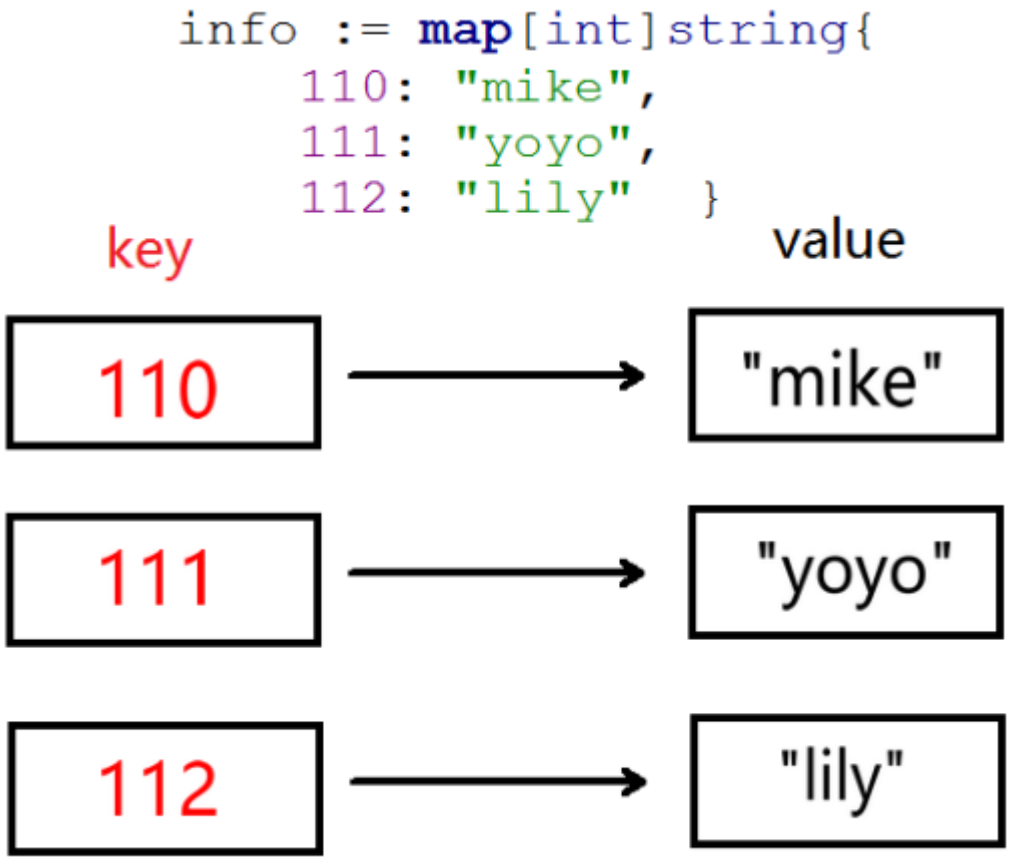
go+区块链培训 讲师:张长志

Map

Go语言中的map(映射，字典)是一种内置的数据结构，它是一个**无序**的key-value对的集合，比如以身份证号作为唯一key来标识一个人的信息

例如：

110-----mike 111--jack 123-----marry



map 的基本使用

map的定义格式 map【**keytype**】valuetype

在一个map里所有的键都是唯一的，而且必须是支持==和!=操作符的类型，**切片、函数以及包含切片的结构类型**这些类型由于具有引用语义，不能作为**映射的键**，使用这些类型会造成编译错误：

dict := map[[]string]int{} //err, invalid map key type []string

案例：

```
import "fmt"

func main(){
   //定义一个map变量，类型为map[int]string,如果使用map一定初始化，make分配空间
   var m1 map[int]string = make(map[int]string)
   fmt.Println("m1=",m1)
   m1[1]= "jake"
   fmt.Println("m1=",m1)

   m2 := make(map[int]string) //make方式创建然后自动推导类型
   fmt.Println("len=",len(m2))
   m2[1] = "mike"
   fmt.Println("m2=",m2)
   fmt.Println("len=",len(m2))

   //map 先给map指定一个可以容纳长度，一旦超过这个长度 从新分配底层空间
   //
   m3 := make(map[int]string,2)
   m3[1]="mile"
   m3[2]="jack"
   m3[3]="go"
   fmt.Println("m3=",m3)
   fmt.Println("len=",len(m3))

   //map另一种初始化并且赋值
   m4 := map[int]string{1:"mike",2:"jack",3:"go"}
   fmt.Println("m4=",m4)
}
```

map赋值

```
import "fmt"

func main(){
   m1 := map[int]string{1:"mike",2:"jack"}
   fmt.Println("m1=",m1)

   m1[1]="marry"  //如果key存在，修改内容
   m1[3]="tom"    //如果没有key 追加内容，切片append
   fmt.Println("m1=",m1)
}
```

map的遍历

```
import "fmt"

func main(){
   m := map[int]string{1:"mike",2:"jack",3:"tom"}
    //第一个返回值为key 第二个返回值为value 遍历结构是无序的
   for key,value :=range m{
      fmt.Printf("%d======>%s\n",key,value)
   }

   //如何判断一个key是否存在，
   //第一个返回值为key的所对应的value，第二个返回值为key是否存在的条件，如果存在ok为true
   //value,ok := m[1]
    if value,ok := m[1];ok == true{
       fmt.Println("m[1]=",value)
   }else {
      fmt.Println("key 不存在")
   }


}
```

map的删除

```
delete(m,2) //删除key为2的内容
fmt.Println("m=",m)
```

map 做函数参数

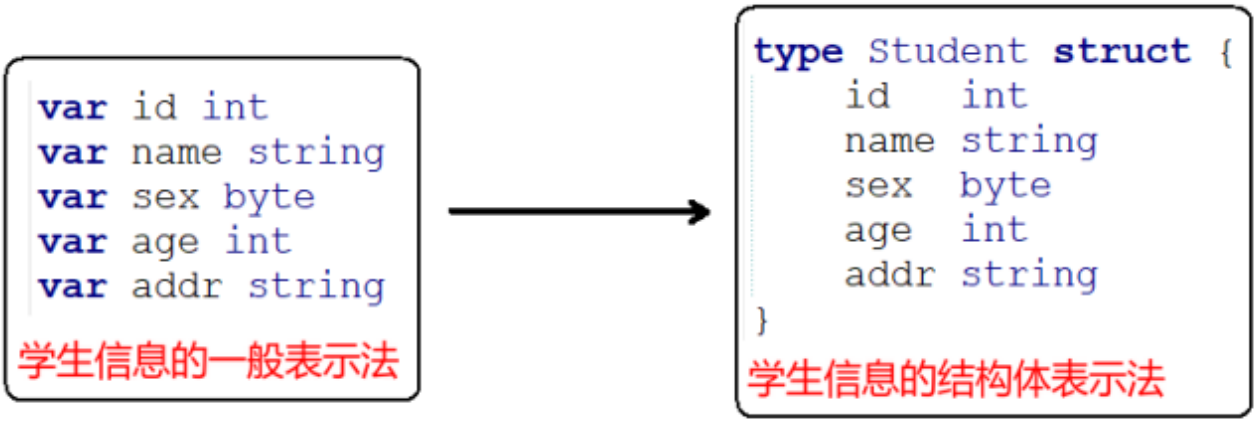
```
package main

import "fmt"

func test(m map[int]string)  { //map 和我们切片一样 他们是引用类型
   delete(m,1)
}

func main(){
   m :=map[int]string{1:"jack",2:"mike",3:"marry"}
   fmt.Println("m=",m)

   test(m)

   fmt.Println("m=",m)

}
```

map做为函数参数它和slice切片一样 都是引用类型

结构体类型

id name sex age addr

有时我们需要将不同类型的数据组合成一个有机的整体，如：一个学生有学号/姓名/性别/年龄/地址等属性。显然单独定义以上变量比较繁琐，数据不便于管理。



结构体是一种聚合的数据类型，它是由一系列具有相同类型或不同类型的数据构成的数据集合。每个数据称为结构体的成员。

结构体初始化

```
import (
   "fmt"
)

//定义一个结构体
type  Student  struct{
   id  int
   name string
   sex byte //字符类型
   age int
   addr string
}

func main(){
  //顺序初始化，每个成员必须初始化
   var  s1 Student =Student{1,"mike",'M',18,"bj"}
   fmt.Println("s1=",s1)
   //指定成员初始化，没有初始化的成员，自动赋值为0
   s2 := Student{name:"mike",addr:"bj"}
```

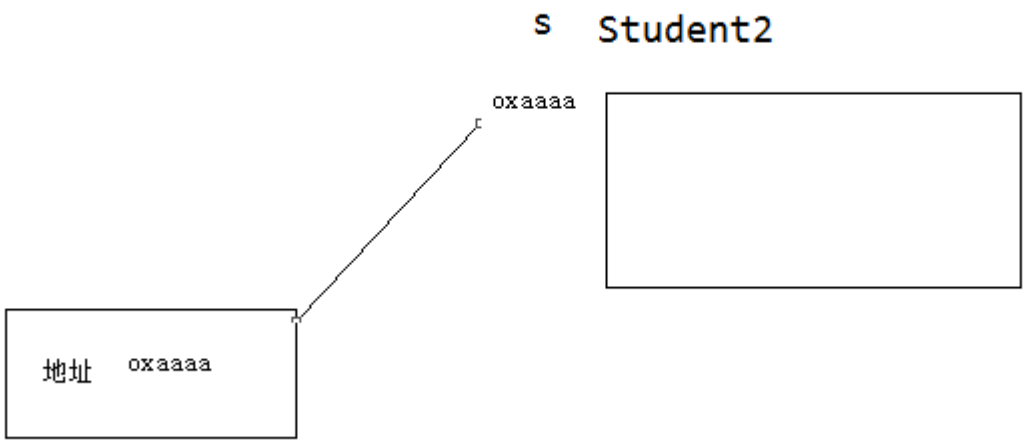
```
    fmt.Println("s2=",s2)
}
```

结构体指针类型初始化

```
func main(){
    //p1 存放是地址Person
    var p1 *Person = &Person{1,"mike",'m',18,"bj"}
    fmt.Printf("p1=%v\n",p1)

    p2 := &Person{name:"mike",addr:"bj"}
    fmt.Printf("p2 type is %T\n",p2)
    fmt.Println("p2=",p2)

}
```

结构体通过. 运算符操作成员

```
//定义一个结构体
type  Student  struct{
    id  int
    name string
    sex byte //字符类型
    age int
    addr string
}

func main(){
  /*//顺序初始化，每个成员必须初始化
    var  s1 Student =Student{1,"mike",'M',18,"bj"}
    fmt.Println("s1=",s1)
    //指定成员初始化，没有初始化的成员，自动赋值为0
    s2 := Student{name:"mike",addr:"bj"}
    fmt.Println("s2=",s2)*/

    //定义一个结构体普通变量
    var  s Student
    //操作成员变量，需要用.
    s.id = 1
    s.name="mike"
    s.sex='m'
    s.age=18
    s.addr="bj"

    fmt.Println("s=",s)

}
```



指针变量操作

```
package main

import "fmt"

type Student2 struct {
   id int
   name string
   sex byte
   age int
   addr string

}

func main(){
   //先定义一个普通变量
  var s Student2
  //定义一个指针变量
  var p1 *Student2
  p1 = &s

  p1.id = 1
  (*p1).name = "jack"
  p1.sex = 'm'
  p1.age = 18
  p1.addr = "bj"
  fmt.Println("s=",s)

  p2 := new(Student2) //new 地址
  p2.id = 1
```

```
  p2.name = "mike"
  p2.sex = 'M'
  p2.age = 18
  p2.addr ="bj"
  fmt.Println("p2=",p2)



}
```

结构体的比较和赋值

```
package main

import "fmt"

type Student3 struct {
  id int
  name string
  sex byte
  age int
  addr string
}

func main(){
   s1 := Student3{1,"mike",'m',18,"bj"}
    s2 := Student3{1,"mike",'m',18,"bj"}
   s3 := Student3{2,"mike",'m',18,"bj"}
   fmt.Println("s1==s2",s1==s2)
   fmt.Println("s1==s3",s1==s3)

   //同类型的2个结构体变量是可以互相赋值的
   var tmp Student3
   tmp = s1
   fmt.Println("tmp=",tmp)

}
```

结构体作为函数参数

普通结构体参数 与指针结构体参数

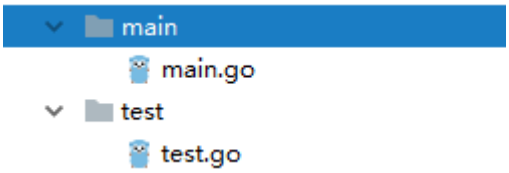
```
import "fmt"

type Student4 struct {
```

```
    id int
    name string
    sex byte
    age int
    addr string
}

func test01(s Student4){
    s.id = 666
    fmt.Println("test01:",s)
}

func test02(s *Student4){
    s.id = 666
    fmt.Println("test01:",s)
}



func main(){
    s := Student4{1,"mike",'m',18,"bj"}
    test01(s)//值传递
    fmt.Println("main:",s)

    test02(&s)//地址传递（引用传递），形参可以改变实参
    fmt.Println("main:",s)


}
```

可见性

Go语言对关键字的增加非常吝啬，其中没有private、 protected、 public这样的关键字。

要使某个符号对其他包（package）可见（即可以访问），需要将该符号定义为以大写字母 开头。



```go
package main

import (
   "LearnGoOnline/test"
   "fmt"
)

func main()  {
    s6 :=test.Student06{Id:1,Name:"mike"}
    fmt.Println("s6=",s6)

}
```

```go
package test

//student01只能在本文件件引用，因为首字母小写
type Student06 struct {
    Id int
    name string
}

//Student02可以在任意文件引用，因为首字母大写
type Student07 struct {
   id int
   name string
}
```

面向对象