# Reflect

# Introduction

Reflect digunakan untuk melakukan  inspeksi variabel, mengambil informasi dari variabel tersebut atau bahkan memanipulasinya.

Cakupan informasi yang bisa didapatkan lewat reflection sangat luas, seperti melihat struktur variabel, tipe, nilai pointer, dan banyak lagi.

Go menyediakan package reflect, berisikan banyak sekali fungsi untuk keperluan reflection.

Dari banyak fungsi yang tersedia di dalam package tersebut, ada 2 fungsi yang paling penting untuk diketahui, yaitu *reflect.ValueOf()* dan *reflect.TypeOf().*

- Fungsi *reflect.ValueOf()* akan mengembalikan objek dalam tipe *reflect.Value*, yang berisikan informasi yang berhubungan dengan nilai pada variabel yang dicari
- Sedangkan *reflect.TypeOf()* mengembalikan objek dalam tipe *reflect.Type*. Objek tersebut berisikan informasi yang berhubungan dengan tipe data variabel yang dicari

# Identifying Data Type & Value

Dengan reflection, tipe data dan nilai variabel dapat diketahui dengan mudah.

Fungsi *reflect.valueOf()* memiliki parameter yang bisa menampung segala jenis tipe data.

Fungsi tersebut mengembalikan objek dalam tipe *reflect.Value*, yang berisikan informasi mengenai variabel yang bersangkutan.

Objek *reflect.Value* memiliki beberapa method, salah satunya *Type()*.

Method ini mengembalikan tipe data variabel yang bersangkutan dalam bentuk string.

```go
package main

import "fmt"
import "reflect"

func main() {
    var number = 23
    var reflectValue = reflect.ValueOf(number)

    fmt.Println("tipe  variabel :", reflectValue.Type())

    if reflectValue.Kind() == reflect.Int {
        fmt.Println("nilai variabel :", reflectValue.Int())
    }
}
```

# Identifying Data Type & Value

Statement *reflectValue.Int()* menghasilkan nilai *int* dari variabel *number*.

Untuk menampilkan nilai variabel *reflect*, harus dipastikan dulu tipe datanya.

Ketika tipe data adalah *int*, maka bisa menggunakan method *Int()*.

Ada banyak lagi method milik struct *reflect.Value* yang bisa digunakan untuk pengambilan nilai dalam bentuk tertentu,

contohnya: *reflectValue.String()* digunakan untuk mengambil nilai string, *reflectValue.Float64()* untuk nilai *float64*, dan lainnya.

```go
package main

import "fmt"
import "reflect"

func main() {
    var number = 23
    var reflectValue = reflect.ValueOf(number)

    fmt.Println("tipe  variabel :", reflectValue.Type())

    if reflectValue.Kind() == reflect.Int {
        fmt.Println("nilai variabel :", reflectValue.Int())
    }
}
```

# Identifying Data Type & Value

Perlu diketahui, fungsi yang digunakan harus sesuai dengan tipe data nilai yang ditampung variabel.

Jika fungsi yang digunakan berbeda dengan tipe data variabelnya, maka akan menghasilkan error.

Contohnya pada variabel menampung nilai bertipe *float64*, maka tidak bisa menggunakan method *String()*.

Diperlukan adanya pengecekan tipe data nilai yang disimpan, agar pengambilan nilai bisa tepat.

Salah satunya bisa dengan cara seperti kode pada gambar disebelah kanan, yaitu dengan mememeriksa terlebih dahulu apa jenis tipe datanya menggunakan method *Kind()*, setelah itu diambil nilainya dengan method yang sesuai.

```go
package main

import "fmt"
import "reflect"

func main() {
    var number = 23
    var reflectValue = reflect.ValueOf(number)

    fmt.Println("tipe  variabel :", reflectValue.Type())

    if reflectValue.Kind() == reflect.Int {
        fmt.Println("nilai variabel :", reflectValue.Int())
    }
}
```

# Identifying Data Type & Value

List konstanta tipe data dan method yang bisa digunakan dalam refleksi di Go bisa dilihat pada link berikut:

https://godoc.org/reflect#Kind

# Accessing Value Using Interface Method

Jika nilai hanya diperlukan untuk ditampilkan ke output, bisa menggunakan .Interface().

Melalui  method tersebut segala jenis nilai bisa diakses dengan mudah.

```
var number = 23
var reflectValue = reflect.ValueOf(number)

fmt.Println("tipe  variabel :", reflectValue.Type())
fmt.Println("nilai variabel :", reflectValue.Interface())
```

# Accessing Value Using Interface Method

Method *Interface()* mengembalikan nilai interface kosong atau interface{}.

Nilai aslinya sendiri bisa diakses dengan meng-casting interface kosong tersebut.

```
var nilai = reflectValue.Interface().(int)
```

# Identifying Method Information

Informasi mengenai method juga bisa diakses lewat reflect, syaratnya masih sama seperti pada pengaksesan property, yaitu harus bermodifier public.

Pada contoh dibawah ini informasi method *SetName* akan diambil lewat reflection.

Siapkan method baru di struct student, dengan nama *SetName*.

```
func (s *student) SetName(name string) {
    s.Name = name
}
```

# Identifying Method Information

Penerapannya di fungsi main.

```
func main() {
    var s1 = &student{Name: "john wick", Grade: 2}
    fmt.Println("nama :", s1.Name)

    var reflectValue = reflect.ValueOf(s1)
    var method = reflectValue.MethodByName("SetName")
    method.Call([]reflect.Value{
        reflect.ValueOf("wick"),
    })

    fmt.Println("nama :", s1.Name)
}
```

# Identifying Method Information

```
func main() {
    var s1 = &student{Name: "john wick", Grade: 2}
    fmt.Println("nama :", s1.Name)

    var reflectValue = reflect.ValueOf(s1)
    var method = reflectValue.MethodByName("SetName")
    method.Call([]reflect.Value{
        reflect.ValueOf("wick"),
    })

    fmt.Println("nama :", s1.Name)
}
```

Pada kode di atas, disiapkan variabel *s1* yang merupakan instance struct student.

Awalnya property *Name* pada variabel tersebut berisikan string "john wick".

Setelah itu, refleksi nilai objek tersebut diambil, refleksi method *SetName* juga diambil.

Pengambilan refleksi method dilakukan menggunakan *MethodByName* dengan argument adalah nama method yang diinginkan, atau bisa juga lewat indeks method-nya *(menggunakan Method(i))*.

Setelah refleksi method yang dicari sudah didapatkan, *Call()* dipanggil untuk eksekusi method.

# Identifying Method Information

```
func main() {
    var s1 = &student{Name: "john wick", Grade: 2}
    fmt.Println("nama :", s1.Name)

    var reflectValue = reflect.ValueOf(s1)
    var method = reflectValue.MethodByName("SetName")
    method.Call([]reflect.Value{
        reflect.ValueOf("wick"),
    })

    fmt.Println("nama :", s1.Name)
}
```

Jika eksekusi method diikuti pengisian parameter, maka parameternya harus ditulis dalam bentuk array *[]reflect.Value* berurutan sesuai urutan deklarasi parameter-nya.

Dan nilai yang dimasukkan ke array tersebut harus dalam bentuk reflect.Value (gunakan *reflect.ValueOf()* untuk pengambilannya).