链滴

Go 边看边练 -《Go 学习笔记》系列（八）

作者：88250

原文链接：https://ld246.com/article/1438311936449

来源网站：链滴

上一篇： https://hacpai.com/article/1438260619759

ToC

4.1 Array

和以往认知的数组有很大不同。

- 数组是值类型，赋值和传参会复制整个数组，而不是指针。
- 数组长度必须是常量，且是类型的组成部分。 [2]int 和 [3]int 是不同类型。
- 支持 "=="、"!=" 操作符，因为内存总是被初始化过的。
- 指针数组 [n]*T，数组指针 *[n]T。

可用复合语句初始化。

```
a := [3]int{1, 2} // 未初始化元素值为 0。
b := [...]int{1, 2, 3, 4} // 通过初始化值确定数组长度。
c := [5]int{2: 100, 4:200} // 使用索引号初始化元素。

d := [...]struct {
    name string
    age uint8
}{
    {"user1", 10}, // 可省略元素类型。
    {"user2", 20}, // 别忘了最后一行的逗号。
```

```
}
```

支持多维数组。

```
a := [2][3]int{{1, 2, 3}, {4, 5, 6}}
b := [...][2]int{{1, 1}, {2, 2}, {3, 3}} // 第 2 纬度不能用 "..."。
```

值拷贝行为会造成性能问题，通常会建议使用 slice，或数组指针。

```
<iframe style="border:1px solid" src="https://wide.b3log.org/playground/84e741fc120579eb9eea9222db5a672.go?embed=true" width="99%" height="500"></iframe>
```

内置函数 len 和 cap 都返回数组长度 (元素数量)。

```
a := [2]int{}
println(len(a), cap(a)) // 2, 2
```

4.2 Slice

需要说明，slice 并不是数组或数组指针。它通过内部指针和相关属性引用数组片段，以实现变长方案。

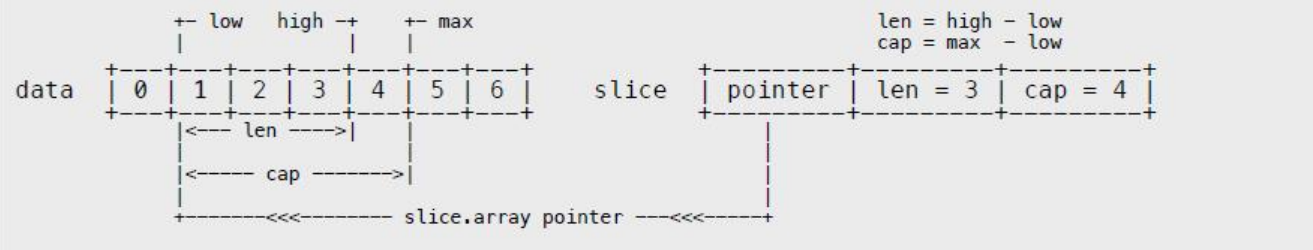
runtime.h

```
struct Slice
{ // must not move anything
    byte* array; // actual data
    uintgo len; // number of elements
    uintgo cap; // allocated number of elements
};
```

- 引用类型。但自身是结构体，值拷贝传递。
- 属性　len 表示可用元素数量，读写操作不能超过该限制。
- 属性　cap 表示最大扩张容量，不能超出数组限制。
- 如果　slice == nil，那么 len、cap 结果都等于 0。

data := [...]int{0, 1, 2, 3, 4, 5, 6}

slice := data[1:4:5] // [low : high : max]



创建表达式使用的是元素索引号，而非数量。

```
data := [...]int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

expression   slice                 len    cap     comment
-----------+---------------------+------+-------+---------------------
data[:6:8]   [0 1 2 3 4 5]         6      8       省略 low.
data[5:]     [5 6 7 8 9]           5      5       省略 high、max。
data[:3]     [0 1 2]               3      10      省略 low、max。
data[:]      [0 1 2 3 4 5 6 7 8 9] 10     10      全部省略。
```

读写操作实际目标是底层数组，只需注意索引号的差别。

```
data := [...]int{0, 1, 2, 3, 4, 5}

s := data[2:4]
s[0] += 100
s[1] += 200

fmt.Println(s)
fmt.Println(data)
```

输出：

```
[102 203]
[0 1 102 203 4 5]
```

可直接创建 slice 对象，自动分配底层数组。

```
s1 := []int{0, 1, 2, 3, 8: 100} // 通过初始化表达式构造，可使用索引号。
fmt.Println(s1, len(s1), cap(s1))

s2 := make([]int, 6, 8) // 使用 make 创建，指定 len 和 cap 值。
fmt.Println(s2, len(s2), cap(s2))

s3 := make([]int, 6) // 省略 cap，相当于 cap = len。
fmt.Println(s3, len(s3), cap(s3))
```

输出：

```
[0 1 2 3 0 0 0 0 100] 9 9
[0 0 0 0 0 0]         6 8
[0 0 0 0 0 0]         6 6
```

使用 make 动态创建 slice，避免了数组必须用常量做长度的麻烦。还可用指针直接访问底层数组，化成普通数组操作。

```
s := []int{0, 1, 2, 3}

p := &s[2] // *int, 获取底层数组元素指针。
*p += 100
```

```
fmt.Println(s)
```

输出：

```
[0 1 102 3]
```

至于 [][]T，是指元素类型为 []T 。

```
data := [][]int{
    []int{1, 2, 3},
    []int{100, 200},
    []int{11, 22, 33, 44},
}
```

可直接修改 struct array/slice 成员。

```
d := [5]struct {
    x int
}{}

s := d[:]

d[1].x = 10
s[2].x = 20

fmt.Println(d)
fmt.Printf("%p, %p\n", &d, &d[0])
```

输出:

[{0} {10} {20} {0} {0}]

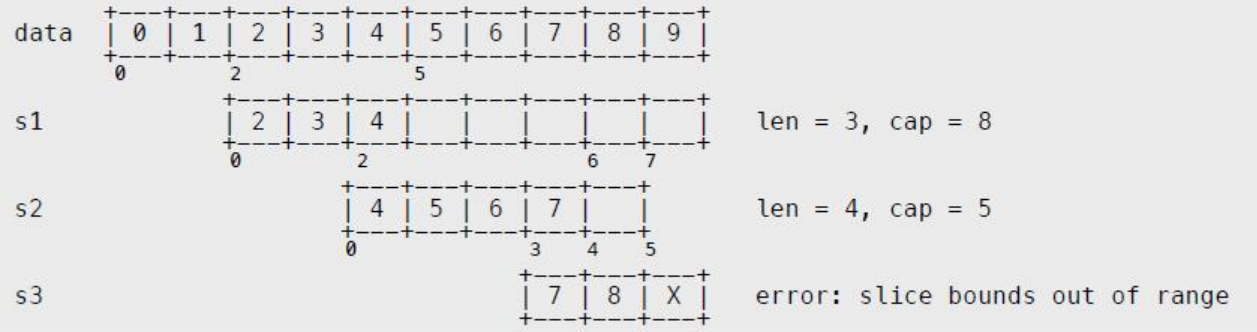
0x20819c180, 0x20819c180

4.2.1 reslice

所谓 reslice，是基于已有 slice 创建新 slice 对象，以便在 cap 允许范围内调整属性。

```
s := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

s1 := s[2:5] // [2 3 4]
s2 := s1[2:6:7] // [4 5 6 7]
s3 := s2[3:6] // Error
```



新对象依旧指向原底层数组。

```
s := []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

s1 := s[2:5] // [2 3 4]
s1[2] = 100

s2 := s1[2:6] // [100 5 6 7]
s2[3] = 200

fmt.Println(s)
```

输出：

```
[0 1 2 3 100 5 6 200 8 9]
```

4.2.2 append

向 slice 尾部添加数据，返回新的 slice 对象。

```
s := make([]int, 0, 5)
fmt.Printf("%p\n", &s)

s2 := append(s, 1)
fmt.Printf("%p\n", &s2)

fmt.Println(s, s2)
```

输出：

```
0x210230000
0x210230040
[] [1]
```

简单点说，就是在 array[slice.high] 写数据。

```
data := [...]int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

s := data[:3]
s2 := append(s, 100, 200) // 添加多个值。

fmt.Println(data)
fmt.Println(s)
fmt.Println(s2)
```

输出：

```
[0 1 2 100 200 5 6 7 8 9]
[0 1 2]
```

```
[0 1 2 100 200]
```

一旦超出原 slice.cap 限制，就会重新分配底层数组，即便原数组并未填满。

```
data := [...]int{0, 1, 2, 3, 4, 10: 0}
s := data[:2:3]

s = append(s, 100, 200) // 一次 append 两个值，超出 s.cap 限制。

fmt.Println(s, data) // 重新分配底层数组，与原数组无关。
fmt.Println(&s[0], &data[0]) // 比对底层数组起始指针。
```

输出：

```
[0 1 100 200] [0 1 2 3 4 0 0 0 0 0 0]
0x20819c180 0x20817c0c0
```

从输出结果可以看出，append 后的 s 重新分配了底层数组，并复制数据。如果只追加一个值，则不超过 s.cap 限制，也就不会重新分配。

通常以 2 倍容量重新分配底层数组。在大批量添加数据时，建议一次性分配足够大的空间，以减少内分配和数据复制开销。或初始化足够长的 len 属性，改用索引号进行操作。及时释放不再使用的 slice 对象，避免持有过期数组，造成 GC 无法回收。

```
s := make([]int, 0, 1)
c := cap(s)

for i := 0; i < 50; i++ {
    s = append(s, i)
    if n := cap(s); n > c {
        fmt.Printf("cap: %d -> %d\n", c, n)
        c = n
    }
}
```

输出:

```
cap: 1 -> 2
cap: 2 -> 4
cap: 4 -> 8
cap: 8 -> 16
cap: 16 -> 32
cap: 32 -> 64
```

4.2.3 copy

函数 copy 在两个 slice 间复制数据，复制长度以 len 小的为准。两个 slice 可指向同一底层数组，允元素区间重叠。

```
<iframe style="border:1px solid" src="https://wide.b3log.org/playground/d8ba515d1dc7b13
44c67ba4ce9d1960.go?embed=true" width="99%" height="500"></iframe>
```

应及时将所需数据 copy 到较小的 slice，以便释放超大号底层数组内存。

下一篇： https://hacpai.com/article/1438596722873

- **本系列是基于雨痕的《Go 学习笔记》（第四版）整理汇编而成，非常感谢雨痕的辛勤付出与分享！**
- 转载请注明：文章转载自： **黑客与画家的社区** [https://hacpai.com]
- 如果你觉得本章节做得不错，请在下面打赏一下吧~

社区小贴士

- 关注标签 [golang] 可以方便查看 Go 相关帖子
- 关注作者后如有新帖将会收到通知