

Go 程式設計課 06

Go 與後端開發



大綱

- ❖ 網頁後端介紹
- ❖ 簡單網頁後端練習
- ❖ 簡易後端實作練習
- ❖ API 概念
- ❖ MVC
- ❖ 分層架構
- ❖ 短網址服務實作:
 - system design
 - implement



網頁後端介紹

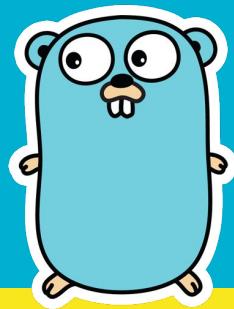
什麼是 Web Server?



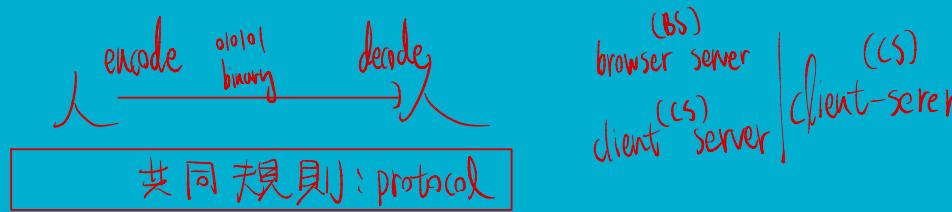
- Web Server 是一種電腦程式,負責接收和處理來自客戶端(通常是網頁瀏覽器)的請求。
- 它會根據請求返回相應的內容,通常是 HTML 頁面、圖片、影音檔案等。
- Web Server 使用 HTTP 協議與客戶端進行通訊。

經常

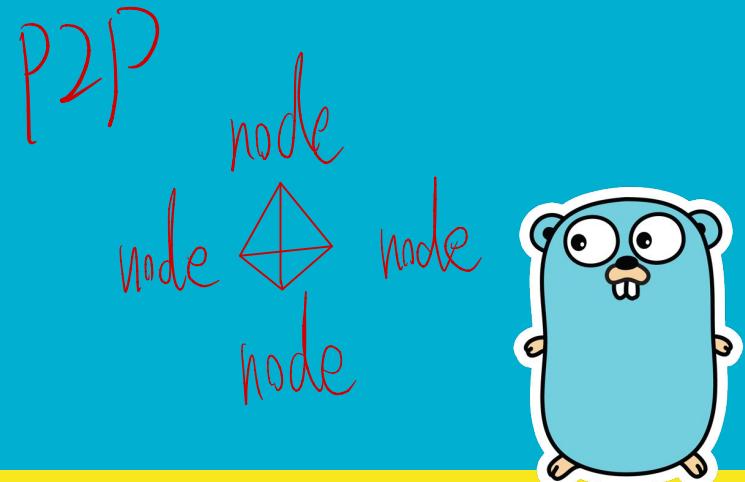
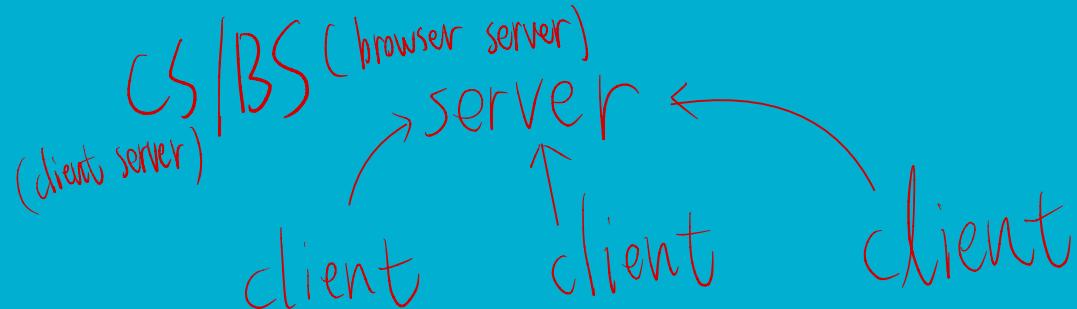
JSON
Y



HTTP 是什麼?

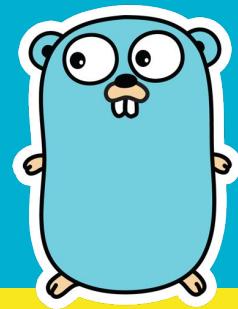


- HTTP 是 HyperText Transfer Protocol(超文本傳輸協議)的縮寫。
- 它是一種應用層協議,定義了客戶端和服務器之間的通訊規則。
- 客戶端(如瀏覽器)發送 HTTP 請求,服務器返回 HTTP 響應。
- 每個 HTTP 請求和響應都包含一個頭部(Header)和一個可選的消息體(Body)。
- 基於主從式架構的協議 ! (client - server)



HTTP 請求 (request)

- 客戶端發送 HTTP 請求到服務器, 請求通常包括:
 - 請求方法(如 GET、POST) ——> 不一定要 restful(?) 可以只用 get 和 post
 - 請求的 URL → `https://domain:port/url-path`
 - HTTP 版本
 - 請求頭部(包含客戶端的附加訊息) ——> 例如: key, 客戶端識別, 協議, ...
 - 請求體(可選, 包含客戶端提交到服務器的資料) ——> body, get request 沒有
- 常見的請求方法有:
 - GET: 獲取資源
 - POST: 提交資料
 - PUT: 更新資源
 - DELETE: 刪除資源



HTTP 響應(response)

- 服務器處理請求後,返回 HTTP 響應到客戶端,響應通常包括:

- HTTP 版本
- 狀態碼(表示請求處理的結果) (000) 2XX: 成功, (301) 3XX: 轉址, (401) 4XX: 沒有權限, (500) 5XX: internal error
- 狀態訊息
- 響應頭部(包含服務器的附加訊息)
- 響應體(包含請求的資源內容)

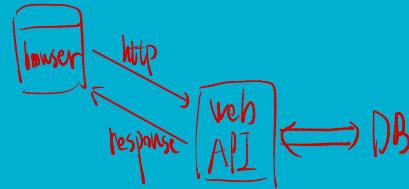


- 常見的狀態碼有:

- 200 OK: 請求成功
- 404 Not Found: 請求的資源不存在
- 500 Internal Server Error: 服務器內部錯誤



Web Server 的作用



Web Server 負責處理 HTTP 請求, 執行相應的操作, 並返回 HTTP 韻應。

主要作用包括:

- 接收和解析 HTTP 請求
- 根據請求完成必須的資料庫數據操作, 並且生成對應的回覆。
- 生成 HTTP 韵應, 包括狀態碼、頭部和響應內容
- 將 HTTP 韵應返回給客戶端

可以把每次的 HTTP 要求視為 API 調用



Lab

簡單網頁後端練習 原生

<https://github.com/leon123858/go-tutorial/tree/main/backend/cmd>



Lab

簡易後端實作練習

用 framework

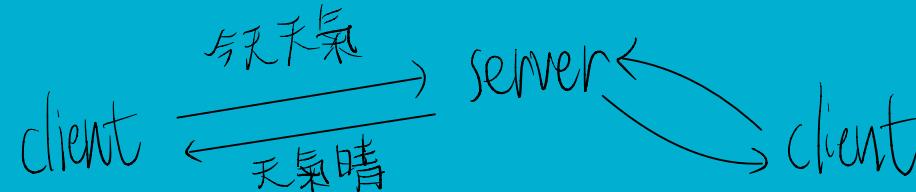
<https://github.com/leon123858/go-tutorial/tree/main/backend/cmd>



API 概念

API簡介

1. 協定
2. 機制
3. 獲取Service



API 是一個「**介面**」，意味著一個事物與另一個事物**互動**的一種方式。

API 是使用一組定義和**協定**讓兩個軟體元件彼此**通訊**的**機制**。同樣，API 是一個軟體與另一個程式**互動**以**獲得所需服務**的方式。

API 架構通常會藉由用戶端和伺服器來說明。傳送要求的應用程式稱為用戶端，傳送回應的應用程式則稱為伺服器。因此在天氣的例子當中，氣象局的天氣資料庫是伺服器，行動應用程式是用戶端。

舉例來說，氣象局的軟體系統包含有每日的天氣資料。您手機中的天氣應用程式會**透過** API 與此系統「交談」並且在您的手機顯示每日天氣的最新消息。



API呼叫的概念

API 呼叫也稱為 API 要求(API Request)，是針對 API 的訊息，用於觸發 API 使用。

API 呼叫必須按照 API 的要求進行格式化才能正常運作。API 的要求稱為其「結構描述」。該結構描述還描述了為每個要求提供的回應類型。

參考 : <https://www.cloudflare.com/zh-tw/learning/security/api/what-is-an-api/>

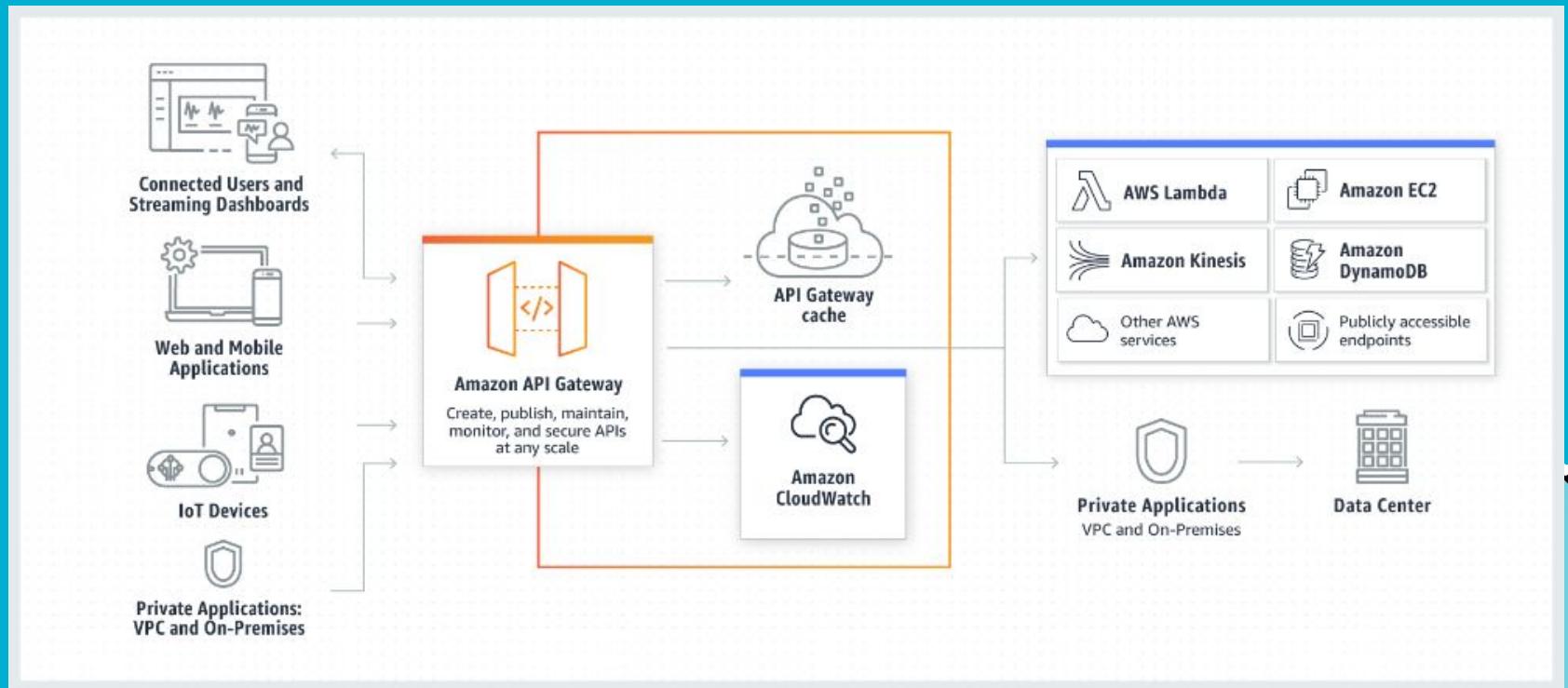


API類別

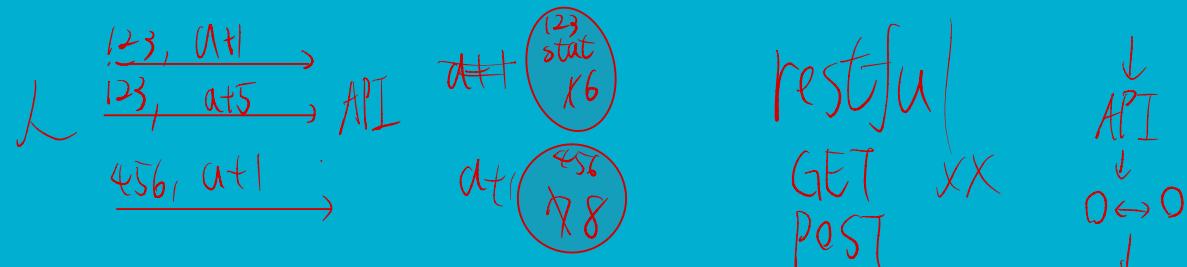
- ❖ REST API → *restful*
- ❖ JSON RPC
- ❖ SOAP API
- ❖ GraphQL
- ❖ GRPC
- ❖ WebSocket API
- ❖ webhook



What is API?



REST API



REST 代表表現層狀態轉換 (Representational State Transfer)。REST 定義了例如 GET、PUT、DELETE 的一組功能，用戶端可以利用它們來存取伺服器資料。用戶端和伺服器使用 HTTP 交換資料。
所以要 \rightarrow cookie, session, JWT ...

REST API 的主要功能為無狀態(stateless)。無狀態的意思是伺服器在要求之間並不會儲存用戶端資料。用戶端對於伺服器的要求類似於您在瀏覽器中輸入以造訪網站的 URL。來自伺服器的回應為普通資料，沒有典型的 Web 圖形呈現。

可以理解成: send [Method+Path+Json], receive [Json].

+
status

參考 : <https://aws.amazon.com/tw/what-is/api/>

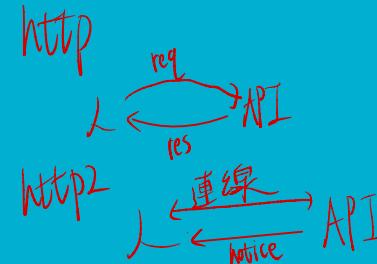


gRPC

HTTP2 vs HTTP1

多工：一次多個資源加載

雙向：server push



gRPC 是一種現代化的、高性能、開源的遠程程序呼叫(RPC)框架，由Google開發。它使用HTTP/2 協議進行通信，並支持多種語言。gRPC 旨在解決不同服務之間的通信問題，特別是在大型分佈式系統中，適用於構建高效、跨語言、跨平台的服務間通信系統。→ micro service

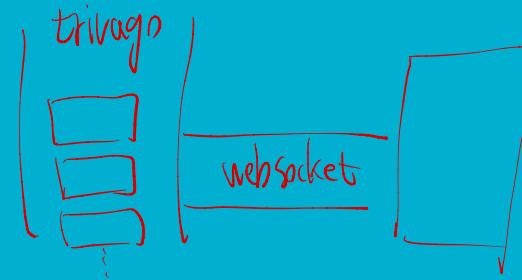
gRPC 使用 Protocol Buffers(ProtocolBuf)作為其接口定義語言(IDL)，這是一種輕量級、高效的二進制序列化工具，可用於定義服務接口和消息格式。這使得 gRPC 提供了高效的數據序列化和網絡傳輸能力。

高速的雙向二進位通訊協議。
HTTP meta data + data → sever

gRPC : data $\xrightarrow{\text{IDL}}$ sever



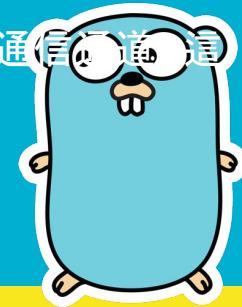
Web Socket簡介



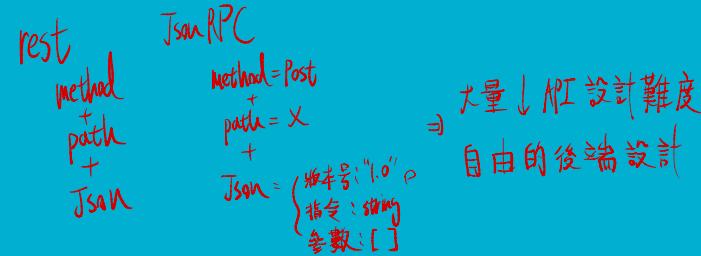
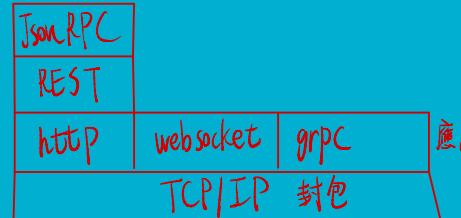
Web Socket 是一種強大的通信協議，可用於實現即時性要求高的Web 應用程序，它提供了簡單、高效、穩定的雙向通信機制，並且在許多現代Web 技術中得到了廣泛應用。

Web Socket的主要特性：

- ❖ 雙向通信：客戶端和服務器之間可以同時發送和接收數據，這使得實時互動應用程序（如聊天應用、遊戲、股票市場數據等）的實現變得更加容易。
- ❖ 持久連接：WebSocket 通道一旦建立，將保持打開狀態，直到客戶端或服務器主動關閉連接，這減少了建立連接所需的開銷，同時也降低了服務器的負載。
- ❖ 低延遲：由於WebSocket 建立在 TCP 協議之上，而不是HTTP，因此它可以實現低延遲的通信，這對於需要即時反饋的應用程序至關重要。
- ❖ 跨域支持：WebSocket 通常不受同源政策的限制，因此可以在不同的網域之間建立通信通道，這使得跨域應用程序的開發變得更加靈活。
- ❖ 較 GRPC 慢，



JSON RPC簡介



JSON-RPC (JSON Remote Procedure Call) 是一種輕量級的遠程程序調用協議，用於在分佈式系統中進行通信。它允許客戶端向服務器發送請求，並獲得相應的響應，通常使用JSON 格式進行數據的序列化和反序列化。

JSON-RPC 的主要特點：

- ❖ **簡單性**: JSON-RPC 使用 JSON (JavaScript Object Notation) 格式進行數據傳輸，這是一種輕量級、易於理解的數據格式，使得協議本身變得簡單易用
- ❖ **輕量級**: 相較於其他RPC 協議(如SOAP)，JSON-RPC 協議相對較輕量，因為它只需要傳輸數據的最小必要信息，這減少了通信的開銷，並提高了效率
- ❖ **跨平台和語言**: 由於JSON 格式的通用性，JSON-RPC 可以在不同平台和編程語言之間進行通信，這使得它適用於各種分佈式系統的開發
- ❖ **獨立於傳輸協議**: JSON-RPC 協議本身與傳輸協議無關，因此可以在多種傳輸協議(如HTTP, TCP, WebSocket 等)上運行，這增加了其靈活性和可擴展性



GraphQL 簡介

- 讓客戶端決定所需的資料
- 通過單一端點獲取所有資料
- 提供清晰易懂的資料模型

Fb 推出

1. API 數量↓
2. 前端更好寫
3. 減少冗餘的 API request

Rest: 把困難留給 frontend
GraphQL: 把困難送給 backend

type User {
 Orders: [Order]
 ID
 name
 email
}
Order
有 order?
型
數
量
有
關
注
事
件



GraphQL

Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```

Ask for what you want

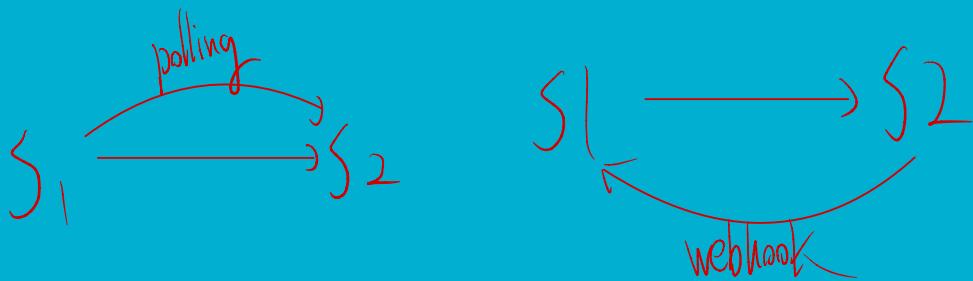
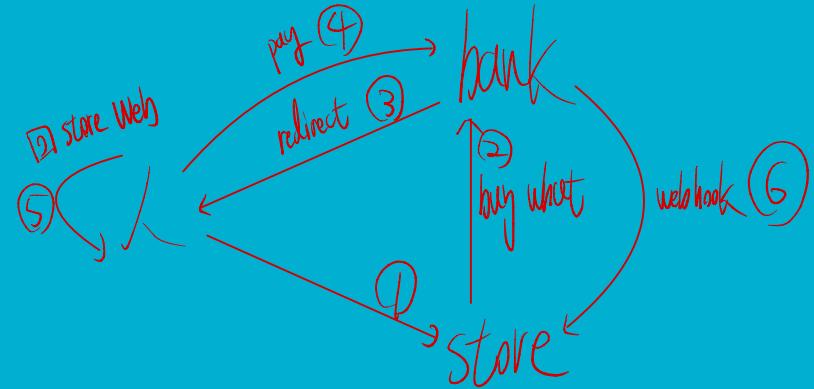
```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```

Webhook 簡介

- 讓服務之間能夠實時傳遞資訊
- 通過註冊回調 URL 接收即時訊息
- 實現系統間的低耦合通訊
- 用 hook 取代 polling



Open API

不要求 follow RESTful API

定義：OpenAPI (以前稱為 Swagger Specification) 是一種描述 RESTful API 的標準,用於定義 API 的介面和資料格式。

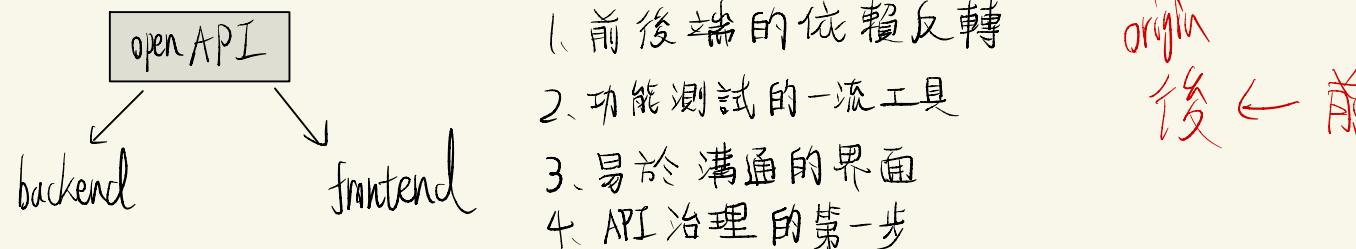
目的：OpenAPI 旨在提供一種通用的、可機器讀取的 API 定義格式,使得 API 的使用和理解更加容易。
優點：

- 提高 API 的可發現性和互通性
- 簡化 API 文件的生成和維護
- 支持自動化測試和程式碼生成
- 促進團隊和合作夥伴之間的協作 front ↔ back

格式：OpenAPI 定義通常採用 YAML 或 JSON 格式,描述 API 的各種資訊,如 HTTP 方法、路徑、參數、請求/響應資料模型等。

工具：OpenAPI 支援多種工具,如 Swagger UI、Postman 等,用於 API 文件的生成和測試。

版本：目前最新版本為 OpenAPI 3.0,相較於 2.0 版本提供了更豐富的功能和更好的可擴展性。



Swagger 練習

1. swagger editor
2. SPEC
 1. 縮短網址
 2. 短網址轉址
 3. 查看特定用戶的短網址轉址記錄

your turn!

<https://github.com/leon123858/go-tutorial/blob/main/short-url/docs/swagger.yaml>

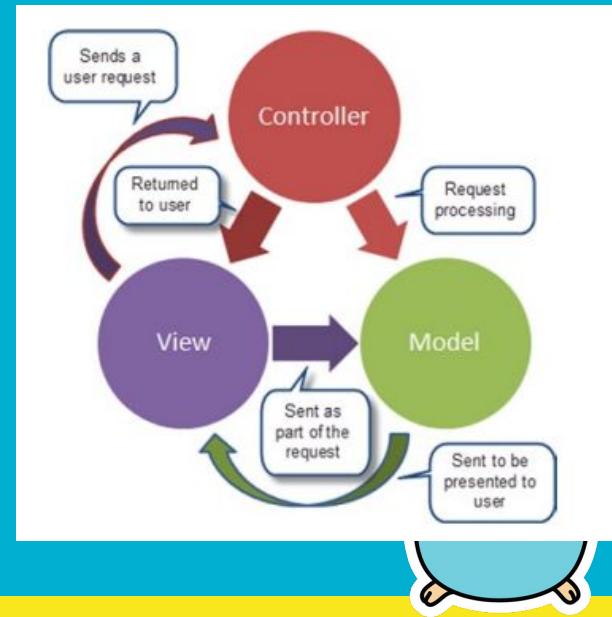
MVC架構

MVC簡介

(MVC) 架構模式由 Trygve Reenskaug 於 1978 年所提出，目的是為了「縮小人類精神面 Mental Model 與 數位系統 Digital Model 間的鴻溝」，企圖透過 MVC 功能性的分層歸類，而達到一個較好的系統設計與互動模型。

MVC劃分為三組主要元件：

- ❖ 模型(Model):
 - 封裝商業邏輯相關的資料，以及對資料操作的處理方法
 - 資料庫的存取
- ❖ 檢視(View): 定義 $obj \rightarrow \text{map} \rightarrow \text{UI}$
 - 負責 UI 顯示，如 HTML、CSS 等介面設計配置
- ❖ 控制器(Controller):
 - 接收來是所有使用者請求 (request)
 - 協調(Coordinator)角色，選擇對應的 View 呈現給 UI 畫面
 - 處理 Model 及回應使用者輸入和互動



MVC簡介

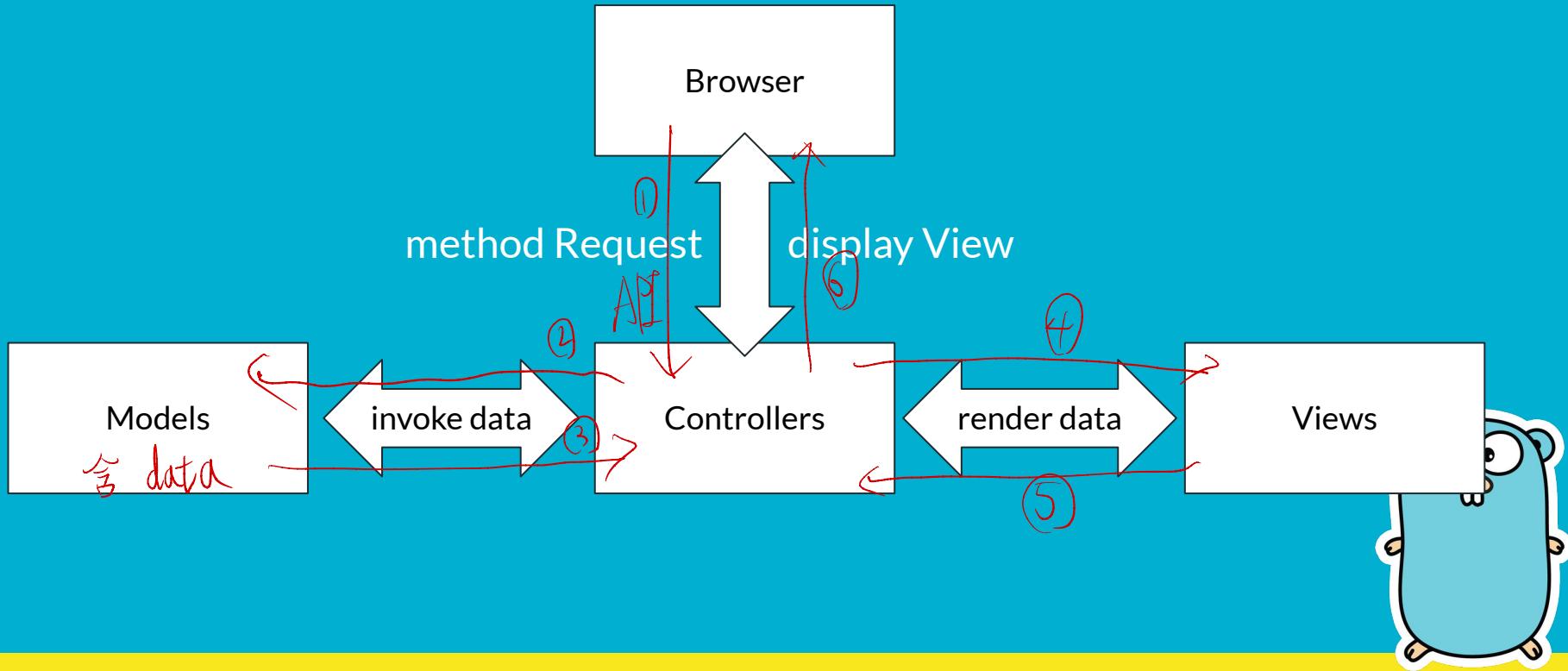
特色：

- ❖ 結構清晰：
 - 以MVC開發的Web在維護上較為容易
- ❖ 開發更有效率：
 - MVC 提供了更好的組織和分工方式，開發人員可以同時進行Model、View 和Controller的開發，從而提高了開發效率。此外，它也促進了團隊協作，因為不同部分的代碼可以獨立地開發和測試。
- ❖ 容易維護：
 - MVC架構讓測試驅動式開發式或單元測式變得容易

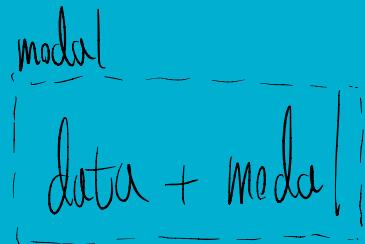
前後端分離
MVC



MVC流程範例



Model



- ❖ 用於封裝商業邏輯相關的資料，以及對資料操作的處理方法
- ❖ Model主要是扮演資料存取層的角色
- ❖ Model沒有限定特定型式的實作，因此可以是Class類別、Entity、DataSet
- ❖ 正常情況下是由Controller呼叫Model，再將Model物件傳送到View中顯示
- ❖ 存取Entity Framework資料



View

- ❖ 檢視(View)是用於呈現和顯示數據的用戶界面元素。View 指需要渲染的頁面，通常是範本頁面，渲染後的內容通常是 HTML。
- ❖ View可以輕鬆處理 XML、JSON 或其他資料類型的渲染。

input Model(object) → output html (UI)



Controller

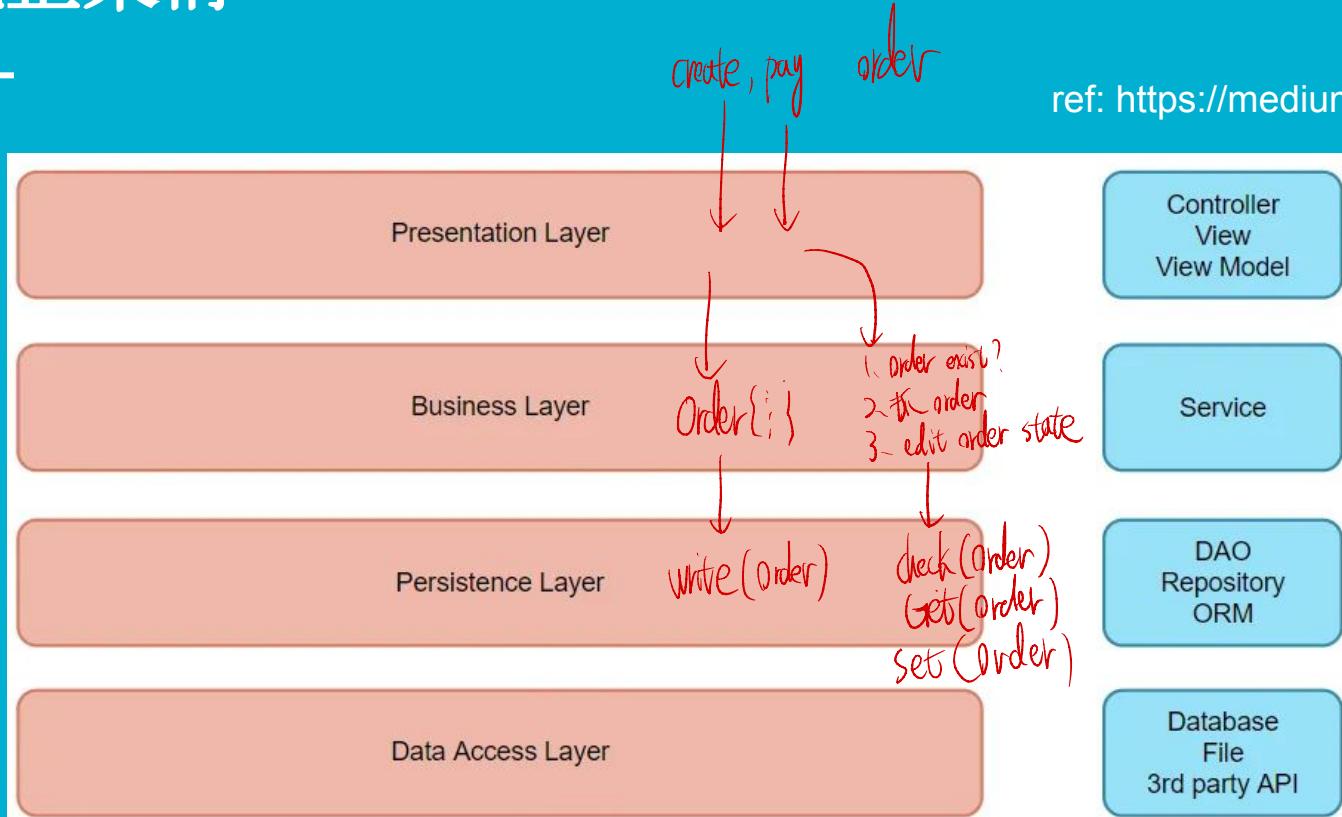
- ❖ Controller是View與Model二者間的一個中介者，負責使用者與系統之間的互動，View與Model間的資料、參考的傳遞與處理，例如Input及Output都是由Controller統籌。
 - Input指的是HTML輸入或URL Request處理
 - Output指的是View的生成

trace code: <https://github.com/leon123858/example-aspnet-mvc>



分層架構

完整架構



ref: <https://medium.com/@chunyeung>



各層說明

- ❖ 展示層 (controller): 調用業務層運行完整 API 流程
- ❖ 業務層 (service): 完成核心業務邏輯, 提供給展示層調用
- ❖ 持久層 (repository): 不包含邏輯, 純粹的外部調用介面
- ❖ 儲存層 (data access): 被調用的外部服務, 如: 資料庫, 第三方

只允許上往下調用, 由上到下可以跨層, 但盡量不要。



目的

1. 易於理解與開發

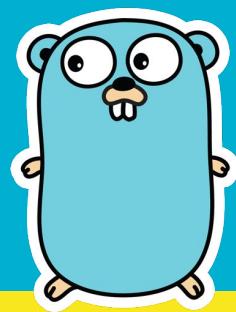
分層模式相信是很容易被理解，因為程式邏輯很線性，資料流主要就是從上到下或是從下往上，實行起來並不復雜。大多數公司亦通過按層來提出UI需求、業務需求等來開發應用程序，所以分層模式自然成為大多數業務應用程序開發的直覺。

2. 關注點分離 (**Separation of concerns, SoC**)

每一層中的組件只處理與該層相關的邏輯。例如，表示層中的組件只處理表示邏輯、業務層中的組件只處理業務邏輯。這種程式分類方法可以有效地將軟體中不同的角色進行分割，而定義明確的組件接口和有限的組件範圍，可以讓開發、測試與維護（可能）變得更簡單。

3. 可測試性

由於分層明確，在單元測試中其他層更容易被模擬，使得分層模式相對容易測試。開發人員可以選擇模擬表示層以對業務組件單獨進行測試，也可以模擬數據層以對持久組件進行測試。

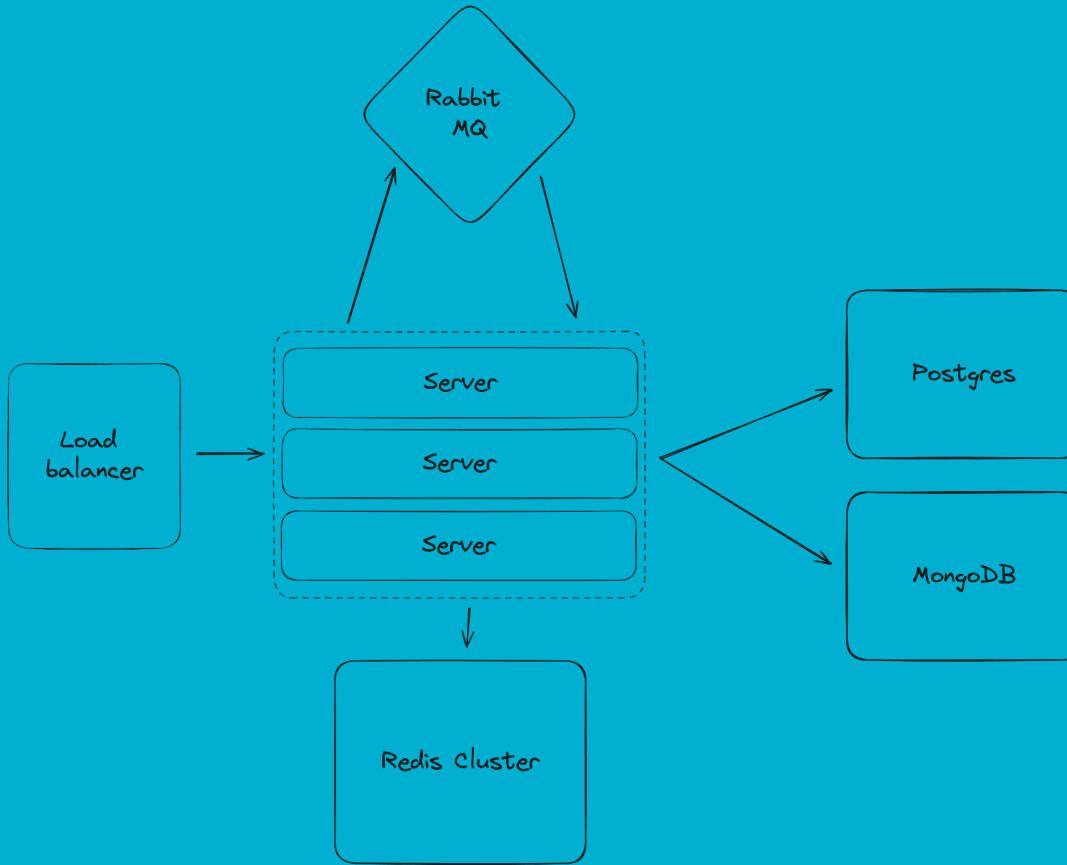


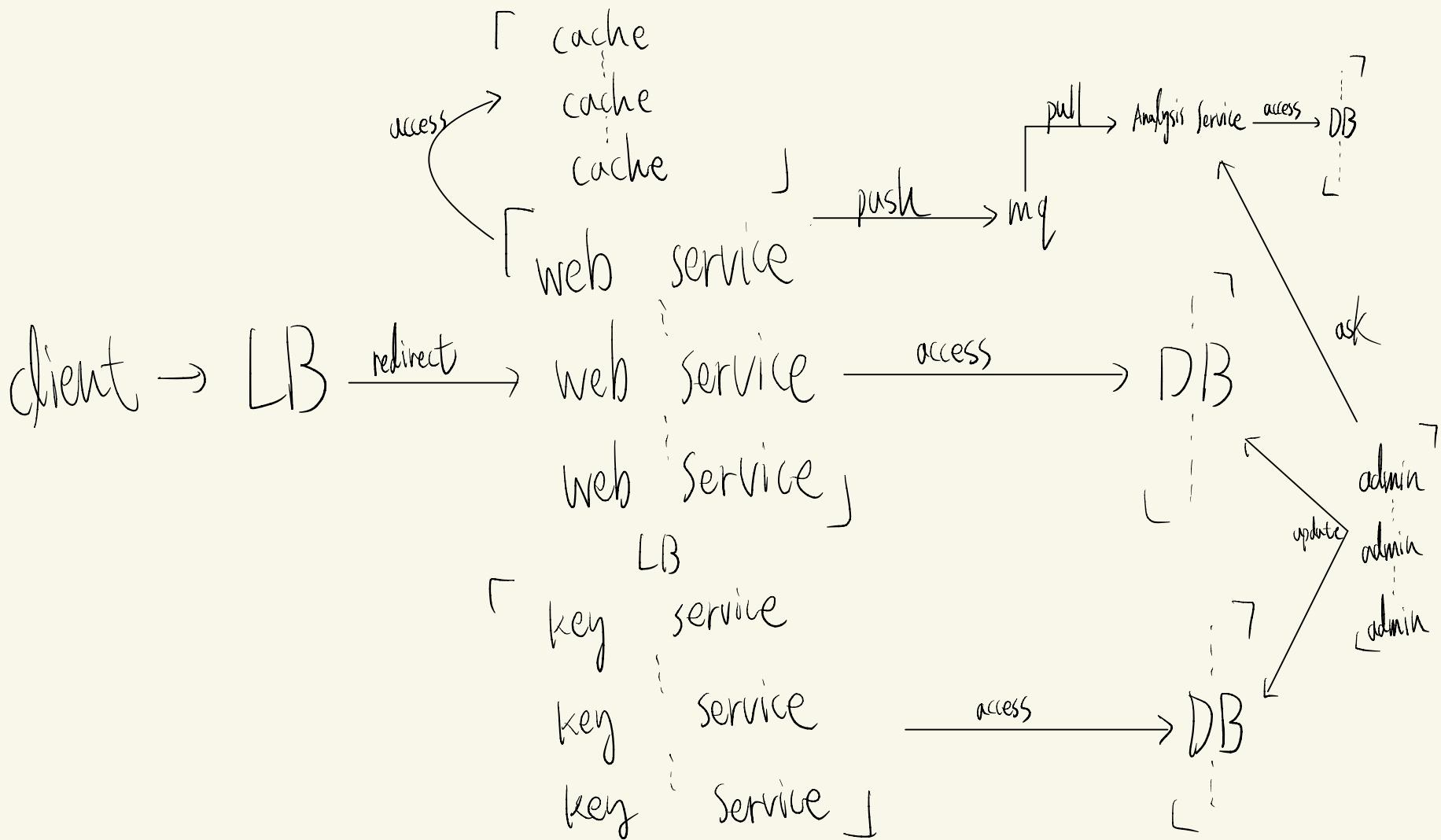
Lab

短網址服務實作:



系統設計





系統設計



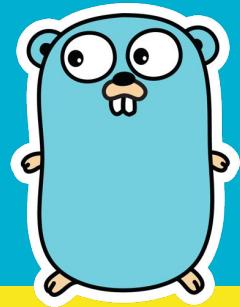
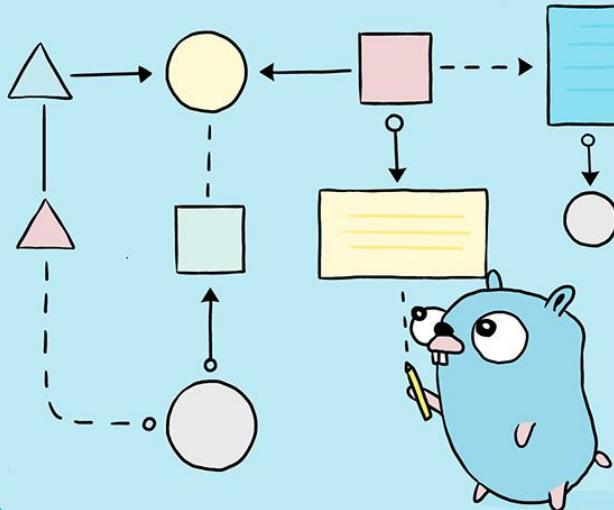
Controller

Service

Repository



Q&A 時間



作業: url shorten service 實作

SPEC

1. 輸入長網址得到縮短網址
2. 輸入短網址轉址長網址
3. 創建用戶, 取得用戶識別號
4. 短網址關聯用戶識別號

Link: <https://github.com/leon123858/go-tutorial/tree/main/short-url>

