Les Pièges du Go 4

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.

Supposons que nous avons des structures `User` dont nous voulons extraire tous les noms. Comme il y en a beaucoup, nous voulons faire une liste de pointeurs vers ces noms. Nous pourrions écrire le code suivant (code sur le Playground) :

```
package main

type User struct {
        Name string
}

func getNames(users []User) []*string {
        var names []*string
        for _, user := range users {
                names = append(names, &user.Name)
        }
        return names
}

func main() {
        users := []User{{Name: "foo"}, {Name: "bar"}}
        names := getNames(users)
        for _, name := range names {
                println(*name)
        }
}
```

Mais lorsque nous exécutons ce code, nous obtenons :

```
$ go run broken.go
bar
bar
```

Pourquoi ce code ne fonctionne-t-il pas ? Comment le corriger ?

Explication

Lorsque nous bouclons sur les `users` avec `range`, Go réutilise la même variable pour le `user` et elle est donc à la même adresse mémoire. Donc à chaque itération, l'adresse du champ `Name` du `user` est identique et le pointeur a la même valeur. Donc le nom dans tout le slice a pour valeur le

champ `Name` à la dernière itération.

Pour corriger ce code, nous pouvons créer une variable pour le nom comme ceci (code sur le Playground) :

```
package main

type User struct {
        Name string
}

func getNames(users []User) []*string {
        var names []*string
        for _, user := range users {
                name := user.Name
                names = append(names, &name)
        }
        return names
}

func main() {
        users := []User{{Name: "foo"}, {Name: "bar"}}
        names := getNames(users)
        for _, name := range names {
                println(*name)
        }
}
```

La variable `name` est crée à chaque itération donc le champ `Name` est à une adresse différente et le code fonctionne comme attendu.

Conclusion

Cet exemple est tiré d'un cas réel :

- Description du problème : https://github.com/long2ice/swagin/issues/6
- Diff du correctif : https://github.com/long2ice/swagin/pull/7/files

Enjoy!