Go 语言Map(集合)

Map 是一种无序的键值对的集合。Map 最重要的一点是通过 key 来快速检索数据，key 类似于索引，指向数据的值。map是一种`key:value`键值对的数据结构容器。

Map 是一种集合，所以我们可以像迭代数组和切片那样迭代它。不过，Map 是无序的，我们无法决定它的返回顺序，这是因为 Map 是使用 hash 表来实现的。

Map 最重要的一点是通过 key 来快速检索数据，key 类似于索引，指向数据的值。

Map 是引用类型的，必须初始化才能使用。

定义 Map

可以使用内建函数 make 也可以使用 map 关键字来定义 Map:

```
/* 声明变量，默认 map 是 nil */
var map_variable map[key_data_type]value_data_type

/* 使用 make 函数 */
var map_variable = make(map[key_data_type]valuetype)。
/*或者简写为：*/
map_variable := make(map[key_data_type]value_data_type)。
```

`map_variable`：map名称

`key_data_type`：key的数据类型

`value_data_type`：value值的数据类型

如果不初始化 map，那么就会创建一个 nil map。nil map 不能用来存放键值对。

示例：

```
package main

import "fmt"

func main() {
    var student map[string]string /*创建集合 */
    student = make(map[string]string)
    fmt.Printf("len=%d type=%T ,map=%v\n", len(student), student, student)
}
#结果
len=0 type=map[string]string ,map=map[]
```

初始化及赋值 Map

```
package main

import "fmt"
```

```
func main() {
    //初始化集合
    student := make(map[string]string)

    /* map插入key - value对 */
    student["name"] = "包子"
    student["age"] = "18"
    student["email"] = "baozi@163.com"
    fmt.Printf("len=%d type=%T ,map=%v\n", len(student), student, student)
    //直接赋值也是初始化，声明的时候填充元素
    var teacher = map[string]string{"name": "肉包子", "age": "20", "email":
"roubaozi@163.com"}
    fmt.Printf("len=%d type=%T ,map=%v\n", len(teacher), teacher, teacher)
}
#结果
len=3 type=map[string]string ,map=map[age:18 email:baozi@163.com name:包子]
len=3 type=map[string]string ,map=map[age:20 email:roubaozi@163.com name:肉包子]
```

Map基本使用

Map中的数据都是成对出现的，所以可以通过下标key，获得Map中的值。

```
package main

import "fmt"

func main() {
    scoreMap := make(map[string]int, 8)
    scoreMap["张三"] = 90
    scoreMap["小明"] = 100
    fmt.Println(scoreMap)
    fmt.Println(scoreMap["小明"])
    fmt.Printf("type of a:%T\n", scoreMap)
}
#结果
map[小明:100 张三:90]
100
type of a:map[string]int
```

Map 容量

map类型的变量默认初始值为nil，需要使用make()函数来分配内存。语法为：

```
make(map[KeyType]ValueType, [cap])
```

其中cap表示map的容量，该参数虽然不是必须的，但是我们应该在初始化map的时候就为其指定一个合适的容量。

和数组不同，map 可以根据新增的 key-value 对动态的伸缩，因此它不存在固定长度或者最大限制。但是你也可以选择标明 map 的初始容量 `capacity`，就像这样：`make(map[keytype]valuetype, cap)`。例如：

```
map2 := make(map[string]float32, 100)
```

当 map 增长到容量上限的时候，如果再增加新的 key-value 对，map 的大小会自动加 1。所以出于性能的考虑，对于大的 map 或者会快速扩张的 map，即使只是大概知道容量，也最好先标明。

这里有一个 map 的具体例子，即将音阶和对应的音频映射起来：

```
noteFrequency := map[string]float32 {
    "C0": 16.35, "D0": 18.35, "E0": 20.60, "F0": 21.83,
    "G0": 24.50, "A0": 27.50, "B0": 30.87, "A4": 440}
```

判断某个键是否存在

Go语言中有个判断map中键是否存在的特殊写法，格式如下:

```
value, ok := map[key]
```

或者和 if 混合使用：

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

在实际应用中，Map可以使用 `val = map[key]` 的方法获取 key 对应的值 value。如果 Map中不存在 key，val ue就是一个值类型的空值。这就会给我们带来困惑了：现在我们没法区分到底是 key 不存在还是它对应的 value 就是空值。

为了解决这个问题，我们可以这么用：`value, ok := map[key]`

ok 返回一个 bool 值：如果 key 存在于 map，vallue 就是 key 对应的 value 值，并且 ok 为true；如果 key 不存在，value 就是一个空值，并且 ok 会返回 false。

示例：

```
package main

import "fmt"

func main() {
    scoreMap := make(map[string]int)
    scoreMap["张三"] = 90
    scoreMap["小明"] = 100
    // 如果key存在ok为true,v为对应的值；不存在ok为false,v为值类型的零值
    v, ok := scoreMap["张三"]
    if ok {
        fmt.Println(v)
    } else {
        fmt.Println("查无此人")
    }
}
#结果
90
```

delete() 函数

delete() 函数用于删除集合的元素, 参数为 map 和其对应的 key。

格式如下:

```
delete(map, key)
```

`map`:表示要删除键值对的map

`key`:表示要删除的键值对的键

示例:

```
package main

import "fmt"

func main() {
    scoreMap := make(map[string]int)
    scoreMap["张三"] = 90
    scoreMap["小明"] = 100
    scoreMap["王五"] = 60
    delete(scoreMap, "小明") //将小明:100从map中删除
    fmt.Printf("scoreMap: %v\n", scoreMap)
}
#结果
scoreMap: map[张三:90 王五:60]
```

Map的遍历

Go语言中使用for range遍历map。

示例:

```
package main

import "fmt"

func main() {
    scoreMap := make(map[string]int)
    scoreMap["张三"] = 90
    scoreMap["小明"] = 100
    scoreMap["王五"] = 60
    fmt.Printf("scoreMap: %v\n", scoreMap)

    for k, v := range scoreMap {
        fmt.Println(k, v)
    }
}
#结果
scoreMap: map[小明:100 张三:90 王五:60]
张三 90
小明 100
王五 60
```

> 注意: 遍历map时的元素顺序与添加键值对的顺序无关。

Map 的排序

Map 默认是无序的，不管是按照 key 还是按照 value 默认都不排序。

如果你想为 Map 排序，需要将 key（或者 value）拷贝到一个切片，再对切片排序，然后可以使用切片的 for-range 方法打印出所有的 key 和 value。

```
package main

import (
    "fmt"
    "math/rand"
    "sort"
)

func main() {
    var scoreMap = make(map[string]int, 10)

    for i := 0; i < 10; i++ {
        key := fmt.Sprintf("stu%02d", i) //生成stu开头的字符串
        value := rand.Intn(10)           //生成0~99的随机整数
        scoreMap[key] = value
    }
    fmt.Printf("scoreMap: %v\n", scoreMap)
    //取出map中的所有key存入切片keys
    var keys = make([]string, 0, 10)
    for key := range scoreMap {
        keys = append(keys, key)
    }
    // //对切片进行排序
    sort.Strings(keys)
    fmt.Printf("keys: %v\n", keys)
    // //按照排序后的key遍历map
    for _, key := range keys {
        fmt.Println(key, scoreMap[key])
    }
}
#结果
scoreMap: map[stu00:1 stu01:7 stu02:7 stu03:9 stu04:1 stu05:8 stu06:5 stu07:0
stu08:6 stu09:0]
keys: [stu00 stu01 stu02 stu03 stu04 stu05 stu06 stu07 stu08 stu09]
stu00 1
stu01 7
stu02 7
stu03 9
stu04 1
stu05 8
stu06 5
stu07 0
stu08 6
stu09 0
```

Map类型的切片

元素为map类型的切片

```
var mapSlice = make([]map[string]string, 3)
```

示例:

```
package main

import (
    "fmt"
)

func main() {
    var mapSlice = make([]map[string]string, 3)
    fmt.Printf("mapSlice: %v\n", mapSlice)
    for index, value := range mapSlice {
        fmt.Printf("index:%d value:%v\n", index, value)
    }
    fmt.Println("after init")
    // 对切片中的map元素进行初始化
    mapSlice[0] = make(map[string]string, 10)
    mapSlice[0]["name"] = "王五"
    mapSlice[1] = make(map[string]string, 10)
    mapSlice[1]["name"] = "张三"
    mapSlice[2] = make(map[string]string, 10)
    mapSlice[2]["name"] = "李四"
    for index, value := range mapSlice {
        fmt.Printf("index:%d value:%v\n", index, value)
    }
    fmt.Printf("mapSlice: %v\n", mapSlice)
}
#结果
mapSlice: [map[] map[] map[]]
index:0 value:map[]
index:1 value:map[]
index:2 value:map[]
after init
index:0 value:map[name:王五]
index:1 value:map[name:张三]
index:2 value:map[name:李四]
mapSlice: [map[name:王五] map[name:张三] map[name:李四]]
```

值为切片类型的map

```
var sliceMap = make(map[string][]string, 3)
```

示例:

```
package main

import (
    "fmt"
)
```

```
func main() {
    var sliceMap = make(map[string][]string, 3)
    fmt.Println(sliceMap)
    fmt.Println("after init")
    key := "中国"
    value, ok := sliceMap[key]
    if !ok {
        value = make([]string, 0, 2)
    }
    value = append(value, "北京", "上海")
    sliceMap[key] = value
    fmt.Println(sliceMap)
}
#结果
map[]
after init
map[中国:[北京 上海]]
```