

**Front-End Engineer**

**前 端 工 程 師 養 成 班**

[www.flycan.com.tw](http://www.flycan.com.tw)

Steve Chang

# 目錄

講師介紹 .....	5
認識前端工程師 F2E .....	6
從網頁設計師到前端工程師 .....	6
網頁設計師的工作 .....	6
前端工程師的工作 .....	6
前端工程師與網頁工程師差異 .....	6
前端工程師 Roadmap .....	7
如何成為更有價值的前端工程師 .....	7
課前準備 .....	8
下載範例檔案 .....	8
安裝 Node.js .....	8
使用 Visual Studio Code 編輯程式碼 .....	8
除錯技巧 .....	10
為何需要除錯 .....	10
介紹開發人員工具 .....	10
HTML 與 CSS 元素編輯 .....	10
Console 主控台使用 .....	11
網路請求監控 .....	11
本地端數據監控 .....	12
第一章: 常用的資料結構 .....	13
Array 介紹 .....	13
Array 資料存取方式 .....	13
Array 篩選資料應用 .....	15
Array 排序資料應用 .....	16
Array 分頁資料應用 .....	17
Array 實際應用範例練習 .....	18
JSON 介紹 .....	19
JSON 資料存取方式練習 .....	21
JSON 輪播應用 .....	22
JSON 輪播相簿加上文字說明 .....	22
JSON 遞迴應用 .....	24
ES6,7,8 簡介 .....	26
Async / Await 介紹 .....	26
箭頭函式介紹 .....	27
let, const 介紹 .....	28
String template 介紹 .....	28

排序演算法介紹 .....	29
第二章: <b>AJAX 技術應用</b> .....	32
<b>AJAX</b> 簡介 .....	32
內容區塊 <b>AJAX</b> 動態載入 .....	37
註冊事件與事件綁定 .....	38
多個 <b>AJAX</b> 同時執行 .....	40
<b>AJAX</b> 下載進度條 .....	41
<b>AJAX</b> 製作自動載入的新聞頁面 .....	43
第三章:    外部 <b>API</b> 使用 .....	46
<b>API</b> 介紹.....	46
<b>RESTful API</b> .....	46
如何存取與使用 <b>API</b> .....	47
<b>Mock server</b> .....	47
會員註冊檢查帳號.....	48
跨域請求概念介紹.....	48
<b>JSONP</b> 介紹 .....	49
<b>CORS</b> 介紹.....	49
<b>Google map API</b> 使用 .....	52
利用 <b>Polling</b> 模擬聊天室畫面 .....	53
獲得即時更新的方法 ( <b>polling</b> 、 <b>comet</b> 、 <b>long polling</b> 、 <b>websocket</b> ) .....	54
第四章:    常用的功能介紹 .....	55
正規表達式.....	55
利用正規表達式取代網頁內容 .....	57
利用正規表達式檢查帳號格式 .....	57
利用 <b>html form</b> 配合正規表達式檢查用戶輸入 .....	57
<b>Cookie</b> 與 <b>Local Storage</b> .....	58
利用 <b>Cookie</b> 紀錄使用者登入資訊 .....	58
利用 <b>Local Storage</b> 紀錄使用者的儲存資訊.....	59
第五章:    開放原始碼專案使用 .....	61
<b>Github</b> 介紹 .....	61
如何找到需要的工具 .....	61
<b>yarn</b> 介紹及安裝 .....	63
利用 <b>yarn</b> 及開放原始碼快速開發網頁.....	63
<b>JS-cookie</b> 使用 .....	63
<b>Moment.js</b> 使用 .....	64
<b>lodash</b> 使用 .....	66
第六章:    高品質程式碼撰寫.....	67
運算式==和===差異.....	67

三元運算子.....	68
位元運算子.....	68
避免使用全域變數.....	69
函式內的程式碼長度.....	69
函式內的程式碼(圈)複雜度 .....	70
函式的變數個數 .....	71
程式碼打包壓縮工具 .....	71
程式碼檢查工具 .....	72
實用工具.....	73

## 講師介紹

姓名: **Steve**

簡介: 具有 **Garena, Yahoo** 等外商工作經驗，崇尚敏捷式開發流程，閒暇時間經營自己的軟體工作室，為客戶開發客製化軟體或公司的教育訓練。

專長: 軟體規劃開發、伺服器架設。

講師 FB: <https://www.facebook.com/zhang.jun.94>

FB 討論社團: <https://www.facebook.com/groups/flycan.webdesign/>

## 認識前端工程師 F2E

- 前端工程師 **Front-End Engineer** ( 簡稱 **F2E** )
- 開發技術主要包含 **HTML, CSS, JavaScript**
- 企業間越來越被受到重視
- 網頁設計師與後端工程師之間的溝通橋樑
- 著重於使用者體驗

## 從網頁設計師到前端工程師

### 網頁設計師的工作

- 與 **PM** 討論好網頁整體的呈現
- 使用繪圖軟體設計網站視覺
- 切版產出初步的 **HTML** 及 **CSS** (依照公司不同，非常態。)

### 前端工程師的工作

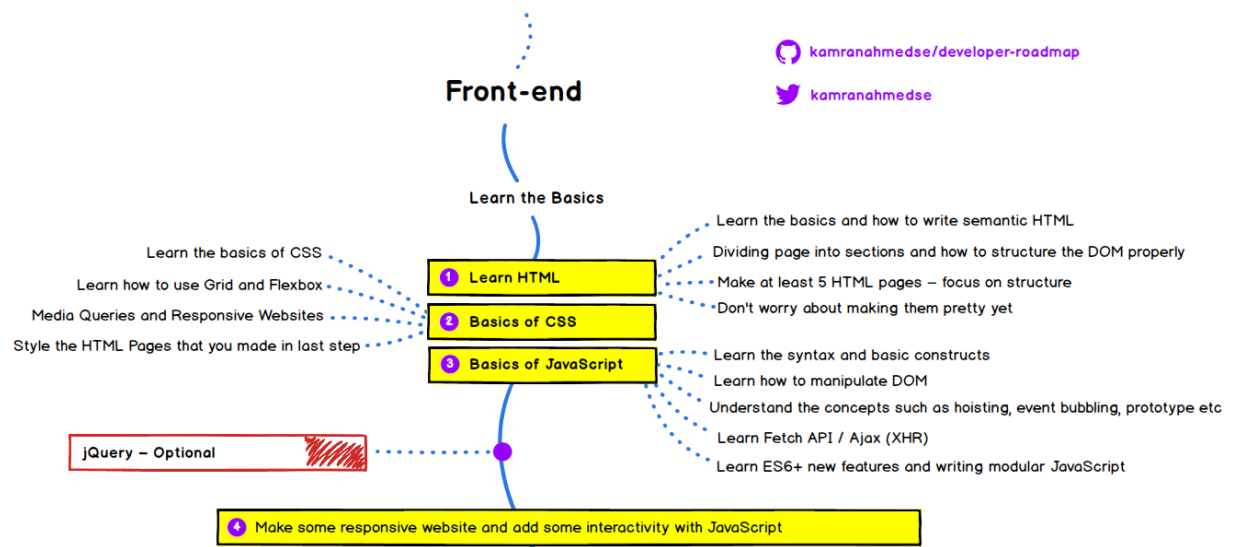
- 接收網頁設計師的 **HTML** 檔案或圖檔進一步完成網頁
- 製作網頁互動效果
- 與後端工程師進行資料串接
- 初步的安全性驗證

### 前端工程師與網頁工程師差異

- 不太會去修改視覺
- 必須更熟悉 **HTML**、**CSS** 以及整個網頁渲染的過程
- 熟悉 **JavaScript**，能使用修改套件或是自己刻功能達到想要的效果
- 基礎資安知識
- 基礎網路通訊知識

## 前端工程師 Roadmap

ref: <https://github.com/kamranahmedse/developer-roadmap>



## 如何成為更有價值的前端工程師

- 版本管理工具 (SVN, GIT)
- 測試概念 (TDD, BDD, Unit test, smoke test ...etc.)
- 測試工具 (Mocha, Selenium ...etc.)
- 學會寫更好維護的 code (低耦合、益擴展、可讀性佳 ...etc.)
- 熟悉 socket 使用
- 學會 Server side rendering (SSR)
- 建構 Single page application 網頁 (SPA)
- 學會建立 AMP, PWA....etc
- 學會寫出有結構高效的程式碼
- .....
- 保持學習



## 課前準備

### 下載範例檔案

ref: <https://github.com/stevekevin1005/flycan-f2e>

### 安裝 Node.js

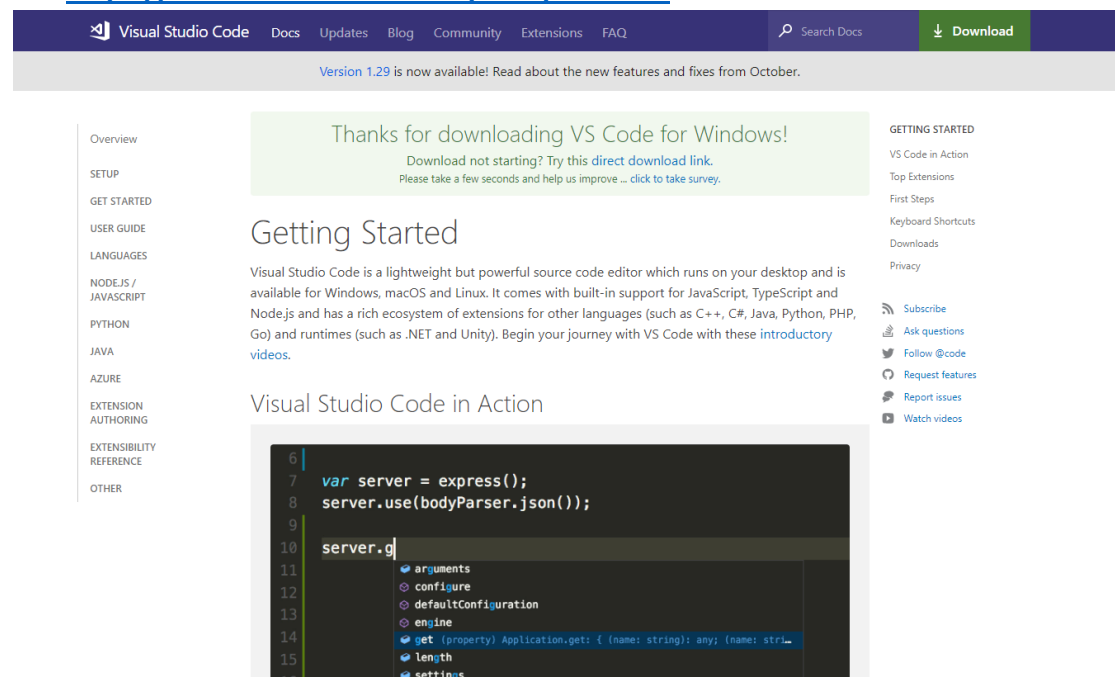
**Node.js** 是一個能夠在伺服器端運行 **JavaScript** 的開放原始碼、跨平台 **JavaScript** 執行環境。採用 **Google** 開發的 **V8** 引擎執行程式碼，使用事件驅動、非阻塞和 非同步輸入輸出模型等技術來提高效能，可優化應用程式的傳輸量和規模。

ref: <https://nodejs.org/en/>

### 使用 Visual Studio Code 編輯程式碼

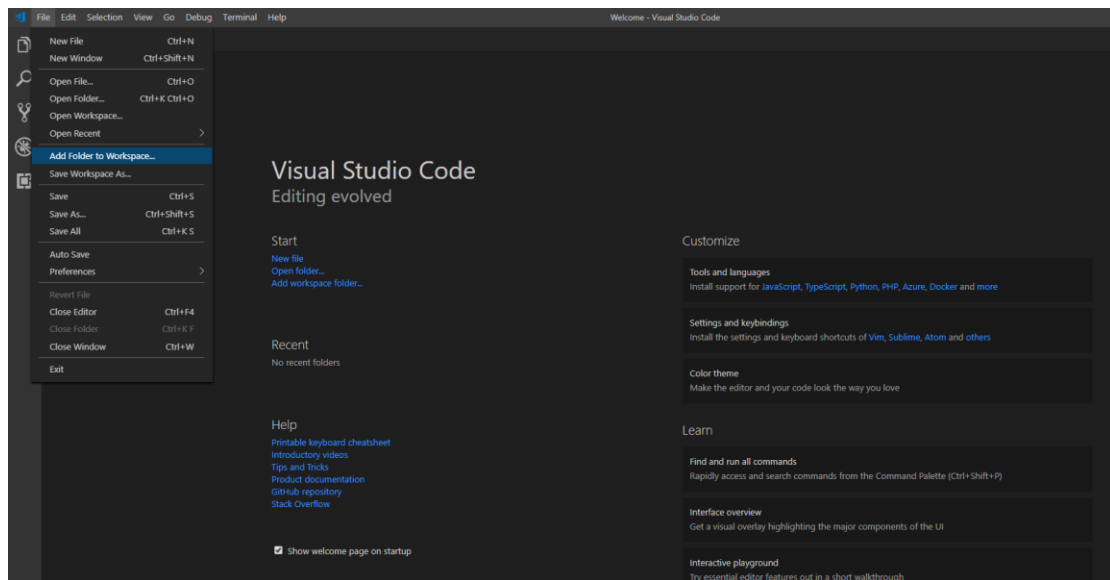
**Visual Studio Code** 是由微軟維護並開源的編輯器，同時支援 **OSX** 及 **WINDOW** 雙平台，本身整合進了 **terminal** 以及 **git** 幫助開發，並有許多擴充套件可以使用，若沒有熟悉的編輯器，推薦同學們可以使用看看。

ref: <https://code.visualstudio.com/docs/?dv=win>





與大多編輯器相同，點選 **FILE -> Add Folder to Workspace** 將專案加入。



VS Code 上也有許多好用的擴充功能，可以幫助工程師進行開發，於右邊的選單點選 **Extensions** 搜尋想加入的擴充功能。



推薦幾個 Extensions 可以使用

- **Bracket Pair Colorize** 將相同的括號上色，幫助使用者分辨括號。

```
1 // forceUniqueOpeningColor: false
2 ((( )))
3 ))(( ))
```

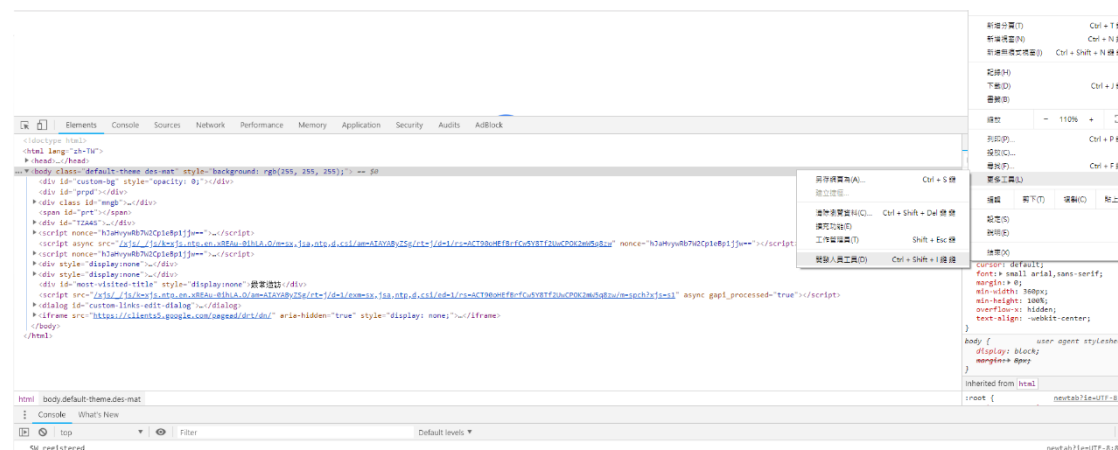
- **HTML Boilerplate** 快速打好 Html5 骨架的程式碼
- **Live Server** 幫助前端開發者快速啟動伺服器
- **Path Intellisense** 檔案路徑自動完成
- **Prettier** 自動排版程式碼
- **lit-html** 在 JavaScript/TypeScript 的文件中，如果有使用到 HTML 語法，提供上色和相應的提示。
- **Trailing Spaces** 將多餘的空格上色顯示

## 除錯技巧

## 為何需要除錯

當開發過程遇到了瓶頸或找不出哪裡有問題時，就需要利用開發人員工具進行除錯解決問題。目前主流瀏覽器都有開發人員工具(IE8 以上都有開發人員工具)，基本操作上幾乎大同小異，本課程將以 **Chrome** 開發人員工具示範。

開啟 Chrome 後按下 **F12**( **Ctrl + Shift + I** )，或點選「自訂及管理 Google Chrome」→「更多工具」→「開發人員工具」，即可叫出開發人員工具。



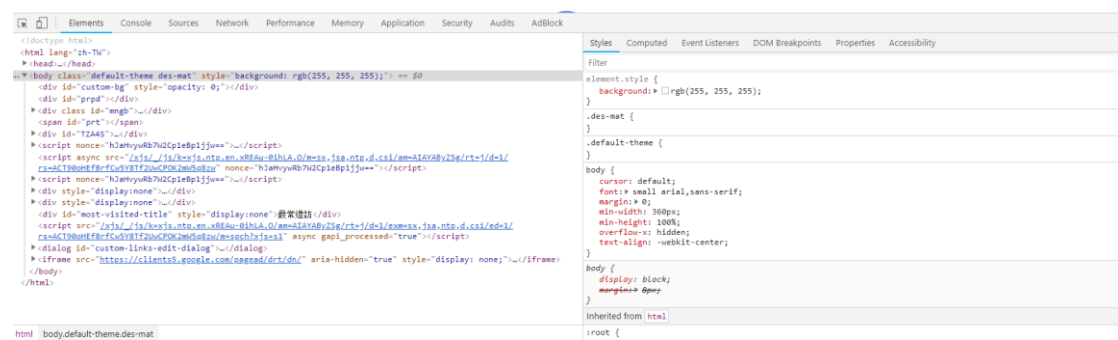
## 介紹開發人員工具

比較常用到的工具會有：

- **Elements:** DOM 元素，可以在上面直接更改 DOM 元素。
- **Console:** 主控台，可以看程式碼所下的 log 或是直接打 code 執行。
- **Network:** 觀察網路發出請求的狀況。
- **Application:** 看 Storage 跟 Cache 裡的資訊。

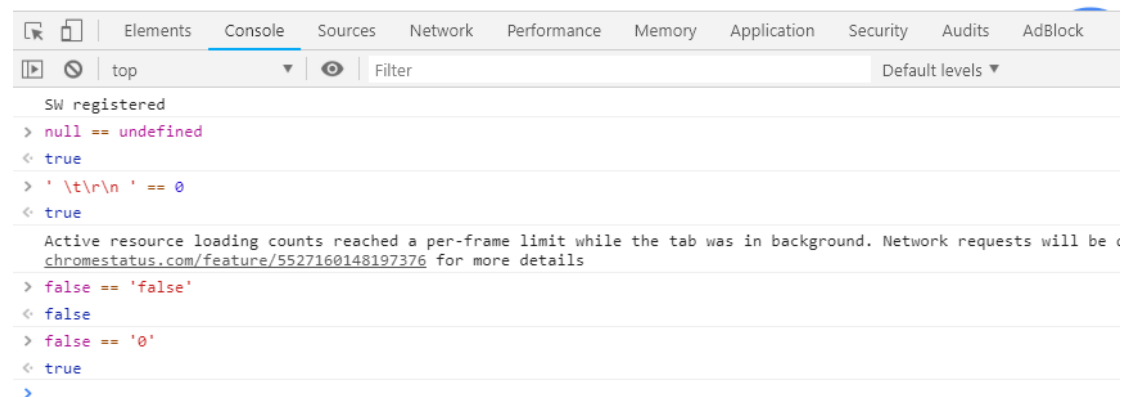
## HTML 與 CSS 元素編輯

可以直接點選裡面的 **DOM** 結構，變更成你想要的結構，畫面會按照你所變更的結構重繪一次。也可能看右邊的 **Style** 看該結構的 **CSS** 是哪些，可以直接在上面調顏色，或是看是哪個 **CSS** 檔載入的。



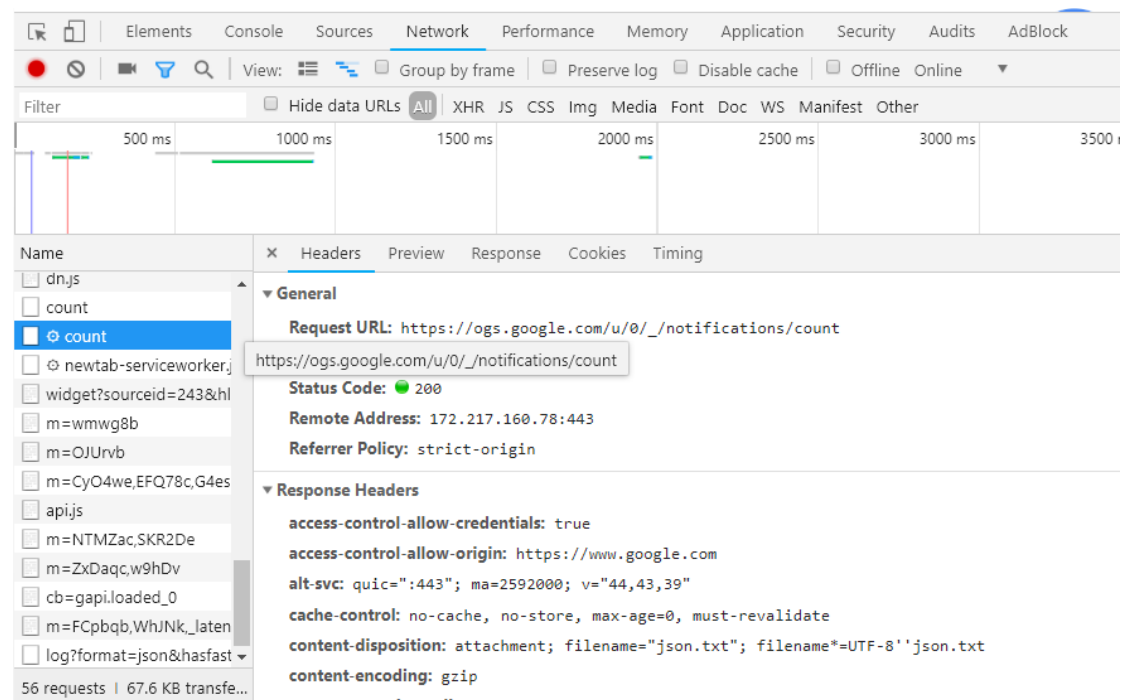
## Console 主控台使用

可以在 **Console** 中看到你在程式裡有下 `console.log` 等的資訊，或是直接在 **console** 打一些 `javascript code` 直接執行看結果。



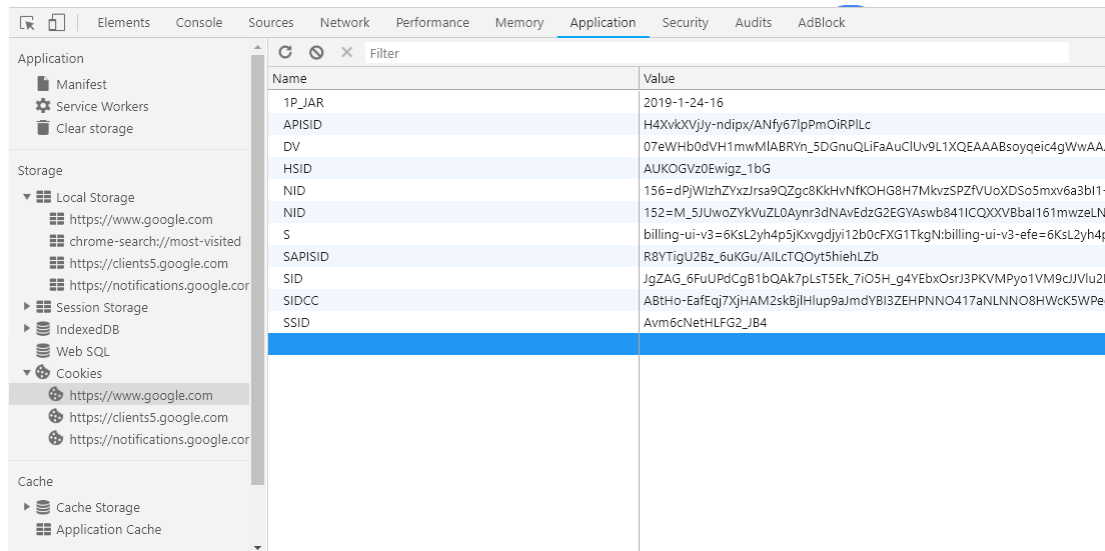
## 網路請求監控

用 **Network** 可以看到瀏覽器現在對那些地方發了請求，這些請求的 **Header** 有哪些資訊，得到的回復有哪些資訊，請求花了多久等等。



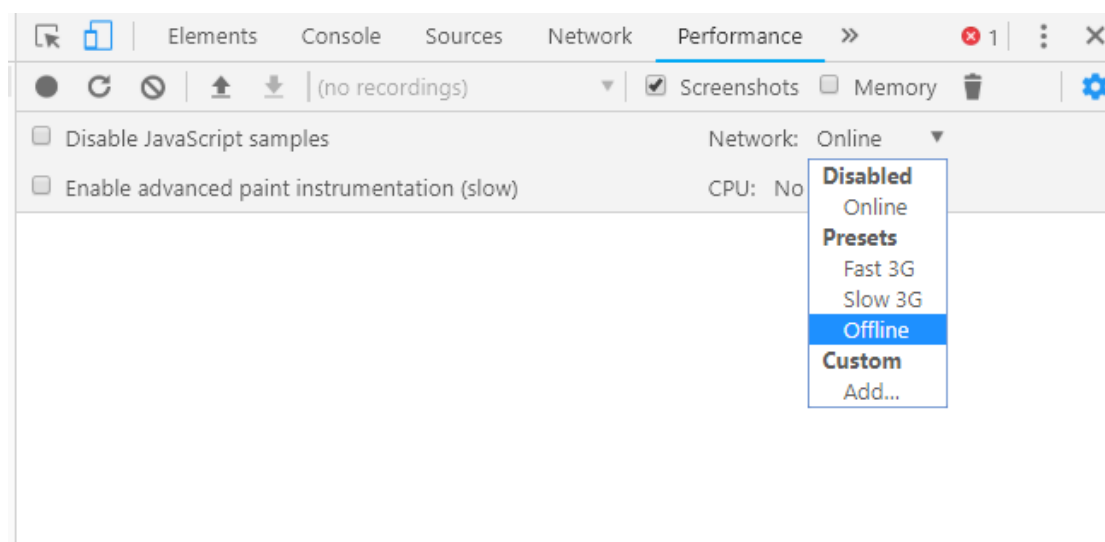
## 本地端數據監控

在 **Application** 可以看到本地端的瀏覽器有存了哪些 **cookie** 或 **local storage** 等資料，也可以直接去更改裡面的值。



## 行動裝置模擬

也可以去模擬是用什麼樣子的手機來瀏覽這個網頁，以及現在的網速是不是處在低網速環境。



## 第一章： 常用的資料結構

### Array 介紹

**Array** 是一種 Javascript 常用的資料結構，讓工程師使用一個變數來儲存複數個變數。

### 沒用 Array

```
1 var student1 = "Kevin";
2 var student2 = "Ken";
3 var student3 = "Amy";
4 var student4 = "Lynn";
```

### 用 Array

```
var students = ["Kevin", "Ken", "Amy", "Lynn"];
```

### Array 資料存取方式

工程師們可以使用索引值來取得 **Array** 內的值，而在 Javascript 裡面索引是從 0 開始計算。

```
students[1] //Ken
```

### 範例 1 – 1

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <title>Example 1 - 1</title>
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <script src="main.js"></script>
9 </head>
10 <body>
11   <h1>Array 存取練習</h1>
12   <h4>Array: students = ["Kevin", "Ken", "Amy", "Lynn"]</h4>
13   選取的學生: <span id="name"></span>
14   <button onclick="chooseStuent(0)" Kevin</button>
15   <button onclick="chooseStuent(1)" Ken</button>
16   <button onclick="chooseStuent(2)" Amy</button>
17   <button onclick="chooseStuent(3)" Lynn</button>
18 </body>
19 </html>
```

```
var students = ["Kevin", "Ken", "Amy", "Lynn"];

function chooseStuent(index) {
    document.getElementById("name").innerText = students[index];
}
```

工程師可以直接透過索引更改 **Array** 內的數值，透過 **Array** 內建的 **push** 函式在 **Array** 的尾端增加一個數值，透過 **Array** 內建的 **pop** 函式移除尾端的數值

## 範例 1 – 2

```
window.onload = function() {

    var students = ["Kevin", "Ken", "Amy", "Lynn"];

    document.getElementById("addStduent").onclick = addStduent;
    document.getElementById("removeStudent").onclick = removeStudent;
    document.getElementById("changeFirstStudent").onclick = changeFirstStudent;

    renderHtml();

    function renderHtml() {
        document.getElementById("demo").innerHTML = students + "<br>total length:  " + students.length;
    }

    function addStduent() {
        var student = document.getElementById("value").value;
        if(student != null) {
            students.push(student);
            document.getElementById("value").value = null;
            renderHtml();
        }
    }

    function changeFirstStudent() {
        var student = document.getElementById("value").value;
        if(student != null) {
            students[0] = student;
            document.getElementById("value").value = null;
            renderHtml();
        }
    }

    function removeStudent() {
        students.pop();
        renderHtml();
    }
}
```

## Array 篩選資料應用

JavaScript 內提供了許多函式來幫助工程師篩選 Array 內的資料，利用這些函式可以增加程式碼的開發速度以及可維護性。

- **filter** 回傳一個陣列，其條件是 **return** 後方為 **true** 的物件，很適合用在搜尋符合條件的資料。
- **find** 與 **filter** 很像，但 **find** 只會回傳一次值，且是第一次為 **true** 的值。
- **map** 使用 **map** 時他需要回傳一個值，他會透過函式內所回傳的值組合成一個陣列，這很適合將原始的變數運算後重新組合成一個新的陣列。
  - 如果不回傳則是 **undefined**
  - 回傳數量等於原始陣列的長度
- **every** 可以檢查所有的陣列是否符合條件，僅會回傳 **true** 或 **false**，可以用來檢查陣列中的內容是否符合特定條件。
- **some** 與 **every** 非常接近，也是回傳 **true** 或 **false**，差異在 **every** 必需完全符合，**some** 只需要部分符合。

### 範例 1-3

```
function filterArray(condition) {  
  switch(condition) {  
    case "all":  
      renderHtml(students);  
      break;  
  
    case "adults":  
      renderHtml(students.filter(function(student, index) {  
        return student.age >= 18;  
      }));  
      break;  
  
    case "children":  
      renderHtml(students.filter(function(student, index) {  
        return student.age < 18;  
      }));  
      break;  
  }  
}
```

```

function showAges() {
    renderHtml(students.map(function(student, index) {
        return student.age;
    }));
}

function findChild() {
    renderHtml(students.find(function(student, index) {
        return student.age < 18;
    }));
}

function everyOneIsAdult() {
    renderHtml(students.every(function(student, index) {
        return student.age >= 18;
    }));
}

function someOneIsAdult() {
    renderHtml(students.some(function(student, index) {
        return student.age >= 18;
    }));
}

```

## Array 排序資料應用

當需要對 Array 做排序的時候，Javascript 內提供了 `sort` 這個函式幫助工程師排序內部資料。

### 範例 1-4

```

window.onload = function() {
    var students = [
        {
            name: "Kevin",
            age: 15
        },
        {
            name: "Ken",
            age: 22
        },
        {
            name: "Amy",
            age: 33
        },
        {
            name: "Lynn",
            age: 12
        }
    ];

    document.getElementById("sortByAge").onclick = sortByAge;

    renderHtml(students);

    function renderHtml(arr) {
        document.getElementById("demo").innerHTML = JSON.stringify(arr);
    }

    function sortByAge() {
        renderHtml(students.sort(function(a, b) {
            return a.age - b.age;
        }));
    }
}

```



可以使用 **sort** 函式做簡單的隨機數應用。(注意! 如果要對較大的群體做更公正的隨機，因去找尋適合的隨機數演算法實作。)

#### 範例 1 – 5

```
1 window.onload = function() {  
2  
3     var numbers = [1,4,34,23,4,3,432,4,32,4,234,32,432,4,32];  
4  
5     document.getElementById("randomArray").onclick = randomArray.bind(null,numbers);  
6  
7     renderHtml(numbers);  
8  
9     function renderHtml(arr) {  
10         document.getElementById("demo").innerHTML = JSON.stringify(arr);  
11     }  
12  
13     function randomArray(arr) {  
14         renderHtml(arr.sort(function(a, b){return 0.5 - Math.random()}));  
15     }  
16 }
```

#### Array 分頁資料應用

**Array** 提供了一個 **slice([begin[, end]])** 函式可以分割陣列，該函式會回傳一個新陣列物件，為原陣列選擇之 **begin** 至 **end**（不含 **end**）部分的淺拷貝，原本的陣列不會被修改。

#### 範例 1 – 6

```
function showPage() {  
    var page = this.dataset.page;  
    renderHtml(pagination(page, PAGE_SIZE, students));  
}  
  
function pagination(pageNo, pageSize, array) {  
    var offset = (pageNo - 1) * pageSize;  
    return (offset + pageSize >= array.length) ? array.slice(offset, array.length) : array.slice(offset, offset + pageSize);  
}
```

## Array 實際應用範例練習

學會基本 Array 的操作後，請同學們練習，當透過一個 API 得到資料後，如何利用 JS 產出一個基本的分頁畫面。

### 範例 1 – 7

```
{
  "class": "F2e class",
  "students": [{
    "name": "Amy",
    "age": 12,
    "email": "test1@gamil.com",
    "fee": 8000
  }, {
    "name": "Althea",
    "age": 23,
    "email": "test2@gamil.com",
    "fee": 12000
  }, {
    "name": "Wallis",
    "age": 14,
    "email": "test3@gamil.com",
    "fee": 12000
  }, {
    "name": "Valerie",
    "age": 8,
    "email": "test4@gamil.com",
    "fee": 8000
  }, {
    "name": "Tracy",
    "age": 12,
    "email": "test5@gamil.com",
    "fee": 8000
  }
]}
```

## F2e class

#	Name	Email	Age	Fee
1	Amy	test1@gamil.com	12	8000
2	Althea	test2@gamil.com	23	12000
3	Wallis	test3@gamil.com	14	12000
4	Valerie	test4@gamil.com	8	8000
5	Tracy	test5@gamil.com	12	8000

1 2 3 4

## JSON 介紹

**JavaScript Object Notation (JSON)** 資料格式讓應用程式 (通常為 RESTful API) 進行網路通訊。JSON 無關技術、專利，幾乎所有常用到的語言都有支援相關的模組(Java、PHP、Python、Groovy、JavaScript、Go)。

一個合法的 JSON 文件可以為

- {} 以大括弧包圍的物件
- [] 以中括弧包圍的陣列

<https://jsonlint.com/>

是一個簡單的網頁可以很簡單的測試該文件是否符合 JSON 定義。

為何使用 JSON?

- 可讀性高
- 方便撰寫 (與過去使用的主要格式 XML 相比)
- 輕量

## JSON 與 XML 比較

### XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

### JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

JSON 有下列核心資料型別

- 鑑 / 值對 (key/value)
- 物件(object)
- 陣列(Array)

JSON 的值型別

- 物件
- 陣列
- 字串
- 數字
- 布林
- 空

字串具有以下屬性

- 由零或更多的 **Unicode** 字元包含在雙引號中組成
- 由單引號包圍的字串不合法
- 加上反斜線可以組成一些跳脫字元
  - `\"` 雙引號
  - `\\` 反斜線
  - `\` 斜線
  - `\b` 後退
  - `\f` 換頁
  - `\r` 回車
  - `\t` Tab
  - `\u` 後面十六進位數字

數字具有以下屬性

- 數字為十進位且無前綴零
- 數字可帶 10 指數，以 e 或 E 記號法與加減法表示正負指數 (e.g. `1.2e+2`)
- 不能為 NaN 或 Infinity

撰寫習慣(非強制規定)

- 大多採用駝峰式 (lowerCamelCase)
- 日期偏好採用 RFC 3339 格式 (e.g. `2020-12-12T22:10:12+08:00`)
- 經緯值(±DD.DDDD±DD.DDDD)一班緯度在前，赤道北為正，東(本初子午線)為正 (e.g. `"40.7842-30.5742"`)

## JS 內常用的兩個 JSON 相關函式

- `parse` :字串轉為物件
- `stringify` :物件轉為字串

## JSON 資料存取方式練習

JSON 存取方式跟基本的物件一樣，可以透過`.`加上鑑的值，或是`[]`裡面加上鑑的值來存取

### 範例 1 – 8

#### 原始 JSON 檔案

```
{
  squadName: "Super hero squad",
  homeTown: "Metro City",
  formed: 2016,
  secretBase: "Super tower",
  active: true,
  - members: [
    - {
      name: "Molecule Man",
      age: 29,
      secretIdentity: "Dan Jukes",
      - powers: [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    - {
      name: "Madame UpperCut",
      age: 39,
      secretIdentity: "Jane Wilson",
      - powers: [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    - {
      name: "Eternal Flame",
      age: 1000000,
      secretIdentity: "Unknown",
      - powers: [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

取出相關資訊做出標題敘述，並以 **members** 的陣列做出英雄們的相關介紹。

# Super hero squad

Hometown: Metro City // Formed: 2016

## Molecule Man

Secret identity: Dan Jukes

Age: 29

Superpowers:

- Radiation resistance
- Turning tiny
- Radiation blast

## Madame Uppercut

Secret identity: Jane Wilson

Age: 39

Superpowers:

- Million tonne punch
- Damage resistance
- Superhuman reflexes

## Eternal Flame

Secret identity: Unknown

Age: 1000000

Superpowers:

- Immortality
- Heat Immunity
- Inferno
- Teleportation
- Interdimensional travel

## JSON 輪播應用

利用 **JSON** 做一個簡單的輪播照片應用

### 範例 1 – 9

```
var i = 0;
function run() {
  if (i >= photos.length) {
    i = 0;
  }
  document.getElementById("image").setAttribute('src', photos[i]);
  i++;
}
setInterval(run, 3000);
run();
```

**setInterval(function, milliseconds, param1, param2, ...):** 延遲所設的時間後，執行該函式(不斷執行)。

**setTimeout(function, milliseconds, param1, param2, ...):** 延遲所設的時間後，執行該函式(執行一次)。

## JSON 輪播相簿加上文字說明

利用 **JSON** 做一個簡單的輪播照片應用，加上相片的文字說明以及 **fade out** 的效果。

### 範例 1 – 10

```
var i = 0;
function run() {
    if (i >= photos.length) {
        i = 0;
    }
    var img = document.getElementById('image');
    var information = document.getElementById('information');
    img.setAttribute('src', photos[i].src);
    information.innerText = photos[i].info;
    var fadeOut = setInterval(function () {
        if (!img.style.opacity) {
            img.style.opacity = 1;
        }
        if (img.style.opacity > 0) {
            img.style.opacity -= 0.1;
        } else {
            img.style.opacity = 1;
            clearInterval(fadeOut);
        }
    }, 200);
    i++;
}

setInterval(run, 2000);
run();
```

### 常用的 JQuery 效果

**\$(selector).fadeIn(speed,callback):** 根據所設的速度，逐漸地將目標透明度變 0，最後執行 callback 傳入的函式。

**\$(selector).fadeOut(speed,callback):** 根據所設的速度，逐漸地將目標度變 1，最後執行 callback 傳入的函式。

## JSON 遞迴應用

當資料量大時，往往會需要遞迴 JSON 內所有數據，來實踐我們想要的結果。

遍歷物件方法:

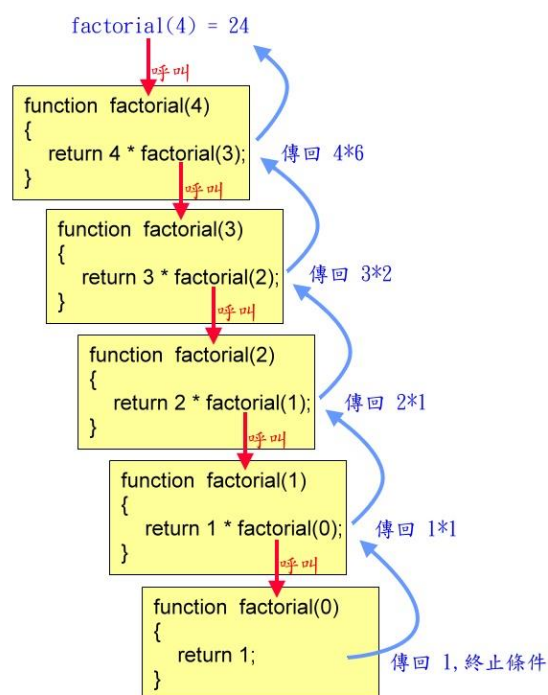
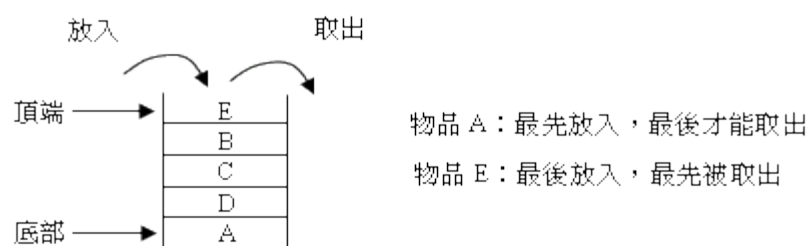
- **jQuery:**

```
$.each(data, function(index, json) {  
});
```

- **pure JS:**

```
for(var i = 0; i < arr.length; i++) {} //array  
for(var key in obj) {} //obj
```

遞迴（**Recursion**）是在函式中呼叫自身同名函式，而呼叫者本身會先被置入記憶體堆疊中，等到被呼叫者執行完畢之後，再從堆疊中取出之前被置入的函式繼續執行。「堆疊」（**Stack**）本身是一種「先進後出」的資料結構，就好比將書本置入箱中，最先放入的書會最後才取出。





## 範例 1 – 12

```
(function () {  
    var req = new XMLHttpRequest();  
    req.onreadystatechange = function () {  
        if (this.readyState == 4 && this.status == 200) {  
            var data = JSON.parse(this.responseText);  
            document.getElementsByClassName("navigation")[0].innerHTML = build(data, 0);  
        }  
    }  
    req.open('GET', 'data.json');  
    req.send();  
  
    function build(datas, parent) {  
        var html = "";  
        for (var i = 0; i < datas.length; i++) {  
            if (datas[i].parent == parent) {  
                html += "<li>";  
                html += "<a href=\"\" + datas[i].url + \"\">\" + datas[i].name + "</a>";  
                html += "<ul>\" + build(datas, datas[i].id) + "</ul>";  
                html += "</li>";  
            }  
        }  
        $.each(datas, function(index, data){  
            if(data.parent == parent) {  
                html += "<li>";  
                html += "<a href=\"\" + data.url + \"\">\" + data.name + "</a>";  
                html += "<ul>\" + build(datas, data.id) + "</ul>";  
                html += "</li>";  
            }  
        });  
        return html;  
    }  
})();
```

## ES6,7,8 簡介

ES 全名 ECMAScript，ECMAScript 是 ECMA 制定的標準化腳本語言，目前所使用的 JavaScript 是由 ECMAScript 在延伸實作的，當 ES 制定新的標準後，JS 也會逐漸地實作該標準。

## Async / Await 介紹

aynsc await 是一個promnise的語法糖，JS是一種非同步的語言，在過去我們為了要讓程式同步執行時，會靠過callback的方式讓程式執行完後再繼續執行，也造成了惡名昭彰的callback波動拳。



```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  });
26 }
```

為了改善這種狀態，後來發明了**promise**來解決同異步問題。

## 範例 1 – 14

```
(function () {  
  function delay() {  
    return new Promise(function (resolve, reject) {  
      setTimeout(function () {  
        alert("1");  
        resolve();  
      }, 1000);  
    });  
  }  
  
  delay().then(function () {  
    alert("2");  
  }).catch(function (error) {  
    console.log(error);  
  });  
})();
```

而`async/await`則是更進一步包裝`promise`

```
(async function () {  
  function delay () {  
    return new Promise(function (resolve, reject) {  
      setTimeout(function () {  
        alert('1')  
        resolve()  
      }, 1000)  
    })  
  }  
  
  await delay();  
  alert("2");  
})();
```

## 箭頭函式介紹

首先，先看一下以往撰寫函式的方法：

```
function show() {  
  console.log("hello world");  
}
```

改成箭頭函數的寫法:

```
var show = () => {  
  console.log("hello world");  
}
```

### 箭頭函數的差異

- 若只需要傳入一個參數的話，不需要使用括號
- 若本身只是要回傳某個值得話，可以直接省略大括號跟 **return**。
- 箭頭函數當中的 **this** 是定義時的對象，而不是使用時的對象。(重要)

### let, const 介紹

- **let** 與 **var** 差不多都是宣告變數。
- **const** 是宣告常數，宣告過後不能再對其值作修改。

**var** 宣告變數時，變數的範圍在 **function** 內，而 **let** 作用域是在 **{}** 內，除了 **function** 以外 **if for** 的 **{}** 都屬於 **let** 的作用域。

### 範例 1 – 15

```
(function () {  
  for (var i = 0; i < 10; i++) {  
    console.log(i);  
    setTimeout(function () {  
      console.log('這執行第' + i + '次');  
    }, 10);  
  }  
})();  
  
(function () {  
  for (let i = 0; i < 10; i++) {  
    console.log(i);  
    setTimeout(function () {  
      console.log('這執行第' + i + '次');  
    }, 10);  
  }  
})();
```

### String template 介紹

樣板字串是由 ``` 所組成的字串，與一般字串相比，他多了幾個特性：

### 範例 1 – 16

- 多行字串：可以直接以空格達成換行。

```
console.log('string text line 1\n' +  
  'string text line 2');  
console.log(`string text line 1  
string text line 2`);
```

- 運算式內插：可以直接將運算式或是變數插入到字串中

```
var a = 5;  
var b = 10;  
console.log('Fifteen is ' + (a + b) + ' and not ' + (2 * a + b) + '.');  
console.log(`Fifteen is ${a + b} and not ${2 * a + b}.`);
```

- 巢狀樣板: 可以在樣板裡插入另一個樣板

```
const classDeatail = {
  teacher: "Steve",
  students: ["Lily", "Paul"]
}

function renderList(students) {
  return `
    <div>上課名單</div>
    <ul>
      ${students.map(student => `<li>${student}</li>`).join('')}
    </ul>
  `
}

let template = `
<div class="template">
<h2>導遊: ${classDeatail.teacher}</h2>
${renderList(classDeatail.students)}
</div>

document.getElementsByTagName("body")[0].insertAdjacentHTML('beforeend', template);
```

## 排序演算法介紹

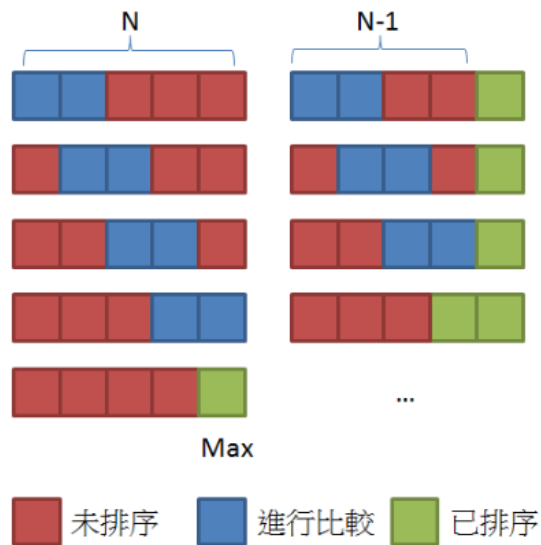
雖然 Js 裡面有內建 `sort` 的方式讓同學們去呼叫來排序，但排序方法在電腦科學裡一直是個經典問題，幾個經典的排序法還是需要瞭解一下。

- $O()$ : 簡稱 **Big-O** 在電腦科學裡指整個程式跑完  $n$  個所需花費的時間。

### 氣泡排序法(Bubble Sort):

最容易理解和實作的一種排序演算法，也翻譯作冒泡排序法，花費時間平均為  $O(n^2)$ 。

1. 比較相鄰的兩個元素，若前面的元素較大就進行交換。
2. 重複進行 1 的動作直到最後面，最後一個元素將會是最大值。
3. 重複進行 1,2 的動作，每次比較到上一輪的最後一個元素。
4. 重複進行以上動作直到沒有元素需要比較。



### 範例 1 – 17

```
function bubble(arr) {
  let len = arr.length;
  for (let i = 0; i < len; i++) {
    for (let j = 0; j < len - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
  return arr;
}
```

### 快速排序法(Quick sort):

又稱快排，在平均狀況下排序  $n$  個項目所花費時間為  $O(n \log n)$ ，最差的情況下則為  $O(n^2)$ ，通常比其他的演算法排的還快。

1. 數列中選擇一元素作為基準點(pivot)。
2. 小於此元素的移至基準的左邊，大於此元素的移至右邊，相等的任意放。
3. 基準點左邊和右邊視為兩個數列，並重複做以上動作直到數列剩下一個或零個元素。



```
function quickSort(arr, left = 0, right = arr.length - 1) {
  let len = arr.length,
      index
  if (len > 1) {
    index = partition(arr, left, right)
    if (left < index - 1) {
      quickSort(arr, left, index - 1)
    }
    if (index < right) {
      quickSort(arr, index, right)
    }
  }
  return arr
}

function partition(arr, left, right) {
  let middle = Math.floor((right + left) / 2),
      pivot = arr[middle],
      i = left,
      j = right

  while (i <= j) {
    while (arr[i] < pivot) {
      i++
    }
    while (arr[j] > pivot) {
      j--
    }
    if (i <= j) {
      [arr[i], arr[j]] = [arr[j], arr[i]]
      i++
      j--
    }
  }
}
```

ref: <http://emn178.pixnet.net/blog>

## 第二章： AJAX 技術應用

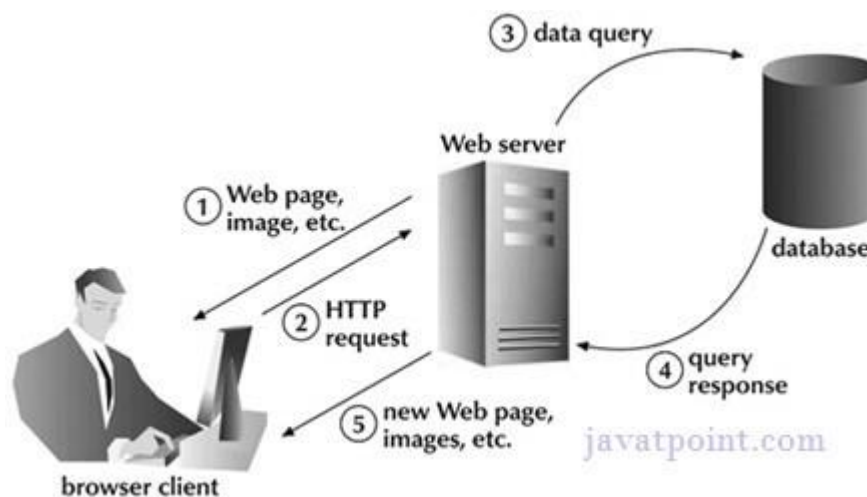
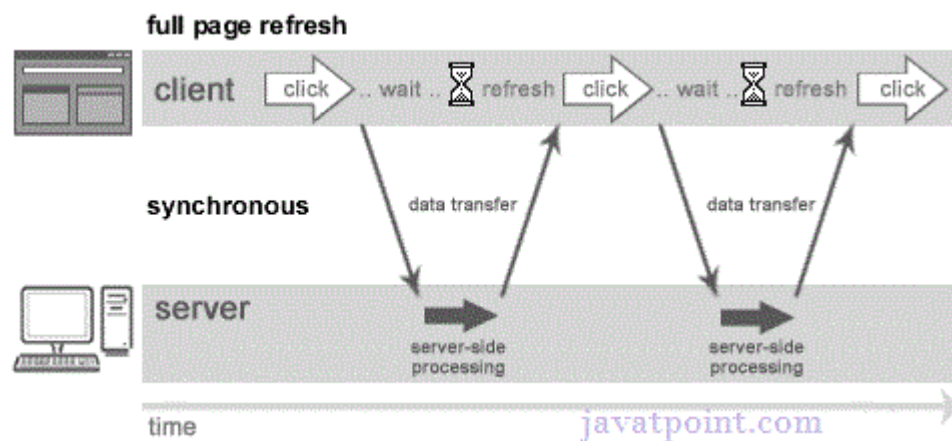
### AJAX 簡介

AJAX(Asynchronous JavaScript and XML)指的是一套綜合了多項技術的瀏覽器端網頁開發技術。

- 傳統上的網頁應用
  - User → Web (Form) → Server → Data with new web page (新的網頁)
- 現代的網頁應用
  - User → Web (AJAX) → Server → Data (原本的網頁)

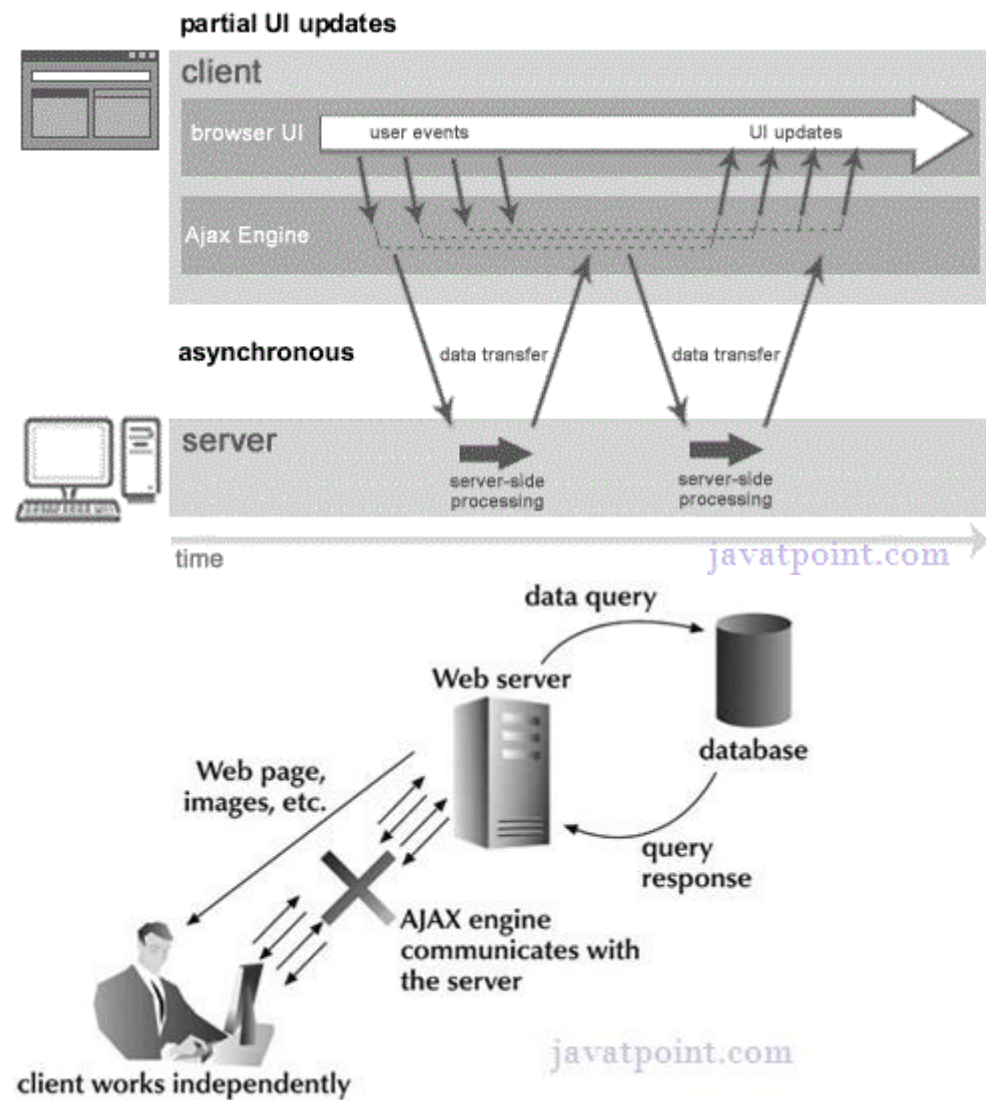
ref: <https://www.javatpoint.com/understanding-synchronous-vs-asynchronous>

古代:





現代:



優點

- 改善用戶體驗
- 減少頻寬使用
- 提高公司產品價值
- 異步與伺服器通信
- 前後端分離

寫在一起的例子:

```
205     <div id="rightpanel">
206         <i><label style="font-size: 40px; font-family: Times Verdana; margin: -20px 0 20px 0;">Search.. Book.. Go!</label></i><br><br>
207         <label style="font-size: 40px">Search Bus Trips</label><br>
208         <!--<i class="fa fa-compass fa-2x searchinput" aria-hidden="true"></i>
209         <input type="text" placeholder="From" name="from" size="40" style="padding-left: 45px;" required><br>
210         <i class="fa fa-map-marker fa-2x searchinput" aria-hidden="true"></i>-->
211         <?php
212             $store_value=array();
213             $store_value2=array();
214             $get_value = "SELECT DISTINCT location FROM location_destination";
215             $result = $DBcon->query($get_value);
216             if ($result->num_rows > 0) {
217                 while($row_result = $result->fetch_assoc()) {
218                     $store_value[]=$row_result["location"];
219                 }
220             }
221             $get_value2 = "SELECT DISTINCT destination FROM location_destination";
222             $result2 = $DBcon->query($get_value2);
223             if ($result2->num_rows > 0) {
224                 while($row_result2 = $result2->fetch_assoc()) {
225                     $store_value2[]=$row_result2["destination"];
226                 }
227             }
228             //FROM SELECT TAG
229             echo '<i class="fa fa-compass fa-2x searchinput" aria-hidden="true"></i>';
230             echo '<select name="from" id="from" style="margin-bottom:7px; padding-left: 42px;" required>';
231
232             for($i=0;$i<sizeof($store_value);$i++)
233             {
234                 echo "<option value='".$store_value[$i]."'>$store_value[$i]</option>";
235             }
236             echo '</select>';
237
```

## 缺點

- SEO 變差
- 程式碼複雜
- 閹割了瀏覽器的 History 和 Back 功能

完全應用 AJAX 技術的網頁應用 → SPA(Single Page Application)

解決網頁 History, Back 等問題 → 透過 html5 history 屬性解決 (三大前端  
框架 Angular, React, Vue 皆有實作)

解決 SEO 問題 → SSR(Server Side Render)

## HTTP Method:

**GET:** 請求展示指定資源。使用 **GET** 的請求只應用於取得資料。(Safe, Idempotent, Cacheable)

**POST:** 用於提交指定資源的實體，通常會改變伺服器的狀態或副作用。

**DELETE:** 會刪除指定資源。(Idempotent)

**PUT:** 會取代指定資源所請求的所有表現。(Idempotent)

## AJAX 實作

- XMLHttpRequest

範例 2-1

```
var req = new XMLHttpRequest();
req.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
        photos = JSON.parse(this.responseText);
    }
}
req.open('GET', 'photos.json', false);
req.send();
```

1

除了 `onreadystatechange` 外，也可以用 `addEventListener(type, function({}))` 來註冊事件

常使用的 `type`:

`progress, load, error, abort`

- \$.ajax (jQuery)

範例 2-2

```
(function () {
    $.ajax({
        url: "data.json",
        type: "get",
        success: function(res) {
            document.getElementById('content').innerHTML = JSON.stringify(res);
        },
        error: function(err) {
            console.log(err);
        }
    });
})();
```

除了這些功能以外，偶而會用到的還有:

`async, data, datatype, timeout`

ref: [https://www.w3schools.com/jquery/ajax\\_ajax.asp](https://www.w3schools.com/jquery/ajax_ajax.asp)

- Fetch

範例 2-3

```
(function () {  
    fetch("data.json", {  
        method: "GET"  
    }).then((response) => {  
        console.log(response);  
        // 可以透過 blob(), json(), text(), formData(), arrayBuffer() 轉成可用的資訊  
        return response.text();  
    }).then((data) => {  
        document.getElementById('content').innerHTML = data;  
    }).catch((err) => {  
        console.log('錯誤:', err);  
    });  
})();
```

Fetch API 是瀏覽器最新提供的一個 Javascript 接口，與 XMLHttpRequest 功能相似，但提供了更方便強大的功能。







- 不用每次重新創造一個新的 XMLHttpRequest 實體。
- 使用 promise 作為回應。
- 用 then 來處理得到的值後續動作，Catch 來處理錯誤。
- IE 不支援
- 預設是不帶 cookie 不接受 cookie 的，若要使用可開啟 credentials

ref:

<https://developer.mozilla.org/enUS/docs/Web/API/WindowOrWorkerGlobalScope/fetch>

- Axios

Axios 是一個開源的 Ajax 庫，支援常見的瀏覽器。

					
Latest ✓	Latest ✓	Latest ✓	Latest ✓	Latest ✓	11 ✓

node.js 環境: 發出 http 請求

瀏覽器環境: 生成 XMLHttpRequests 請求

支持 Promise API

自動轉換 JSON 資料

## 範例 2-4

```
(function () {
  axios({
    method: "get",
    url: "data.json",
    responseType: 'json'
  }).then(function (response) {
    console.log(response);
    document.getElementById('content').innerHTML = JSON.stringify(response.data);
  });
})();
```

```
▼ {data: {...}, status: 200, statusText: "OK", headers: {...}, config: {...}, ...} ⓘ
  ► config: {adapter: f, transformRequest: {...}, transformResponse: {...}, timeout: ...
  ► data: {name: "steve", age: 27, sex: "male"}
  ► headers: {date: "Sat, 12 Jan 2019 07:38:37 GMT", last-modified: "Sat, 12 Jan ...
  ► request: XMLHttpRequest {onreadystatechange: f, readyState: 4, timeout: 0, wi...
    status: 200
    statusText: "OK"
  ► __proto__: Object
```

ref: <https://github.com/axios/axios>

## 內容區塊 AJAX 動態載入

很多時候，我們希望呈現給使用者看的資訊，可能是隨時會更動的，或是來自第三方的資料，這時候就要以 **AJAX** 當使用者每次使用時網頁，動態的抓取資料並顯示給使用者看，我們以政府開放 **API** 平台上的獨立音樂活動資訊 (<https://data.gov.tw/dataset/6006>) 做一個簡單的活動資訊應用。

## 範例 2-5

文化部整合各公、民營單位之獨立音樂活動資訊。

活動名稱	活動圖片	活動連結	開始時間	結束時間
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/01/30	2019/01/30
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/02/27	2019/02/27
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/03/27	2019/03/27
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/04/24	2019/04/24
THE LOST FUTURE---失控未來		<a href="#">Link</a>	2019/01/20	2019/01/20
南迴樂 We Are South 2019		<a href="#">Link</a>	2019/01/18	2019/01/20
樂學講堂   音樂職業規劃-樂手   林正如		<a href="#">Link</a>	2019/01/14	2019/01/14
樂學講堂   音樂職業規劃-製作   林尚德		<a href="#">Link</a>	2019/01/21	2019/01/21

當使用者每次登入該頁面時，所得到的資訊都是當下即時更新的，確保使用者不會看到舊的資訊。

## 註冊事件與事件綁定

很多時候我們利用 **AJAX** 獲得資料後，載入該資料的內容給使用者看後，還希望在該內容上綁定一些 **Javascript** 動作，讓使用者可以點擊資料做操作。我們修改 2-5 的範例，在每一個活動後新增一個刪除的按鈕，讓使用者點選按鈕後，可以刪除該行資訊。

## 範例 2-6

活動名稱	活動圖片	活動連結	開始時間	結束時間	
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/01/30	2019/01/30	刪除
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/02/27	2019/02/27	刪除
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/03/27	2019/03/27	刪除
週3聚樂部-羅大佑與音樂瘋子的傳奇派對		<a href="#">Link</a>	2019/04/24	2019/04/24	刪除
THE LOST FUTURE---失控未來		<a href="#">Link</a>	2019/01/20	2019/01/20	刪除

```
(function () {
    axios({
        method: "get",
        url: "https://cloud.culture.tw/frontsite/trans/SearchShowAction.do?method=doFindTypeJ&category=5",
        responseType: 'json',
    }).then(function (response) {
        var activitys = response.data;
        var html = ""
        activitys.forEach(function(activity) {
            html += `<tr>
                <td>${activity.title}</td>
                <td><img src=${activity.imageUrl}></td>
                <td><a href=${activity.webSales}>Link</a></td>
                <td>${activity.startDate}</td>
                <td>${activity.endDate}</td>
                <td><button class="delete">刪除</button></td>
            </tr>`;
        });
        document.getElementById('content').innerHTML = html;
    });

    Array.from(document.getElementsByClassName("delete")).forEach(function (element) {
        element.addEventListener('click', function () {
            var targetRow = this.parentNode.parentNode;
            this.parentNode.parentNode.parentNode.removeChild(targetRow);
        });
    });
})();
```

試著想想看，這樣子按刪除按鈕後，會成功刪除嗎？

因為 **AJAX** 是非同步執行的緣故，瀏覽器執行 **addEventListener** 的時候，該元素還沒有被加入 **DOM** 中，所以沒有辦法將事件綁定到該按鈕上面，通常這種情況會有幾種解決辦法：

- 使用 `async / await` 使 AJAX 變成同步執行。

```
(async function () {
  await axios({
    method: "get",
    url: "https://cloud.culture.tw/frontsite/trans/SearchShowAction.do?method=doFindTypeJ&category=5",
    responseType: 'json',
  }).then(function (response) {
    var activitys = response.data;
    var html = ""
    activitys.forEach(function(activity) {
      html += `<tr>
        <td>${activity.title}</td>
        <td><img src=${activity.imageUrl}></td>
        <td><a href=${activity.webSales}>Link</a></td>
        <td>${activity.startDate}</td>
        <td>${activity.endDate}</td>
        <td><button class="delete">刪除</button></td>
      </tr>`;
    });
    document.getElementById('content').innerHTML = html;
  });

  Array.from(document.getElementsByClassName("delete")).forEach(function (element) {
    element.addEventListener('click', function () {
      var targetRow = this.parentNode.parentNode;
      this.parentNode.parentNode.parentNode.removeChild(targetRow);
    });
  });
})();
```

- 將綁定的程式碼移到 AJAX 程式碼片段裡面。

```
(function () {
  axios({
    method: "get",
    url: "https://cloud.culture.tw/frontsite/trans/SearchShowAction.do?method=doFindTypeJ&category=5",
    responseType: 'json',
  }).then(function (response) {
    var activitys = response.data;
    var html = ""
    activitys.forEach(function(activity) {
      html += `<tr>
        <td>${activity.title}</td>
        <td><img src=${activity.imageUrl}></td>
        <td><a href=${activity.webSales}>Link</a></td>
        <td>${activity.startDate}</td>
        <td>${activity.endDate}</td>
        <td><button class="delete">刪除</button></td>
      </tr>`;
    });
    document.getElementById('content').innerHTML = html;
    Array.from(document.getElementsByClassName("delete")).forEach(function (element) {
      element.addEventListener('click', function () {
        var targetRow = this.parentNode.parentNode;
        this.parentNode.parentNode.parentNode.removeChild(targetRow);
      });
    });
  });
})();
```

- 綁定目標的父元素，判斷點擊時是否是原本要綁定的目標。

```
(function () {
  axios({
    method: "get",
    url: "https://cloud.culture.tw/frontsite/trans/SearchShowAction.do?method=doFindTypeJ&category=5",
    responseType: "json",
  }).then(function (response) {
    var activitys = response.data;
    var html = ""
    activitys.forEach(function(activity) {
      html += `<tr>
        <td>${activity.title}</td>
        <td><img src=${activity.imageUrl}></td>
        <td><a href=${activity.webSales}>Link</a></td>
        <td>${activity.startDate}</td>
        <td>${activity.endDate}</td>
        <td><button class="delete">刪除</button></td>
      </tr>`;
    });
    document.getElementById("content").innerHTML = html;
  });
  document.getElementById("content").addEventListener("click", function(e){
    if (e.target.className == 'delete') {
      var targetRow = e.target.parentNode.parentNode;
      e.target.parentNode.parentNode.parentNode.removeChild(targetRow);
    }
  });
})();
```

### 多個 AJAX 同時執行

有時候我們會遇到一個狀況是，需要從多個 API 獲取資料後，再顯示其資料給使用者看。例如，當系統比較龐大時，管理員要看所有用戶的資料及該用戶的權限時，會要從用戶資料的 API 獲取用戶列表，並從權限資料的 API 獲取權限對應列表，再將兩邊的資料結合起來給使用者看。因為 AJAX 是非同步執行的，我們不能確定哪邊的 API 會先回傳 API 回來，這時候為了正確顯示所有的資料給使用者看，我們可以將 AJAX 調用成同步方式，一個一個拿到資料後再去顯示資料給使用者。但這樣會拉長整個調用資料的反應時間，所以更好的方式是，讓 AJAX 一同發出，等所有 AJAX 都接收到回應時，再去跑顯示資料的函式，實作方式可以用 `axios.all()` 的方式或是將每一個 ajax 包成 `promise` 物件，使用 `promise.all()` 的方式實作。



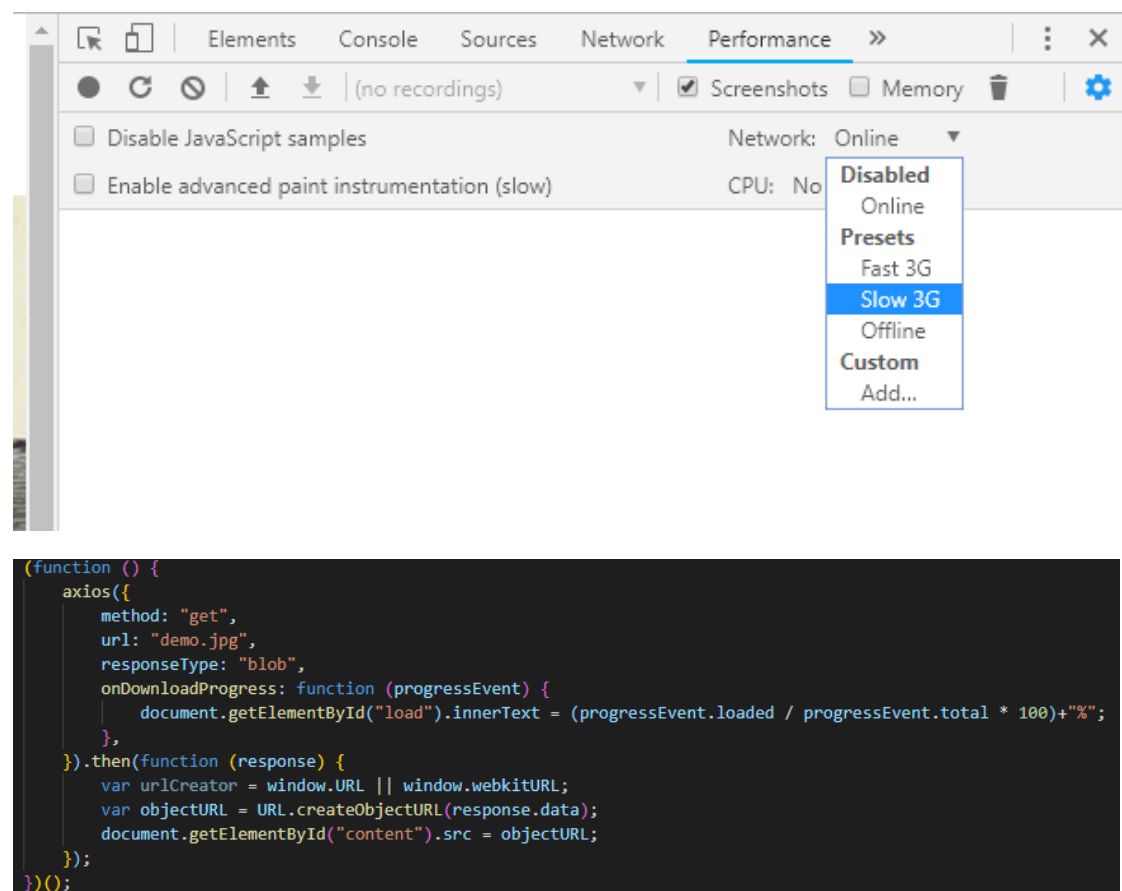
## 範例 2-7

```
(function () {  
  
    axios.all([getUser(), getLesson()])  
    .then(axios.spread(function (users, lessons) {  
        renderTable(users.data, lessons.data);  
    }));  
  
    function getUser() {  
        return axios.get('user.json');  
    }  
  
    function getLesson() {  
        return axios.get('lesson.json');  
    }  
  
    function renderTable(students, lessons) {  
        var html = '';  
        students.forEach(function(student) {  
            var lesson = lessons[student.name];  
            html += `<tr>  
                <td>${student.name}</td>  
                <td>${lesson.title}</td>  
                <td>${student.age}</td>  
                <td>${lesson.time}</td>  
            </tr>`  
        })  
        document.getElementById('content').innerHTML = html;  
    }  
})();
```

## AJAX 下載進度條

我們常會看到很多網路上有許多應用會顯示使用者的下載進度(通常在遊戲上或動畫上會出現)，我們利用 **CHROME** 提供的模擬低速網路功能，利用下載一張圖片來實作這個功能。

## 範例 2-8



這裡有用到了一個叫做 `createObjectURL` 的 api，當我們在做上傳資料(通常是圖片)的時候，我們希望能有一個預覽的功能，讓使用者知道他到底選擇了甚麼檔案準備上傳，但是若讓使用者選完後就把資料都上傳再丟預覽圖給使用者，會造成伺服器負擔過大，因此 `createObjectURL` 這個 api 可以幫助我們在本地端開啟一個暫時的連結，裡面存放著我們指定的資料。

## 範例 2-9

```
(function () {
    window.URL = window.URL || window.webkitURL;

    const fileSelect = document.getElementById("fileSelect"),
        fileElem = document.getElementById("fileElem"),
        fileList = document.getElementById("fileList");

    fileSelect.addEventListener("click", function (e) {
        if (fileElem) {
            fileElem.click();
        }
        e.preventDefault();
    }, false);

    fileElem.addEventListener("change", handleFiles);

    function handleFiles() {
        var files = this.files;
        if (!files.length) {
            fileList.innerHTML = "<p>No files selected!</p>";
        } else {
            fileList.innerHTML = "";
            const list = document.createElement("ul");
            fileList.appendChild(list);
            for (let i = 0; i < files.length; i++) {
                const li = document.createElement("li");
                list.appendChild(li);

                const img = document.createElement("img");
                img.src = window.URL.createObjectURL(files[i]);

                img.onload = function () {
                    window.URL.revokeObjectURL(this.src);
                }
                li.appendChild(img);
                const info = document.createElement("span");
                info.innerHTML = files[i].name + ": " + files[i].size + " bytes";
                li.appendChild(info);
            }
        }
    }
})();
```

## AJAX 製作自動載入的新聞頁面

現在人使用網頁的習慣，越來越不能接受每要看一次新的資料，就要整個網頁重新載入一次，因此很多網頁都逐漸改成每往下滑，便不斷的載入新的資料給使用者看，例如痞客邦、臉書、時刻旅行等等。

要達成這個功能，首先我們必須要判斷畫面是否到了最底端。

## 範例 2-10

```
function getScrollTop() {
    var scrollTop = 0, bodyScrollTop = 0, documentScrollTop = 0;
    if (document.body) {
        bodyScrollTop = document.body.scrollTop;
    }
    if (document.documentElement) {
        documentScrollTop = document.documentElement.scrollTop;
    }
    scrollTop = (bodyScrollTop - documentScrollTop > 0) ? bodyScrollTop : documentScrollTop;
    return scrollTop;
}

function getScrollHeight() {
    var scrollHeight = 0, bodyScrollHeight = 0, documentScrollHeight = 0;
    if (document.body) {
        bSH = document.body.scrollHeight;
    }
    if (document.documentElement) {
        dSH = document.documentElement.scrollHeight;
    }
    scrollHeight = (bSH - dSH > 0) ? bSH : dSH;
    return scrollHeight;
}

function getWindowHeight() {
    var windowHeight = 0;
    if (document.compatMode == "CSS1Compat") {
        windowHeight = document.documentElement.clientHeight;
    } else {
        windowHeight = document.body.clientHeight;
    }
    return windowHeight;
}
```

我們需要知道三個高度：滾動的高度、整個網頁的總高度、使用者看到畫面的高度。當滾動的高度加上使用者看到畫面的高度等於整個網頁的總高度時，就表示網頁已經到了最底端，這時候就要去透過 **AJAX** 拿新的資料回來接在下方，讓使用者可以繼續閱覽資料。

```
window.onscroll = async function () {
    while (getScrollTop() + getWindowHeight() >= getScrollHeight()) {
        await axios({
            method: 'get',
            url: `news${i}.html`
        }).then(function (response) {
            document.getElementById('content').innerHTML += response.data;
        });
        i++;
    }
}

window.onload = window.onscroll;
```

備註: 如果有使用 JQuery 的話，判斷畫面是否到最底端的功能可以更簡單實現。

```
$(window).scroll(function () {  
    var scrollTop = $(this).scrollTop();  
    var scrollHeight = $(document).height();  
    var windowHeight = $(this).height();  
});
```

## AJAX 上傳檔案

一般上傳檔案會用兩種方式:

- 將本地檔案轉換成編碼後上傳(最常見於將圖片轉換成 **base64** 後透過普通 **post** 上傳到後端)
- 透過 **form** 表單提交

### 範例 2 – 11

```
(function () {  
    document.getElementById("upload_file1").addEventListener('change', function(){  
        var file = this.files[0];  
        var f = new FileReader();  
        f.onload = function () {  
            console.log(f.result);  
            axios({  
                method: 'post',  
                url: `example.com`,  
                data: f.result  
            }).then(function (response) {  
                document.getElementById('content').innerHTML += response.data;  
            });  
        }  
        f.readAsDataURL(file);  
    })  
})  
  
document.getElementById("upload_file2").addEventListener('change', function () {  
    var file = this.files[0];  
    var formData = new FormData();  
    formData.append('img', file, file.name);  
    console.log(formData.get("img"));  
    let config = {  
        headers: { 'Content-Type': 'multipart/form-data' }  
    };  
    axios.post('example.com', formData, config).then(response => {  
        console.log(response.data);  
    });  
})  
})();
```

## 第三章： 外部 API 使用

### API 介紹

應用程式介面（**application programming interface**），就是軟體系統不同組成部分銜接的約定方法。由於近年來軟體的規模日益龐大，常常需要把複雜的系統劃分成小的組成部分(微服務當道)，**API** 的設計便十分重要。程式設計的實作中，**API** 的設計首先要使軟體系統的職責得到合理劃分。良好的設計可以降低系統各部分的相互依賴程度，並降低組成單元間的耦合程度，從而提高系統的可維護性和延伸性。

### RESTful API

**Representational State Transfer (REST)**，是指一種設計架構的風格，並不是一種硬性的標準。

已取得會員資料為例

獲取使用者資料 /GET /users

獲取使用者資料 /GET /user/1

新增使用者資料 /POST /user

更新使用者資料 /PUT /user/1

刪除使用者資料 /DELETE /user/1

如何存取與使用 API

常見的 API Doc:

The screenshot shows an API documentation interface for a POST endpoint. At the top, it says 'POST /user Create user'. Below this, a note states 'This can only be done by the logged in user.' and there is a 'Try it out' button. The 'Parameters' section is divided into a table with 'Name' and 'Description' columns. The 'body' parameter is listed as 'required' and its description is 'Created user object'. An 'Example Value' is provided as a JSON object: 

```
{  "id": 0,  "username": "string",  "firstName": "string",  "lastName": "string",  "email": "string",  "password": "string",  "phone": "string",  "userStatus": 0}
```

. Below the example, there is a 'Parameter content type' dropdown menu set to 'application/json'. The 'Responses' section also has a table with 'Code' and 'Description' columns. The 'default' response is shown with a description of 'successful operation'. A 'Response content type' dropdown menu is set to 'application/xml'.

注意:

- 傳送的方法
- 傳送的網址
- 要帶的 **query**
- 要帶的 **params**
- 是否需要帶 **token**
- **header** 是否要加驗證
- 回傳的狀態碼以及描述

**Mock server**

在與後端工程師開發的時候，我們往往會遇到一個狀況是，**API** 還沒有開發完成，但是我們已經需要測試我們的程式是否可以正常運行，我們就需要一個 **Mock**(假裝它已經做好)一個後端伺服器來幫助我們進行測試，這裡推薦一個

網站 <https://www.easy-mock.com/>，easy-mock 可以幫助你建立一個簡單的假伺服器，回傳你想接到的 JSON 資料。

### 會員註冊檢查帳號

我們的會員系統，通常會要求會員的帳號或密碼有一定的格式，或是不能重複註冊，後端可能會提供我們一個驗證的 API 來讓我們去呼叫它驗證使用者輸入的帳號是否可以註冊的，我們這裡當每次使用者輸入鍵盤時，去呼叫一個 ease-mock 製作的簡單 API 來做驗證。

### 範例 3 - 1

```
(function () {
  document.getElementById("account").addEventListener("keyup", function(){
    var account = this.value;
    axios({
      method: "get",
      url: `https://www.easy-mock.com/mock/5c421ea1d13fff0d542dc782/example/account/check?account=${account}`,
      responseType: "json",
    }).then(function (response) {
      if (response.data.status == false) {
        document.getElementById("hint").innerText = "帳號格式錯誤";
      } else {
        document.getElementById("hint").innerText = "";
      }
    });
  });
})();
```

### 跨域請求概念介紹

現在的瀏覽器都有實作同源策略(通常情況下只能呼叫同網域下的資源)，常常我們會想要從別的服務上獲得他們的資料來用，會跑出 CORS 錯誤，因而被擋下來，就是因為同源防護的緣故。

所以，瀏覽器為什麼要搞事情？就是不想給好日子我們過？對於這樣的質問，瀏覽器官方文件只說了：「同源策略限制了從同一來源加載的文檔或腳本如何與來自另一個源的資源進行交互。這是一個用於隔離潛在惡意文件的重要安全機制。」

試想，如果沒有同源限制的 API 情況下會發生什麼事情：

1. 你正在逛你的網路銀行。
2. 好友貼了一個連結給你，你點進去瀏覽中。
3. 該網站背地裡幫你呼叫了你的網路銀行匯款 API，因沒有跨域限制，也是從你的瀏覽器發出的請求，銀行那端就會自動認為是你想要匯款而將你的錢轉走。
4. 你變很窮

另一個情況，當沒有同源限制的情況下，我們可以申請一個跟目標網頁相似的網域名(e.g. facebook.com vs ffacebook.com)，並在其中用一個 iframe 連結至目標網頁，這樣子我們就可以很輕易的在我們自己的網頁內拿到使用者的所



有資訊，甚至偽裝成該使用者進行非法的操作，這些也就是很常說到的 **CSRF** 攻擊。

### 範例 3-2

```
(function () {  
  document.getElementById("attack").addEventListener("click", function(){  
    var account = window.frames["fake"].document.getElementById("account").value;  
    document.getElementById("hint").innerText = account;  
  })  
})();
```

### JSONP 介紹

原理: 在 **HTML** 標籤里，一些標籤例如 **img** 獲取資源是沒有跨域限制的。

實作: 與後端約定好一個函數的名字，回傳的資料變成執行該函式，創立一個標籤去獲取該函式，瀏覽器便會立即執行。

### 範例 3-3

```
(function () {  
  window.callback = function (data) {  
    alert('我是本地函數，JSONP帶來的資料是' + data.data);  
  };  
  
  var script = document.createElement('script');  
  script.src = 'https://www.easy-mock.com/mock/5c421ea1d13fff0d542dc782/example/jsonp/test?jsonp_param_name=callback';  
  script.type = 'text/javascript';  
  document.body.append(script);  
  script.remove();  
})();
```

本地端先創立一個 **callback** 函數，**jsonp** API 回傳的資料則是 **callback({data: test})**，創立一個 **script** 標籤後立即去執行該函式，即可獲得 **JSON** 資料。

### CORS 介紹

跨域資源共享（**Cross-origin resource sharing CORS**）是一個 **W3C** 標準，這是一個對於跨域問題的處理標準，而 **CORS** 有兩種請求的類別:

#### 簡單請求(Simple request):

- 請求方法為: **HEAD**, **GET**, **POST**。
- **Header** 訊息不包含超過: **Accept**, **Accept-Language**, **Accept-Language**, **Last-Event-ID**。
- **content-Type** 只能為 **application/x-www-form-urlencoded**、**multipart/form-data**、**text/plain** 其中之一。

當發出跨域的 **AJAX** 請求時，瀏覽器判斷此次請求為簡單請求便會在 **Header** 內自動加上一個 **Origin** 值告訴伺服器，伺服器便可以藉由 **Header** 內的協議、域名、請求端口來判斷要不要回應值。

e.g.

```
GET /cors HTTP/1.1
Origin: http://api.example.com
Host: 127.0.0.1
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

如果伺服器判斷這是許可範圍內的來源，在回應的 **Header** 內便會加上幾個片段：

- **Access-Control-Allow-Origin:** 該值必須，要碼是 **Origin** 的值，要碼是 **\***。
- **Access-Control-Allow-Credentials:** 該值是一個布林值，只能設為 **true**，當是 **true** 的時候，代表伺服器允許瀏覽器傳送 **Cookie**，瀏覽器端對於 **CORS** 請求默認不會帶 **cookie**，如果要帶則要在設定打開。
- **Access-Control-Expose-Headers:** 該值可選，在一般的 **XMLHttpRequest** 的時候，**getResponseHeader** 這個方法只能拿到 **Cache-Control**、**Content-Language**、**Content-Type**、**Expires**、**Last-Modified**、**Pragma** 的值，如果想要拿到其他索引的值，就可以在這裡指定。

e.g.

```
Access-Control-Allow-Origin: http://api.example.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: example
Content-Type: text/html; charset=utf-8
```

**預檢請求(Preflight request):**

有些人又會稱為非簡單請求，主要是對於伺服器有特殊要求時會用到，例如 **PUT** 或 **DELETE** 方法，或著 **Content-Type** 值是 **application/json** 的請求，瀏覽器便會再正式開始通訊前增加一次查詢請求，稱為預檢請求。

e.g.

```
var url = 'http://api.example.com/';
var xhr = new XMLHttpRequest();
xhr.open('PUT', url, true);
xhr.setRequestHeader('X-Custom-Header', 'value');
xhr.send();
```

```
OPTIONS / HTTP/1.1
Origin: http://127.0.0.1
```

**Access-Control-Request-Method: PUT**  
**Access-Control-Request-Headers: X-Custom-Header**  
**Host: api.alice.com**  
**Accept-Language: en-US**  
**Connection: keep-alive**  
**User-Agent: Mozilla/5.0...**

預檢請求使用的呼叫方法是 **OPTIONS**，表示這個請求是用來詢問的，像簡單請求一樣會有 **Origin**，除此之外還會多 **Access-Control-Request-Method**、**Access-Control-Request-Headers** 字段。

當伺服器收到預檢請求判斷 **Origin**、**Access-Control-Request-Method** 和 **Access-Control-Request-Headers** 後，允許跨域請求便會做出回應。

e.g.

**HTTP/1.1 200 OK**  
**Date: Mon, 01 Dec 2018 01:15:39 GMT**  
**Server: Apache/2.0.61 (Unix)**  
**Access-Control-Allow-Origin: http://127.0.0.7**  
**Access-Control-Allow-Methods: GET, POST, PUT**  
**Access-Control-Allow-Headers: X-Custom-Header**  
**Access-Control-Allow-Credentials: true**  
**Access-Control-Max-Age: 1728000**  
**Content-Type: text/html; charset=utf-8**  
**Content-Encoding: gzip**  
**Content-Length: 0**  
**Keep-Alive: timeout=2, max=100**

**Connection: Keep-Alive**

**Content-Type: text/plain**

這裡要特別注意的有四個值

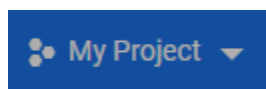
- **Access-Control-Allow-Methods:** 該字段為必須，它的值是以逗號分隔的一個字串，表示伺服器支持的所有的跨域請求方法。
- **Access-Control-Allow-Headers:** 如果瀏覽器請求包括 **Access-Control-Request-Headers**，則 **Access-Control-Allow-Headers** 為必需的。它也是一個逗號分隔的字串，表示伺服器支持的所有 **Header** 字段，不限於瀏覽器在"預檢"中請求的字段。
- **Access-Control-Allow-Credentials:** 與簡單請求時意義相同。
- **Access-Control-Max-Age:** 該字段可選，用來指定本次預檢請求的有效期，單位為秒。

預檢請求通過後，接下來的所有請求就跟簡單請求一樣了。

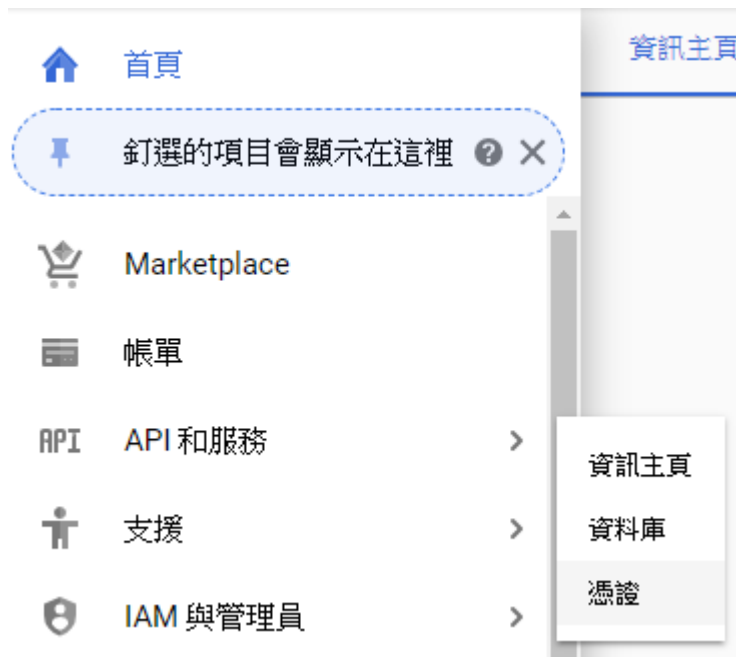
### Google map API 使用

瞭解 API 的相關基礎知識後，當我們要使用其他企業所提供的 API 服務時，通常會要使用 API Key、Header token 的方式取得使用權限，我們已申請 Google API 為例。

Step1. 到 Google 服務後台(<https://console.cloud.google.com>)註冊一個自己的專案。



Strp2. 選取 API 和服務，點選憑證頁面。



### Step3. 點選建立 API 金鑰



Step4. 得到金鑰後便能透過金鑰取得 Google map sdk 來透過 API 創立自己的地圖應用。

<https://maps.googleapis.com/maps/api/js?key={key}>

利用 Polling 模擬聊天室畫面

除了一般單次呼叫 API 的方式外，在上古時代的網路世界中，為了要達到讓使用者即時通訊的效果，我們便會在使用者進入該聊天畫面後，每隔一小段時間去呼叫一次 API 查詢是否有新訊息，這種做法我們稱之為 Polling。

### 範例 3 – 4

```
(function () {
  function getMessage() {
    axios({
      method: "get",
      url: `https://www.easy-mock.com/mock/5c421ea1d13fff0d542dc782/example/message`,
      responseType: "json",
    }).then(function (response) {
      document.getElementById("content").innerHTML += `<div>mockUser: ${response.data.message}</div>`;
    });
  }
  setInterval(getMessage, 1500);
})();
```

`setInterval(function, milliseconds, param1, param2, ...)`，該函式會不根據你所設定的毫秒不斷的重複呼叫第一個參數傳入的函數，而 `setInterval` 在呼叫時會回傳一個 `id`，如果要停止呼叫則使用 `clearInterval(id)` 即可。

### 範例 3 – 5

```
(function () {
  function getMessage() {
    axios({
      method: "get",
      url: `https://www.easy-mock.com/mock/5c421ea1d13fff0d542dc782/example/message`,
      responseType: "json",
    }).then(function (response) {
      document.getElementById("content").innerHTML += `<div>mockUser: ${response.data.message}</div>`;
    });
  }
  var id = setInterval(getMessage, 1500);
  document.getElementById("stop").addEventListener("click", function () {
    clearInterval(id);
  });
  document.getElementById("start").addEventListener("click", function () {
    id = setInterval(getMessage, 1500);
  });
})();
```

雖然現在已經有更好的方法來解決即時通訊的問題了，但是 **Polling** 還是被廣泛運用在 **email**、**監控版**、**廣播**等方式上。

獲得即時更新的方法 (**polling**、**comet**、**long polling**、**websocket**)

**Polling(輪詢):**

利用 Javascript 中的 `setInterval` 或 `setTimeout` 在固定的時間內向 Server 端發起 **AJAX** 請求取得最新的資料。

**Comet:**

把發出的 **AJAX** 請求像彗星的尾巴一樣拉的很長，這樣一來就可以讓伺服器保持連線的狀態，持續讓伺服器端回傳資料回來，這樣的作法其實就是把 **Polling** 做在 Server 端。

**Long Polling:**

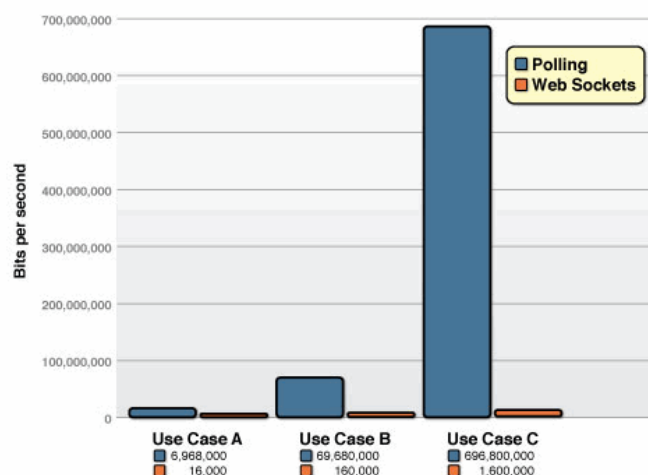
長時間輪詢的作法是 Comet 演化過後的方式，也是目前 Facebook、Plurk 實現動態更新的方法。這樣的作法是發一個長時間等待的 Request，當 Server 端有資料 Response 時立刻斷掉、接著在發一個新的 Request。

與 Polling 不同之處的在於他比較有效率，可以等到 Timeout 或拿到新資料的時候在重新發送，相對之下減少很多網路資源的浪費。

```
(function polling() {  
  axios({  
    method: "get",  
    url: `https://www.easy-mock.com/mock/5c421ea1d13fff0d542dc782/example/message`,  
    responseType: "json",  
    timeout: 30000  
  }).then(function (response) {  
  }).catch(function (error) {  
  }).then(function(){  
    polling();  
  });  
})();
```

#### WebSocket:

基於 TCP 協議上實現的雙向資料傳輸，支持雙向通訊，對於二進位制更好的支援，性能比 Polling 更好。



ref: <http://www.websocket.org>

## 第四章： 常用的功能介紹

### 正規表達式

正規表達式是被用來匹配字串中字元組合的模式，在 JavaScript 中，正規表達式也是物件。主要由數個簡易字元組成，例如 `/abc/`，或是由簡易字元及特殊符號組合而成，例如 `/ab*c/`、`/Chapter (\d+)\. \d*/`。

可以利用兩種方式創立正規表達式:

```
var re = /ab+c/;  
var re = new RegExp('ab+c');
```

常用到的特殊符號:

\	<p>將下一個字元標記為一個特殊字元（<b>File Format Escape</b>，清單見本表）、或一個原義字元（<b>Identity Escape</b>，有<code>^\$()*+?.[\ </code>共計 12 個）、或一個向後參照（<b>backreferences</b>）、或一個八進位轉義符。例如，「<code>n</code>」符合字元「<code>n</code>」。</p> <p>「<code>\n</code>」符合一個換行符。序列「<code>\\</code>」符合「<code>\</code>」而「<code>\ </code>」則符合「<code>(</code>」。</p>
^	<p>符合輸入字串的開始位置。如果設定了 <b>RegExp</b> 物件的 <b>Multiline</b> 屬性，<code>^</code>也符合「<code>\n</code>」或「<code>\r</code>」之後的位置。</p>
\$	<p>符合輸入字串的結束位置。如果設定了 <b>RegExp</b> 物件的 <b>Multiline</b> 屬性，<code>\$</code>也符合「<code>\n</code>」或「<code>\r</code>」之前的位置。</p>
*	<p>符合前面的子運算式零次或多次。例如，<code>zo*</code>能符合「<code>z</code>」、「<code>zo</code>」以及「<code>zoo</code>」。<code>*</code>等價於<code>{0,}</code>。</p>
+	<p>符合前面的子運算式一次或多次。例如，「<code>zo+</code>」能符合「<code>zo</code>」以及「<code>zoo</code>」，但不能符合「<code>z</code>」。<code>+</code>等價於<code>{1,}</code>。</p>



?

符合前面的子運算式零次或一次。例如，「do(es)?」可以符合「do」或「does」中的「do」。?等價於{0,1}。

其餘的可參考: <https://zh.wikipedia.org/wiki/正则表达式>

利用正規表達式取代網頁內容

利用 **replace** 方法將符合正規表達式的字串轉換成新的字串。

#### 範例 4-1

```
(function () {  
    document.getElementById("content").innerHTML =  
        document.getElementById('content').innerHTML.replace(/(手機[^\s, \. \-]*)/g, `<span>$1</span>`);  
})();
```

**replace** 第二個參數可以帶\$1,\$2.....\$99 表示第幾個正規表達式對到的字串。

利用正規表達式檢查帳號格式

除了 RWD 可以配合網頁大小調整 CSS 外，因為每個裝置上的瀏覽器都有些許不同的實作差異，偶時我們必須從 **navigator** 內的資訊判斷該裝置是甚麼作業系統、瀏覽器等等資訊。

#### 範例 4-2

```
(function() {  
    var text = "PC";  
    if (/android/i.test(navigator.userAgent)) {  
        text = "Android";  
    } else if (/iphone/i.test(navigator.userAgent)) {  
        text = "iPhone";  
    }  
    document.getElementsByTagName("body")[0].innerText = `${text} 版網頁`;  
})();
```

利用 **html form** 配合正規表達式檢查用戶輸入

html5 有提供一些 **form** 的新屬性來做驗證功能，像是 **required**、**pattern**、**type** 等，在送出表單時即可驗證使用者輸入是否符合規範，但是有時候我們並不想用送出表單的方式傳送資料，這時候可以藉由 JS 去單獨呼叫 **form api** 的驗證跟提示功能，幫助我們提示使用者是否有資料輸入錯誤。

### 範例 4 – 3

```
(function() {  
    document.getElementById("check").addEventListener("click", function() {  
        var form = document.getElementById('form');  
        if (!form.checkValidity()) {  
            form.reportValidity();  
        }  
    })  
})();
```

### Cookie 與 Local Storage

為了增進使用者個人化的體驗，有些時候我們會想要存一些個人資訊(當然這涉及安全性問題，這又是另一個議題了。)、使用者習慣等，讓使用者下次造訪我們的應用時，可以為他打造個人化的訊息。

除此之外，在不同頁面之間，我們若需要不透過後端來讓不同頁面間傳遞資料，也會需要儲存想傳遞的資料，在需要的頁面提取出來，最常見的應用就是購物車。我們可以使用 **Web Storage API** 來解決我們儲存資料的問題，而這其中我們最常用的兩個功能便是 **Cookie** 跟 **local storage**。

- **Cookie:** 如同英文意思，一塊小餅乾，可以儲存的資料非常小，大約限制在 **4kb** 左右，是以 **key – value** 對應的方式儲存，與伺服器溝通時，可以夾帶於 **header** 中，可以設置失效的時間，若沒有設置，預設是瀏覽器關閉後失效。
- **Local storage:** 於 **Html5** 中心加入的技術，儲存大小限制一般約為 **5MB**，只會在客戶端保存，不參與與後端間的通訊，除非被清除，不然該資料會永久保存在客戶端。

使用 **storage API** 的時候要特別注意 **XSS** 攻擊的可能性，還不熟悉 **XSS** 攻擊的話，只需要記得妥善使用 **innerHTML** 則可以避免大部分危險。

### 範例 4 – 4

```
(function() {  
    document.getElementById("hi").addEventListener("click", function(){  
        document.getElementById("show_name").innerHTML = document.getElementById("name").value;  
    });  
})();
```

試試看輸入 `<img src=# onerror="alert(1)">` 會發生甚麼事情?

### 利用 Cookie 紀錄使用者登入資訊

**Cookie** 因為有時效性，很適合使用來儲存一些跟認證相關的東西。

### 範例 4 – 5

```

(function() {
    var cookie = document.cookie;
    if (cookie.includes("name")) {
        document.getElementById('message').innerText = "Hello " + getCookie("name");
    }
    else {
        document.getElementById('message').innerText = 'please login';
    }

    document.getElementById("login").addEventListener("click", function(){
        document.cookie = " name=" + document.getElementById("name").value;
    });

    function getCookie(name) {
        var match = document.cookie.match(new RegExp('^| ' + name + '=[^;]+)'));
        if (match) return match[2];
    }
})();

```

除了登入資訊外，另一個很常見到的應用就是儲存使用者訪問狀態，當使用者在一段時間內登入應用時，只有在第一次需要跳一個公告給使用者看。

#### 範例 4-6

```

(function() {
    var cookie = document.cookie;
    if (cookie.includes("check")) {
        alert("notice");
    }

    document.getElementById("cancel").addEventListener("click", function(){
        document.cookie = 'check=true; expires=Fri, 31 Dec 1970 23:59:59 GMT';
    });

    document.getElementById("check").addEventListener("click", function () {
        document.cookie = 'check=true; expires=Fri, 31 Dec 9999 23:59:59 GMT';
    });

    function getCookie(name) {
        var match = document.cookie.match(new RegExp('^| ' + name + '=[^;]+)'));
        if (match) return match[2];
    }
})();

```

#### 利用 Local Storage 紀錄使用者的儲存資訊

因為 **Local Storage** 可以永久的保存數據且可以存放大容量的數據，解決了 **Cookie** 可能會消失以及存取容量小的問題，因此過去很多用 **cookie** 製作的功能如購物車、統計到站次數等的功能，便轉移到 **Local Storage** 上實作。

#### 範例 4-7

```
(function() {  
    var buttons = document.getElementsByClassName("add");  
  
    for(var i = 0; i < buttons.length; i++) {  
        buttons[i].addEventListener("click", addToCart);  
    }  
  
    function addToCart() {  
        var name = this.dataset.name;  
        var count = localStorage.getItem(name);  
        if(count != null) {  
            localStorage.setItem(name, parseInt(count)+1);  
        }  
        else {  
            localStorage.setItem(name, 1);  
        }  
    }  
})();
```

```
(function(){  
    for (var i = 0; i < localStorage.length; i++) {  
        var name = localStorage.key(i);  
        var count = localStorage.getItem(name);  
        document.getElementsByTagName("body")[0].insertAdjacentHTML('beforeend', `<div>${name} 共 ${count}個<div>`);  
    }  
  
    document.getElementById("clear").addEventListener("click", function(){  
        localStorage.clear();  
        location.reload();  
    });  
})();
```

#### 範例 4-8

```

document.getElementsByClassName("list")[0].innerHTML = html;

var buttons = document.getElementsByClassName("add");

for(var i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", addToCart);
}

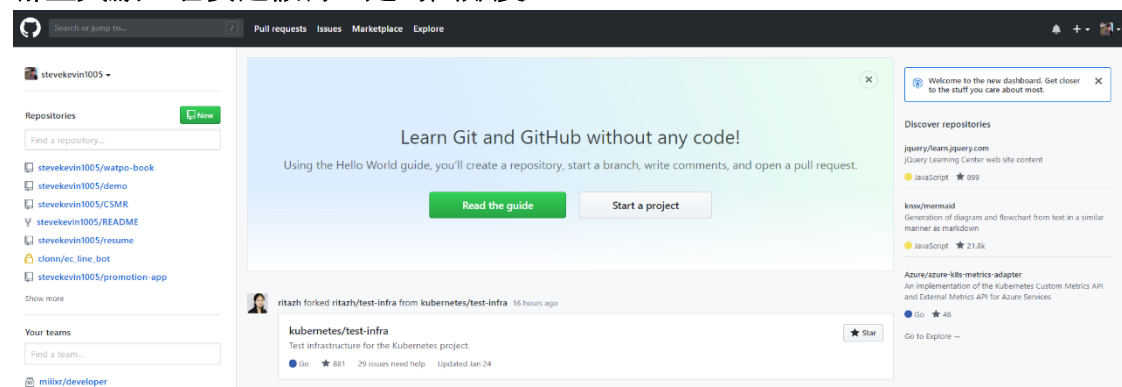
function addToCart() {
    var name = this.dataset.name;
    var list = localStorage.getItem("cart_list");
    if (list != null) {
        list = JSON.parse(list);
        if(list[name] != null) {
            list[name] += 1;
        }
        else {
            list[name] = 1;
        }
        localStorage.setItem("cart_list", JSON.stringify(list));
    }
    else {
        list = {};
        list[name] = 1;
        localStorage.setItem("cart_list", JSON.stringify(list));
    }
}

```

## 第五章： 開放原始碼專案使用

### Github 介紹

**Github** 是一個商業網站(剛被微軟收購)，也是目前全球最大的 **Git Server**。在上面你可以建立自己的公開或是私人專案(如果有五個協同者以上要收費)，同時，它也是開發者最好的履歷，可以在上面看到你曾經做過哪些專案、做過哪些貢獻，若要造假有一定的困難度。



如何找到需要的工具

今天遇到一個需求，我們需要做一個日曆，上面顯示那些使用者在哪一天有事情，如果我們從零開始自己刻，便會花費非常多的時間，這時候我們第一步就是上 **github** 搜尋有沒有相關做好的套件。

在搜尋“calendar”後，我們可以看到有數萬種專案可以參考，這時候我們要先根據我們所使用的語言作進一步篩選。

The screenshot shows the GitHub search interface for the term "calendar". The left sidebar has the "Languages" filter selected, which has narrowed the results to 13,073 repositories. The main area displays a list of repository results, including:

- huanghaibin-dev/CalendarView** (Java, 4.2k stars)
- react-component/calendar** (JavaScript, 1.3k stars)
- traex/CalendarListView** (Java, 1.5k stars)

The "Languages" sidebar lists the following counts:

Language	Count
JavaScript	13,073
Java	7,232
PHP	3,623
Python	3,318
HTML	2,511
Ruby	2,129
CSS	1,405
C#	1,384

篩選後還有一萬多種專案，這時候我們有幾個指標可以看：

- 得到的 **star** 數
- **fork** 數
- 最後的更新時間

The screenshot shows the GitHub search results for "calendar" sorted by "Most stars". The results are as follows:

Repository	Language	Stars
<b>fullcalendar/fullcalendar</b> Full-sized drag & drop event <i>calendar</i>	JavaScript	8.6k
<b>amsul/pickadate.js</b> The mobile-friendly, responsive, and lightweight jQuery date & time input picker.	JavaScript	7.5k
<b>nhnent/tui.calendar</b> A JavaScript <i>calendar</i> that has everything you need.	JavaScript	6.8k

## yarn 介紹及安裝

yarn 是一個模組管理器，是由 Google、Facebook、Exponent 以及 Tilde 聯合推出的資源依賴管理器，其目的在於改善 npm(原本最知名的 node.js 管理器) 且支援 bower，將下載方式改為平行下載，並創建.lock 檔案，確保每一台機器所安裝的版本一致。

## 利用 yarn 及開放原始碼快速開發網頁

yarn install: 安裝 json.pacakge 裡所有設定的依賴

yarn add/remove [package]: 安裝/套件，並儲存在 json.package 中的 dependencies

yarn add/remove [package] -dev: 安裝套件，並儲存在 json.package 中的 devDependencies

yarn global add: 安裝在電腦全域中

## JS-cookie 使用

JS-cookie(<https://github.com/js-cookie/js-cookie>)是一個輕量化簡單的處理 cookie 套件，因原生的 cookie api 並不友善，推薦同學們需要用到 cookie 時就使用此套件來處理 cookie 相關的功能。

使用方法只要記住使用:

- Cookies.set('name', 'value', { expires: 365 }) //設定 cookie
- Cookies.get('name') // 取得 cookie
- Cookies.remove('name') 移除 cookie

利用該套件修改先前使用 cookie 的方式。

## 範例 5-1

```
(function() {  
  if (Cookies.get('name') != undefined) {  
    document.getElementById('message').innerText = "Hello " + Cookies.get('name');  
  }  
  else {  
    document.getElementById('message').innerText = 'please login';  
  }  
  
  document.getElementById("login").addEventListener("click", function(){  
    Cookies.set("name", document.getElementById("name").value);  
  });  
})();
```

再更進一步做出登出的按鈕，再使用者登入後判斷有沒有 cookie 資訊才顯示相對應的資訊，登出實則清出 cookie 資訊就好，這也是最基礎的前後端分離的登入架構。

## 範例 5-2

```
(function() {
  function checkLogin() {
    if (Cookies.get('name') != undefined) {
      let html = `Hello ${Cookies.get('name')} <button id="logout">Log out</button>`;
      document.getElementById('content').innerHTML = html;
      document.getElementById('logout').addEventListener('click', function () {
        Cookies.remove('name');
        checkLogin();
      });
    }
    else {
      let html = `<h3>please login</h3>
        <input type="text" id="name"><button id="login">login</button>`;
      document.getElementById('content').innerHTML = html;
      document.getElementById('login').addEventListener("click", function () {
        Cookies.set("name", document.getElementById("name").value);
        checkLogin();
      });
    }
  }
  checkLogin();
})();
```

## Moment.js 使用

**Moment.js** (<https://momentjs.com/>)，是一個專門處理時間日期相關的套件，時間在軟體處理上是一個很常見也很麻煩的問題，不僅有多種格式需要處理(ISO、RFC、timestamp..etc)，不同國家也有不同的表示方式(2019/1/1、2019-1-1)，以及時區閏年等需要處理，在比較不同時間先後及相差時間也無法直接用運算式加減來處理，而 **Moment.js** 這個套件則幫開發者們把這些痛點全部解決掉，需要處理時間相關問題時，建議直接使用該套件進行開發。

## 範例 5-3

使用方式皆直接用 **moment(...)**...來使用，以下介紹幾個常用的功能:

- **moment([yyyy, mm, dd]).format():**  
將時間格式化若該時間不存在則回傳 **Invalid date**，**format** 內可以填入 **Y、M、D** 等資訊自訂格式
- **moment('要驗證的日期').isBetween('起始日', '截止日'):**  
驗證時間是否在範圍內
- **moment().add(數量, 單位):**  
數量可以輸入正數或是負數來決定加減時間，單位則可參考。



Key	Shorthand
years	y
quarters	Q
months	M
weeks	w
days	d
hours	h
minutes	m
seconds	s
milliseconds	ms

- **moment().from()**、**moment().to()**:  
可以計算某個時間到另一個時間之間的時間差

利用 **moment** 我們可以做一個簡易的打卡系統。

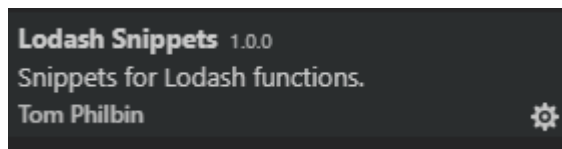
#### 範例 5 – 4 (作者: Vasilis Tsirimokos)

Clock In	Clock Out	Total	星期六 - 1:50:53
• Date: 09/02/2019 Time: 01:50:40	• Date: 09/02/2019 Time: 01:50:42	• Total: 00:00:02	
• Date: 09/02/2019 Time: 01:50:45			<a href="#">WORK</a>

## lodash 使用

lodash (<https://lodash.com/>) 根據官網所述，是一個擴充 Javascript 的套件，裡面提供許多可以增進開發效率的方法，使用方式則是 `_` 要呼叫的函式。

\*Vscode 有提供相關套件幫助開發。



介紹幾個常用到的功能:

### 範例 5 – 5

- `_.range( 開始, 結束):`  
創立一個包含開始到結束所有數字的陣列。
- `_.filter( Json, 函式):`  
根據函式裡面自己設定條件，篩選陣列或是物件並回傳一個陣列包含符合的值。
- `_.has(obj, 目標):`  
判斷物件裡有沒有名為目標的索引。
- `_.random( 開始, 結束):`  
在範圍內隨機一個數字。
- `_.map(Json, 函式):`  
迭代每個數值，並回傳陣列。
- `_.forEach(Json, 函式):`  
迭代每個數值，但不會回傳任何東西。
- `_.remove(陣列, 函式):`  
與 `filter` 相同，根據條件篩選出所要的值，差別在 `remove` 會改變原先的陣列。
- `_.reverse(陣列):`  
將整個陣列反轉。
- `_.union(陣列, 陣列):`  
求聯合。
- `_.uniq(...陣列):`  
求交集。
- `_.xor(...陣列):`  
求捕集。
- `_.uniqueId([prefix=""]):`  
產生一個唯一的 ID，若有給 `prefix` 則會在 ID 前面加上。
- `_.sampleSize(Json, 數量):`  
隨機取出指定數量的數值。

## 第六章： 高品質程式碼撰寫

運算式==和===差異

請使用開發者工具 **Console** 測試以下運算式：

- `"1024" == 1024`
- `"1024" === 1024`
- `0 == false`
- `1 == true`
- `"" == 0`
- `undefined == null`
- `0 === false`
- `1 === true`
- `"" === 0`
- `undefined === null`

使用 `==` (一般相等) 時，會將比較值轉換成同型別後比較。轉換後（可能一個或兩個都被轉換），接著進行的幾乎和嚴格比較（`===`）一樣。一般相等會對稱：`A == B` 等同 `B == A`，無論 `A` 和 `B` 是什麼。（除了型別轉換的順序）

嚴格相等比較兩個值，而被比較的兩個值都不會轉換成其他型別。如果值是不同型別，就會被視為不相等。如果兩值型別相同但不是數字，若值相同，則為相等。此外，如果兩個值皆為數字，只要他們是 **NaN** 以外的同一值，或者 `+0` 和 `-0`，則為相等。

ref: <https://developer.mozilla.org/>

## 三元運算子

條件(三元)運算子是 JavaScript 中唯一需要三個運算元的運算子。這個運算子接受兩個運算元作為值且一個運算元作為條件。

- 條件 ? 值 1 : 值 2

如果 條件 為 **true**，運算子回傳 值 1，否則回傳 值 2。

### 範例 6-1

```
(function () {  
    if (new Date().getHours < 12 ) {  
        document.getElementById("content").innerText = "上午";  
    }  
    else {  
        document.getElementById("content").innerText = "下午";  
    }  
  
    new Date().getHours < 12 ? document.getElementById("content").innerText = '上午' : document.getElementById("content").innerText = "下午";  
})();
```

## 位元運算子

位元運算子 把運算元當作 32 位元的集合來看待 (0 和 1)，而不是十進位，十六進位，或八進位。例如，十進位數字 9 以二進位表示就是 1001。位元運算子將運算元以上述二進位的形式處理，但是回傳 Javascript 中的數字類型值。

### 範例 6-2

#### 位元運算子

運算子	用法	描述
位元 AND	<code>a &amp; b</code>	回傳兩個運算元對於每個bit做AND的結果。
位元 OR	<code>a   b</code>	回傳兩個運算元對於每個bit做OR的結果。
位元 XOR	<code>a ^ b</code>	回傳兩個運算元對於每個bit做XOR的結果。
位元 NOT	<code>~ a</code>	將運算元中的每個bit反轉(1->0,0->1)。
左移	<code>a &lt;&lt; b</code>	將 a 的每個bit向左移動 b 個bits，空餘的位數以0填滿。
有號右移	<code>a &gt;&gt; b</code>	將 a 的每個bit向右移動 b 個bits，空餘位數以最高位補滿。
以0填充的右移	<code>a &gt;&gt;&gt; b</code>	將 a 的每個bit向右移動 b 個bits，空餘的位數以0填滿。

ref: <https://developer.mozilla.org/>

## 避免使用全域變數

當網頁應用越來越大型時，裡面所使用的 JS 檔案不會只有一個，且很可能會使用到外部的一些專案，這時候若我們在全域宣告了一個 **user** 變數，過很長一段時間後，在別的地方又重新宣告了一次 **user** 變數，便會覆蓋掉原本的變數，若實際上遇到需求需要用到全域變數/函式的時候(共用的處理套件)，建議將其均綁在一個變數底下。

## 函式內的程式碼長度

心理學研究指出，人們一次能看的文章長度最多七行，當一個函式內的程式碼長度太長時，會造成維護上困難，不論是同事或是自己未來回來看的時候，都會看不懂到底在幹嘛，因此當函式開始越來越肥大的時候，最好將其拆分。

e.g.

```
function register() {  
  var user = {};  
  user.name = document.getElementById("name");  
  
  :  
  :  
  :  
  :  
  :  
  axios({ : });  
}
```

```
function register() {  
  var user = getUser();  
  register(user);  
}
```

## 函式內的程式碼(圈)複雜度

程式碼複雜度泛指該程序內會有幾種可能性，若其中有一個 **if else** 則複雜度為 **2**，以此類推。程式的複雜度越高時，後續維護上以及寫測試上會造成困難度增加，更會造成執行速度下降，因此經常我們會要求複雜度不要超過 **10**，隨著開發時間的增加，複雜度越來越多是必然的，這時候我們可以藉由 **JS** 的物件鑑值是哈希表的特性來降低複雜度提升執行速度。

e.g.

```
if (a === 'dog') {  
  .....  
} else if (a === 'cat') {  
  .....  
} else if (a === 'mice') {  
  .....  
} else if (a === 'duck') {  
  .....  
} else if (a === 'pig') {  
  .....  
} ...
```

```
var animals = {  
  'dog': function() {},  
  'cat': function() {},  
  'mice': function() {},  
  'duck': function() {},  
  'pig': function() {},  
  ...  
};  
  
animals[a]();
```

## 函式的變數個數

一個函式可能會牽扯到多個變數，但是當變數越來越多時，不只會造成後續維護理解的困難，更會增加寫出 **bug** 的可能性(不知道到底哪個變數要填在哪個位置)，因此當變數變多時，適當的利用物件包裝起來是必要的。

```
function setUserDetail(name, address, email, sex, age, tall) {  
    :  
    :  
    :  
    :  
}
```

```
var user = {  
    "name": "",  
    "address": "",  
    "email": "",  
    "sex": "",  
    "age": "",  
    "tall": ""  
}  
  
function setUserDetail(user) {  
    :  
    :  
    :  
    :  
}
```

## 程式碼打包壓縮工具

當應用程式日漸肥大後，引用的套件也會跟著變多，受於瀏覽器一次最多建立六條連線以及頻寬的限制，實作上我們會將所有用到的套件打包並壓縮成一個 **js** 檔案，增進載入效率，比較有名的工具有 **Parcel**、**webpack**、**browserify**，除了打包壓縮程式碼以外，為了配合比較低階的瀏覽器(萬惡的 **IE**)，我們也會藉由這些工具外掛 **babel** 來降低 **JS** 程式碼的版本。

ref: <https://babeljs.io/>

## 程式碼檢查工具

從事軟體開發的人往往會有一種經驗，自己寫的程式碼，一個月後回過頭來看卻無比陌生，更何況是他人寫的程式碼，因此相對應的程式碼風格檢查工具也隨之出現，這裡介紹一款名為 **ESLint** 的檢查工具，而他最主要的功能有：

- 找出語法錯誤  
沒宣告變數就拿來用、少了括號等等常見的語法錯誤。
- 確保你遵循最佳實踐  
不使用全域變數、建議使用 `===` 而非 `==`、不使用 `eval` 等。
- 提醒你刪掉多餘的程式碼  
有些變數宣告了卻沒有使用，空的 `class` 或 `function`，`import` 卻沒有使用等等。
- 統一基本的程式碼風格  
分號加與否，一個 `tab` 是幾個空格，使用單引號或雙引號，使用 `tab` 或空格等等。

1. 先以 `yarn init / npm init` 配置基本專案設置檔
2. 先以 `yarn add global eslint / npm install -g eslint` 安裝
3. 在專案的目錄下輸入 `eslint -init` 初始化配置，一開始沒有特殊需要建議以 `airbnb`、`google`、`standard` 挑一個風格來用即可
4. 若跑 `eslint` 有出現錯誤 `npm install -g / yarn add global` 相對應套件
5. `.eslintrc.js` 增加 `env: { "browser": true }`

### 範例 6 - 3

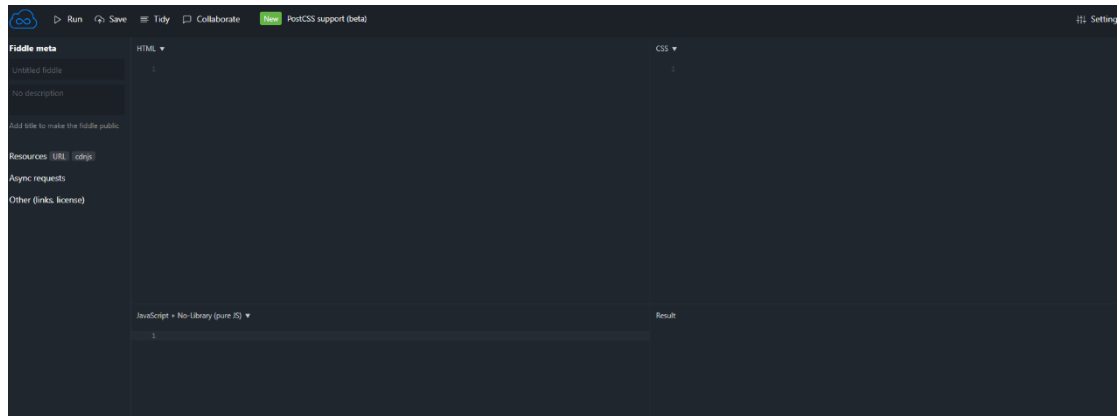
試著輸入 `eslint main.js` 看看有沒有成功檢查吧！



## 實用工具

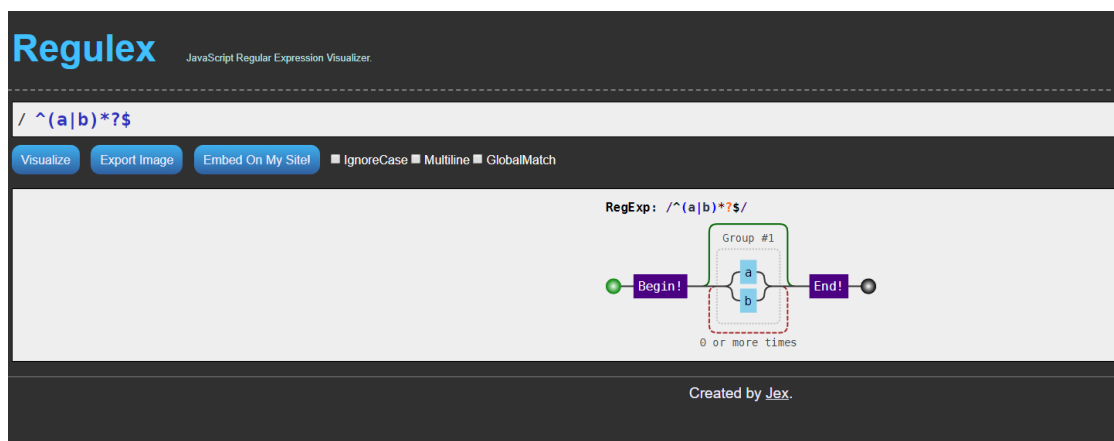
### 1. JS 線上測試&展示

ref: <https://jsfiddle.net/>



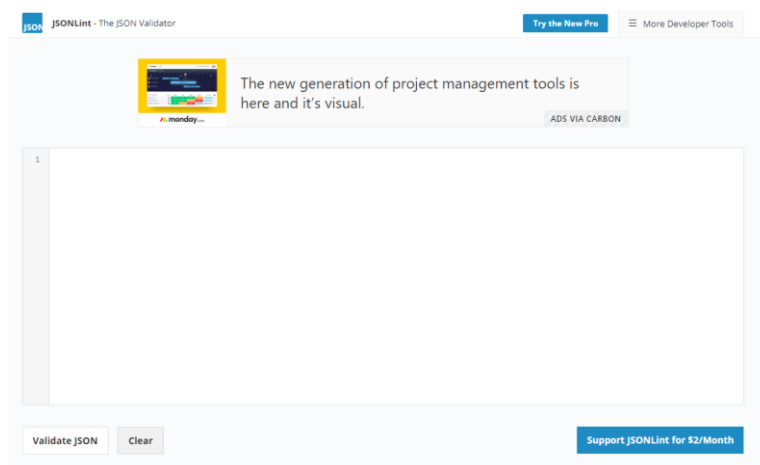
### 2. 圖形化正規式

ref: <https://jex.im/regulex>



### 3. JSON 驗證及檢視

ref: <https://jsonlint.com/>

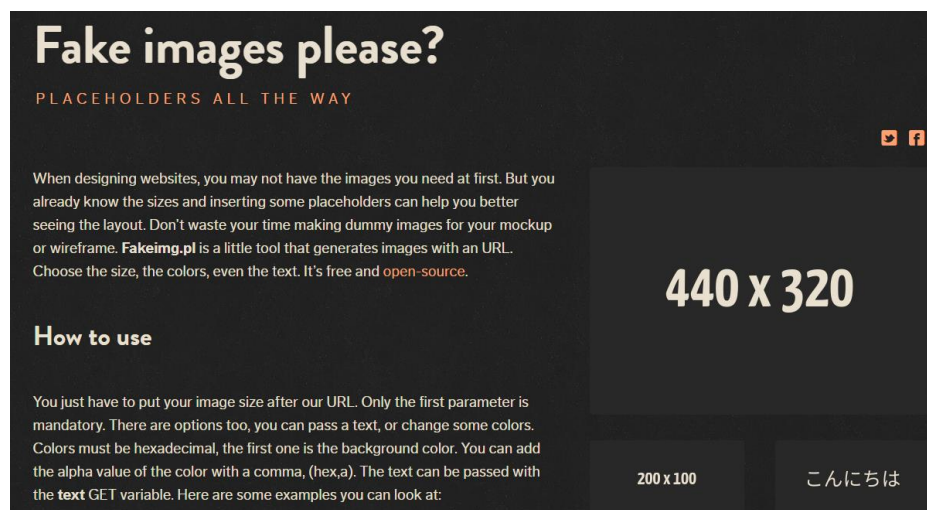


#### 4. JSON VIEW extension



#### 5. 假圖產生器

ref: <https://fakeimg.pl/>



#### 6. 測量工具

ref: <https://picpick.app/zh-tw/>

