数据结构

映射

映射Map，也有语言称为字典。

- 长度可变
- 存储的元素是key-value对（键值对），value可变
- key无序不重复
- 不可索引，需要通过key来访问
- **不支持零值可用**，也就是说，必须要用**make**或**字面常量构造**
- 引用类型
- 哈希表

哈希算法

哈希Hash算法特征

- hash(x) 一定得到一个y值
- 输入可以是任意长度，输出是固定长度
- hash函数一般设计的计算效率很高
- 由于输入空间（可以理解为取值范围）远远大于输出空间，有可能不同的x经过hash得到同样的y，这称为碰撞，也称冲突
- 不同的x计算出的y值应当在输出空间中分布均匀，减少碰撞
- 不能由y反推出x，hash算法不可逆
- 一个微小的变化，哪怕x是一个bit位的变化，也将引起结果y巨大的变化

常见算法

- SHA（Secure Hash Algorithm）安全散列算法，包含一个系列算法，分别是SHA-1、SHA-224、SHA-256、SHA-384，和SHA-512。
    - 数字签名防篡改
- MD5（Message Digest Algorithm 5）信息摘要算法5，输出是128位。运算速度叫SHA-1快
    - 用户密码存储
    - 上传、下载文件完整性校验
    - 大的数据的快速比对，例如字段很大，增加一个字段存储该字段的hash值，对比内容开是否修改

```go
package main

import (
    "crypto/sha256"
    "fmt"
)

func main() {
    // https://pkg.go.dev/crypto/sha256#example-New
    h := sha256.New()
    h.Write([]byte("abc"))
    r := h.Sum(nil)
    s := fmt.Sprintf("%x", r)
```

```
    fmt.Printf("%T %s %d\n", r, s, len(s))
}
```

```
package main

import (
    "crypto/md5"
    "fmt"
)

func main() {
    // https://pkg.go.dev/crypto/md5#example-New
    h := md5.New()
    h.Write([]byte("abc"))
    fmt.Printf("%T %[1]x\n", h.Sum(nil)) // []uint8
900150983cd24fb0d6963f7d28e17f72
    h.Reset()
    h.Write([]byte("abd"))
    fmt.Printf("%T %[1]x\n", h.Sum(nil)) // []uint8
4911e516e5aa21d327512e0c8b197616
}
```

内存模型

map采用哈希表实现。Go的map类型也是引用类型，有一个标头值hmap，指向一个底层的哈希表。

哈希表Hash Table

- 开辟一块内存空间，分割出“房间”，这个房间称为bucket桶，按照y值为“房间”编号
- 使用给出的x计算出对应的y值，可以按照某种关系计算出数据将被存储到的“房间号码”，将数据存入该房间
- 即使是hash函数设计的好，数据分布均匀，但是存储的数据很多（超过负载因子），则需要扩容，否则再加入数据后，冲突将很多

理解的hash函数原理，可以思考除留余数法，即hash(x) = key mod p，p是hash表大小，看做房间个数。

hash(x_0) => $Room_k$，计算出一个确定的房间号码。

hash冲突

- 房间有人占了，就重新找个空房间让客人住，这是开放地址法
- 房间有人占了，就挤在同一个房间内，将值用链表存储在一起，这是链地址法，也称拉链法，Go语言采用，但做了一定的优化

思考：什么是相同的key？冲突的key有什么异同？

构造

```
var m1 map[string]int // nil，很危险。map不是零值可用
fmt.Println(m1, m1 == nil)
m1["t"] = 200 // panic
```

```
// 1 字面量
var m0 = map[string]int{} // 安全，没有一个键值对而已

var m1 = map[string]int{
    "a": 11,
    "b": 22,
    "c": 33,
}

// 2 make
m2 := make(map[int]string) // 一个较小的起始空间大小
m2[100] = "abc"
m3 := make(map[int]string, 100) // 容量100，长度为0，为了减少扩容，提前给合适的容量
```

新增或修改

```
var m = make(map[string]int)
m["a"] = 11 // key不存在，则创建新的key和value对
m["a"] = 22 // key已经存在，则覆盖value
```

查找

- 使用map一般需要使用key来查找，时间复杂度为**O(1)**

```
fmt.Println(m["a"]) // 存在返回22
fmt.Println(m["b"]) // 不存在返回零值0，这样不能判断"b"这个key存在否，需要解析返回值
if _, ok := m["b"]; !ok {
    fmt.Println("不存在", v)
}
```

key访问map最高效的方式

长度

```
len(m) // 返回kv对的个数
```

注意：由于map的特殊构造，不能使用cap。

移除

```
delete(m, "a") // 存在，删除kv对
delete(m, "b") // 不存在，删除操作也不会panic
```

遍历

```go
var m = map[string]int{
    "a": 11,
    "b": 22,
    "c": 33,
}

for k, v := range m {
    fmt.Println(k, v)
}
```

注意：map的key是无序的，千万不要从遍历结果来推测其内部顺序

问题：数组、切片、映射谁遍历效率更高?

排序

Go的标准库提供了sort库，用来给线性数据结构排序、二分查找。

```go
// 切片排序
a := []int{-1, 23, 5, 9, 7}
// sort.Sort(sort.IntSlice(a)) // 默认升序，有快捷写法Ints
sort.Ints(a)   // 就地修改原切片的底层数组
fmt.Println(a) // 默认升序

// 降序 sort.IntSlice(a)强制类型转换以施加接口方法
sort.Sort(sort.Reverse(sort.IntSlice(a)))
fmt.Println(a)
```

```go
// 二分查找
a := []int{-1, 23, 5, 9, 7}
sort.Ints(a)

// 二分查找，必须是升序
// 二分查找的前提是 有序
i := sort.SearchInts(a, 7)
fmt.Println(i)
```

思考：什么是相同的key？冲突的key有什么异同?

有冲突的key就是相同的key吗？也就是说，如果2个key计算的hash值相同就是同一个key吗？key计算的hash值相同只能说明hash冲突，如果key也相等，才能说明是用一个key。同一个key计算的hash值一定冲突，但是hash冲突不一定是同一个key。

马哥教育
IT人的高薪职业学院