {
16         fmt.Printf("input a int score to scores1[%v]\n", i)
17         fmt.Scanln(&temp) //录入当前成绩到temp
18         scores1 = append(scores1, temp) //将在切片的最后追加元素temp
19     }
20     return scores1 //当元素个数超过切片容量的时候, 就会空间再分配, 一定要返回新地址
21 }
```

- func Input(scores1 []int) []int{...} 这种中括号[]中没有值的情况都表示切片, 不是数组
- 批量切片元素的录入 或更改, 可采用append方法
 - scores1 = append(scores1, temp) 表示在切片的最后追加元素temp,超出切片的容量时, 切片的容量将自动翻倍
 - append参数可以有多个, 第一个是切片名, 其他参数为追加的元素列表, 各元素间用逗号间隔

1.2开发一个简单的成绩排序管理系统(续)

□ 实现4——采用切片作为数据结构-排序

- 排序不改变切片的内容，因此无需返回值

```
22 func Sort(scores1 []int) {  
23     n := len(scores1)           //计算切片元素的个数  
24     for i := 0; i < n-1; i++ { //比较n-1轮  
25         for j := 0; j < n-i-1; j++ { //每一轮比较n-i-1次  
26             if scores1[j] < scores1[j+1] {  
27                 temp := scores1[j]  
28                 scores1[j] = scores1[j+1]  
29                 scores1[j+1] = temp  
30             }  
31         }  
32     }  
33 }
```

1.3总结

□ 数组——是一组相同数据类型的结合，是值类型

■ 定义方式及初始化

- `arr:=[5]int{1,2,3,4,5}`定义5个元素的数组，并全部初始化
- `arr:=[...]int{1,2,3,4,5}`如果不指定长度，中括号内放的...不能省略，根据初始化值自动计算元素个数，这里为5
- `arr:=[5]int{1,2}`定义5个元素的数组，对前两个初始化为1,2
- `arr:=[5]int{2:1,3:2,4:3}`定义5个元素的数组，将下标为2,3,4的元素的值分别初始化为1,2,3
- `arr:=[...]int{2:1,4:3}` 由于指定了最大索引4对应的值为3，据此可确定数组的长度为5.

■ 数组元素访问方式——通过下标，下标从0开始

```
arr := [5]int{5, 4, 3}
for index, value := range arr { //带range关键字的循环结构,可以得到元素在集合中的索引和值
    fmt.Println("arr[%d]=%d\n", index, value)
}
```

- 数组一经定义，长度固定，无法修改长度
- 数组名作为函数参数的时候，传递的是值，不是地址
- 通过`len(数组名)`获取长度即数组中元素的个数，

1.3总结(续)

□ 切片——长度可变数组，引用类型

■ 定义及初始化

□ 用内置函数make直接定义 `var s = make([]int,5,10)`

■ 5代表元素个数，10代表容量即目前容纳的最多元素个数
`s:=[]int{1,2,3}`//[]表示切片类型，初始值为1,2,3.

□ `s:=arr[sIndex,snum]` 将arr中从下标sIndex开始的snum个元素创建为一个新切片s

□ `s:=arr[sIndex:]`将arr中从下标sIndex开始的全部元素创建为一个新切片s

□ `s:=arr[:snum]`将arr中从第一个元素开始的snum个元素创建一个新切片s

□ `s1:=s[sIndex,snum]`将切片s中从下标sIndex开始的snum个元素创建为一个新切片s1

■ 通过len(切片名)获取长度即元素个数，

■ 通过cap(切片名)获取切片的容量

■ 通过append追加1个或多个元素到切片中

□ `s:=append(s,1,2,3,4)`//追加4个元素在切片s的后面，并返回新切片地址给s，如果追加的时候，超过当前容量，s的容量将自动翻倍，内存地址也可能改变，因此要返回新的地址给s

□ `s:=append(s,s1...)`//将切片s1追加到s后面，...不能省略

1.3总结(续)

- ❑ 数组作为函数参数的时候传递的是传递数组个元素的值给形参数组各元素，形参和实参在类型和元素个数上必须保持一致
- ❑ 切片作为函数参数的时候传递的是地址给形参切片，即形参和实参共用内存空间
- ❑ 切片作为函数参数的时候，如果函数内部对切片的容量有改变，可能因为增加切片的元素而扩容，从而改变切片的地址，因此函数的返回值必须是切片，保证实参和形参的一致性。如果函数对切片的容量没有改变，则不需要设定返回值。
- ❑ 用make函数建立切片的时候，习惯上将第二个参数即元素个数设置为0，以降低对切片元素录入代码的编写复杂度

1.3总结(续)

- 内置函数`copy(s1,s2)`实现将s2的内容拷贝到s1中
 - 用于将内容从一个数组切片复制到另一个数组切片。如果加入的两个数组切片不一样大，就会按其中较小的那个数组切片的元素个数进行复制。下面的示例展示了`copy()`函数的行为：
 - `slice1 := []int{1, 2, 3, 4, 5}`
 - `slice2 := []int{5, 4, 3}`
 - `copy(slice2, slice1)` // 只会复制slice1的前3个元素到slice2中
 - `copy(slice1, slice2)` // 只会复制slice2的3个元素到slice1的前3个位置

1.4思考

□ 实现方式4中

- 如果将切片的定义改为
- `var scoreSlice = make([]int, 5, 5)`
- 请运行程序，分别以学生数量为5,6,7为值，观察程序的运行结果，并给出原因。

□ 一个班级有30个学生学习Go语言，每个学生包括姓名、学号、成绩三个属性，请编程

- 录入每个学生的姓名，学号及Go语言成绩
- 对学生成绩按照从大到小的顺序进行冒泡排序
- 输出排序后的成绩单

1.4思考

□ 写出下面这段代码的结果

```
1 package main
2 import "fmt"
3 func main() {
4     s1 := make([]int, 5, 10)
5     s2 := make([]int, 3, 10)
6     for i, _ := range s1 {
7         s1[i] = i
8     }
9     fmt.Println("before s1", s1)
10    for i, _ := range s2 {
11        s2[i] = i + 2
12    }
13    copy(s1, s2)
14    fmt.Println("afeter s1", s1)
15 }
```

Thank you very much

*Any comments and suggestions
are beyond welcome*