Learning to `reflect`

@suhailpatel - London Gophers

Monzo
Suhail Patel

Jump to...

All Unreads
Threads

Starred

#semaphore

0 | 3

Search

January 11th, 2018

5:02 AM
acquire 1

6:14 AM
acquire 2

🌴 9:10 AM
acquire 3

9:23 AM
acquire 4

9:36 AM
acquire 5

10:09 AM
acquire 7

release 7

10:36 AM
release 5

12:13 PM
acquire 5

release 5

1:50 PM
acquire 5

1:58 PM
back to the void with you, 5!

3:17 PM
Saving you from the void, 5

3:23 PM
Right back in there 5

6:29 PM

Monzo
Suhail Patel

Jump to...

All Unreads
Threads

Starred

#semaphore

0 | 3

Search

February 19th, 2018

Slackbot 4:00 AM
Reminder: ALL SMARTCLIENTS RELEASED 💥

[redacted]
`acquire 1`

[redacted]
`release 1`

[redacted]
`acquire 1`

[redacted]
`acquire 2`

[redacted]
`acquire 3`

[redacted]
`release 3`

[redacted]
`acquire 4` (edited)

[redacted]
`acquire 4` `again`

[redacted]
`acquire 1`

[redacted]
`acquire 2`

[redacted]
`release 1`

[redacted]
`acquire 1`

Learning to `reflect`

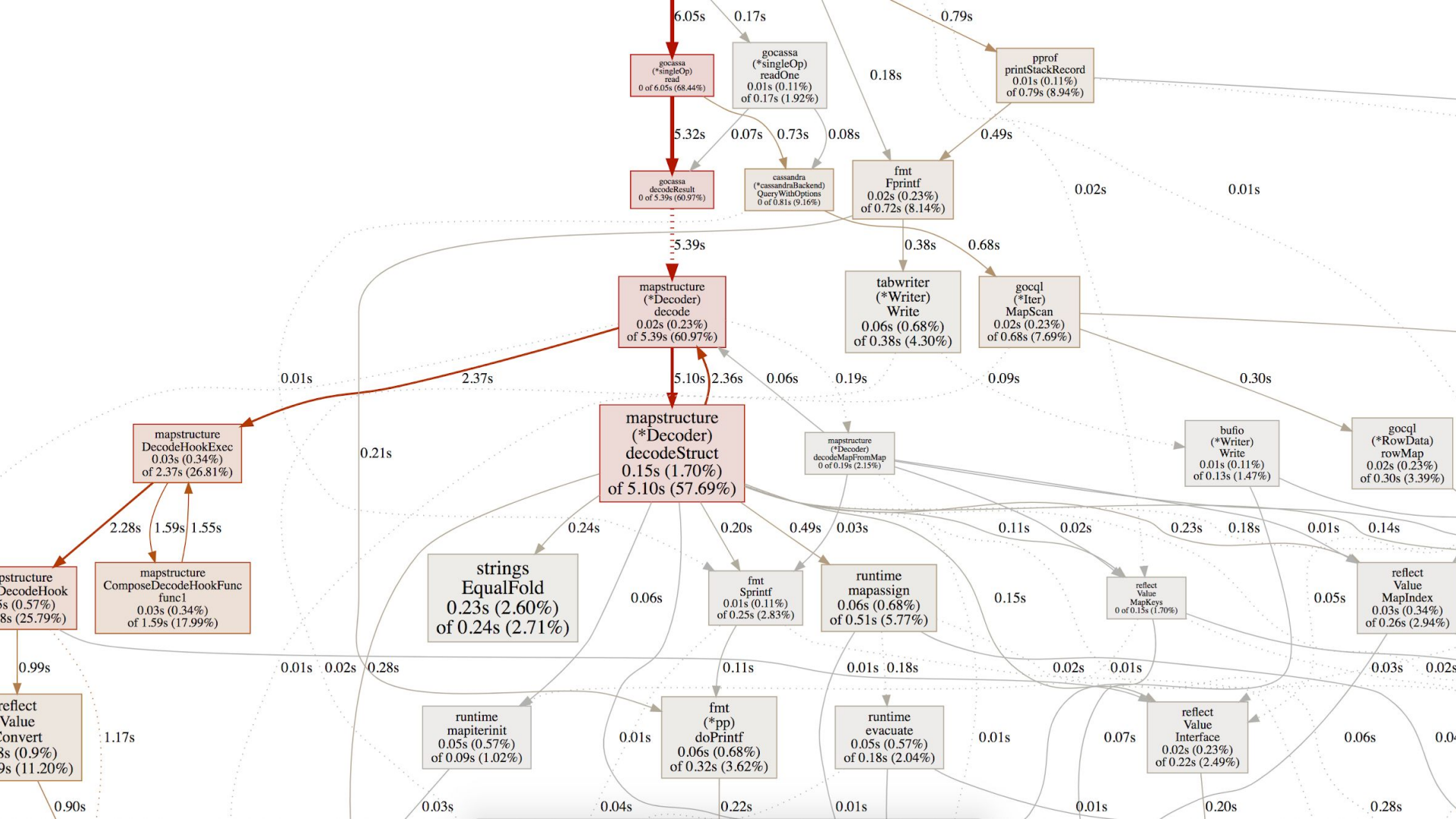
@suhailpatel - London Gophers

Hi, i’m Suhail

I’m an Engineer at Monzo on the Platform squad.
We help build the base so other engineers can
ship their services and applications.

Email: **hi@suhailpatel.com**
Twitter: **@suhailpatel**


monzo
5354 2000 0000 0000
EXP
END 08/1
469-624-882


6.05s
0.17s
0.79s
gocassa
(*singleOp)
read
0 of 6.05s (68.44%)
gocassa
(*singleOp)
readOne
0.01s (0.11%)
of 0.17s (1.92%)
0.18s
pprof
printStackRecord
0.01s (0.11%)
of 0.79s (8.94%)
5.32s
0.07s
0.73s
0.08s
0.49s
gocassa
decodeResult
0 of 5.39s (60.97%)
cassandra
(*cassandraBackend)
QueryWithOptions
0 of 0.81s (9.16%)
fmt
Fprintf
0.02s (0.23%)
of 0.72s (8.14%)
0.02s
0.01s
5.39s
0.38s
0.68s
mapstructure
(*Decoder)
decode
0.02s (0.23%)
of 5.39s (60.97%)
tabwriter
(*Writer)
Write
0.06s (0.68%)
of 0.38s (4.30%)
gocql
(*Iter)
MapScan
0.02s (0.23%)
of 0.68s (7.69%)
0.01s
2.37s
5.10s
2.36s
0.06s
0.19s
0.09s
0.30s
mapstructure
DecodeHookExec
0.03s (0.34%)
of 2.37s (26.81%)
0.21s
mapstructure
(*Decoder)
decodeStruct
0.15s (1.70%)
of 5.10s (57.69%)
mapstructure
(*Decoder)
decodeMapFromMap
0 of 0.19s (2.15%)
bufio
(*Writer)
Write
0.01s (0.11%)
of 0.13s (1.47%)
gocql
(*RowData)
rowMap
0.02s (0.23%)
of 0.30s (3.39%)
2.28s
1.59s
1.55s
0.24s
0.20s
0.49s
0.03s
0.11s
0.02s
0.23s
0.18s
0.01s
0.14s
pstructure
DecodeHook
5s (0.57%)
8s (25.79%)
mapstructure
ComposeDecodeHookFunc
func1
0.03s (0.34%)
of 1.59s (17.99%)
strings
EqualFold
0.23s (2.60%)
of 0.24s (2.71%)
0.06s
fmt
Sprintf
0.01s (0.11%)
of 0.25s (2.83%)
runtime
mapassign
0.06s (0.68%)
of 0.51s (5.77%)
0.15s
reflect
Value
MapKeys
0 of 0.15s (1.70%)
0.05s
reflect
Value
MapIndex
0.03s (0.34%)
of 0.26s (2.94%)
0.99s
0.01s
0.02s
0.28s
0.11s
0.01s
0.18s
0.02s
0.01s
0.03s
0.02s
reflect
Value
Convert
8s (0.9%)
9s (11.20%)
1.17s
runtime
mapiterinit
0.05s (0.57%)
of 0.09s (1.02%)
0.01s
fmt
(*pp)
doPrintf
0.06s (0.68%)
of 0.32s (3.62%)
runtime
evacuate
0.05s (0.57%)
of 0.18s (2.04%)
0.01s
0.07s
reflect
Value
Interface
0.02s (0.23%)
of 0.22s (2.49%)
0.06s
0.90s
0.03s
0.04s
0.22s
0.01s
0.01s
0.20s
0.28s

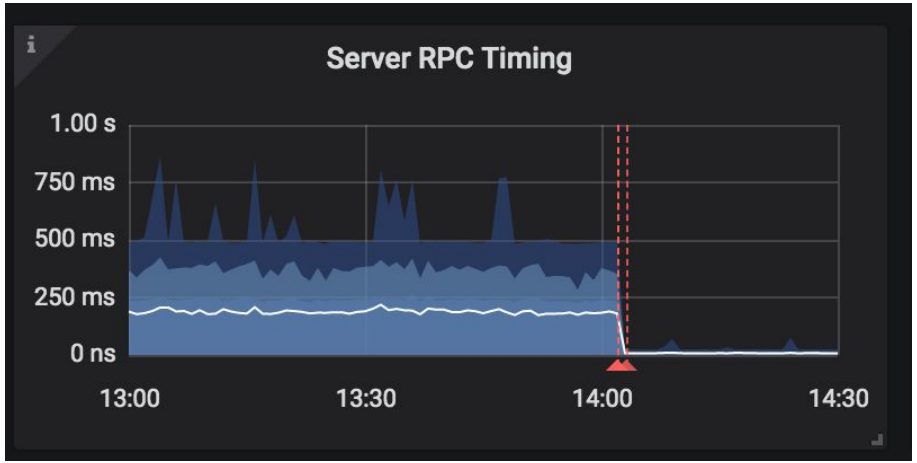
Why Reflection?

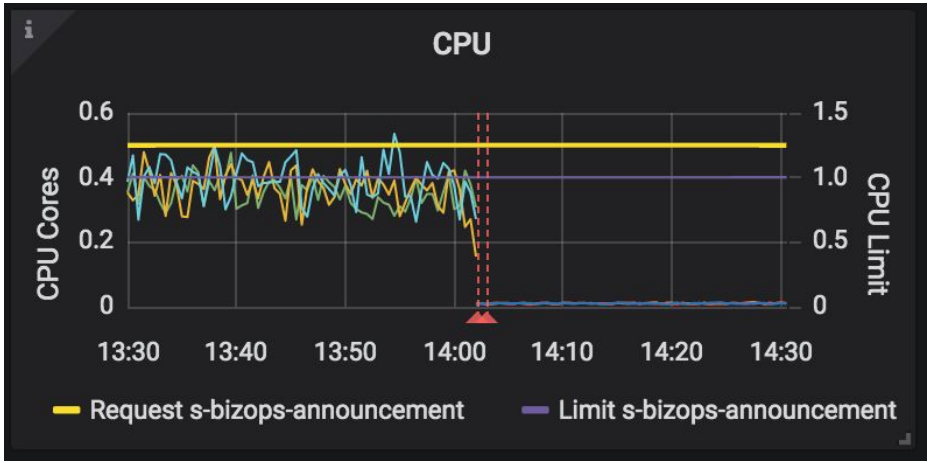
```go
map[string]interface{}{
  "name": "Suhail",
  "job":  "Backend Engineer",
  "role": "Platform"
}
```



```go
type Employee struct {
  Name string
  Job  string
  Role string
}

Employee{
  Name: "Suhail",
  Job:  "Backend Engineer",
  Role: "Platform"
}
```





https://github.com/monzo/gocassa/pull/49

func DeepEqual

```
func DeepEqual(x, y interface{}) bool
```

DeepEqual reports whether x and y are “deeply equal,” defined as follows. Two values of identical type are deeply equal if one of the following cases applies. Values of distinct types are never deeply equal.

Array values are deeply equal when their corresponding elements are deeply equal.

Struct values are deeply equal if their corresponding fields, both exported and unexported, are deeply equal.

Func values are deeply equal if both are nil; otherwise they are not deeply equal.

Interface values are deeply equal if they hold deeply equal concrete values.

Map values are deeply equal when all of the following are true: they are both nil or both non-nil, they have the same length, and either they are the same map object or their corresponding keys (matched using Go equality) map to deeply equal values.

Pointer values are deeply equal if they are equal using Go's == operator or if they point to deeply equal values.

Slice values are deeply equal when all of the following are true: they are both nil or both non-nil, they have the same length, and either they point to the same initial entry of the same underlying array (that is, &x[0] == &y[0]) or their corresponding elements (up to length) are deeply equal. Note that a non-nil empty slice and a nil slice (for example, []byte{} and []byte(nil)) are not deeply equal.

Other values - numbers, bools, strings, and channels - are deeply equal if they are equal using Go's == operator.

```go
me := map[string]interface{}{
    "id":       1,
    "name":     "Suhail Patel",
    "pronouns": []string{"he", "him", "his"},
    "location": "London, UK",
    "bio":      "I look at 📉 charts",

    "job": map[string]interface{}{
      "role":    "Backend Engineer",
      "squad":   "Platform",
      "joined":  time.Date(2018, 7, 2, 7, 0, 0, 0, time.UTC),
    },
}
```

```go
type UserJob struct {
  Role   string
  Squad  string
  Joined time.Time
}

type User struct {
  ID       uint64
  Name     string
  Pronouns []string
  Location string
  Bio      string

  Job *UserJob
}

func ToUser(in map[string]interface{}) User {
  u := User{}
  // magic transformation
  return u
}
```

```go
func ToUser(in map[string]interface{}) User {
  u := User{}
  if id, ok := in["id"]; ok {
    u.ID = v.(uint64)
  }
  if name, ok := in["name"]; ok {
    u.Name = name.(string)
  }
  if pronouns, ok := in["pronouns"]; ok {
    u.Pronouns = pronouns.([]string)
  }
  if location, ok := in["location"]; ok {
    u.Location = location.(string)
  }
  // ... so on
  return u
}
```

Law 1

Reflection is a way to get from `interface{}` value to a `reflect` object

https://blog.golang.org/laws-of-reflection

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

func main() {
  name := "Suhail"
  fmt.Printf("TypeOf: %s, Type: %T\n", reflect.TypeOf(name), reflect.TypeOf(name))
  fmt.Printf("ValueOf: %s, Type: %T\n", reflect.ValueOf(name), reflect.ValueOf(name))
}

// TypeOf: string, Type: *reflect.rtype
// ValueOf: Suhail, Type: reflect.Value
```

Law 2

Reflection goes from `reflect` object back to `interface{}` value

https://blog.golang.org/laws-of-reflection

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

func main() {
  name := "Suhail"
  v := reflect.ValueOf(name) // reflect.Value
  n := v.Interface().(string)
  fmt.Printf("Value: %s, Type: %T\n", n, n)
}

// Value: Suhail, Type: string
```

Law 3

To modify a reflection object, the value must be settable

https://blog.golang.org/laws-of-reflection

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

func main() {
  name := "Suhail"
  v := reflect.ValueOf(name) // reflect.Value
  v.SetString("Bingo")
  fmt.Printf("Value: %s\n", v)
}
```

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

func main() {
  name := "Suhail"
  v := reflect.ValueOf(name) // reflect.Value
  fmt.Printf("CanSet: %v\n", v.CanSet()) // CanSet: false
  v.SetString("Bingo")
  fmt.Printf("Value: %s\n", v)
}

// panic: reflect: reflect.flag.mustBeAssignable using unaddressable value
//
// goroutine 1 [running]:
// reflect.flag.mustBeAssignableSlow(0x98, 0x7f32)
//     /usr/local/go/src/reflect/value.go:247 +0x180
```

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

func main() {
  name := "Suhail"
  reflect.ValueOf(name) // pass by value
  reflect.ValueOf("Suhail")  // pass by value
}
```

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

func main() {
  name := "Suhail"
  v := reflect.ValueOf(&name) // reflect.Value of type *string
  v.Elem().SetString("Bingo")
  fmt.Printf("Value: %s\n", v.Elem())
}

// Value: Bingo
```

```go
type UserJob struct {
  Role   string
  Squad  string
  Joined time.Time
}

type User struct {
  ID       uint64
  Name     string
  Pronouns []string
  Location string
  Bio      string

  Job *UserJob
}

func ToUser(in map[string]interface{}) User {
  u := User{}
  // magic transformation
  return u
}
```

```go
func ToUser(in map[string]interface{}) (User, error) {
  u := User{}
  err := Unmarshal(in, &u)
  return err
}

func Unmarshal(in map[string]interface{}, target interface{}) error {
  if reflect.TypeOf(target).Kind() != reflect.Ptr { // target is type '*User'
    return fmt.Errorf("expected a pointer to a struct, got %T", target)
  }

  targetStruct := reflect.ValueOf(target).Elem() // targetStruct is type 'User'
  if reflect.TypeOf(targetStruct).Kind() != reflect.Struct {
    return fmt.Errorf("expected a pointer to a struct, got %T", target)
  }

  // continue unmarshalling here...
  return nil
}
```

```go
func Unmarshal(in map[string]interface{}, target interface{}) error {
  // validation code here...

  structValue := reflect.ValueOf(target).Elem()
  structType := reflect.TypeOf(target).Elem()

  // go through each of our fields in the struct and give
  // it the corresponding lowercase field value from the map
  for i := 0; i < structType.NumField(); i++ {
    field := structType.Field(i) // reflect.StructField
    if val, ok := in[strings.ToLower(field.Name)]; ok {
      structValue.Field(i).Set(reflect.ValueOf(val))
    }
  }

  return nil
}
```

```go
type UserJob struct {
  Role   string
  Squad  string
  Joined time.Time
}

type User struct {
  ID       uint64
  Name     string
  Pronouns []string
  Location string
  Bio      string

  Job *UserJob
}

func ToUser(in map[string]interface{}) User {
  u := User{}
  // magic transformation
  return u
}
```

```go
me := map[string]interface{}{
    "id":       1,
    "name":     "Suhail Patel",
    "pronouns": []string{"he", "him", "his"},
    "location": "London, UK",
    "bio":      "I look at 📉 charts",

    "job": map[string]interface{}{
      "role":    "Backend Engineer",
      "squad":   "Platform",
      "joined":  time.Date(2018, 7, 2, 7, 0, 0, 0, time.UTC),
    },
}
```

```go
func Unmarshal(in map[string]interface{}, target interface{}) error {
  // validation code here...

  structValue := reflect.ValueOf(target).Elem()
  structType := reflect.TypeOf(target).Elem()

  // go through each of our fields in the struct and give
  // it the corresponding lowercase field value from the map
  for i := 0; i < structType.NumField(); i++ {
    field := structType.Field(i) // reflect.StructField
    if val, ok := in[strings.ToLower(field.Name)]; ok {
      structValue.Field(i).Set(reflect.ValueOf(val))

      // if val is a map[string]interface{} AND
      // val is a field.Type is a struct or ptr to a struct
      // then recursively call Unmarshal with val
    }
  }

  return nil
}
```

https://github.com/suhailpatel/talks/tree/master/london-gophers-2019/

```
panic: reflect: NumField of non-struct type

goroutine 1 [running]:
reflect.(*rtype).NumField(0xf8f80, 0x40c138, 0x1614c0, 0xf8f80)
	/usr/local/go/src/reflect/type.go:981 +0x80
main.Unmarshal(0x41a6e8, 0xf8580, 0x40c138, 0x3, 0x41a76c, 0x0)
	/tmp/sandbox798052193/prog.go:69 +0x320
main.ToUser(...)
	/tmp/sandbox798052193/prog.go:46
main.main()
	/tmp/sandbox798052193/prog.go:41 +0x320

Program exited.
```

```
$ go test -benchmem -bench=.
goos: darwin
goarch: amd64
BenchmarkHandRolled-4      12442106           93.2 ns/op           0 B/op           0 allocs/op
BenchmarkReflect-4           902896           1374 ns/op         352 B/op          21 allocs/op
PASS
ok    /Users/Suhail/Development/Talks/london-gophers-2019/src     3.342s
```

func TypeOf

```
func TypeOf(i interface{}) Type
```

TypeOf returns the reflection Type that represents the dynamic type of i. If i is a nil interface value, TypeOf returns nil.

func ValueOf

```
func ValueOf(i interface{}) Value
```

ValueOf returns a new Value initialized to the concrete value stored in the interface i. ValueOf(nil) returns the zero Value.

https://golang.org/pkg/reflect

Thanks!

Email: **hi@suhailpatel.com**
Twitter: **@suhailpatel**