**GO安裝與執行**

1. 下載Golang : https://go.dev/dl/go1.18.2.windows-amd64.msi
2. 終端機 :
   * $ apt install golang-go –y
3. 預設的GOROOT(可用$go env GOROOT檢查) :

* **C:\go (Windows)**
* **/root/go (Linux)**
* 放內建函式庫

1. 預設的GOPATH(可用$go env GOPATH檢查) :

* **C:\Users\USER\go (Windows)**
* **/Users/USER/go (Linux)**
* 放第三方套件

1. 自己的程式碼(潛規則) :

* **建立一個專案資料夾**
* **主程式 : main.go (package main/import其他package)**
* **其他功能.go :分別在與檔名同名的資料夾底下,可用來import**
* **同一個Package都在同一個子資料夾內**
* **資料夾底下的go file,彼此全域變數及函式可以互見(但開頭要大寫)**

1. **$ vi 檔名.go :**

package main

import "fmt" >>> 載入封包(相當於模組) >>> fmt是最基本會用到的

func main() { >>> 呈現功能

….

}

//註解 >>> 多行的話用 /\*註解\*/

1. **產生驅動程式的依賴檔(最剛開始需要建置)**

* **$ go mod init 專案名**  => 會出現一個.mod檔,以專案名為module名
* **$ go mod tidy**  => 補足運行所需的基本package
* **$ go get pkg路徑** => 補足自己建立的package

1. **執行**
   * **$ go run檔名.go**
   * 或 **$ go build 檔名.go(會新增一個exe檔) => $ 檔名**
   * $ go test ./… : 執行所有這目錄底下的程式

**資料型別**

1. int

2. float64

3. string

4. bool

5. rune字符(EX:'a') >>> 實際用數字存取

**列印**

1. fmt.Println(x) : 換行
2. fmt.Print(x) : 不換行
3. fmt.Printf("a: %d\n", x) : 正規表示法 >>> a: {x} 換行
4. 常用正規表示法

* %d: digit
* %s: string
* %v: value
* %+v: key+value
* %#v: struct+key+value

1. log.Println(x) : 會連帶印出時間點
2. log.SetPrefix("文字") : 在時間點前面都加入”文字”
3. log.Fatalln("發生錯誤") : 發生錯誤時印這行並終止
4. fmt.Print("\n") : 容許跳脫字元
5. fmt.Print(`\n`) : 真的印\n
6. X := fmt.Sprintln(s1, s2, s3) : 把str(s1,s2,s3)組合起來
7. fmt.Scanln(&x,&y) : 相當於input,輸入時用空格隔開

**變數(宣告了就一定要用!!)**

1. var x int =??? >>> 宣告全域變數
2. 也可寫成 x := “????" >>> 常用在生命週期短的區域變數
3. **常數** : 無法更動、刪除的變數

* 宣告用法跟var一樣 => **const**
* 單字通常為全大寫
* const (

A = iota >>> iota為從0開始的遞增常數(懶人打法) // 0

B // 1

C // 2

D = iota \* 0.1 // 0.3

E // 0.4

)

1. **指標(存放記憶體位址)** : &x

* 給予指標變數 : var xPtr \*(x的dtype)=&x
* 反解指標變數 : \*xPtr >>> =x >>> 可拿來當全域變數使用

1. **struct結構(類似字典)**

* 在func main()以外的地方宣告
* type 結構名稱 struct{

欄位一名稱 dtype

欄位二名稱 dtype

}

* var x =結構名稱{欄位一:"值一",欄位二:"值二"}
* 更改值 : x.欄位一 = "新值"

1. **陣列**

* var 陣列名稱 [長度]dtype=[長度]dtype{值一,值二} >>> 沒定義值就預設空白或0 >>> 不能更改長度
* 或 陣列名稱 := […]dtype{值一,值二} >>> 長度讓GO自己判斷(懶)
* 呼叫或賦予值 : 陣列名稱[位置]
* 長度 : len(陣列名稱)

1. **Slice**

* 跟陣列差在[]裡不用打長度
* Slice名 := make([]dtype, Len現在長度, Cap最大長度)
* 或 Slice名 := []dtype{值一,值二}
* 擴充 : Slice名 = append(Slice名, 4, 5, 6)
* cap(Slice名) : 看容量
* slice... : 把slice內的東西一一取出來(比如某處型別期望是int而不是slice時就要這麼做)

1. **Map**

* var Map名 = map[key的dtype]value的dtype{

"KEY1": VALUE1,

"KEY2": VALUE2,

}

* 刪除某一個key : delete(Map名,"key")
* 迭代印出內容 :

for index, n := range Slice/Map名{

fmt.Println(index, n)

}

1. **Interface**

* 用法 : var a interface{}

>>> 相容任何型別

>>> 未定義a是甚麼的時候,a就是<nil>

* 型別斷言 : 直接宣稱interface是哪種型別

var x interface{} = "hello"

helloStr := x.(string) >>> 也就是"hello"

**邏輯運算**

* 運算子

1. ! : 反
2. && : AND
3. || : OR

* 判斷式

1. 一般寫法

If 條件{

.....

}else if{

break

}else{

continue

}

1. 很多個條件的話

switch {

case 條件一 :

.....

case 條件二 :

.....

case 條件三 :

.....

default :

.....

}

* 流程控制

1. 第一種寫法

for …{

……

}

1. 第二種寫法

for 初始值(X=0),條件式(X<50),重複值型的命令(X+=1){

……

}

1. 無窮迴圈

for true{

}

**函式**

* 命名 : 字首大寫 >>> 全域通用
* func 函式名(變數 型別) (A的dtype,B的dtype) {

return A,B

}

* 可變函式
* 參數長度可變(用...代替)
* 但...只能出現一次(一種型別)且放在最後
* func xxx(int, **...**string) {}
* panic() : 主動引發錯誤的函數
* **defer(拖延)**
* **defer 動作** : **拖延到整個函式結束才做**(常用在提醒關檔或關db)
* 有兩個defer,越早分派越晚達成
* 當下變數是多少就是多少,拖到最後也不會變

**Go併發（Goroutines）**

1. 跟**主程式同時進行**的函式,主程式結束他**就要被迫中止**
2. 用法 : **go + 函式**
3. 有多個函式一起GO的話系統會自己排程
4. **runtime.GOMAXPROCS(n)** : 限制程式執行時用到的最大CPU數量,預設1
5. 匿名函式 : 不取名字直接用的函式,常跟併發一起用(**匿名併發**)

>>> func main() {

**go func() {**

**............**

**}**() >>> 記得括號,因為要直接呼叫

}

1. **通道(chan)** : 為了讓併發函式可以順利回傳值,因此需要通道
   * C := make(chan dtype) >>> 只能傳輸特定型別的資料,但無限制資料量
   * c := make(chan dtype, 2) >>> 限制一次最多塞2筆資料,要先取出來再塞才不會Deadlock(死結)
   * 用法 :

func main() {

ch := make(chan int)

go func1(ch)

ch <- 100 >>> 丟100進入通道

}

func func1(ch chan int) {

i := <-ch >>> 從通道裡挖出什麼

fmt.Println(i)

}

1. **sync WaitGroup** : 用來等待確認全部併發線程跑完

wg := sync.WaitGroup{}

wg.Add(2) >>> 要等待的事件有兩個

wg.Done() >>> 任務完成,Counter減去1

wg.Wait() >>> 等待直到Counter歸0

1. **sync.Mutex(鎖頭)** : 確保併發函式能順利執行,變數不會被主程式搶走

var m sync.Mutex

m.Lock() >>> 鎖起來(確保接下來的步驟不會被主程式搶走)

.....

m.Unlock() >>> 解鎖

**讀寫檔案**

1. 讀取檔案路徑

import (

"encoding/csv"

"fmt"

"io"

"log"

"os"

"path/filepath"

)

var Pwd string

var FilePath string

var FileName = "檔名.csv"

func init() {

Pwd, \_ = os.Getwd() // 取得目前所在位置

FilePath = filepath.Join(Pwd, FileName)

}

1. 讀取CSV

func Load() {

file, err := os.OpenFile(FilePath, os.O\_RDONLY, 0777)

// os.O\_RDONLY 表示只讀 // 0777 表示(owner/group/other)權限

if err != nil {

log.Fatalln("找不到CSV檔案路徑:", FilePath, err)

}

r := csv.NewReader(file)

r.Comma = ','

for {

record, err := r.Read()

if err == io.EOF {

break

}

if err != nil {

log.Fatalln(err)

}

var a = record[0] // 第一欄

var b = record[1] // 第二欄

fmt.Println(a, b)

}

}

1. 寫入CSV

func Write() {

file, err := os.OpenFile(FilePath, os.O\_WRONLY, 0777)

// os.O\_WRONLY 表示只寫

// 如果檔案不是空白文件,要用**O\_APPEND**才不會覆蓋原始資料!!!

if err != nil {

log.Fatalln("找不到CSV檔案路徑:", FilePath, err)

}

w := csv.NewWriter(file)

**// 寫入單行**

x := []string{"999"}

w.Write(x)

**// 寫入多行**

x := [][]string{{"999", "1"}, {"998", "997"}}

w.WriteAll(x)

w.Flush()

}

**網頁框架 gin**

1. linux : $ cd /root/go/src/golang.org/x/net/http2

windows : $cd go/sys

1. $ go get -u github.com/gin-gonic/gin
2. $ vi 檔名.go :

package main

import (

"github.com/gin-gonic/gin"

"net/http"

)

func main() {

router := gin.Default()

router.GET("/URL文字", XXX)

router.Run(":8080")

}

func XXX (c \*gin.Context) {

c.JSON(200, gin.H{

"message": "網頁要呈現的文字"

})

或是

str := []byte("網頁要呈現的文字")

c.Data(http.StatusOK, "text/plain; charset=utf-8;", str)

或是

m := map[string]string{"status": "ok"}

j, \_ := json.Marshal(m)

c.Data(http.StatusOK, "application/json", j)

或是

type Result struct {

Status string `json:"status"`

Message string `json:"message"` >>> 轉換為JSON模式時可對照

} >>> Status字首一定要大寫!

var result = Result{

Status: "OK",

Message: "This is Json",

}

c.JSON(http.StatusOK, result)

}

1. $ go run 檔名.go
2. 瀏覽器輸入 : localhost:8080/ping
3. 詳細說明
4. GET

* 查詢參數(….?XXX=XXX)

**input := c.Query("input")** >>> 用?input=XXX來輸入參數

* 路徑參數(…./XXX/)
* **router.GET("/XXXX/:input",函式)**
* **msg := c.Param("input")**  >>> /後面要接的文字

**c.String(http.StatusOK, "網頁文字%s", msg)**

1. POST
   * **router.RedirectFixedPath = true**

**router.POST("/post",函式)**

* + **msg := c.PostForm("input")**  >>> 用POSTMAN確認時的key

**c.String(http.StatusOK, "網頁文字%s", msg)**

* + 用POSTMAN確認方法 :

URL : 127.0.0.1/post

KEY => input / VALUE => xxx

**Swaggo**

1. 安裝 : **$ go install github.com/swaggo/swag/cmd/swag**

**$ go get github.com/swaggo/gin-swagger**

**$ go get github.com/swaggo/gin-swagger/swaggerFiles**

1. 初始化 : 在有main.go的專案資料夾執行 **$ swag init** **=> 會出現一個docs資料夾**
2. main.go :

* **import “\_ docs路徑”**
* **新增文件註解**

1. 每次修改文件後都要**$ swag init** (以對docs內文件進行自動修改)
2. **$ go run main.go** => 產生Swagger網頁 **(http://127.0.0.1:8080/swagger/index.html)**
3. **註解**說明

* Summary : 出現在每一隻API後的簡單描述
* Description : 點開每一隻API後出現的詳細描述
* Tags : 分組用(大標)
* Accept : 接受Parameters格式(通常是**mpfd**)
* Produce : Response格式(通常是**json**)
* **Param : 輸入的Parameters 輸Parameters方式 Parameters的dtype 是否必填 對Parameters的簡單說明 (其他功能:可搭配使用)**
* **輸Parameters方式**分為 : **query(GET)、formData(POST)**
* **其他功能**分為 :

1. 選單 : **Enums(值1, 值2, 值3)**
2. 定義Parameters值的範圍 : **minlength(5) maxlength(10)**
3. 定義預設Parameters值 : **default(A)**

* **Success/Failure : Code Parameters型別 Response資料型別 舉例**
  + 如果response格式已有定義好的struct或array格式,可改為:
* **200 {array} response.ret(struct名稱)**
* **400 {object} response.ret(struct名稱)**

**MySQL**

* 安裝驅動Driver : $ go get -u github.com/go-sql-driver/mysql
* 檔案寫法

package main

import (

"database/sql"

\_ "github.com/go-sql-driver/mysql"

"log"

)

var db \*sql.DB

func init() { // 連線設置(單純成功連線不會有任何反應)

dbConnect, err := sql.Open(

"mysql",

"**root:0710@tcp(127.0.0.1:3306)/newschema**",

)

if err != nil {

log.Fatalln(err)

}

err = dbConnect.Ping()

if err != nil {

log.Fatalln(err)

}

db = dbConnect

db.SetMaxOpenConns(10) // 設置最大DB連線數，設<=0則無上限

db.SetMaxIdleConns(10) // 設置最大idle閒置連線數

}

func main() {

**rows, err := db.Exec("SQL語法" + ";")**

**rows, err := db.Query("SELECT `name`, `age` FROM.....")**

// 資料只有一筆時用**QueryRow**

**for rows.Next() {** // 相當於fetchall()

**var tName string**

**var tAge int**

**err = rows.Scan(&tName, &tAge)**

**if err != nil {**

**log.Fatalln(err)**

**}**

**fmt.Printf("%q %d\n", tName, tAge)**  // %q:用引號包起字串

**}**

**defer rows.Close()**

if err != nil { // 以防錯誤訊息

log.Fatalln(err)

}

fmt.Print(err)

rowCount, err := rows.RowsAffected()

rowId, err := rows.LastInsertId()

if err != nil {

log.Fatalln(err)

}

fmt.Printf("新增 %d 筆資料，id = %d \n", rowCount, rowId) >>> 以防萬一確認用

}

* 可用”**?**”作為**變數**

func XXX(age int) {

rows, err := db.Query("SELECT...WHERE `age` > ?;", age)

}

**鍵值資料庫 Redis**

1. 終端機安裝 :

$ cd go/sys => $ go get github.com/go-redis/redis => 執行 : $ redis-cli

1. GO檔寫法

package main

import (

"fmt"

"github.com/go-redis/redis"

)

func main() {

c := NewClient()

test(c)

}

func NewClient() \*redis.Client {

client := redis.NewClient(&redis.Options{

Addr: "localhost:6379",

Password: "",

DB: 0,

})

pong, err := client.Ping().Result() // 實作個PINGPONG確認有連上

fmt.Println(pong, err)

return client

}

func test(c \*redis.Client) {

err := **c.Set("key", "value", 0).Err()**

// 實作個設定鍵值 (0代表永不過期)

if err != nil {

panic(err)

}

val, err := **c.Get("key").Result()**

if err != nil {

panic(err)

}

fmt.Println("key", val)

}