切片Slice

- 其本身并不是数组，它指向底层的数组
- 作为变长数组的替代方案，可以关联底层数组的局部或全部
- 为引用类型
- 可以直接创建或从底层数组获取生成
- 使用len()获取元素个数，cap()获取容量
- 一般使用make()创建
- 如果多个slice指向相同底层数组，其中一个的值改变会影响全部

- make([]T, len, cap)
- 其中cap可以省略，则和len的值相同
- len表示存数的元素个数，cap表示容量

创建一个新的slice

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      var s1 []int
9.      fmt.Println(s1)
10. }
```

```
成功: 进程退出代码 0.
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[]
成功: 进程退出代码 0.
```

使用这种方法来声明一个slice，说明是一个空的slice

创建一个新的数组，slice使用这个数组里面的值：

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      a := [10]int{}
9.      fmt.Println(a)
10.     s1 := a[9]
11.     fmt.Println(s1)
```

```
12. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
[0 0 0 0 0 0 0 0 0 0]
0
成功: 进程退出代码 0.
```

从这个数组当中截取一部分，创建一个slice，slice后面只放一个数，那就截取这个数组当中索引为9的这个元素。

如果要取多个元素怎么办？

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      a := [10]int{}
9.      fmt.Println(a)
10.     s1 := a[5:10] // a[5 6 7 8 9]
11.     fmt.Println(s1)
12. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0]
成功: 进程退出代码 0.
```

`s1 := a[5:10]`

取索引5，后五位元素。那就是索引5，冒号就是到，几到几，这里就是索引为5到几到几的意思。为什么这里是10呢？

使用这种语法，它是包含起始索引，不包含终止索引，所以说 这里实际上相当于使用了数组的a[5 6 7 8 9]，它不会包含索引为10。

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      a := [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9}
9.      fmt.Println(a)
10.     s1 := a[5:10] // a[5 6 7 8 9]
11.     fmt.Println(s1)
12. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6 7 8 9 0]
[6 7 8 9 0]
成功: 进程退出代码 0.
```

如果我不知道这个数组有多少个元素，我只是知道它一定会有5个元素，那该怎么办呢？有俩种方法：
第一种，可以使用**len()**内置函数，来取得这个数组的长度

```go
package main

import (
    "fmt"
)

func main() {
    a := [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9}
    fmt.Println(a)
    s1 := a[5:len(a)] // a[5 6 7 8 9]
    fmt.Println(s1)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6 7 8 9 0]
[6 7 8 9 0]
成功: 进程退出代码 0.
```

第二种就是直接将冒号的数字s1 := a[5:10]去掉s1 := a[5:] 也就是说 索引为第5个元素一直这个数组的尾部，就把从第5索引为5的开始，把元素全部取光

如果我想取前5个元素：

```go
package main

import (
    "fmt"
)

func main() {
    a := [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9}
    fmt.Println(a)
    s1 := a[:5] // a[5 6 7 8 9]
    fmt.Println(s1)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6 7 8 9 0]
[1 2 3 4 5]
成功: 进程退出代码 0.
```

这里取出来的元素不会包含索引为5的元素，也就是说不会包含第六个元素，这也就是一定有5个元素，只想要前5个元素，那就可以省略前面的`s1 := a[0:5]`前面的零`s1 := a[:5]` 直接 :5

slice的另外一种声明方法：

s1 := make([]int, 3, 10)

可以使用**make()**函数，用这种方法比较正式，可以指定slice的完整的属性。

让slice保存为int，第一个参数就要放它的类型，第二个参数就是我现在这个slice它包含多少个元素，比如我想让这个slice包含3个元素，那么它就是初始化3个int，然后放到这个slice里所指向的数组当中。第三个参数叫做slice的容量。那么因为slice指向是一个底层的数组，而且我们可以把它当作一个变长数组解决方案，所以说slice它的一个长度从感觉上来讲经常在改变的。那么在改变的时候，如果说数组底层 用一块连续使用的内存，所以说就导致了很难去改变它的一个长度，那么如果我们使用不断一个指针指向不通位置来达到一个动态数组效果的话，那么显然就会不断的分配新的数组指向某一块连续的内存，所以说第三个参数是slice的一个初始容量。比如说初始容量是10，它先分给我10小块连续的内存，那么这样子如果我的元素一直都在10个之内，尽管我现在只有3个元素，但是我一直涨到8个9个10个的时候呢，它都不需要重新分配内存地址，就是一直连续使用那一块内存地址，因为我的长度没有超标，但是如果说我想要给它第11个元素的时候，显然它这块连续的内存没有办法在继续使用了，因为长度只有10，而我想要存放11个元素，这个时候它就会得到重新分配长度为11的，实际上它在go语言当中，它是每次它都会增加一倍，会变成20，这个时候如果说充满了10个元素，如果在往slice里面加元素的话呢，它会自动扩容为20，然后会重新给我分配一块长度为20的连续的内存块，元素在一直加，14 17 18，它的第三个容量值 不会改变了，它只会改变一次。当我加到22个元素的时候，显然内存块又不够长，它又会增加一倍，它就会分配一块长度为40的连续的内存块。那么为什么这样做呢？因为重新分片内存地址是比较效率低的一件事，如果事先知道这个slice可能会用到多少元素的时候呢，先告诉编译器，这样在某些方面提升程序的效率。如果不是很关心效率呢，那就从1开始，它就会不断的重新分片，这样就会显示效率很低。

```go
package main

import (
    "fmt"
)

func main() {
    s1 := make([]int, 3, 10)
    fmt.Println(len(s1), cap(s1))
}
```

```
成功: 进程退出代码 0.
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
3 10
成功: 进程退出代码 0.
```

打印出s1的slice的个数，使用len()函数。使用**cap()**函数 打印s1的slice的容量

如果说不设置slice的最大容量，可不可以呢？

```go
package main

import (
    "fmt"
)

func main() {
    s1 := make([]int, 3)
    fmt.Println(len(s1), cap(s1))
}
```

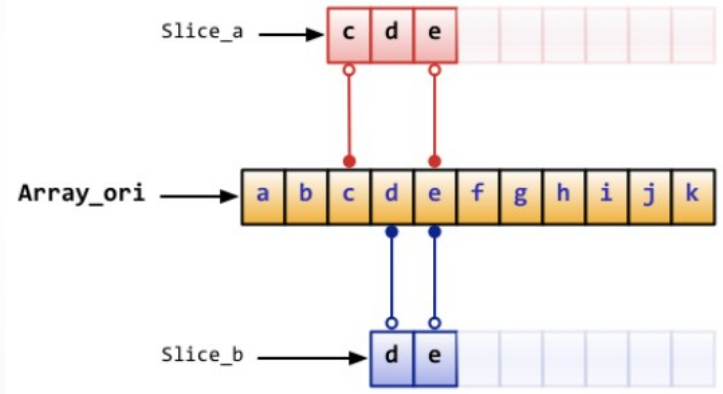
```
成功: 进程退出代码 0.
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
3 3
成功: 进程退出代码 0.
```

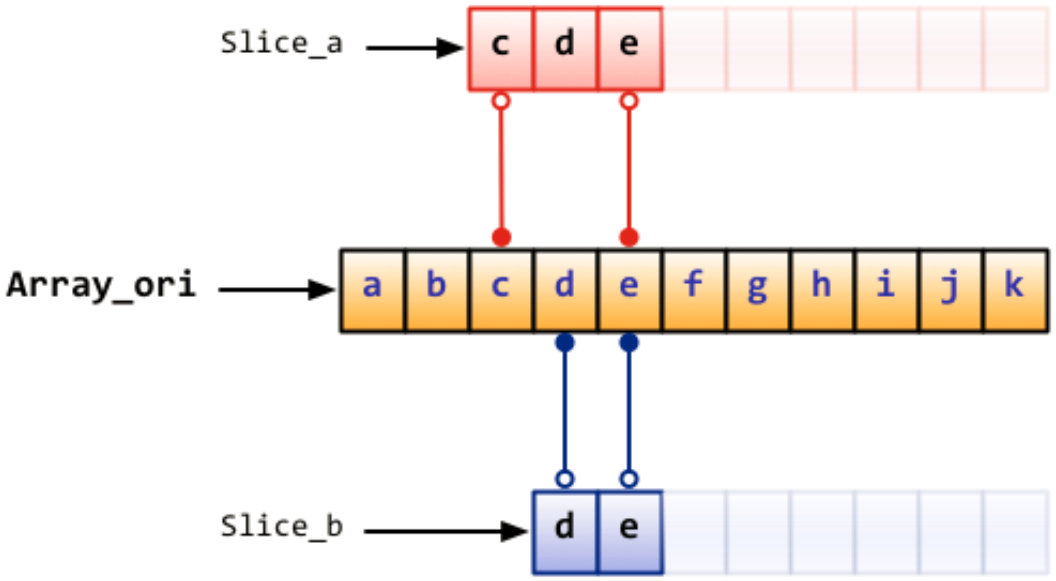
可是可以的，slice它会认为最大容量就是slice元素最大长度
打印s1的slice的数值是什么：

```go
package main

import (
    "fmt"
)

func main() {
    s1 := make([]int, 3)
    fmt.Println(s1)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[0 0 0]
成功: 进程退出代码 0.
```

Slice与底层数组的对应关系





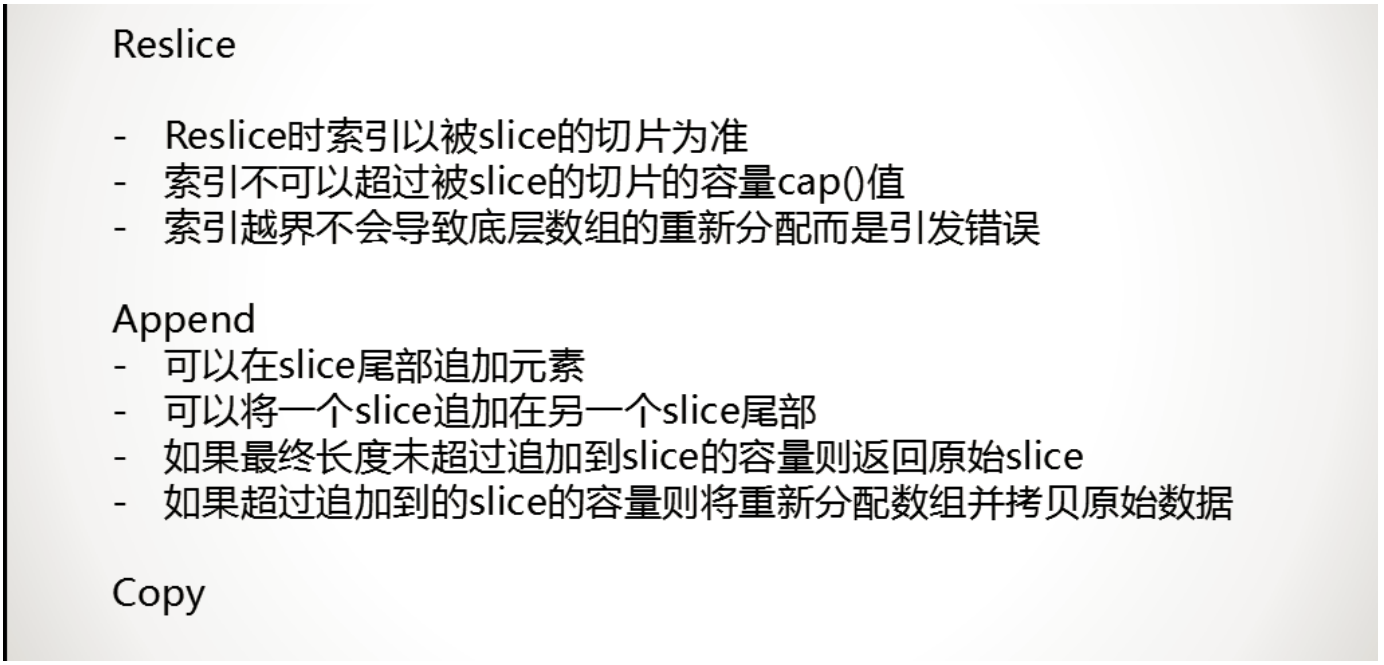

```go
package main

import (
    "fmt"
)

func main() {
    a := []byte{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'}
    sa := a[2:5]
    fmt.Println(string(sa))
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
cde
成功: 进程退出代码 0.
```

```go
package main
```

```
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      a := []byte{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'}
9.      sb := a[3:5]
10.     fmt.Println(string(sb))
11. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
de
成功: 进程退出代码 0.
```



reslice 就是从一个slice当然获取一个新的slice

- Reslice时索引以被slice的切片为准
- 索引不可以超过被slice的切片的容量cap()值
- 索引越界不会导致底层数组的重新分配而是引发错误

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      a := []byte{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'}
9.      sa := a[2:5]
10.     sb := sa[1:3]
```

```
    fmt.Println(string(sb))
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
de
成功: 进程退出代码 0.
```

这就是reslice。sa从数组总取得 def。sb是一个reslice，从sa的slice中值取得de。

```
package main

import (
    "fmt"
)

func main() {
    a := []byte{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'}
    sa := a[2:5]
    sb := sa[3:5]
    fmt.Println(string(sb))
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
fg
成功: 进程退出代码 0.
```

为啥会出现 fg 结果呢？ 因为slice指向的数组是一个连续的内存块容量，所以收它的最大内存容量极限就是这个数组的尾部。这里数组有是11个元素，它所配分的最大容量就是9，同样，sb的容量比sa的容量少一个，是8个。

```
package main

import (
    "fmt"
)

func main() {
    a := []byte{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'}
    sa := a[2:5]
    fmt.Println(len(sa), cap(sa))
    sb := sa[3:5]
    fmt.Println(string(sb))
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
3 9
fg
成功: 进程退出代码 0.
```

```
package main
```

```go

import (
    "fmt"
)

func main() {
    a := []byte{'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'}
    sa := a[2:5]
    sb := sa[9:11]
    fmt.Println(string(sb))
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
panic: runtime error: slice bounds out of range

goroutine 1 [running]:
panic(0x47a920, 0xc42000a180)
        /usr/local/go/src/runtime/panic.go:500 +0x1a1
main.main()
        /home/jiemin/code/GOlang/go/src/slice/slice.go:10 +0x3a
错误: 进程退出代码 2.
```

意思就是索引不可以超过被slice的切片的容量cap()值，所以索引越界了

append()函数。

它是slice最重要的函数

- 可以在slice尾部追加元素
- 可以将一个slice追加在另一个slice尾部
- 如果最终长度未超过追加到slice的容量则返回原始slice
- 如果超过追加到的slice的容量则将重新分配数组并拷贝原始数据

使用**append()**函数

第一个参数就是被追加的slice

第二个参数开始就是追加进去的元素

```go
package main

import (
    "fmt"
)

func main() {
    s1 := make([]int, 3, 6)
    fmt.Printf("%p\n", s1)
    s1 = append(s1, 1, 2, 3)
    fmt.Printf("%v %p\n", s1, s1)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
0xc4200122a0
[0 0 0 1 2 3] 0xc4200122a0
成功: 进程退出代码 0.
```

第一个打印的内存地址和第二个内存地址是相同的。

在追加三个元素

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      s1 := make([]int, 3, 6)
9.      fmt.Printf("%p\n", s1)
10.     s1 = append(s1, 1, 2, 3)
11.     fmt.Printf("%v %p\n", s1, s1)
12.     s1 = append(s1, 4, 5, 6)
13.     fmt.Printf("%v %p\n", s1, s1)
14. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
0xc4200122a0
[0 0 0 1 2 3] 0xc4200122a0
[0 0 0 1 2 3 4 5 6] 0xc42006e060
成功: 进程退出代码 0.
```

追加完成之后，元素的个数小于或者等于slice的容量，内存地址不会改变，会返回原始的slice。如果追加之后，元素的个数大于这个slice的容量，它会重新分片一个内存地址给你，然后将原始的数据拷贝过去之后，然后在追加你想要追加的元素。

假设我们有一个数组，包含 1 2 3 4 5

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      a := []int{1, 2, 3, 4, 5}
9.      s1 := a[2:5]
10.     s2 := a[1:3]
11.     fmt.Println(s1, s2)
12. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[3 4 5] [2 3]
成功: 进程退出代码 0.
```

将第s1的第二个元素修改为9

```go
package main

import (
    "fmt"
)

func main() {
    a := []int{1, 2, 3, 4, 5}
    s1 := a[2:5]
    s2 := a[1:3]
    fmt.Println(s1, s2)
    s1[0] = 9
    fmt.Println(s1, s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[3 4 5] [2 3]
[9 4 5] [2 9]
成功: 进程退出代码 0.
```

发现都改变为9，这就说明它是指向最底层的数组。当多个slice指向数组的时候，一个slice修改一个元素，多个slice也会改变

如果使用append()函数去追加元素，超过了最底层的数组，也就slice的容量。那么多个slice也会改变吗？答案是不会的。

```go
package main

import (
    "fmt"
)

func main() {
    a := []int{1, 2, 3, 4, 5}
    s1 := a[2:5]
    s2 := a[1:3]
    fmt.Println(s1, s2)
    s2 = append(s2, 6, 7, 8, 9, 0)
    s1[0] = 9
    fmt.Println(s1, s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[3 4 5] [2 3]
[9 4 5] [2 3 6 7 8 9 0]
成功: 进程退出代码 0.
```

当append的个数超过了slice的容量，它就会指向新的底层数组，而不再指向数组a了，s1依旧是指向数组a，所以说s1在这个时候修改了数组元素的值，就不会影响s2了。当你在使用多个slice的时候需要注意的部分。

copy() 函数
第一个参数是copy到的slice
第二个参数是被copy的slice

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      s1 := []int{1, 2, 3, 4, 5, 6}
9.      s2 := []int{7, 8, 9, 10}
10.     copy(s2, s1)
11.     fmt.Println(s1, s2)
12. }
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6] [1 2 3 4]
成功: 进程退出代码 0.
```

如果想拷贝slice1的一部分呢

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      s1 := []int{1, 2, 3, 4, 5, 6}
9.      s2 := []int{7, 8, 9, 10}
10.     copy(s2, s1[1:3])
11.     fmt.Println(s1, s2)
12. }
```

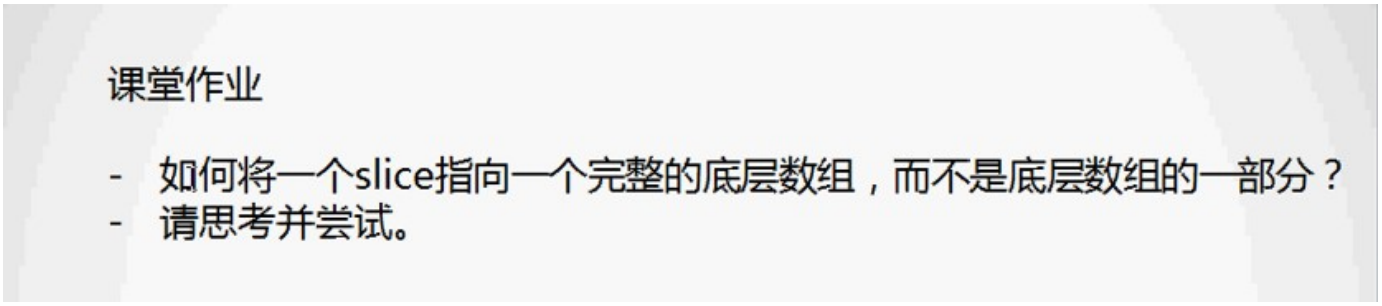
```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6] [2 3 9 10]
成功: 进程退出代码 0.
```

拷贝到slice的指定位置

```
1.  package main
2.
3.  import (
```

```
    "fmt"
)

func main() {
    s1 := []int{1, 2, 3, 4, 5, 6}
    s2 := []int{7, 8, 9, 10}
    copy(s2[2:4], s1[1:3])
    fmt.Println(s1, s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6] [7 8 2 3]
成功: 进程退出代码 0.
```



作业：

使用reslice方式s2的slice

```
package main

import (
    "fmt"
)

func main() {
    s1 := []int{1, 2, 3, 4, 5, 6}
    s2 := s1[0:6]
    fmt.Println(s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6]
成功: 进程退出代码 0.
```

或者使用简写方式有俩种

```
package main

import (
    "fmt"
)

func main() {
    s1 := []int{1, 2, 3, 4, 5, 6}
```

```
    s2 := s1[:6]
    fmt.Println(s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6]
成功: 进程退出代码 0.
```

```
package main

import (
    "fmt"
)

func main() {
    s1 := []int{1, 2, 3, 4, 5, 6}
    s2 := s1[:]
    fmt.Println(s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
[1 2 3 4 5 6]
成功: 进程退出代码 0.
```

如果要超出slice的容量界限就会错

```
package main

import (
    "fmt"
)

func main() {
    s1 := []int{1, 2, 3, 4, 5, 6}
    s2 := s1[:7]
    fmt.Println(s2)
}
```

```
/home/jiemin/code/GOlang/go/src/slice/slice  [/home/jiemin/code/GOlang/go/src/slice]
panic: runtime error: slice bounds out of range

goroutine 1 [running]:
panic(0x47a8e0, 0xc42000a180)
        /usr/local/go/src/runtime/panic.go:500 +0x1a1
main.main()
        /home/jiemin/code/GOlang/go/src/slice/slice.go:9 +0x53
错误: 进程退出代码 2.
```