Go语言 标准库 包

go语言提供了一种机制，在编译时不知道类型的情况下，可更新变量，在运行时查看值，调用方法以及直接对他们的布局进行操作。这种机制称为反射(reflection)。

反射是指在程序运行期对程序本身进行访问和修改的能力。程序在编译时，变量被转换为内存地址，变量名不会被编译器写入到可执行部分。在运行程序时，程序无法获取自身的信息。

支持反射的语言可以在程序编译期将变量的反射信息，如字段名称、类型信息、结构体信息等整合到可执行文件中，并给程序提供接口访问反射信息，这样就可以在程序运行期获取类型的反射信息，并且有能力修改它们。

为什么使用反射?

有时候我们需要写一个函数有能力统一处理各种值类型的函数，而这些类型可能无法共享同一个接口，也可能布局未知，也有可能这个类型在我们设计函数时还不存在。甚至这个类会同时存在上面三个问题。这时你会发现，如果类型很多或者更多不确定的类型就会很麻烦。所以我们引入了reflect包。

Go语言中的反射是由 reflect 包提供支持的，它定义了两个重要的类型 Type 和 Value 任意接口值在反射中都可以理解为由 reflect.Type 和 reflect.Value 两部分组成，并且 reflect 包提供了 reflect.TypeOf 和 reflect.ValueOf 两个函数来获取任意对象的 Value 和 Type。

Type反射的类型对象

这是一个接口，真正使用该接口的实例是reflect.rtype，该实例持有动态类型的所有信息。

kind

kind() 获取具体类型的底层类型。

Elem

Elem()这个方法返回的是原始变量的元素的类型。

Value

这是一个struct,持有动态值的所有信息。

Type

Type方法返回接口变量的动态类型信息，也就是传入ValueOf方法的原始变量的类型。

Kind

与Type的kind方法一样，返回的是原始类型。

Interface

把一个reflect.Value对象还原回一个空接口类型的变量，可以通过类型断言：x, ok := v.Interface().(int)

Elem

调用该方法的Value对象。

示例：

```
package main

import (
    "fmt"
    "reflect"
)

type myInt int64

func reflectType(x interface{}) {
    t := reflect.TypeOf(x)
    fmt.Printf("type:%v kind:%v\n", t.Name(), t.Kind())
}

func main() {
    var a *float32 // 指针
    var b myInt    // 自定义类型
    var c rune     // 类型别名//代表int32
    reflectType(a) // type: kind:ptr
    reflectType(b) // type:myInt kind:int64
    reflectType(c) // type:int32 kind:int32

    type person struct {
        name string
        age  int
    }
        var d = person{
        name: "wang",
        age:  18,
    }

    reflectType(d) // type:person kind:struct
}
```

TypeOf

TypeOf函数接收任何的interface{}参数，并且把接口中的动态类型以reflect.Type形式返回。

示例：

```
package main

import (
    "fmt"
    "reflect"
)

func reflectType(x interface{}) {
    v := reflect.TypeOf(x)
```

```
    fmt.Printf("type:%v\n", v)
}
func main() {
    var a float32 = 3.14
    reflectType(a) // type:float32
    var b int64 = 100
    reflectType(b) // type:int64
}
```

ValueOf

ValueOf函数可以接收任意的interface{}并将接口的动态值以reflect.Value的形式返回。与reflect.TypeOf类似，reflect.ValueOf的返回值也是具体值，不过reflect.Value也可以包含一个接口值。

对于不同类型，我们用reflect.Value的kind方法来区分不同类型。但类型的分类（kind）只有少数几种：

- 基础类型： Bool, String以及数字类型
- 聚合类型：Array, struct
- 引用类型：Chan, Func, Ptr, Slice, Map
- 接口类型：interface
- Invalid类型：表示没有任何值。reflect.Value的零值就属于Invalid类型。

示例：

```
package main

import (
    "fmt"
    "reflect"
)

func reflectValue(x interface{}) {
    v := reflect.ValueOf(x)
    k := v.Kind()
    switch k {
    case reflect.Int64:
        // v.Int()从反射中获取整型的原始值，然后通过int64()强制类型转换
        fmt.Printf("type is int64, value is %d\n", int64(v.Int()))
    case reflect.Float32:
        // v.Float()从反射中获取浮点型的原始值，然后通过float32()强制类型转换
        fmt.Printf("type is float32, value is %f\n", float32(v.Float()))
    case reflect.Float64:
        // v.Float()从反射中获取浮点型的原始值，然后通过float64()强制类型转换
        fmt.Printf("type is float64, value is %f\n", float64(v.Float()))
    }
}
func main() {
    var a float32 = 3.14
    var b int64 = 100
    reflectValue(a) // type is float32, value is 3.140000
    reflectValue(b) // type is int64, value is 100
    // 将int类型的原始值转换为reflect.Value类型
    c := reflect.ValueOf(10)
    fmt.Printf("type c :%T\n", c) // type c :reflect.Value
```

```
}
```

value.Elem设置值

上面知识获取了值及类型，如果想要修改，可以通过elem，但必须传递的是指针。

```
package main

import (
    "fmt"
    "reflect"
)

func reflectSetValue2(x interface{}) {
    v := reflect.ValueOf(x)
    // 反射中使用 Elem()方法获取指针对应的值
    if v.Elem().Kind() == reflect.Int64 {
        v.Elem().SetInt(200)
    }
}
func main() {
    var a int64 = 100
    reflectSetValue2(&a)
    fmt.Println(a)
}
```

常用的类型判断

```
func Dis(path string, v reflect.Value)  {
    switch v.Kind() {
    case reflect.Invalid:  // 空
        fmt.Printf("%s= Invalid\n", path)
    case reflect.Slice, reflect.Array:
        for i := 0; i<v.Len(); i++{
            Dis(fmt.Printf("%s[%d]", path, i), v.Index(i))
        }
    case reflect.Struct:
        for i := 0; i< v.NumField(); i++{
            feildPath := fmt.Sprintf("%s.%s",path, v.Type().Field(i).Name)
            Dis(feildPath, v.Field(i))
        }
    case reflect.Map:
        for _, key :=range v.MapKeys(){
            fmt.Printf("%s", v.MapIndex(key))
        }
    case reflect.Ptr:
        if v.IsNil(){
            fmt.Printf("%s= nil\n", path)
        }else {
            Dis(fmt.Sprintf("*%s",path),v.Elem())
        }
    case reflect.Interface:
        if v.IsNil(){
            fmt.Printf("%s=nil\n", path)
```

```
        }else {
            fmt.Printf("%s.type=%s\n",path, v.Elem().Type())
        }
    default: // 基础类型，chan, 函数
        fmt.Printf("%s\n",path)
    }
}
```