链滴

Golang 入门笔记 -06-Map

作者：zyk

原文链接：https://ld246.com/article/1602066740004

来源网站：链滴

声明和初始化 Map

map 是一种特殊的数据结构，通过键 key 和值 value 来保存数据，可以快速地根据 key 找到其对应的 alue，与 python 中的字典和 Java 中的 HashMap 类似。

map 是**引用类型**，声明方式如下：

```
var variable map[keyType]valueType
```

在声明的时候不需要知道 map 的长度，map 是可以**动态增长**的。

未初始化的 map 的值是 nil。

map 中的 key 只能是可以用 == 和 != 来比较的变量类型，例如 string，int 和 float 等，不能是数，结构体和切片，但指针和接口可以作为 key。如果要使用结构体来作为 key，需要提供 Key() 和 Ha h() 方法，value 可以为任意类型。

可以通过赋值符号来设置 key1 对应的 value：

```
map1[key1] = value
```

也可以根据赋值符号获取 key1 对应的 value，若 map1 中没有 key1 对应的值，val 将会被赋值 ma 1 的值类型的空值。

```
val := map1[key1]
```

示例：

```
package main
import "fmt"

func main() {
```

```
    var mapLit map[string]int
    var mapAssigned map[string]int

    mapLit = map[string]int{"one": 1, "two": 2}
    mapCreated := make(map[string]float32)
    mapAssigned = mapLit

    mapCreated["key1"] = 4.5
    mapCreated["key2"] = 3.14159
    mapAssigned["two"] = 3

    fmt.Printf("Map literal at \"one\" is: %d\n", mapLit["one"])
    fmt.Printf("Map created at \"key2\" is: %f\n", mapCreated["key2"])
    fmt.Printf("Map assigned at \"two\" is: %d\n", mapLit["two"])
    fmt.Printf("Map literal at \"ten\" is: %d\n", mapLit["ten"])
}
```

上述代码运行结果为：

```
Map literal at "one" is: 1
Map created at "key2" is: 3.14159
Map assigned at "two" is: 3
Mpa literal at "ten" is: 0
```

可以用 {key1: val1, key2: val2} 来初始化 map。

map 是引用类型，因此用 make 来分配内存。

map 的常规初始化方式：

```
var map1[keyType]valueType = make(map[keyType]valueType)
```

也可以简写为：

```
map1 := make(map[keyType]valueType)
```

mapCreated := make(map[string]float) 相当于：mapCreated := map[string]float{}。

Map 容量

map 会动态扩容，因此它不存在长度限制。但也可以在初始化 map 时指定它的初始容量，格式如下：

```
map1 := make(map[string]int, 100) // map1 的初始容量为 100
```

当 map 增长到最大容量时，容量会自动加 1。但对于性能要求较高的场景，若 map 中保存的数据较或易快速扩张，建议在初始化时指定其初始容量。

判断键值对是否存在

我们可以通过如下的格式来判断 map 中的 key 是否存在：

```
val, ok := map1[key]
```

若 ok 的值为 true，则表明该 key 存在，否则不存在。

若只是想判断 key 是否存在，而不需要获取 key 对应的 value，可以这样写：

```
_, ok := map1[key]
```

还可以和 if 混用：

```
if _, ok := map1[key]; ok {
    // ...
}
```

删除元素

删除 map 中的元素，可以通过 delete 函数，如删除 map1 中 key 对应的元素：

```
delete(map1, key)
```

即使要删除的 key 不存在，也不会报错。

遍历 Map

可以通过 for range 循环来遍历 map：

```
for key, value := range map1 {
    // ...
}
```

如果只需遍历 map 中的 value，可以用匿名变量来接收 key：

```
for _, value := range map1 {
    // ...
}
```

如果只需要遍历 map 的 key，可以省略 value：

```
for key := range map1 {
    // ...
}
```

我们来看一个例子：

```
package main
import "fmt"

func main() {
    map1 := make(map[int]float32)
    map1[1] = 1.0
    map1[2] = 2.0
    map1[3] = 3.0
    map1[4] = 4.0
    for key, value := range map1 {
        fmt.Printf("key is: %d - value is: %f\n", key, value)
    }
}
```

上述代码运行结果为：

```
key is: 3 - value is: 3.000000
key is: 1 - value is: 1.000000
key is: 4 - value is: 4.000000
key is: 2 - value is: 2.000000
```

注意：map 中元素不是按照 key 来排序的。

Map 类型的切片

如果我们要创建一个 map 类型的切片，一定要使用两次 make() 函数，第一次分配切片的内存，第次分配 map 的内存。

来看一个例子：

```
package main
import "fmt"

func main() {
    // Version A:
    items := make([]map[int]int, 5)
    for i:= range items {
        items[i] = make(map[int]int, 1)
        items[i][1] = 2
    }
    fmt.Printf("Version A: Value of items: %v\n", items)

    // Version B: NOT GOOD!
    items2 := make([]map[int]int, 5)
    for _, item := range items2 {
        item = make(map[int]int, 1) // item 仅仅是切片 items2 元素的拷贝
        item[1] = 2
    }
    fmt.Printf("Version B: Value of items: %v\n", items2)
}
```

运行结果为：

```
Version A: Value of items: [map[1:2] map[1:2] map[1:2] map[1:2] map[1:2]]
Version B: Value of items: [map[] map[] map[] map[] map[]]
```