Formation Go(lang)

Travaux Pratiques

2022-05-19#4a78795

TP01 - Premiers pas

1. Installation des Go Tools
   - Télécharger les Go Tools https://golang.org/dl/. Sur linux, préférer la version `tar.gz` plutôt que le package manager de votre distribution. Cela permet de récupérer la dernière version de Go et tous les outils du SDK.
   - Si nécessaire ajouter le répertoire `bin/` des Go Tools dans le `PATH` d'exécution
2. Vérifier l'installation : `go version`
3. Copier les tps dans le répertoire de votre choix :

```
formation-go ├── TP01 ├── TP02 ├── TP03 ...
```

4. Dans le répertoire `TP01/`, créer le ficher `HelloWorld.go` et l'exécuter avec la commande `go run .` :

```go
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello, world!")
}
```

TP02 - Packages

Le programme `tp02.go` affiche le nombre de jours qui se sont écoulés à partir du lancement de Go (10 novembre 2009). Le but de cet exercice est de déplacer une partie de ce programme dans un nouveau package `golaunch`.

1. Lancer `tp02.go` avec la commande `go run tp02.go` et vérifier que le message s'affiche bien.
2. Créer le module `formation-go/TP02` en exécutant la commande `go mod init formation-go/TP02`
3. Créer un dossier `golaunch` et :
    - créer un nouveau fichier `date.go` dans le dossier `golaunch/`
4. Déplacer la fonction `getDaysSinceGoLaunch()` dans le fichier `date.go` :
    - Le package de `date.go` doit être `golaunch`
    - Importer le package `golaunch` dans le fichier `tp02.go` : `import "formation-go/TP02/golaunch"`
    - Le nom de la fonction `getDaysSinceGoLaunch()` doit commencer par une majuscule pour qu'elle soit *visible* de l'extérieur
5. Lancer `tp02.go` avec la commande `go run tp02.go` et vérifier que le message s'affiche bien.

TP03 - Variadic functions et closures

Le but de cet exercice est de modifier le code de `tp03.go` pour transformer `isPI()` en une fonction variadique et utiliser des closures pour afficher les messages :

1. Vérifier que le programme fonctionne bien : `go run tp03.go`
2. Dé-commenter le code commenté dans `tp03.go` et vérifier qu'il ne compile plus : `go run tp03.go`
3. Modifier la déclaration de la fonction `isPI()` pour supprimer les erreurs de compilation (N.B. un paramètre variadique est de type `slice`. C'est également le type de la variable `digits` au début de `isPI()`. Même si on n'a pas encore vu le type `slice`, vous pourrez itérer sur le paramètre variadique de la même façon qu'on itère sur la variable `digits`)
4. Assigner les deux variables `successMessage` et `failureMessage` en utilisant deux fonctions

TP04 - Pointeurs et mutabilité

1. Executer `tp04.go` avec la commande `go run tp04.go`
2. Vérifier que l'âge de Ken Thompson n'est pas incrémenté au deuxième affichage
3. Modifier le fichier `database.go` pour que GetPerson renvoie un pointeur
4. Executer `tp04.go` à nouveau, vérifier que l'âge de Ken Thompson s'incrémente bien

TP05 - defer, panic et recover

Le code du tp05 compte de 0 à 10. Le but du TP est de modifier le code en utilisant `defer`, `panic` et `recover` pour avoir l'output suivant :

```
0 1 2 3 4 5 6 7 8 9 10 10 9 8 7 6 5 4 3 2 1 0 done!
```

1. Executer `tp05.go` avec la commande `go run tp05.go`
2. Modifier la fonction `count` en utilisant `defer` pour effectuer un compte à rebours en plus du compte sans utiliser de boucle.
3. Modifier la fonction `count` pour que lorsque `i == to`, on déclenche une panic avec la valeur "done!".
4. Modifier la fonction `main` pour que lorsque la panic se produit, on affiche quand même le message de la panic en terminant normalement le programme.

TP06 - Utilisation des types

1. Définir 2 nouveaux types `Euros` et `Dollars` basés sur `float64` en utilisant la définition de type
2. Dans la fonction `formatMoney`, utiliser le switch de type afin de renvoyer une chaîne représentant le montant correctement formaté
    - Avec le bon sigle (`"?"` si la valeur n'est pas une monnaie mais bien un float)
    - Avec 2 chiffres après la virgule
    - Renvoyer `"??"` si la valeur est de type inconnu
3. Dans la fonction `eurosToDollars`, utiliser la conversion de type afin de renvoyer le résultat de la conversion (utiliser la constante `eurosDollarsRate`)

TP07 - Introduction aux structs

1. Définir un type `book` qui est une `struct` avec les champs suivants : `title`, `author`, `year`.
2. Implémenter la fonction `isBookAuthor()` qui vérifie si l'auteur du livre `book` est `author`.

TP08 - Composition de struct

À partir du code du TP07 changer le type `book` : le champ `author` est maintenant de type `person` :

```
    type book struct {
        title  string
        author person
        year   int
    }
```

1. Compléter la définition du type `person` avec les champs `firstName`, `lastName`
2. Adapter le reste de tp08.go afin qu'il compile

TP09 - Alias de slices et capacité

1. Définir les slices `europeanCountries` et `africanCountries` en utilisant le slice operator sur le slice worldCountries
2. Compléter la boucle `for` en ajoutant les pays de la collection `americanCountries` à `worldCountries`
3. Observer l'évolution de la capacité de `worldCountries`

TP10 - maps

Compléter le code de `tp10.go` afin qu'il affiche la liste des pays et leur capitale :

1. Déclarer la variable `worldCapitals` comme une `map` avec clés et valeurs de type `string`
2. Compléter la boucle `for` afin de peupler `worldCapitals` avec les capitales des différents pays
3. Exécuter `go run tp10.go`

TP11 - if et for

Compléter le code de `tp11.go` :

1. Compléter la boucle `for` dans `main()`
2. Compléter la condition qui vérifie si un pays (`country`) est bien dans la `map` `worldCapitals` . Si c'est le cas le pays est rajouté dans la collection `knownCountries` .
3. Faire en sorte que la collection `knownCountries` soit triée par ordre alphabétique (utiliser le package `sort` de la standard library).

TP12 - Méthodes

1. Créer la méthode `certificationsNum()` qui a comme récepteur un `worker` et qui retourne le nombre de certifications du worker
2. Créer la méthode `isOfficeCompanyHeadquarters()` qui a toujours un `worker` comme récepteur et qui retourne `true` si le bureau du worker se trouve à la même adresse que le headquarter
3. Compléter le `main()` qui itère sur les éléments d'un tableau de `worker` et affiche le nombre de certifications de chaque worker et s'il travaille ou pas au headquarter.

TP13 - Interfaces

On souhaite trier les développeurs par âge.

Pour cela, le type `DevelopersByAge` doit implémenter l'interface `sort.Interface` :

Sur le type `DevelopersByAge` :

1. Implémenter la méthode `Len` qui doit renvoyer la taille de la liste
2. Implémenter la méthode `Less` qui doit déterminer si un élément est "plus petit" qu'un autre
3. Implémenter la méthode `Swap` qui doit intervertir deux éléments

TP - Generics

Find - Trouver un élément dans une liste

La fonction `Find` trouve le premier élément respectant un *predicate* dans une liste.

Un *predicate* est une fonction vérifiant une condition sur un élément, et renvoyant `true` si la condition est vrai et `false` sinon.

Pour le moment la fonction `Find` ne fonctionne que pour le type `int`, vous devez la rendre générique.

Afin de valider le refactoring, exécuter les tests avec la commande suivante :

```
go test -run=Find -v
```

Contains - Vérifier la présence d'un élément dans une liste

La fonction `Contains` vérifie si un élément donné est présent dans une liste.

Pour le moment la fonction `Contains` ne fonctionne que pour le type `int`, vous devez la rendre générique.

Afin de valider le refactoring, exécuter les tests avec la commande suivante :

```
go test -run=Contains -v
```

MeanBy - Calculer une moyenne à partir d'une liste d'éléments

La fonction `MeanBy` calcule une moyenne à partir d'une liste d'éléments.

Elle a besoin d'un *mapper*, une fonction renvoyant un nombre pour chaque élément (exemple : si les éléments sont des étudiants, le *mapper* renvoie une note pour un étudiant donné).

Implémenter la fonction générique `MeanBy` :

- elle doit être capable accepter une liste de n'importe quel type d'élément
- elle doit être capable de renvoyer n'importe quel type de nombre

Afin de valider l'implémentation, exécuter les tests avec la commande suivante :

```
go test -run=MeanBy -v
```

TP14 - goroutines

Dans cet exercice on utilisera la méthode `curl()` qui récupère une page web et affiche le nom, les bytes et le temps écoulé :

```
zenika 37623 [0.36s]
```

`tp14.go` utilise `curl()` de façon séquentielle pour récupérer une liste d'urls. Le temps qu'il emploie pour les récupérer correspond donc à la somme de chaque `curl()` :

```
zenika 37623 [0.36s] facebook 72097 [1.95s] apple 29962 [2.72s] microsoft 73704
```

On veut améliorer `tp14.go` pour récupérer les url en parallèle.

1. Executer `tp14.go` : `go run tp14.go` et noter le temps que le programme emploie pour récupérer les 5 urls
2. Utiliser les goroutines pour que l'exécution de chaque fonction `curl()` se fasse dans une goroutine distincte
3. Mettre un timeout dans `main()` pour attendre la fin de toutes les goroutines (on peut utiliser la fonction `time.Sleep()`) et exécuter

TP15 - channels

En repartant du TP précédent, nous allons utiliser les *channels* pour profiter au maximum de la concurrence des `goroutines`. `curl()` pourra communiquer avec `main()` pour lui dire qu'il a terminé son exécution à l'aide d'un channel.

1. Instancier un `chan` de type `string` au début du `main()`
2. Passer cette variable `chan` comme paramètre à la fonction `curl()`. Celle-ci devra être modifiée afin qu'elle écrive les messages sur le channel et non plus sur `stdout`.
3. À la place de l'instruction `time.Sleep()` dans `main()`, il faudra écrire une boucle `for` qui, pour chaque `goroutine` affichera le message reçu via le channel (`fmt.Print(<-c)`).
4. (Facultatif) Remplacer l'utilisation du `chan` par un `sync.WaitGroup`.

TP16 - select et timeout

`tp16.go` visite récursivement le contenu du dossier passé en paramètre (la fonction `filepath.Walk()` de la librairie standard est utilisée).

```
    $ go run tp16.go .
    filepath.Walk() visited 5 elements
```

La visite des répertoires est faite dans une `goroutine` : `go doWalk(root, c)` Une autre `goroutine` est créée pour le spinner : `go spinner(100 * time.Millisecond)`

Le but de cet exercice est de mettre en place un timeout qui termine le programme si la fonction `doWalk()` ne rend pas la main au bout de 5 secondes. Pour mettre en place le timeout, il faudra utiliser l'instruction `select`.

TP17 - Test

L'objectif est d'écrire des tests pour la fonction `isAnagram()` du TP06.

1. Créer un fichier `tp17_test.go`
2. Écrire des tests dans `tp17_test.go` en s'inspirant du `main()` de `tp17.go`
3. Exécuter les tests : `go test -v`

TP17 bis - Context

`tp17bis.go` génére des nombres aléatoires entre 0 et 9 jusqu'à trouver 8.

Le but de cet exercice est d'ajouter un timeout, à l'aide du package `context`, en prévenant la goroutine générant des nombres qu'elle peut s'arrêter au bout de 6s, ou dès qu'on a trouvé 8.

TP18 - Couverture des tests

1. Analyser la couverture des tests avec `go test -cover formation-go/TP18/anagram`
2. Générer un rapport de profiling avec les deux commandes :

```
go test -coverprofile=c.out formation-go/TP18/anagram
go tool cover -html=c.out -o coverage.html
```

3. Ajouter un test pour faire passer la couverture de code à 100%

TP19 - Benchmark

`tp19.go` contient l'implementation de trois algorithmes de tri : `bubblesort()`, `selectionsort()` et `quicksort()`. L'objectif de ce TP est de comparer les performances de ces trois algorithmes en utilisant l'outil de benchmark de `go`.

1. Créer un fichier de tests `tp19_test.go` avec trois fonctions de test `BenchmarkBubbleSort()`, `BenchmarkSelectionsort()` et `BenchmarkQuickSort()`.

2. Lancer l'outil de benchmark : `go test -bench=. -benchmem`

TP20 - testing.Example

Comme pour le TP18, l'objectif est d'écrire des tests pour la fonction `AreAnagrams()` du TP06. Cette fois-ci en utilisant une fonction `Example`.

1. Dans le dossier `anagram` créer un fichier `anagram_test.go`
2. Écrire une fonction `ExampleAreAnagrams()` dans `anagram_test.go` en s'inspirant du `main()` de `tp20.go`
3. Exécuter les tests : `go test -v`
4. Démarrer un serveur de documentation avec `godoc -http :8080`, et visiter la documentation du package `anagram` : http://localhost:8080/pkg/formation-go/TP20/anagram/

TP21 - gofmt

Le code de `tp21.go` est celui utilisé par l'exercice de refactoring de code Gilded Rose. Nous nous en servirons pour tester les outils de analyse de code.

Lancer `gofmt` avec différentes options :

```
gofmt tp21.go
gofmt -d tp21.go
gofmt -d -s tp21.go
gofmt -l -s tp21.go
gofmt -w -s tp21.go
```

TP22 - go vet et golint

Exécuter les commandes suivantes et analyser leurs outputs :

```
go vet tp22govet.go
golint tp22golint.go
```

TP23 - Gestion de dépendance

Dans ce TP on développera une application qui utilise une librairie externe `github.com/briandowns/openweathermap` pour utiliser les API de `openweathermap` (https://home.openweathermap.org/).

1. Essayer d'exécuter le programme : `go run tp23.go`
2. Créer le package liée à ce mini projet: `go mod init formation-go/TP23`
3. Récupérer la dépendance : `go get github.com/briandowns/openweathermap`
4. Re-essayer d'éxécuter le programme. Cette fois-ci vous devrez avoir un output de ce type :

```
Paris: ensoleillé, 4.1C Cannes: ensoleillé, 3.3C Rome: ensoleillé, 19.3C
```

TP24 - Web service et JSON

Dans ce TP nous allons développer une API REST qui renvoie la météo d'une ville. Les données météo sont récupérées du service MetaWeather. Le TP se déroule en 3 phases :

1. Compléter le client Metaweather pour décoder les réponses json :
   - Lancer les test de l'API yahoo et vérifier qu'ils ne passent pas : `go test formation-go/TP24/metaweather`
   - Compléter les deux fonctions `unmarshalWeatherResult` et `unmarshalWoeidResult` du fichier `metaweather_api.go` qui font le mapping entre le format json (`data`) et les struct de type `weatherQueryResult` et `woeidQueryResult` (`result`)
   - Une fois les fonctions complétées vérifier que les tests passent
2. Ajouter la route pour servir l'API REST /cities en utilisant le package `http` de la standared library`
   - Lancer le serveur `go run tp24.go`. Vérifier que le serveur répond bien à l'adresse http://localhost:8000/
   - Dans le `main` de `tp24.go` ajouter un deuxième appel à `http.HandleFunc()`. Le path à servir sera `/cities` et `citiesHandler()` sera la fonction qui servira ce deuxième path.
   - Relancer le serveur et vérifier que l'adresse http://localhost:8000/cities répond une liste d'objets de type `Weather` en format json.
3. Utiliser un package de routage plus avancé (github.com/gorilla/mux) pour compléter l'API.
   - Dans le `main` de `tp24.go` commenter les premières lignes, celles ou le routage est fait avec le package `http` de la standard library. Et décommenter les lignes ou le routage est faite avec le package `mux` qui offre plus de options par rapport au premier.
   - Relancer le serveur et vérifier qu'il répond correctement aux URL http://localhost:8000/ et http://localhost:8000/cities
   - Ajouter une option de routage pour servir les path du type `/cities/{name}` avec la fonction `cityHandler`
   - Relancer le serveur et vérifier que l'application répond bien aux adresses :
     - http://localhost:8000/
     - http://localhost:8000/cities
     - http://localhost:8000/cities/Paris
     - http://localhost:8000/cities/Rome
     - http://localhost:8000/cities/Cannes

TP25 - Démarrons avec une nouvelle Application

1. Boostrap de l'application

```
$ go mod init formation-go/TP25
```

2. Démarrons avec une petite application qui renvoie just un simple HelloWorld via un server HTTP.

    - Ecrire une petite application qui renvoie Hello World quand on lance une requête HTTP sur le port 8080:

      ```
      curl localhost:8080
      Hello World!
      ```

3. Améliorations et Refactoring

    - Externaliser la fonction handlerFunc gérant une requête sur "/" en une fonction helloHandlerFunc,
    - Encapsuler l'utilisation http.ListeAndServe dans un package specifique,
    - Ajouter une log pour indiquer sur quel port à démarrer le serveur HTTP,
    - Déplacer la function helloHandlerFunc vers un package spécifique,
    - Créer un logger spécifique à votre application,
    - Ajouter une trace dans votre handler "/",
    - Externaliser l'addresse d'écoute du serveur HTTP afin qu'on puisse la modifier au lancement.(aka: native flag package would be helpful),

4. Ajout de fonctionnalité via des "Middleware".

   Pour cela, créer un package middleware et ajouter les fonctions suivantes:

    - tracer chaque requêtes envoyé au serveurs,
    - implémenter une pseudo sécurisation via jeton (une vérification de l'existance d'un jeton est suffisante)

5. Bonus: Faire en sorte que votre logger trace selon un niveau de log pouvant être paramétrable (aka: custom type would be helpful),

TP26 - Utilisons désormais des libs externes

Dans ce TP, nous allons désormais utiliser des bibliothèques plutôt que d'utiliser directement la bibliothèque standard Go. Les libs utilisés seront les suivants:

- log : logrus under MIT License
- cobra: cobra under Apache-2.0 License
- macaron: macaron under Apache-2.0 License

Objectif

Refactorer le code du TP précédents (TP25) pour passer des libs standards aux différentes library ci dessus.

Petit tips: penser à y aller petit à petit