Go语言_反射篇

2012-06-10 22:51 by 轩脉刃, 1158 阅读, 0 评论, 收藏, 纠

Go语言的基本语法的使用已经在前几篇陆陆续续学完了，下面可能想写一些Go的标准库的使用了。

先是reflect库。

reflect库的godoc在http://golang.org/pkg/reflect/

Type和Value

首先，reflect包有两个数据类型我们必须知道，一个是Type，一个是Value。

Type就是定义的类型的一个数据类型，Value是值的类型

具体的Type和Value里面包含的方法就要看文档了：

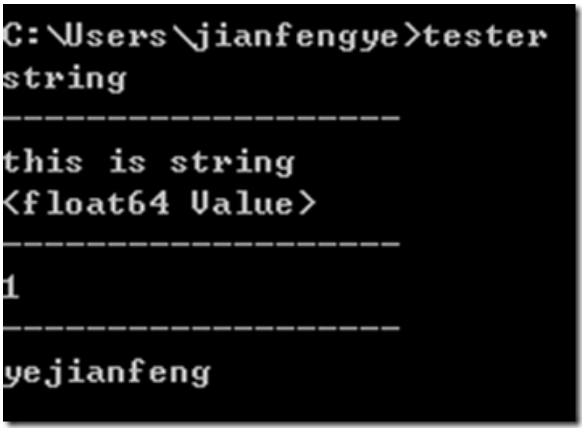
http://golang.org/pkg/reflect/

这里我写了个程序来理解Type和Value：

```
1  package main
2
3  import(
4      "fmt"
5      "reflect"
6  )
7
8  type MyStruct struct{
9      name string
10 }
11
12 func (this *MyStruct)GetName() string {
13     return this.name
14 }
15
16 func main() {
17     s := "this is string"
18     fmt.Println(reflect.TypeOf(s))
19     fmt.Println("--------------------")
20
21     fmt.Println(reflect.ValueOf(s))
22     var x float64 = 3.4
23     fmt.Println(reflect.ValueOf(x))
24     fmt.Println("--------------------")
25
```

```
26      a := new(MyStruct)
27      a.name = "yejianfeng"
28      typ := reflect.TypeOf(a)
29
30      fmt.Println(typ.NumMethod())
31      fmt.Println("--------------------")
32
33      b := reflect.ValueOf(a).MethodByName("GetName").Call([]reflect.Value{})
34      fmt.Println(b[0])
35
36  }
```

输出结果：



这个程序看到几点：

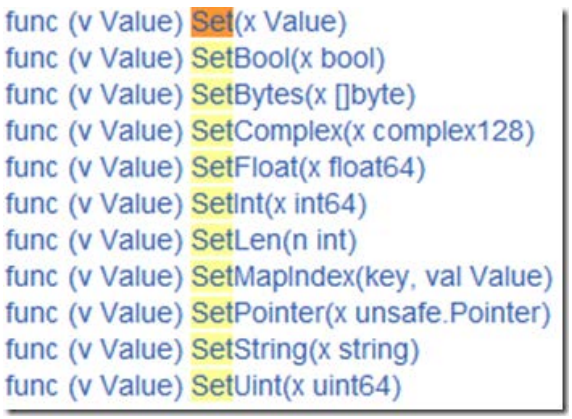
1 TypeOf和ValueOf是获取Type和Value的方法

2 ValueOf返回的<float64 Value>是为了说明这里的value是float64

3 第三个b的定义实现了php中的string->method的方法，为什么返回的是reflect.Value[]数组呢？当然是因为Go的函数可以返回多个值的原因了。

Value的方法和属性

好了，我们看到Value的Type定义了这么多Set方法：



下面看这么个例子：

```
package main

import(
    "fmt"
    "reflect"
)

type MyStruct struct{
    name string
}

func (this *MyStruct)GetName() string {
    return this.name
}

func main() {
    fmt.Println("---------------")
    var a MyStruct
    b := new(MyStruct)
    fmt.Println(reflect.ValueOf(a))
    fmt.Println(reflect.ValueOf(b))

    fmt.Println("---------------")
    a.name = "yejianfeng"
    b.name = "yejianfeng"
    val := reflect.ValueOf(a).FieldByName("name")

    //painc: val := reflect.ValueOf(b).FieldByName("name")
    fmt.Println(val)

    fmt.Println("---------------")
    fmt.Println(reflect.ValueOf(a).FieldByName("name").CanSet())
    fmt.Println(reflect.ValueOf(&(a.name)).Elem().CanSet())

    fmt.Println("---------------")
    var c string = "yejianfeng"
    p := reflect.ValueOf(&c)
    fmt.Println(p.CanSet())   //false
    fmt.Println(p.Elem().CanSet())  //true
    p.Elem().SetString("newName")
    fmt.Println(c)
}
```

返回：

```
C:\Users\jianfengye>go install tester

C:\Users\jianfengye>tester
--------------
<main.MyStruct Value>
<*main.MyStruct Value>
--------------
yejianfeng
--------------
false
true
--------------
false
true
newName
```

这段代码能有一些事情值得琢磨：

1 为什么a和b的ValueOf返回的是不一样的？

a是一个结构，b是一个指针。好吧，在Go中，指针的定义和C中是一样的。

2 reflect.ValueOf(a).FieldByName("name")

这是一个绕路的写法，其实和a.name是一样的意思，主要是要说明一下Value.FieldByName的用法

3 val := reflect.ValueOf(b).FieldByName("name") 是有error的，为什么？

b是一个指针，指针的ValueOf返回的是指针的Type，它是没有Field的，所以也就不能使用FieldByName

4 fmt.Println(reflect.ValueOf(a).FieldByName("name").CanSet())为什么是false?

看文档中的解释：

func (Value) CanSet

```
func (v Value) CanSet() bool
```

CanSet returns true if the value of v can be changed. A Value can be changed only if it is addressable and was not obtained by the use of unexported struct fields. If CanSet returns false, calling Set or any type-specific setter (e.g., SetBool, SetInt64) will panic.

好吧，什么是addressable，and was not obtained by the use of unexported struct fields?

CanSet当Value是可寻址的时候，返回true，否则返回false

看到第二个c和p的例子，我们可以这么理解：

当前面的CanSet是一个指针的时候（p）它是不可寻址的，但是当是p.Elem()(实际上就是*p)，它就是可以寻址的

这个确实有点绕。

总而言之，reflect包是开发过程中几乎必备的包之一。能合理和熟练使用它对开发有很大的帮助。