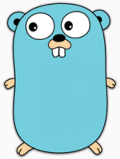
Programming in Go

Matt Holiday
Christmas 2020



Slices in Detail

Empty vs nil slice

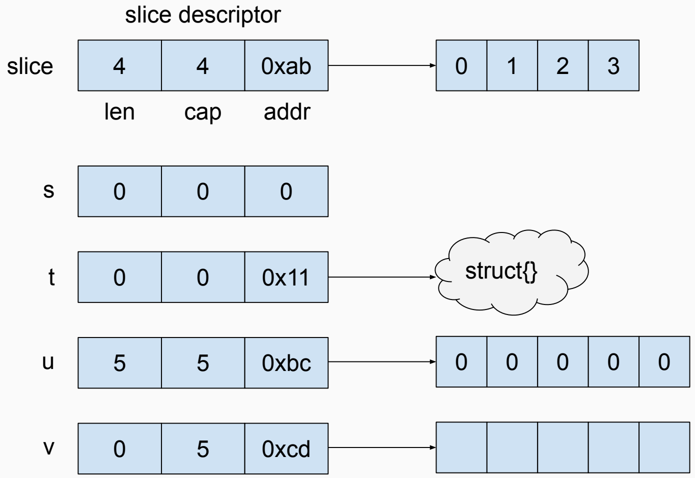
```go
var s []int

t := []int{}
u := make([]int, 5)
v := make([]int, 0, 5)

fmt.Printf("%d, %d, %T, %5t, %#[3]v\n", len(s), cap(s), s, s == nil)
fmt.Printf("%d, %d, %T, %5t, %#[3]v\n", len(t), cap(t), t, t == nil)
fmt.Printf("%d, %d, %T, %5t, %#[3]v\n", len(u), cap(u), u, u == nil)
fmt.Printf("%d, %d, %T, %5t, %#[3]v\n", len(v), cap(v), v, v == nil)
```

```
0, 0, []int,  true, []int(nil)
0, 0, []int, false, []int{}
5, 5, []int, false, []int{0, 0, 0, 0, 0}
0, 5, []int, false, []int{}
```

Slice follow-up



Empty vs nil slice

Slices (and maps) encoding differently in JSON when nil

```go
package main

import ("encoding/json"; "fmt")

func main() {
    var a []int

    j1, _ := json.Marshal(a)
    fmt.Println(string(j1))    // null

    b := []int{}

    j2, _ := json.Marshal(b)
    fmt.Println(string(j2))    // []
}
```

Ugly #1: Slice length vs capacity

```
a := [3]int{1, 2, 3}
b := a[0:1]                // b is a slice of a's first item

fmt.Println("a =", a)      // a = [1 2 3]
fmt.Println("b =", b)      // b = [1]

c := b[0:2]                // WTF? but the array has 3 entries

fmt.Println("a =", a)      // a = [1 2 3]
fmt.Println("c =", c)      // c = [1 2]

fmt.Println(len(b))        // prints 1
fmt.Println(cap(b))        // prints 3

fmt.Println(len(c))        // prints 2
fmt.Println(cap(c))        // prints 3
```

Ugly #1: Slice length vs capacity

Go 1.2 added the “three index” slice operator `[i:j:k]` where length is `j-i` and capacity is `k-i`

```
d := a[0:1:1]            // this is what you probably meant

fmt.Println("a =", a)    // a = [1 2 3]
fmt.Println("d =", d)    // d = [1]

fmt.Println(len(d))      // prints 1
fmt.Println(cap(d))      // prints 1
```

Ugly #2: Slice mutating underlying array

```
a := [3]int{1, 2, 3}; b := a[0:1]; c := b[0:2]

b = append(b, 4)                        // grows b, mutates a
fmt.Printf("a[%p] = %v\n", &a, a)  // a[0xc000014020] = [1 4 3]
fmt.Printf("b[%p] = %[1]v\n", b)   // b[0xc000014020] = [1 4]

c = append(c, 5)                        // grows c, mutates a
fmt.Printf("a[%p] = %v\n", &a, a)  // a[0xc000014020] = [1 4 5]
fmt.Printf("c[%p] = %[1]v\n", c)   // c[0xc000014020] = [1 4 5]

c = append(c, 6)                        // forces allocation!
fmt.Printf("a[%p] = %v\n", &a, a)  // a[0xc000014020] = [1 4 5]
fmt.Printf("c[%p] = %[1]v\n", c)   // c[0xc000078030] = [1 4 5 6]

c[0] = 9                                // mutates a different array!
fmt.Printf("a[%p] = %v\n", &a, a)  // a[0xc000014020] = [1 4 5]
fmt.Printf("c[%p] = %[1]v\n", c)   // c[0xc000078030] = [9 4 5 6]
```