



# 取得 GET 和 POST 參數



designed by freepik

Estimated time:

50 min.

 資訊工業策進會 Institute for Information Industry

【Key Points】：

## 學習目標

- 7-1: 解析 Query String
- 7-2: 使用 body-parser 套件
- 7-3: Postman 測試



7-1

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

1. 在這節課中我們將會介紹 HTTP 協定中的請求方法 (Request Method)
2. 學習如何使用 Node.js 自帶的 `QueryString API` 去解析和格式化URL中的查詢字串。
3. 使用 `body-parser` 這個很常用的 Express 中介軟體，目標是對 `post`請求的內容進行解析。
4. 最後，使用 Postman 這套工具來測試我們的功能。
5. 在最後的練習會希望學員們撰寫GET、POST 請求類型的API 各一個，當前端post JSON 資料時需要被 `body-parser` 正確解析成物件。

### 【Key Points】：

如何使用 Node.js 自帶的 `QueryString API` 去解析和格式化URL中的查詢字串  
`body-parser` 是個很常用的 Express 中介軟體，目標是對 `post`請求的內容進行解析  
使用 Postman 這套工具來測試我們的功能

## 7-1: 解析 QueryString

- 什麼是 Query String
- 關於QueryString API
- 動手寫看看



designed by freepik



designed by freepik

7-2

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

1. 簡介 Query String 。
2. <https://ccu.edu.tw:8080/path/?query/#fragment> ← 裡面的 query 欄位就是了
3. 這節我們會學習如何使用 Node.js 自帶的 QueryString API 去解析和格式化URL中的查詢字串。
4. 解析出來的 Query String 雖然可以在後端中被當object 使用
5. 不過也因為原先是字串型別，因此不像JSON一樣可以解析出其他型態 (例如 int、bool)

【Key Points】：

什麼是 Query String?

<https://ccu.edu.tw:8080/path/?query/#fragment> ← 裡面的 query 欄位就是了  
如何使用 Node.js 自帶的 QueryString API 去解析和格式化URL中的查詢字串

# 什麼是 Query String

- 還記得我們前面介紹的 URL 格式嗎？

**https :// ccu.edu.tw : 8080 / path / [? query] / [# fragment]**  
協議 :// Host : Port / 路徑 / 資源查詢 / 標記片段

- 它的寫法就是在路徑後面加上 ? 這個符號，並且接上查詢參數
- 參數名稱跟值用 = 符號相連，有多個參數的話用 & 符號隔開

舉個例子

**http://xx.com/hello?name=Shan&age=18**

- 前面灰色的部分為資源的路徑，接著用 ? 表示後面為Query String，而上面範例代表我們要查一個 name = Shan 且 age = 18 的資源

7-3



- QueryString ( 查詢字串 ) 就是 URL 結尾附掛的資訊。
- 主要在Url上傳遞資料，可能是一個搜尋字串、頁碼、某項特定的指標之類的
- 中間紫色的部分就是 URL 中用於表達資源查詢的字串(QueryString)
- 在URL 中 保留了 ? 這個符號作為查詢字串的識別開頭，凡是接在後面的字串都是用於查詢的參數
- 每個參數的名稱跟值用 = 符號來相連，是一種KEY / Value的組合。
- 如果有多個參數的話要用 & 符號來隔開

【Key Points】：

QueryString ( 查詢字串 ) 就是 URL 結尾附掛的資訊

每個參數的名稱跟值用 = 符號來相連，是一種KEY / Value的組合。

如果有多個參數的話要用 & 符號來隔開

# 關於 QueryString API

- 在 Node.js 裡面提供了 `querystring` 模組，來解析查詢字串
- 提供方法有 `parse` ; `stringify` ; `escape` ; `unescape`  
分別為 **解析字串**；**把物件序列化為QS**；**URL編碼**；**URL解碼**
- 我們可以在程式碼中用`require('querystring')`引用它

```
const express = require('express');
const querystring = require('querystring');
```

- 官方文件 <https://nodejs.org/api/querystring.html>

7-4



1. Node.js 使用 `querystring` 模組來解析 Query String 。
2. 在引用之後可以呼叫 `parse()` 函數來為我們解析網址上的查詢字串 。
3. 除了 `parse()` 的方法之外，還有其他常用的方法如下
  - `parse() = decode()` · 把URL上的查詢字串解析成 key/value 集合
  - `stringify() = encode()` · 把 js object 序列化成 Query String
  - `escape()` · 把查詢字串用 URL百分比編碼，例如：中文 → %E4%B8%AD%E6%96%87
  - `unescape()` · 把用URL百分比邊碼的字串還原成指定編碼格式(預設 UTF-8)，例如：  
%E4%B8%AD%E6%96%87 → 中文

## 【Key Points】：

Node.js 使用 `querystring` 模組來解析 Query String

呼叫 `parse()` 函數來為我們解析網址上的查詢字串

除了 `parse()` 的方法之外，還有其他常用的方法(`stringify`, `escape`, `unescape`)

## 動手寫看看 — 開新專案

- 輸入：**mkdir myapi** 建立名為 **myapi** 的資料夾作為專案名稱。
- 輸入：**cd myapi** 切換到該目錄

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. 著作權所有，並保留一切權利。
Active code page: 65001

C:\Work>mkdir myapi
C:\Work>cd myapi
C:\Work\myapi>
```

7-5

讓我們從頭開始創建一個 Node 專案。

在 c:\Work ( 例如 ) 建立一個叫做 myapi 的資料夾作為專案名稱 ( 也可改為你想要的名字 ) 。

1. 按下組合鍵「Windows + R」
2. 輸入「cmd」後，點按「確定」按鈕。 ( 啟動「命令提示字元」 )
3. 上述步驟1+2, 也可以: 按下組合鍵「Windows + S」，然後鍵盤輸入「cmd」找到「命令提示字元」。
4. 輸入：**mkdir myapi** 建立名為 **myapi** 的資料夾作為專案名稱。
5. 輸入：**cd myapi** 切換到該目錄

常用cmd指令整理：

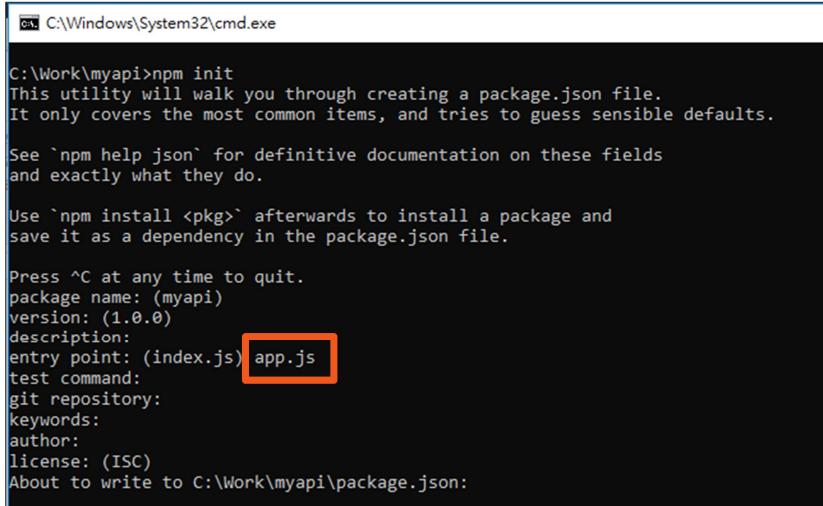
查詢目錄 (dir)、建立目錄 (md, mkdir)、變更目錄 (cd, chdir)、刪除目錄 (rd, rmdir)、檔案重新命名 (ren, rename)

### 【Key Points】：

查詢目錄 (dir)  
建立目錄 (md, mkdir)  
變更目錄 (cd, chdir)

# 動手寫看看 — 專案初始化

- 輸入：**npm init** 初始化一個專案，這裡我們在 **entry point** 選項時把入口文件改成 **app.js**，其他就維持預設，最後記得輸入 **yes**。



```
C:\Windows\System32\cmd.exe
C:\Work\myapi>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (myapi)
version: (1.0.0)
description:
entry point: (index.js) app.js
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Work\myapi\package.json:
```

7-6



1. 建立完後在資料夾中運行 **npm init** 將該資料夾轉成一個 node 專案。
2. 若無特殊需求，可一路按確認鍵到底。
3. 中途我們在 **entry point** 選項時需要把入口文件改成 **app.js**（或其他名稱，例如：**index.js**）
4. 在 **package.json** 這個檔案中，使用者可以定義應用名稱 (**name**)、應用描述 (**description**)、關鍵字 (**keywords**)、版本號 (**version**)、應用配置 (**config**)、主頁 (**homepage**)、作者(**author**)、版本庫 (**repository**)、bug的提交地址 (**bugs**)、授權方式(**licenses**)... 等。
5. 如果在有 **package.json** 的專案目錄下，執行 **npm install**，**npm** 便會依照 **package.json** 的內容去下載套件並且佈置好執行環境。

## 【Key Points】：

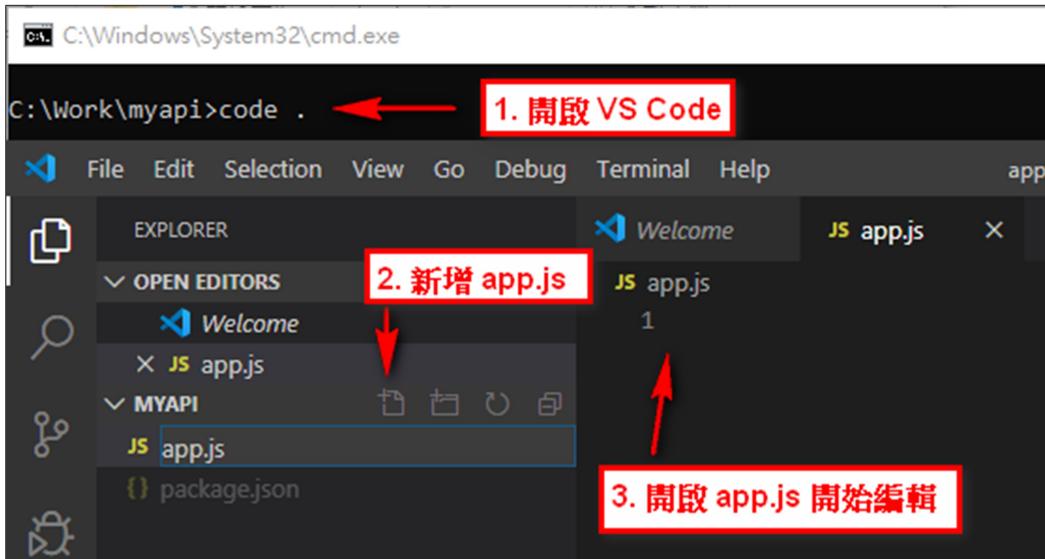
**npm init**

**entry point** 選項時需要把入口文件改成 **app.js**

老師可以帶著學生看一下 **package.json** 這個檔案，說明每個項目可能需填的內容

# 動手寫看看 — app.js

- 輸入：`code .` 在當下目錄開啟 VS Code，並新增 app.js 文件



7-7

- 在當前目錄下執行 `code .`，即可執行 VS Code 並且編輯當前的工作目錄
- 在 VS Code 裡面操作新增 app.js 文件
- 打開該文件編輯程式碼

<Note>也可以在檔案總管，滑鼠右鍵點按資料夾，從快捷功能表選擇「以Code開啟」

<Note>建議以Code開啟資料夾而非單一檔案，因為，專案通常由一組相關的檔案構成。

## 【Key Points】：

執行 `code .`，即可執行 VS Code 並且編輯當前的工作目錄  
也可以在檔案總管，滑鼠右鍵點按資料夾，從快捷功能表選擇「以Code開啟」  
前面章節已經有請同學們安裝過 VS Code 了，如果有還沒安裝的在請老師協助安裝

# 動手寫看看 — 引用與測試 parse()

- 因為 “`querystring`” 為 `Node.js` 內建的模組，因此我們不必另外安裝就可以直接 `require` 引用它。
- 接著我們就可以使用裡面的方法了，首先我們來看看解析字串用的 `parse()` 方法，我們設計一個查詢字串 `name=Shan&age=18`

- 使用 `querystring.parse()` 讀入  
解析成 `object`

```
var querystring = require('querystring');
var str = 'name=Shan&age=18';
var obj = querystring.parse(str);

console.log(obj);
```

- 可以試印出  
`obj.name` 跟 `obj.age`

```
C:\Work\myapi>node app.js
[Object: null prototype] { name: 'Shan', age: '18' }
```

7-8



`parse()` 函數有四個參數：

`querystring.parse(str, [sep], [eq], [options])`

- str 是 Query String
- sep 是「Separator」，也就是字串的分隔字元，預設是 ‘&’，通常不做變數
- eq 則是字串與值的對應字元，預設是 ‘=’，通常不做變數
- options 是在 v6.0.0 版後才加入的新功能，裡面有兩個屬性可以設定分別是
  - `decodeURIComponent` [<Function>](#) 解碼查詢字符串中百分比編碼的字符時使用的函數。  
默認值：`querystring.unescape()`
  - `maxKeys` [<number>](#) 指定要解析的鍵的最大數量。指定0刪除鍵計數限制。  
默認值：1000。

當執行 `parse` 解析成 `object` 後我們便可以直接在 js 裡面操作該資料

值得注意的是因為解析的對象是字串，所以 `age=18` 這邊解析的 18 不是整數型別，而是字串型別

【Key Points】：

`parse()` 函數的四個參數

執行 `parse` 解析成 `object`

值得注意的是：解析的對象是字串，所以 `age=18` 這邊解析的 18 不是整數型別，而是字串型別

## 動手寫看看 — `stringify()`

- 剛剛我們把查詢字串解析成物件(object)了，因此可以在 js 中使用它，如果要把它再轉回去 Query String 呢 ?? 交給 `stringify()` 吧！
- 這邊我們簡單改寫一下，把剛剛的 name 跟 age 值修改並且執行 `stringify()`，轉成新的 Query String 印出來。

```
var querystring = require('querystring');
var str = 'name=Shan&age=18';
var obj = querystring.parse(str);

obj.name = "珊";
obj.age = 20;

var newstr = querystring.stringify(obj);
console.log(newstr);
```

```
C:\Work\myapi>node app.js
name=%E7%8F%8A&age=20
```

7-9



- `stringify` 也跟 `parse` 一樣有四個參數
- `querystring.stringify(obj[, sep[, eq[, options]]])`**
- 只是把第一個參數的 query string 改成 object 而已，其他都一樣
- 當遇到除了英數字的字元跟一些保留符號外的字都會進行 URL Encode
- URL Encode 屬於 % + 16進制的組合

### 【Key Points】：

`stringify` 也跟 `parse` 一樣有四個參數

第一個參數的 query string 改成 object 而已，其他都一樣

URL Encode 屬於 % + 16進制的組合

# URL Encode

- 有觀察到嗎？

剛剛把 name 改成中文字的部分，被編碼成 **URLEncoder** 的形式了

```
C:\Work\myapi>node app.js  
name=%E7%8F%8A&age=20
```

- 在 **URLEncoder** 裡面 會把除了英數字 (0~9 a~z A~Z) · 以及保留的符號 \$-\_.+!\*‘()，以外的字元都全都轉化成 %16進制 的形式。

- 為什麼要做URL編碼呢？

- 
1. ASCII 的控制字元
  2. 一些非ASCII字元
  3. 一些保留字元
  4. 一些不安全的字元

7-10



在 URL encode 裡面 會把除了英數字 (0~9 a~z A~Z) · 以及保留的符號 \$-\_.+!\*‘()，以外的字元都需要被轉化

哪些字元是需要轉換的呢？

## 1. ASCII 的控制字元

這些字元都是不可打印的，自然需要進行編碼。

## 2. 一些非ASCII字元

這些字元自然是非法的字元範圍。轉化也是理所當然的了。

## 3. 一些保留字元

很明顯最常見的就是 “&” 了，這個如果出現在url中了，那你認為是url中的一個字元呢，還是特殊的參數分割用的呢？

## 4. 就是一些不安全的字元了。

例如：空格。為了防止引起歧義，需要被轉化為 “+”。

明白了這些，也就知道了為什麼需要轉換了，而轉換的規則也是很簡單。

按照每個字元對應的字元編碼，不是符合我們範圍的，統統的轉化為%的形式也就是了。自然也是16進制的形式

### 【Key Points】：

哪些字元是需要轉換的呢？

要注意的是 urlencode 是基於字元編碼的，

如果中文字用不同的編碼類型去編比如說 big5 或 gbk，那麼URL encode 出來的結果也會不同

## 動手寫看看 — escape()、unescape()

- 剛剛我們學習了 URL Encode 的編碼形式，在 parse 跟 stringify 實際會預設幫我們處理轉換。
- 如果另外操作的話 querystring 模組也提供兩個函數來給我們調用
  - escape()
  - unescape()

```
var querystring = require('querystring');
var str = "name=珊&age=20";
console.log("原始字串 = " + str);
var enc_str = querystring.escape(str);
console.log("編碼後的字串 = " + enc_str);
var dec_str = querystring.unescape(enc_str);
console.log("解碼後的字串 = " + dec_str);
```

```
C:\Work\myapi>node app.js
原始字串 = name=珊&age=20
編碼後的字串 = name%3D%E7%8A%26age%3D20
解碼後的字串 = name=珊&age=20
```

7-11



- 剛介紹完 URL Encode，在 querystring API 裡面也有兩個方法是關於處理 encode 跟 decode 的
- escape(str) 可以對原始字串進行 URL 編碼
- unescape(str) 可以對已經編碼過的字串進行解碼
- 其實 parse 跟 stringify 兩個方法本身就會自動處理 URLEncode 了，沒特殊情況是不用再調用

到這邊 querystring API 的用法大概就到一個段落了，

接下來要講如何使用 Express 裡面的 body-parser 套件，來解析更多的請求內容

### 【Key Points】：

escape(str) 可以對原始字串進行 URL 編碼

unescape(str) 可以對已經編碼過的字串進行解碼

parse 跟 stringify 兩個方法本身就會自動處理 URLEncode

## 7-2: 使用 body-parser 套件

- 安裝 body-parser 套件
- 使用 body-parser
  - app.set()
  - app.get()



designed by freepik



designed by freepik

7-12

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 前面的章節只能解析帶在 URL 上面的資料
- 不過有些資料並不適合顯示在網址上面，例如：密碼
- 接下來，我們會學習如何使用 body-parser 套件來解析來自 POST 請求中包在 Body 的參數
- 並且銜接下一知識點的 Postman
- 如果同學有興趣還可以用前面教過的 jQuery 的 post() 方法來發送請求測試。

### 【Key Points】：

有些資料並不適合顯示在網址上面

如何使用 body-parser 套件來解析來自 POST 請求中包在 Body 的參數

銜接下一知識點的 Postman

# 安裝 body-parser 套件

- 通常在安裝 Express.js 時就會一起安裝了，如果沒有安裝的話請輸入：`npm install body-parser --save`
- 安裝完可以輸入：`npm list -s --depth=1` 檢查有沒有成功

```
C:\Work\myapi>npm list -s --depth=1
myapi@1.0.0 C:\Work\myapi
`-- express@4.17.1
  +-- accepts@1.3.7
  +-- array-flatten@1.1.1
  +- body-parser@1.19.0
    +-- content-disposition@0.5.3
    +-- content-type@1.0.4
    +-- cookie@0.4.0
    +-- cookie-signature@1.0.6
      +-- parseSignedCookie@1.0.0
```

7-13



- Express 在Node.js中是個跟 http 對應的第三方核心模組，用於處理http請求。
- 其中有個好用的套件(在Express 4.0之後獨立出來)，就是 body-parser
- body-parser屬於中介軟體(middleware)，它會攔截和解析所有的請求，協助解析用戶端POST過來的資料。
- Express 在安裝時若已自動安裝的話就可以不用在裝獨立套件
- 如果引用過程有出錯在另外安裝即可，安裝指令: `npm install body-parser --save`

【Key Points】：

Express 用於處理 http 請求

body-parser 屬於中介軟體(middleware)可以攔截和解析所有的請求

安裝指令: `npm install body-parser --save`

# 使用 body-parser

- 安裝完成後我們可以使用下面方式啟用 body-parser 的功能。

```
const bodyParser = require('body-parser');
const express = require('express');
const app = express();
app.use(bodyParser.json()); //處理 JSON 資料
```

- body-parser 套件主要可以解析 4 種方式的 POST 帶過來的參數。
  - bodyParser.json() → 處理 JSON 資料 (上面的範例)
  - bodyParser.urlencoded() → 處理 UTF-8 編碼的資料
  - bodyParser.raw() → 處理 Buffer 流資料
  - bodyParser.text() → 處理文字資料

7-14



body-parser 功能放在我們叫做 **middleware** 的中介層上，

使得每個路由被拜訪前，都會經過這個 **middleware**，由這個 **middleware** 先進行參數上的解析，最後在拜訪我們的路由。

換句話說就是攔截和解析所有的請求的參數。

而它解析的處理方式依照內容有不同的方法，分別是

- bodyParser.json() → 處理 JSON 資料 (上面的範例)
- bodyParser.urlencoded() → 處理 UTF-8 編碼的資料
- bodyParser.raw() → 處理 Buffer 流資料
- bodyParser.text() → 處理文字資料

在範例中我們從 app.use( ) 方法中呼叫 bodyParser.json()，這邊沒有另外設定路由，表示會對該 app 的所有的請求做 JSON Parser 的處理。

## 【Key Points】：

body-parser 功能放在我們叫做 **middleware** 的中介層上

bodyParser.json() 與 bodyParser.urlencoded() 這兩個常用

app.use( )

# 使用 body-parser

- 剛剛使用 `app.use()` 方式呼叫的話對所有的請求資料做處理(全域)。
- 如果我們想要不同請求間個別處理的話就得用express的 `post` 或 `get` 去呼叫，這麼一來便會是區域性。

```
const bodyParser = require('body-parser');const express = require('express');
const app = express();
var jsonParser = bodyParser.json() // 解析 JSON 資料
var urlencodedParser = bodyParser.urlencoded() //解析 Form Data

app.post('/json', jsonParser, function (req, res) {
    console.log(req.body); // 可以從 req.body 中抓取解析的資料
    res.send('Form Data = ' + req.body)
})
app.post('/form', urlencodedParser, function (req, res) {
    console.log(req.body);
    res.send('JSON Data = ' + req.body)
})
app.listen(3000);
```

- 我們把 `json` 跟 `form` 資料請求分別用兩個 POST 路由分開處理。
- 因為資料的性質不同，所以得宣告不同的 `bodyParser` 來使用。
- 一旦收到對應路由的請求後就會觸發不同的 Parser 來解析
- 一旦解析完成我們就可以在後續的程式中從 `req.body` 中取得解析完的 `object`
- 使用 `app.use()` 方式呼叫的話對所有的請求資料做處理(全域)。
- 如果我們想要不同請求間個別處理的話就得用express的 `post` 或`get` 去呼叫，這麼一來便會是區域性。

## 【Key Points】：

使用 `app.use()` 方式呼叫的話對所有的請求資料做處理(全域)

想要不同請求間個別處理的話就得用express的 `post` 或`get` 去呼叫，這麼一來便會是區域性

本例，`json` 跟 `form` 資料請求分別用兩個 POST 路由分開處理

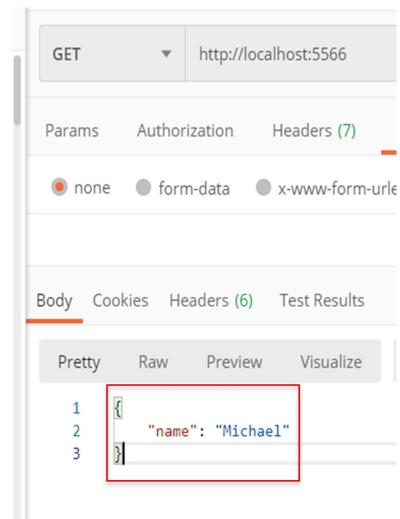
## app.set( )

```
const express = require('express')
const app = express()

app.set('name', 'Michael');

app.get('/', (req, res, next) => {
  res.json({name: app.get('name')})
})

module.exports = app
```



7-16

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- app有一個set( )方法可以存放key與value。
- app.set( )可以傳入兩個參數，第一個是key，第二個是value。
- es6語法中有新的數據結構Map，類似字典，app.set( )跟Map的語法一樣都是透過set( )方法設定key value。
- app.set( )的value還可以放array、object、false、null、空字串等。
- app.set( )的value不建議放undefined，因為一但是回傳JSON或有對app.get( )做JSON處理，將會直接刪除該Object的key value。

### 【Key Points】：

app有一個set( )方法可以存放key與value

app.set( )可以傳入兩個參數，第一個是key，第二個是value

app.set( )的value還可以放array、object、false、null、空字串等

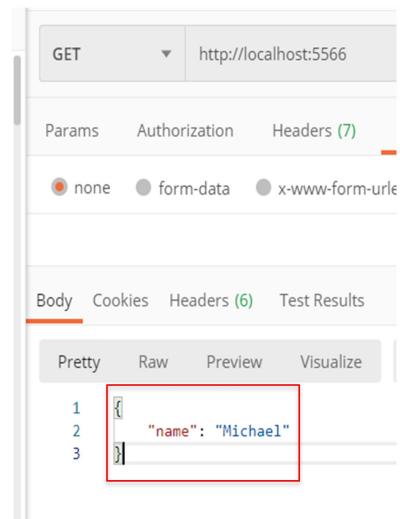
## app.get( )

```
const express = require('express')
const app = express()

app.set('name', 'Michael');

app.get('/', (req, res, next) => {
  res.json({name: app.get('name')})
})

module.exports = app
```



7-17

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- `app.get()` 在早先前介紹他可以當作router handler function。
- 但其實`app.get()`可以透過key取得value。
- `app.get()` 可以只傳入一個參數，將字典的key帶入當參數。
- es6語法中有新的數據結構Map，類似字典，`app.get()`跟Map的語法一樣都是透過`get()`方法取得 value。
- `app.get()`會依據參數，回傳該物件相對應的值。

### 【Key Points】：

`app.get()` 在早先前介紹他可以當作router handler function

其實`app.get()`可以透過key取得value

`app.get()` 可以只傳入一個參數，將字典的key帶入當參數；依據參數，回傳該物件相對應的值

## 7-3: Postman 測試

- 安裝 Postman
- 發送 GET 請求
- 發送 POST 請求



designed by freepik



designed by freepik

7-18

III 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

剛剛我們寫好了我們的 API Server，裡面有兩個 POST 方法等著我們去調用，不過在沒有寫前端網頁的情況下我們是否可以用其他工具來發送 HTTP 請求來做測試呢??

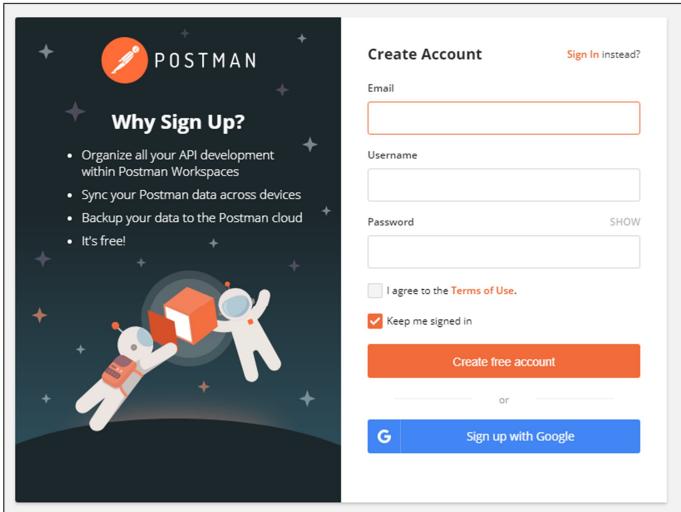
- 這邊就一定要來介紹這個測試 API 的神器 - Postman !
- 本章節會介紹 Postman 的基本用法
- 實作發送 GET
- 實作發送 POST
- 除了 Postman 之外還有 RESTer、Firebug... 等工具

【Key Points】：

測試 API 的神器 - Postman  
介紹 Postman 的基本用法  
實作發送 GET 與 POST

# 安裝 Postman

- 到 <https://www.postman.com/downloads/> 下載桌面版的 Postman
- 下載完後點擊 exe 執行檔進行安裝，如果看到以下畫面表示完成



7-19



- Postman 是一個可以模擬 HTTP Request 的工具，其中包含常見的 HTTP 的請求方式  
例如：GET、POST、PUT、DELETE
- 而它的主要功能就是能夠快速的測試你的 API 是否能夠正常的請求資料，並得到正確的請求結果。
- 除了快速測試的功能以外，Postman 還擁有非常容易使用的介面，以及 Collection 的功能，我們會先介紹 Request 的功能，並實際操作一次 GET 與 Post 這兩個 Method。
- Postman 有 Chrome 線上應用程式的版本也有桌面版本
- 這邊以桌面版本為主，老師希望可以跨平台教學的話可以請同學安裝 Chrome APP 版

桌面版：

<https://www.postman.com/downloads/>

Chrome 版：

<https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjb dggehcdcbncddomop?hl=zh-TW>

【Key Points】：

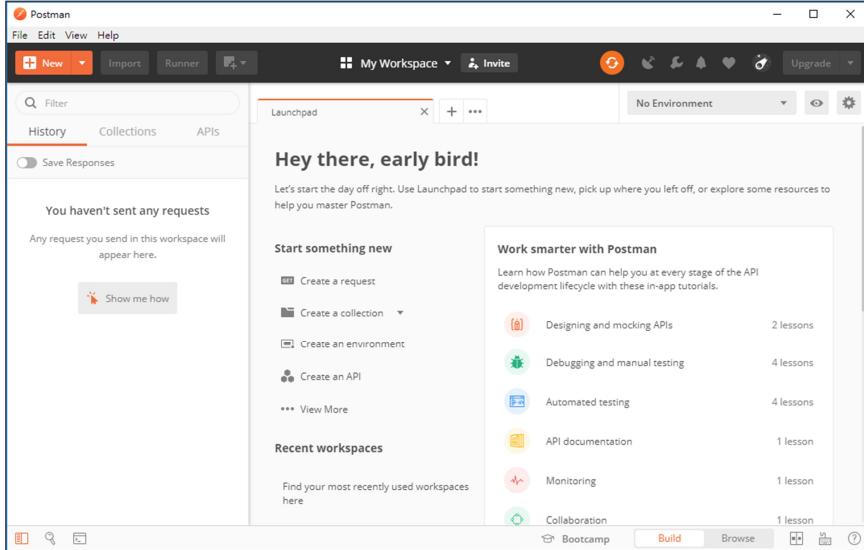
Postman 是一個可以模擬 HTTP Request 的工具

常見的 HTTP 的請求方式: GET、POST、PUT、DELETE

Postman 有 Chrome 線上應用程式的版本也有桌面版本，本課程以桌面版本為準

# 安裝 Postman

- Postman 使用需要註冊帳號，同學們可以使用Google帳號快速登入，或是用 Email 辦理，出現以下畫面就是登入成功了。



7-20



- Postman 使用需要註冊帳號
- 同學們可以使用Google帳號快速登入，或是用 Email 辦理
- 出現圖中的畫面就是登入成功了

- Postman 可以把請求資料、歷史數據...等跟帳號一同綁定，如果要再多平台或裝置間測試很方便
- Postman 有協作功能，也可以把人拉進同個工作群組

## 【Key Points】：

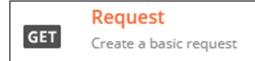
Postman 使用需要註冊帳號

使用Google帳號快速登入，或是用 Email 辦理

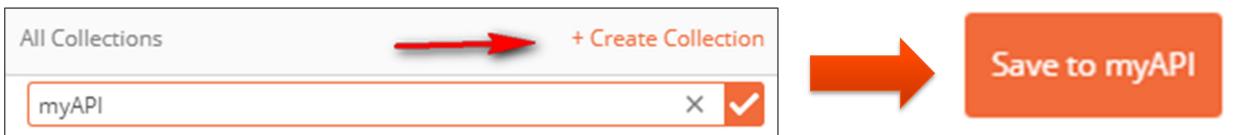
Postman 可以把請求資料、歷史數據...等跟帳號一同綁定

# 發送 GET 請求

- 點擊左上角的  按鈕，選擇新增一個 Request
- 在表單中填入這個請求的名稱，命名好懂就好



- 接著我們在下方新增一個目錄來存放這個Request，然後保存。



7-21

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 新增一個請求的方法就是按左上角的 New 鈕
- 選擇要新增的 Request 選項
- 接著需要輸入一些說明文字，這邊依照個人喜好設定即可
- 最後需要創建一個資料夾來保存這次的 HTTP Request
- 目錄的部分我們先暫時用 myAPI 即可，如果之後有團隊合作的需求會建議統一命名

## 【Key Points】：

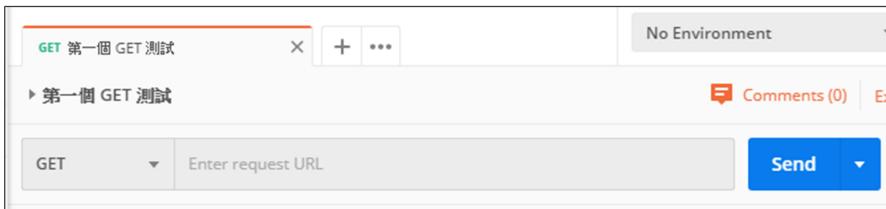
新增一個請求的方法就是按左上角的 New 鈕

選擇要新增的 Request 選項

接著需要輸入一些說明文字

# 發送 GET 請求

- 看到這個分頁畫面就是成功了



- 回到 app.js 我們加入一個 GET 請求的測試路由

```
app.get('/test', function (req, res) {  
    res.send(' query string = ' + req.query.name);  
});
```

- 記得要在終端機上執行 node app.js 把 server 運行起來

7-22

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 如果有出現一個剛剛命名的分頁代表新增成功了
- 回到 app.js 我們加入一個 GET 請求的測試路由
- 記得要到終端機上重新執行 node app.js 把 server 運行起來
- 在URL中加入 name=XXX 的查詢字串
- 成功的話就會在瀏覽器上顯示出來

## 【Key Points】：

回到 app.js 我們加入一個 GET 請求的測試路由  
記得要到終端機上重新執行 node app.js 把 server 運行起來  
在URL中加入 name=XXX 的查詢字串

## 發送 GET 請求

- 回到 Postman 上面，在 URL 框中輸入 <http://127.0.0.1:3000/test>



- 按下 Send 之後應該可以看到下方有 Response Body 的內容

The screenshot shows the Postman interface after sending a GET request to 'http://127.0.0.1:3000/test'. The status bar at the top indicates a successful 200 OK response with a time of 38ms and a size of 229 B. The response body is displayed below, showing the single line of text 'query\_string = undefined'. The 'Body' tab is selected in the navigation bar.

7-23

- 到 Postman 上在 URL 框中輸入 <http://127.0.0.1:3000/test>
- 按下 Send 之後把該請求送到Server
- 在 Postman 中我們可以很輕易地發送請求的 header 跟參數內容
- 也可以看到 Server 回應的狀態；內容、Cookies、Headers 等等資訊
- 這次執行結果會顯示 undefined 的原因是因為我們沒有在GET請求中加入參數

### 【Key Points】：

到 Postman 上在 URL 框中輸入 <http://127.0.0.1:3000/test>

按下 Send 之後把該請求送到Server

在 Postman 中我們可以很輕易地發送請求的 header 跟參數內容

## 發送 GET 請求

- 注意到剛剛的請求結果為 **query string = undefined**，因為我們還沒設置 Query String。Postman 可以很簡單的新增參數欄位

The screenshot shows the Postman interface with a 'GET' request type and URL 'http://127.0.0.1:3000/test?name=你的名子'. The 'Params' tab is selected. In the 'Query Params' table, there is one row with 'name' as the key and '你的名子' as the value. The entire table row is highlighted with a red box.

- 把 KEY 跟 VALUE 打上即可加在 URL 上面。再次執行結果

1 query string = 你的名子

7-24

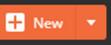
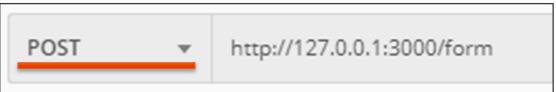


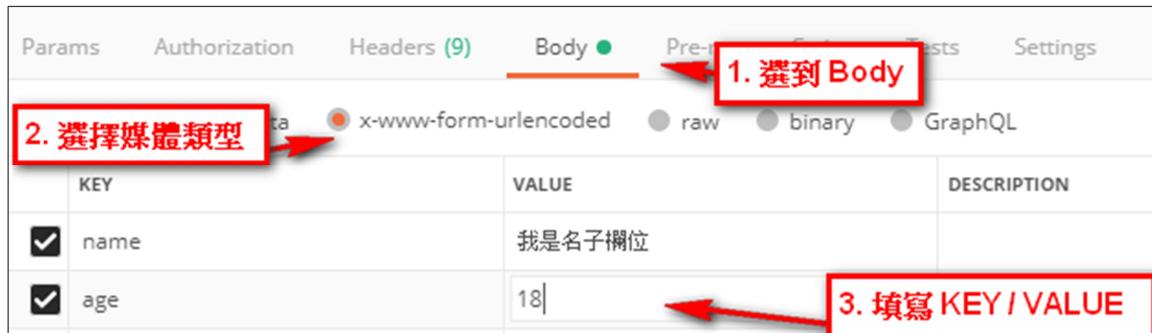
- 請求結果為 **query string = undefined**，因為我們還沒設置 Query String
- Postman 可以很簡單的新增參數欄位，切換到下面 Params 分頁下，可以直接再 KEY/VALUE 欄位加上數值
- 在左邊也可以輸入方便自己查看的描述欄位 (不會帶入 URL)
- 如果設置上去的話可以觀察到上方的 URL 欄位也跟著在變動
- 再次 Send 出去可以看到 Response 回來的內容已經是我們剛剛填入的值了，成功!

### 【Key Points】：

切換到下面 Params 分頁下，可以直接再 KEY/VALUE 欄位加上數值  
如果設置上去的話可以觀察到上方的 URL 欄位也跟著在變動  
再次 Send 出去可以看到 Response 回來的內容已經是我們剛剛填入的值了，成功!

# 發送 POST 請求

- 一樣點擊左上角的  按鈕，我們新增一個 POST 請求
- 把左邊請求方法改成 POST，填上路徑 
- 參數的部分選到 Body 分頁，類型選擇 x-www-form-urlencoded



The screenshot shows the Postman interface with the 'Body' tab selected. A red box labeled '1. 選到 Body' highlights the tab. Another red box labeled '2. 選擇媒體類型' points to the 'Content-Type' dropdown which is set to 'x-www-form-urlencoded'. A third red box labeled '3. 填寫 KEY / VALUE' points to the table where parameters are listed: 'name' with value '我是名子欄位' and 'age' with value '18'.

7-25

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 跟GET一樣新增一個請求(按左上角的 New 鈕)
- 選擇要新增的 Request 選項
- 接著我們需要把預設的 GET 改成 POST 請求
- 參數的部分選到 Body 分頁，類型選擇 x-www-form-urlencoded
- 之所以要選擇 x-www-form-urlencoded 類型是因為 form 預設是用 UTF-8 的 urlencoded 格式編碼

## 【Key Points】：

跟GET一樣新增一個請求(按左上角的 New 鈕)

把預設的 GET 改成 POST 請求

類型選擇 x-www-form-urlencoded

## 發送 POST 請求

- 可以發現回應的結果已經被解析成 Object 了
- 回到終端機上可以看到剛剛傳進來的資料被解析成 Object 並且打印出來，代表 `bodyParser.urlencoded()` 的解析方法成功！

```
Form Data = [object Object]
```

```
C:\Work\myapi>node app.js
body-parser deprecated undefined extended: provide ext
:6:35
{ name: '我是名子欄位', age: '18' }
-
```

7-26



1. 再次發送請求後可以發現我們送出去的表單已經被正確解析成 object 了
2. 如果這邊希望可以顯示內容的話，可以用 `JSON.stringify(obj)`，將 obj 序列化成 JSON 格式的字串
3. 後台的在終端機上會看到 Object 的內容被印了出來
4. form 其實還可以選擇其他格式，有興趣的同學可以先查詢
5. 我們下一節課也會帶著大家實做上傳文件功能的表單

### 【Key Points】：

再次發送請求後可以發現我們送出去的表單已經被正確解析成 object 了  
用 `JSON.stringify(obj)` 可將 obj 序列化成 JSON 格式的字串  
後台的在終端機上會看到 Object 的內容被印了出來

## 發送 POST 請求

- 接著我們試試看 JSON 請求的部分，假如我們只把 URL 改成 /



- 執行 Send 之後，回到終端機上會發現沒有任何資料，這是因為 jsonParse 無法解析 form 的內容格式

```
C:\Work\myapi>node app.js
body-parser deprecated undefined extend
:6:35
{}
```

7-27

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 再來我們試試看JSON API
- 首先把URL 上的 /form 改成 /json
- 接著執行 Send 把 表單送出
- 執行 Send 之後，回到終端機上會發現沒有任何資料
- 這是因為我們前端維持用 x-www-form-urlencoded 的模式去送，後端的 jsonParse 無法解析 form 的內容格式

### 【Key Points】：

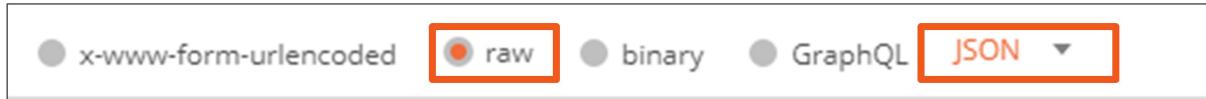
試試看JSON API

URL 上的 /form 改成 /json

前端維持用 x-www-form-urlencoded 的模式去送，後端的 jsonParse 無法解析 form 的內容格式

# 發送 POST 請求

- 如果要正確發送 JSON 媒體的話需要改用 raw 類型，並且在右邊選單中選擇媒體語言為 JSON 格式。



- 接著下面輸入正確的 JSON 內容

```
1 ~ {  
2   "name": "JSON 名子",  
3   "age": 18  
4 }  
5
```

- 再次送出就可以看到正確解析的結果了而且 age 正確被解析成整數類型了！

```
C:\Work\myapi>node app.js  
body-parser deprecated undefined  
:6:35  
{}  
{ name: 'JSON 名子', age: 18 }
```

7-28

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 因此如果要正確發送 JSON 媒體的話需要改用 raw 類型，並且在右邊選單中選擇媒體語言為 JSON 格式
- 注意這邊媒體格式一定要選對 JSON，否則還是會解析錯誤
- 另外Postman 會提醒你輸入的 JSON 格式是否正確
  - 旁邊還有 Beautify 的按鈕可以把內容縮排美化，請同學可以多多善用。
- JSON 的好處是可以傳遞字串以外的類型例如，Bool、Int、Array、Object ... 等等
- 因此在前後端都是用 JavaScript 開發得情況下會建議同學資料傳遞格式用 JSON 比較適合。

## 【Key Points】：

如果要正確發送 JSON 媒體的話需要改用 raw 類型，在右邊選單中選擇媒體語言為 JSON 格式  
媒體格式一定要選對 JSON，否則還是會解析錯誤  
前後端都是用 JavaScript 開發得情況下會建議同學資料傳遞格式用 JSON 比較適合

# Summary 〈 精華回顧 〉

- 在 URL 上預設會用 “?” 字元來識別 Query String 開始，並用 “&” 隔開。
- Node.js 有提供 **querystring API** 來解析查詢字串。
- URL Encode 是基於字元編碼的。
- 獨立安裝 body-parser 的指令是 **npm install body-parser --save**
- 查看已安裝的套件指令是 **npm list -s --depth=1**
- body-parser 有4種解析器可以  
解析 **JSON**、**UTF-8**、**Buffer**、**Text** 類型  
的資料。
- Postman 是一個可以模擬 **HTTP Request** 的  
工具，有 **Chrome APP** 跟桌面版。



7-29

III 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

1. 在 URL 上預設會用 “?” 字元來識別 Query String 開始，並用 “&” 隔開
  - Query String 是用 KEY-VALUE的形式，中間用 “=” 符號相連
  - 符號雖然預設是用 ? 與 & 識別，不過querystring 庫在解析上也可以選擇其他自定義分隔符
2. Node.js 有提供 **querystring API** 來解析查詢字串
3. URL Encode 是基於字元編碼的
  - 如果今天同一個中文字用不同的編碼類型去編比如說 big5 或 gbk，那麼URL encode出來的結果也會不同
4. 獨立安裝 body-parser 的指令是 **npm install body-parser --save**
  - express.js 在4.0版之後把 body-parser 獨立出來成為一個套件，因此如果只想安裝它的話可以單獨安裝
5. 查看已安裝的套件指令是 **npm list -s --depth=1**
6. body-parser 有4種解析器可以解析 **JSON**、**UTF-8**、**Buffer**、**Text** 類型的資料
7. Postman 是一個可以模擬 **HTTP Request** 的工具，有 **Chrome APP** 跟桌面版

## 【Key Points】：

在 URL 上預設會用 “?” 字元來識別 Query String 開始，並用 “&” 隔開  
body-parser 有4種解析器可以解析 **JSON**、**UTF-8**、**Buffer**、**Text** 類型的資料  
Postman 是一個可以模擬 **HTTP Request** 的工具，有 **Chrome APP** 跟桌面版

# 線上程式題

- **7-1 解析 Query String 的內容**  
請閱讀程式與說明，修改出正確的程式？
- **7-2 透過 app.set( ) 儲存 POST 的資料**  
請問，Web API 的程式該怎麼透過 app.set( ) 將收到的 `{"name": "Jack"}` 存起來。
- **7-3 如何讀出之前 app.set() 存入的資料**  
如何讀出之前 app.set() 存入的資料？

7-30



題目名稱: 7-1 解析 Query String 的內容

內容說明:

請閱讀程式與說明，修改出正確的程式

題目名稱: 7-2 透過 app.set( ) 儲存 POST 的資料

內容說明:

請問，Web API 的程式該怎麼透過 app.set( ) 將收到的 `{"name": "Jack"}` 存起來。

題目名稱: 7-3 如何讀出之前 app.set() 存入的資料

內容說明:

如何讀出之前 app.set() 存入的資料？

請閱讀教學系統的程式與說明，然後，到「// 作答區」填入答案。

答案有區分大寫小寫。

【Key Points】：

7-1 解析 Query String 的內容

7-2 透過 app.set( ) 儲存 POST 的資料

7-3 如何讀出之前 app.set() 存入的資料

# 課後練習題(Lab)

- **Module 7-1 – 解析 QueryString**
  - Lab01: 安裝Node.js 環境
  - Lab02: 建立主要執行檔 app.js
  - Lab03: 測試stringify
- **Module 7-2 – 使用 body-parser + Postman 測試**
  - Lab01: 安裝body-parser、Postman 環境
  - Lab02: 建立主要執行檔 app.js + Postman 測試
- 完成後的程式與檔案，請參考 Example 資料夾的內容

Estimated time:

40 minutes

7-31



## Module 7-1 – 解析 QueryString

使用querystring 中的parse、stringify、escape、unescape等方法，最終可完成一簡單的成績單功能。

Lab01: 安裝Node.js 環境

Lab02: 建立主要執行檔 app.js

Lab03: 測試stringify

## Module 7-2 – 使用 body-parser + Postman 測試

撰寫GET、POST 請求類型的API 各一個，當前端post JSON 資料時需要被body-parser 正確解析成物件。

最後用 Postman 來模擬前端的API呼叫。

Lab01: 安裝body-parser、Postman 環境

Lab02: 建立主要執行檔 app.js + Postman 測試

### 【Key Points】：

使用querystring 中的parse、stringify、escape、unescape等方法

撰寫GET、POST 請求類型的API 各一個

用 Postman 來模擬前端的API呼叫

# 範例程式使用說明

- 範例程式資料夾: Module\_07\_example
- 使用步驟:
  1. 安裝 Node.js
  2. 安裝 Visual Studio Code
  3. 以 Visual Studio Code 開啟本模組的範例資料夾的「7-1」或「7-2」
  4. 在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗。
  5. 在終端機視窗，輸入下列指令，安裝必要的模組套件:  
npm install querystring express body-parser
  6. 在終端機視窗，輸入 node app
  7. 若是「7-2」子資料夾，啟動瀏覽器，連接 <http://localhost:3000>

7-32



使用步驟:

1. 安裝 Node.js  
<https://nodejs.org/en/>
2. 安裝 Visual Studio Code  
<https://code.visualstudio.com/>
3. 以 Visual Studio Code 開啟範例資料夾內的「7-1」或「7-2」  
在檔案總管，滑鼠右鍵點按「本範例資料夾」，從快捷功能表點選「以Code開啟」或者，啟動 Visual Studio Code 之後，功能表 File | Open Folder，選擇資料夾
4. 在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗。
5. 在終端機視窗，輸入下列指令，安裝必要的模組套件:  
npm install querystring express body-parser
6. 在終端機視窗，輸入 node app.js
7. 若是「7-2」子資料夾，啟動瀏覽器，連接 <http://localhost:3000>

【Key Points】：

程式執行環境 Node.js

程式開發編輯環境: Visual Studio Code

在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗，輸入：「node 主程式.js」