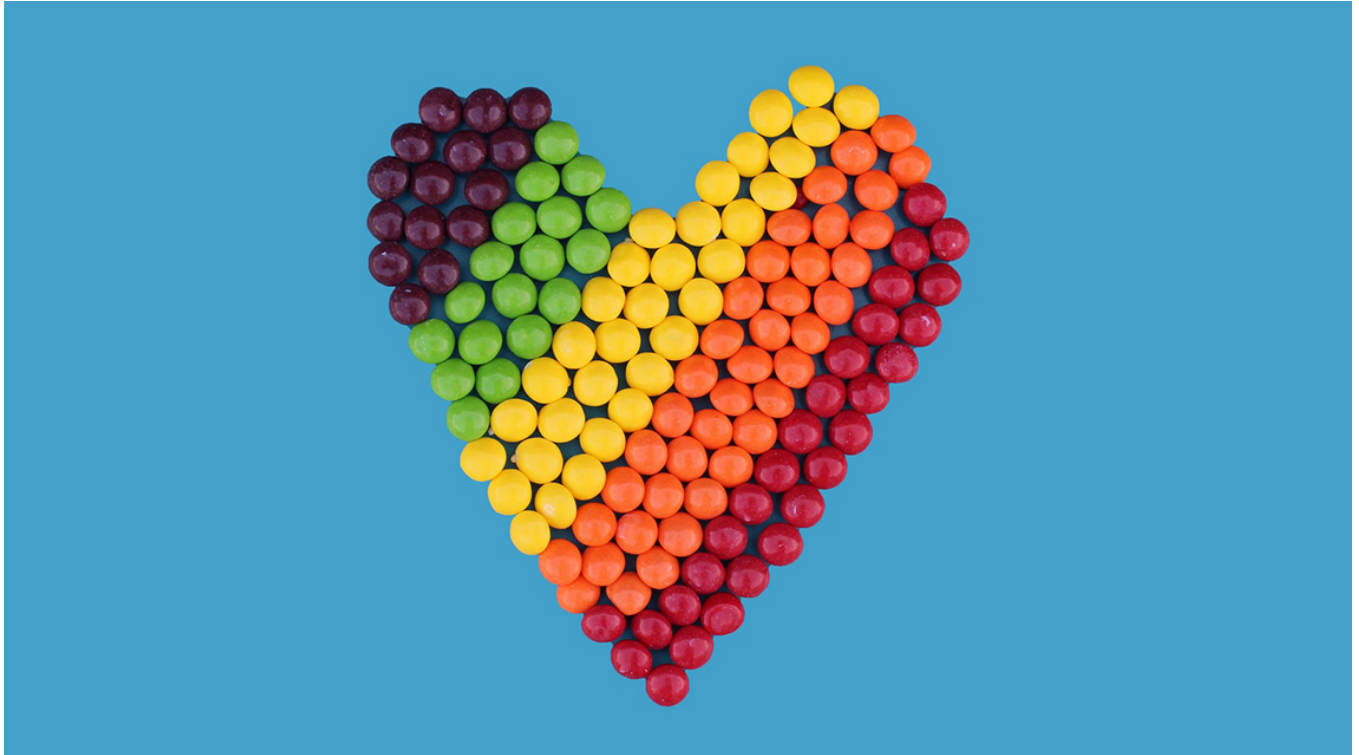


07 | 数组和切片

2018-08-27 郝林



07 | 数组和切片

朗读人：黄洲君
12'11" | 5.58M

0:00 / 12:11

【Go 语言代码较多，建议配合文章收听音频。】

从本篇文章开始，我们正式进入了模块 2 的学习。

在这之前，我们已经聊了很多的 Go 语言和编程方面的基础知识，相信你已经对 Go 语言的开发环境配置、常用源码文件写法，以及程序实体（尤其是变量）及其相关的各种概念和编程技巧（比如类型推断、变量重声明、可重名变量、类型断言、类型转换、别名类型和潜在类型等）都有了一定的理解。

它们都是我认为 Go 语言编程基础中比较重要的部分，同时也是后续文章的基石。如果你在后面的学习过程中感觉有些吃力，那可能是基础仍未牢固，可以再回去复习一下。

我们这次主要讨论 Go 语言的数组（array）类型和切片（slice）类型。它们有时候会让初学者感到困惑。它们都属于集合类的类型，它们的值也都可以用来存储某一种类型的值（或者说元素）。

不过，它们最重要的不同是：数组类型的值（以下简称数组）的长度是固定的，而切片类型的值（以下简称切片）是可变长的。

数组的长度在声明它的时候就必须给定，并且在之后不会再改变。可以说，数组的长度是其类型的一部分。

比如`[1]string`和`[2]string`就是两个不同的数组类型；而切片的类型字面量中只有其元素的类型，而没有其长度。切片的长度可以自动地随着其中元素数量的增长而增长，但不会随着元素数量的减少而减少。

我们其实可以把切片看做是对数组的一层简单的封装，因为在每个切片的底层数据结构中，一定会包含一个数组。后者可以被叫做前者的底层数组，而前者也可以被看作是对后者的某个连续片段的引用。

也正因为如此，Go 语言的切片类型属于引用类型，同属引用类型的还有后面会讲到的字典类型、通道类型、函数类型等；而 Go 语言的数组类型则属于值类型，同属值类型的有基础数据类型以及结构体类型。

注意，Go 语言里不存在像 Java 等编程语言中那种令人困惑的“传值或传引用”问题。在 Go 语言中，我们判断所谓的“传值”或者“传引用”只要看被传递的值的类型就好了。

如果传递的值是引用类型的，那么就是“传引用”。如果传递的值是值类型的，那么就是“传值”。从传递成本的角度讲，引用类型的值往往要比值类型的值低很多。


再说回主题。我们在数组和切片之上都可以应用索引表达式，得到的都会是某个元素。我们在它们之上也都可以应用切片表达式，也都会得到一个新的切片。

通过调用内建函数`len`，我们可以得到它们的长度。通过调用内建函数`cap`，我们可以得到它们的容量。

但要注意，数组的容量永远等于其长度，都是不可变的。而切片的容量却不是这样，并且它的变化是有规律可寻的。下面我们就通过一道题来了解一下。我们今天的问题就是：怎样正确估算切片的长度和容量？

为此，我编写了一个简单的命令源码文件 `demo15.go`。

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // 示例 1。
7     s1 := make([]int, 5)
```

 复制代码

```
8      fmt.Printf("The length of s1: %d\n", len(s1))
9      fmt.Printf("The capacity of s1: %d\n", cap(s1))
10     fmt.Printf("The value of s1: %d\n", s1)
11     s2 := make([]int, 5, 8)
12     fmt.Printf("The length of s2: %d\n", len(s2))
13     fmt.Printf("The capacity of s2: %d\n", cap(s2))
14     fmt.Printf("The value of s2: %d\n", s2)
15 }
```

我描述一下它所做的事情。首先，我用内建函数`make`声明了一个`[]int`类型的变量`s1`。我传给`make`函数的第二个参数是5，从而指明了该切片的长度。我用几乎同样的方式声明了切片`s2`，只不过多传入了一个参数8以指明该切片的容量。

现在，具体的问题是：切片`s1`和`s2`的容量都是多少？

这道题的典型回答：切片`s1`和`s2`的容量分别是5和8。

问题解析

解析一下这道题。`s1`的容量为什么是5呢？因为我在声明`s1`的时候把它的长度设置成了5。

当我们用`make`函数初始化切片时，如果不指明其容量，那么它就会和长度一致。如果在初始化时指明了容量，那么切片的实际容量也就是它了。这也正是`s2`的容量是8的原因。

我们顺便通过`s2`再来明确下长度、容量以及它们的关系。我在初始化`s2`代表的切片时同时指定了它的长度和容量。

在这种情况下，它的容量实际上代表了它的底层数组的长度，这里是8。注意，切片的底层数组等同于我们前面讲到的数组，其长度不可变。

现在你需要跟着我一起想象：有一个窗口，你可以通过这个窗口看到一个数组，但是不一定能看到该数组中的所有元素，有时候只能看到连续的一部分元素。

现在，这个数组就是切片`s2`的底层数组，而这个窗口就是切片`s2`本身。`s2`的长度实际上指明的就是这个窗口的宽度，决定了你透过`s2`可以看到其底层数组中的哪几个连续的元素。

由于`s2`的长度是5，所以你可以看到其底层数组中的第1个元素到第5个元素，对应的底层数组的索引范围是`[0, 4]`。

切片代表的窗口也会被划分成一个一个小格子，就像我们家里的窗户那样。每个小格子都对应着其底层数组中的某一个元素。


我们继续拿`s2`为例，这个窗口最左边的那个小格子对应的正好是其底层数组中的第一个元素，即索引为0的那个元素。因此可以说，`s2`中的索引从0到4所指向的元素恰恰就是其底层数组中

索引从0到4代表的那 5 个元素。

请记住，当我们用make函数或切片值字面量（比如[]int{1, 2, 3}）初始化一个切片时，该窗口最左边的那个小格子总是会对应其底层数组中的第 1 个元素。

但是当我们通过切片表达式基于某个数组或切片生成新切片的时候，情况就变得复杂起来了。我们再来看一个例子：

```
1 s3 := []int{1, 2, 3, 4, 5, 6, 7, 8}
2 s4 := s3[3:6]
3 fmt.Printf("The length of s4: %d\n", len(s4))
4 fmt.Printf("The capacity of s4: %d\n", cap(s4))
5 fmt.Printf("The value of s4: %d\n", s4)
```

 复制代码

切片s3中有 8 个元素，分别是1到8的整数。s3的长度和容量都是8。然后，我用切片表达式s3[3:6]初始化了切片s4。问题是，这个s4的长度和容量分别是多少？

这并不难，用减法就可以搞定。首先你要知道，切片表达式中的方括号里的那两个整数都代表什么。我换一种表达方式你也许就清楚了，即：[3, 6)。

这是数学中的区间表示法，常用于表示取值范围，我其实已经在本专栏用过好几次了。由此可知，[3:6]要表达的就是透过新窗口能看到的，s3中元素的索引范围是从3到5（注意，不包括6）。

这里的3可被称为起始索引，6可被称为结束索引。那么s4的长度就是6减去3，即3。因此可以说，s4中的索引从0到2指向的元素对应的是s3及其底层数组中索引从3到5的那 3 个元素。

再来看容量。我在前面说过，切片的容量代表了它的底层数组的长度，但这仅限于使用make函数或者切片值字面量初始化切片的情况。更通用的规则是：一个切片的容量可以被看作是透过这个窗口最多可以看到的底层数组中元素的个数。

由于s4是通过在s3上施加切片操作得来的，所以s3的底层数组就是s4的底层数组。又因为，在底层数组不变的情况下，切片代表的窗口可以向右扩展，直至其底层数组的末尾。所以，s4的容量就是其底层数组的长度8减去上述切片表达式中的那个起始索引3，即5。

注意，切片代表的窗口是无法向左扩展的。也就是说，我们永远无法透过s4看到s3中最左边的那 3 个元素。

最后，顺便提一下把切片的窗口向右扩展到最大的方法。对于s4来说，切片表达式s4[0:cap(s4)]就可以做到。我想你应该能看懂。该表达式的结果值（即一个新的切片）会是[]int{4, 5, 6, 7, 8}，其长度和容量都是5。

知识扩展

1. 问题：怎样估算切片容量的增长？

一旦一个切片无法容纳更多的元素，Go 语言就会想办法扩容。但它并不会改变原来的切片，而是会生成一个容量更大的切片，然后将把原有的元素和新元素一并拷贝到新切片中。

在一般的情况下，你可以简单地认为新切片的容量（以下简称新容量）将会是原切片容量（以下简称原容量）的 2 倍。

但是，当原切片的长度（以下简称原长度）大于或等于 1024 时，Go 语言将会以原容量的 1.25 倍作为新容量的基准（以下简称新容量基准）。

新容量基准会被调整（不断地与 1.25 相乘），直到结果不小于原长度与要追加的元素数量之和（以下简称新长度）。最终，新容量往往会比新长度大一些，当然，相等也是可能的。

另外，如果我们一次追加的元素过多，以至于使新长度比原容量的 2 倍还要大，那么新容量就会以新长度为基准。

注意，与前面那种情况一样，最终的新容量在很多时候都要比新容量基准更大一些。更多细节可参见runtime包中 slice.go 文件里的growslice及相关函数的具体实现。

我把展示上述扩容策略的一些例子都放到了 demo16.go 文件中。你可以去试运行看看。

2. 问题：切片的底层数组什么时候会被替换？

确切地说，一个切片的底层数组永远不会被替换。为什么？虽然在扩容的时候 Go 语言一定会生成新的底层数组，但是它也同时生成了新的切片。它是把新的切片作为了新底层数组的窗口，而没有对原切片及其底层数组做任何改动。

请记住，在无需扩容时，append函数返回的是指向原底层数组的新切片，而在需要扩容时，append函数返回的是指向新底层数组的新切片。

所以，严格来讲，“扩容”这个词用在这里虽然形象但并不合适。不过鉴于这种称呼已经用得很广泛了，我们也没必要另找新词了。

顺便说一下，只要新长度不会超过切片的原容量，那么使用append函数对其追加元素的时候就不会引起扩容。这只会使紧邻切片窗口右边的（底层数组中的）元素被新的元素替换掉。你可以运行 demo17.go 文件以增强对这些知识的理解。

总结

总结一下，我们今天一起探讨了数组和切片以及它们之间的关系。切片是基于数组的、可变长的，并且非常轻快。一个切片的容量总是固定的，而且一个切片也只会与某一个底层数组绑定在

一起。

此外，切片的容量总会是在切片长度和底层数组长度之间的某一个值，并且还与切片窗口最左边对应的元素在底层数组中的位置有关系。那两个分别用减法计算切片长度和容量的方法你一定要记住。

另外，`append`函数总会返回新的切片，而且如果新切片的容量比原切片的容量更大那么就意味着底层数组也是新的了。+++还有，你其实不必太在意切片“扩容”策略中的一些细节，只要能够理解它的基本规律，并可以进行近似的估算就可以了。

思考题

这里仍然是聚焦于切片的问题。

1. 如果有多个切片指向了同一个底层数组，那么你认为应该注意些什么？
2. 怎样沿用“扩容”的思想对切片进行“缩容”？请写出代码。

这两个问题都是开放性的，你需要认真思考一下。最好在动脑的同时动动手。

[戳此查看 Go 语言专栏文章配套详细代码。](#)

 极客时间

GO语言核心36讲

3个月带你通关 GO 语言

郝林

《Go 并发编程实战》作者
GoHackers 技术社群发起人
前轻松筹大数据负责人



版权归极客邦科技所有，未经许可不得转载

写留言

精选留言



清风徐来

语言描述有点啰嗦太学术化，和我当时看go并发编程第二版开头几章同样的感觉，希望能更加精简一些，直接突出重点要好很多。

👍 36

2018-08-29



melon

👍 20

初始时两个切片引用同一个底层数组，在后续操作中对某个切片的操作超出底层数组的容量时，这两个切片引用的就不是同一个数组了，比如下面这个例子：

```
s1 := []int {1,2,3,4,5}
```

```
s2 := s1[0:5]
```

```
s2 = append(s2, 6)
```

```
s1[3] = 30
```

此时s1[3]的值为30，s2[3]的值仍然为4，因为s2的底层数组已是扩容后的新数组了。

2018-08-27



sky

👍 8

老师您好！我对源码demo16中示例1、3实际运行结果与预期结果表示ok，但唯独示例2的运行结果觉得没有什么规则可供参考，为何不是下面我预期的结果呢，对于实际的运行结果表示不理解，还烦请老师有空帮忙解答下，感谢！

代码如下：

```
// 示例2
```

```
s7 := make([]int, 1024)
```

```
fmt.Printf("The capacity of s7: %d\n", cap(s7))
```

```
s7e1 := append(s7, make([]int, 200)...) 
```

```
fmt.Printf("s7e1: len: %d, cap: %d\n", len(s7e1), cap(s7e1))
```

```
s7e2 := append(s7, make([]int, 400)...) 
```

```
fmt.Printf("s7e2: len: %d, cap: %d\n", len(s7e2), cap(s7e2))
```

```
s7e3 := append(s7, make([]int, 600)...) 
```

```
fmt.Printf("s7e3: len: %d, cap: %d\n", len(s7e3), cap(s7e3))
```

```
fmt.Println()
```

实际运行结果：

```
The capacity of s7: 1024
```

```
s7e1: len: 1224, cap: 1280
```

```
s7e2: len: 1424, cap: 1696
```

```
s7e3: len: 1624, cap: 2048
```

预期运行结果：

```
The capacity of s7: 1024
```

```
s7e1: len: 1224, cap: 1280
```

```
s7e2: len: 1424, cap: 1600
```

```
s7e3: len: 1624, cap: 2000
```

2018-09-15



Laughing

👍 5

1.当两个长度不一的切片使用同一个底层数组，并且两切片的长度均小于数组的容量时，对其
中长度较小的一个切片进行append操作，但不超过底层数组容量，这时会影响长度较长切片
中原来比较小切片多看到的值，因为底层数组被修改了。

2. 可以截取切片的部分数据，然后创建新数组来扩容

2018-08-27



涛哥爱学习

👍 3

老师可以多些图表在文章里，方便阅读

2018-08-30



有铭

👍 3

谢谢老师，今天这篇文才让我意识到以前对切片的认知是不全面的。但也带来一个新问题，大
部分语言里，类似切片的数据结构的实质就是可变数组，他们都没有窗口这个设计，golang是
为啥设计了窗口这个功能呢？这个功能在实际开发中能如何应用呢？我想golang这种极简设计
思想的语言，绝不会搞多余设计，必然是有某种场景，不用切片的窗口就搞不定。但是我想不
出是什么

2018-08-27



mateye

👍 2

老师您好，就像您说的，切片赋值的话会，如果完全赋值，会指向相同的底层数组，`s1 := []int {1,2,3,4}` `s2 := s1[0:4]` 就像这样，这样的话改变s2会影响s1，如何消除这种影响呢

2018-08-30

作者回复

可以用copy函数，或者自己深拷贝。

2018-09-01



小小笑儿

👍 2

切片扩容之后还是会引用底层的原数组，这有时候会造成大量扩容之后的多余内容没有被垃圾
回收。可以使用新建一个数组然后copy的方式。

2018-08-29

作者回复

没错

2018-09-01



wjq310

👍 2

老师，请问下demo16.go的示例三的几个cap值是怎么来的？看这后面的值，不像是2的指数
倍。更奇怪的是，我在不同的地方运行（比如把代码贴到<https://golang.org/go>）得到的结
果还不一样，不知道为什么，麻烦帮忙解答一下，感谢了

2018-08-28

作者回复

每次发现容量不够都会翻一倍，你可以从头算一下。另外，一旦超过1024每次只会增大1.25
倍。

2018-08-29



余泽锋

👍 2

1.底层数组的变动会影响多个切片

2.每一次扩容都需要生成新的切片

2018-08-27



mrly

1

老师，我对demo16的运行结果有疑惑，按 $1024*1.25*1.25*1.25$ 来说，结果应该是这样：
实际运行结果：

```
The capacity of s7: 1024
s7e1: len: 1224, cap: 1280
s7e2: len: 1424, cap: 1696
s7e3: len: 1624, cap: 2048
```

预期运行结果：

```
The capacity of s7: 1024
s7e1: len: 1224, cap: 1280=1024*1.25
s7e2: len: 1424, cap: 1600=1024*1.25*1.25
s7e3: len: 1624, cap: 2000=1024*1.25*1.25*1.25
```

为什么结果不一样呢？
2018-09-27



Empty

1

王老师，能解释一下demo16里面的第三个示例么
2018-09-26



Beau Zhang

1

```
s9 := make([]int, 44)
fmt.Printf("The capacity of s9: %d\n", cap(s9))
s9a := append(s9, make([]int, 45)...)
fmt.Printf("s9a: len: %d, cap: %d\n", len(s9a), cap(s9a))
```

下面是输出结果

```
The capacity of s9: 44
s9a: len: 89, cap: 96
```

按照扩容的规则 $cap = 90$ 才对啊

这是怎么回事？

2018-09-14



徐宁

1

能不能少用点前者后者这类语言，很容易困惑又回头去看
2018-09-06



qiujiangzhe

1

循序渐进，由浅入深，让我这个新手也能跟得上了。
2018-08-27



V

1

老师可以详细讲一下内置函数`make()`和`new()`的区别吗
2018-08-27

作者回复

关于这两个函数的用法和相应规则，Go语言规范里都说的很清楚了。你可以去 golang.google.cn/doc/spec 看一下，描述得很清晰。

2018-08-27



Nixus

0

内容很棒！知识拓展中的数组扩容，好像只讲了一半。看growslice中的源码，好像是只讲了一半，switch的这部分，应该是造成扩容时，没有严格按照2倍 或 1.25倍的罪魁祸首！还没弄明白.....自己需要多努力了！

谢谢老师，祝您早日康复！

2018-10-18



张茹冰

0

请问老师，切片是数组的底层数组吗？这个前者和后者分别指什么

2018-10-16



陈悬高

0

虽然 slice 间接引用了底层数组的元素，但是其指针、长度和容量却是它自己的属性。要更新一个 slice 的指针、长度或容量必须使用显式的赋值。从这个角度看，slice 并不是“纯粹”的引用类型，而是像下面这种聚合类型：

...

```
type IntSlice struct {  
    ptr *int  
    len, cap int  
}
```

...

所以，不仅是在调用 `append` 函数时需要更新 slice 变量。另外，对于任何函数，只要有可能改变 slice 的长度或者容量，或者使得 slice 指向不同的底层数组，都需要更新 slice 变量。

2018-10-14



顾骨

0

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

```
b := a[3:6]
```

```
fmt.Printf("%x\n", *(*int)((unsafe.Pointer)((uintptr)(unsafe.Pointer(&b[0])) - 0x8)))
```

输出结果为a[2]的值，也就是3，这样不就是通过b看到a的前面3个元素了吗

2018-10-12



liyuan35023

0

```
s := make([]int, 2, 3)
```

```
fmt.Printf("%p\n", s)
```

```
s = append(s, 2)
```

```
fmt.Printf("%p", s)
```

使用append，如果没有扩容，那么并不会返回一个新的切片啊，我用的golang 1.11和版本有关么？

2018-10-09



Cyberpunk

0

```
func main() {
    s9 := make([]int, 100)
    fmt.Printf("len:%d, The capacity of s9: %d\n", len(s9), cap(s9))
    s9a := append(s9, make([]int, 100)...)
    fmt.Printf("s9a: len: %d, cap: %d\n", len(s9a), cap(s9a))
    fmt.Println(s9a[:223][222])
    return
}
```

100,100并不是200，而是224为啥呢？

len:100, The capacity of s9: 100

s9a: len: 200, cap: 224

0

2018-10-04



ricktian

0

[3,5)这个好记~ 不错，脑补了好多细节

2018-09-26



Spike

0

我觉得叫指针结构的包装，比叫引用类型更严谨。我查阅了官方文档，没有说slice是引用类型。<https://github.com/golang/go/commit/b34f0551387fcf043d65cd7d96a0214956578f94>

在go的注释里去掉了slice是引用类型的语句

2018-09-23



拖鞋村长

0

java没有引用传递，误认为的引用传递也是复制了一个引用，针对这个引用本身还是值传递

2018-09-05



Mr.Liu

0

每次发现容量不够都会翻一倍，你可以从头算一下。另外，一旦超过1024每次只会增大1.25倍。运行了下示例，发现有的结果符合这个规律，有的不符合。这个和go版本有关系吗？

2018-09-04

作者回复

我记得是没有，你可以再算一下，或者直接去看源码。

2018-09-04



jacke

0

不太明白append返回的都是新的slice、所谓新的只什么？新的内存地址对于的新对象？

2018-08-30

作者回复

新的slice是指新的slice数据结构，和之前的不同，内存地址也不同。

2018-09-01



yfdream

0

用数组为基础进行切片，会把原来数组当做底层数组吗？还是拷贝个新的当底层数组？
2018-08-29



Spike

0

slices are not passed by reference, nothing is passed by reference in go. Everything is a copy, every assignment, every parameter to a function, there are no exceptions to this rule. 这是Dave Cheney的原话 slice不是指针也不是引用 希望作者参考下<https://dave.cheney.net/2018/07/12/slices-from-the-ground-up> 这篇博文
2018-08-28

作者回复

Dave Cheney 又不是Go语言团队中的人。我原创文章中的所有内容都只遵从Go语言官方文档和Go语言源码。你可以再好好看看Go语言规范。再说一遍slice是引用类型之一。另外他的这句话你就没理解对。Go语言里不会传引用，这我也多次强调过。但是，内建数据类型中会有值类型和引用类型之分。

2018-08-29



王宏达达达

0

请问老师，golang入门的书籍是否有推荐？
2018-08-28

作者回复

《Go并发编程实战》第一版就可以。

2018-08-29



许森森

0

1 改了一个切片的元素，其他的可能都受影响。如果扩容超过容量，底层会指向新的数组。从而而不受影响。

2生成新的slice

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

```
    s1 = shrinkSlice(s1)
```

```
    printSlice("s1", s1)  
}
```

```
func shrinkSlice(x []int) []int{  
    if( cap(x) > 0 ) {  
        x = x[0:cap(x)-1]  
    }  
    return x
```

```
}
```

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

输出结果

s1 len=5 cap=5 [1 2 3 4 5]

s1 len=4 cap=5 [1 2 3 4]

2018-08-28

作者回复

嗯对，一切都从底层数组上找答案。

2018-08-29



Kyle Liu

老师厉害，以前的疑惑一下就明白了

2018-08-27

👍 0