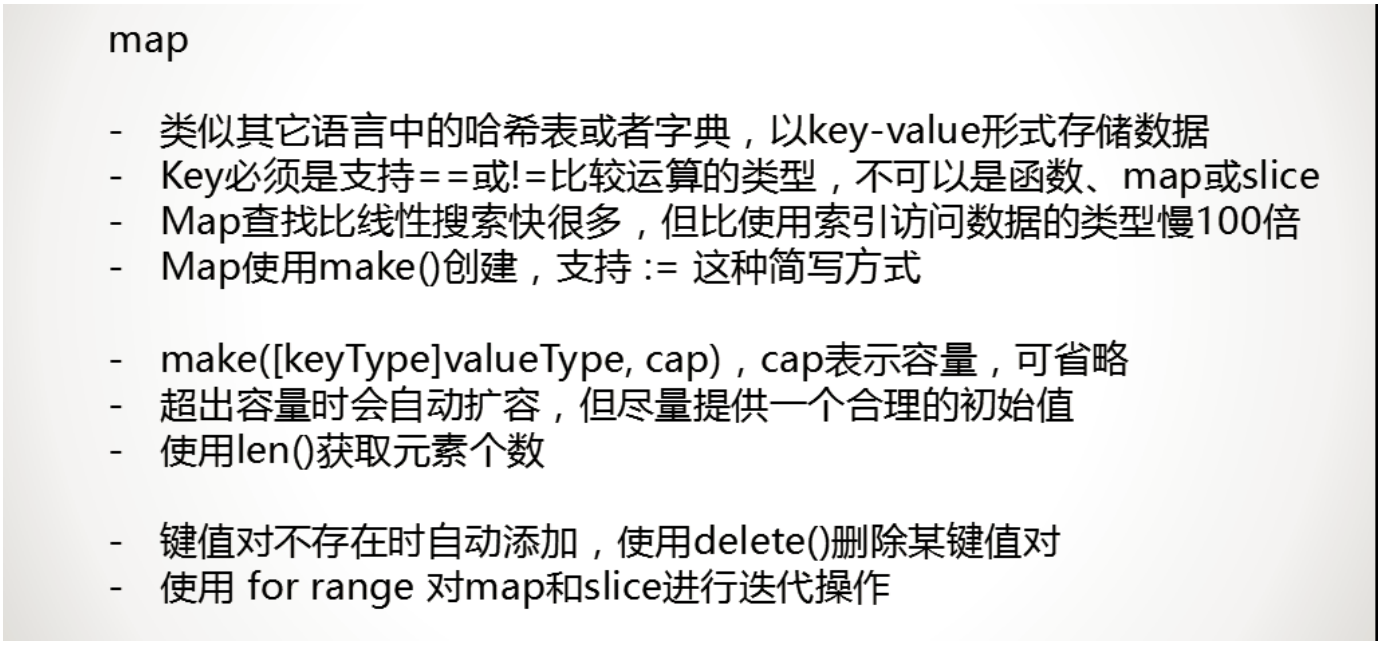


map

- 类似其它语言中的哈希表或者字典，以key-value形式存储数据
- Key必须是支持==或!=比较运算的类型，不可以是函数、map或slice
- Map查找比线性搜索快很多，但比使用索引访问数据的类型慢100倍
- Map使用make()创建，支持 := 这种简写方式

map的创建

第一种形式：和声明变量是非常相似的，首先使用**map**关键字，然后是一对中括号[]，中括号括起来的是 Key 的类型，最后紧接着是 value 的类型

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      var m map[int]string     /*首先使用map关键字，然后是一对中括号[]，中括号括起来的是 K
9.                               最后紧接着是 value 的类型 */
10.     m = map[int]string{}     //对m变量经行map的初始化
11.     fmt.Println(m)           //打印m变量的初始化值
12. }
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
map[]
成功: 进程退出代码 0.
```

这个m就是一个空的map[]

也可以使用**make()**函数

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]string /*首先使用map关键字，然后是一对中括号[]，中括号括起来的是 Key
    最后紧接着是 value 的类型 */
    m = make(map[int]string) //使用make()函数对m变量经行map的初始化
    fmt.Println(m)           //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
map[]
成功: 进程退出代码 0.
```

简写也可以这么做：

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]string = make(map[int]string) /* 可以把map[int]string 看成type。
    中括号括起来的是 Key 的类型,再然后紧接着是 value 的类型，
    最后使用make()函数对m变量经行map的初始化 */
    fmt.Println(m) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
map[]
成功: 进程退出代码 0.
```

还可以使用更简单的方式创建map

```go
package main

import (
    "fmt"
)

func main() {
    m := make(map[int]string) /*使用make()函数对 map 关键字，然后是一对中括号[]，
    中括号 [] 括起来的是 Key 的类型,再然后紧接着是 value 的类型，最后m变量经行map的初始化
    fmt.Println(m) //打印m变量的初始化值
```

```
11.  }
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
map[]
成功: 进程退出代码 0.
```

然后学习怎么使用map

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      m := make(map[int]string) /*使用make()函数对 map 关键字，然后是一对中括号[],
9.      中括号 [] 括起来的是 Key 的类型,再然后紧接着是 value 的类型，最后m变量经行map的初始化
10.     m[1] = "OK"      //通过key对value的存取，将 OK 存到 Key 值为1的键值(key-value)当中
11.     fmt.Println(m) //打印m变量的初始化值
12. }
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
map[1:OK]
成功: 进程退出代码 0.
```

显然我们成功的存进去了 map m呢，就有了key值1，键值对

那么如果要取出来这个值，怎么取？

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  func main() {
8.      m := make(map[int]string) /*使用make()函数对 map 关键字，然后是一对中括号[],
9.      中括号 [] 括起来的是 Key 的类型,再然后紧接着是 value 的类型，最后m变量经行map的初始化
10.     m[1] = "OK" //通过key对value的存取，将 OK 存到 Key 值为1的键值(key-value)当中
11.     a := m[1]   //取出map m的key为1的键值对的值
12.     fmt.Println(a) //打印m变量的初始化值
13. }
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
OK
成功: 进程退出代码 0.
```

如果没有这个值

```
package main

import (
    "fmt"
)

func main() {
    m := make(map[int]string) /*使用make()函数对 map 关键字，然后是一对中括号[],
    中括号 [] 括起来的是 Key 的类型,再然后紧接着是 value 的类型，最后m变量经行map的初始化
    a := m[1]       //取出map m的key为1的键值对的值
    fmt.Println(a) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]

成功: 进程退出代码 0.
```

所以a的值就是空的

如果我增加了键值对，中途不想要了，可以使用**delete()**函数来操作，首先要输入操作的map名称，然后要出入它的这个键key 。我们这里的键类型是int类型，所以输入1，要删除的map key的值是1

```
package main

import (
    "fmt"
)

func main() {
    m := make(map[int]string) /*使用make()函数对 map 关键字，然后是一对中括号[],
    中括号 [] 括起来的是 Key 的类型,再然后紧接着是 value 的类型，最后m变量经行map的初始化
    m[1] = "OK"     //通过key对value的存取，将 OK 存到 Key 值为1的键值(key-value)当中
    delete(m, 1)    //首先要输入操作的map名称，然后要出入它的键key 。我们这里的键类型是in
    a := m[1]       //取出map m的key为1的键值对的值
    fmt.Println(a) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]

成功: 进程退出代码 0.
```

a输出为空，说明map m键值1的值被删除了

复杂的map操作:

怎么样才算一个复杂的map呢？

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string
    m = make(map[int]map[int]string)
    m[1] = "OK"
    a := m[1]
    fmt.Println(a) //打印m变量的初始化值
}
```

```
/usr/local/go/bin/go build -i [/home/jiemin/code/GOlang/go/src/map]
# _/home/jiemin/code/GOlang/go/src/map
./map.go:10: cannot use "OK" (type string) as type map[int]string in assignment
错误: 进程退出代码 2.
```

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string
    m = make(map[int]map[int]string)
    m[1][1] = "OK"
    a := m[1][1]
    fmt.Println(a) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
panic: assignment to entry in nil map

goroutine 1 [running]:
panic(0x4902a0, 0xc420070190)
        /usr/local/go/src/runtime/panic.go:500 +0x1a1
main.main()
        /home/jiemin/code/GOlang/go/src/map/map.go:10 +0x117
错误: 进程退出代码 2.
```

它告诉我们 试图将一个值赋给new。也就是我们这个map，它这时候还没有被成功的初始化，可是已经成功使用make()了，为什么它没有被成功的初始化呢？这里可以看到这里的make()只是将

最外层的map进行了初始化，这里面的 value 还是处于一个未被初始化的状态，在这里我们先要对这个map进行一个初始化。

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    m[1] = make(map[int]string)     // 初始化make(map[int]map[int]string)语句里面val
    m[1][1] = "OK"                   // 给m变量[1][1]赋值value为 OK
    a := m[1][1]                     // a变量取出来m[1][1]的键对值
    fmt.Println(a)                   //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
OK
成功: 进程退出代码 0.
```

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    m[1] = make(map[int]string)      // 初始化上面map语句里面 value 的map[int]string
    m[2][1] = "OK"                   // 给m变量[1][1]赋值value为 OK
    a := m[2][1]                     // a变量取出来m[1][1]的键对值
    fmt.Println(a)                   //打印m变量的初始化值
}
```

已经进行了第2级的map的初始化，在来取这个值，执行：

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
panic: assignment to entry in nil map

goroutine 1 [running]:
panic(0x4902a0, 0xc42000a340)
        /usr/local/go/src/runtime/panic.go:500 +0x1a1
main.main()
        /home/jiemin/code/GOlang/go/src/map/map.go:12 +0x18e
错误: 进程退出代码 2.
```

怎么又来说map 没有进行初始化？很显然这里我们使用1的时候，只对这个key为1的键值对中的map进行了初始化。而使用了key为2的键值对，这时候key为2的键值对中的map没有进行初始化，所以说，它还是会提示我们试图将一个值赋给new的map，这个时候怎么办呢？我们怎么知道它有没有被初始化呢？这时候有俩种方法：
第一种方式就是我们先来取这个值，返回一个空的字符串。这个时候就表示它没有初始化。

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    a := m[2][1]                     // a变量取出来m[2][1]的键对值
    fmt.Println(a)                   //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]

成功: 进程退出代码 0.
```

但是这种方法不是很保险，万一我真的存了空的东西怎么办呢？
这时候就需要使用多返回值的优势了，那么多返回值它会返回的什么呢？当我们只有一个返回值的时候，它会返回对应的value。当我们有俩个返回值的时候，它的第二个返回值是一个bool类型的。它就会告诉你这个键值对是否存在。

```go
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    a, ok := m[2][1]                 /*使用多返回值的方法：第1个返回值，它会返回对应的v
    第2个返回值，它会返回是一个bool类型的(true/false)*/
```

```
    fmt.Println(a, ok) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
false
成功: 进程退出代码 0.
```

它这里输出一个 false，表示这一对键值对是不存在的。这个时候有什么好处呢？那这里就可以来一个 if 语句，if 取反

```
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    a, ok := m[2][1]                 /*使用多返回值的方法：第1个返回值，它会返回对应的v
    第2个返回值，它会返回是一个bool类型的(true/false)*/
    if !ok { // if语句取反 ! ok
        m[2] = make(map[int]string) // 把第2个map进行初始化
    }
    m[2][1] = "GOOD"    // 赋值给第2个map的键对值
    a = m[2][1]         // a变量取第2个map的键对值
    fmt.Println(a, ok) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
GOOD false
成功: 进程退出代码 0.
```

为什么是 false，因为是取上面的ok返回值。

```
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    a, ok := m[2][1]                 /*使用多返回值的方法：第1个返回值，它会返回对应的v
    第2个返回值，它会返回是一个bool类型的(true/false)*/
    if !ok { // if语句取反 ! ok
        m[2] = make(map[int]string) // 把第2个map进行初始化
```

```
    }
    m[2][1] = "GOOD" // 赋值给第2个map的键对值
    a = m[2][1]      // a变量取第2个map的键对值
    fmt.Println(a)   //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
GOOD
成功: 进程退出代码 0.
```

如果在取一次ok的话

```
package main

import (
    "fmt"
)

func main() {
    var m map[int]map[int]string /* 创建一个复杂的map，map类型为int，
    value为一个int型的map */
    m = make(map[int]map[int]string) // 初始化map[int]string
    a, ok := m[2][1]                 /*使用多返回值的方法：第1个返回值，它会返回对应的v
    第2个返回值，它会返回是一个bool类型的(true/false)*/
    if !ok { // if语句取反 ! ok
        m[2] = make(map[int]string) // 把第2个map进行初始化
    }
    m[2][1] = "GOOD"   // 赋值给第2个map的键对值
    a, ok = m[2][1]    // a变量取第2个map的键对值
    fmt.Println(a, ok) //打印m变量的初始化值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
GOOD true
成功: 进程退出代码 0.
```

结果值就变为 true。这就是当有多个map，map嵌套map的时候要注意的事项。每一级的map，都要进行单独的初始化。否则的话就出现这种运行时候的错误，这种错误在编译的时候是没有办法发现的，同样的道理，我们有更复杂的三层的map嵌套的话呢，我们就需要进行三次的make()，那么每一次最好都要进行每一次检查，如果说是不存在的话，就要先进行一次make()，不然进行赋值操作就会发生异常。

迭代操作：

就会设置到slice，slice和map都可以进行迭代操作。那么它是一个什么的形式呢？
首先是一个 **for 关键字**，然后会配合 **range 关键字**来使用，它这里有一个语句。
它的一般形式是这样的： for i, v := range slice {} / for k, v := range map {}

如果说我们对这个slice进行一个迭代的话，它会返回俩个值。第1个值就是相对应slice的索引，slice的索引就是 1 2 3 4 5 6 7 。这个时候 i 就是一个计数器，它的类型是int型；第2个值v 是slice当中所存储的值，它就会取出来这个值，赋值给这个v，需要注意的是v拿到的这个值是slice的值的拷贝。所以对这个v进行任何修改都不会影响这个slice的本身。如果想要修改slice的本身，应该怎么做呢？就可以使用这种形式：已经取出这个索引了，就可以使用这个索引来对slice本身来进行操作。

那么当我们把slice替换成为map的时候，map没有索引，那么 i 变成什么呢？当然这个 i 就变成了 k 了。所以对这个map进行迭代操作，它就会返回一个键值对。同样你对键值对进行任何操作，都是一个拷贝的操作，都不会对这个map本身起到任何作用。如果要对这个map的键值对进行操作的时候，就需要通过这个 k 来直接操作这个map。

```go
package main

import (
    "fmt"
)

func main() {
    sm := make([]map[int]string, 5) //以map为元素类型的slice,以及指定slice的长度5 需要
    for _, v := range sm {          // 对slice迭代操作,不需要使用索引,使用_忽略掉.只是
        v = make(map[int]string, 1) // 使用make()函数对slice当中的map进行初始化操作
        v[1] = "OK"                 // 赋值
        fmt.Println(v)              // 打印变量v的值
    }
    fmt.Println(sm) // 打印slice的值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
map[1:OK]
map[1:OK]
map[1:OK]
map[1:OK]
map[1:OK]
[map[] map[] map[] map[] map[]]
成功: 进程退出代码 0.
```

在迭代当中明显看到map这个值已经被初始化了。可是我们在最后打印map看到的又是空的map，这是怎么回事呢？这就是刚才说的 v 是一个拷贝，对这个v进行任何操作都不会影响到slice本身，那么咋办呢？显然我就不需要这个v了，我就只需要一个 i 就行。这个时候就需要使用这种：

```go
package main

import (
    "fmt"
)

func main() {
    sm := make([]map[int]string, 5)//以map为元素类型的slice,以及指定slice的长度5, 需要
    for i := range sm {             // 对slice迭代操作,对slice本身进行操作，就需要使用
```

```
        sm[i] = make(map[int]string, 1) // slice需要使用i索引来对slice当中的map进行初
        sm[i][1] = "OK"                 // 赋值
        fmt.Println(sm[i])              // 打印对应map的值
    }
    fmt.Println(sm) // 打印slice的值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map [/home/jiemin/code/GOlang/go/src/map]
map[1:OK]
map[1:OK]
map[1:OK]
map[1:OK]
map[1:OK]
[map[1:OK] map[1:OK] map[1:OK] map[1:OK] map[1:OK]]
成功: 进程退出代码 0.
```

在迭代当中，这个map进行了初始化。迭代结束之后在来打印这个slice，所有的map都被成功的初始化，所以说这里需要注意，这个里value得到的是一个拷贝，并不是一个真实的值。如果一定要对这个slice进行操作的话， 被迭代对象进行操作的话，就必须利用这个索引或者key来对被迭代对象进行直接操作，才能影响到它的本身。不然的话，都只是一个拷贝的操作。任何操作都是没有意义的。

对map的间接排序：

map是无序的，不能进行一个直接排序。我们可以通过对它一个key操作进行一个间接排序，这里还是需要配合使用一个slice，那么怎么进行一个排序。

先创建一个int类型的map，给它赋值字面值。使用len()函数来对map进行计算元素的个数之后使用make()函数创建int类型的slice和slice的容量个数。就需要对map进行迭代操作，因为map没有索引i，只有key的k，就只能直接创建一个变量i的计数器来替代slice的索引i，进行对map迭代操作，使用slice的 i 等于map的k 值。让变量 i 的计数器每次递增1，就把所有的key存到了slice中。最后打印slice的值。

```
package main

import (
    "fmt"
)

func main() {
    // 创建int类型的map，对map赋值给它字面值
    m := map[int]string{1: "a", 2: "b", 3: "c", 4: "d", 5: "e"}
    s := make([]int, len(m)) // 使用make()函数创建int类型的slice，容量为map的元素个数
    i := 0                   // 创建一个变量i为计数器
    for k, _ := range m {    // 对map 进行迭代操作
        s[i] = k // 使用slice的int类型索引等值map的int类型key
        i++      // 每次+1,这样的操作就让所有的key都存到slice当中
    }
    fmt.Println(s) // 打印slice的值
}
```

```
/usr/local/go/bin/go build -i [/home/jiemin/code/GOlang/go/src/map]
成功: 进程退出代码 0.
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
[2 3 4 5 1]
成功: 进程退出代码 0.
```

但是每次打印出来的slice的值是无序的。怎么对slice的值弄成有序的呢？需要使用**sort包**

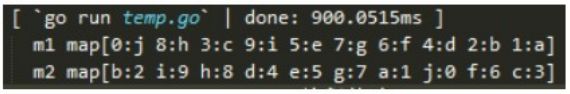
```go
package main

import (
    "fmt"
    "sort"
)

func main() {
    // 创建int类型的map，对map赋值给它字面值
    m := map[int]string{1: "a", 2: "b", 3: "c", 4: "d", 5: "e"}
    s := make([]int, len(m)) // 使用make()函数创建int类型的slice，容量为map的元素个数
    i := 0                   // 创建一个变量i为计数器
    for k, _ := range m {    // 对map 进行迭代操作
        s[i] = k // 使用slice的int类型索引等值map的int类型key
        i++      // 每次+1,这样的操作就让所有的key都存到slice当中
    }
    sort.Ints(s)   //使用sort包中的Ints对slice进行排序
    fmt.Println(s) // 打印slice的值
}
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
[1 2 3 4 5]
成功: 进程退出代码 0.
```

导入**sort包**，使用**sort.Ints()**对slice进行排序。 Ints是对int类型进行排序，因为key是int类型的。传入slice，通过使用**sort.Ints()**的调用发现这里的slice是引用类型的。因为并没有返回，直接slice传递进去之后它就会进行一个排序。这个排序它只对slice这个本身进行操作。这样就完成了对map当中的key进行一个排序，这个时候就可以根据key进行有序的取出map的value的值。这样就是对一个map进行一个间接的排序。

课堂作业

- 根据在 for range 部分讲解的知识，尝试将类型为map[int]string的键和值进行交换，变成类型map[string]int
- 程序正确运行后应输出如下结果：



[`go run temp.go` | done: 900.0515ms]
m1 map[0:j 8:h 3:c 9:i 5:e 7:g 6:f 4:d 2:b 1:a]
m2 map[b:2 i:9 h:8 d:4 e:5 g:7 a:1 j:0 f:6 c:3]

作业：

```go
package main

import (
    "fmt"
)

func main() {
    m1 := map[int]string{0: "j", 8: "h", 3: "c", 9: "i", 5: "e", 7: "g", 6: "f", 4:
    fmt.Println(m1)
    m2 := make(map[string]int)

    for k, v := range m1 {
        m2[v] = k
    }
    fmt.Println(m2)
}
```

```
/home/jiemin/code/GOlang/go/src/map/map  [/home/jiemin/code/GOlang/go/src/map]
map[1:a 3:c 5:e 7:g 2:b 4:d 0:j 8:h 9:i 6:f]
map[c:3 e:5 g:7 b:2 h:8 f:6 a:1 j:0 i:9 d:4]
成功: 进程退出代码 0.
```