

Services REST en GoLang

GoLang



📌 **Les grandes lignes**

📌 **Installation**

📌 **HelloWorld**

★ Et la cross compilation :)

GoLang : Les grandes lignes (1/2)



Langage compilé et concurrent

- ★ Première version en 2009
- ★ Créé pour la programmation système puis applicative
- ★ compilation statique, possibilité de cross-compilation vers différentes cibles
- ★ Concurrence intégrée au langage dès le début au travers des « goroutines » qui sont une variante des coroutines
- ★ Communication entre les goroutines par des « channels »
- ★ Typage statique fort fondé sur l'inférence de types avec possibilité de typage explicite
- ★ Mémoire gérée par un ramasse-miettes
- ★ Pas de programmation générique, pas de surcharge de méthodes, pas d'arithmétique des pointeurs, pas d'exception

Créé par des « habitués »

- ★ Ken Thompson (B, Unix, UTF-8) (Bell Labs, Google)
- ★ Rob Pike (Unix, Plan9, UTF-8, Limbo) (Bell Labs, Google)
- ★ Robert Griesemer (Google)

GoLang : Les grandes lignes (2/2)



Pourquoi apprendre GoLang ?

- ★ S'apprend en une journée, se maîtrise en une semaine (voir moins)
- ★ Orienté réseau et concurrence
- ★ Support natif de JSON et HTTP(S)
- ★ Un vrai compilateur
- ★ Poussé par une communauté
- ★ Cross compilation intégrée nativement dans le compilateur
- ★ Il n'est pas révolutionnaire, donc on s'y retrouve très vite, mais ce qu'il fait il le fait bien et efficacement

- ★ Ah oui, c'est Google ... ok et ?

GoLang : Installation (1/2)

<https://golang.org/dl/>

- ★ `cd /usr/local && tar zxf ~/go1.8.3.linux-amd64.tar.gz`
- ★ `export PATH=$PATH:/usr/local/go/bin`
- ★ `mkdir GoWork && cd GoWork/`
- ★ `mkdir bin pkg src`
- ★ `export GOPATH=`pwd``

GOPATH

- ★ La variable d'environnement GOPATH doit pointer sur tous les dossiers ou vous avez placé des ressources nécessaires a la compilation, donc code source, code compilé ou librairies
- ★ Chaque répertoire pointé par une entrée de GOPATH va contenir une structure de 3 répertoires
 - bin
 - pkg
 - src

GoLang : Installation (2/2)



GOPATH

- ★ Il peut être intéressant de séparer les librairies que l'on va récupérer par `go get` du code de notre application
- ★ Par exemple dans GOPATH pointer sur un répertoire (GoTools) dans le quel on va retrouver un certain nombre de binaires (dans bin)
 - `gocode`, `golint`, `gorename`, `godef`,
- ★ ou des librairies (dans pkg)
 - `lib/pq`
- ★ ou des sources (dans src)
 - `src/github.com/fromkeith/gorest/`

GoLang : Hello World (1/3)

Création d'un environnement de compilation Go

- ★ mkdir GoWork && cd GoWork
- ★ mkdir bin pkg src
- ★ export GOPATH=`pwd`

Création du projet

- ★ mkdir src/HelloWorld

Création du HelloWorld

- ★ vi src/HelloWorld/myFistHello.go

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Finished")
}
```

GoLang : Hello World (2/3)

Compilation

★ go build HelloWorld

- Va compiler le programme et générer le binaire dans le répertoire courant

Installation

★ go install HelloWorld

- Va compiler le programme et générer le binaire dans le répertoire d'installation pointé par GOPATH (donc le répertoire bin)

Cross Compilation

★ export GOARCH=arm

★ export GOOS=linux

★ go build HelloWorld

GoLang : Hello World (3/3)

Exemple de cross compilation

```
[iMacPtro:GoWork ptro$ uname -a
Darwin iMacPtro.local 16.7.0 Darwin Kernel Version 16.7.0: Thu Jun 15 17:36:27 PDT 2017
; root:xnu-3789.70.16~2/RELEASE_ARM_T8020 armv7t8m
[iMacPtro:GoWork ptro$ export GOARCH=arm
[iMacPtro:GoWork ptro$ export GOOS=linux
[iMacPtro:GoWork ptro$ go build HelloWorld
[iMacPtro:GoWork ptro$ ./HelloWorld
-bash: ./HelloWorld: cannot execute binary file
[iMacPtro:GoWork ptro$ scp HelloWorld pi@192.168.0.31:/home/pi
HelloWorld                                100% 1412KB  10.6MB/s   00:00
[iMacPtro:GoWork ptro$ ssh pi@192.168.0.31

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Oct 30 15:22:23 2017 from 192.168.0.30
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.9.28-v7+ #998 SMP Mon May 15 16:55:39 BST 2017 armv7l GNU/Linux
pi@raspberrypi:~ $ ./HelloWorld
Finished
pi@raspberrypi:~ $ █
```

Serveur REST JSON



- 📌 **Les dépendances**
- 📌 **Marshalling / Unmarshalling JSON**
- 📌 **Package Rest**
- 📌 **GET**
- 📌 **POST**
- 📌 **PUT**
- 📌 **DELETE**
- 📌 **Exemple complet jusqu'à la base PostgreSQL**

Les dépendances

On commence par se créer un environnement pour installer les dépendances

- ★ mkdir GoTools ; cd GoTools
- ★ mkdir src bin pkg
- ★ export GOPATH=`pwd`

Ensuite on demande à Go d'y télécharger les dépendances

- ★ go get -u github.com/fromkeith/gorest
 - Serveur gorest, cette commande va rapatrier les sources de gorest, leurs dépendances, et compiler la librairie
- ★ go get -u github.com/lib/pq
 - La librairie postgres si vous souhaitez connecter votre serveur REST à une base Postgres

il existe des gestionnaires de package

- ★ <https://github.com/Masterminds/glide>

Marshalling / Unmarshalling JSON (1/5)



 Directement dans des types Go

```
Magasin.go  x
package main

type Magasin struct {
    Idt int64 `json:"idt,omitempty"`
    Nom string `json:"nom,omitempty"`
}

func NewMagasin(idt_ int64, nom_ string) *Magasin {
    return &Magasin{Idt: idt_, Nom: nom_}
}
```

Marshalling / Unmarshalling JSON (2/5)



- 📌 Il va créer en mémoire une instance de Magasin avec les valeurs trouvées dans le JSON

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
)

func testUnmarshallOneMagasin() {
    magasinJSON := `{"idt":3,"nom":"Toulouse"}`
    fmt.Printf("testMarshallOneMagasin %s\n", magasinJSON)
    magasin := new(Magasin)
    errUnmarshal := json.Unmarshal([]byte(magasinJSON), magasin)
    if errUnmarshal != nil {
        log.Fatal(errUnmarshal)
    } else {
        fmt.Printf("testUnmarshallOneMagasin//Unmarshal %+v\n", magasin)
    }
}
```

Marshalling / Unmarshalling JSON (3/5)

- 📌 Ou un tableau si c'est un tableau qui est dans le JSON

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
)

func testUnmarshallArrayOfMagasins() {
    magasinsJSON := `[{"idt":3,"nom":"Toulouse"}, {"idt":2,"nom":"Gourdon"}]`
    var magasins []Magasin
    errUnmarshal := json.Unmarshal([]byte(magasinsJSON), &magasins)
    if errUnmarshal != nil {
        log.Fatal(errUnmarshal)
    } else {
        for _, mag := range magasins {
            fmt.Printf("testUnmarshallArrayOfMagasins//Unmarshal %+v\n", mag)
        }
    }
}
```

Marshalling / Unmarshalling JSON (4/5)

- 📌 Si la structure du JSON est plus complexe il est possible de revenir à un parsing plus traditionnel

```
func testUnmarshallComposite() {
    unknowJSON := `{"idt":1,"nom":"Toulouse","rayons":[{"idt":1,"nom":"Eaux"},{"idt":2,"nom":"Viandes"}]}`
    var parsed map[string]interface{}
    errUnmarshal := json.Unmarshal([]byte(unknowJSON), &parsed)
    if errUnmarshal != nil {
        log.Fatal(errUnmarshal)
    } else {
        idtMagasin := parsed["idt"].(float64)
        nomMagasin := parsed["nom"].(string)
        magasin := NewMagasin(int64(idtMagasin), nomMagasin)
        fmt.Printf("testUnmarshallComposite//Unmarshal %+v\n", magasin)

        arrayOfRayons := parsed["rayons"].([]interface{})
        for _, oneRayon := range arrayOfRayons {
            rayonMap := oneRayon.(map[string]interface{})
            idtRayon := rayonMap["idt"].(float64)
            nomRayon := rayonMap["nom"].(string)
            fmt.Printf("    Rayon :: %d %s\n", int64(idtRayon), nomRayon)
        }
    }
}
```


Marshalling / Unmarshalling JSON (5/5)



Tests avec le programme d'exemple

★ go install Marshalling && ./bin/Marshalling

```
testMarshallOneMagasin {"idt":3,"nom":"Toulouse"}
testUnmarshallOneMagasin//Unmarshal &{Idt:3 Nom:Toulouse}
testUnmarshallArrayOfMagasins//Unmarshal {Idt:3 Nom:Toulouse}
testUnmarshallArrayOfMagasins//Unmarshal {Idt:2 Nom:Gourdon}
testUnmarshallComposite//Unmarshal &{Idt:1 Nom:Toulouse}
  Rayon :: 1 Eaux
  Rayon :: 2 Viandes
Finished
```

Il existe des librairies pour vous aider sur GitHub

★ <https://github.com/jmoiron/jsonq>

Pour créer votre serveur REST il va falloir réaliser 3 étapes

- ★ Définir les types qui vont devenir du JSON
- ★ Définir un service par type de service que vous voulez
- ★ Charger tout ça dans le main()

Définir les types

- ★ Ce sont des types GoLang comme on a vu Magasin dans les slides précédents, ils contiennent le mapping JSON

Défini un service par type de service

- ★ Ce sont vos services REST, cela va prendre la forme d'un type struct Go dans lequel on va retrouver
 - Le goest.RestService, avec le path, ce qu'il consomme et produit ainsi que d'autres parametres (charset, allowGzip, ...)
 - Le binding entre les EndPost et les verbes HTTP

Package Rest (1/3)



HelloService.go

```
package main

import (
    "github.com/fromkeith/gorest" // go get github.com/fromkeith/gorest
)

//*****Define Service*****
type HelloService struct {
    gorest.RestService `root:"/first/"`

    helloWorld gorest.EndPoint `method:"GET" path:"/hello/" output:"string"`
}

func (serv HelloService) HelloWorld() string {
    serv.ResponseBuilder().SetResponseCode(200)
    return "GoLang Hello World"
}
```

Package Rest (2/3)

FirstRestServer.go

```
package main

import "net/http"
import (
    "fmt"
    "github.com/fromkeith/gorest"
)

func main() {

    helloService := new(HelloService)
    gorest.RegisterService(helloService)

    http.Handle("/", gorest.Handle())
    http.ListenAndServe(":8181", nil)
    fmt.Print("Finished")
}
```

Package Rest (3/3)



Compilation execution et test

★ go build FirstRestServer

★ ./FirstRestServer

2017/10/30 16:18:31 All EndPoints for service [HelloService] , registered under root path: /first/
2017/10/30 16:18:31 Registered service: HelloService endpoint: GET first/hello

```
http://localhost:8080:> get http://localhost:8181/first/hello
> GET http://localhost:8181/first/hello

< 200 OK
< Content-Type: application/json
< Date: Mon, 30 Oct 2017 15:19:15 GMT
< Content-Length: 18
<
"GoLang Hello World"
```

Cas d'étude : Les magasins d'une enseigne

Magasin

- idt
- nom
- Tableau de Rayons

Rayon

- idt
- nom
- nomImage
- Fait partie d'un magasin

GET Sans Paramètres (1/3)

📌 Liste des magasins

★ définition du type JSON qui sera retourné : Magasin

```
package main

type Magasin struct {
    Idt int64 `json:"idt,omitempty"`
    Nom string `json:"nom,omitempty"`
}

func NewMagasin(idt_ int64,nom_ string) *Magasin {
    return &Magasin{Idt:idt_,Nom:nom_}
}
```

```
MagasinDAO.go x Magasin.go x Ma
package main

var bouchon []Magasin

func loadMagasins() []Magasin {
    bouchon := make([]Magasin, 2)

    bouchon[0] = *NewMagasin(0, "Toulouse")
    bouchon[1] = *NewMagasin(1, "Gourdon")

    return bouchon
}
```

GET Sans Paramètres (2/3)

Liste des magasins

★ définition du service

```

MagasinService.go  x  MagasinDAO.go  x  Magasin.go  x  RESTServer.go  x
package main

import (
    "fmt"
    "github.com/fromkeith/gorest" // go get github.com/fromkeith/gorest
)

//*****Define Service*****
type MagasinService struct {
    gorest.RestService `root:"/StoreWS/api/magasin/" consumes:"application/json" produces:"application/json"`

    magasinList gorest.EndPoint `method:"GET" path:"/" output:"[]Magasin"`
}

func (serv MagasinService) MagasinList() []Magasin {
    serv.ResponseBuilder().AddHeader("Access-Control-Allow-Origin", "*")

    fmt.Printf("MagasinList\n")

    serv.ResponseBuilder().SetResponseCode(200)
    return loadMagasins()
}

```

GET Sans Paramètres (3/3)

Execution

- ★ export GOPATH=/Users/ptro/Dropbox/GoLangRest/GoTools:`pwd`
- ★ go build ListMagasins
- ★ ./ListMagasins

```
$ ./ListMagasins
2017/10/31 11:24:02 All EndPoints for service [MagasinService] , registered under root path: /StoreWS/api/
magasin/
2017/10/31 11:24:02 Registered service: MagasinService endpoint: GET StoreWS/api/magasin
```

```
[http://localhost:8080:> get http://localhost:8080/StoreWS/api/magasin
> GET http://localhost:8080/StoreWS/api/magasin

< 200 OK
< Access-Control-Allow-Origin: *
< Content-Type: application/json
< Date: Tue, 31 Oct 2017 10:26:49 GMT
< Content-Length: 46
<
[ {
  "nom" : "Toulouse"
}, {
  "idt" : 1,
  "nom" : "Gourdon"
} ]
```


GET Avec Paramètre (1/3)

📌 Récupération d'un seul Magasin

★ On complète juste le bouchon du DAO

```
MagasinDAO.go x MagasinService.go x RESTServer.go x
package main

var bouchon []Magasin

func loadMagasins() []Magasin {
    bouchon := make([]Magasin, 2)

    bouchon[0] = *NewMagasin(0, "Toulouse")
    bouchon[1] = *NewMagasin(1, "Gourdon")

    return bouchon
}

func loadMagasin(idt_ int64) *Magasin {
    bouchon := make([]Magasin, 2)

    bouchon[0] = *NewMagasin(0, "Toulouse")
    bouchon[1] = *NewMagasin(1, "Gourdon")

    if idt_ == 0 {
        return &bouchon[0]
    } else if idt_ == 1 {
        return &bouchon[1]
    }
    return nil
}
```

GET Avec Paramètre (2/3)

📌 Récupération d'un seul Magasin

★ Modification du service

```

MagasinService.go x MagasinDAO.go x RESTServer.go x Magasin.go x
package main

import (
    "fmt"
    "github.com/fromkeith/gorest" // go get github.com/fromkeith/gorest
)

//*****Define Service*****
type MagasinService struct {
    gorest.RestService `root:"/StoreWS/api/magasin/" consumes:"application/json" produces:"application/json"`

    magasinList    gorest.EndPoint `method:"GET" path:"/" output:"[]Magasin"`
    magasinDetails gorest.EndPoint `method:"GET" path:"/{Id:int64}" output:"Magasin"`
}

func (serv MagasinService) MagasinList() []Magasin { ...
}

func (serv MagasinService) MagasinDetails(Id int64) (m Magasin) {
    serv.ResponseBuilder().AddHeader("Access-Control-Allow-Origin", "*")

    fmt.Printf("MagasinDetails parameter %d\n", Id)
    magasin := loadMagasin(Id)
    if nil != magasin {
        serv.ResponseBuilder().SetResponseCode(200)
        return *magasin
    } else {
        serv.ResponseBuilder().
            SetResponseCode(500).
            WriteAndOverride([]byte("{\"message\" : \"Le Magasin indique n'existe pas\" }"))
    }

    return
}

```

GET Avec Paramètres (3/3)

Execution

- ★ export GOPATH=/Users/ptro/Dropbox/GoLangRest/GoTools:`pwd`
- ★ go build OneMagasins
- ★ ./OneMagasins

```
$ ./OneMagasin
```

```
2017/10/31 11:46:32 All EndPoints for service [MagasinService] , registered under root path: /StoreWS/api/magasin/
```

```
2017/10/31 11:46:32 Registered service: MagasinService endpoint: GET StoreWS/api/magasin
```

```
2017/10/31 11:46:32 Registered service: MagasinService endpoint: GET StoreWS/api/magasin/{Id:int64}
```

```
[http://localhost:8080:> get http://localhost:8080/StoreWS/api/magasin/1  
> GET http://localhost:8080/StoreWS/api/magasin/1
```

```
< 200 OK  
< Access-Control-Allow-Origin: *  
< Content-Type: application/json  
< Date: Tue, 31 Oct 2017 10:42:22 GMT  
< Content-Length: 25  
<  
{  
  "idt" : 1,  
  "nom" : "Gourdon"  
}
```

Le JSON du body de la requête est directement transformé

- ★ Dans le postdata nous allons indiquer l'objet vers lequel effectuer le marshalling du JSON
- ★ Ainsi dans la méthode on manipulera directement le type correspondant

Modification du DAO

```
MagasinDAO.go  x  Magasin.go  x  MagasinService.go  x
package main

var nbMagasin int64
var bouchon []Magasin

func loadMagasins() []Magasin { ...
}

func loadMagasin(idt_ int64) *Magasin { ...
}

func createMagasin(nom_ string) *Magasin {
    bouchon := make([]Magasin, 3)

    bouchon[0] = *NewMagasin(0, "Toulouse")
    bouchon[1] = *NewMagasin(1, "Gourdon")
    nbMagasin = 2

    magasin := NewMagasin(nbMagasin, nom_)
    bouchon[nbMagasin] = *magasin
    nbMagasin++
    return magasin
}
```

Ajout du Service

```
MagasinService.go x MagasinDAO.go x Magasin.go x RESTServer.go x
package main

import (
    "fmt"
    "github.com/fromkeith/gorest" // go get github.com/fromkeith/gorest
)

//*****Define Service*****
type MagasinService struct {
    gorest.RestService `root:"/StoreWS/api/magasin/" consumes:"application/json" produces:"application/json"`

    magasinList    gorest.EndPoint `method:"GET" path:"/" output:"[]Magasin"`
    magasinDetails gorest.EndPoint `method:"GET" path:"/{Id:int64}" output:"Magasin"`
    postMagasin    gorest.EndPoint `method:"POST" path:"/" postdata:"Magasin" output:"Magasin"`
}

func (serv MagasinService) PostMagasin(m Magasin) (mag Magasin) {
    serv.ResponseBuilder().AddHeader("Access-Control-Allow-Origin", "*")

    fmt.Printf("AddMagasin\n")

    if len(m.Nom) == 0 {
        serv.ResponseBuilder().
            SetResponseCode(400).
            WriteAndOverride([]byte(`{"message" : "Il faut indiquer le nom du magasin dans le json en entree"}`))
    } else {
        mag := createMagasin(m.Nom)
        if nil != mag {
            serv.ResponseBuilder().SetResponseCode(200)
            return *mag
        }
    }
    return
}
```

Execution

- ★ export GOPATH=/Users/ptro/Dropbox/GoLangRest/GoTools:`pwd`
- ★ go build PostMagasin
- ★ ./PostMagasin

```
$ ./PostMagasin
```

```
2017/10/31 12:16:03 All EndPoints for service [MagasinService] , registered under root path: /StoreWS/api/magasin/
```

```
2017/10/31 12:16:03 Registered service: MagasinService endpoint: GET StoreWS/api/magasin
```

```
2017/10/31 12:16:03 Registered service: MagasinService endpoint: GET StoreWS/api/magasin/{Id:int64}
```


```
2017/10/31 12:16:03 Registered service: MagasinService endpoint: POST StoreWS/api/magasin
```

```
[http://localhost:8080:> post http://localhost:8080/StoreWS/api/magasin --data {"nom":"Lille"}
> POST http://localhost:8080/StoreWS/api/magasin

< 200 OK
< Access-Control-Allow-Origin: *
< Content-Type: application/json
< Date: Tue, 31 Oct 2017 11:21:50 GMT
< Content-Length: 23
<
{
  "idt" : 2,
  "nom" : "Lille"
}
```

- 📌 **PUT fonctionne comme POST, les données sont reçues aussi dans le body de la requête en JSON**
- ★ Rien ne change par rapport à POST, seule l'action effectuée sera un remplacement au lieu d'une création

DELETE

 **Le DELETE va prendre ses paramètres sur l'URL, cela fonctionne donc comme un GET avec un paramètre**

★ Rien ne change par rapport au GET avec un paramètre dans l'URL, seule l'action effectuée sera une suppression au lieu d'une interrogation

Exemple complet



Dans les projets exemple ouvrir le projet GoStoreWS

- ★ 3 services : Magasin, Produit, Rayon
- ★ Les principaux exemples de requêtes
 - GET
 - POST
 - PUT
 - DELETE
- ★ Gestion de la base de données par des DAO

Pour aller un peu plus loin (1/3)

Pool de connexion ?

★ <https://golang.org/pkg/database/sql/>

▸ Type DB : SetMaxIdleConns

★ <http://go-database-sql.org/connection-pool.html>

JWT ?

★ <https://github.com/dgrijalva/jwt-go>

Cryptographie ?

★ <https://github.com/jamesruan/sodium>

★ <https://github.com/jasonmccampbell/GoSodium>

★ <https://github.com/GoKillers/libsodium-go>

Pour aller un peu plus loin (2/3)

Invoquer d'autres WebServices

```
res, err := http.Get("http://localhost:8080/StoreWS/api/magasin")
```

```
strBody := "{\"nom\":\"" + name + "\"}"  
res, err := http.Post("http://localhost:8080/StoreWS/api/magasin", "application/json",
```

```
strBody := "{\"idt\":\"" + fmt.Sprintf("%d", idt) + "\", \"nom\":\"" + name + "\"}"  
  
client := &http.Client{  
request, err := http.NewRequest("PUT", "http://localhost:8080/StoreWS/api/magasin",  
strings.NewReader(strBody))  
request.ContentLength = int64(len(strBody))  
response, err := client.Do(request)
```

Regardez le projet GoClientWS qui permet d'invoquer les WS dont nous avons parlé

Pour aller un peu plus loin (3/3)

📌 Dans cette présentation le choix a été fait d'utiliser gorest, ce n'est pas la seule solution

📌 Framework HTTP pour GoLang

★ <https://github.com/gin-gonic/gin>

📌 URL Router dispatcher pour GoLang

★ <https://github.com/gorilla/mux>

📌 Les alternatives ne manquent pas sur GitHub