



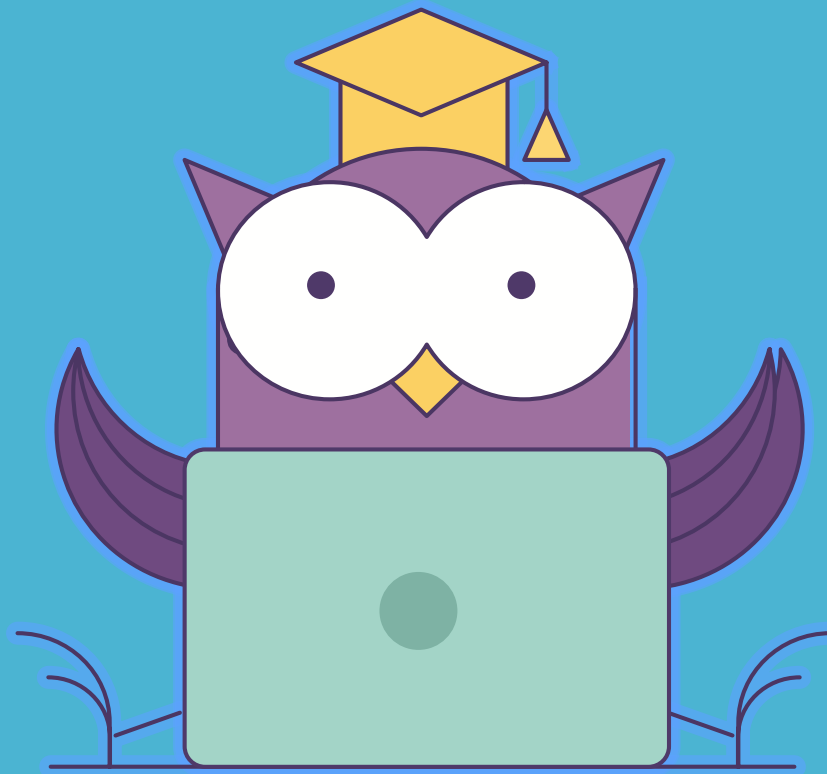
ОНЛАЙН-ОБРАЗОВАНИЕ

Рефлексия

Дмитрий Смаль, Елена Граховац



Как меня слышно и видно?



> Напишите в чат

+ если все хорошо

- если есть проблемы со звуком или с видео

!проверить запись!

- Введение
- Структура интерфейсов
- Рефлексия на уровне языка, `type switch`
- Использование пакета `reflect`
- `Type` и `Value`
- `unsafe.Pointer`

Пример рефлексии: возможность работы с переменными (включая тип и другие свойства) в рантайме.

Рефлексия в Go:

- на уровне языка
- в пакете `reflect`

Примеры использования рефлексии:

- `encoding/json` И `encoding/xml`
- `text/template` И `html/template`
- <https://github.com/kelseyhightower/envconfig>

- Для работы с типами, которые не удовлетворяют одному интерфейсу
- Для работы с данными, которые не определены во время компиляции

- Более чувствительный код: узнаем об ошибке не во время компиляции, а только в рантайме
- В коде нет информации о типах: статические анализаторы не применимы
- Нужна дополнительная документация (слишком много `interface{}` , не всегда понятно, что именно там ждут)
- Скорость

Каждая переменная в Go обладает статическим типом, переменные разных типов нельзя присваивать (в большинстве случаев)

```
type MyInt int

var i int
var j MyInt
j = i // ошибка компиляции
```


Однако мы можем присвоить переменную типа `T` переменной типа интерфейс `I`, если тип `T` реализует методы `I`.

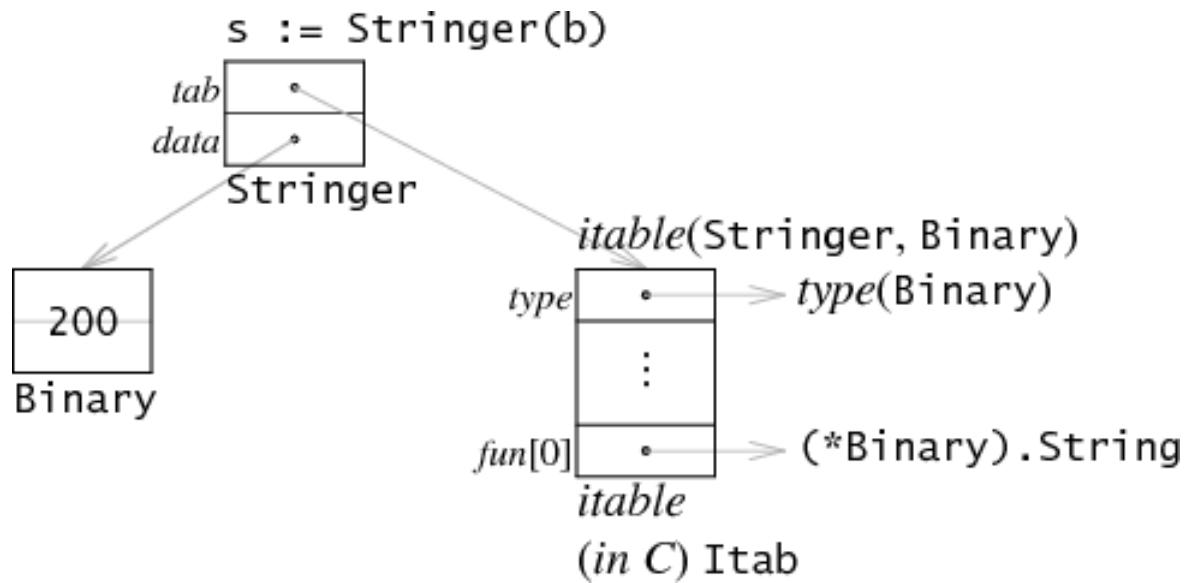
```
var fh *os.File = os.Open("bla.txt")
var rw io.ReadWriter = fh
var r io.Reader = rw
var any interface{} = r
any = []int{0}
```

Для переменной `r`:

- статический тип - `io.Reader`
- динамический тип - `*os.File`

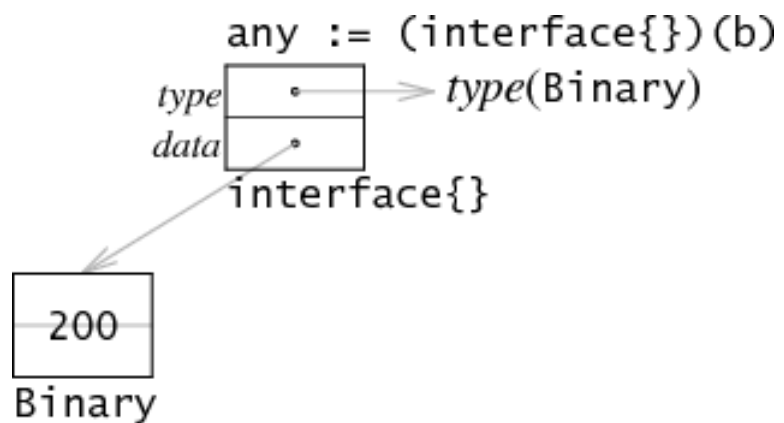
Переменная типа интерфейс представляет собой пару

```
// src/runtime/runtime2.go
type iface struct {
    tab *itab          // адрес таблицы методов
    data unsafe.Pointer // адрес значения
}
type itab struct {
    inter *interfacetype // тип интерфейса
    _type *_type         // тип значения
    ...
}
```



В случае с пустым интерфейсом (`interface{}`) структура еще проще

```
// src/runtime/runtime2.go
type eface struct {
    _type *_type          // тип значения
    data  unsafe.Pointer    // адрес значения
}
```



Рефлексия - это возможность в программе исследовать и изменять свою структуру (в частности переменные) на этапе выполнения.

Некоторые возможности вынесены на уровень языка

Например type assertion

```
var r io.Reader
var f *os.File

f, ok = r.(*os.File)
```

Или type switch

```
switch v := i.(type) {
case int:      // here v has type int
    i = v + 1
case string:  // here v has type string
    i = v + "1"
default:     // no match; here v has the same type as i
}
```

Пакет `reflect` в Go представляет API для работы с переменными заранее неизвестных типов.
Почитать: <https://blog.golang.org/laws-of-reflection>

Значения типа `reflect.Value` представляют собой программную обертку над значением произвольной переменной.

```
var i int = 42
var s = struct{string; int}{"hello", 42}

iv := reflect.ValueOf(i) // тип reflect.Value
sv := reflect.ValueOf(&s) // тип reflect.Value
```

Какие методы есть у `reflect.Value` ?

```
value.Type() reflect.Type // вернуть тип обертку над типом
value.Kind() reflect.Kind // вернуть "базовый" тип

value.Interface() interface{} // вернуть обернутое значение как interface{}
value.Int() int64 // вернуть значение как int64
value.String() string // вернуть значение как string

value.CanSet() bool // возможно ли изменить значение ?
value.SetInt(int64) // установить значение типа int64
value.Elem() reflect.Value // разыменованье указателя или интерфейса
```

```
func assertString(iv interface{}) (string, bool) {
    rv := reflect.ValueOf(iv)
    s := ""
    ok := false
    if rv.Kind() == reflect.String {
        s = rv.String()
        ok = true
    }
    return s, ok
}

func main() {
    var iv interface{} = "hello"
    s, ok := assertString(iv)
    fmt.Println(s, ok)
    s2, ok := assertString(42)
    fmt.Println(s2, ok)
}
```

<https://play.golang.org/p/b0s13oTTQt1>

`reflect.Kind` представляет собой базовый тип для значения. `reflect.Kind` определяет какие методы имеют смысл для конкретного `reflect.Value`, а какие вызовут панику.

```
const (  
    Invalid Kind = iota  
    Bool  
    Int  
    Int8  
    Int16  
    ...
```

`reflect.Type` представляет собой информацию о конкретном типе: имя, пакет, список методов и т.д.

```
t.Name() string           // имя типа  
t.PkgPath() string       // пакет, в котором определен тип  
t.Size() uintptr         // размер в памяти, занимаемый значением  
t.Implements(u Type) bool // реализует ли интерфейс u ?  
t.MethodByName(string name) reflect.Value // метод по имени
```

Большинство методов продублировано в `reflect.Value`

Неправильный способ:

```
var x float64 = 3.4
v := reflect.ValueOf(x) // ???
v.SetFloat(7.1) // panic: reflect.Value.SetFloat using unaddressable value
fmt.Println(v.CanSet()) // false
```

Правильный способ:

```
var x float64 = 3.4
p := reflect.ValueOf(&x) // адрес переменной x
fmt.Println(p.Type())    // *float64
fmt.Println(p.CanSet())  // false

v := p.Elem()            // переход по указателю
fmt.Println(v.Type())    // float64
fmt.Println(v.CanSet())  // true
v.SetFloat(7.1)
fmt.Println(x)           // 7.1
```

`reflect.Value.Elem()` - переходит по указателю или к базовому объекту интерфейса.

Если `v` это рефлексия значения структуры (`reflect.Value`), то

```
v.NumField() int           // возвращает кол-во полей в структуре
v.Field(i int) reflect.Value // возвращает рефлексии для отдельного поля
v.FieldByName(s string) reflect.Value // тоже, но по имени поля
```

Если `t` это рефлексия типа структуры `t := v.Type()`, то

```
t.NumField() int           // возвращает кол-во полей в структуре
t.Field(i int) reflect.StructField // возвращает рефлексии для конкретного поля
t.FieldByName(name string) (reflect.StructField, bool) // тоже, но по имени поля
```

СВОЙСТВА `reflect.StructField`

```
Name      string           // имя поля
Type      reflect.Type     // рефлексия типа поля
Tag       reflect.StructTag // описание тэгов конкретного поля
Offset    uintptr         // смещение в структуре
```

Преобразование структуры в map

```
func structToMap(iv interface{}) (map[string]interface{}, error) {
    v := reflect.ValueOf(iv)
    if v.Kind() != reflect.Struct {
        return nil, errors.New("not a struct")
    }
    t := v.Type()
    mp := make(map[string]interface{})
    for i := 0; i < t.NumField(); i++ {
        field := t.Field(i) // reflect.StructField
        fv := v.Field(i)    // reflect.Value
        mp[field.Name] = fv.Interface()
    }
    return mp, nil
}
```

<https://play.golang.org/p/B7QEHLgNSTG>

```
func mapToStruct(mp map[string]interface{}, iv interface{}) (error) {
    v := reflect.ValueOf(iv)
    if v.Kind() != reflect.Ptr {
        return errors.New("not a pointer to struct")
    }
    v = v.Elem()
    if v.Kind() != reflect.Struct {
        return errors.New("not a pointer to struct")
    }
    t := v.Type()
    for i := 0; i < t.NumField(); i++ {
        field := t.Field(i) // reflect.StructField
        fv := v.Field(i)    // reflect.Value
        if val, ok := mp[field.Name]; ok {
            mfv := reflect.ValueOf(val)
            if mfv.Kind() != fv.Kind() {
                return errors.New("incomatible type for " + field.Name)
            }
            fv.Set(mfv)
        }
    }
    return nil
}
```

<https://play.golang.org/p/6bdq9ZkPCY0>

Получив рефлексию функции/метода, мы можем исследовать количество и типы ее аргументов:

```
f := fmt.Printf
v := reflect.ValueOf(f)
t := v.Type()

t.NumIn()    // количество аргументов
t.NumOut()   // количество возвращаемых значений
alt := t.In(0) // reflect.Type первого аргумента
olt := t.Out(0) // reflect.Type первого возвращаемого значения
t.IsVariadic() // принимает ли функция переменное число аргументов ?
```

Мы можем получить список методов определенных над типом:

```
type Int int
func (i Int) Say() string {
    return "42"
}

func main() {
    var obj Int
    v := reflect.ValueOf(obj)
    for i := 0; i < v.NumMethod(); i++ {
        meth := v.Method(i) // reflect.Value
    }
    sayMethod := v.MethodByName("Say") // reflect.Value
}
```

```
import (  
    "fmt"  
    "reflect"  
)  
  
func main() {  
    f := fmt.Printf  
    v := reflect.ValueOf(f)  
    args := []reflect.Value{  
        reflect.ValueOf("test %d\n"),  
        reflect.ValueOf(42),  
    }  
    ret := v.Call(args) // []reflect.Value  
    fmt.Println(ret)  
}
```

<https://play.golang.org/p/kqIu1l6B19u>

В случае неправильного количества / типа аргументов случится паника.

В Go указатели на разные типы не совместимы между собой (т.к. сами являются разными типами)

```
type St struct{ a, b int32}

var b [8]byte
bp := &b
var sp *St
sp = bp // nop
sp = (*St)(bp) // nop
```

Однако тип `unsafe.Pointer` является исключением. Компилятор Go позволяет делать явное преобразование типа любого указателя в `unsafe.Pointer` и обратно (а так же в `uintptr`)

```
import "unsafe"

var b [8]byte
bp := &b
var sp *St
var up unsafe.Pointer
up = unsafe.Pointer(bp)
sp = (*St)(up)
sp.a = 12345678
fmt.Println(b) // [78 97 188 0 0 0 0 0]
```

<https://go101.org/article/unsafe.html>

Зачем нужен пакет unsafe ?

В первую очередь он используется в самом Go, например в пакетах `runtime` и `reflect`

```
// src/reflect/value.go
type Value struct {
    typ *rtype
    ptr unsafe.Pointer
    ...
}

func (v Value) SetFloat(x float64) {
    v.mustBeAssignable()
    switch k := v.kind(); k {
    default:
        panic(&ValueError{"reflect.Value.SetFloat", v.kind()})
    case Float32:
        *(*float32)(v.ptr) = float32(x)
    case Float64:
        *(*float64)(v.ptr) = x
    }
}
```

Написать функцию, которая определяет, реализует ли тип X интерфейс Y :

- без помощи рефлексии для конкретного интерфейса Y (например, можно взять `Writer` или `Stringer`);
- с помощью рефлексии для любого переданного Y .

Для тех, кто хочет побольше хардкора:

- Про рефлексю от Роба Пайка: <https://blog.golang.org/laws-of-reflection>
- Про unsafe: <https://go101.org/article/unsafe.html>
- Про интерфейсы от Рассы Кокса: <https://research.swtch.com/interfaces>
- Про интерфейсы от Ивана Данилюка: <https://habr.com/ru/post/276981/>
- Полезные примеры на рефлексю (с пояснениями): <https://github.com/a8m/reflect-examples>

Заполните пожалуйста опрос

<https://otus.ru/polls/4904/>



Спасибо за внимание!

