Les Pièges du Go 1

Michel Casabianca
casa@sweetohm.net

Le langage de programmation Go a la réputation d'être simple à apprendre. Cependant, il recèle quelques pièges qui peuvent être difficiles à détecter pour un novice. Cette série d'articles propose d'en désamorcer quelques uns.

Considérons ce code (code sur le Playground):

```
package main

func modify(array [3]string) {
    array[0] = "modified!"
}

func main() {
    var array = [3]string{"unchanged", "foo", "bar"}
    modify(array)
    println(array[0])
}
```

Dans cet exemple, nous passons un tableau par valeur à une fonction qui le modifie. Comme nous le passons par valeur, le tableau original n'est pas modifié. Donc si nous lançons ce code :

```
$ go run array.go
unchanged
```

Maintenant remplaçons le tableau par un slice, tableau de taille variable (code sur le Playground):

```
package main

func modify(slice []string) {
        slice[0] = "modified!"
}

func main() {
        var slice = []string{"unchanged", "foo", "bar"}
        modify(slice)
        println(slice[0])
}
```

Lorsque nous lançons ce code :

```
$ go run broken.go
modified!
```

Ceci peut sembler étrange car nous avons passé le slice par valeur, donc le slice original ne devrait pas avoir été modifié !

Prenez cinq minutes pour essayer d'expliquer ce phénomène avant de regarder l'explication ci-dessous :

Explication

Lorsque nous passons le slice par valeur, tout se passe comme si nous l'avions passé par référence. Le slice se comporte comme un tableau passé par référence (code sur le Playground) :

```
package main

func modify(array *[3]string) {
        array[0] = "modified!"
}

func main() {
        var array = [3]string{"unchanged", "foo", "bar"}
        modify(&array)
        println(array[0])
}
```

Qui se comporte comme le slice :

```
$ go run reference.go
modified!
```

L'explication est simple : un slice est une structure qui comporte une référence vers un tableau, comme nous pouvons le voir dans le code source du slice:

```
type slice struct {
        array unsafe.Pointer
        len   int
        cap   int
}
```

Par conséquent, lorsque nous passons le slice par valeur, la structure est copiée, mais la référence vers le tableau garde la même valeur, donc nous modifions le même tableau. Le slice d'origine pointe vers le même tableau que nous avons modifié et est donc aussi modifié.

Conclusion

Attention, lorsque vous passez un slice par valeur, tout se passe comme si vous le passiez par référence.

Enjoy!