问题：向slice添加元素

```
arr := [...]int{0, 1, 2, 3, 4, 5, 6, 7}
s1 := arr[2:6]
s2 := s1[3:5]
s3 := append(s2, 10)
s4 := append(s3, 11)
s5 := append(s4, 12)


* s3,s4,s5的值为?
* arr的值为?

* arr=  [0 1 2 3 4 5 6 10]
* s1=[2 3 4 5], len(s1)=4, cap(s1)=6
* s2=[5 6], len(s2)=2, cap(s2)=3
* s3=[5 6 10], len(s3)=3, cap(s3)=3
* s4=[5 6 10 11], len(s4)=4, cap(s4)=6
* s5=[5 6 10 11 12], len(s5)=5, cap(s5)=6

* 1.添加元素时如果超过cap，系统会重新分配更大的底层数组
* 2.由于值传递的关系，必须接收append的返回值(append操作可能会使slice的len和cap变大，需要新的slice接收)
* 3.s = append(s, val)
```

创建slice：

```
func printSlice(slice []int) {
   fmt.Printf("len=%d, cap=%d\n", len(slice), cap(slice))
}

// 创建slice
func createSlice() {
   var slice1 []int // Zero value for slice is nil
   for i := 0; i < 100; i++ {
      slice1 = append(slice1, 2 * i + 1)
      printSlice(slice1)
   }
   fmt.Println(slice1)
}


* 结果:
len=1, cap=1
len=2, cap=2
len=3, cap=4
len=4, cap=4
len=5, cap=8
len=6, cap=8
len=7, cap=8
len=8, cap=8
len=9, cap=16
len=10, cap=16
len=11, cap=16
len=12, cap=16
len=13, cap=16
len=14, cap=16
len=15, cap=16
len=16, cap=16
len=17, cap=32
len=18, cap=32
len=19, cap=32
len=20, cap=32
len=21, cap=32
len=22, cap=32
len=23, cap=32
len=24, cap=32
len=25, cap=32
len=26, cap=32
len=27, cap=32
len=28, cap=32
len=29, cap=32
len=30, cap=32
len=31, cap=32
len=32, cap=32
len=33, cap=64
len=34, cap=64
len=35, cap=64
len=36, cap=64
len=37, cap=64
len=38, cap=64
len=39, cap=64
len=40, cap=64
len=41, cap=64
len=42, cap=64
len=43, cap=64
len=44, cap=64
len=45, cap=64
len=46, cap=64
len=47, cap=64
len=48, cap=64
len=49, cap=64
len=50, cap=64
len=51, cap=64
len=52, cap=64
len=53, cap=64
len=54, cap=64
len=55, cap=64
len=56, cap=64
len=57, cap=64
len=58, cap=64
len=59, cap=64
len=60, cap=64
len=61, cap=64
len=62, cap=64
len=63, cap=64
len=64, cap=64
len=65, cap=128
len=66, cap=128
len=67, cap=128
len=68, cap=128
len=69, cap=128
len=70, cap=128
len=71, cap=128
len=72, cap=128
len=73, cap=128
len=74, cap=128
len=75, cap=128
len=76, cap=128
len=77, cap=128
len=78, cap=128
len=79, cap=128
len=80, cap=128
len=81, cap=128
len=82, cap=128
len=83, cap=128
len=84, cap=128
len=85, cap=128
len=86, cap=128
len=87, cap=128
len=88, cap=128
len=89, cap=128
len=90, cap=128
len=91, cap=128
len=92, cap=128
len=93, cap=128
len=94, cap=128
len=95, cap=128
len=96, cap=128
len=97, cap=128
len=98, cap=128
len=99, cap=128
len=100, cap=128
[1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99 101 103 105 107 109 111 113 115 117 119 121 123 125 127 129 131 133 135 137 139 141 143 145 147 149 151 153 155 157 159 161 163 165 167 169 171 173 175 177 179 181 183 185 187 189 191 193 195 197 199]
```

Slice创建的其他几种方式：

```
// 1.
s1 := []int{2, 4, 6, 8}

// 2.
s2 := make([]int, 16)

// 3.
s3 := make([]int, 10, 32)

func printSlice(slice []int) {
   fmt.Printf("slice=%v, len=%d, cap=%d\n", slice, len(slice), cap(slice))
}

printSlice(s1)
printSlice(s2)
printSlice(s3)

* 结果:
slice=[2 4 6 8], len=4, cap=4
slice=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], len=16, cap=16
slice=[0 0 0 0 0 0 0 0 0 0], len=10, cap=32
```

Copying slice:

```
// 1.
s1 := []int{2, 4, 6, 8}

// 2.
s2 := make([]int, 16)

printSlice(s1)
printSlice(s2)
fmt.Println("Copying slice")
copy(s2, s1)
printSlice(s2)

* 结果:
slice=[2 4 6 8], len=4, cap=4
slice=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0], len=16, cap=16
Copying slice
slice=[2 4 6 8 0 0 0 0 0 0 0 0 0 0 0 0], len=16, cap=16
```

Deleting elements from slice:

```
// 1.
s1 := []int{2, 4, 6, 8}

// 2.
s2 := make([]int, 16)
copy(s2, s1)

printSlice(s2)
// 将s2中的元素8删除
fmt.Println("Deleting elements from slice")
s2 = append(s2[:3], s2[4:]...)
printSlice(s2)

* 结果:
slice=[2 4 6 8 0 0 0 0 0 0 0 0 0 0 0 0], len=16, cap=16
Deleting elements from slice
slice=[2 4 6 0 0 0 0 0 0 0 0 0 0 0 0], len=15, cap=16
```

Popping from front:
Popping from back:

```
// 1.
s1 := []int{2, 4, 6, 8}


fmt.Println("Popping from front")
front := s1[0]
s1 = s1[1:]
fmt.Println("front =", front)
fmt.Println("s1 =", s1)


fmt.Println("Popping from back")
back := s1[len(s1) - 1]
s1 = s1[:len(s1) - 1]
fmt.Println("back =", back)
fmt.Println("s1 =", s1)

* 结果:
Popping from front
front = 2
s1 = [4 6 8]

Popping from back
back = 8
s1 = [4 6]
```