|  |
| --- |
| <http://blog.51cto.com/speakingbaicai/1707637> |
| <https://www.jianshu.com/p/53adb1e92710> |
| https://studygolang.com/articles/6324 |
| https://studygolang.com/articles/12348#reply3 |

Golang反射reflect性能：

golang反射很慢，主要与其API设计有关，在java中我们一般使用反射都是这样来处理

|  |
| --- |
| Field field = clazz.getField(“hello”)  field.get(obj1)  field.get(obj2) |
| 这个取得的反射对象类型时 java.lang.reflect.Field 它是可以服用的，传入不同的obj，就可以取得这个obj上对应的field |

Golang的反射是：

type\_ := reflect.Typeof(obj)

field, \_ := type\_.FieldByName(“hello”)

这里取出的field对象是 reflect.StructField类型，但是它没有办法取得对应对象上的值，如果要取值，得用另外一套object，而不是type的反射

type\_ := reflect.Valueof(obj)

fieldValue := type\_.FieldByName(“hello”)

这里取出来的fieldValue类型时reflect.Value，它是一个具体的值，而不是一个可复用的反射对象了，每次反射都需要malloc这个reflect.Value结构体，并且涉及到GC

Glang feflcet慢主要两个原因：

1. 涉及到内存分配以及后续的GC
2. Reflect实现里面有大量的枚举，也就是for循环，比如类型之类的

|  |
| --- |
| 反射可以大大提高程序的灵活性，使得interface{}有更大的发挥余地  反射必须结合interface才行  变量的type要是concrete type的类型才有反射一说(也就是interface变量) |
| 反射可以将“接口类型变量”转换成”反射类型对象”  反射使用Typeof和Valueof函数从接口中获取目标对象信息 |
| 反射可以将“反射类型对象”转换为“接口类型变量”  Reflect.value.Interface().(已知类型)  遍历reflect.Type的Field获取其Field |
| 反射可以修改反射类型对象，但是其值必须是“addressable”  想要利用反射修改对象状态，前提是interface.data时settable,即pointer-interface |
| 通过反射可以”动态”调用方法 |
| Golang本身不支持模版，因此以往需要使用模版的场景下往往就需要使用反射来实现 |

本质上说，反射是一种检查接口变量的类型和值的机制。最基本的有reflect.Type 和 reflect.Value。可以通过reflect.Type()和reflect.Value()来得到接口变量的Type和Value

反射是在golang程序运行时检查变量所具有类型的一种机制。由于反射可以得出关于变量结构的数据（即 关于数据的数据 ），所以这被认为是golang元编程的基础。

反射三原则：

1. 从接口值到反射对象的反射

代表方法 reflect.valueof 和 reflect.typeof

1. 从反射对象到接口值的反射

从reflect.value可以使用interface方法还原接口值 y := v.Interface().(float64)

|  |
| --- |
| func main() {     var x float64 = 1.1     fmt.Println("reflect.Value:", reflect.ValueOf(x))     fmt.Println("reflect.Type:", reflect.TypeOf(x))     v := (reflect.ValueOf(x))     fmt.Println("reflect.Type:", v.Type())     fmt.Println("actual value(interface):", v.Interface())     fmt.Println("kind is float64?", v.Kind() == reflect.Float64)  } |
| wKioL1YxpHGgRbc8AACqUduMpKM284.jpg |
| wKiom1Yx5KyB-6Y_AAEc8sb5uKQ756.jpg |

1. 为了修改反射对象，其值必须可设置

|  |
| --- |
| import (      "fmt"      "reflect"  )    func main() {      var x float64 = 1.1      v := reflect.ValueOf(x)      fmt.Println("settability of v:",v.CanSet())      v.SetFloat(1.2)  } |
| wKioL1YxpQrBY-onAAIabYSpE_Q032.jpg |

v := reflect.valueof(x) 这个函数为值传递，因此不能被真正的修改

|  |
| --- |
| func main() {      var x float64 = 1.1      p := reflect.ValueOf(&x)      fmt.Println("type of p:",p.Type())      fmt.Println("settability of p:",p.CanSet())  } |
| wKiom1YxpTOB-8JJAAB6QfJi1gA741.jpg |

P的实际类型是 \*float64，而非float64，这样修改相当于要直接修改地址了

我们可以借助Elem方法，通过指针来修改指针指向的具体值。

|  |
| --- |
| func main() {      var x float64 = 1.1      p := reflect.ValueOf(&x)      fmt.Println("type of p:",p.Type())      v := p.Elem()      fmt.Println("type of v:",v.Type())      fmt.Println("settability of v:",v.CanSet())  } |
| wKioL1YxpeaQJjobAACWdkY8w_E223.jpg |

结构体操作：

当使用反射改动结构体的字段的时候，也就是当有结构体的地址的时候，能够改动它的字段。

因为希望对结构体进行改动，所以从它的地址创建了反射对象。设置了typeof T为其类型。然后遍历其字段，注意从结构类型中解析了字段名字，可是字段本身是原始的 reflect.value 对象

|  |
| --- |
| type T struct {  A int  B string  }  t := T{23, "skidoo"}  s := reflect.ValueOf(&t).Elem()  typeOfT := s.Type()  for i := 0; i < s.NumField(); i++ {  f := s.Field(i)  fmt.Printf("%d: %s %s = %v\n", i,  typeOfT.Field(i).Name, f.Type(), f.Interface())  } |
| 0: A int = 23  1: B string = skidoo |

设置性的要点：T的字段名要大写（可导出），由于仅仅是可导出的字段是可设置的。

由于s包括可设置的反射对象，所以能够改动结构体的字段

|  |
| --- |
| s.Field(0).SetInt(77)  s.Field(1).SetString("Sunset Strip")  fmt.Println("t is now", t) |
| t is now {77 Sunset Strip} |