课程安排，请关注**微信公众平台**或者**官方微博**

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，

欢迎来到 **字节教育** 课程的学习

字节教育官网：www.ByteEdu.Com

腾讯课堂地址：Gopher.ke.qq.Com

技术交流群 ： 221 273 219

微信公众号 ：Golang 语言社区

微信服务号 ：Golang 技术社区

目录：

第一季　Golang 语言社区-综合面试题

第三节 slice 的使用

一、公众账号：

回复关键字：客服　　获取课程助教的微信

二、核心内容讲解：

Go 语言中的 slice 类型可以理解为是数组 array 类型的描述符，包含了三个因素：

1. 指向底层数组的指针
2. slice 目前使用到的底层数组的元素个数，即长度
3. 底层数组的最大长度，即容量

因此当我们定义一个切片变量，s := make([]int, 5, 10)，即为指向了一个最大长度为 10 的底层数组，目前切片 s 使用到的长度为 5。

基于 slice 的定义，在使用 slice 时，有以下几点注意事项：

1.对 slice 进行切分操作

对 slice 进行切分操作会生成一个新的 slice 变量，新 slice 和原来的 slice 指向同一个底层数组，只不过指向的起始位置可能不同，长度及容量可能也不相同。

- 当从左边界有截断时，会改变新切片容量大小
- 左边界默认 0，最小为 0；右边界默认 slice 的长度，最大为 slice 的容量
- 当然，因为指向同一个底层数组，对新 slice 的操作会影响到原来的 slice

示例代码 https://play.golang.org/p/3GO0cIbZ9u

2.slice 的赋值及函数间传递

```
1  a := []int{1, 2, 3, 4, 5}
2  b := a
```

示例代码 https://play.golang.org/p/lcjiulu-X2

如上所示，则 a, b 指向同一个底层数组，且长度及容量因素相同，对 b 进行的操作会影响到 a。

```
1  func main() {
2          a := []int{1, 2, 3, 4, 5}
3          modifySlice(a)
4          //[10 2 3 4 5]
5          fmt.Println(a)
6  }
7
8  func modifySlice(s []int) {
9          s[0] = 10
10 }
```

示例代码 https://play.golang.org/p/ioOXLoAz3W

如上所示，将 slice 作为参数在函数间传递的时候是值传递，产生了一个新的 slice，只不过新的 slice 仍然指向原来的底层数组，所以通过新的 slice 也能改变原来的 slice 的值

```
1  func main() {
2          a := []int{1, 2, 3, 4, 5}
3          modifySlice(a)
4          //[1 2 3 4 5]
5          fmt.Println(a)
6  }
7
8  func modifySlice(s []int) {
9          s = []int{10, 20, 30, 40, 50}
10 }
```

示例代码 https://play.golang.org/p/LbFovzP-Rj

但是，如上所示，在调用函数的内部，将 s 整体赋值一个新的 slice，并不会改变 a 的值，因为 modifySlice 函数内对 s 重新的整体赋值，让 s 指向了一个新的底层数组，而不是传递进来之前的 a 指向的那个数组，之后 s 的任何操作都不会影响到 a 了。

3.slice 的 append 操作

append 操作最容易踩坑，下面详细说明一下。

append 函数定义： func append(s []T, x ...T) []T

Append 基本原则：对于一个 slice 变量，若 slice 容量足够，append 直接在原来的底层数组上追加元素到 slice 上；如果容量不足，append 操作会生成一个新的更大容量的底层数组。

第一种情况：

```
1  func main() {
2          a := make([]int, 2, 4)
3
4          //通常 append 操作都是将返回值赋值给自己,
5          //此处为了方便说明,没有这样做
6
```

```
        b := append(a, 1)
        //改变 b 的前 2 个元素值 会影响到 a
        b[0] = 99

        //a: [99 0]       &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a, " &a[0]:", &a[0], " len:", len(a), " cap:", cap(a))
        //b: [99 0 1]   &b[0]: 0x10410020   len: 3   cap: 4
        fmt.Println("b:", b, " &b[0]:", &b[0], " len:", len(b), " cap:", cap(b))
}
```

示例代码 https://play.golang.org/p/rQQy4u0vCq

如上所示，对 a 进行 append 操作，若 append 后的新 slice 的实际元素个数没有超出原来指向的底层数组的容量，所以仍然使用原来的底层数组:a, b 的第一个值的地址一样, 改变 b 的前 2 个元素也会影响到 a。其实这时候 a,b 指向的同一个底层数组的第 3 位(索引 2)已经变成了数值 1，但是对 slice 而言，除了底层数组，还有长度，容量两个因素，这时候 a 的长度仍然是 2，所以输出的 a 的值没有变化。

第二种情况：

```
func main() {

        a := make([]int, 2, 4)

        //通常 append 操作都是将返回值赋值给自己,
        //此处为了方便说明,没有这样做
        b := append(a, 1, 2, 3)
        //改变 b 的前 2 个元素值 不会影响到 a
        b[0] = 99

        //a: [0 0]                &a[0]: 0x10410020   len: 2   cap: 4
```

```
        fmt.Println("a:", a,  " &a[0]:", &a[0],  " len:", len(a),  " cap:", cap(a))

        //b: [99 0 1 2 3]   &b[0]: 0x10454000   len: 5   cap: 8

        fmt.Println("b:", b,  " &b[0]:", &b[0],  " len:", len(b),  " cap:", cap(b))

}
```

示例代码 https://play.golang.org/p/e-gvTVx4vZ

如上所示，若 append 后的新 slice 即 b 的实际元素个数已经超出了原来的 a 指向的底层数组的容量，那么就会分配给 b 一个新的底层数组，可以看到，a，b 第一个元素的地址已经不同，改变 b 的前两个元素值也不会影响到 a，同时容量也发生了变化。

第三种情况：

```
func main() {
        a := make([]int, 2, 4)
        a[0] = 10
        a[1] = 20
        //a: [10 20]   &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a,  " &a[0]:", &a[0],  " len:", len(a),  " cap:", cap(a))

        //进行 append 操作
        b := append(a[:1], 1)

        //a: [10 1]     &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a,  " &a[0]:", &a[0],  " len:", len(a),  " cap:", cap(a))

        //b: [10 1]     &b[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("b:", b,  " &b[0]:", &b[0],  " len:", len(b),  " cap:", cap(b))
```

```
}
```

示例代码 https://play.golang.org/p/Efb42G5Wt9

如上所示，若 append 后的 b 的实际元素个数没有超过原来的 a 指向的底层数组的容量，那么 a，b 指向同一个底层数组。

注意此时 append 的操作对象是：对 a 进行切分之后的切片，只取了 a 的第一个值，相当于一个新切片，长度为 1，和 a 指向同一个底层数组，我们称这个切分后的新切片为 c 吧，那么就相当于 b 其实是基于 c 切片进行 append 的，直接在长度 1 之后追加元素，所以 append 之后 a 的第二个元素变成了 1。【所以切分操作和 append 操作放一起的时候，一定要小心】

第四种情况：

```
func main() {

        a := make([]int, 2, 4)
        a[0] = 10
        a[1] = 20
        //a: [10 20]              &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a, " &a[0]:", &a[0], " len:", len(a), " cap:", cap(a))

        //进行 append 操作
        //append 是在第一个元素后开始追加，所以要超过容量，至少要追加 4 个，而不是之前例子的 3 个
        b := append(a[:1], 1, 2, 3, 4)

        //a: [10 20]              &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a, " &a[0]:", &a[0], " len:", len(a), " cap:", cap(a))

        //b: [10 1 2 3 4]   &b[0]: 0x10454020   len: 5   cap: 8
        fmt.Println("b:", b, " &b[0]:", &b[0], " len:", len(b), "
```

```
cap:", cap(b))
}
```

示例代码 https://play.golang.org/p/wzNbO9vDJ0

如上所示，这种情况主要用来与第三种情况对比，如果 append 的元素数较多，超过了原来的容量，直接采用了新的底层数组，也就不会影响到 a 了。

上述的四种情况所用例子都比较简单，所以比较容易看清。要小心如果在函数间传递 slice，调用函数采用 append 进行操作，可能会改变原来的值的，如下所示：

```
func main() {

        a := make([]int, 2, 4)
        a[0] = 10
        a[1] = 20
        //a: [10 20]   &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a,  " &a[0]:", &a[0],  " len:", len(a),  " cap:", cap(a))

        testAppend(a[:1])

        //a: [10 1]     &a[0]: 0x10410020   len: 2   cap: 4
        fmt.Println("a:", a,  " &a[0]:", &a[0],  " len:", len(a),  " cap:", cap(a))

}

func testAppend(s []int) {
        //进行 append 操作
        s = append(s, 1)
        //s: [10 1]   &s[0]: 0x10410020   len: 2   cap: 4
```

```
        fmt.Println("s:", s, " &s[0]:", &s[0], " len:", len(s), " cap:", cap(s))
}
```