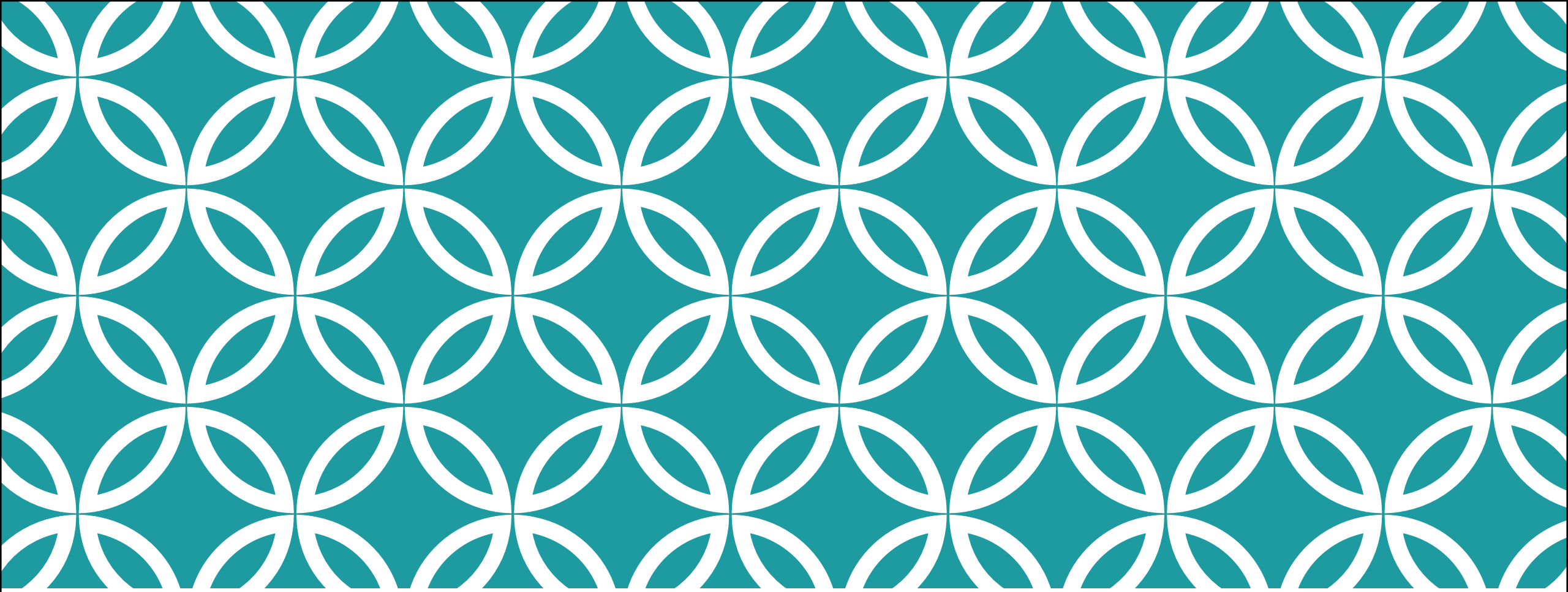
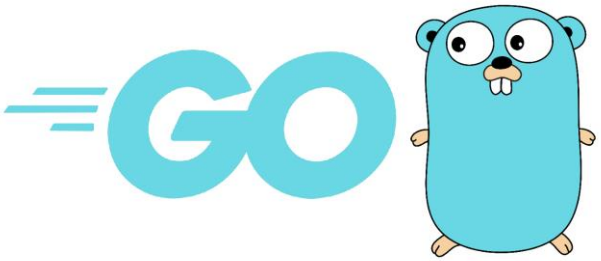


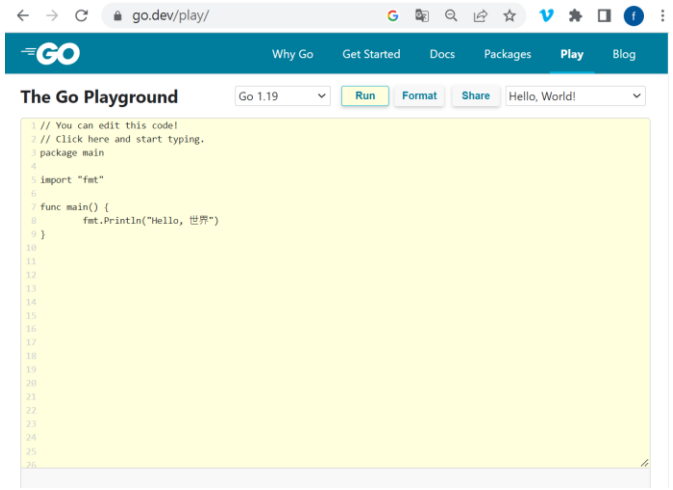
http server & http client



http server

使用golang來做點web相關的事情，如果是寫php，在一開始都需要安裝apache來跑php，但如果是使用golang的話，就可以省略這個步驟，Golang本身就包含net/http套件，只需要在import引用就可以。

但要注意的是，如果先前都是使用golang online(Go Playground)來練習，就需要先安裝golang了，**尤其還需要建置view html檔案等，這部分就是線上編輯器無法支援的**。



http server

在這邊會用兩個 package 來介紹，一個是 golang standard lib，另一個是很常用的 gorilla 這個團隊推出的 mux 套件。

net/http

- 這是一個 golang 標準內建的 http package，這要如何正確使用呢？

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    http.HandleFunc("/ping", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Hello World")
    })

    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

跑一下專案 go run 之後在瀏覽器中輸入：http://localhost:8080/ping，這裡的ListenAndServe，寫8080是使用8080port，想改什麼數字的就自行改變，但唯一要注意程式碼與網址的數字要相同，以及有些常用的port號可能會被別的軟體使用掉。

就可以看到hello world囉，所以真的不用安裝apache。

https://go.dev/play/p/B7D_PNtJJr4

http server

net/http

- 那如果要輸出 json 呢？ 其實也很簡單如下

```go
package main

import (
    "encoding/json"
    "log"
    "net/http"
)

type UserInfo struct {
    Name string `json:"name"`
    Age  int    `json:"age"`
}

func main() {

    http.HandleFunc("/api/query", func(w http.ResponseWriter, r *http.Request)
{
        u := &UserInfo{
            Name: "syhlion",
            Age:  18,
        }
        b, err := json.Marshal(u)
        if err != nil {
            log.Println(err)
            return
        }
        w.Header().Set("Content-Type", "application/json;charset=UTF-8")
        w.WriteHeader(http.StatusOK)
        w.Write(b)
    })

    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

只要存取 http://{ip}/api/query ，就能得到下面的 json 的 response：

```
{
    name:"syhlion",
    age:18
}
```

https://go.dev/play/p/IKAF5TX6bMW

http server

net/http

- json(JavaScript Object Notation)
  - json 為 JavaScript Object Notation 的簡稱，此為 JavaScript 物件的標準格式。常用於網站上的資料呈現與傳輸，以純文字去儲存和傳送**結構**資料。雖然是 JavaScript 的，但是其他語言也支援解析 json，當然也包含 Go 語言，而作為後端工程師的話，會常常遇到 json 格式。稍微看一下格式長得怎樣：

```
{
    "class" : "A" ,
    "persons" : [
        {"id" : 1 , "name" : "JC"},
        {"id" : 2 , "name" : "Eric"},
        {"id" : 3 , "name" : "Zhen"}
    ]
}
```

http server

net/http

- encoding/json
  - Go 語言中的 encoding/json 套件，**提供了函式 json.Unmarshal 可以把 json 字串轉成 struct，而 json.Marshal 可以將 struct 轉成 json 字串**，以下直接示範一個範例：

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Student struct {
    Id   int64  `json:"id"`
    Name string `json:"name"`
}

func main() {
    data := []byte(`{"id" : 1 , "name" : "JC"}`)
    var student Student
    json.Unmarshal(data, &student)
    fmt.Println(student)
    jsondata, _ := json.Marshal(student)
    fmt.Println(string(jsondata))
}
```

宣告一個名為 Student 的 struct，裡面包含兩個結構，且與之前不太一樣的是，必須告訴 struct 的 key 對應到的 json 格式的 key 為何，如此一來，就可以使用 **Unmarshal** 和 **Marshal** 來對 json 和 struct 做轉換。

https://go.dev/play/p/IAUKVw06M7V

http server

net/http

- encoding/json
  - json 可能會比較複雜，或許會包含一些 array，這時就要去宣告子母 struct，以下示範一個例子：

```go
package main

import (
    "encoding/json"
    "fmt"
)

type Student struct {
    Id   int64  `json:"id"`
    Name string `json:"name"`
}

type Class struct {
    Class    string    `json:"class"`
    Students []Student `json:"students"`
}

func main() {
    data := []byte(`{"class" : "A", "students" : [{"id" : 1 , "name" : "JC"},
        {"id" : 2 , "name" : "Zhen"}]}`)
    var class Class
    json.Unmarshal(data, &class)
    fmt.Println(class)
    jsondata, _ := json.Marshal(class)
    fmt.Println(string(jsondata))
}
```

可以以自己的需求去設計 struct，去達到解析 json 的功能！

處理 json 是後端工程師幾乎都會遇到的課題，而在 Go 語言中提供很方便的套件，可以在 json 和 struct 之間作轉換，只是在設計 struct 要多費心。

https://go.dev/play/p/-_E86_8ENvD

http server

類似的寫法(Marshal與MarshalIndent)：

```go
package main

import (
    "encoding/json"
    "fmt"
)

type response1 struct {
    Id int
    Os []string
}

func main() {

    Page_json := response1{Id: 2, Os: []string{"windows", "mac", "linux"}}

    res, _ := json.Marshal(Page_json)
    fmt.Println(string(res))

    res, _ = json.MarshalIndent(Page_json, "", "  ")
    fmt.Println(string(res))

}
```

```
{"Id":2,"Os":["windows","mac","linux"]}
{
  "Id": 2,
  "Os": [
    "windows",
    "mac",
    "linux"
  ]
}
```

https://go.dev/play/p/AMhbk1II298

http server

template：(go → html)

- 如果golang要使用作為網站的話，不太可能都只靠print來處理事情，所以其實如果需要HTML檔案的話，是也可以使用Temaple這個套件來輔助完成。**需要在import 引用"html/template"這個套件**。
- 下面是一個很簡單的範例，需要在自己的程式執行區域，建立一個view的資料夾，然後建立一個檔案index.html，裡面大概大概寫個html code，如：index.html

```
<div>hello gogo</div>
```

- 然後go的檔案如下：

```go
package main

import (
    "html/template"
    "net/http"
)

func tmpl(w http.ResponseWriter, r *http.Request) {
    t1, err := template.ParseFiles("view/index.html")
    if err != nil {
        panic(err)
    }
    t1.Execute(w, "hello world")
}

func main() {

    http.HandleFunc("/", tmpl)

    http.ListenAndServe(":8000", nil)
}
```

https://go.dev/play/p/_LB8gUbkVYI

這樣執行起來，輸入localhost:8000就有看到index.html了。

http server

template：(go → html)

- 那如果要傳變數的話呢 該如何處理？此時需要在html中變化成這樣，目前看起來是把title作為一個變數並且等待接值：

```
<div>hello, gogo</div>
<h1>{{ .Title }}</h1>
```

- Golang的程式碼：

```
package main

import (
    "html/template"
    "net/http"
)

func tmpl(w http.ResponseWriter, r *http.Request) {
    t1, err := template.ParseFiles("view/index.html")
    if err != nil {
        panic(err)
    }
    t1.Execute(w, struct {
        Title string
    }{
        "My Title is gogo",
    })
}

func main() {

    http.HandleFunc("/", tmpl)

    http.ListenAndServe(":8000", nil)
}
```

這樣初步就可以run起來一個簡單的golang搭配html template的方式，但其實這樣的模式真的只會在練習或者自用的情境下才可能真的如此應用，所以還是得找一下golang的**MVC框架**來建置。

https://go.dev/play/p/g_SaEF-2c1K

http server

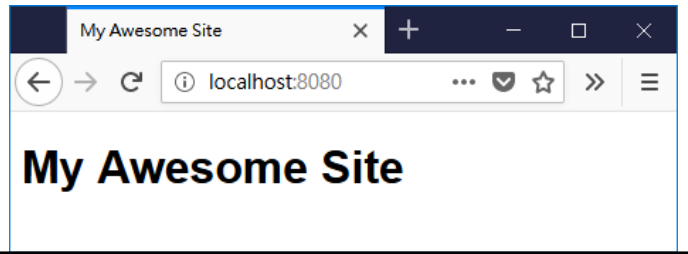
template：(go → html)

- 變數 (Variable)
  - 目標是將變數 {{ .Title }} 的地方代換掉。來看一下模板相關的程式碼：

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ .Title }}</title>
</head>
<body>
    <h1>{{ .Title }}</h1>
</body>
</html>
```

```
func index(w http.ResponseWriter, r *http.Request, p httprouter.Params)
    var tmpl = template.Must(template.ParseFiles("views/index.html"))

    tmpl.Execute(w, struct {
        Title string
    }{
        "My Awesome Site",
    })
}
```

  - 在此處用匿名結構 (anonymous struct) 帶入變數，這在 Go 語言中是一個常見的手法。以本例來說，{{ .Title }} 最後會代換成 "My Awesome Site"。
  - 程式執行結果如下：



http server

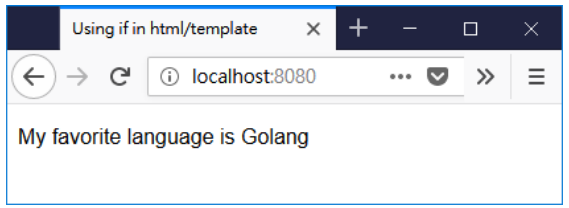
template：(go → html)

- if 敘述
  - 要注意 Go 模板語言的 if 無法做出複雜的判斷式，僅能判斷變數的真偽。這是因為模板語言的目標只是要帶入資料，而非用來建立複雜的程式邏輯。
  - 接著來看程式碼的部分：

```
<!DOCTYPE html>
<html>
<head>
    <title>Using if in html/template</title>
</head>
<body>
    {{ if .Lang }}
        <p>My favorite language is {{.Lang}}</p>
    {{ end }}
</body>
</html>
```

```
func index(w http.ResponseWriter, r *http.Request, p httprouter.Params)
    var tmpl = template.Must(template.ParseFiles("views/index.html"))

    tmpl.Execute(w, struct {
        Lang string
    }{
        "Golang",
    })
}
```

  - 程式執行結果如下：



http server

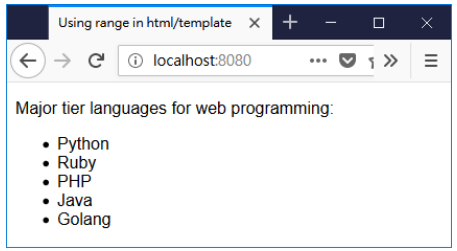
template：(go → html)

- range 敘述
  - range 是用來走訪容器 (collection) 的語法。容器的元素會將 {{ . }} 的部分代換掉。用 range 做動態的網頁清單是常見的手法。
  - 接著來看程式碼的部分：

```
<!DOCTYPE html>
<html>
<head>
    <title>Using range in html/template</title>
</head>
<body>
    <p>Major tier languages for web programming:</p>
    <ul>
        {{ range .Langs }}
        <li>{{ . }}</li>
        {{ end }}
    </ul>
</body>
</html>
```

```
func index(w http.ResponseWriter, r *http.Request, p httprouter.Params)
    var tmpl = template.Must(template.ParseFiles("views/index.html"))

    tmpl.Execute(w, struct {
        Langs []string
    }{
        []string{"Python", "Ruby", "PHP", "Java", "Golang"},
    })
}
```

  - 在變數 Langs 所指向的資料型態為字串陣列，所以可在模板進行迭代，程式執行結果如下：



http server

template：(go → html)

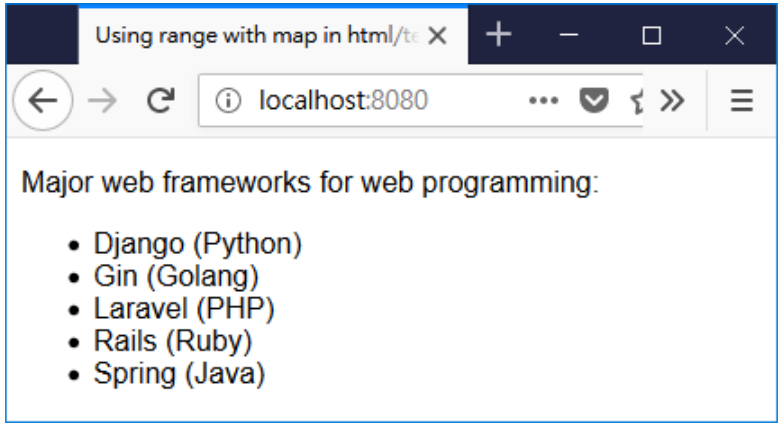
- 搭配映射 (Map) 的 range 敘述
  - 前面是用陣列搭配 range，在本例中改用映射來搭配。

```
<!DOCTYPE html>
<html>
<head>
    <title>Using range with map in html/template</title>
</head>
<body>
    <p>Major web frameworks for web programming:</p>
    <ul>
        {{range $key, $value := .Frameworks}}
        <li>{{ $key }} ({{ $value }})</li>
        {{end}}
    </ul>
</body>
</html>
```

```
{{ range $index, $element := .Array }}

  {{/* Use the variables here */}}

{{ end }}
```

  - 在這個模板中，帶入新的變數 $key 和 $value，在 range 區塊中就可以使用。如果是陣列，帶入新變數的方式如右上：
  - 接著來看程式碼的部分：

```
func index(w http.ResponseWriter, r *http.Request, p httprouter.Params)
    var tmpl = template.Must(template.ParseFiles("views/index.html"))

    tmpl.Execute(w, struct {
        Frameworks map[string]string
    }{
        map[string]string{
            "Django":  "Python",
            "Rails":   "Ruby",
            "Laravel": "PHP",
            "Spring":  "Java",
            "Gin":     "Golang",
        },
    })
}
```



  - 在此處，變數 Frameworks 指向的資料型態為映射，該映射的鍵和值皆為字串。帶入的映射可在模板中迭代。
  - 程式執行結果如右上：

http server

表單：(html → go)

- Golang表單的部分，需要使用兩個檔案的方案來demo，首先在資料夾中隨意建立一個.html結尾的檔案：

```html
<!DOCTYPE html>

<html>
    <head>
        <meta charset="utf-8"/>
        <title>表單</title>
    </head>
    <body>
        <form action="http://localhost:8000/" method="get">
          <p>First name: <input type="text" name="first name" /></p>
          <p>Last name: <input type="text" name="last name" /></p>
          <input type="submit" value="Submit" />
        </form>
    </body>
</html>
```

- 要注意的是action 必須跟golang檔案所開的url是相呼應的，所以如果跑起來沒反應，注意一下port是不是相同

https://go.dev/play/p/GM3Prtdb8oJ

http server

```go
package main

import (
    "fmt"
    "log"
    "net/http"
)

 func GetForm(w http.ResponseWriter, r *http.Request) {

    r.ParseForm()

    for k, v := range r.Form {
        fmt.Println("key is: ", k)
        fmt.Println("val is: ", v)
        fmt.Fprintf(w, "key is: %s \n", k)
        fmt.Fprintf(w, "val is: %s \n", v)
    }
    fmt.Fprintf(w, "hello world")
}

func main() {

    http.HandleFunc("/", GetForm)

    err := http.ListenAndServe("localhost:8000", nil)

    if err != nil {

        log.Fatal("ListenAndServe: ", err)

    }

}
```

https://go.dev/play/p/iPCvcxRWac3

這樣就簡單地跑了一個簡單的**接收HTML表單**的程式!!

要特別注意的部分是：

```go
fmt.Println("key is: ", k)

fmt.Fprintf(w, "key is: %s \n", k)
```

一個是在終端機上輸出，一個是在瀏覽器上輸出。

如果要在golang使用json的話，需要引用encodeing/json這個套件

```go
package main

import (
    "fmt"
    "encoding/json"
)


type response1 struct {
    Id   int
    Os []string
}

func main() {

   Page_json   := response1 {Id   : 2, Os: []string{"windows", "mac", "linux"}}
   byteSlice, _ := json.MarshalIndent(Page_json   , "", "   ")
   fmt.Println(string(byteSlice))

}
```

https://go.dev/play/p/qcUYyJn4Z06

http server

net/http

- 那如果要抓路由怎麼做呢?

```go
package main

import (
    "fmt"
    "net/http"
)

func indexHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, world!\n")
}

func echoHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, r.URL.Path)
}

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc("/hello", indexHandler)
    mux.HandleFunc("/", echoHandler)

    http.ListenAndServe(":8000", mux)
}
```

https://go.dev/play/p/aWgVDXeD8RK

https://pkg.go.dev/net/http

http server

github.com/gorilla/mux

- 再來介紹的是第三方套件 mux，mux提供的 api 可以跟原生的參數完全相容，沒有太多多餘的包裝，但又有很多方便寫 router 的方法：`go get github.com/gorilla/mux`

```go
package main

import (
    "encoding/json"
    "log"
    "net/http"

    "github.com/gorilla/mux"
)

type UserInfo struct {
    Name string `json:"name"`
    Age  int    `json:"age"`
}

func main() {
    r := mux.NewRouter()
    r.HandleFunc("/api/query/{name}", func(w http.ResponseWriter, r *http.Request) {
        vars := mux.Vars(r)

        //這邊就把 name 帶入 UserInfo
        u := &UserInfo{
            Name: vars["name"],
            Age:  18,
        }
        b, err := json.Marshal(u)
        if err != nil {
            log.Println(err)
            return
        }
        w.Header().Set("Content-Type", "application/json;charset=UTF-8")
        w.WriteHeader(http.StatusOK)
        w.Write(b)

    })

    //聽8000 port
    log.Fatal(http.ListenAndServe(":8000", r))
}
```

可以看到 mux.Vars 自動 parse router 裡面設定的 {name}，

它有支援簡易的正則判斷，詳細可以參閱官網

https://go.dev/play/p/k4mrZ72hrDW

http client

HttpClient 類別執行個體，是做為工作階段使用以傳送 HTTP 要求，實現 HTTP 和 HTTPS 協議客戶端的類別，而 HttpClient 提供了八種方法，而比較常用的為 GET/POST/DELETE/PUT，以 GET 來示範取得網頁內容：

```go
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
)

func main() {
    resp, err := http.Get("https://www.taiwanlottery.com.tw/index_new.aspx")
    if err != nil {
        panic(err)
    }

    defer resp.Body.Close()

    fmt.Println("Response status:", resp.Status)

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        panic(err)
    }

    fmt.Println(string(body))
}
```

用ioutil.ReadAll 去讀 res.Body ，這樣就能輕鬆取回 http response，這麼一來，就可以取得網頁的內容。

這邊要注意一下，res.Body 讀取完，要記得 Close，不然會有 memory leak 等相關問題

https://go.dev/play/p/ebGp0K1415M

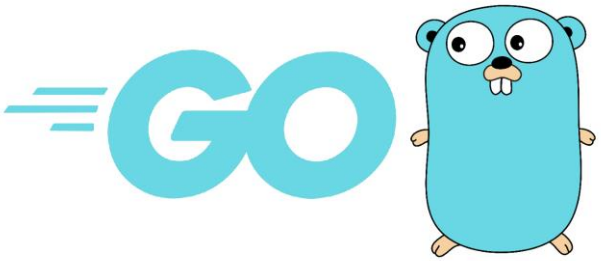
http client

另一種形式變形的用法：

```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
)

func main() {
    client := &http.Client{}

    //這邊可以任意變換 http method  GET、POST、PUT、DELETE
    req, err := http.NewRequest("GET", "https://www.taiwanlottery.com.tw/index_new.aspx", nil)
    if err != nil {
        log.Println(err)
        return
    }
    req.Header.Add("If-None-Match", `W/"wyzzy"`)
    resp, err := client.Do(req)
    if err != nil {
        log.Println(err)
        return
    }
    defer resp.Body.Close()
    sitemap, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatal(err)
        return
    }

    fmt.Printf("%s", sitemap)
}
```

https://go.dev/play/p/OlJbA16Vq-b

Go with MySQL



Go with MySQL

資料庫(Database)分成非常多種：

- 星狀資料庫
- **關聯式資料庫**
- 階層式資料庫
- 物件導向資料庫
- ...

最常用到的是關聯式資料庫(Relational Database)簡稱RDB以及非關聯式資料庫(NoSQL)(**NOSQL不是沒有SQL，也不是對SQL說NO，而指的是Not Only SQL**)

DB、SQL、RDBMS 的關係

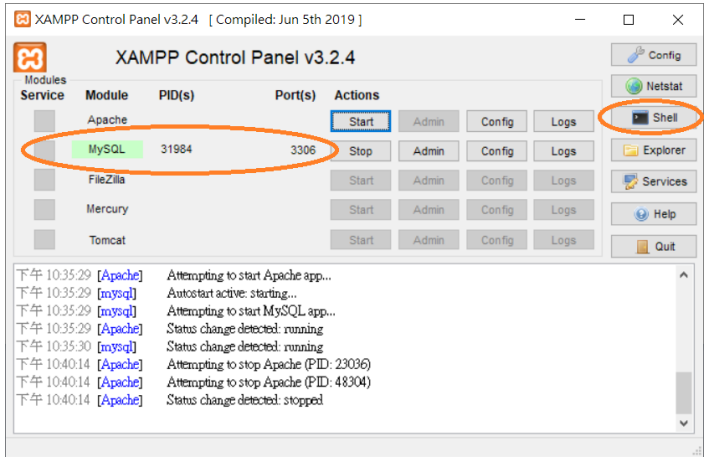
- DB 是資料庫，是一群數據的集合體，數據所在的位置，不管存了什麼都叫資料庫。(DBA 是資料庫管理員，負責資料庫的維護、備份、安全管理的人。)
- SQL 是結構化查詢語言，是一種程式語言，語法關鍵字有SELECT、INSERT、WHERE、DROP 等等，用來操作關聯式資料庫。
- RDBMS 是關聯式資料庫管理系統，是實作出來可以支援SQL語法的系統、服務，如MySQL、SQLite、MariaDB、Oracle Database、Access 等等。
- NoSQL 是相對於RDBMS的 非關聯式資料庫管理系統，也有非常眾多的系統，如Redis、memcached、MongoDB 等等，其中有支援鍵值(Key–value)的、也有支援JSON格式的。
- 關聯式資料庫基本上就是Microsoft Office的Excel，都是二維的表格，都有行列欄位。只不過用法更加的靈活，可以設置唯一識別 主鍵(Primary Key)，不可重複 Unique 等等條件。

Go with MySQL

MySQL

- 做為一個後端工程師，在架設網站時，免不了碰到資料上的問題，可能在顧客交易後，需要存取訂單，以保留資訊，以利往後查詢，這時就必須要有一個資料庫，提供存取資料的功能。
- 安裝MySQL
  - MySQL是一個系統、一項服務。要跑起服務前，需要在電腦上下載與安裝。
  - 以下會統一把最高權限帳戶root的密碼改成root。
  - Windows(XAMPP)
    - 除了可以下載 MySQL官網來安裝，也可以下載簡單好用的 XAMPP 來啟動MySQL服務：
    - XAMPP的MySQL預設密碼為空，點擊Shell，**將root帳號的密碼為root**

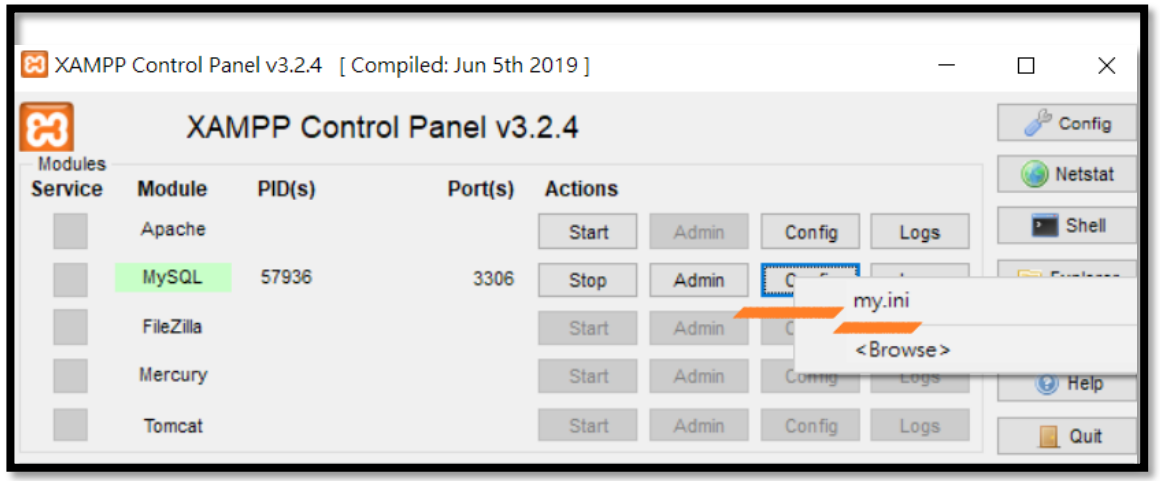
```
$ mysqladmin.exe -u root password root
```

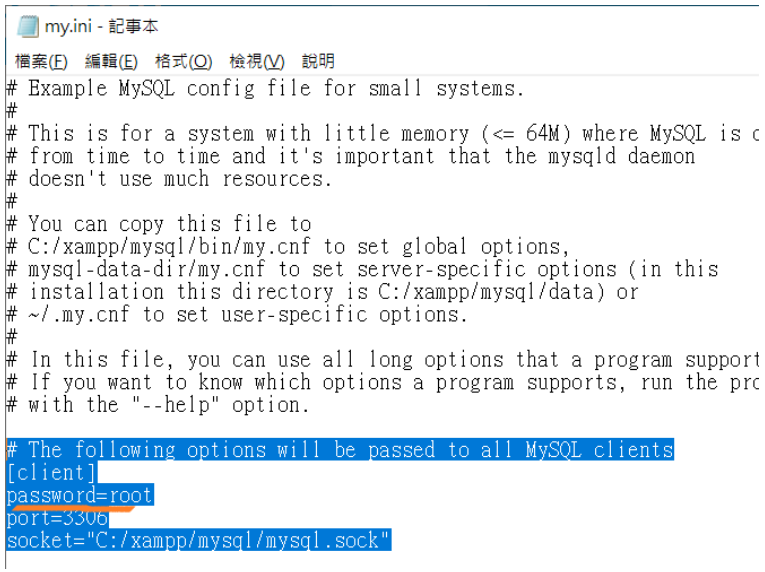


Go with MySQL

MySQL

- 安裝MySQL
  - Windows(XAMPP)
    - 這裡也要做更改：

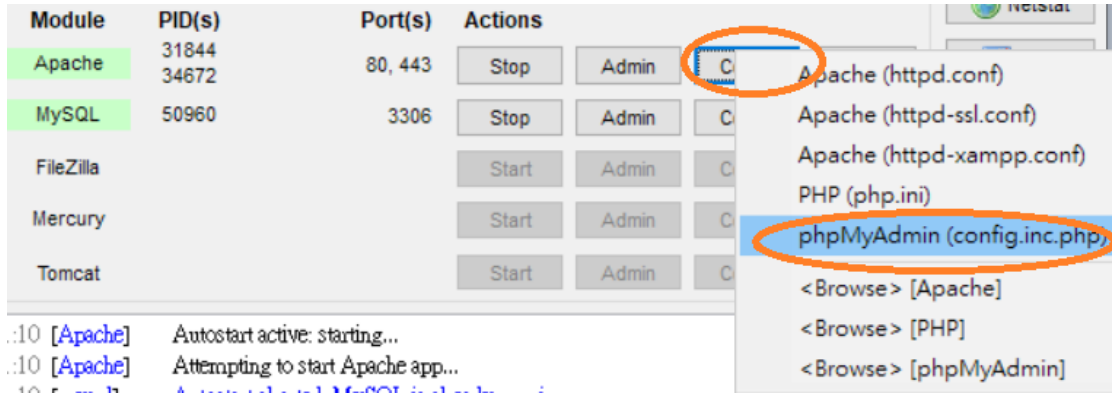




Go with MySQL

MySQL

- 安裝MySQL
  - Windows(XAMPP)
    - 如果日後會用到phpmyadmin介面的話，這邊也要改做修改：



    - **接著重啟MySQL服務。**

Go with MySQL

MySQL

- 安裝MySQL
  - MacOS
    - 用Homebrew來安裝執行：
```
$ brew install mysql@8.0
$ brew services start mysql@8.0
```
    - 更改root密碼：
```
$ /usr/local/bin/mysqladmin -u root password root
```
    - 登入MySQL：
```
$ mysql -u root -p
```
    - 結束MySQL：
```
$ brew services stop mysql@8.0
```

Go with MySQL

MySQL

- MySQL服務預設會開在Server端的3306 Port。而Server端儲存的root密碼是root，Client端連進去的密碼就要是root，否則無法登入。安裝完成也確認過可以用帳號密碼root/root登入後，便可以開始來寫程式。
- 在這邊Server端跟Client端指的都是自己目前的電腦。接下來會以程式的方式來模擬Client端，Client端能對Server端進行登入、操作資料等等。
- 在開始之前，需要架設好自己的 MySQL 環境。在架設好後，需要先建立 Database 和 Table：

```
CREATE TABLE `student` (
`id` INT(10) NOT NULL AUTO_INCREMENT,
`name` VARCHAR(64) NULL DEFAULT NULL,
`gender` VARCHAR(64) NULL DEFAULT NULL,
PRIMARY KEY (`id`)
);
```

- **但在用Golang 模擬 Client端的行為之前，要先安裝驅動Driver，否則不能支援：**
  - Go-MySQL-Driver：https://github.com/go-sql-driver/mysql

```
$ go get -u github.com/go-sql-driver/mysql
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- golang 其實在 database 的存取上，設計了一個 sql 抽象介面叫做 database/sql，接下來只要有不同人的遵照這個 interface 分開去實作 mysql、sqlite ... 等等，這是一個很棒的設計，如果未來有測試或是抽換需求，其實只要更新 driver ，但完全不需要更動程式相關地方
- 在這邊介紹 golang mysql driver ，最知名的應該是這套 go-sql-driver/mysql，先從連線做起：

```
import "database/sql"
import _ "github.com/go-sql-driver/mysql"


//完整的資料格式連線如下
//[username[:password]@][protocol[(address)]]/dbname[?param1=value1&...&paramN=valueN]
db, err := sql.Open("mysql", "user:password@/dbname")
```

```
db, err := sql.Open("mysql", "root:my-secret-pw@tcp(127.0.0.1:3306)/user")
if err != nil {
panic(err)
}
// 釋放連線
defer db.Close()
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 連線MySQL的第一支程式。以下運行成功的話不會出現任何東西。

```go
package main

import (
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
    "log"
)

func main() {
    dbConnect, err := sql.Open(
        "mysql",    // 因為只有這裡才用到這個`引數`，並沒有直接使用到了mysql.XXX 相關的函式或物件，會被認為沒有用到mysql這個依賴而 被go編譯器省略import
        "root:root@tcp(127.0.0.1:3306)/",
    )

    if err != nil {
        log.Fatalln(err)
    }

    err = dbConnect.Ping() //Ping() 這裡才開始建立連線。上面 sql.Open 只建立物件、容器，並未進行連線，無法連線並不造成err。
    if err != nil {
        log.Fatalln(err)
    }
}
```

- 沒開啟服務、沒連接到MySQL的話會出現：dial tcp 127.0.0.1:3306: connected: No connection could be made because the target machine actively refused it.
- 密碼錯誤會出現：Error 1045: Access denied for user 'root'@'localhost' (using password: YES)
- 沒有import Driver：sql: unknown driver "mysql" (forgotten import?)

https://go.dev/play/p/LdU6wmIfiCo

Go with MySQL

go-sql-driver/mysql

- 在成功連上SQL資料庫後，可以在一開始先寫好init() 初始化資料庫的連線：

```go
var db *sql.DB

// 與DB連線。 init() 初始化, 時間點比 main() 更早。
func init() {
    dbConnect, err := sql.Open(
        "mysql",
        "root:root@tcp(127.0.0.1:3306)/",
    )

    if err != nil {
        log.Fatalln(err)
    }

    err = dbConnect.Ping()
    if err != nil {
        log.Fatalln(err)
    }

    db = dbConnect // 用全域變數接

    db.SetMaxOpenConns(10) // 可設置最大DB連線數, 設<=0則無上限（連線分成 in-Use正在執行任務 及 idle執行完成後的閒置 兩種）
    db.SetMaxIdleConns(10) // 設置最大idle閒置連線數。
    // 更多用法可以 進 sql.DBStats{}、sql.DB{} 裡面看
}
```

```go
db, err := sql.Open(
    "mysql",
    "root:root@tcp(127.0.0.1:3306)/dbName",
)
```

```go
db.Exec("USE `school`")
```

- Golang中的*sql.DB這一物件抽了一層介面出來，日後不想使用MySQL時，只需更換 driver 即可。
- 若在使用上出現 **Error 1046 no database selected** 的錯誤，表示沒有指定資料庫，可改成以上指定該資料庫來連線，或者用SQL語法的USE來指定資料庫。

https://go.dev/play/p/LdU6wmIfiCo

Go with MySQL

go-sql-driver/mysql

- 下SQL語法關鍵字時，通常會使用全大寫(雖然小寫也可以，但大寫較容易區分)
- 建立資料庫：

```
db.Exec("CREATE DATABASE `test1`")
```

  - 加了IF NOT EXISTS之後，會先檢查要建立的物件存不存在，如果不存在再執行建立。
  - 否則如果無法建立物件時(未成功執行)，err 會回傳 Error 1007: Can't create database '..'; database exists。

```
_, err := db.Exec("CREATE DATABASE IF NOT EXISTS `test1`")
```

- 刪除資料庫：
  - 同樣能先判斷IF EXISTS看欲刪除的物件存不存在。

```
db.Exec("DROP DATABASE IF EXISTS `test1`")
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 此外，SQL語法可以用變數代入、拉出來寫成一個func：

```
func main() {
    createDb("`school`") // 比較好的習慣會使用 `` 反引號（重音號）來區隔名字
    // deleteDb("`school`")
}

func createDb(dbName string) {
    _, err := db.Exec("CREATE DATABASE IF NOT EXISTS " + dbName + ";")
    if err != nil {
        log.Fatalln(err)
    }
    fmt.Print(err)
}

func deleteDb(dbName string) {
    _, err := db.Exec("DROP DATABASE IF EXISTS " + dbName + ";")
    if err != nil {
        log.Fatalln(err)
    }
}
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 建立、刪除表格：
  - 資料表中每個欄位都必須要有類型、並設置Byte長度，才能存放該型別的資料。常見的變數類型有：
    - INT(每長度 4 bytes)：數字，存放整數資料
    - CHAR(每長度 1 byte)：固定長度字串
    - VARCHAR：可變長度字串，用多少長度就多少
    - TEXT：文字敘述、說明(長度較長)
    - DATE：時間格式
    - TIMESTAMP：時間戳記

Go with MySQL

go-sql-driver/mysql

- **建立**、刪除、更新表格：**Create** Table
  - 建立student的表格

```go
func createTable1() {
    // SQL: CREATE TABLE IF NOT EXISTS `school.student`
    // 初始化Table 沒給任何欄位時, 會出現 `A table must have at least 1 column` 的錯誤

    _, err := db.Exec("CREATE TABLE IF NOT EXISTS `school`.`student`(`name` VARCHAR (10))")
    if err != nil {
        log.Fatalln(err)
    }

    // 或者用 USE 來指定該資料庫
    // db.Exec("USE `school`")
    // _, err := db.Exec("CREATE TABLE IF NOT EXISTS `student`(`name` VARCHAR (10))")
}
```

  - 再建立一個teacher的表格，這次存放name(VARCHAR)、age(INT)兩個欄位，並且設置name為PRIMARY KEY(唯一不重複的識別ID)。

```go
_, err := db.Exec("CREATE TABLE IF NOT EXISTS `school`.`teacher`(`name` VARCHAR (10) PRIMARY KEY, `age` INT (1))")
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 建立、**刪除**、更新表格：**Detele** Table

```
_, err := db.Exec("DROP TABLE IF EXISTS `school`.`student`")
```

- 建立、刪除、**更新**表格：**Alter** Table
  - 替student的Table加上 id 這個會自動遞增(AUTO_INCREMENT)的欄位，並設定成主鍵(數字遞增的特性：唯一、不重複，且Not Null)：

```
_, err := db.Exec("ALTER TABLE `school`.`student` ADD `id` INT AUTO_INCREMENT PRIMARY KEY;")
```

- 反引號 vs 雙引號
  - 資料表、欄位會用反引號``包起來，以防與SQL語法關鍵字搞混。
  - 而雙引號 " " 常用在字串的組合、組成SQL語法傳遞給資料庫。

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作
  - CRUD是四種對 row 操作的頭字詞單字縮寫：**C**reate、**R**ead、**U**pdate、**D**elete
  - 插入(Insert)：

```
_, err := db.Exec("INSERT INTO `school`.`teacher`(`name`, `age`) VALUES ('Tiger' , 28)")
```

    - 也可以使用佔位符(Placeholder)來當作臨時變數。不過每個RDBMS的佔位符不一定相同，在MySQL是以?當作佔位符號。

```
_, err := db.Exec("INSERT INTO `school`.`teacher`(`name` , `age`) VALUES (? , ?)", teacherName, teacherAge)
```

```
func insertTeacher(teacherName string, teacherAge int) {
    _, err := db.Exec("INSERT INTO `school`.`teacher`(`name` , `age`) VALUES (? , ?)", teacherName, teacherAge)
    ...
}
func insertStudent(studentName string) {
    rs, err := db.Exec("INSERT INTO `school`.`student`(`name`) VALUES (?)", studentName)
    if err != nil {
        log.Println(err)
    }

    rowCount, err := rs.RowsAffected()
    rowId, err := rs.LastInsertId() // 資料表中有Auto_Increment欄位才起作用，回傳剛剛新增的那筆資料ID

    if err != nil {
        log.Fatalln(err)
    }
    fmt.Printf("新增 %d 筆資料, id = %d \n", rowCount, rowId)
}
```

https://go.dev/play/p/LdU6wmIfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作
  - CRUD是四種對 row 操作的頭字詞單字縮寫：**C**reate、**R**ead、**U**pdate、**D**elete
  - 選擇、查詢(Select)
    - 用Select選擇name跟age欄位來作印出：

```
rows, err := db.Query("SELECT `name`, `age` FROM `school`.`teacher`")
// rows, err := db.Query("SELECT * FROM `school`.`teacher`") // 也可以使用`Select *`選取全部欄位。
```

```
for rows.Next() { // rows.Next() 前往下一個項目。如果成功（還有下一項的話）返回True、失敗（沒有下一項可讀）則返回False
    var tName string //兩個欄位，依SELECT的順序用兩個變數來接
    var tAge int
    err = rows.Scan(&tName, &tAge) // 掃描後存進變數中
    if err != nil {
        log.Fatalln(err)
    }
    fmt.Printf("%q %d\n", tName, tAge) // %q:quoted 用引號包起字串
}
defer rows.Close() // 當完整迭代rows.Next()完後會自動關閉rows，但以防萬一 最後記得要關閉rows 。
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作
  - CRUD是四種對 row 操作的頭字詞單字縮寫：**C**reate、**R**ead、**U**pdate、**D**elete
  - 選擇、查詢(Select)
    - 使用條件 Where 語法來查詢年紀超過age的老師：

```go
func queryTeacherByAge(age int) {
    rows, err := db.Query("SELECT `name`, `age` FROM `school`.`teacher` WHERE `age` > ?;", age)
}
```

    - db.QueryRow (單筆) ：用在確定只有一筆資料row回傳的時候。
    - db.Query (多筆)：回傳符合結果的多筆資料rows，

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作
  - CRUD是四種對 row 操作的頭字詞單字縮寫：**C**reate、**R**ead、**U**pdate、**D**elete
  - 更新(Update)
    - 更新名字為name的老師的年齡為age：

```
func updateTeacherAge(teacherName string, age int) {
    _, err := db.Exec("UPDATE `school`.`teacher` SET `age`= ? WHERE `name` = ?;", age, teacherName)
}
```

  - 刪除(Delete)
    - 刪除名字為name的老師的資料列

```
func deleteTeacher(teacherName string) {
    _, err := db.Exec("DELETE FROM `school`.`teacher` WHERE `name` = ?;", teacherName)
}
```

    - 在MySQL中，INSERT 可以沒有 INTO，但是 DELETE 不能沒有 FROM

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作(**Prepare**)
  - 執行寫入

```go
stmt, err := db.Prepare("INSERT `student` SET `name`=?,`gender`=?")
if err != nil {
    panic(err)
}

res, err := stmt.Exec("JC", "F")
if err != nil {
    panic(err)
}

id, err := res.LastInsertId()
if err != nil {
    panic(err)
}
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作(**Prepare**)
  - 更新資料

```
stmt, err = db.Prepare("update `student` set gender=? where `id`=?")
if err != nil {
     panic(err)
}
res, err = stmt.Exec("M", 1)
checkErr(err)
update, err := res.RowsAffected()
if err != nil {
    panic(err)
}
fmt.Println(update)
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- CRUD基本操作(**Prepare**)
  - 刪除資料

```
stmt, err = db.Prepare("delete from `student` where id=?")
if err != nil {
    panic(err)
}
res, err = stmt.Exec(1)
checkErr(err)
del, err = res.RowsAffected()
if err != nil {
    panic(err)
}
fmt.Println(del)
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 常用SQL函式：因為這些用法是SQL函式，關鍵字後必須加上小括號()
  - 筆數(Count)
    - 年紀資料不為null的老師總數：

```go
func numOfTeacher() {
    row := db.QueryRow("SELECT COUNT(`age`) FROM `school`.`teacher`;")
    var count int
    row.Scan(&count)
    fmt.Println(count)
}
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 常用SQL函式：因為這些用法是SQL函式，關鍵字後必須加上小括號()
  - 最大最小值(Max, Min)
    - 查詢年紀最大的的老師：

```
func oldestAge() {
    row := db.QueryRow("SELECT MAX(`age`) FROM `school`.`teacher`;")
    var tAge int
    row.Scan(&tAge)
    fmt.Println("老師年紀最大為:", tAge)
}
```

    - 相反，用MIN則查詢年紀最小的老師。
    - 也可以使用ORDER BY DESC, Limit來查詢年紀最大者。

```
func oldestTeacher() {
    row := db.QueryRow("SELECT `name`, `age` FROM `school`.`teacher` ORDER BY `age` DESC Limit 1 ;")
    var tName string
    var tAge int
    row.Scan(&tName, &tAge)
    fmt.Println(tName, tAge)
}
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 常用SQL函式：因為這些用法是SQL函式，關鍵字後必須加上小括號()
  - 加總(Sum)
    - 回傳老師們年紀的加總：

```
func SumOfTeachersAge() {
    row := db.QueryRow("SELECT SUM(`age`) FROM `school`.`teacher`;")
    var sum int
    row.Scan(&sum)
    fmt.Println(sum)
}
```

https://go.dev/play/p/LdU6wmlfiCo

Go with MySQL

go-sql-driver/mysql

- 如果要 transcation：

```
tx, err := db.Begin()
if err != nil {
    return
}
defer func(){

    //如果最後有錯誤，則執行 rollback
    if err != nil {
        tx.RollBack()
    }
}

...
...
...


tx.Commit()
```

https://go.dev/play/p/LdU6wmlfiCo