JSON

JSON（JavaScript Object Notation）是一種由道格拉斯·克羅克福特(Douglas Crockford)設計、輕量級的資料格式。它以文字為基礎，且易於閱讀。JSON 資料格式與語言無關，脫胎於 JavaScript，但目前很多程式語言都支援 JSON 格式資料的生成和解析。JSON 的官方 MIME 類型是 application/json，副檔名是 .json。

JSON 是物件、陣列、數字、字串、布林值、還有 null 的語法。它以JavaScript語言中的物件字面為基礎設計，但並不完全相同，JSON並不完全是JavaScript的子集(參考)。

JSON格式

JSON主要以"鍵/值(key/value)"的格式來描述數值，一組數值稱為一個鍵值對(key/value pair)

- 鍵(Key): 使用雙引號包含的字串
- 值(Value): 可以是字串(string)、數字(number)、布林(boolean)、陣列(array)或物件(object)
- 鍵值對(Key/Value Pair): 鍵與值間使用冒號(:)分隔，鍵值對之間使用逗號(,)分隔

如下面的範例:

```
"foo" : "bar"
```

陣列

```
"foo" : {
  "bar" : "Hello",
  "baz" : [ "quuz", "norf" ]
}
```

物件

```
"foo" : {
  "bar" : "Hello"
}
```

JavaScript中的相關方法

JSON.parse

將一個JSON字串轉變為JavaScript中的數值，如下範例:

```
JSON.parse('{}');              // {}
JSON.parse('true');            // true
JSON.parse('"foo"');           // "foo"
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');            // null
```

JSON.stringify

將一個JavaScript中的數值轉變為JSON字串，如下範例:

```
JSON.stringify({});                  // '{}'
JSON.stringify(true);                // 'true'
JSON.stringify('foo');               // '"foo"'
JSON.stringify([1, 'false', false]); // '[1,"false",false]'
JSON.stringify({ x: 5 });            // '{"x":5}'
JSON.stringify({ x: [10, undefined, function(){}, Symbol('')] });
// '{"x":[10,null,null,null]}'
```

AJAX與XMLHttpRequest

AJAX這個技術名詞的出現是在十年前(2005)，其中內容包含XML、JavaScript中的XMLHttpRequest物件、HTML與CSS等等技術的整合應用方式，這個名詞並非專指某項特定技術或是軟體，Google在時所推出的Gmail服務與地圖服務，獲得很大的成功，當時這個技術名詞以此作為主要的案例說明。實際上這個技術的實現是在更早之前(2000年之前)，一開始是微軟公司實作了一個Outlook與郵件伺服器溝通的介面，後來把它整合到IE5瀏覽器上。在2006年XMLHttpRequest正式被列入W3C標準中，現在已被所有的瀏覽器品牌與新版本所支援。

所謂的AJAX技術在JavaScript中，即是以XMLHttpRequest物件(簡稱為XHR)為主要核心的實作。正如它的名稱，它是用於客戶端對伺服器端送出httpRequest(要求)的物件，使用的資料格式是XML格式(但後來JSON格式才是最為流行的資料格式)。流程即是建立一個XMLHttpRequest(XHR)物件，打開網址然後送出要求，成功時最後由回調函式處理伺服器傳回的Response(回應)。整體的流程是很簡單的，但經過這麼長久的使用時間(11年)，它在使用上產生不少令人頭痛的問題，例如:

- API設計得過於高階(簡單)，所有的輸出與輸入、狀態，都只能與這個XHR物件溝通取得，進程狀態是用事件來追蹤。
- XHR是使用以事件為基礎(event-based)的模組來進行異步程式設計。
- 跨網站的HTTP要求(cross-site HTTP request)與CORS(Cross-Origin Resource Sharing)不易實作。
- 對非文字類型的資料處理上不易實作。
- 除錯不易。

XHR在使用上都是像下面的範例程式碼這樣，其實你可以把它視作一種事件處理的結構，大小事都是依靠XHR物件來作，語法中並沒有把每件事情分隔得很清楚，而比較像是擠在一團:

```
function reqListener() {
  const data = JSON.parse(this.responseText);
  console.log(data)
}

function reqError(err) {
  console.log('Fetch Error :-S', err)
}

const oReq = new XMLHttpRequest();
oReq.onload = reqListener
oReq.onerror = reqError
oReq.open('get', './sample.json', true)
oReq.send()
```

在今天瀏覽器功能相當強大，以及網站應用功能複雜的時代，XHR早就已經不敷使用，它在架構上明顯的有太多的問題，尤其在很多功能的應用情況，程式碼會顯得複雜且不易維護。除非你是有一定要使用原生JavaScript的強迫症，要不然現在要作AJAX功能時，程式設計師並不會使用原生XHR物件來撰寫，大部份時候會使用外部函式庫。因為一個AJAX的程式，並不是單純到只有對XHR的要求與回應這麼簡單，例如你可能會對伺服器要求一份資料，當成功得到資料後，後面還有需要進一步的資料處理流程，這樣就會涉及到異步程式的執行結構，原生XHR並沒有提供可用的方式，它只是單純的作與伺服器互動那件事而已。

XHR Level 2(第2級)

XHR並不是沒有在努力進步，在約5年前已經有制定XHR的第2級新標準，但它仍然與原有XHR向下相容，所以整體的模型架構並沒有重大的改變，只是針對問題加以補強或是擴充。目前XHR第2級在9成以上的瀏覽器品牌新版本都已經支援全部的功能，除了IE系列要版本10之後才支援，以及Opera Mini瀏覽器完全不支援，還有一小部份功能在不同瀏覽器上實作細節會有所不同。XHR第2級(5年前)相較於原有的XHR(11年前)多加了以下的功能，這也是現在我們已經可以使用到的XHR的新特性:

- 指定回應格式
- 上傳文件與blob格式檔案

- 使用FormData傳送表單
- 跨來源資源共享(CORS)
- 監視傳輸的進程

不過，XHR第2級的新標準並沒有太引人注目的新功能，它比較像是解決長期以來的一些嚴重問題的補強版本。而且，XHR在原本上的設計就是這樣，常被批評的是它的語法結構不論在使用與設定都相當的零亂。補強或擴充都還是跳脫不了基本的結構，現今是HTM5、CSS3與ES6的時代，有許多新的技術正在蓬勃發展，說句實在話，就是XHR技術已經舊掉了，當時的設計不符合現在時代需求了，這也無關對或錯。

jQuery中的作法

外部函式庫例如jQuery很早就看到XHR物件中在使用的問題，使用像jQuery的函式庫來撰寫AJAX相關功能，不光是在解決不同瀏覽器中的不相容問題，或是提供簡化語法這麼簡單而已。jQuery它擴充了原有的XHR物件為jqXHR物件，並加入類似於Promise的介面與Deferred Object(延遲物件)的設計。

為何要加入類似Promise的介面？可以看看它的說明中，是為了什麼而加入的？

> 這些方法可以使用一個以上的函式傳入參數，當 `$.ajax()` 的要求結束時呼叫它們。這可以讓你在單一個(request)要求中指定多個callbacks(回調)，甚至可以在要求完成後指定多個callbacks(回調)。 ~譯自jQuery官網The jqXHR Object

原生的XHR根本就沒有這種結構，Promise的結構基本上除了是一種異步程式設計的架構，它也可以包含錯誤處理的流程。簡單地來說，jQuery的目標並不是只是簡化語法或瀏覽器相容性而已，它的目標是要"**取代以原生XHR物件的AJAX語法結構**"，雖然本質上它仍然是以XHR物件為基礎。

jQuery作得相當成功，十分受到程式設計師們的歡迎，它的語法結構相當清楚，可閱讀性與設定彈性相當高，常讓人忘了原來的XHR是有多不好使用。在Promise還沒那麼流行的前些年，裡面就已經有類似概念的設計。加上現在的新版本(3.0)已經支援正式的Promise標準，說實在沒什麼理由不去使用它。以下是jQuery中ajax方法的範例:

```
// 使用 $.ajax() 方法
$.ajax({

    // 進行要求的網址(URL)
    url: './sample.json',

    // 要送出的資料 (會被自動轉成查詢字串)
    data: {
        id: 'a001'
    },

    // 要使用的要求method(方法)，POST 或 GET
    type: 'GET',

    // 資料的類型
    dataType : 'json',
})
  // 要求成功時要執行的程式碼
  // 回應會被傳遞到回調函式的參數
  .done(function( json ) {
     $( '<h1>' ).text( json.title ).appendTo( 'body' );
     $( '<div class=\'content\'>').html( json.html ).appendTo( 'body' );
  })
  // 要求失敗時要執行的程式碼
  // 狀態碼會被傳遞到回調函式的參數
  .fail(function( xhr, status, errorThrown ) {
    console.log( '出現錯誤，無法完成!' )
    console.log( 'Error: ' + errorThrown )
    console.log( 'Status: ' + status )
    console.dir( xhr )
  })
  // 不論成功或失敗都會執行的回調函式
  .always(function( xhr, status ) {
    console.log( '要求已完成!' )
  })
```

把原生的XHR用Promise包裹住，的確是一個好作法，有很多其他的函式庫也是使用類似的作法，例如axios與SuperAgent，相較於jQuery的多功能，這些是專門只使用於AJAX的函式庫，另外這些函式庫也可以用在伺服器端，它們也是有一定的使用族群。

相關工具/技術

- json-server: 這是一個架在Node.js上的用於測試JSON格式資料的伺服器，而且是REST API，裡面用了lowdb，所以其實它是一個具有資料庫寫入、讀取、查詢的功能的伺服器。這個工具在Github上有1萬8千個星，如果你要開發像我們這種JavaScript應用，而且是用json交換資料的應用，學會用這個工具，保証讓你開發的日子輕鬆很多。
- Postman: 這工具則是一個客戶端程式，也有Chrome的外掛擴充版本，這是用來模擬由客戶端發送各種要求(Request)用的，像POST、GET、PUT...等等。也就是例如在還沒開始寫程式前，先作測試，看你的資料傳到伺服器(json-server)上能不能正確查詢到資料，或是新增資料等等。當然它也有很多其他的功能，不過我只用這部份而已。
- Fetch API: 這是一個API，是HTML5標準中的新特性，當然它是一個實作在JavaScript的新API，Fetch API是一個完全以為基礎的API，Fetch並不是一個單純的XHR擴充加強版或改進版本，它是一個用不同角度思考的設計，雖然是可以作類似的事情。此外，Fetch還是基於Promise語法結構的，而且它的設計足夠低階，這表示它可以依照實際需求進行更多彈性設定。相對於XHR的功能來說，Fetch已經有足夠的相對功能來取代它，但Fetch並不僅於此，它還提供更多有效率與更多擴充性的作法。
- REST: 表現層狀態轉換（英語：Representational State Transfer，縮寫：REST）是Roy Thomas Fielding博士(HTTP協議、Apache網站伺服器共同創立者之一)於2000年在他的博士論文中提出來的一種全球資訊網軟體架構風格，目的是便於不同軟體/程式在網路（例如網際網路）中互相傳遞資訊。表現層狀態轉換是根基於超文字傳輸協定（HTTP）之上而確定的一組約束和屬性，是一種設計提供全球資訊網絡服務的軟體構建風格。符合或相容於這種架構風格（簡稱為 REST 或 RESTful）的網路服務，允許使用者端發出以統一資源標識符存取和操作網路資源的請求，而與預先定義好的無狀態操作集一致化。相較於SOAP、XML-RPC更為簡潔容易使用，也是眾多網路服務中最為普遍的API格式，像是Amazon、Yahoo!、Google等提供的API服務均有REST介面。

更多參考：什麼是REST? 認識 RESTful API 路由語義化設計風格與

json-server

> 註：json-sever是模擬一個"伺服器"+"資料庫"的測試環境用，而且是有REST API的網站伺服器

以下說明json-server的簡單用法。

首先安裝它，直接用在命令列工具(終端機)中用npm工具就可以安裝，裝在全域即可:

```
npm install -g json-server
```

或使用yarn工具來安裝：

```
yarn global add json-server
```

建立一個資料夾，內有一個先建立一個 `db.json` 檔案，與JS中的物件格式很像，但它是個純文字檔而已。它的內容如下所示:

```
{
  "items": [
    {
      "id": 1482513391121,
      "title": "聽演唱會",
      "isCompleted": true
    }
  ]
}
```

啟動它也是用命令列工具(終端機)的指令如下所示:

```
json-server --watch --port 5555 db.json
```

這個指令代表要啟動一個在埠號為5555的json-server伺服器，之後就可以用瀏覽器打開"http://localhost:5555/items"，json-server伺服器的網址後加上items會自動只列出items裡面的json資料，它稱之為Routes(路由):

> 註: 我們開發測試React應用的是3000埠，json-server伺服器是5555埠，相當於在同一個電腦啟動了兩個不同的伺服器。

> 註: 上面的5555是隨便取的埠號，你要用8888或9999也可以，不過埠號有一定的範圍就是。

因為json-server伺服器是個用REST API的伺服器，而且又有小型資料庫，又該如何新增、讀取、更新...裡面的資料？

首先你要先理解REST是什麼，也就是在傳資料時的method各自分別要作不同的事情之用，例如下面幾個:

- POST = 新增
- GET = 讀取
- PUT = 更新
- DELETE = 刪除
- PATCH = 取代部份資料

下面就幾種常用的要作某些事情的範例:

獲得資料

獲得所有資料，預設是用id由小至大(ASC)排序:

> GET /items

用id排序，由大至小:

> GET /items?_sort=id&_order=DESC

獲得單筆的資料:

> GET /items/4

新增一筆資料

> POST /items

資料範例:

```
{
  "id": 4,
  "title": "44444",
  "isCompleted": false
}
```

更新一筆資料

> PUT /items/4

`PUT` 更新是需要把有更新與沒更新的所有資料也要送給伺服器才行的。

資料範例:

```
{
  "id": 4,
  "title": "4321",
  "isCompleted": false
}
```

刪除一筆資料

> DELETE /items/4