DEV1 : les modules en Go

loig.jezequel@univ-nantes.fr

Objectifs du cours

Pour l'instant

- Un fichier par exercice : pas pratique pour un gros projet.
- Toujours un main : pas adapté pour une bibliothèque.

Bibliothèque

Ensemble de fonctions prêtes à être utilisées par des programmes.

Le cours d'aujourd'hui

- Comment organiser son code pour un gros projet ?
- Comment fabriquer ses propres bibliothèques ?
- Comment gérer les dépendances de son code à des bibliothèques externes ?

La solution à tous nos problèmes : les modules

Organiser son code

Au sein d'un module on peut organiser son code en plusieurs fichiers et dossiers pour bien séparer les différentes fonctionnalités.

Fabriquer ses bibliothèques

Un module est un moyen de compartimenter son code et de le distribuer aux autres programmeurs.

Gérer les dépendances

Un module enregistre les bibliothèques utilisées par le code qu'il contient avec des informations précises sur la version exacte à utiliser

Qu'est-ce qu'un module ?

Paquet (*package*)

Collection de fichiers sources, tous situés dans le même répertoire, et qui sont compilés ensemble.

- Code partagé entre les fichiers (une fonction définie dans un fichier est utilisable dans un autre).
- Unité d'importation (en Go on importe des paquets).

Module

Collection de paquets qui sont publiés, versionnés et distribués ensemble.

- Identifié par un chemin (*module path*).
- Fichier go.mod à la racine.
- Fichier go.sum à la racine.

Organisation d'un module

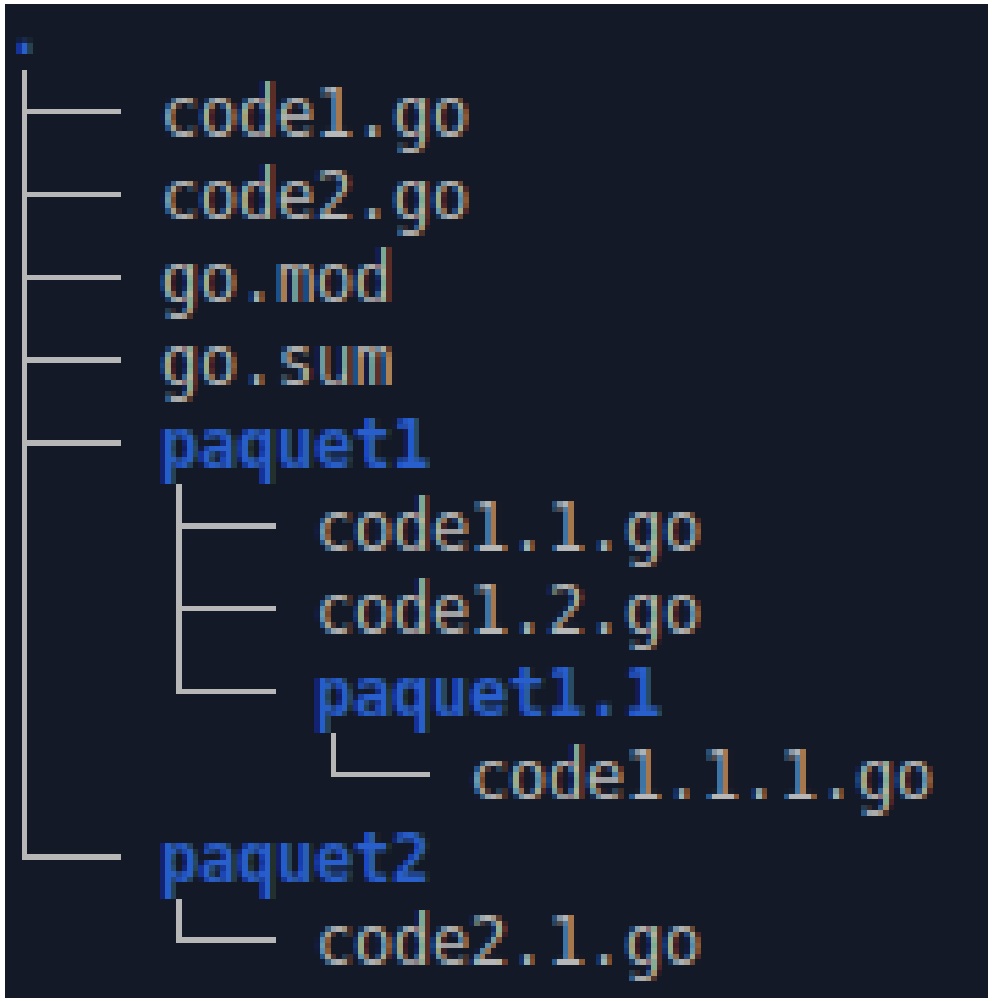

Figure: Organisation classique d'un module

Organisation d'un module, suite

```
module github.com/loig/monmodule

go 1.17

require github.com/hajimehoshi/ebiten v1.12.12

require (
        github.com/go-gl/glfw/v3.3/glfw v0.0.0-20200707082815-5321531c36a2 // indirect
        golang.org/x/exp v0.0.0-20190731235908-ec7cb31e5a56 // indirect
        golang.org/x/image v0.0.0-20200801110659-972c09e46d76 // indirect
        golang.org/x/mobile v0.0.0-20210208171126-f462b3930c8f // indirect
        golang.org/x/sys v0.0.0-20200918174421-af09f7315aff // indirect
)
```

Figure: Contenu d'un fichier go.mod

```
github.com/BurntSushi/xgb v0.0.0-20160522181843-27f122750802/go.mod h1:IVnqGOEym/WlBOVXweHU+Q+/VP0lqqI8lqeDx9IjBqo=
github.com/go-gl/glfw/v3.3/glfw v0.0.0-20200707082815-5321531c36a2 h1:Ac1OEHHkbAZ6EUnJahF0GKcU0FjPc/V8F1DvjhKngFE=
github.com/go-gl/glfw/v3.3/glfw v0.0.0-20200707082815-5321531c36a2/go.mod h1:tQ2UAYgL5IevRw8kRxooKSPJfGvJ9fJQFa0TUsXzTg8=
github.com/gofrs/flock v0.8.0 h1:MSdYCl1jsF3PbENUUEx85nkWfJSGfzYI9yEBZOJz6CY=
github.com/gofrs/flock v0.8.0/go.mod h1:F1TvTiK9OcQqauNUHlbJvyl9Qa1QvF/gOUDKA14jxHU=
github.com/golang/freetype v0.0.0-20170609003504-e2365dfdc4a0/go.mod h1:E/TSTwGwJL78qG/PmXZO1EjYhfJinVAhrmmHX6Z8B9k=
github.com/hajimehoshi/bitmapfont v1.3.0/go.mod h1:/Qb7yVjHYNUV4JdqNkPs6BSZwLjKqkZOMIp6jZD0KgE=
github.com/hajimehoshi/ebiten v1.12.12 h1:JvmF1bXRa+t+/CcLWxrJCRsdjs2GyBYBSiFAfIqDFlI=
github.com/hajimehoshi/ebiten v1.12.12/go.mod h1:1XI25ImVCDPJiXox4h9yK/CvN5sjDYnbF4oZcFzPXHw=
github.com/hajimehoshi/file2byteslice v0.0.0-20200812174855-0e5e8a80490e/go.mod h1:CqqAHp7Dk/AqQiwuhV1yT2334qbA/tFWQW0MD2dGqUE=
```

Figure: Contenu (partiel) d'un fichier go.sum

Créer un module

Commande (dans le dossier où sera le code)

go mod init chemin

Choix du chemin

Le chemin d'un module permettra de l'importer dans d'autres programmes. Il doit respecter certaines règles.

Bibliothèque publique le chemin est l'adresse où la trouver.

Bibliothèque privée le chemin est ce qu'on veut.

Programme exécutable le chemin est ce qu'on veut.

Ceci permet d'assurer qu'il n'y a pas deux bibliothèques (modules) utilisables par tous qui ont le même chemin.

Compiler un module

Commande (à la racine du module)

go build

Main

Si votre module contient de quoi s'exécuter à la racine (package main + fonction main), un exécutable du nom du module sera produit.

Commandes alternatives (à la racine du module)

Installation go install
Les fichiers compilés sont placés dans l'espace de travail Go (variable d'environnement).

Exécution go run
Seulement si le module contient un main, il s'exécute directement.

Créer des nouveaux paquets

Pour quoi faire ?

Rendre le code plus lisible, mieux organisé, en le séparant en plusieurs parties plus ou moins indépendantes.

Un exemple

Le paquet *image* de la bibliothèque standard (création et manipulation d'images en Go) contient un paquet dédié au traitement des images png et un autre dédié aux images jpeg.

Comment ?

1. Créer un dossier avec le nom du paquet,
2. faire commencer les fichiers source Go placés dans ce paquet par *package xxx* où xxx est le nom du paquet.

Utiliser des paquets

Nom long pour l'importation

Pour importer un paquet on utilise son nom long, c'est à dire le chemin du module suivi du chemin du paquet depuis la racine du module.

Nom court pour l'utilisation

Pour utiliser un paquet on se sert de son nom court, c'est à dire le nom du dossier où sont les sources (qui est aussi celui qu'on indique au début des fichiers du paquet).

Visibilité, attention

Les fonctions/variables/types définis dans un paquet sont visibles (c'est-à-dire utilisable à l'extérieur du paquet) si et seulement si leur nom commence par une majuscule.

Paquets provenant d'un autre module

Bibliothèque standard

- Nom court pour importer et utiliser.

Module disponible publiquement

- Nom long pour importer (et nom court pour utiliser).
- Commande go mod tidy pour récupérer automatiquement le paquet si nécessaire, ainsi que mettre à jour automatiquement go.mod et go.num.

Module M non disponible publiquement

- Nom long pour importer (et nom court pour utiliser).
- Ajouter replace M => chemin dans go.mod.
- Commande go mod tidy pour récupérer le paquet si nécessaire, ainsi que mettre à jour automatiquement go.mod et go.num.

Quelques références

Structurer son code Go

`https://golang.org/doc/code`

Gérer les dépendances de ses programmes

`https://golang.org/doc/modules/managing-dependencies`

Fonctionnement précis des modules

`https://golang.org/ref/mod`