Développement Go: Présentation du langage

Stéphane Karraz
ESGI - cours électif de Go - 2022

Objectifs

- Comprendre la syntaxe du Go
- Connaître les atouts et faiblesses du langages
- Utiliser les librairies HTTP pour créer une API
- Exécuter du code en asynchrone

Déroulé de la semaine

- Mise en place de l’environnement de développement
- Présentation de la syntaxe, des types du langage
- Réalisation de petits exercices basiques
- Comment utiliser le Go pour créer un serveur HTTP classique
- TP: réalisation d’une API basique selon une architecture précise
- Go et la gestion de l’exécution de code parallèle: qu’est ce que sont les goroutines et les channels
- TP avec soutenance: simulation d’un serveur de jeu vidéo basé sur le protocole HTTP

Mise en place de l’environnement de développement

https://golang.org/doc/install

Suivre les instructions pour installer le compilateur sur votre machine.

Vous pouvez utiliser votre IDE préféré, le langage est bien supporté.

VSCode permet d’interagir assez efficacement avec le langage:
https://code.visualstudio.com/docs/languages/go

GoLand de chez intelliji fonctionne bien aussi:
https://www.jetbrains.com/fr-fr/go/

Présentation du langage

Imaginé par des ingénieurs de Google avec comme objectifs:

- Facilité d'écriture, ressemblant à un langage de script
- Assimilable rapidement par des étudiants connaissant d'autres langages comme le C, le Python ou d'autres appris durant leur cursus
- Un langage compilé, rapide à exécuter
- Un langage moderne, supportant l'exécution concurrente sur plusieurs coeurs de processeur, pensé pour le Web "de demain"

Syntaxe

```
package main     // Déclaration du module main, point
                 // d'entrée du programme

import (         // Importation des différentes modules, ici
                 // fmt et time
    "fmt"
    "time"
)

func main() {    // première fonction exécuté
    fmt.Println("Hello world. Now the time is", time.Now())
}
```

Déclaration d’une fonction

```
func addNumbers(x int, y int) int {
    return x + y
}
```

Le prototype de la fonction se lit naturellement:
On déclare une fonction nommée “addNumbers” prenant un paramètre nommé “x” de type “int”, un second paramètre “y” aussi de type “int”, et cette fonction renverra une valeur de type “int”.

Déclaration des paramètres

Si des paramètres ont le même type, le compilateur nous permet la simplification suivante:

```
func addNumbers(x, y int) int {
    return x + y
}
```

Retour de fonction

Il n’est pas obligatoire de ne renvoyer qu’une seule valeur au maximum comme ça l’est dans d’autres langages:

```
func addAndSubstractsNumbers(x, y int) (int, int) {
    return x + y,   x - y
}
```

Cette fonction renvoie deux valeurs de type int

Déclaration de variables

Le Go est un langage typé, le type d'une variable ne peut pas changer dans son scope. On peut les déclarer avec un type précis comme ceci:

```
var nb int // On déclare une variable nommée “nb” de type int
nb = 3
```

Nous aurions pu aussi écrire directement:

```
var nb int = 3
```

On peut simplifier cette déclaration en écrivant:

```
nb := 3
```

On peut ainsi déclarer une variable ici nommée nb, et lui attribuer une valeur de 3, et le compilateur va pouvoir généralement déduire le type de la variable attendu.

Appel d’une fonction

Nous pouvons appeler la fonction précédente dans notre main:

```
func main() {
    res1, res2 := addAndSubstractsNumbers(3, 5)
    fmt.Println(“output:”, res1, res2)    // output:  8 -2
}
```

Les types des variables res1 et res2 seront déduits par le compilateurs grâce au prototype de la fonction dont les valeurs de retours auront comme type int et int

Boucles

En Go, nous n’avons que le mot-clef “for” pour déclarer une boucle:

```
for i := 0; i < 5; i++ {
    fmt.println(i)
}
```

Il n’y a pas de parenthèse autour de ce qui compose la boucle.
Le mot-clé “while” n’existe pas, mais on peut se servir du “for” de la même manière:

```
i := 0
for i < 5 {
    i++
}
```

Conditions

Il existe les mots clefs “if”, “else if” et “else” pour déclarer des conditions d’exécution, et comme pour les boucles, une condition n’a pas besoin de parenthèse, mais nécessite par contre des accolades.

```
if input == “connect” {
    fmt.Println(“connecting...”)
} else if input == “disconnect” {
    fmt.Println(“bye”)
} else {
    fmt.Println(“unexpected input”)
}
```

Conditions - 2

Le mot-clé “if” peut aussi s’utiliser directement pour tester une valeur retourné par un appel:

```
if v := math.Pow(x, n); v < lim {
    return v
}
return lim
```

Conditions - 3

Vous pouvez utiliser un switch pour éviter de déclarer une longue suite de “if/else if”. À la différence d’autres langages comme le C, l’exécution s’arrêtera au premier “case” vérifié.

```
switch input {
    case "connect":
     fmt.Println("connecting...")
    case "disconnect":
     fmt.Println("bye.")
    default:
     fmt.Printf("unexpected input: %s.\n", input)
}
```

Package

En Go, nous organisons notre code par package. La présence d'un package nommé `main` indique que nous écrivons un programme exécutable, et non pas juste une librairie partagée de fonctions.

Plusieurs fichiers peuvent appartenir au même package.

Les fonctions et variables déclarés dans un fichier d'un package A seront accessibles dans tous les autres fichiers du package A.

Pour qu'une fonction ou variable soit accessible en dehors du package, elle doit être publique. Pour être publique, elle doit commencer par une lettre majuscule.

Exemple:

```
fmt.Println("hello")
```