**实例变量赋值给接口时的拷贝**

* 将type实例赋值给接口变量时，会发生值拷贝行为，接口内部存储的是指向这个复制品的指针。⽽且我们无法修改这个复制品的状态，也无法获取其指针。
* 将type 实例地址赋值给接口变量时，会发生实例地址值拷贝行为。接口内部存储的是指向这个实例地址复制品的指针。我们可以通过接口变量读取和修改其所指向的type 实例变量的值。（原因是通过指针传递参数，虽然发生拷贝，但是所有拷贝的指针都指向一个存储位置）

|  |
| --- |
| //源码  1 package main  2  3 import (  4 "fmt"  5 "reflect"  6 )  7  8 type User struct {  9 Id int  10 Name string  11 }  12  13 func (u User) Test (){  14 fmt.Println("ok")  15 }  16  17 type Tester interface{  18 Test()  19 }  20  22 func main(){  23 u := User{110,"tata"}  24 i := Tester(u)  25 var j Tester  26 j = &u  27  28 u.Id = 333  29 u.Name = "hi"  30 fmt.Println("u=",u,"i=",i,"j=",j)  31 j.(\*User).Id = 888  32 j.(\*User).Name = "ok"  33 fmt.Println("u=",u,"i=",i,"j=",j)  34  35 fmt.Println(reflect.TypeOf(i).String())  36 fmt.Println(reflect.TypeOf(u).String())  37 fmt.Println(reflect.TypeOf(j).String())  38 var m Tester  39 //fmt.Println(reflect.TypeOf(m).String())  40 fmt.Printf("%p\n",&m)  41 fmt.Printf("%#v\n",m)  42 //未初始化的接口变量是不能进行反射操作的，因为其itab 和 void \* data 没有初始化  43 //TypeOf(m)返回的是nil  44 //fmt.Println(reflect.TypeOf(m).String())  45 //fmt.Println(reflect.TypeOf(m). NumMethod())  46 fmt.Println(reflect.TypeOf(m))  47 } |
| 结果：  u= {333 hi} i= {110 tata} j= &{333 hi}  u= {888 ok} i= {110 tata} j= &{888 ok}  main.User  main.User  \*main.User  0x18400130  <nil>  <nil> |

**切记：**指针传递经过多少次value copy ，所有的拷贝还是指向同一个地方，通过任何一个拷贝都能修改其指向值。

* 接口是可被实例化的类型，不仅是语言上的约束规范；当我们创建接变量时，将会为其分配部分内存空间；当将type实例赋值时会给接口变量时才分配全部空间（包括实例变量值或指针的拷贝）和初始化全部数据结构。单单声明一个接口变量，而不对其赋值，是没有意义的。

|  |
| --- |
| *$GOROOT/src/pkg/runtime/runtime.h*  *178 struct Iface*  *179 {*  *180 Itab\* tab;*  *181 void\* data;*  *182 };*        *449 struct Itab*  *450 {*  *451 InterfaceType\* inter;*  *452 Type\* type;*  *453 Itab\* link;*  *454 int32 bad;*  *455 int32 unused;*  *456 void (\*fun[])(void);*  *457 };* |
| 上个例子中：  var m Tester 仅仅创建Iface 数据结构，但是 Itab 和 void \* data 都没有赋值（nil），此时变量m 没有任何意义，使用m 会引发运行时panic .  只有给m 赋值一个type 实例变量，才会初始化Itab 和 void \* data，接口变量才能真的使用。 |