反射

```go
package main

import (
    "reflect"
    "fmt"
)

type Student struct {
    Name string
    Age  int
    Sex  byte
}

func info(obj interface{})  {
    o := reflect.TypeOf(obj)
    fmt.Println("type:",o.Name())
    val:= reflect.ValueOf(obj)
    fmt.Println("value:",val)

    len := o.NumField()
    for i := 0; i < len; i++ {
        field := o.Field(i)
        value := val.Field(i).Interface()
        fmt.Printf("%8s: %v %v\n", field.Name,field.Type,value)
    }

}
func main() {
    s := Student{"kobe",24,1}
    info(s)
}

// type: Student
// value: {kobe 24 1}
//     Name: string kobe
//      Age: int 24
//      Sex: uint8 1
```

反射方法

```go
package main
```

```go
import (
    "reflect"
    "fmt"
)

type Student struct {
    Name string
    Age  int
    Sex  byte
}

func (s Student) Say() {
    fmt.Println(s.Name)
}
func info(obj interface{}) {
    o := reflect.TypeOf(obj)
    fmt.Println("type:", o.Name())
    val := reflect.ValueOf(obj)
    fmt.Println("value:", val)

    len := o.NumField()
    for i := 0; i < len; i++ {
        field := o.Field(i)
        value := val.Field(i).Interface()
        fmt.Printf("%8s: %v %v\n", field.Name, field.Type, value)
    }

    for i := 0; i < o.NumMethod(); i++ {
        method := o.Method(i)
        fmt.Printf(" %v %v\n", method.Name, method.Type)
    }
}

func main() {
    s := Student{"kobe", 24, 1}
    info(s)
}
```

注意，本实例无法反射非struct，如果传入指针将会编译错误。我们可以通过判断反射后的类型来过滤非struct

```go
if k := o.Kind(); k != reflect.Struct {
    fmt.Println("non struct type")
    return
}
```

如何反射匿名或嵌套组合字段

```
package main

import (
    "reflect"
    "fmt"
)

type Student struct {
    Name string
    Age  int
    Sex  byte
}

type Teacher struct {
    Student
    Class string
}

func (s Student) Say() {
    fmt.Println(s.Name)
}

func info(obj interface{}) {
    t := reflect.TypeOf(obj)
    fmt.Println(t.Field(0))
    fmt.Printf("%v\n", t.Field(0))
    fmt.Printf("%#v", t.Field(0))
}

func main() {
    teacher := Teacher{Student{"zhangsan", 20, 1}, "english"}
    info(teacher)
}

// {Student  main.Student  0 [0] true}
// {Student  main.Student  0 [0] true}
// reflect.StructField{Name:"Student", PkgPath:"", Type:(*reflect.rtype)(0x1
0a6020), Tag:"", Offset:0x0, Index:[]int{0}, Anonymous:true}
```

注意

1. 格式化字符串 `%#v`，可以通过官网fmt文档查询到使用方法。
2. Anonymous:true 我们反射了匿名嵌套字段User，之所以为匿名字段，这里一目了然。

3. 匿名字段其实是只有类型的,其字段名称由于和类型一样，故省略，一般在嵌套组合字段中常用。

```
type A struct {
    Name string
    User User
    Teacher // 字段名称省略 Teacher Teacher
}
```

1. 如何获取嵌套类型的具体字段-- 使用FieldByIndex方法和切片

```
fmt.Printf("%#v\n", t.FieldByIndex([]int{0,0}))
fmt.Printf("%#v\n", t.FieldByIndex([]int{0,1}))
fmt.Printf("%#v\n", t.FieldByIndex([]int{0,2}))

// reflect.StructField{Name:"Name", PkgPath:"", Type:(*reflect.rtype)(0x1097
9c0), Tag:"", Offset:0x0, Index:[]int{0}, Anonymous:false}
// reflect.StructField{Name:"Age", PkgPath:"", Type:(*reflect.rtype)(0x10974
40), Tag:"", Offset:0x10, Index:[]int{1}, Anonymous:false}
// reflect.StructField{Name:"Sex", PkgPath:"", Type:(*reflect.rtype)(0x1097b
40), Tag:"", Offset:0x18, Index:[]int{2}, Anonymous:false}
```

通过反射修改值

`reflect.ValueOf(i interface{})` 返回值为Value类型，需要取出其类型对应的底层值，我们传递指针，并且通过Elem()方法获取到底层对象，然后设置修改值。

```
x := 123
xx:= reflect.ValueOf(&x) // 返回类型为Value
//xx =192 // can not use 192(type untyped int) as type Value in assignment
xx.Elem().SetInt(192)
fmt.Printf("%v", x)
```

修改更为复杂的值
```Go
package main

import (
"reflect"
"fmt"
)
```

```
type Student struct {
Name string
Age int
Sex byte
}

type Teacher struct {
Student
Class string
}

func (s Student) Say() {
fmt.Println(s.Name)
}

func main() {
stu := Student{"kobe", 24, 1}
Set(&stu)
fmt.Println(stu)
}

func Set(o interface{}) {
v := reflect.ValueOf(o)
if v.Kind() == reflect.Ptr && !v.Elem().CanSet() {
fmt.Printf("xxx")
return
}
v = v.Elem()
nameField := v.FieldByName("Name")
if !nameField.IsValid() {
fmt.Println("bad param")
return
}
```

```
if nameField.Kind() == reflect.String {
    nameField.SetString("kobe24")
}
```

```
}
```

~~~~
### 反射方法进行调用

```Go
package main

import "fmt"

type User struct {
    Name string
    Age int
    Sex byte
}

func (u User) SayHello(name string) {
    fmt.Println("hello",name,"my name is",u.Name)
}

func main() {
    u := User{"kobe",24,1}
    v := reflect.ValueOf(u)
    m := v.MethodByName("SayHello")
    args := []reflect.Value{reflect.ValueOf("james")}
    m.Call(args)
}
~~~~