Node.js教學

https://nodejs.org

Part 01 基本概念

Wei-Tsung Su

Changelog

V1.0　2018/05/18　完成第1版

簡介

Node.js (簡稱Node)是一個JavaScript執行環境(runtime)，具有幾個特點：

以Google所開發的JavaScript引擎V8為基礎

使用事件驅動(event-driven)與非阻斷式I/O (non-blocking I/O)模型，效率高

擁有一個超大的套件生套圈npm

歷史

Ryan Dahl原本只是想要開發一個支援非阻斷式I/O的網站開發語言(所以起初命名為web.js)，但後來發現功能更多...

2009年CommonJS規範發佈，同年5月Node.js於Github釋出

Node適用的應用

以下內容節錄自Mapping the Journey對Ryan Dahl的專訪

“Node.js的非阻斷式I/O在JavaScript這種沒有執行緒(threads)的環境中表現得非常出色，但問題是需要追蹤許多無名的回呼函式(callback)才能知道程式在做什麼！不過，這個問題已在ES7中利用async與await等關鍵字獲得改善。”

“Node.js不但在客戶端大放異彩，也可以用來開發伺服端服務的周邊任務如Browserify)或小型伺服端服務，但如果要開發如DNS這種大型的分散式系統，我不會選用Node.js”

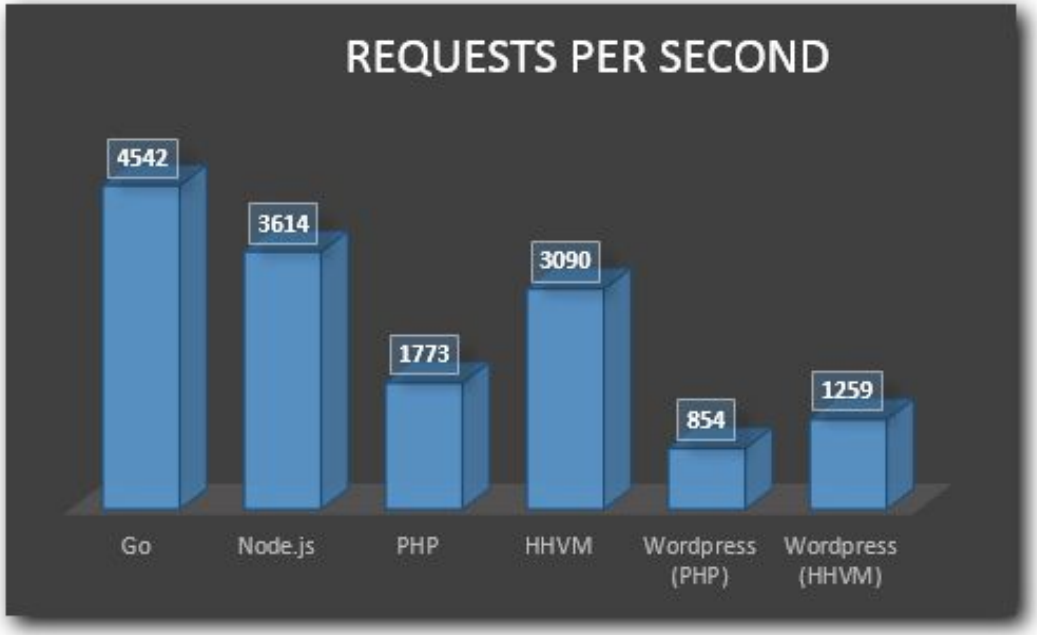
使用(或曾使用)Node.js技術的公司/產品

效能比較

Go效能好，但學習門檻高

Node.js比HHVM好
(PHP7與HHVM效能接近)

註:HHVM為Facebook開發的高效能PHP虛擬機



http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/
2018/1/11

Node安裝與版本管理

Node Version Manager (nvm)
https://github.com/creationix/nvm

使用nvm管理Node版本

由於不同的套件(packages)對Node版本的相依性不同，所以在開發過程中可能需要切換版本。

例如：開發初期使用的套件A只要求Node v6.13.0版本以上，但開發後期需要額外使用到的套件B卻要求Node v8.9.4版本以上，這時就需要安裝新的Node版本。

使用nvm的好處就是可以很容易地安裝與切換Node版本。

安装nvm

Windows

- 參考https://github.com/coreybutler/nvm-windows

Linux / Mac

- 參考https://github.com/creationix/nvm#install-script

使用nvm

查看nvm的使用方式

```
$ nvm
```

查看目前所有Node版本

```
$ nvm ls-remote
```

安裝特定版本(8.9.4)的Node到本機

```
$ nvm install 8.9.4
```

查看所有已安裝與使用的Node版本

```
$ nvm list
```

切換Node版本(8.9.4)

```
$ nvm use 8.9.4
```

移除本機上特定版本(8.9.4)的Node

```
$ nvm uninstall 8.9.4
```

Workshop

練習使用nvm

1. 安裝nvm
2. 查詢所有Node版本
3. 以nvm安裝兩個以上的Node版本
4. 切換Node版本並確認是否成功

Node程式撰寫與執行

以PrintFile專案為例

練習：PrintFile專案

專案目標

開啟檔案並列印檔案內容

外部套件

無

專案目錄結構

PrintFile

|_____ *hello.txt*

|_____ *file.js*

準備文字檔

準備專案目錄(PrintFile)並產生文字檔(hello.txt)包含內容如下

```
{
    "id":1,
    "data":[
        {"name":"temperature","value":28},
        {"name":"humidity","value":0.5}
    ]
}
```

專案目錄結構

PrintFile

|_____ ***hello.txt ←***

|_____ *file.js*

撰寫程式(file.js)

在專案目錄(PrintFile)中產生程式檔(file.js)並撰寫程式如下

```
1.  const file = require('fs') //檔案系統(fs)模組不需另外安裝
2.
3.  //當readFile()讀取檔案結束後，會執行回呼函式function
4.  file.readFile('./hello.txt', function(err, data) {
5.      if(err) throw err
6.      console.log(data.toString())
7.  })
8.  console.log('Reading file ...')
```

在專案目錄中執行：`$ node file.js`

專案目錄結構

PrintFile

|_____ *hello.txt*

|_____ ***file.js*** ←

ES6: 箭頭函式

箭頭函式是ES6標準中一種簡化函式表達的語法

函式的原始寫法

```
function(x, y) { return x + y }
```

箭頭函式的寫法

```
(x, y) => { return x + y }
```

Workshop

練習使用箭頭函式

將PrintFile專案中的函式改寫為箭頭函式並執行看看

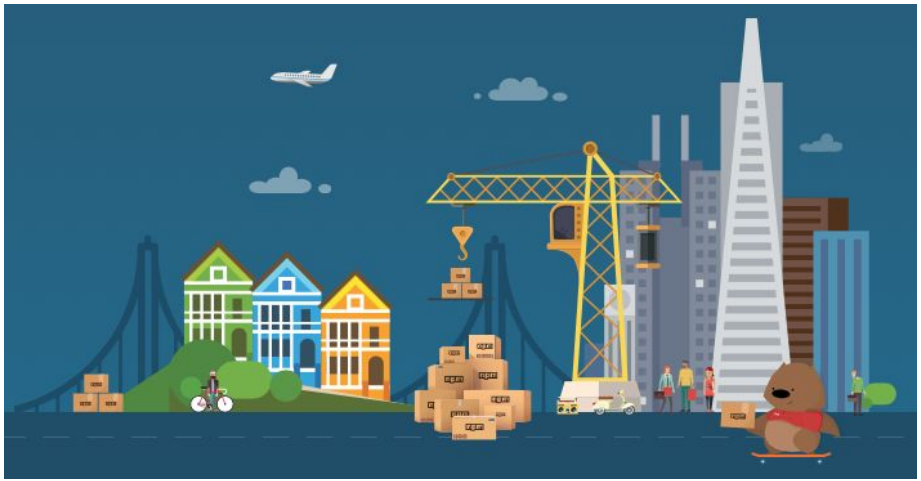
套件管理

Node Package Manager (npm)
https://www.npmjs.com/

Node的套件花花世界 - npm

在Node的世界裡，套件(package)就像是許多用來蓋房子的材料，如果能夠善用這些不同功能的材料就可以堆疊出更華麗的大樓。

npm是在專案中用來管理Node套件的工具。



使用npm

查看npm的使用方式

```
$ npm
```

安裝套件

```
$ npm install [套件名稱]
```

移除套件

```
$ npm uninstall [套件名稱]
```

查看套件版本

```
$ npm info [套件名稱] version
```

產生套件說明檔(package.json)

```
$ npm init
```

根據套件說明檔安裝套件

```
$ npm install
```

練習：ExeCmd專案

專案目標

執行本機端的執行檔並顯示輸出結果

外部套件

node-cmd

專案目錄結構

ExeCmd
|_____ node_modules
| |_____ node-cmd
|_____ *cmd.js*

安裝套件

到npm網站搜尋發現node-cmd套件的功能符合需求

在專案目錄(ExeCmd)中安裝node-cmd套件(需要網路)

```
$ npm install node-cmd
```

結束後會在專案目錄下產生套件目錄(node_modules)並包含安裝好的node-cmd套件。

專案目錄結構

撰寫程式(cmd.js)

在專案目錄(ExeCmd)中產生程式檔(cmd.js)並撰寫程式如下

```
1.  const cmd = require('node-cmd')
2.  var usrcmd = (process.argv[2]) ? process.argv[2]:'pwd'
3.
4.  cmd.get(user_cmd, (err, data, stderr)=> {
5.      if(err) throw err
6.      console.log('ExeCmd Result:\n' + data)
7.  })
```

在專案目錄中執行：`$ node cmd.js pwd`

專案目錄結構

ExeCmd

|_____ node_modules

| |_____ node-cmd

|_____ ***cmd.js*** ←

套件說明檔

用來描述專案的檔案(包含相依的外部套件)
https://docs.npmjs.com/files/package.json

套件說明檔

套件說明檔(package.json)是用來描述專案的檔案。因為專案說明檔包含專案會使用到的外部套件，所以衍生了一個方便的功能。

問題：當專案使用的外部套件越來越多時，套件目錄中的檔案就會越來越多(大)而造成在複製專案時需要花費許多時間。所以，可不可以只複製跟自己專案有關的檔案而不複製外部套件?

答案：可以。因為套件說明檔中包含了會使用到的外部套件，所以npm可以根據套件說明檔內容來重新下載所有外部套件。

產生套件說明檔

在專案目錄(ExeCmd)中執行下列命令產生套件說明檔。

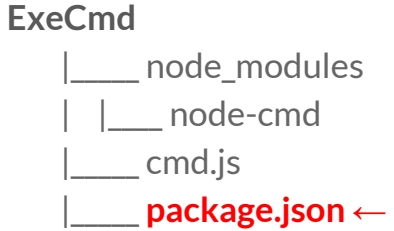
```
$ npm init
```

接著輸入名稱、版本、描述、進入點、測試命令(如果有)、git位址(如果有)、關鍵字、作者、授權方式等資訊後就會自動在專案目錄下產生套件說明檔。

將套件目錄(node_modules)刪除後執行下列命令下載套件

```
$ npm install
```

專案架構



Workshop

運用套件與套件說明檔

1. 在npm網站中選擇一個套件
2. 利用npm安裝此套件
3. 撰寫程式運用此套件
4. 產生套件說明檔
5. 撰寫投影片介紹此套件

Q & A

補充資料

非阻斷式I/O

```
1.  const fs = require('fs');
2.
3.  fs.readFile('/hello.txt', (err, data) => {
4.      //以下程式要等檔案讀取完成(或出現錯誤)後才回被執行
5.      if (err) throw err;
6.      console.log(data);
7.  });
8.  // moreWork(); 這裡的程式碼會馬上接著執行
```

變數定義

JavaScript是弱型別語言。

定義變數時，可以使用var、let、與const等修飾字。(三個的差異之後再談)

使用變數時，如果該變數沒有被定義，則會自動以var修飾字定義變數。

```
1.  var value = 100  //定義變數value
2.  cost = 10        //定義變數cost
3.  cost = 20        //使用變數cost
4.  console.log(value/cost)
```

變數範圍

在JavaScript中，區塊是由一組{ }所包圍的區域，而且允許巢狀區塊。

當變數定義沒有被任何區塊包圍時，稱為全域變數。反之，稱為區域變數，其區域範圍為所在區塊。

當全域變數與區域變數同名時，會使用區域變數。

```
1.  var cost = 100          //定義全域變數
2.  function fun() {
3.    cost = 50             //使用全域變數
4.    {
5.      var cost = 25       //定義區域變數
6.      console.log(cost)
7.    }
8.    console.log(cost)
9.  }
10. fun()
11. console.log(cost)
```

變數定義修飾字var

以var定義的變數，在其區塊及區塊內的巢狀區塊都是有效的。

例如，程式碼第2行以var修飾字定義了cost變數，所以第4行以var定義的cost變數其實等於是使用第2行定義的cost變數。

```
1.  function fun() {
2.    var cost = 50        //cost
3.    {
4.      var cost = 25      //cost
5.      console.log(cost) //cost
6.    }
7.    console.log(cost)    //cost
8.  }
9.  fun()
```

變數定義修飾字let

以let定義的變數，只在其區塊內是有效的。

例如，因為第4行是以let定義cost變數，所以只在第3-6行這個區塊內有效。

```
1.  function fun() {
2.    var cost = 50          //cost1
3.    {
4.      let cost = 25        //cost2
5.      console.log(cost)    //cost2
6.    }
7.    console.log(cost)      //cost1
8.  }
9.  fun()
```

變數定義修飾字const

以const修飾字定義的變數，在其變數範圍內無法被變更。

另外，const跟let修飾字一樣只在其區塊內是有效的。

```
1.  function fun() {
2.    var cost = 50
3.    {
4.      const cost = 25
5.      console.log(cost)
6.      cost = 50            //NG
7.    }
8.    cost = 100             //OK
9.    console.log(cost)
10. }
11. fun()
```