Reflection

Admin

-
- For next week, please install gocv (openCV with go)
  - https://gocv.io/getting-started/
- After these slides – read chapter 14 in the book
  - Or chapter 16 if you ended up getting the second edition a few weeks ago
- And reflect on it deeply
  - Yup go is such a punny language

Reflection

- How many of you have done some work in reflection in programming?

Reflection

- How many of you have done some work in reflection in programming?
  - How ‘bout if I call it ‘meta-programming’?
  - Any more of you?

Assumption

- Based on my recent weeks with you all I’m going to assume that a couple of you have heard of it – mostly a long time ago.
  - And the rest this will be new for.

Reflection/Metaprogramming

- Reflection (also sometimes called ‘Meta-programming’)
  - Run-time inspection of program data to find their type
  - How does this happen in a strongly typed, statically typed programming language?
    - Whose feeling like a lucky volunteer?

Reflection/Metaprogramming

- Reflection (also sometimes called ‘Meta-programming’)
  - Run-time inspection of program data to find their type
  - How does this happen in a strongly typed, statically typed programming language?
  - How about bringing data down from an API
  - Or building a generic data base insert generator
    - Take arbitrary struct and build query to insert as database record

The empty interface

- Recall the empty interface
  - var mystery interface{}
  - The empty interface matches every struct that has at least zero methods defined for it
  - So it matches what?
    - Whose feelin' lucky?

The empty interface

- Recall the empty interface
  - var mystery interface{}
  - The empty interface matches every struct that has at least zero methods defined for it
  - So it matches what?
    - EVERYTHING!!

Basic reflection in Go

- Need reflection package
  - From standard library
- reflect.TypeOf(<put your unknown here>)
  - Returns a reflect.Type object
- <type object>.Kind()
  - Returns the kind of thing that type is

So lets try it

- Lets break out goland and try out the basics:
  - Lets see what the difference is between type and kind
- Import the reflect package

```
func basicReflectionDemo(mystery interface{}){
    mysteryType := reflect.TypeOf(mystery)
    kind := mysteryType.Kind()
    fmt.Println("The type of the parameter is: ", mysteryType)
    fmt.Println("The kind of that parameter is: ", kind)
}
```

- Lets take a look at this function

Lets start small

- Lets start easy:

```
func main() {
	var1:= 4
	basicReflectionDemo(var1)
	fmt.Println("==============================")

}
```

- Lets try this out

Lets start small

- Lets start easy:

```
func main() {
	var1:= 4
	basicReflectionDemo(var1)
	fmt.Println("==============================")

}
```

- Lets try this out
  - Hmmm the type and kind are the same

Now lets add a slice

- Lets try a slice as our next var
  - How does this change things?
  -

Now lets add a slice

- Lets try a slice as our next var
  - How does this change things?
- Now lets define a small struct
  - Create one
  - And send it as the parameter to our basicReflectionDemo

Now lets add a slice

- Lets try a slice as our next var
  - How does this change things?
- Now lets define a small struct
  - Create one
  - And send it as the parameter to our basicReflectionDemo
- Now we can see better the difference between type and kind

ValueOf

- One more important function from reflect
  - reflect.ValueOf(<something here>)
    - Returns a reflect.Value object
    - That you can then use for reflection
- Code from the example project:
  - Lets look

```
    jobValue := reflect.ValueOf(Job) //cheating here since I haven't covered reflection yet
    //jobType := jobValue.Type()
    for fieldNum := 0; fieldNum < jobValue.NumField(); fieldNum++ {
            outputFile.SetCellValue("Comp490 Jobs", fmt.Sprintf("%s%d", string(rune(65+fieldNum)), line),
                    jobValue.Field(fieldNum))
    }


}
```

Some uses

- Some uses of reflection:
- How do we usually create a map in go?
  - Lucky Volunteer?

Some uses

- Some uses of reflection:
- How do we usually create a map in go?
  - Using make
  - HashTable := make (map[string]int)
  - But what if you don’t know what kind of numbers you have?
  - SecondTable:= make(map[string]float)
  -

Make map

- Now lets write a function that takes an interface param
  - If it is of type int make the map from string to int
  - Otherwise make the map from string to float

Some hints for later

- mapType1 := reflect.TypeOf(HashTable)
- mapType2 := reflect.TypeOf(SecondTable)
- switch/if magic goes here
- actualMap = reflect.MakeMap(mapType1)

Reflect with structs

- When using reflection with structs
  - We can find out how many fields it has
  - And what kind
    - NumField()
      - Use on a reflect.Value
      - Returns the number of fields in struct
    - Field(i)
      - Use of reflect.Value
      - Where i is the field position
  - Lets try it.

References:

https://golangbot.com/reflection/

https://www.geeksforgeeks.org/reflection-in-golang/

Your Book: chapter 14

https://medium.com/capital-one-tech/learning-to-use-go-reflection-822a0aed74b7

https://golang.org/pkg/reflect/

Rob Pike’s Take:

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