



取得GET參數



designed by freepik

Estimated time:

50 min.

III 資訊工業策進會 Institute for Information Industry

【Key Points】：

學習目標

- 14-1: 使用 url 套件
- 14-2: 解析 QueryString
- 14-3: GET 參數和路由



14-1

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

我們即將一起學習：

- URL 網址格式
- 使用 `url.parse()` 解析網址
- 以 `url.format()` 建構網址
- 以 `url.resolve()` 組裝網址
- 關於 `QueryString` API
- 引用與測試 `parse()`
- URL Encode
- `escape()`、`unescape()`
- 使用網址參數
- 傳遞參數的範例
- Restful API 網址參數慣例
- Restful API 回傳的資料結構

【Key Points】：

- 14-1: 使用 url 套件
- 14-2: 解析 QueryString
- 14-3: GET 參數和路由

14-1：使用 url 套件

- URL 網址格式
- 使用 `url.parse()` 解析網址
- 以 `url.format()` 建構網址
- 以 `url.resolve()` 組裝網址



designed by freepik



designed by freepik

14-2

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

- 介紹 URL 網址的格式與規範
- `url.parse()` 會解析 URL 網址的全部內容。
- `url.parse()` 的第二個參數若為 `true`，`parse()` 會解析 Query String 並將之轉換成物件。
- `url.format()` 會根據它接收到的 `url` 物件來建構網址。
- `url.format()` 若省略部分設定，仍可建構出完整的網址。

【Key Points】：

使用 `url.parse()` 解析網址
以 `url.format()` 建構網址
以 `url.resolve()` 組裝網址

URL 網址格式

- 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

14-3



- 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

使用 url.parse() 解析網址

1. 先引用 url 模組:

```
const url = require("url");
```

2. 模擬某次請求的網址如下:

```
const address="http://localhost/search?q=apple&scope=all";
```

3. 呼叫 url.parse(address); 解析 address 網址:

```
const urlInfo = url.parse(address);
```

- Query String 的內容會放 query 屬性，型態為字串：

```
{..., query: 'q=apple&scope=all', ... }
```

- 呼叫 url.parse() 時，若是額外指定第二參數為 true 值，表示要將 Query String 的內容轉換成物件：

```
const urlInfo = url.parse(address, true); // 結果如下:
```

```
{..., query: { q: 'apple', scope: 'all' }, ... }
```

14-4



1. 先引用 url 模組: const url = require("url");

2. 模擬某次請求的網址如下:

```
const address="http://localhost/search?q=apple&scope=all";
```

3. 呼叫 url.parse(address); 解析 address 網址:

```
const urlInfo = url.parse(address);
```

- 解析的結果是物件，Query String 的內容會放在物件的 query 屬性，型態為字串：

```
{..., query: 'q=apple&scope=all', ... }
```

- 呼叫 url.parse() 時，若是額外指定第二參數為 true 值，表示要將 Query String 的內容轉換成物件：

```
const urlInfo = url.parse(address, true); // 結果如下:
```

```
{..., query: { q: 'apple', scope: 'all' }, ... }
```

【Key Points】：

先引用 url 模組: const url = require("url")

呼叫 url.parse(address); 解析 網址

呼叫 url.parse() 時，若是指定第二參數為 true 值，表示要將 Query String 的內容轉換成物件

以 url.format() 建構網址

- **url.format() 接受一個 url 物件為參數**
- **傳回組合好的網址字串**
- **省略部分設定，仍可建構出完整的網址**
- **如果沒有 query 屬性就不生成問號之後的 Query String**

```
const url=require('url');
const urlObject={
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'ecshweb.pchome.com.tw',
  port: null,
  hostname: 'ecshweb.pchome.com.tw',
  hash: null,
  search: '?q=musk&scope=all',
  query: { q: 'musk', scope: 'all' },
  pathname: '/search/v3.3/',
  path: '/search/v3.3/?q=musk&scope=all',
  href: 'https://ecshweb.pchome.com.tw/search/v3.3/?q=musk&scope=all'
}
console.log(url.format(urlObject));
//https://ecshweb.pchome.com.tw/search/v3.3/?q=musk&scope=all
```

14-5



- url.format() 接受一個 url 物件為參數。
- url.format() 根據 url 物件的內容，傳回組合好的網址字串
- 省略 slashes、auth、port、hash，url.format() 仍可完整建構出網址。
- 如果沒有 query 屬性，則網址字串便不會生成問號之後的Query String
- 這種設計模式是透過外部參數改變內部output，是一種好的設計，底層程式碼完全不需要更動。

【Key Points】：

url.format() 接受一個 url 物件為參數

url.format() 根據 url 物件的內容，傳回組合好的網址字串

省略部分設定，仍可建構出完整的網址

以 url.resolve() 組裝網址

- **url.resolve()** 的作用是將兩個參數的字串相連成一個新字串。
- **程式範例:**
const originUrl = "https://google.com";
url.resolve(originUrl, "me"); // 的結果:
`https://google.com/me`
- **路由(route) :**
url.resolve(originUrl, "/123/456"); // `https://google.com/123/456`
- **query string:**
url.resolve(originUrl, "?q=apple"); // `https://google.com/?q=apple`
- **標記片斷:**
url.resolve(originUrl, "#apple"); // `https://google.com/#apple`

14-6



- url.resolve() 的作用是將兩個參數的字串相連成一個新字串。
- 如果第二個參數前面沒有「/」，url.resolve() 會在回傳結果自動補上一個「/」。
const originUrl = "https://google.com";
url.resolve(originUrl, "me"); // 的結果:
`https://google.com/me`
- 路由(route)也可以透過 url.resolve() 字串相加:
url.resolve(originUrl, "/123/456"); // `https://google.com/123/456`
- query string也可以透過url.resolve()字串相加:
url.resolve(originUrl, "?q=apple"); // `https://google.com/?q=apple`
- 標記片斷也可以透過 url.resolve() 字串相加:
url.resolve(originUrl, "#apple"); // `https://google.com/#apple`

【Key Points】：

url.resolve() 的作用是將兩個參數的字串相連成一個新字串
如果第二個參數前面沒有「/」，url.resolve() 會在回傳結果自動補上「/」
query string 也可以透過url.resolve()字串相加

14-2: 解析 QueryString

- 關於 **QueryString API**
- 引用與測試 **parse()**
- **URL Encode**
- **escape() 、 unescape()**



designed by freepik



designed by freepik

14-7

 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 **QueryString** 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 **QueryString**？

<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

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>

14-8



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>
```

14-9



讓我們從頭開始創建一個 Node 專案。在桌面建立一個叫做 myapi 的資料夾作為專案名稱(可改為你想要的名字)。

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

常用 cmd 指令整理：

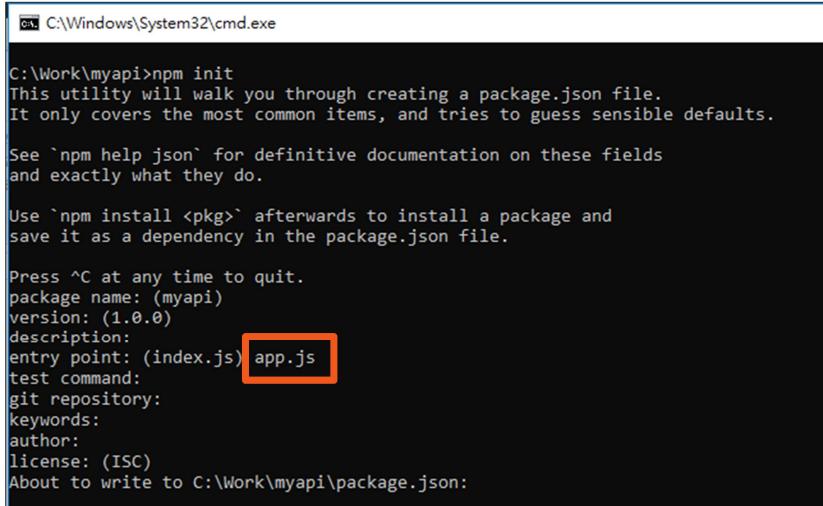
查詢目錄 (dir)、建立目錄 (md, mkdir)、變更目錄 (cd, chdir)、刪除目錄 (rd, rmdir)、檔案重新命名 (ren, rename)

【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 install** 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:
```

14-10



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**
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. 如果再有 **package.json**的專案目錄下執行 **npm install** 便會依照 **package.json** 的內容去佈署執行環境

【Key Points】：

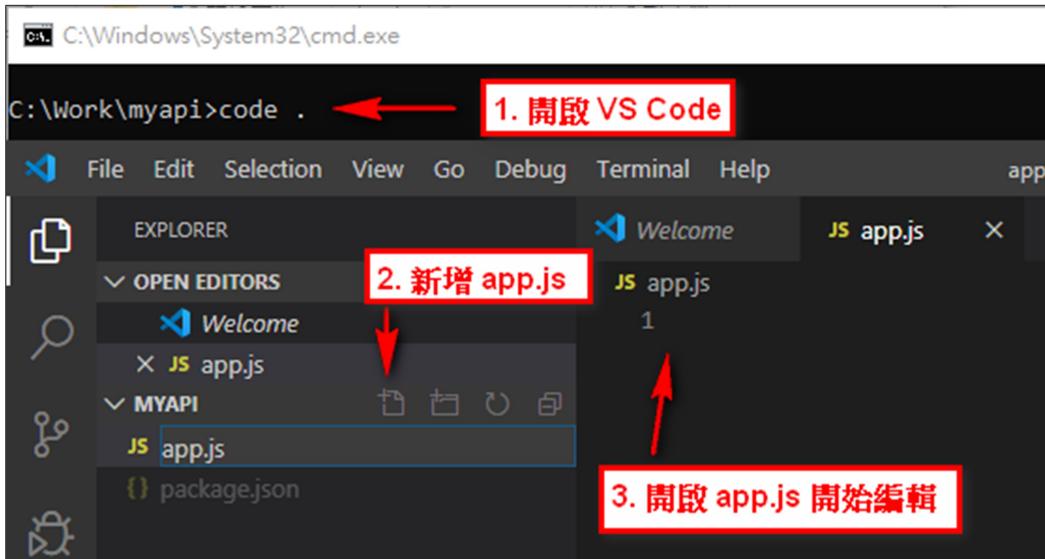
npm init

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



14-11

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

- 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' }
```

14-12



`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
```

14-13



- `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 **URLencode** 的形式

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

- 在 **URLencode** 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

14-14



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
```

14-15



- 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

14-3: GET 參數和路由

- 使用網址參數
- 傳遞參數的範例
- Restful API 網址參數慣例
- Restful API 回傳的資料結構



designed by freepik



designed by freepik

14-16

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

這個知識點，我們來研究一下路由的命名慣例與傳回資料的資料結構：

- /:id意思是取得指定編號的資料，常見的設計是直接指定資料庫的某一筆。
- 單複數route命名也會決定response的資料量
- query string可因應多變的查詢條件
- GET /products，路由使用到複數名稱，意味著有多個資料
- GET /product/:id，例如: /product/1 意味著只要「1」號這筆資料
- GET /products/?queryString，由於是依據條件做查詢，符合條件的筆數可能不只一筆

【Key Points】：

使用網址參數

Restful API 網址參數慣例

Restful API 回傳的資料結構

使用網址參數

- `http://localhost/hello/lin`
- `http://localhost/hello/:who`
- 定義參數時，在名稱前綴一個冒號「:」
- 以 `app.get("/hello/:who", callback);` 為例
若是瀏覽器 GET `http://localhost/hello/lin`，那麼，
後端的程式透過 `req.params.who` 讀到的值便是：「lin」
- 參數值的型態是 `string`

14-17



- 有別於query string，Request Parameter 需要在 route url 定義時，在名稱前綴一個冒號「:」。
- 例如，`app.get("/hello/:who", ...);` 宣告了`:who`，就代表這個路由有個叫 `who` 的參數 (parameter)。
- `app.get("/hello/:who", callback);` callback 函式的第一個參數 `req` 是一個物件，`req` 物件有個叫 `params` 的屬性，所有路由參數，都登記在 `request.params` 名下。
- 以 `app.get("/hello/:who", callback);` 為例，若是瀏覽器 GET `http://localhost/hello/lin`，那麼，
後端的程式透過 `req.params.who` 讀到的值便是：「lin」。
- 跟query string一樣，`req.params.`參數名稱，參數值的型態也是 `string`。

【Key Points】：

在名稱前綴一個冒號「:」

所有路由參數，都登記在 `request.params` 名下

以 `app.get("/hello/:who", callback);` 為例，過 `req.params.who` 讀取參數值

傳遞參數的範例

- **GET query string 範例**

```
const express = require('express');
const router = express.Router();

/* GET home page. */

router.get('/query', function(req, res, next) {
  console.log(req.query); // { id: '123' }
  console.log(typeof req.query.id); // string
  res.render('index', { title: 'Query String' });
});
```

- **GET parameter 範例**

```
router.get('/param/:id', function(req, res, next) {
  console.log(req.params); // { id: '123' }
  console.log(typeof req.params.id); // string
  res.render('index', { title: 'Parameter' });
});
```

- **GET query string 和 parameter 混用範例**

```
router.get('/query_param/:id', function(req, res, next) {
  console.log(req.params); // { id: '123' }
  console.log(req.query); // { id: '123' }
  res.render('index', { title: 'Query string and Parameter' });
});
```

```
module.exports = router;
```

14-18



財團法人資訊工業策進會

INSTITUTE FOR INFORMATION INDUSTRY

- 使用 GET 傳遞參數的三個範例。
- 第一個是GET query string 範例。
- 第二個是GET parameter 範例。
- 第三個是混用 GET query string 和parameter 範例。
- 提醒: parameter一定要記得加上冒號「:」。

【Key Points】：

- 第一個是GET query string 範例。
- 第二個是GET parameter 範例。
- 第三個是混用 GET query string 和parameter 範例。

Restful API 網址參數慣例

- **GET /products**
 - 取得產品清單
 - 取得全部的產品資料
- **GET /product/:id**
 - 取得特定編號的產品
 - 取得一筆產品資料
- **GET /products?q=shoes**
 - 透過條件取得符合條件的產品清單
 - query string可因應多變的查詢條件

14-19



- GET 在Restful API是蠻常使用的HTTP method。
- 單複數route命名也會決定response的資料量。
- /:id意思是取得指定編號的資料，常見的設計是直接指定資料庫的某一筆。
- query string常見的應用情境是，前端可能會以各種條件做查詢，後端只需要提供一支查詢的Restful API就可以滿足前端所有查詢需求。
- query string與parameter可以一起使用。

【Key Points】：

/:id意思是取得指定編號的資料，常見的設計是直接指定資料庫的某一筆。

單複數route命名也會決定response的資料量

query string可因應多變的查詢條件

Restful API 回傳的資料結構

- GET /products

```
{ "total": 2, list: [ {"name": "NIKE"}, {"name": "Apple"} ]}
```

- GET /products/:id

/products/1

```
{ "name": "NIKE" }
```

- GET /products?name=NIKE

```
{ "total":1, list:[ {"name": "NIKE"} ] }
```

14-20



- GET /products，路由使用到複數名稱，意味著有多個資料，所以回傳內容有 total 與 list。
- GET /product/:id，例如: /product/1 意味著只要「1」號這筆資料，所以直接回傳 JSON Object。
- GET /products/?queryString，由於是依據條件做查詢，符合條件的筆數可能不只一筆，所以回傳 total 與 list。
- 嚴謹一點的設計，會在傳回的資料也多傳一份唯一請求編號(unique requestId)，作為該次回傳的一個編號。
- Restful API已經流行好一陣子，另外，推薦另一個作法：GraphQL，能讓前端有更大的彈性對後端進行請求。

【Key Points】：

GET /products，路由使用到複數名稱，意味著有多個資料

GET /product/:id，例如: /product/1 意味著只要「1」號這筆資料

GET /products/?queryString，由於是依據條件做查詢，符合條件的筆數可能不只一筆

Summary 〈 精華回顧 〉

- 了解URL網址規範
- 了解url套件使用方式
- 了解如何讀取 Query String
- 了解如何使用網址參數。
- 了解 Restful API 中的 GET 設計



14-21

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

- URL網址規範包括: protocol: 請求的protocol、host:主機位置(包含port)、port:程式監聽的端口。
- 呼叫 url.parse(address); 解析 address 網址
- 以 querystring.parse() 解析 Query String 成物件
- 以 app.get("/hello/:who", callback); 為例，若是瀏覽器 GET http://localhost/hello/lin，那麼，後端的程式透過 req.params.who 讀到的值便是: 「lin」
- GET /products，路由使用到複數名稱，意味著有多個資料，所以回傳 total 與 list。

【Key Points】：

解析 URL 網址

解析 Query String

透過 req.params 讀取網址參數

線上程式題

- **14-1 url.format() 的參數格式**
請修改程式碼，使其能夠正常console.log()出query string。
- **14-2 如何讀取 query string？**
請修改程式碼，使其能夠 console.log() 出 query string 與 parameter。
- **14-3 取得單一筆資料的Restful API**
想要寫一支取得單一筆資料的API。請修改以下程式碼，使其符合Restful API。
 - 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。

14-22



14-1 url.format() 的參數格式

請修改程式碼，使其能夠正常console.log()出query string。

14-2 如何讀取 query string？

請修改程式碼，使其能夠 console.log() 出 query string 與 parameter。

14-3 取得單一筆資料的Restful API

想要寫一支取得單一筆資料的API。請修改以下程式碼，使其符合Restful API。

14-1 url.format() 的參數格式

14-2 如何讀取 query string？

14-3 取得單一筆資料的Restful API

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】：

14-1 url.format() 的參數格式

14-2 如何讀取 query string？

14-3 取得單一筆資料的Restful API

課後練習題(Lab)

- **情節描述:**
網站已十分普及，後端API也朝向Restful API發展了數年，本次會注重在Restful API的GET練習與後端轉址，並且以電商平台產品查詢為例。
- **預設目標:**
 - 理解後端轉址
 - 理解GET query string
 - 理解GET API回傳設計
- **Lab01:理解後端轉址**
- **Lab02:理解GET query string**
- **Lab03:定義傳回值的資料結構**

Estimated time:
20 minutes

14-23



【情節描述】

網站已十分普及，後端API也朝向Restful API發展了數年，本次會注重在Restful API的GET練習與後端轉址，並且以電商平台產品查詢為例。

【預設目標】

理解後端轉址

理解GET query string

理解GET API回傳設計

Lab01:理解後端轉址

Lab02:理解GET query string

Lab03:定義傳回值的資料結構

完成後的程式與檔案，請參考 Example 資料夾的內容。

【Key Points】：

Lab01:理解後端轉址

Lab02:理解GET query string

Lab03:定義傳回值的資料結構

範例程式使用說明

- 範例程式資料夾: Module_14_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
 4. 在 Visual Studio Code 按下「Ctrl + 滑鼠」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
 6. 在終端機視窗，輸入 npm start
 7. 啟動瀏覽器，連接 http://localhost:3001

14-24



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
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「本範例資料夾」
4. 在 Visual Studio Code 按下「Ctrl + 滑鼠」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
6. 在終端機視窗，輸入 npm start
7. 啟動瀏覽器，連接 http://localhost:3001

【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」開啟 terminal 終端機視窗，輸入：「npm start」