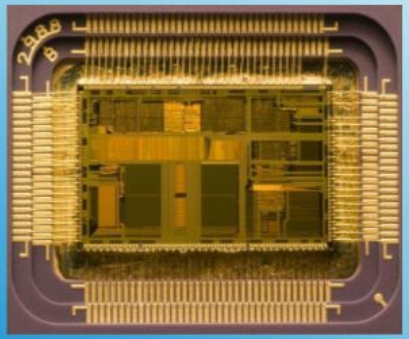
網路程式設計

Web Programming



- *HTML*
- *CSS*
- *JavaScript*
- *Node.js*

作者：陳鍾誠

Web 程式簡介

Network、Internet 與 Web

在英文當中、Network、Internet 與 Web 這些名詞是區分得很清楚的，但是對於中文使用者而言，這三個名詞卻很難釐清，不過對於程式人員而言，區分這幾個名詞是很重要的是，特別是 Internet 與 Web 這兩個名詞，首先讓我們來看看這些名詞的差異。

英文中的 Network 泛指任何的網路，包含像公路也叫做 Traffic Network，因此即使在電腦領域，Network 仍然泛指任何網路，包含區域網路、廣域網路、Internet、Ethernet 等等都是 Network。

Internet 是指 1960 年代美國國防部 (Department of Defense, DoD) 所發展出來的那個網路，原本稱為 ARPANET，後來延伸到全世界之後，就稱為 Internet 了。

1990 年 Tim Burner Lee 創造了 HTML 、URL、與全世界第一個 WebServer，從此開始創造出一個基於 Internet 的網路架構，這個網路架構被稱為 World Wide Web (簡稱為 Web 或 WWW)，後來在 1992 年當時 Marc Andreessen 招聘了一些程式員開發出一個稱為 Mosaic 的瀏覽器 (Browser, 也就是 Web 的 Client 介面) 之後，推動了 Web 的急速發展。

所以，凡是透過瀏覽器接觸到的網路，通常就是 Web 。而不是透過瀏覽器接觸到網路的軟體，則通常是基於 Internet 的應用程式。所以您用 Internet Explorer, Firefox, Chrome, Safari 等瀏覽器所看到的網站，其實都是一個一個的 Web 程式所組成的，而那些不經過瀏覽器，而是要另外安裝的應用程式，像是 Skype、

MSN 等，則是 Internet 應用程式。

本書的焦點是 Web 的程式設計，也就是網站的設計，而非像 Skype、MSN 這樣的 Internet 應用程式。

如果您想學習的是 Internet 應用程式的設計，筆者另外有寫一本採用 C# 的程式設計書籍，請參閱該書，該書的網址如下：

- https://dl.dropbox.com/u/101584453/cs/htm/book.html

Web 的相關技術

HTTP 是 Web 全球資訊網 (萬維網) 的基礎協定，該協定架構在 TCP/IP 架構之上，屬於應用層的協定。構成 HTTP 的主要兩個應用程式是瀏覽器 (Browser) 與網站 (Web Server)。HTTP 是一個典型的 Client-Server (客戶端-伺服端) 架構的協定，使用者透過 Client 端的瀏覽器連結到 Server 的伺服器，然後由伺服端將結果以 HTML 的網頁格式傳回。 HTML 的網頁當中包含了許多超連結 (Hyperlink)，這些超連結連接到某些網址 (URL)，於是使用者可以透過瀏覽器中的超連結，進一步點選其他的網頁，進行網路瀏覽的行為。

絕大部分的使用者，現在都是使用 Internet Explorer, Firefox, Chrome, Safari 等瀏覽器在上網，因此所使用到的就是 Web 程式，本書的焦點也正是這種基於 Web 的程式技術。

Web 相關的技術非常的多，其中最核心的部分是 HTTP/HTML/URL/CSS/JavaScript 等技術，而圍繞著這些核心技術上所發展出來的技術則難以數計，像是 PHP、ASP/ASP.NET、JSP、Ruby on Rail、Python/Dejango、Node.js、XML、JSON、XMLHTTP、AJAX、jQuery 等等，本書後面的章節將採用一個最少語言的架構，以 JavaScript 為核心，採用 HTML + CSS + JavaScript 為主要技術，並搭配 JavaScript 所衍

生出的 jQuery 與 Node.js 等技術，以建構出完整的 Web 程式設計概念。

> 本書 = HTTP + HTML + CSS + JavaScript + jQuery + Node.js 所組成的 Web Programming 技術

其中、HTTP 是所有技術的起點，讓我們先來看看 HTTP 到底做了甚麼事。

HTML 與 HTTP

當您用 Browser 看網站的時候，到底 Browser 傳遞什麼訊息給 Server，而 Server 又回傳神麼訊息給 Browser 呢？

粗略的說、一個最簡單的 Web Server 之功能包含下列三個步驟：

- 步驟一 : 接收瀏覽器所傳來的網址。
- 步驟二 : 取出相對應的檔案。
- 步驟三 : 將檔案內容傳回給瀏覽器。

然而、在這個接收與傳回的過程中，所有的資訊都必須遵照固定的格式，規範這個接收/傳送格式的協定，稱為超文字傳送協定 (Hyper Text Transfer Protocol)，簡稱為 HTTP 協定。

HTTP 協定格式的基礎，乃是建構在網址 URL 上的傳輸方式，早期只能用來傳送簡單的 HTML 檔案，後來經擴充後也可以傳送 其他類型的檔案，包含 影像、動畫、簡報、Word 文件等。

在本文中，我們將先簡介 HTTP 協定的訊息內容，然後在介紹如何以 Java 語言實作 HTTP 協定，以建立一個簡單的 Web Server。

HTTP 協定

當你在 Browser 上打上網址(URL)後， Browser 會傳出一個HTTP訊息給對應的 Web Server，Web Server 再接收到這個訊息後， 根據網址取出對應的檔案，並將該檔案以 HTTP 格式的訊息傳回給瀏覽器，以下是這個過程的一個範例。

豬小弟上網，在瀏覽器中打上 http://ccc.kmit.edu.tw/index.htm，於是、瀏覽器傳送下列訊息給 ccc.kmit.edu.tw 這台電腦。

```
GET /index.htm HTTP/1.0
Accept: image/gif, image/jpeg, application/msword, */*
Accept-Language: zh-tw
User-Agent: Mozilla/4.0
Content-Length:
Host: ccc.kmit.edu.tw
Cache-Control: max-age=259200
Connection: keep-alive
```

當 ccc.kmit.edu.tw 電腦上的 Web Server 程式收到上述訊息後，會取出指定的路徑 /index.htm ，然後根據預

設的網頁根目錄 (假設為 c:)，合成一個 c:.htm 的絕對路徑，接著從磁碟機中取出該檔案，並傳回下列訊息給豬小弟的瀏覽器。

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 438
&lt;html&gt;
  ....
&lt;/html&gt;
```

其中第一行 HTTP/1.0 200 OK 代表該網頁被成功傳回，第二行 Content-Type: text/html 代表傳回文件為 HTML 檔案， Content-Length: 438 代表該 HTML 檔案的大小為 438 位元組。

換言之、 Browser 會傳遞一個包含網頁路徑的表頭資訊給 Server，而 Server 則會回傳一個 HTML 檔案給 Browser，這樣的傳送格式與規則，就稱為 HTTP 協定。

在 Web 開始成長之後，這樣的模式就顯得有點不足了，因為很多網頁需要使用者填入一些欄位，像是帳號密碼等資訊，但是上述的表頭並沒有欄位資訊預留空間，於是為了回傳這些使用者填入的訊息，有人想到了可以在網址的後面補上參數，這種方式就稱為 GET 的參數傳送方式，以下是一個傳送帳號密碼的表頭訊息，其中的 ?user=ccc&password=1234567 是 Browser 傳送給 Server 的參數。

```
GET /login?user=ccc&password=1234567 HTTP/1.0
```

```
Accept: image/gif, image/jpeg, application/msword, */*
Accept-Language: zh-tw
User-Agent: Mozilla/4.0
Content-Length:
Host: ccc.kmit.edu.tw
Cache-Control: max-age=259200
Connection: keep-alive
```

但是後來，這些填入的欄位越來越多，傳送的參數訊息也越來越長，而這些訊息有些也不希望被顯示在網址列上讓大家看到，因此又發展出了另外一種參數傳遞方式，這種方式就稱為 POST，以下是POST訊息的一個範例。

```
POST /msg.php HTTP/1.0
Accept: image/gif, image/jpeg, application/msword, */*
Accept-Language: zh-tw
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0
Content-Length: 66
Host: ccc.kmit.edu.tw
Cache-Control: max-age=259200
Connection: keep-alive
```

```
user=ccc&password=1234&msg=Hello+%21+%0D%0AHow+are+you+%21%0D%0A++
```

上述表頭中的最後一句
「user=ccc&password=1234&msg=Hello+%21+%0D%0AHow+are+you+%21%0D%0A++」是三個欄位參數被編碼後的結果，其中的 user 欄位值是 ccc，password 欄位值是 1234，而 msg 的值則是下列文章。

```
Hello !
How are you !
```

POST 訊息與 GET 訊息不同的地方，除了在 HTTP 的訊息開頭改以 POST 取代 GET 之外，並且多了一個 Content-Length:66 的欄位，該欄位指示了訊息結尾會有 66 個位元組的填表資料，這些資料會被編碼後以文字模式在網路上傳遞。

一但 Server 取得了 Browser 傳來的 GET 或 POST 訊息後，就可以根據其訊息以進行對應的動作，像是檢查帳號密碼是否正確、將某些訊息存入資料庫、然後在透過 HTML 的方式傳送回應畫面給 Browser，這種透過 HTTP 與 HTML 的簡單互動方式，正是由 World Wide Web 所帶來的核心技術，也讓全世界都捲入了這個影響深遠的網路革命當中。

若您對 WebServer 的運作方式有興趣，可以參考筆者的三個程式與文章，這三個程式分別以 Java、C# 與 JavaScript/Node.js 寫成，這會讓您能夠更深入的體會 WebServer 與 HTTP 的運作原理。

- 如何設計簡單的 WebServer? (Java 版) -- http://ccckmit.wikidot.com/code:webserver

- 如何設計簡單的 WebServer? (C# 版) -- http://cs0.wikidot.com/webserver
- 如何設計簡單的 WebServer? (JavaScript/Node.js 版) -- http://ccckmit.wikidot.com/js:nodewebserver

註：curl 這個工具聽說很適合用來觀察各種協定的運作原理，不過筆者還沒用過，未來會了再寫出來給大家看看！ * http://curl.haxx.se/download.html

參考文獻

- How Java Web Servers Work -- http://www.onjava.com/pub/a/onjava/2003/04/23/java_webserver.html
- JDK API : java.net.ServerSocket -- http://java.sun.com/j2se/1.4.2/docs/api/java/net/ServerSocket.html
- Hypertext Transfer Protocol (HTTP/1.0) -- http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html

HTML、CSS 與 JavaScript

HTML、CSS 與 JavaScript 是讓 Web Browser 運作的三大技術，要學會 Web 程式設計的第一步，就是要學會這三項技術，在本章中，我們將從最基礎的 HTML 開始。

如果您想要學習這些技術，筆者強烈的推薦您觀看 w3schools 這個網站，網址如下。

- http://www.w3schools.com/

學習 HTML、CSS、JavaScript 的順序，最好先從 HTML 開始，因為 HTML 最簡單，而且是整個 Web 技術的表現語言，學會 HTML 之後就可以學習 CSS，看看如何對 HTML 進行格式化套表呈現的動作，然後

再進一步學習 JavaScript 這個小型程式語言，以便讓網頁更加動態，增強網頁的的互動功能。

以下是 w3schools 當中這三個主題的網頁，如果您是一個程式設計師，相信您可以用很短的時間就學會這些主題。

- http://www.w3schools.com/html/default.asp
- http://www.w3schools.com/css/default.asp
- http://www.w3schools.com/js/default.asp

一但學會了 HTML、CSS、JavaScript 這三種技術，您就可以開始向 Server 端的技術邁進了。

簡易網頁

HTML 網頁是一種純文字檔案，您只要用記事本這樣的文字編輯器就可以輕易的編出來，以下是一個範例：

```
<html>
<body>
Hello!
</body>
</html>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/hello1.htm

您可以看到在上述範例中，Hello! 訊息被夾在 <body> 與 </body> 中間，而外層又有 <html> 與 </html> 包覆，像這樣的結構稱為標記語言 (Markup Language)。

如果我們在上述文件中再加上一句 Yahoo! 這樣的標記，那麼網頁中將會出現 Yahoo! 這個詞彙，該詞彙通常是藍色文字，而且可以用滑鼠點擊，當您點擊之後網頁就會被導到 Yahoo 台灣的首頁，這種點選後會導入另一個網頁的機制是 Web 的最大特色，稱為 Hyperlink (超連結)，而那些被 <a> 標記起來的文字，就稱為 Hyper Text (超文字)。

```
<html>
<body>
Hello!
<a href="http://tw.yahoo.com/">Yahoo!</a>
</body>
</html>
```

(筆者註：標記 a 當中的 href 是 Hyperlink Reference (超連結引用) 的縮寫，而 a 則是 anchor (錨) 的意思)。

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/hello2.htm

而這也正是 HTML 之全名 Hyper Text Markup Language 的意義，也就是一種含有超連結的標記語言。

由於 HTML 標記語言是有層次性的，因此在排版時我們經常都會使用縮排以凸顯出這種層次感，例如以上的範例經過縮排之後就變成了如下的情況。

```
<html>
  <body>
    Hello!
    <a href="http://tw.yahoo.com/">Yahoo!</a>
  </body>
</html>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/hello3.htm

這樣的結構看起來更加一目了然，能夠凸顯出每個層次之間的包含關係。

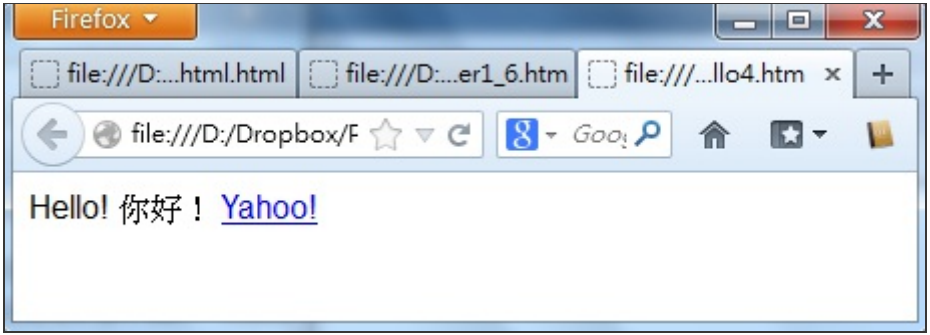
HTML 的表頭

雖然上述範例已經是 HTML 網頁了，而且也通常可以正常檢視了，但是還不夠完整，一個完整的 HTML 網頁通常要包含足夠的表頭資訊，以便讓瀏覽器能正確的辨認這個網頁的類型與編碼方式，以下是一個範例：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8'>
```

```
    </head>
    <body>
        Hello! 你好！
        <a href="http://tw.yahoo.com/">Yahoo!</a>
    </body>
</html>
```

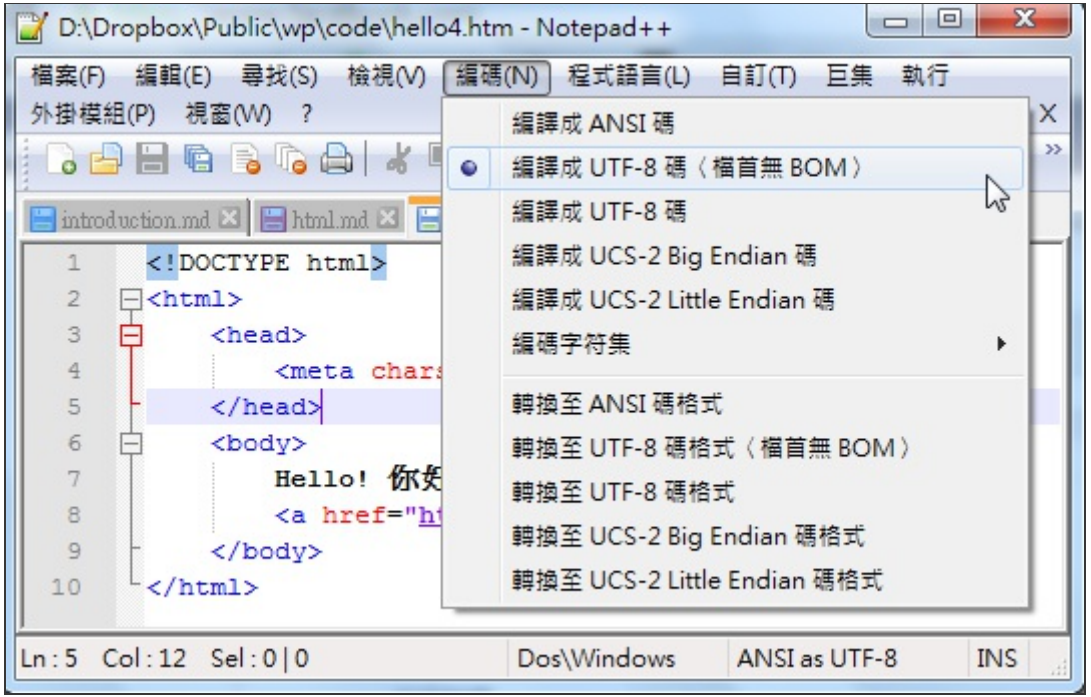
檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/hello4.htm



圖、上述網頁的檢視畫面

其中第一句的 <!DOCTYPE html> 是告訴瀏覽器，這個文件是一個 HTML 檔案，而 <meta charset='utf-8'> 這句話，則是告訴瀏覽器這個文件的編碼是用 UTF-8 的格式。

所以您必須將上述檔案，儲存成 unicode 的 UTF-8 格式，這樣網頁才能正確運作，否則就有可能出現亂碼的情況，以下是筆者使用 Notepad++ 設定 UTF-8 格式的情況。



圖、筆者使用 Notepad++ 設定 UTF-8 格式的畫面

格式化

從上述範例中，您可以看到即使我們在 HTML 的原始文字檔案中進行換行，但是呈現出來的結果卻沒有換行，如果您真的想要強制換行的話，在 HTML 當中必須用
 這個標記，才能讓文字真正換行。

但是強制換行並不是一個好的寫法，比較好的方法是用段落標記 ... 把您的段落包起來，這樣在段落結束後就會換行了。

另外、還有 ... 標記 (也可以用 ...) 可以讓文字變粗體，而 ... 指令則可以讓文字呈現斜體， ... 標記可以強調文字，而 <code>...</code> 可以讓文字以類似程式碼的字型輸出。

若要讓文字變成上標，可以用 ^{...}，或用 _{...} 則可以讓文字變下標，以下範例綜合的呈現了這些格式化標記的結果。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8'>
  </head>
  <body>
    一般字體 (Plain Text) <BR/>
```

```
    <i>斜體 (Italic)</i> <BR/>
    <b>粗體 (Bold)</b> <BR/>
    <p>
      <strong>明顯 (Strong)</strong>
      <em>強調 (Emphasized)</em>
    </p>
    <p>
      這是 <sup>上標</sup> 與 <sub>下標</sub> 的顯示情況！
    </p>
  </body>
</html>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/format.htm

圖、各種格式標記的網頁檢視畫面

標題

HTML 的章節標題共有六種層次，從 h1 到 h6，通常 h1 代表最大的第一層標題，會用最大的字型顯示，而 h6 則是最小的第六層標題，會用最小的字型顯示。

```
<h1>第 1 層段落標題</h1>
<h2>第 2 層段落標題</h2>
```

```
<h3>第 3 層段落標題</h3>
<h4>第 4 層段落標題</h4>
<h5>第 5 層段落標題</h5>
<h6>第 6 層段落標題</h6>
```

而整個網頁的標題則是用 <title>...</title> 進行標記，如下所示。

```
<title>網頁標題</title>
```

以下是這些標題的使用範例，您可以點選看看。

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/header1_6.htm

圖、Header 1-6 的網頁檢視畫面

影像 Image

在 HTML 中，要顯示影像或圖片，只要使用 <img src="圖片相對路徑"> 就行了。如果要指定大小，就可以加上 width 與 height 屬性，如以下範例所示。

```
<img src="F7501.gif"/>
<img src="F7501.gif" width="200" height="300"/>
<img src="F7501.gif" width="20%"/>
```

上述第一個指令 會按照原圖大小顯示，而 則會將圖形縮放到寬度 200，高度 300 的大小，最後一個用 20% 代表縮放到寬度為瀏覽器畫面視窗的百分之二十那麼大。

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/img.htm

圖、img 指令的網頁檢視畫面

評論：竟然趁機為自己的書打廣告，真是太惡劣了！....

表格 Table

在 HTML 當中、有很多排版都是使用表格的方式製作的，例如「側欄、上方橫幅」等等，表格的語法如以下範例所示。

```
<table>
<tr>
<th></th>
<th>欄 1</th>
<th>欄 2</th>
</tr>
<tr>
<th>列 1</th>
<td>1,1</td>
<td>1,2</td>
</tr>
<tr>
<th>列 2</th>
<td>2, 1</td>
<td>2, 2</td>
```

```
</tr>
</table>
```

表格的語法都是由 <table>...</table> 所夾住的，其中 <tr>...</tr> 會夾住一個列，而每個列又可以分成數個欄，這些欄可能用 <th>..</th> 或 <td>...</td> 夾住，其中 th 通常是用在標頭、而 td 則是用在內容。

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/table.htm

表格範例的顯示結果

項目 list

HTML 的項目是用 <li>... 語法所框住的，如果在項目的外面加上 <ul>... ，那就會顯示無編號的項目清單，若加上 <ol>...，那就會顯示有編號的項目清單。

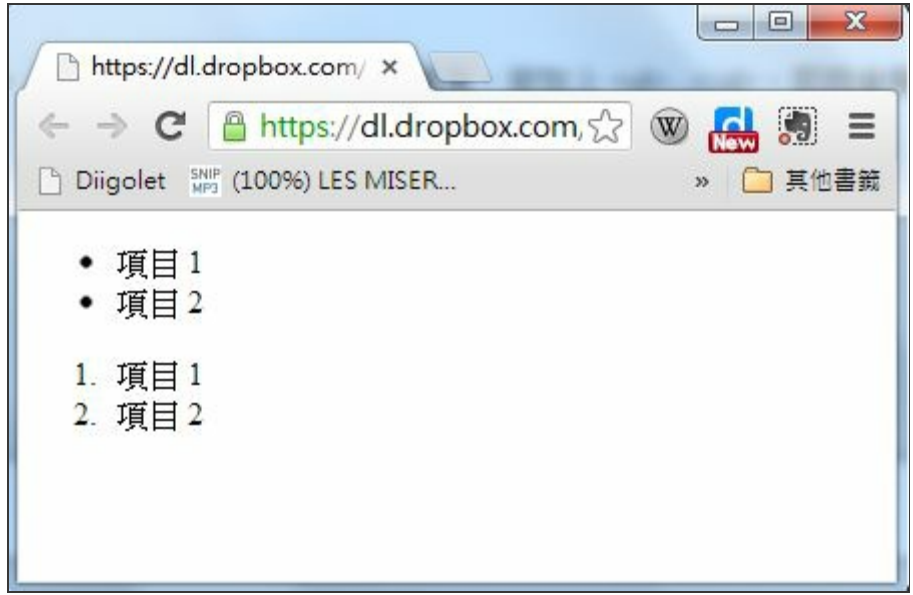
無編號項目清單：

```
<ul>
<li>項目 1</li>
<li>項目 2</li>
</ul>
```

有編號項目清單：

```
<ol>
<li>項目 1</li>
<li>項目 2</li>
</ol>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/list.htm



項目清單的顯示結果

表單 Form

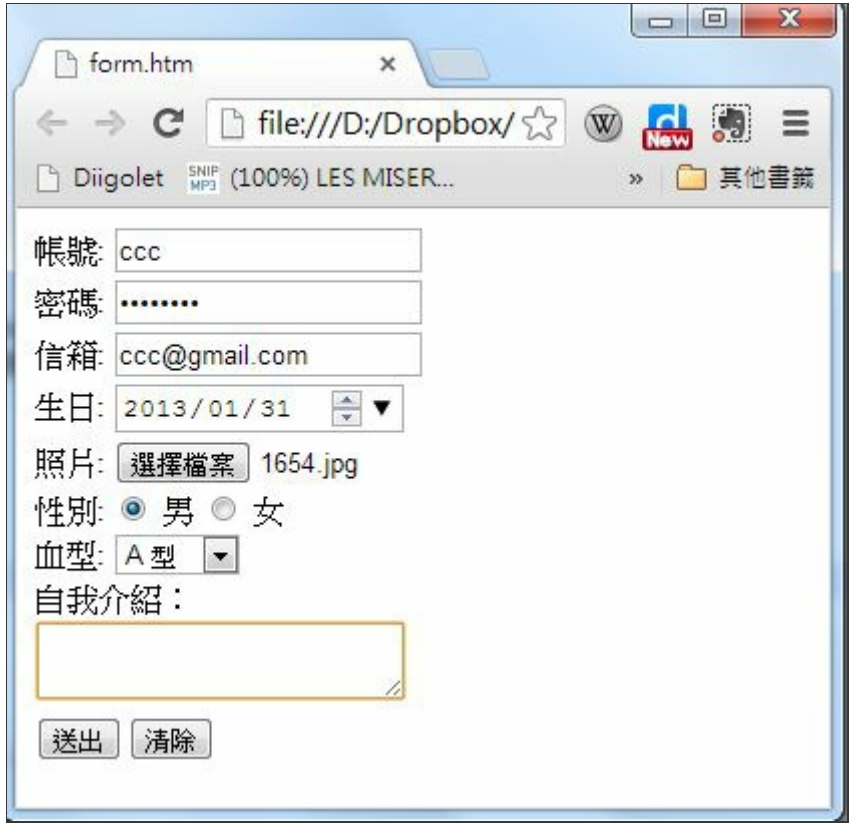
在 HTML 當中，表單 (Form) 是指可以讓使用者進行輸入的元件，其語法是用 <form> ...</form> 夾住一堆的輸入元件，這些輸入元件包含 input (輸入)、 textarea (文字區)、 select (選項) 等，其中的 input 還可以根據其 type 欄位顯示成 checkbox, color, date, datetime, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week 等各種不同的輸入欄形式，以下是一個表單

的範例。

```
<form action="signup" method="post">
帳號: <input type="text" name="user"/><br/>
密碼: <input type="password" name="password"/><br/>
信箱: <input type="email" name="email"/><br/>
生日: <input type="date" name="birthday"/><br/>
照片: <input type="file" name="picture"/><br/>
性別: <input type="radio" name="sex" value="male" checked/> 男
    <input type="radio" name="sex" value="female"/> 女<br/>
血型: <select name="BloodType">
      <option value="A">A 型</option>
      <option value="B">B 型</option>
      <option value="AB">AB 型</option>
      <option value="O">O 型</option>
    </select> <br/>
自我介紹： <br/>
<textarea name="AboutMe">
</textarea> <br/>
```

```
<input type="submit" value="送出"/><input type="reset" value="清除"/><br/>
</form>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/form.htm



表單的顯示結果

在上述的範例中，當 submit 類型的送出鈕被按下後，瀏覽器會將這些填寫的資訊，以第一章所說的

GET/POST 方式，發送給伺服器，如果 method 欄位是 GET，那麼會採用在 HTTP 表頭網址處傳送 singup?user=xxx&password=xxx .....這樣的形式送出，這種方式會將密碼顯示在瀏覽器的網址列上，比較容易被看到，若 method 欄位是 POST，則會在 HTTP 表頭尾端加上 user=xxx&password=xxx ... 的資訊，不會在網址列上被看到。當然、如果有人監控網路上訊息的話，還是會看得到這些輸入資訊。

若要更安全，則必須採用 HTTPS 的 SSL 方式傳遞，這種方式會對訊息加密編碼，就比較不會有輸入訊息外洩的危險。

參考文獻

- W3School : HTML Turorial -- http://www.w3schools.com/html/default.asp

CSS 版面設計

CSS 簡介

在前一章當中，我們介紹了 HTML 的語法，我們可以透過 HTML 的表格排出某些格式，但是卻很難讓整個頁面足夠美觀。

舉例而言，假如我們想讓表格的標頭欄以黑底白字的方式顯示，那麼就得靠 CSS 來幫忙了。

CSS 是 Cascading Style Sheets 的簡稱，中文可翻譯為「串接樣式表」，CSS 主要是用來設定 HTML 的顯示版面，包含「字體、大小、顏色、邊框、背景」等等，一個好的 CSS 可以讓您的網頁變得很美觀，而且不需要大幅修改網頁的內容，只要加入一個 CSS 引用即可。

當套用不同的 CSS 時，網頁的風格就可以產生完全不同的感覺，這讓你的網頁可以輕易的更換成不同的版型。事實上、當您在 blogspot 或 wordpress 網誌平台套用不同版型的時候，只不過是更換了 CSS 樣版而已。

在 HTML 當中，CSS 有兩種寫法，一種是內嵌式的寫法，另一種是外連式的寫法。

為了說明 CSS 的語法，先讓我們回顧一下，一個簡單沒有格式化的 HTML 表格，其寫法如下：

```
<html>
```

```
<head><meta charset='utf-8'></head>
<body>
<table>
<tr><th></th><th>欄 1</th><th>欄 2</th></tr>
<tr><th>列 1</th><td>1,1</td><td>1,2</td></tr>
<tr><th>列 2</th><td>2,1</td><td>2,2</td></tr>
</table>
</body>
</html>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/table.htm

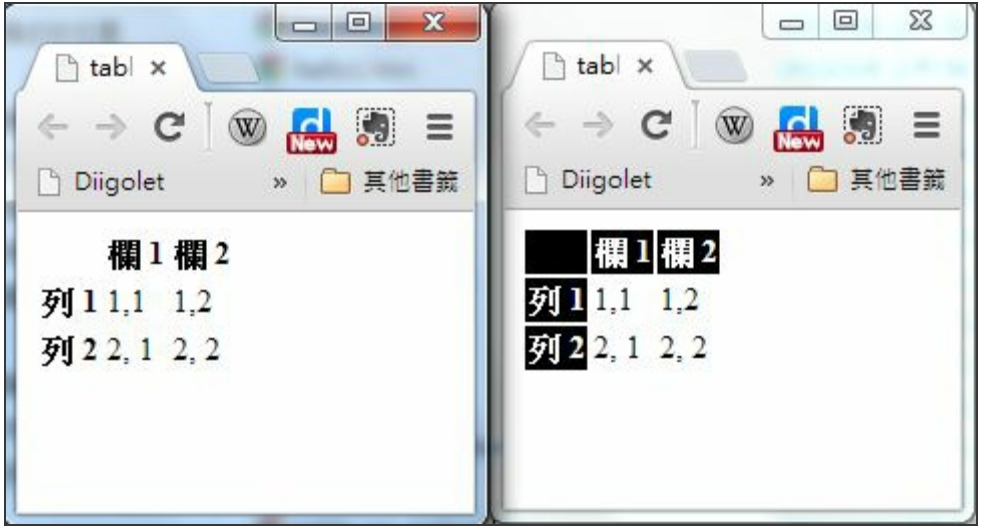
如果我們將上述表格的 th 部分，都加上 style="background-color:black; color:white;" 這樣的 CSS 語法，那麼我們就會得到一個黑底白字的表格。

```
<html>
<head><meta charset='utf-8'></head>
<body>
<table>
<tr>
<th style="background-color:black; color:white;"></th>
```

```
<th style="background-color:black; color:white;">欄 1</th>
<th style="background-color:black; color:white;">欄 2</th>
</tr>
<tr>
<th style="background-color:black; color:white;">列 1</th>
<td>1,1</td>
<td>1,2</td>
</tr>
<tr>
<th style="background-color:black; color:white;">列 2</th>
<td>2, 1</td>
<td>2, 2</td>
</tr>
</table>
</body>
</html>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/table_css_embed.htm

我們使用 CSS 排版前與排版後的結果可以對照如下：



表單的顯示結果

您可以看到在這種寫法當中，我們幾乎都一直在 th 重複撰寫 style="background-color:black; color:white;" 這一行文字，總共重複了 5 次，這顯然是很浪費時間的行為。

如果我們可以改用統一的方式，寫出「對於所有 th 而言，我都要用 background-color:black; color:white; 這樣的方式顯示」那不就簡單多了嗎？

是的、CSS 確實可以讓您這樣做，讓我們將上述範例修改成統一設定格式的版本，如下所示。

```
<html>
<head>
<meta charset='utf-8'>
<style>
th { background-color:black; color:white; }
</style>
</head>
<body>
<table>
<tr><th></th><th>欄 1</th><th>欄 2</th></tr>
<tr><th>列 1</th><td>1,1</td><td>1,2</td></tr>
<tr><th>列 2</th><td>2,1</td><td>2,2</td></tr>
</table>
</body>
</html>
```

上述統一將 CSS 寫在 <head><style>...</style></head> 裏的這種寫法，比起內嵌式的顯然簡短多了，這種寫法正是 CSS 的精華之所在。

但是、如果我們想要對整個網站的所有檔案，都套用同一種格式的話，那麼在每個 HTML 的頭部都要嵌入一整套 CSS 語法，那就不太方便了，萬一我們想要改變版面格式，不就每個 HTML 檔案都要更改，這

顯然還不夠好用。

要解決這個問題，可以將表頭的 CSS 獨立到一個檔案中，然後再用連結的方式引用，這樣只要修改那個 CSS 檔案，整個網站的風格就跟著改變，所以上述檔案就可以改寫如下。

檔案：table_css.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
<body>
<table>
<tr><th></th><th>欄 1</th><th>欄 2</th></tr>
<tr><th>列 1</th><td>1,1</td><td>1,2</td></tr>
<tr><th>列 2</th><td>2,1</td><td>2,2</td></tr>
</table>
</body>
</html>
```

檔案：mystyle.css

```
th { background-color:black; color:white; }
```

透過這種方式，我們就可以為網頁設計統一的風格，也比較容易更換網頁的排版風格了，這就是 CSS 語法的用途了。

當然、上述的網頁還不夠好看，如果我們將 mystyle.css 修改如下，您就會發現表格變得好看多了。

```
table { border-collapse: collapse; border: 1px solid #373737; }
th { background-color:black; color:white; padding:10px; margin:10px; }
td { padding:10px; margin:10px; }
```

較優美的表單顯示結果

路徑, tag, id 與 class

即使有了可以統一套用的方式，有時我們還是會感覺到不方便，舉例而言，如果我們想要對某個表格套上 A 格式，然後在對另一個表格套上 B 格式，這時採用上述方法就不太夠用了。

為了解決這種問題，HTML 發展出了 id 與 class 這兩個屬性，我們可以根據 id 或 class 的名稱分別套用不同的 CSS 樣式。

舉例而言，以下網頁中有兩個 div 框，我們可以分別為這兩個框套用不同的格式，其中 classA 前面加了點符號，成為「.classA」，代表要比對的是 class 屬性，如果想要比對的是 id 屬性，那麼就可以加上井字符號 # (像是範例中 #topbar 的情況)。

```
<html>
<head>
<meta charset='utf-8'>
<style>
#topbar { background-color:gray; color:blue; padding:10px; margin:10px; }
.classA { background-color:black; color:white; padding:10px; margin:10px; }
.classB { background-color:blue; color:yellow; padding:10px; margin:10px; }
</style>
</head>
<body>

<div id="topbar">頂區塊</div>

<div class="classA">A 區塊</div>

<div class="classB">B 區塊</div>
```

```
</body>
</html>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/div_css.htm

兩種 CSS class 的顯示結果

到目前，我們可以看到有三種 css 的指定方式，也就是「標記 tag、代號 id 與類別 class」，可以用 CSS 來定位。事實上、這些指定方式還可以互相串接，以下是一些範例：

```
.classA table { ... }            // 套用到 class="classA" 內部的 table 標記
#topbar { ....}                  // 套用到 id="topbar" 的元素上
#topbar table .classA a { .... }   // 套用到 table 內具有 class="classA" 類別裏的超連結 a 標記上。
```

更棒的是、這些選取方式之間還可以共用，只要利用逗點符號「,」 分隔就可以了，以下是一些範例：

```
.classA, #topbar { ... }          // 套用到 class="classA" 或 id="topbar" 的元素上
#topbar a, .classA a { ....}       // 套用到 id="topbar" 或 class=".classA" 裏的 a 元素上
```

一個實用的 CSS 的範例

如果您想進一步學習 CSS 的各個屬性與用法，請參考下列網址，我們將不一一說明這些屬性的用法。

- http://www.w3schools.com/css/default.asp

在此、筆者將自己常用的一個 CSS 檔案列出，讀者觀察後可以自行複製修改使用。

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
```

```
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
  margin: 0;
  padding: 0;
  border: 0;
  font: inherit;
  vertical-align: baseline;
  line-height:160%;
}

h1, h2, h3, h4, h5, h6 {
  color:#333333;
  margin: 0;
  font-family: '標楷體', 'Times New Roman';
```

```
  font-weight: bold;
}

p {
  margin: 10px 0 15px 0;
  font-size:100%;
}

li {
  font-size:100%;
}

footer p {
  color: #f2f2f2;
}

a {
  text-decoration: none;
  color: #007edf;
  text-shadow: none;
```

```
  transition: color 0.5s ease;
  transition: text-shadow 0.5s ease;
  -webkit-transition: color 0.5s ease;
  -webkit-transition: text-shadow 0.5s ease;
  -moz-transition: color 0.5s ease;
  -moz-transition: text-shadow 0.5s ease;
  -o-transition: color 0.5s ease;
  -o-transition: text-shadow 0.5s ease;
  -ms-transition: color 0.5s ease;
  -ms-transition: text-shadow 0.5s ease;
}

table {
  border-collapse: collapse;
  border-spacing: 0;
  border: 1px solid #373737;
  margin-bottom: 20px;
  text-align: left;
  margin-left:auto;
  margin-right:auto;
```

```
}

th {
    padding: 10px;
    background-color:black;
    color:white;
}

td {
  padding: 10px;
  border: 1px solid #373737;
}

em { font-weight:bold; }

#topbar {
  margin: 0;
  padding: 1px;
  border: 0;
  font: inherit;
  vertical-align: baseline;
```

```
  background-color:black;
  color:white;
  color:white;
  width:95%;
  text-align:right;
  font-weight:bold;
}

#content {
  margin:10px;
  padding:10px;
}

pre {
  border: 1px solid #373737;
  background-color:#dddddd;
  color:#333333;
  font-size:medium;
  width:95%;
  padding:10px;
```

```
}

img {
  border: 1px solid #373737;
  margin-left: auto;
  margin-right: auto;
  display: block;
}

.figure .caption {
  text-align:center;
}

#footer {
  text-align:center;
  font-size:small;
  color:#666666;
  margin: 10px;
  padding: 10px;
}
```

參考文獻

- http://www.w3schools.com/css/default.asp
- http://css.maxdesign.com.au/listutorial/

JavaScript 語言基礎

JavaScript 是目前在瀏覽器上唯一通用的程式語言，這種語言經常遭受到許多誤解，像是「JavaScript 就是 Java 語言的簡化版」、「JavaScript 語言很難用」、「JavaScript 語言設計很差勁」等等。

然而，這些誤解其實是我們不瞭解 JavaScript 所造成的。如果您用心去瞭解 JavaScript，您會發現這是一個「簡單、輕巧又優美」的語言，其原型導向的設計方式，用很簡單的概念達成了物件導向語言的功能，真的很適合做為瀏覽器上的共通語言。

JavaScript 雖然是一種物件導向語言，但是更精確的說，JavaScript 事實上是一種「原型導向」的語言，其中每個物件的 _prototype 欄位都可以指向他的原型，然後用來 clone (自體繁殖) 出新的物件，您可以用 function 型態宣告一個物件，如此該物件就自動具有建構函數了，這種做法是非常簡單、奇特、但卻又彈性十足的方法。

雖然 JavaScript 語言有許多優點，但是也有不少缺點，例如有些語法的邏輯很奇怪，而且 JavaScript 的物件導向並非像 Java, C#, Ruby 等語言那樣傳統，而是採用原型導向的方式，這讓傳統物件導向的使用者在學習 JavaScript 時會感到困難。不過一但您理解了原型導向的精神之後，就會發現這種方法真的有不少優點了。為了較深入的理解 JavaScript ，讓我們先來看看 JavaScript 的歷史！

Web 技術在 1990 年開始成形，早期網際網路只有文字，沒有圖像、音效，且操作非常繁瑣。當時還是大學生的馬克·安德生（Marc Andreessen）被伊利諾伊大學的電腦應用中心聘為臨時工作人員，馬克·安德生提出設計一種簡單的瀏覽程式想法，能方便的檢索網路資料，於是招聘了幾個程式設計師花了六週的時間

開發。於是誕生了 Mosaic 瀏覽器。 Mosaic的出現，算是點燃了後期網際網路熱潮的火種。後來在1994年4月，馬克.安德生和Silicon Graphics公司的創始人吉姆·克拉克（Jim Clark）在美國加州設立了一家公司，進一步改善瀏覽器技術，並且創造出了 Netscape 瀏覽器。

當時瀏覽器只能顯示文字與圖片，無法做出像功能表這樣需要即時互動的網頁，於是 Netscape 公司的 Brendan Eich設計出了 LiveScript，這是一種動態、弱型別、基於原型的語言，用來寫一些讓網頁可以動起來的小程式。

由於Netscape在與昇陽（Sun）合作的關係，所以 Netscape 將 LiveScript 改名為JavaScript，後來甲骨文 (Oracle) 公司買下了昇陽，所以 JavaScript是甲骨文公司的註冊商標。為了避免侵犯商標權，於是 Ecma國際（前身為歐洲電腦製造商協會）將其以JavaScript為基礎所制定的語言改稱為 ECMAScript。所以現在我們所說的 JavaScript，其真正意義其實應該是 ECMAScript。

現在、JavaScript 已經成為一種非常重要的程式語言，JavaScript 除了用在瀏覽器之外，也被用在許多其他領域，像是您可以用 node.js 在 Server端撰寫 JavaScript 的程式，也可以在 Unity3D 當中用 JavaScript 撰寫遊戲程式，更可以在 Tatinum 當中用 JavaScript 撰寫手機的 APP 然後放到 iPhone, iPad 或 Android 上面去執行。

JavaScript 基本語法

JavaScript 最常被嵌入到 HTML 網頁中，以便讓網頁顯式出一些特別的動態效果，因此含有 JavaScript 的網頁通常也被稱為動態網頁，以下是一個 HTML 內嵌 JavaScript 的簡單範例與顯示結果。

原始碼

```
<html>
<body>
<script type="text/javascript">
var x =5, y=7;
var s = "Hello! "
t = s + x;
z = x * y;
document.write("<pre>x="+x+"\ny="+y+"\ns="+s+"\nt="+t+"\nz="+z+"</pre>");
</script>
</body>
</html>
```

執行結果

JavaScript 的 if 語法也很簡單，基本上與 C 、Java、C# 等語言都相同，以下是一個簡單的範例。

原始碼

```
<html>
<body>
```

```
<script type="text/javascript">
var score = 61;
if (score >= 60)
  document.write("及格");
else
  document.write("不及格");
</script>
</body>
</html>
```

執行結果

JavaScript 的迴圈語法也是繼承 C 語言的，以下是一個範例：

原始碼

```
<html>
<body>
<script type="text/javascript">
for (i=1;i<=10;i++) {
  if (i == 3) continue;
```

```
  if (i == 8) break;
  document.write("i="+i+"<BR/>");
}
</script>
</body>
</html>
```

執行結果

在 JavaScript 當中，函數的定義有兩種寫法，一種是正常的函數宣告，另一種是把函數當變數的指定方法。事實上，JavaScript 的函數也是一種基本型態，具有這種特點的語言通常稱為函數式語言。這種特性非常好用，因為在函數式語言當中，函數在也不是次等公民，而是一種基本型態，以下是一個 JavaScript 的函數宣告範例。

原始碼

```
<html>
<body>
<script type="text/javascript">
// 第一種寫法，將匿名函數指定給變數。
var add = function(a,b) {
  return a+b;
```

```
}
//第二種寫法，直接宣告函數，該函式是一個函數物件
function sub(a,b) {
  return a-b;
}
document.write("add(3,5)="+add(3,5) +" sub(7,2)="+sub(7,2));
</script>
</body>
</html>
```

執行結果

在 JavaScript 當中，您若要宣告一個陣列時，可以採用 var a = new Array() 的方法，建立一個陣列物件，然後用 a[i] 這樣的方法，存取陣列中的元素。

原始碼

```
<html>
<body>
<script type="text/javascript">
var x;
var friends = new Array();
```

```
friends[0] = "John";
friends[1] = "Mary";
friends[2] = "George";
for (p in friends) {
  document.write(p + ":"+ friends[p] + "<br/>");
}
</script>
</body>
</html>
```

執行結果

JavaScript 的物件導向語法

雖然說，JavaScript 也能實作很好的物件導向功能，但是如果我們說 JavaScript 是一種物件導向語言，那麼會很容易造成混淆。因為 JavaScript 的物件導向非常非常的特別，與 Java, C++, C# 等語言的物件導向實作方法有很大的不同。

嚴格的說，JavaScript 是一種原型導向語言，原型導向是一種特別簡單的物件導向實作機制，以下是幾個 JavaScript 的物件範例：

```
obj = new Object()
```

```
obj.x = 3;      // 為 obj 新增一個欄位 x，其值設定為 3
obj.y = 5;      // 為 obj 新增一個欄位 y，其值設定為 5
obj.z = obj.x + obj.y;   // 為 obj 新增一個欄位 z，其值設定為 x+y
```

對於曾經使用像 Java 這種傳統物件導向語言的人而言，會感覺到上述的程式很奇特，因為 obj 一開始只是一個空物件，並沒有包含任何的欄位，但是我們透過指定的方式，動態的為物件增添了 x, y, z 等欄位。

物件導向基本上有三大特性，1.封裝 2. 繼承 3. 多型，瞭解這三個特性的實作方式，通常就可以學會一種語言的物件導向語法了，以下我們將分別針對 JavaScript 中的這三大特性進行介紹。

JavaScript 物件的封裝

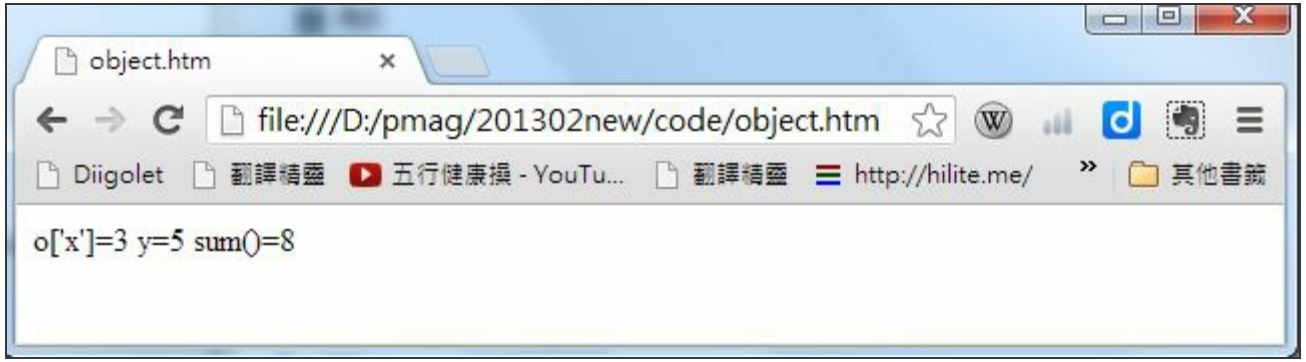
物件導向中的封裝特性，是指將「資料」與「函數」封入一種稱為「物件」的結構當中，以下是 JavaScript 的一個物件範例，其中 x, y 是資料，而 sum 則是函數，這些成員都被封裝在 obj 這樣一個建構函數當中，因此我們呼叫 var o = new obj() 這個指令時，就會建立一個新的物件，並傳回給 o 變數。

程式範例: object.htm

```
<html>
<body>

<script type="text/javascript">
function obj() {
```

```
  this.x = 3;       // 第一種寫法，為 obj 新增一個欄位 x，其值設定為 3
  this["y"] = 5;    // 第二種寫法，為 obj 新增一個欄位 y，其值設定為 5
  this.sum = function() { return this.x + this.y; } // 為 obj 新增一個欄位 add，其值為一個匿名函數
}

var o = new obj();

document.write("o['x']="+o['x']+" y="+o.y+" sum()="+o.sum());
</script>

</body>
</html>
```

執行結果



JavaScript 物件的繼承

物件導向中的繼承特性，是指「子物件」可以繼承「父物件」的資料與函數，並且進行修改，這樣我們就不需要重複的實作父物件已經有的函數，或者定義父物件已經有的資料，以達成程式碼重用的目的。

以下範例中的 Student 物件就繼承了 Person 物件，其方法是在 Student 物件的建構函數中，指定 this.prototype = Person 以達到繼承的目的，然後再呼叫 this.prototype(name, age) 以呼叫父物件的建構函數，建立出 name, age, toStr() 物件內容。

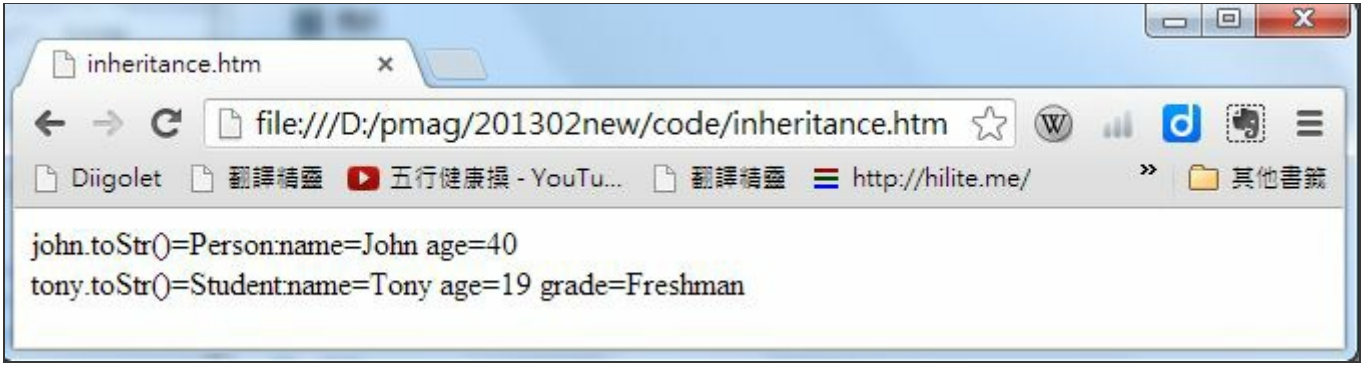
程式範例: inheritance.htm

```
<html>
<body>
```

```
<script type="text/javascript">
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.toStr = function() {
    return "Person:name="+this.name+" age="+this.age;
  }
}
var john = new Person("John", 40);

document.write("john.toStr()="+john.toStr()+"<BR/>");

function Student(name, age, grade) {
  this.prototype = Person;
  this.prototype(name, age);
  this.grade = grade;
  this.toStr = function() {
    return "Student:name="+this.name+" age="+this.age+" grade="+this.grade;
  }
}
```

```
var tony = new Student("Tony", 19, "Freshman");

document.write("tony.toStr()="+tony.toStr()+"<BR/>");
</script>

</body>
</html>
```

執行結果



JavaScript 物件的多型

物件導向中的多型特性，是指當不同的「子物件」繼承同一個「父物件」時，我們可以透過宣告父物件容器，卻將內容指向子物件的方式，以便呼叫到不同子物件中的對應函數。

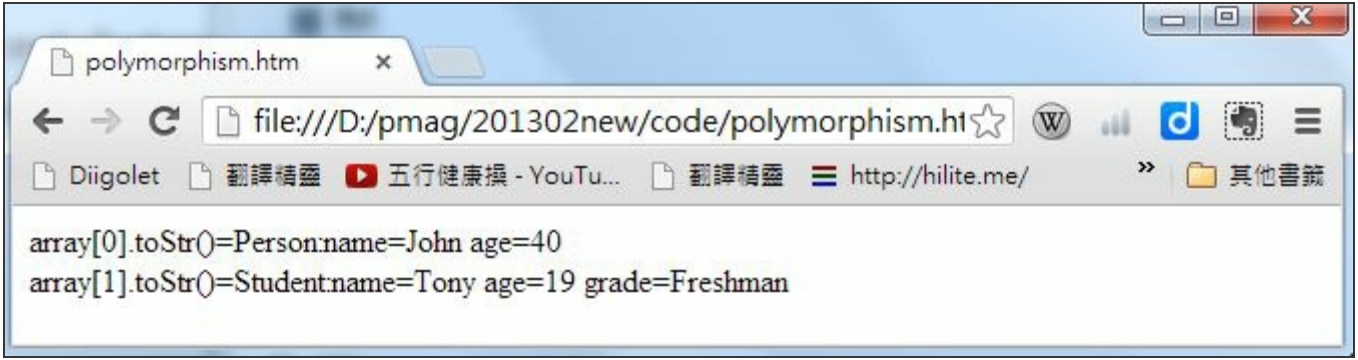
以下範例中的 var array = [john, tony] 這一行，其中的 john 是 Person 類型的物件，tony 是 Student 類型的物件，但是由於 Student 繼承了 Person，而且兩者都有 toStr() 函數，因此當我們用 for (i in array) ... array[i].toStr() ... 這樣的方式呼叫 array[i].toStr() 時，對 john 物件會呼叫到 Person 的 toStr(), 而對 tony 物件會呼叫到 Student 的 toStr(), 這樣就達到了多型的結果。

程式範例: polymorphism.htm

```
<html>
<body>

<script type="text/javascript">
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.toStr = function() {
    return "Person:name="+this.name+" age="+this.age;
  }
}
var john = new Person("John", 40);
```

```
function Student(name, age, grade) {
  this.prototype = Person;
  this.prototype(name, age);
  this.grade = grade;
  this.toStr = function() {
    return "Student:name="+this.name+" age="+this.age+" grade="+this.grade;
  }
}

var tony = new Student("Tony", 19, "Freshman");

var array = [ john, tony ];

for (i in array)
  document.write("array["+i+"].toStr()="+array[i].toStr()+"<BR/>");
</script>

</body>
</html>
```

執行結果



JavaScript 物件的精簡表示法 -- JSON 物件資料交換格式

如果我們想直接在程式中宣告一個複雜的物件，可以使用 JavaScript 中的 {...} 與 [...] 的語法組合，用簡單的語法建構出整個物件。這種格式也常被用在網頁程式的資料交換當中，因此有一個很特別的名稱叫 JSON （Javascript Object Notation）。

目前網路上最常使用的資料交換格式是 XML，但是 XML 文件很繁瑣且囉嗦，讓使用者撰寫不方便，而且不容易嵌入網頁中進行處理。為了讓網頁上的共通程式語言 JavaScript 可以輕易的交換資料，網頁程式的設計者也常用 JSON 取代 XML 進行資料交換。

以下是一個採用 JSON 格式的朋友資料範例，該範例中有兩個朋友，一個是 John, 22 歲，另一個是 Mary, 28 歲。

```
{
  "friends": [
    {"name": "John", "age": 22 }
    {"name": "Mary", "age": 28 }
  ]
}
```

程式範例：json.htm

```
<html>
<body>

<script type="text/javascript">

var george = {
  "name": "George",
  "age": 25,
  "friends": [
    {"name": "John", "age": 22 },
    {"name": "Mary", "age": 28 }
  ]
```

```
};

document.write("george.age="+george.age+"<br/>");
document.write("george.friends:\n<ol>");
var friends = george.friends;
for (i in friends)
  document.write("<li>"+friends[i].name+" is "+friends[i].age+"years old!</li>");
document.write("</ol>");
</script>

</body>
</html>
```

執行結果

以上我們簡單的介紹了 JavaScript 的物件導向功能，這種物件導向的實作方式由於是以原型為核心的，因此筆者喜歡稱 JavaScript 為原型導向 (prototype oriented) 的語言，雖然並不是很多人使用「原型導向」這個用法。

JavaScript 的特殊技巧 - Closure 與匿名函數

在以下的程式中，我們同時展示了 JavaScript 當中的全域變數、區域變數與匿名函數、Closure 等機制。在本文中，我們會用這個程式逐步解釋一些 JavaScript 語言中重要但詭異的概念。

很多語言都有全域變數與區域變數之分，在 JavaScript 當中看來也是如此，但事實上 JavaScript 的全域變

數其實只是最外層領域的區域變數而已，在瀏覽器當中，這個最外層領域就是 window，所以以下程式中最後一部分的 sum 與 windows.sum 都同樣是 8。

程式：closure.htm

```
<html>
<body>

<script type="text/javascript">
var sum = 0;
function add(x) {
  sum += x;
  document.write("typeof:x="+typeof x+" y="+typeof y+" z="+typeof z+" sum="+typeof sum+"<br/>");
  var y = 2;
  return function() {
    var z = x+y;
    document.write("value :x="+x+" y="+y+" z="+z+" sum="+sum+"<br/>");
    document.write("typeof:x="+typeof x+" y="+typeof y+" z="+typeof z+" sum="+typeof sum+"<br/><br/>");
  }
}
```

```
f = add(3);
f();

add(5)();

document.write("typeof:x="+typeof x+" y="+typeof y+" z="+typeof z+" sum="+typeof sum+"<br/><br/>

</script>

<script type="text/javascript">
document.write("sum="+sum+" window.sum="+window.sum);
</script>

</body>
</html>
```

因此、JavaScript 可以說沒有全域變數的概念，全域變數的只不過是最外層物件的區域變數罷了。

在 JavaScript 每一層領域當中，都可以定義區域變數，例如在上述範例中，sum 是最外層變數，因此所有的程式區段都可以存取這個變數。

參數也是一種區域變數，像是 add(x) 當中的 x，也有領域慨念，其作用範圍僅適用於 add 裏面。而 add 當

中所定義的區域變數 y，則只有在定義之後才會生效，因此其領域範圍是從 var y=2; 這行程式開始，一直到 add 函數結束為止。

JavaScript 當中的函數，可以沒有名稱，這種函數稱為匿名函數，像是上述範例的函數 add 中，就傳回了一個匿名函數，如下所示：

```
function add(x) {
  sum += x;
  document.write("typeof:x="+typeof x+" y="+typeof y+" z="+typeof z+" sum="+typeof sum+"<br/>");
  var y = 2;
  return function() {
    var z = sum+x+y;
    document.write("value :x="+x+" y="+y+" z="+z+" sum="+sum+"<br/>");
    document.write("typeof:x="+typeof x+" y="+typeof y+" z="+typeof z+" sum="+typeof sum+"<br/><br/>");
  }
}
```

變數 z 是該匿名函數的區域變數，由於設定為 sum+x+y，因此存取了外層的 y、x 與更外層的 sum，領域的存取規則是內層可以存取外層的變數，但是外層卻不能存取內層的變數。

筆者認為，JavaScript 中的一個最特別且強大的特性，莫過於函數既可以塞給一般變數，也可以當參數傳遞。例如以下程式區段就是用 f 去接收 add(x) 函數所傳回來的匿名函數，然後再用 f() 函數去執行這個匿

名函數而已。

```
f = add(3);
f();

add(5)();
```

上列程式碼的最後一行 add(5)()，只不過是將這兩個動作合起來一起作，也就是呼叫 add(5) 後，傳回值是個匿名函數，然後再用「匿名函數()」這樣的方法呼叫函數，去執行該匿名函數而已。(這個功能有點像 C 當中的函數指標)

在上述的 closure.htm 程式中，還有個很特別的地方，就是這些匿名函數被傳回來後，就已經回到了最外層的領域範圍了 (window)，那麼當該匿名函數執行時，照理說由於 x, y 屬於內層 add 函數的區域變數，應該會存取不到才對，但為何程式執行結果卻都還是正常，沒有發生錯誤呢？

這牽涉到 JavaScript 當中一個非常特別的機制，稱為 Closure (閉包)。

閉包與 JavaScript 的領域 (Scope) 特性有關，因為 JavaScript 採用 Lexical Scope，也就是變數作用範圍依照程式定義時的上下文 (Context) 所決定，而不是根據執行時期的上下文所決定的。

因此，上述匿名函數所服從的領域規則，仍然是定義時的領域，也就是 add 函數的子領域，而不是執行時期 window 的子領域，因此仍然可以正確存取 x,y,z 等變數，不會發生錯誤。

參考文獻

- Node.js 中文電子書 » JavaScript 與 NodeJS
  - http://book.nodejs.tw/zh-tw/node_javascript.html#scope-closure
- [图解] 你不知道的 JavaScript - “this”
  - http://www.cnblogs.com/ruxpinsp1/archive/2008/04/20/1162463.html
- Javascript - 淺談this與Closure
  - http://blog.darkthread.net/post-2009-03-11-js-this-and-closure.aspx
- The this keyword
  - http://www.quirksmode.org/js/this.html
- JavaScript中的closure好難懂，它和匿名函式有什麼不同?
  - http://ithelp.ithome.com.tw/question/10000477

JavaScript 的字串處理

字串物件 String

String 物件的成員

屬性/函數	說明	範例	結果
constructor	傳回建構函數	"Hello".constructor	function String() { [native code] }
length	傳回長度	"Hello".length	function String() { [native code] }
prototype	傳回原型	"Hello".prototype	undefined
charAt()	傳回第 i 個字元	"Hello".charAt(1)	e
charCodeAt()	傳回第 i 個字元的 Unicode	"Hello".charCodeAt(1)	101

concat()	連接兩個以上的字串	"Hello".concat(" World", " !")	Hello World !
fromCharCode()	將 Unicode 代碼轉為字元	"Hello".fromCharCode(101, 102)	ef
indexOf()	傳回子字串的位置	"Hello".indexOf("el")	1
lastIndexOf()	傳回子字串的位置 (倒著掃瞄)	"Hello".lastIndexof("l")	3
match()	搜尋正規表達式	"Hello".match("[aeiou]")	2
replace()	取代正規表達式	"Hello".replace("l", "L")	HeLLo
search()	搜尋正規表達式	"Hello".search("[aeiou]")	e
slice()	切出字串	"Hello".slice(-3)	llo
split()	分割字串	"Hello".split("e")	H,llo
substr()	取出 from 長 len 的子字串	"Hello".substr(2,2)	ll

substring()	取出 from 到 to 的子字串	"Hello".substring(2,4)	llo
toLowerCase()	轉為小寫	"Hello".toLowerCase()	hello
toUpperCase()	轉為大寫	"Hello".toUpperCase()	HELLO
valueOf()	傳回原型值	"Hello".valueOf()	Hello

程式範例

```
<html>
<body>
<script type="text/javascript">
var s = "Hello";
document.write("s = 'Hello'");
document.write("s.constructor = "+s.constructor+"<BR/>");
document.write("s.length = "+s.length+"<BR/>");
document.write("s.prototype = "+s.prototype+"<BR/>");
document.write("s.charAt(1) = "+s.charAt(1)+"<BR/>");
document.write("s.charCodeAt(1) = "+s.charCodeAt(1)+"<BR/>");
document.write("s.concat(' World', ' !') = "+s.concat(' World', ' !')+"<BR/>");
```

```
document.write("String.fromCharCode(72,69,76,76,79) = "+String.fromCharCode(72,69,76,76,79)+"<BR
document.write("s.indexOf('el') = "+s.indexOf('el')+"<BR/>");
document.write("s.lastIndexOf('l') = "+s.lastIndexOf('l')+"<BR/>");
document.write("s.match('[aeiou]') = "+s.match('[aeiou]')+"<BR/>");
document.write("s.replace('l', 'L') = "+s.replace('l', 'L')+"<BR/>");
document.write("s.search('[aeiou]') = "+s.search('[aeiou]')+"<BR/>");
document.write("s.slice(2,4) = "+s.slice(2,4)+"<BR/>");
document.write("s.slice(2) = "+s.slice(2)+"<BR/>");
document.write("s.slice(-3) = "+s.slice(-3)+"<BR/>");
document.write("s.split('e') = "+s.split('e')+"<BR/>");
document.write("s.substr(2,2) = "+s.substr(2,2)+"<BR/>");
document.write("s.substring(2,4) = "+s.substr(2,4)+"<BR/>");
document.write("s.toLowerCase() = "+s.toLowerCase()+"<BR/>");
document.write("s.toUpperCase() = "+s.toUpperCase()+"<BR/>");
document.write("s.valueOf() = "+s.valueOf()+"<BR/>");
</script>
</body>
</html>
```

執行結果

```
s = 'Hello's.constructor = function String() { [native code] }
s.length = 5
s.prototype = undefined
s.charAt(1) = e
s.charCodeAt(1) = 101
s.concat(' World', ' !') = Hello World !
String.fromCharCode(72,69,76,76,79) = HELLO
s.indexOf('el') = 1
s.lastIndexOf('l') = 3
s.match('[aeiou]') = e
s.replace('l', 'L') = HeLlo
s.search('[aeiou]') = 1
s.slice(2,4) = ll
s.slice(2) = llo
s.slice(-3) = llo
s.split('e') = H,llo
s.substr(2,2) = ll
s.substring(2,4) = llo
s.toLowerCase() = hello
s.toUpperCase() = HELLO
s.valueOf() = Hello
```

正規表達式

正規語法 (Regular Grammar) 是一種相當簡單的語法，這種語法被 Perl 語言成功的用於字串比對，接著成為重要的程式設計工具。 此種標準的正規語法後來被稱為正則表達式 (Regular Expression)。目前，大部分的語言都已納入正則表達式的函式庫， 正規表達是可以說是程式設計師必定要瞭解的工具，也就是常識的一部分。系統程式設計師更應該要瞭解正則表達式， 因為正規語法是程式語言當中，用來描述基本詞彙 (Vocabulary)，並據以建構詞彙掃描器 (Lexer) 的基礎語法， Lexer 是編譯器的基本元件之一。

假如我們要用正則表達式描述整數數字，那麼，可以用 [0123456789]+ 這個表達式，其中的中括號 [與] 會框住一群 字元，用來代表字元群，加號 + 所代表的是重複 1 次或以上，因此，該表達式就可以描述像 3702451 這樣的數字。 然而，在正則表達式中，為了更方便撰寫，於是允許用 [0-9]+ 這樣的式子表達同樣的概念，其中的 0-9 其實就 代表了 0123456789 等字元，這是一種簡便的縮寫法。甚至，可以再度縮短後以 []+ 代表，其中的 就代表 數字所成的字元集合。

利用範例學習是理解正則表達式的有效方法，表格 1 就顯示了一些具有代表性的正則表達式範例。

表格 1. 正則表達式的範例

語法 正	規表達式 範	例
整數 [0	-9]+ 37	04

有小數點的實數 [	0-9]+.[0-9]+ 7	.93
英文詞彙	[A-Za-z]+	Code
變數名稱	[A-Za-z_][A-Za-z0-9_]*	_counter
Email	[a-zA-Z0-9_]+@[a-zA-Z0-9._]+	ccc@kmit.edu.tw
URL	http://[a-zA-Z0-9./_]+	http://ccc.kmit.edu.tw/mybook/

為了協助讀者理解這些範例，我們有必要對範例中的一些正規表達式符號進行說明。

在實數的範例中，使用 . 代表小數點符號 .，不熟悉正則表達式的讀者一定覺得奇怪，為何要加上斜線符號 呢？ 這是因為在正則表達式當中，有許多符號具有特殊意義，例如點符號 . 是用來表示任意字元的，星號 * 是代表 0 次 或以上，加號 + 代表一次或以上，在正則表達式當中，有許多這類的特殊字元，因此用斜線 代表跳出字元，就像 C 語言當中 printf 函數內的用途一樣。因此，當我們看到 符號時，必須繼續向後看，才能知道其所代表的意義。 表格 2 列出了正則表達式當中大部份的重要符號之意義，以供讀者參考。

表格 2. 正則表達式當中的符號之意義

字元 描　　述

^	比對開始位置
$	比對結束位置。
*	零次或以上
+	一次或以上
?	零次或一次
{n}	n 次。
{n,}	n 次或以上
{n,m}	n 到 m 次
?	非貪婪模式
.	比對除""之外字元
(?:pattern)	比對 pattern 樣式

(?=pattern)	正向預查，例如，"Windows (?=95|98|NT|2000)" 可比對到 "Windows 2000" 中的 "Windows"，但不能比對 Windows XP 中的 Windows。
(?!pattern)	負向預查，例如 "Windows(?!95|98|NT|2000)" 能比對 "Windows XP"中的 "Windows"，但不能比對 "Windows 2000" 中 的"Windows"。
x|y	比對 x 或 y。
[xyz]	包含 xyz 等字元。
[^xyz]	不包含 xyz 等字元。
[a-z]	字元範圍 a-z。
[^a-z]	不包含字元範圍 a-z。
\b	比對一個單詞邊界，也就是指單詞和空格間的位置。例如，"er"可以比對"never"中的"er"，但不能比對"verb"中的"er"。
\B	比對非單詞邊界。"er"能比對"verb"中的"er"，但不能比對"never"中的"er"。
\cx	比對由x指明的控制字元。例如，-M或回車符。x的值必須為A-Z或a-z之一。否則，將c視

	為一個原義的"c"字元。
\d	比對一個數位字元。等價於[0-9]。
\D	比對一個非數位字元。等價於[^0-9]。
\f	比對一個換頁符。等價於0c和。
\n	比對一個分行符號。等價於0a和。
\r	比對一個回車符。等價於0d和。
\s	比對任何空白字元，包括空格、定位字元、換頁符等等。等價於[]。
\S	比對任何非空白字元。等價於[^\f\n\r\t\v]。
\t	比對一個定位字元。等價於09和。
\v	比對一個垂直定位字元。等價於0b和。
\w	比對包括底線的任何單詞字元。等價於"[A-Za-z0-9_]"。
\W	比對任何非單詞字元。等價於"[^A-Za-z0-9_]"。

\xn	比對n，其中n為十六進位轉義值。十六進位轉義值必須為確定的兩個數位長。例如，"41"比對"A"。"041"則等價於"04"&"1"。規則運算式中可以使用ASCII編碼。
\num	比對num，其中num是一個正整數。對所獲取的比對的引用。例如，"(.)"比對兩個連續的相同字元。
\n	標識一個八進制轉義值或一個向後引用。如果，則n為向後引用。否則，如果n為八進位數字(0-7)，則n為一個八進制轉義值。
\nm	標識一個八進制轉義值或一個向後引用。 如果，則nm為向後引用。如果，則n為一個後跟文字m的向後引用。如果前面的條件都不滿足，若n和m均為八進位數字(0-7)，則將比對八進制轉義值nm。
\nml	如果n為八進位數字(0-3)，且m和l均為八進位數字(0-7)，則比對八進制轉義值nml。
\un	比對n，其中n是一個用四個十六進位數位表示的Unicode字元。例如，00A9比對版權符號（©）。

正則表達式在許多語言當中 (像是 Java, C#, Ruby, Python 等) 都已經有支援良好的函式庫。

```
<html>
```

```
<body>

<script type="text/javascript">
document.write("<pre>\n");
String.prototype.trim = function() { return this.replace(/(^\s*)|(\s*$)/g, ""); }
document.write('" abc '.trim()='|"+' abc '.trim()+"|\n");
var re = new RegExp("\\d+", "gi");
var str = "name:john age:20 birthday:1990/8/31";
var m = null;
while (m = re.exec(str))
  document.write(m + "\n");
var p = parse(str);
document.write("p.name="+p.name+" age="+p.age+" year="+p.year+" month="+p.month+" day="+p.day);
document.write("</pre>\n");

function parse(data) {
    var e=new RegExp("name:(\\w+) age:(\\d+) birthday:(\\d+)/(\\d+)/(\\d+)", "gi");

    if (data.match(e)) {
        return  {exp: RegExp['$&'],
                name: RegExp.$1,
```

```
            age:RegExp.$2,
            year:RegExp.$3,
            month:RegExp.$4,
            day:RegExp.$5};
    }
    else {
        return null;
    }
}
</script>

</body>
</html>
```

執行結果

```
' abc '.trim()=|abc|
20
1990
8
31
```

```
p.name=john age=20 year=1990 month=8 day=31
```

eval 函數

```
<script type="text/javascript">
var a=3, b=5;
alert('a+b='+eval('a+b'));
</script>
```

參考文獻

- Wikipedia:Regular expression -- http://en.wikipedia.org/wiki/Regular_expression
- 維基百科:正規表式 -- http://zh.wikipedia.org/zh-tw/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F
- 鳥哥的 Linux 私房菜:第十二章、正規表示法與文件格式化處理 -- http://linux.vbird.org/linux_basic/0330regularex.php
- 石頭閒語:Regular Expression (RegExp) in JavaScript -- http://blog.roodo.com/rocksaying/archives/2670695.html
- 字串樣版 Regexp: 兼談長線學習投資 -- http://www.cyut.edu.tw/~ckhung/b/re/
- 在 C 程式中，使用 Regex (Regular Expression) library -- http://blog.roodo.com/rocksaying/archives/3866523.html

動態網頁 - JavaScript 在瀏覽器中控制 DOM 網頁物件

JavaScript、Browser 與 DOM

JavaScript 之所以重要，是因為 JavaScript 是瀏覽器當中規定的標準語言。這是因為早期的瀏覽器 Netscape 將 JavaScript 內建於其中， 以變能夠進行一些互動程式的效果，因此才讓 JavaScript 成為瀏覽器的的標準語言。

在瀏覽器中，要用 JavaScript 設計出互動效果，畢需借助 DOM (Document Object Model) 這個 HTML 的抽象化物件模型，這個模型將 整個 HTML 文件視為一個樹狀結構的物件，於是 JavaScript 可以透過操控物件屬性的方式，達成某些互動性的顯示效果。這種互動性的 顯示效果形成了瀏覽器特有的視覺化互動模式，可以說是近 15 年來 Web 的發展重點之所在。

接著、就讓我們透過一些簡單的程式範例，看看如何用 JavaScript 操控 DOM 物件模型，以便完成這種互動效果。

取得內容：innerText 與 innerHTML

以下範例會取得 hi 節點的 innerText 與 innerHTML 顯示出來，請觀察其不同點。

```
<html>
  <head>
    <title>節點存取示範</title>
  </head>
  <body>
    <div id ="hi"><b>你好!</b></div>
    <input type="button" value="hi.innerText"
           onclick="alert(document.getElementById('hi').innerText)");
    <input type="button" value="hi.innerHTML"
           onclick="alert(document.getElementById('hi').innerHTML)");
  </body>
</html>
```

顯示與隱藏（Show and hide）

```
<html>
  <head>
    <title>範例 -- 顯示隱藏</title>
  </head>
  <body>
    <div id ="hi"><b>你好!</b></div>
```

```
    <input type="button" value="hi.show()" onclick="document.getElementById('hi').style.visibility='visible'");
    <input type="button" value="hi.hide()" onclick="document.getElementById('hi').style.visibility='hidden'");
  </body>
</html>
```

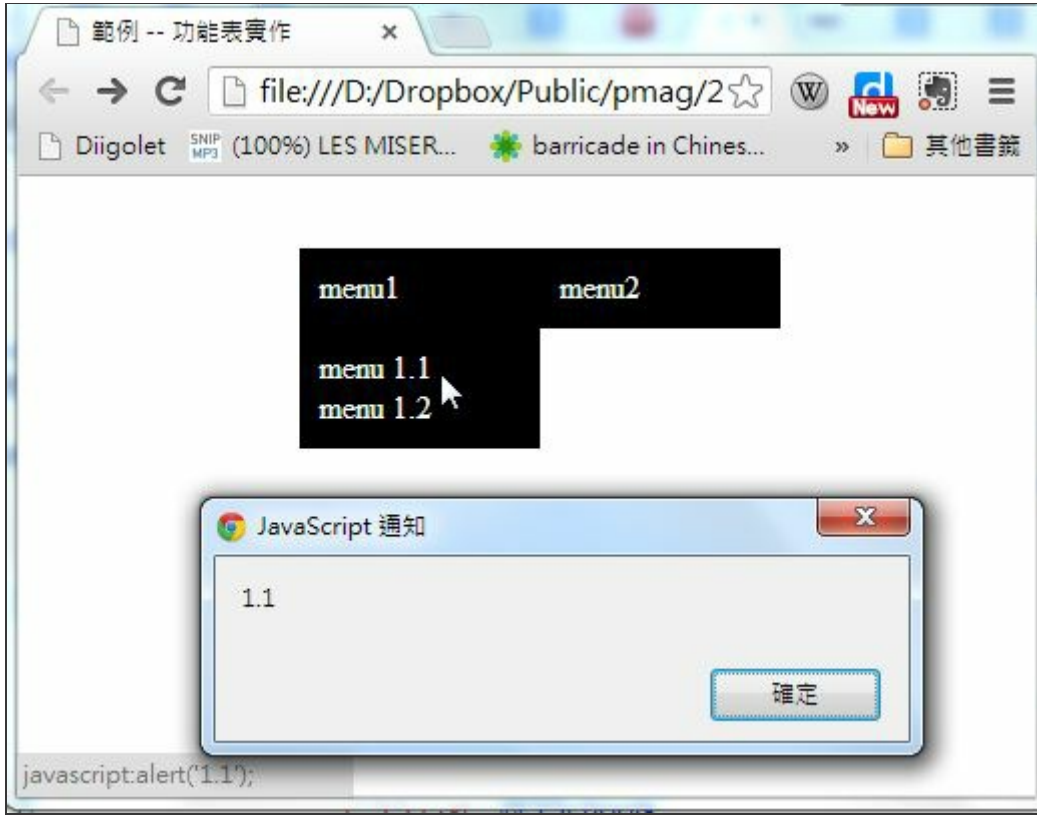
功能表程式

JavaScript 是唯一被各家瀏覽器所共同支援的程式語言，因此在設計網站的時候，我們如果不採用像 Flash 或 Silverlight 這樣的外掛技術，就必須採用 JavaScript 來設計互動式網頁。

在 node.js 這樣的伺服端 JavaScript 開發平台推出之後，我們就能夠採用 JavaScript 同時設計 Client 端與 Server 端的程式，這樣的模式相當的吸引人，我們會在後續的文章中介紹這樣的網站設計方法。

在本節當中，我們將透過 JavaScript 設計一個互動網頁的功能表，以便展示瀏覽器中的 JavaScript 程式是如何運作的。

以下是一個功能表的程式的執行結果，當我們的滑鼠移到功能項上時，就會浮現子功能表，而當我們點下子功能表中的項目時，就會出現一個 alert 視窗，顯示該功能子項被點選的訊息。



功能表程式的執行畫面

以下是這個網頁的原始 HTML 程式碼，其中 <style>...</style> 部分是 CSS 原始碼，而 <script ...</script> 部

分則是 JavaScript 程式碼。

```
<html>
<head>
<title>範例 -- 功能表實作</title>
<style>
.menu   { background-color:black; color:white; padding:10px;
        vertical-align:top; width:100px; list-style-type:none; }
.menu a { color:white; text-decoration:none; }
</style>
<script type="text/javascript">
function show(id) {
  document.getElementById(id).style.visibility='visible';
}

function hide(id) {
  document.getElementById(id).style.visibility='hidden';
}
</script>
  </head>
  <body onload="JavaScript:hide('popup1');hide('popup2');">
```

```
    <ul onmouseover="show('popup1');"  onmouseout="hide('popup1')"
       style="position:absolute; left:100px; top:20px">
      <li id="menu1" class="menu">menu1</li>
      <ul id="popup1" class="menu">
        <li><a href="JavaScript:alert('1.1');">menu 1.1</a></li>
        <li><a href="JavaScript:alert('1.2');">menu 1.2</a></li>
      </ul>
    </ul>
    <ul onmouseover="show('popup2');" onmouseout="hide('popup2')"
       style="position:absolute; left:220px; top:20px">
      <li id="menu2" class="menu">menu2</li>
      <ul id="popup2" class="menu">
        <li><a href="JavaScript:alert('2.1');">menu 2.1</a></li>
        <li><a href="JavaScript:alert('2.2');">menu 2.2</a></li>
        <li><a href="JavaScript:alert('2.3');">menu 2.3</a></li>
      </ul>
    </ul>
  </body>
</html>
```

雖然以上程式只是一個小小的功能表程式碼，但是要能夠讀懂，而且寫得出來，卻要懂相當多的技術才

行，這些技術包含 HTML, CSS , JavaScript 與 Document Object Model (DOM)。

程式解析

在 HTML 的一開始，我們用以下語法描述了功能表所需要的 CSS 樣式，當我們套用在像 <li id="menu2" class="menu">menu2 這樣的 HTML 項目上時，就會呈現比較好看的功能表排版格式，而這正是 CSS 樣式的功用。

```
<style>
.menu   { background-color:black; color:white; padding:10px;
        vertical-align:top; width:100px; list-style-type:none; }
.menu a { color:white; text-decoration:none; }
</style>
```

以上的 CSS 語法中，要求功能表要以黑底白字的方式 (background-color:black; color:white;) 顯示，邊緣補上 10 點的空白(padding:10px;)，而且是以向上靠攏 (vertical-align:top;) 的方式，每個功能表的寬度都是 100 點 (width:100px;)，然後不要顯示項目前面的點符號 (list-style-type:none;)。

接著是一段 JavaScript 程式碼的語法，定義了 show(id) 與 hide(id) 這兩個函數，我們可以用這兩個函數在適當的時候讓功能表顯示出來或者是隱藏掉，這樣才能做到「浮現」的功能。

```
<script type="text/javascript">
```

```
function show(id) {
  document.getElementById(id).style.visibility='visible';
}

function hide(id) {
  document.getElementById(id).style.visibility='hidden';
}
</script>
```

然後，開始進入 HTML 的 body 區塊，其中定義了兩組功能表，第一組的內容如下：

```
    <ul onmouseover="show('popup1');"  onmouseout="hide('popup1')"
       style="position:absolute; left:100px; top:20px">
      <li id="menu1" class="menu">menu1</li>
      <ul id="popup1" class="menu">
        <li><a href="JavaScript:alert('1.1');">menu 1.1</a></li>
        <li><a href="JavaScript:alert('1.2');">menu 1.2</a></li>
      </ul>
    </ul>
```

上述區塊最外層的 <ul>... 定義了整個功能表的結構，是由功能母項 <li id="menu1"

class="menu">menu1 與子功能表 <ul id="popup1" class="menu">... 所組合而成的，而 <ul onmouseover="show('popup1');" onmouseout="hide('popup1')" style="position:absolute; left:100px; top:20px"> 這一段除了定義了該功能表要顯示在絕對位置 (100,20) 的地方之外，還定義了 onmouseover 與 onmouseout 的事件，這兩個事件讓功能表可以在滑鼠移入時顯示出來，然後在滑鼠移出時隱藏起來，因而做到了浮現式功能表所要求的條件。

由於我們在 <li>... 內的超連結 menu 2.1 使用了 JavaScript 語法，因此在該超連結被點選時，就會有警告視窗顯示 2.1 的訊息，這個訊息僅僅是讓我們知道該功能項被點選了而已，沒有真實的功能。

同樣的、第二個功能表的程式碼也是非常類似的，請讀者看看是否能夠讀者其內容。

```
<ul onmouseover="show('popup2');" onmouseout="hide('popup2')"
    style="position:absolute; left:220px; top:20px">
  <li id="menu2" class="menu">menu2</li>
  <ul id="popup2" class="menu">
    <li><a href="JavaScript:alert('2.1');">menu 2.1</a></li>
    <li><a href="JavaScript:alert('2.2');">menu 2.2</a></li>
    <li><a href="JavaScript:alert('2.3');">menu 2.3</a></li>
  </ul>
</ul>
```

看到這裡，讀者應該大致理解了上述功能表網頁的運作原理，但事實上、我們還漏掉了一行重要的程式碼，那就是 <body onload="JavaScript:hide('popup1');hide('popup2');"> 這一行，這一行讓浮現功能表能在一開始就處於隱藏狀態，才不會一進來就看到兩個功能表都浮現出來的錯誤情況。

小結

從上述範例中，您可以看到瀏覽器中的 JavaScript ，通常是透過調整網頁某些項目的 CSS 屬性，以達成互動性的功能，這種互動網頁技術，事實上是結合了 HTML+CSS+JavaScript 等技術才能達成的功能，因此這三項技術在瀏覽器當中幾乎是合為一體、可以說是缺一不可的。

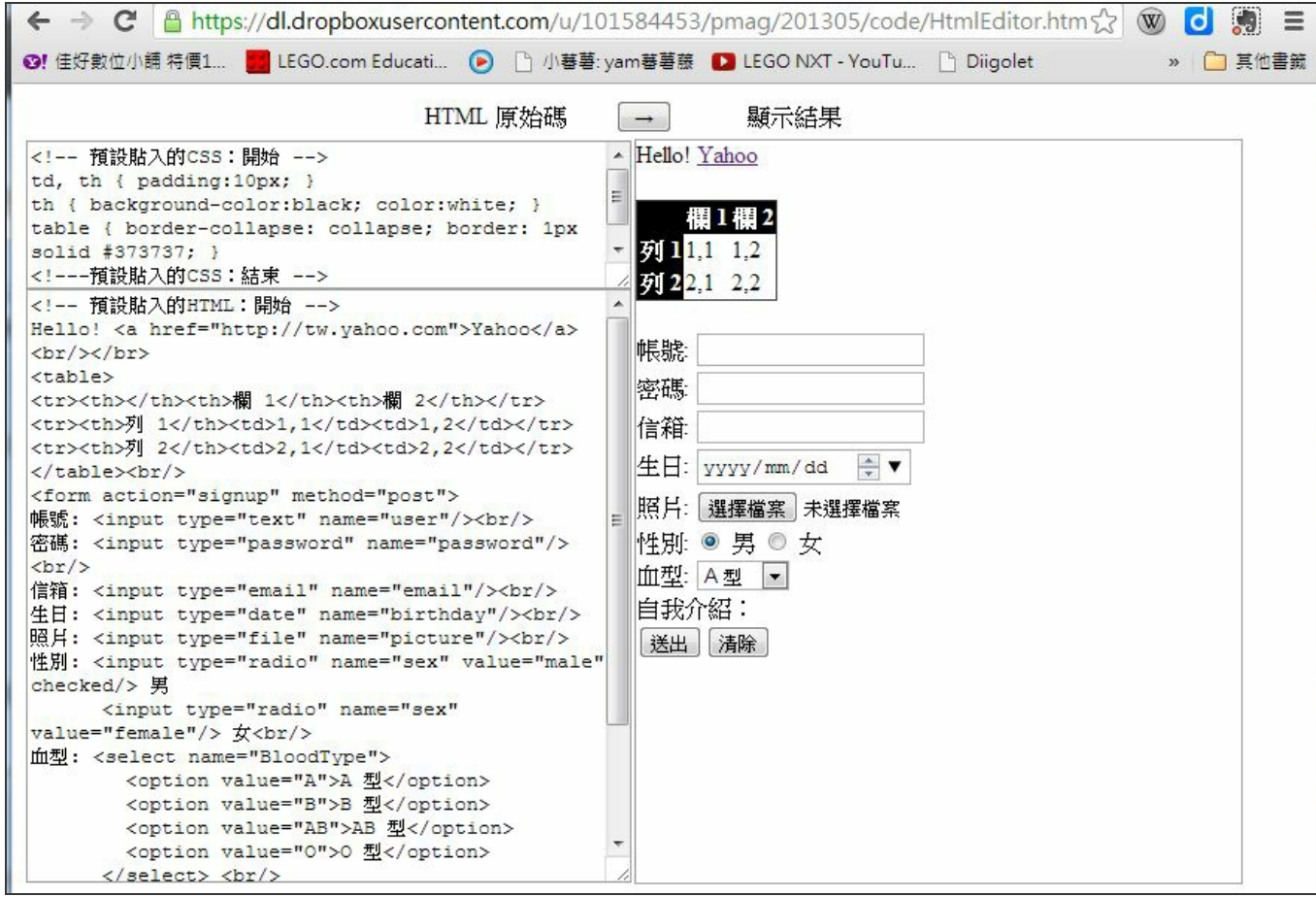
在本期中我們說明了互動式網頁的設計方式，但這樣的設計方式非常冗長，對程式人員而言是很大的負擔，因此在互動網頁興起之後，就逐漸出現了各式各樣的互動性 JavaScript 框架，也就是現成的 JavaScript 函式庫，讓我們可以輕易做出很好的互動性，像是 jQuery, ExtJS, YUI, Prototype, Dojo 等互動性函式庫，以便減輕 JavaScript 程式人員的負擔，增加程式員的生產力，在下一期當中，我們將使用最常被使用的 jQuery 框架，再度說明互動網頁的寫法，我們下期見！

HTML 編輯器

在本節當中，我們將透過 JavaScript 設計一個 HTML 編輯器，讓您可以直接在瀏覽器當中看到 HTML 網頁的呈現結果， 筆者認為這個範例對學習動態網頁設計是一個非常簡單、卻又很有啟發性的程式。因為他很明顯的展現了動態網頁的原理。

以下是該 HTML 編輯器的執行結果，當我們在 CSS 與 HTML 區塊輸入對應的原始碼之後，就可以按下

「→」按鈕，然後在 呈現的 <div id="showbox" ... 區塊看到兩者搭配時的呈現結果。



HTML 編輯器的執行畫面

您也可以點選下列連結以實際檢視該網頁：

- HTML 編輯器：https://dl.dropboxusercontent.com/u/101584453/pmag/201305/code/HtmlEditor.htm

以下是這個網頁的原始 HTML 程式碼，其中有兩大段是我們預設填入的 CSS 與 HTML 原始碼，這兩段原則上可以去掉，但是 為了測試方便起見，我們就留在檔案中，請讀者閱讀的時候仔細區分之。

```
<html>
<style>
textarea, #showbox { border: 1px solid #9f9f9f; }
</style>
<style id="showboxstyle">
</style>
<script type="text/javascript">
function convert() {
  var cssbox  = document.getElementById("cssbox");
  var editbox = document.getElementById("editbox");
  var showbox = document.getElementById("showbox");
  var showboxstyle = document.getElementById("showboxstyle");
  showbox.innerHTML = editbox.value;
  showboxstyle.innerHTML = cssbox.value;
```

```
}
</script>
<body>
<form>
<table width="95%" style="border-collapse: collapse; border: 0px;"><tr>
<tr><td colspan="2" style="text-align:center">HTML 原始碼        <input type="button" value=" → " onclick="Java
<td width="50%">
<textarea id="cssbox" style="width:100%; height:100px;" >
<!-- 預設貼入的CSS：開始 -->
td, th { padding:10px; }
th { background-color:black; color:white; }
table { border-collapse: collapse; border: 1px solid #373737; }
<!---預設貼入的CSS：結束 -->
</textarea>
<textarea id="editbox" style="width:100%; height:400px;">
<!-- 預設貼入的HTML：開始 -->
Hello! <a href="http://tw.yahoo.com">Yahoo</a><br/></br>
<table>
<tr><th></th><th>欄 1</th><th>欄 2</th></tr>
<tr><th>列 1</th><td>1,1</td><td>1,2</td></tr>
<tr><th>列 2</th><td>2,1</td><td>2,2</td></tr>
```

```
</table><br/>
<form action="signup" method="post">
帳號: <input type="text" name="user"/><br/>
密碼: <input type="password" name="password"/><br/>
信箱: <input type="email" name="email"/><br/>
生日: <input type="date" name="birthday"/><br/>
照片: <input type="file" name="picture"/><br/>
性別: <input type="radio" name="sex" value="male" checked/> 男
    <input type="radio" name="sex" value="female"/> 女<br/>
血型: <select name="BloodType">
      <option value="A">A 型</option>
      <option value="B">B 型</option>
      <option value="AB">AB 型</option>
      <option value="O">O 型</option>
    </select> <br/>
自我介紹： <br/>
<input type="submit" value="送出"/><input type="reset" value="清除"/><br/>
</form>
<!-- 預設貼入的HTML：結束 -->
</textarea>
```

```
</td>
<td>
<div id="showbox" style="width:100%; height:500px;">
</div>
</td>
</tr></table>
</form>
</body>
</html>
```

程式解析

上述程式當中有三個主要的區塊，分別是：

1. CSS 填入區塊： <textarea id="cssbox" ...</textarea>
2. HTML 填入區塊：<textarea id="editbox" ...</textarea>
3. HTML 顯示區塊：<div id="showbox" ...

此網頁的核心程式部分真的很簡單，只有如下短短的一小段：

1. 利用 var editbox = document.getElementById("editbox"); 取得 HTML 區塊內容
2. 利用 showbox.innerHTML = editbox.value; 這個指令將該 HTML 原始碼填入 showbox 當中

這樣就完成顯示 HTML 的動作了。

```
function convert() {
  var cssbox  = document.getElementById("cssbox");
  var editbox = document.getElementById("editbox");
  var showbox = document.getElementById("showbox");
  var showboxstyle = document.getElementById("showboxstyle");
  showbox.innerHTML = editbox.value;
  showboxstyle.innerHTML = cssbox.value;
}
```

但是這樣作並沒有加入 cssbox 的內容到 HTML 當中，因此我們加入了下列原始碼：

1. 在整個網頁的頭部事先用 <style id="showboxstyle">...</style> 這個標記加入一個 CSS style 顯示區塊
2. 利用 showboxstyle.innerHTML = cssbox.value 這個指令將 cssbox 的內容填入到該表頭的 style 區塊中

這樣就達成了套用 CSS 內容到網頁中的目的，完成了整個 HTML 編輯器的功能。

小結

在本節中，我們用非常簡單的程式，建構了一個 HTML 編輯器。事實上我們只不過是把網頁內容從編輯區域移動到顯示區域，然後流覽器就會 自動解釋這些內容進行呈現了。

在本系列的文章中，關於瀏覽器部分的 JavaScript 程式，我們將至此告一個段落，在下一期當中，我們將開始進入伺服端的 javascript 程式。 我們將利用 node.js 這個伺服端 JavaScript 執行平台，進一步探索 JavaScript 程式的奧秘，我們下期見！

參考文獻

- http://stackoverflow.com/questions/1720320/how-to-dynamically-create-css-class-in-javascript-and-apply
- http://dev.opera.com/articles/view/dynamic-style-css-javascript/

Node.js 命令列 - 讓 JavaScript 作為一般程式執行

Node.js 通常被用來當成與 PHP、Ruby on Rail 或 ASP.NET 類似的平台，用來撰寫 Web 的網站程式， 但是這種觀點主要是使用 node.js 的 Web 函式庫功能，直接從 Web 程式的實作開始，而非從基礎的 JavaScript 開始，我們認為這樣的方式並不好，因為入門瓶頸太高，很容易會造成讀者的學習障礙。

事實上、Node.js 當中包含了一個由 Google Chrome 計畫中所釋出的 JavaScript 編譯引擎 -- V8，這是一個速度很快的 JavaScript 執行工具，在本章中，我們將從命令列基本操作開始，將 node.js 單純當作一個 JavaScript 解譯器使用，以便透過 node.js 進一步學習 JavaScript 的語法與基本函式庫的用法。

Node.js 的安裝

本文使用 node.js 0.10.0 版示範，您可以從 node.js 官網上下載到最新的版本，以下是 node.js 的網址。

- http://nodejs.org/

node.js 0.10 之後的版本，在 windows 當中的安裝非常容易，只要一直點選下一步，就能完全安裝完畢。

Node.js 的互動操作環境

安裝完 Node.js 之後，您就可以啟動 Node.js 的互動操作環境，然後開始輸入一些基本的 JavaScript 程式，以便學習 JavaScript 的語法，以下是筆者的一個操作畫面。

圖、Node.js 的操作畫面

```
> x = 3
3
```

```
> y = 5
5
> z = x+y
8
> w = x*y
15
> x, y, z, w
15
> eval("x+y+z+w")
31
> s = "Hello!"
'Hello!'
> t = x+s
'3Hello!'
> s+t
'Hello!3Hello!'
> score = 80
80
> add = function(a,b) { return a+b; }
[Function]
> add(3,5)
```

```
8
> add(374, 567)
941
> sum = function(n) {
... s = 0;
... for (i=1; i<=n; i++)
...   s += i;
... return s
... }
[Function]
> sum(10)
55
> sum(100)
5050
> max = function(a,b) {
... if (a>b)
...   return a;
... else
...   return b;
... }
[Function]
```

```
> max(3,5)
5
> max(9,2)
9
```

從以上的操作中，您可以看到我們不只可以進行基本的運算操作，也可以直接在裏面定義函數，以及呼叫這些函數。

當然、您也可以將 JavaScript 的程式存檔，然後用 node.js 去執行，

Node.js 的命令列工具

雖然互動操作環境感覺很方便，但是要撰寫真正的程式時，就不太好用了，此時您可以使用 Node 的 命令列工具 -- Node.js command prompt 來執行程式。舉例而言，我們可以先撰寫了一個 hello.js 的程式， 如下所示：

檔案：hello.js

```
console.log("Hello!");
```

然後啟動 Node.js command prompt 去執行該程式，以下是我們的執行 hello.js 與下述 sum.js 程式的結果：

圖、用 Node.js command prompt 執行程式

檔案：sum.js

```
function sum(n) {
  var s = 0;
  for (i=1; i<=n; i++)
```

```
    s+=i;
  return s;
}

console.log("sum(10)="+sum(10));

console.log("sum(100)="+sum(100));
```

於是、透過 Node.js 的開發環境，我們可以將原本只能在瀏覽器中測試的 JavaScript 程式， 放到 Node.js 當中執行，這樣就能更方便的學習 JavaScript 的語法與函式庫，大大的提高了 我們學習 JavaScript 的方便程度。

目前、筆者在撰寫 JavaScript 程式時，通常會先用 Node.js 進行測試，沒有問題之後，在放到 瀏覽器當中去執行，也就是將 Node.js 當作單元測試的工具，這對筆者的幫助很大。

接著、讓我們透過 Node.js 平台，來學習一些 JavaScript 基本函式庫的用法，首先讓我們來看看 動態解譯函數 eval()。

Eval 函數

Eval 函數可以用來執行任何 JavaScript 程式碼，這是解譯式語言特有的函數，以下是一個使用 eval() 函數的程式範例。

檔案：eval.js

```
x = 3;
y = 5;
console.log("eval('x+y')="+eval('x+y'));

function max(x,y) {
    return (x>y)?x:y;
}

console.log("eval('max')="+eval('max'));

console.log("eval('max(x,y)')="+eval('max(x,y)'));

console.log("eval('max(2*x,y)')="+eval('max(2*x,y)'));
```

執行結果：

```
D:\code\node>node eval.js
eval('x+y')=8
eval('max')=function max(x,y) {
```

```
    return (x>y)?x:y;
}
eval('max(x,y)')=5
eval('max(2*x,y)')=6
```

雖然 eval() 通常是在解釋環境之下才具備的功能，但是在 Node.js 這種動態編譯環境之下， 卻也能夠順利的執行，這對那些開發 C/C++, Java, C# 等語言的程式人員而言，勢必會感到 非常的羨慕才對。

註：如果您曾經在資料結構的課程上，嘗試去寫運算式處理的中序轉前序程式，只是為了計算 運算式的值，就會知道這樣的功能在那些編譯式語言當中是多麼的難寫，而要寫一個支援任意 運算式執行的程式，基本上就是重新撰寫一個語言解譯器，那就相當於重新建造了整套開發工具， 這是非常非常困難的。

接著、讓我們來看看 Node.js 當中的一些重要的函式庫與觀念，包含「輸出入、檔案、以及模組的定義與引用等」。

讀取檔案

程式：readfile.js

```
var fs = require('fs'); // 引用檔案物件
var file = fs.readFileSync(process.argv[2], "utf8"); // 讀取檔案
console.log(file); // 顯示在螢幕上
```

執行結果

```
D:\code\node>node readfile.js readfile.js
var fs = require('fs'); // 引用檔案物件
var file = fs.readFileSync(process.argv[2], "utf8"); // 讀取檔案
console.log(file); // 顯示在螢幕上
```

寫入檔案

我們用複製檔案的程式來示範寫入檔案的功能。

程式：copyfile.js

```
var fs = require('fs');
var file = fs.readFileSync(process.argv[2]);
console.log(file);
fs.writeFileSync(process.argv[3], file);
```

執行結果：

```
D:\code\node>node copyfile.js copyfile.js copyfile.js.bak
```

```
<Buffer 76 61 72 20 66 73 20 3d 20 72 65 71 75 69 72 65 28 27 66 73 27 29 3b 0a
76 61 72 20 66 69 6c 65 20 3d 20 66 73 2e 72 65 61 64 46 69 6c 65 53 79 6e 63 28
 ...>
```

模組定義與引用

當您用 javascript 寫出物件或函式庫時，可以提供給其他程式使用。

在 node.js 當中，模組的定義大致有兩種類型，一種是「匯出物件」的靜態模組，另一種是匯出「建構函數」的動態模組。

靜態模組：匯出物件

以下是一個匯出物件的靜態模組定義。

模組定義：circle.js

```
var math = {
   PI:3.14,
   square:function(n) {
      return n*n;
   }
```

```
}

module.exports = math;
```

接著您可以使用 require 這個指令動態的引入該模組，注意 require 必須採用相對路徑，即使在同一個資料夾底下，也要加上 ./ 的前置符號，代表在目前資料夾之下。以下是一個引用上述模組的範例。

模組使用：CircleTest.js

```
var m=require("./math");

console.log("PI=%d square(3)=%d", m.PI, m.square(3));
```

執行結果：

```
D:\Dropbox\Public\pmag\201306\code>node mathTest
PI=3.14 square(3)=9
```

動態模組：匯出建構函數

以下是一個定義圓形 circle 的物件。

模組定義：circle.js

```
var PI = 3.14;
Circle = function (radius) {
    this.radius = radius
    this.area = function() {
        return PI * this.radius * this.radius;
    }
};

module.exports = Circle;
module.exports.PI = PI;
```

在引用「匯出建構函數」的程式當中，由於取得的是建構函數，因此必須再使用 new 的方式建立物件之後才能使用 (例如以下的 c = new cir(5) 這個指令，就是在透過建構函數 cir() 建立物件。

模組使用：CircleTest.js

```
var cir = require('./circle');          //注意，./代表 circle 與此程式放在同一個資料夾底下。
var c = new cir(5);
console.log("cir.PI="+cir.PI);
console.log("c.PI="+c.PI);
console.log("c.area()="+c.area());
```

執行結果

```
D:\code\node>node circleTest.js
cir.PI=3.14
c.PI=undefined
c.area()=78.5
```

您現在應該可以理解為何我們要將 Circle 定義為一個函數了吧！這只不過 Circle 類別的建構函數而已， 當他被 module.exports = Circle 這個指令匯出時，就可以在 var cir = require('./circle') 這個指令 接收到建構函數，然後再利用像 var c = new cir(5) 這樣的指令，呼叫該建構函數，以建立出物件。

然後，您也應該可以看懂為何我們要用 module.exports.PI = PI 將 PI 獨立塞到 module.exports 裏了吧！ 因為只有這樣才能讓外面的模組在不執行物件建構函數 (不建立物件) 的情況之下就能存取 PI。

跨平台模組：定義各種平台均能使用的 JavaScript 模組

在很多開放原始碼的 JavaScript 專案模組中，我們會看到模組的最後有一段很複雜的匯出動作。舉例而言， 在 marked.js 這個將 Markdown 語法轉為 HTML 的模組最後，我們看到了下列這段感覺匪夷所思的匯出橋段， 這種寫法其實只是為了要讓這個模組能夠順利的在「瀏覽器、node.js、CommonJS 與其 Asynchronous Module Definition (AMD) 實作版的 RequireJS」等平台當中都能順利的使用這個模組而寫的程式碼而已。

```
/**
 * Expose
 */

marked.Parser = Parser;
marked.parser = Parser.parse;

marked.Lexer = Lexer;
marked.lexer = Lexer.lex;

marked.InlineLexer = InlineLexer;
marked.inlineLexer = InlineLexer.output;

marked.parse = marked;

if (typeof exports === 'object') {
  module.exports = marked;
} else if (typeof define === 'function' && define.amd) {
  define(function() { return marked; });
} else {
```

```
  this.marked = marked;
}

}).call(function() {
  return this || (typeof window !== 'undefined' ? window : global);
}());
```

對這個超複雜匯出程式有興趣的朋友可以看看以下的文章，應該就可以大致理解這種寫法的來龍去脈了。

- http://www.angrycoding.com/2012/10/cross-platform-wrapper-function-for.html

參考文獻

- Node.js 台灣社群協作電子書 -- http://book.nodejs.tw/
- Node入門 -- http://www.nodebeginner.org/index-zh-tw.html
- 深入浅出Node.js相关的内容 -- http://www.infoq.com/cn/master-nodejs
    - 深入浅出Node.js（一）：什么是Node.js -- http://www.infoq.com/cn/articles/what-is-nodejs
    - 深入浅出Node.js（三）：深入Node.js的模块机制 -- http://www.infoq.com/cn/articles/nodejs-module-mechanism
- node.js 檔案讀取 -- http://book.nodejs.tw/zh-tw/node_basic.html#id2
- http://stackoverflow.com/questions/2496710/nodejs-write-to-file
- Node.js, Require and Exports

- http://openmymind.net/2012/2/3/Node-Require-and-Exports/
- Node.js v0.8.5 Manual & Documentation/Modules
  - http://nodejs.org/api/modules.html#modules_modules
- Understanding Node.js' "require"
  - http://jherdman.github.com/2010-04-05/understanding-nodejs-require.html

Node.js -- Web 程式

第一個 Web 程式

程式：HelloWeb.js

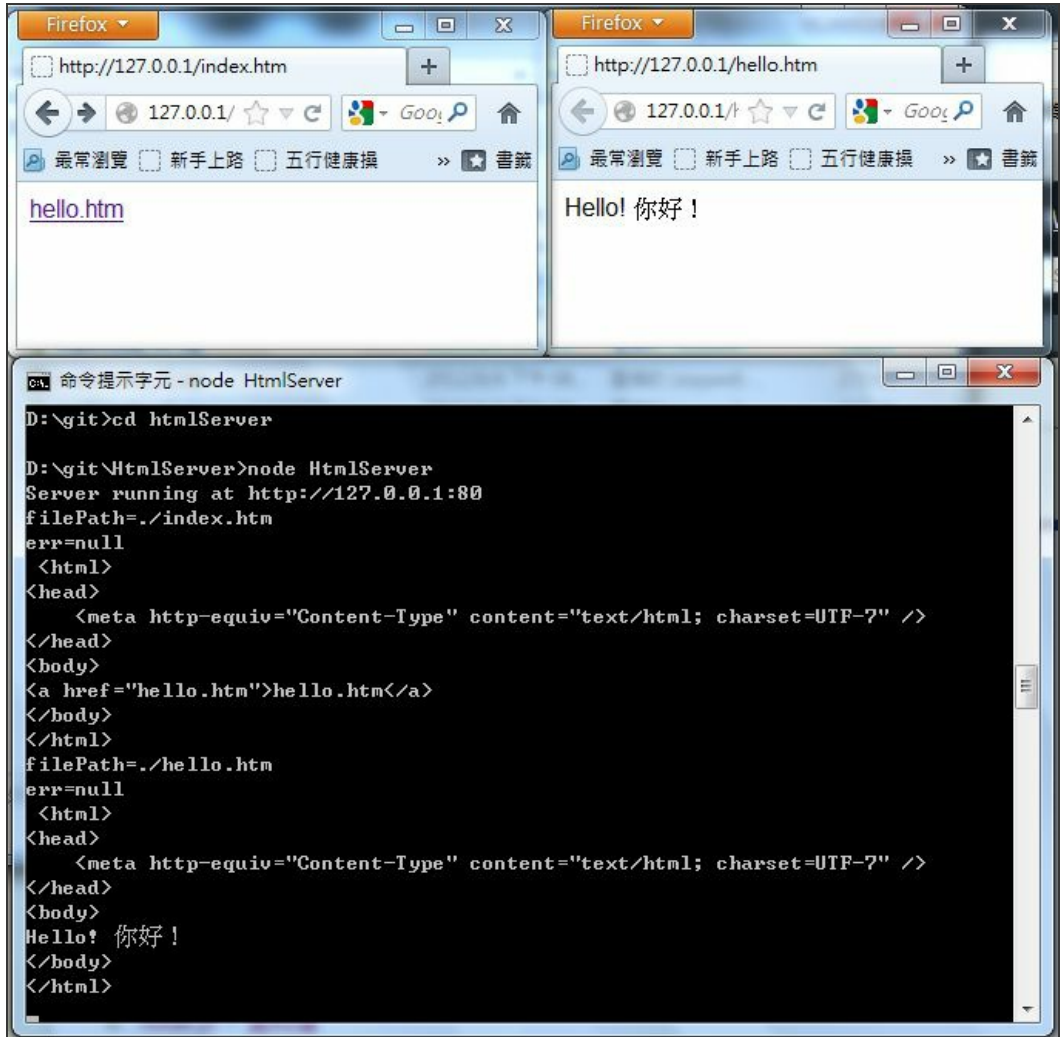
```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello Node.js!\n');
}).listen(80, '127.0.0.1');
console.log('Server running at http://127.0.0.1:80/');
```

執行結果

```
D:\ccc101\nodejs>node WebHello.js
Server running at http://127.0.0.1:80/
```

HtmlServer

執行結果：


Firefox
http://127.0.0.1/index.htm
127.0.0.1/
最常瀏覽 新手上路 五行健康操 書籤
hello.htm
Firefox
http://127.0.0.1/hello.htm
127.0.0.1/
最常瀏覽 新手上路 五行健康操 書籤
Hello! 你好！
命令提示字元 - node HtmlServer
D:\git>cd htmlServer
D:\git\HtmlServer>node HtmlServer
Server running at http://127.0.0.1:80
filePath=./index.htm
err=null
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-7" />
</head>
<body>
hello.htm
</body>
</html>
filePath=./hello.htm
err=null
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-7" />
</head>
<body>
Hello! 你好！
</body>
</html>

```
var server,
    ip   = "127.0.0.1",
    port = 80,
    http = require('http'),
    fs = require("fs"),
    folderPath = ".",
    url = require('url'),
    path,
    filePath,
    encode = "utf8";

server = http.createServer(function (req, res) {
    path = url.parse(req.url);
    filePath = folderPath + path.pathname;
    console.log("filePath="+filePath);
    fs.readFile(filePath, encode, function(err, file) {
      console.log("err="+err);
      if (err) {
          res.writeHead(404, {'Content-Type': 'text/plain'});
          res.end();
```

```
        return;
    }

    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(file);
    console.log(file);
    res.end();
  });
});

server.listen(port, ip);

console.log("Server running at http://" + ip + ":" + port);
```

簡易 WebServer

由於 node.js 採用 Event Queue 的方式執行 (類似 Win3.1 中的協同式多工)，而沒有採用 Thread 的方式，因此對所有 輸出入 IO 都應該盡可能採用非同步的方式執行，這樣才能讓 node.js 程式發揮最大的效能，否則就會因為 IO 而卡住。

因此以下兩個版本當中，以非同步版速度較快。

簡易 WebServer ：同步版（效能較差，不符合 node.js 的設計理念）

```
var path = require('path');
var fs = require("fs");
var qs = require('querystring');
var express = require("express");
var app = express();
app.listen(80);

var response = function(res, type, text) {
    res.writeHead(200, {'Content-Type': type});
    res.write(text);
    console.log(text);
    res.end();
}

app.get('*', function(req, res){
  try {
    var path = '.' + req.url;
    fs.stat(path, function(err, pathStat) {
        if (err) {
```

```
        response(res, "text/plain", err.toString());
        return;
    }
    if (pathStat.isFile()) {
        fs.readFile(path, "utf8", function(err, file) {
            response(res, "text/html", file);
        });
    } else if (pathStat.isDirectory()) {
        var dirPath = req.url;
        if (dirPath.substring(-1)!=="/")
            dirPath = dirPath+"/";
        var html = "<html><body><h1>"+req.url+"</h1>\n";
        fs.readdir(path, function(err, files) {
            for (f in files) {
                fname = files[f];
                filePath = dirPath+fname;
                html += "<li><a href='"+filePath+"'>"+fname+"</a></li>\n";
            }
            html += "<body></html>";
            response(res, "text/html", html);
        });
```

```
      }
    });
  } catch (err) {
    response(res, "text/plain", err.toString());
  }
});

console.log('start WebServer\n');
```

簡易 WebServer ：非同步版（效能較好，符合 node.js 的設計理念）

```
var path = require('path');
var fs = require("fs");
var qs = require('querystring');
var express = require("express");
var app = express();
app.listen(80);

var response = function(res, type, text) {
    res.writeHead(200, {'Content-Type': type});
    res.write(text);
```

```
    console.log(text);
    res.end();
}

app.get('*', function(req, res){
  try {
    var path = '.' + req.url;
    fs.stat(path, function(err, pathStat) {
        if (pathStat.isFile()) {
            fs.readFile(path, "utf8", function(err, file) {
                response(res, "text/html", file);
            });
        } else if (pathStat.isDirectory()) {
            var html = "<html><body>";
            fs.readdir(path, function(err, files) {
                for (f in files) {
                    fname = files[f];
                    filePath = req.url+fname;
                    html += "<li><a href='"+filePath+"'>"
                            +filePath+"</a></li>\n";
                }
```

```
            html += "<body></html>";
            response(res, "text/html", html);
          });
        }
    });
  } catch (err) {
    response(res, "text/plain", err.toString());
  }
});

console.log('start WebServer\n');
```

較完整的 WebServer（非同步版）

程式：WebServer2.js

```
var path = require('path');
var fs = require("fs");
var qs = require('querystring');
var express = require("express");
var app = express();
```

```
var mimeType = {
   "jpg":"image/jpeg",
   "gif":"image/gif",
   "png":"image/png",
   "svg":"image/svg",
   "zip":"application/zip",
   "pdf":"application/pdf",
   "xls":"application/vnd.ms-excel",
   "ppt":"application/vnd.ms-powerpoint",
   "doc":"application/msword",
   "htm":"text/html",
   "html":"text/html"
};

function getMimeType(ext) {
    var type = mimeType[ext];
    if (!type)
        return "text/plain";
    else
        return type;
```

```
}

app.listen(80);

var response = function(res, type, data) {
    res.writeHead(200, {'Content-Type': type});
    if (type.indexOf("text/")>=0)
        res.end(data);
    else
        res.end(data, "binary");
}

app.get('*', function(req, res){
  try {
    var path = '.' + req.url;
    fs.stat(path, function(err, pathStat) {
        if (err) {
            response(res, "text/plain", err.toString());
            return;
        }
        if (pathStat.isFile()) {
```

```
        fs.readFile(path, function(err, file) {
            var tokens = path.split(".");
            var ext = tokens[tokens.length-1];
            response(res, getMimeType(ext), file);
        });
    } else if (pathStat.isDirectory()) {
        var dirPath = req.url;
        if (dirPath.substring(-1)!=="/")
            dirPath = dirPath+"/";
        var html = "<html><body><h1>"+req.url+"</h1>\n";
        fs.readdir(path, function(err, files) {
            for (f in files) {
                fname = files[f];
                filePath = dirPath+fname;
                html += "<li><a href='"+filePath+"'>"+fname+"</a></li>\n";
            }
            html += "<body></html>";
            response(res, "text/html", html);
        });
    }
});
```

```
  } catch (err) {
    response(res, "text/plain", err.toString());
  }
});

console.log('start WebServer\n');
```

参考文献

- 深入浅出Node.js相关的内容 -- http://www.infoq.com/cn/master-nodejs
  - 深入浅出Node.js（一）：什么是Node.js -- http://www.infoq.com/cn/articles/what-is-nodejs
  - 深入浅出Node.js（三）：深入Node.js的模块机制 -- http://www.infoq.com/cn/articles/nodejs-module-mechanism

Node.js -- 表單資料的處理

Web 的表單資料

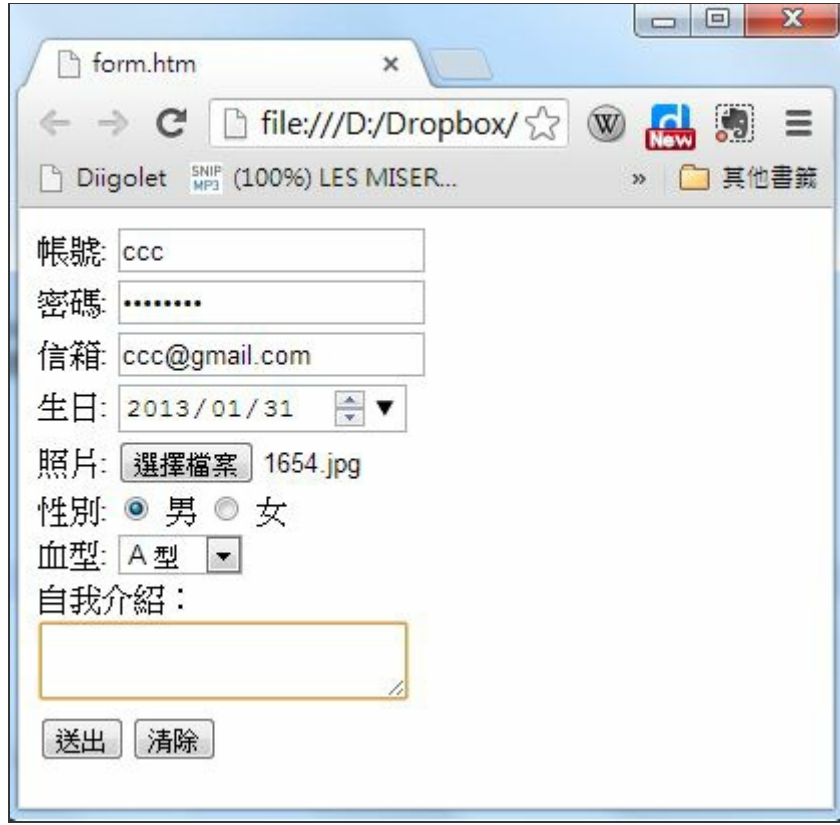
在「HTML 網頁設計」這個章節裏，我們曾經介紹過 form 這個標記的用法，以及包含於其中的一群表單標記，讓我們再次複習一下這個主題。

在 HTML 當中，表單 (Form) 是指可以讓使用者進行輸入的元件，其語法是用 <form> ...</form> 夾住一堆的輸入元件，這些輸入元件包含 input (輸入)、 textarea (文字區)、 select (選項) 等，其中的 input 還可以根據其 type 欄位顯示成 checkbox, color, date, datetime, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week 等各種不同的輸入欄形式，以下是一個表單的範例。

```
<form action="signup" method="post">
帳號: <input type="text" name="user"/><br/>
密碼: <input type="password" name="password"/><br/>
信箱: <input type="email" name="email"/><br/>
生日: <input type="date" name="birthday"/><br/>
照片: <input type="file" name="picture"/><br/>
性別: <input type="radio" name="sex" value="male" checked/> 男
```

```
    <input type="radio" name="sex" value="female"/> 女<br/>
血型: <select name="BloodType">
      <option value="A">A 型</option>
      <option value="B">B 型</option>
      <option value="AB">AB 型</option>
      <option value="O">O 型</option>
    </select> <br/>
自我介紹： <br/>
<textarea name="AboutMe">
</textarea> <br/>

<input type="submit" value="送出"/><input type="reset" value="清除"/><br/>
</form>
```

檢視檔案：https://dl.dropbox.com/u/101584453/wp/code/form.htm



表單的顯示結果

在上述的範例中，當 submit 類型的送出鈕被按下後，瀏覽器會將這些填寫的資訊，以先前所說的

GET/POST 方式，發送給伺服器，如果 method 欄位是 GET，那麼會採用在 HTTP 表頭網址處傳送 singup?user=xxx&password=xxx .....這樣的形式送出，這種方式會將密碼顯示在瀏覽器的網址列上，比較容易被看到，若 method 欄位是 POST，則會在 HTTP 表頭尾端加上 user=xxx&password=xxx ... 的資訊，不會在網址列上被看到。當然、如果有人監控網路上訊息的話，還是會看得到這些輸入資訊。

若要更安全，則必須採用 HTTPS 的 SSL 方式傳遞，這種方式會對訊息加密編碼，就比較不會有輸入訊息外洩的危險。

node.js 對於 GET/POST 訊息的處理，是以 url 與 querystring 為基礎，然後還有個額外的套件 express 可以進行方便的進階處理，以下我們將介紹這些主題。

GET 的處理

node.js 在處理 GET 表單訊息時，最基礎的方式是使用 url.parse() 與 querystring.parse() 等兩個函數為基礎的，以下是一個簡單的範例。

檔案：httpget.js

```
http = require('http');
qs = require('querystring');
url = require('url');
util = require('util');
```

```
format = function() {
    return util.format.apply(null, arguments);
};

log = console.log;
ip   = "127.0.0.1";
port = 8080;

server = http.createServer(function (req, res) {
  var path = url.parse(req.url),
      parameter = qs.parse(path.query);

  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.write(format("path=%j\n", path));
  res.write(format("parameter=%j\n", parameter));
  res.end();
});

server.listen(port, ip);

console.log(format("Server running at http://%s:%d\n", ip, port));
```

執行結果：

```
D:\Dropbox\Public\wp\code>node httpget
Server running at http://127.0.0.1:8080
```

由於 GET 的傳送方式是透過網址欄，因此您只要在網址上進行修改，就可以看到上述程式會有不同的執行結果。

POST 的處理

對於 POST 訊息而言，就無法透過修改網址的方式測試了，要示範如何用 node.js 處理 post 訊息，我們必須先有一個 包含表單的 HTML 網頁，然後在 form 欄位內指定 method="POST" ，這樣當您填完按下 submit 送出鈕時，網頁中所 填寫的資訊就會附加在 HTTP 表頭的尾部傳送出去。

檔案：httppost.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
</head>
  <body>
    <form id="signup" method="POST" action="http://localhost:8080/post">
      <label>帳號：</label><input type="text" id="user" name="user" value="ccc"/><br/>
      <label>密碼：</label><input type="text" id="password" name="password" value="1234567"/><br/>
      <label>標題：</label><input type="text" id="title" name="title" size="80" value="當您死後，您在意的是公
      <textarea name="text" rows=30 cols=80>
如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！
```

但是我會在意自己的「照片、錄影、網誌、facebook、YouTube上的影片、電子書、以及我所創辦的程式

我不希望死後 facebook 上的資訊就被視為靜止戶而刪除，我不希望網誌上的資訊雜亂無章，我不希望數

我希望這些資訊可以變成我的日記、著作，有人幫我剪接這些影片，在喪禮上可以回顧我的一生，我希

如果我的老婆小孩經濟困難，我希望可以從我的數位遺產中找到有價值出版的作品，幫助他們度過難關

要完成這些事，就必須要有一個組織，提供「數位遺產管理」的服務。這需要有一些工具可以擷取出這

這是一個全新的領域 -- 「數位殯葬業 + 數位遺產管理事業」，當您抱怨網路行業都已經被 Google 、Face

```
    </textarea><br/>
    <input type="submit" value="送出" />
  </form>
 </body>
</html>
```


httppost.htm
file:///D:/Dropbox/Public/wp/code/httppost.htm
每2位香港媽媽，就...
佳好數位小舖 特價1...
LEGO.com Educati...
XRAY
其他書籤
帳號：ccc
密碼：1234567
標題：當您死後，您在意的是公墓裏的那塊牌子，還是 facebook 上的日誌與照片呢？
如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！
但是我會在意自己的「照片、錄影、網誌、facebook、YouTube上的影片、電子書、以及我所創辦的程式人雜誌」是否有人接手、整理或出版。
我不希望死後 facebook 上的資訊就被視為靜止尸而刪除，我不希望網誌上的資訊雜亂無章，我不希望數位相機的照片與錄影從此消失，我不希望那些美好的記憶在家人的腦海裡逐漸模糊！
我希望這些資訊可以變成我的日記、著作，有人幫我剪接這些影片，在喪禮上可以回顧我的一生，我希望我的小孩有空時能透過這些資訊瞭解我，甚至是與已經身在天堂的我對話，當他有的生命遇到疑問時，可以從我的人生經驗當中得到啟發，或者些許的指引與安慰！
如果我的老婆小孩經濟困難，我希望可以從我的數位遺產中找到有價值出版的作品，幫助他們度過難關，如果他們的經濟情況不錯，我希望這些價值就捐獻給公益團體。
要完成這些事，就必須要有一個組織，提供「數位遺產管理」的服務。這需要有一些工具可以擷取出這些數位遺產，需要一些人力可以幫組織與整理，並且需要服務人員對家屬提供適當的協助。所以我想，這需要一整個公司，甚至整個供應鏈的配合。
這是一個全新的領域 -- 「數位殯葬業 + 數位遺產管理事業」，當您抱怨網路行業都已經被 Google 、Facebook 、淘寶等公司壟斷的時候，實體的世界當中仍然有很多的創業機會，等著您去開發！
送出

接著、我們就可以用請求參數 req 的 on("data", ...) 事件與接收資料，然後在 on("end", ...) 事件 時完成接收並處理 POST 訊息，如以下程式所示。

檔案：httppost.js

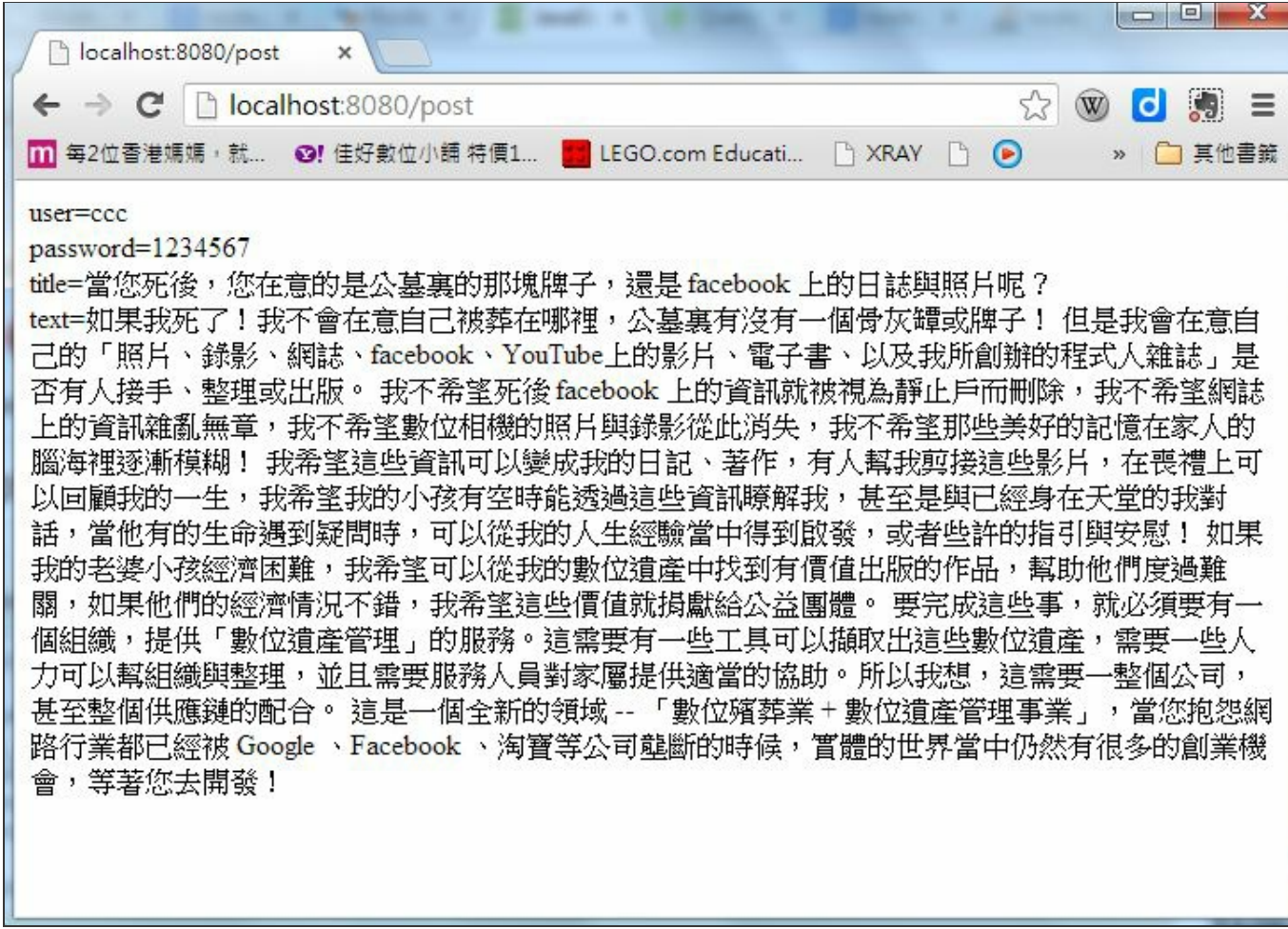
```
http = require('http');
qs = require('querystring');
url = require('url');
util = require('util');

format = function() {
   return util.format.apply(null, arguments);
};

log = console.log;
ip   = "127.0.0.1";
port = 8080;

server = http.createServer(function (req, res) {
  var path = url.parse(req.url, true),
```

```
    parameter = qs.parse(path.query);

res.writeHead(200, {'Content-Type': 'text/plain'});
formData = '';
req.on("data", function(data) {
  return formData += data;
});
req.on("end", function() {
  res.writeHead(200, {"Content-Type":"text/html; charset=utf-8"});
  post = qs.parse(formData);
  log(format("post=%j\n", post));
  user = post.user;
  password = post.password;
  title = decodeURIComponent(post.title);
  text  = decodeURIComponent(post.text);
  res.write("user="+user+"<br/>");
  res.write("password="+password+"<br/>");
  res.write("title="+title+"<br/>");
  res.write("text="+text+"<br/>");
  res.end();
});
```

```
});

server.listen(port, ip);

console.log(format("Server (POST test) running at http://%s:%d\n", ip, port));
```

執行結果：

```
user=ccc
password=1234567
title=當您死後，您在意的是公墓裏的那塊牌子，還是 facebook 上的日誌與照片呢？
text=如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！ 但是我會在意
...
...
等著您去開發！
```


localhost:8080/post
localhost:8080/post
每2位香港媽媽，就...
佳好數位小鋪 特價1...
LEGO.com Educati...
XRAY
其他書籤
user=ccc
password=1234567
title=當您死後，您在意的是公墓裏的那塊牌子，還是 facebook 上的日誌與照片呢？
text=如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！ 但是我會在意自己的「照片、錄影、網誌、facebook、YouTube上的影片、電子書、以及我所創辦的程式人雜誌」是否有人接手、整理或出版。 我不希望死後 facebook 上的資訊就被視為靜止戶而刪除，我不希望網誌上的資訊雜亂無章，我不希望數位相機的照片與錄影從此消失，我不希望那些美好的記憶在家人的腦海裡逐漸模糊！ 我希望這些資訊可以變成我的日記、著作，有人幫我剪接這些影片，在喪禮上可以回顧我的一生，我希望我的小孩有空時能透過這些資訊瞭解我，甚至是與已經身在天堂的我對話，當他有的生命遇到疑問時，可以從我的人生經驗當中得到啟發，或者些許的指引與安慰！ 如果我的老婆小孩經濟困難，我希望可以從我的數位遺產中找到有價值出版的作品，幫助他們度過難關，如果他們的經濟情況不錯，我希望這些價值就捐獻給公益團體。 要完成這些事，就必須要有一個組織，提供「數位遺產管理」的服務。這需要有一些工具可以擷取出這些數位遺產，需要一些人力可以幫組織與整理，並且需要服務人員對家屬提供適當的協助。所以我想，這需要一整個公司，甚至整個供應鏈的配合。 這是一個全新的領域 -- 「數位殯葬業 + 數位遺產管理事業」，當您抱怨網路行業都已經被 Google 、Facebook 、淘寶等公司壟斷的時候，實體的世界當中仍然有很多的創業機會，等著您去開發！

網路記事本

接著、讓我們用一個完整的「網路記事本」範例，來說明 node.js 的表單處理過程，在這個範例中， 我們用了 express 這個套件，以便能更方便的處理網址所對應的不同功能，在執行程式前，請先用 下列指令安裝 express 套件。

```
$ npm install express -g
```

整個網路記事本的專案，您可以從下列網址下載：

網路記事本 (下載點)：noteserver1.zip

以下是該程式的執行過程：

```
D:\Dropbox\Public\wp\code\noteserver1>node NoteServer.js
start NoteServer



note ifidie.txt
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="utf-8" />
</head>
....
```

以下是該程式的一些執行畫面，當您在網址上輸入以下內容時，您將可以編輯 ifidie.txt 這個檔案。

```
http://127.0.0.1/note/ifidie.txt
```


127.0.0.1/note/ifidie.txt
127.0.0.1/note/ifidie.txt
每2位香港媽媽，就...
【流雲字幕組】進擊...
星际争霸战_在线观...
迪士尼動畫 天外奇...
其他書籤
檔案路徑：ifidie.txt
如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！
但是我會在意自己的「照片、錄影、網誌、facebook、YouTube上的影片、電子書、以及我所創辦的程式人雜誌」是否有人接手、整理或出版。
我不希望死後 facebook 上的資訊就被視為靜止戶而刪除，我不希望網誌上的資訊雜亂無章，我不希望數位相機的照片與錄影從此消失，我不希望那些美好的記憶在家人的腦海裡逐漸模糊！
我希望這些資訊可以變成我的日記、著作，有人幫我剪接這些影片，在喪禮上可以回顧我的一生，我希望我的小孩有空時能透過這些資訊瞭解我，甚至是與已經身在天堂的我對話，當他有的生命遇到疑問時，可以從我的人生經驗當中得到啟發，或者些許的指引與安慰！
如果我的老婆小孩經濟困難，我希望可以從我的數位遺產中找到有價值出版的作品，幫助他們度過難關，如果他們的經濟情況不錯，我希望這些價值就捐獻給公益團體。
要完成這些事，就必須要有一個組織，提供「數位遺產管理」的服務。這需要有一些工具可以擷取出這些數位遺產，需要一些人力可以幫組織與整理，並且需要服務人員對家屬提供適當的協助。所以我想，這需要一整個公司，甚至整個供應鏈的配合。
這是一個全新的領域 -- 「數位殯葬業 + 數位遺產管理事業」，當您抱怨網路行業都已經被 Google 、Facebook 、淘寶等公司壟斷的時候，實體的世界當中仍然有很多的創業機會，等著您去開發！
xxx
送出

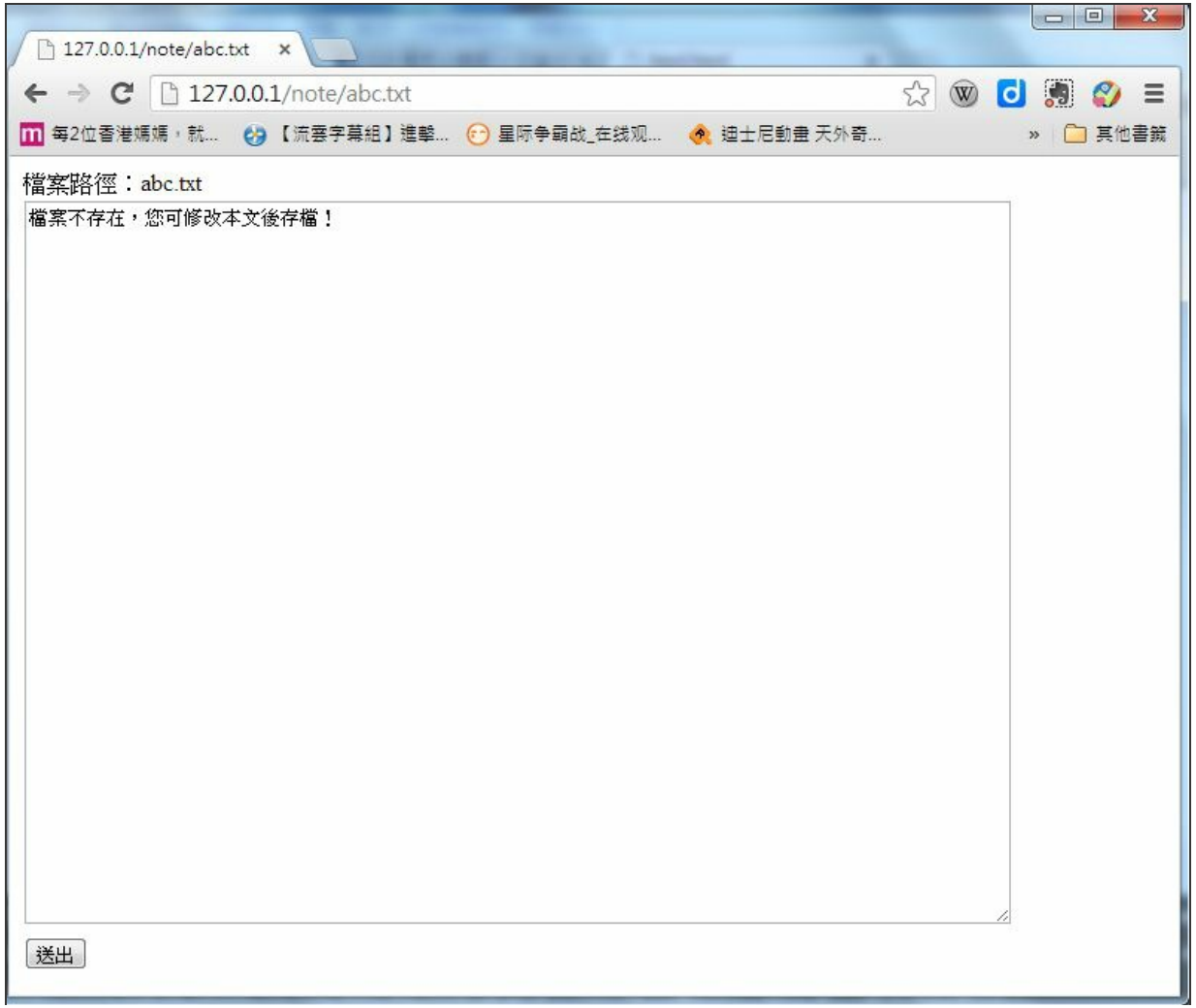
編輯已存在的檔案 ifidie.txt

編輯完後送出，您可以看到下列儲存訊息，此時如果您在點選 ifidie.txt 這個超連結，就能回到編輯畫面，繼續編輯其內容。


127.0.0.1/note/ifidie.txt
127.0.0.1/note/ifidie.txt
每2位香港媽媽，就...
【流雲字幕組】進擊...
星际争霸战_在线观...
迪士尼動畫 天外奇...
其他書籤
檔案路徑：ifidie.txt
如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！
但是我會在意自己的「照片、錄影、網誌、facebook、YouTube上的影片、電子書、以及我所創辦的程式人雜誌」是否有人接手、整理或出版。
我不希望死後 facebook 上的資訊就被視為靜止戶而刪除，我不希望網誌上的資訊雜亂無章，我不希望數位相機的照片與錄影從此消失，我不希望那些美好的記憶在家人的腦海裡逐漸模糊！
我希望這些資訊可以變成我的日記、著作，有人幫我剪接這些影片，在喪禮上可以回顧我的一生，我希望我的小孩有空時能透過這些資訊瞭解我，甚至是與已經身在天堂的我對話，當他有的生命遇到疑問時，可以從我的人生經驗當中得到啟發，或者些許的指引與安慰！
如果我的老婆小孩經濟困難，我希望可以從我的數位遺產中找到有價值出版的作品，幫助他們度過難關，如果他們的經濟情況不錯，我希望這些價值就捐獻給公益團體。
要完成這些事，就必須要有一個組織，提供「數位遺產管理」的服務。這需要有一些工具可以擷取出這些數位遺產，需要一些人力可以幫組織與整理，並且需要服務人員對家屬提供適當的協助。所以我想，這需要一整個公司，甚至整個供應鏈的配合。
這是一個全新的領域 -- 「數位殯葬業 + 數位遺產管理事業」，當您抱怨網路行業都已經被 Google 、Facebook 、淘寶等公司壟斷的時候，實體的世界當中仍然有很多的創業機會，等著您去開發！
xxx
送出

編輯完後的儲存訊息 ifidie.txt

假如您輸入一個原本不存在的檔案名稱時，那麼內容就會顯示「檔案不存在，您可修改本文後存檔！」，然後您同樣可以編輯後 按下「送出」鍵儲存，此時上述程式將會建立一個新的 abc.txt 檔案，其內容即為您所輸入的內容。


127.0.0.1/note/abc.txt
127.0.0.1/note/abc.txt
每2位香港媽媽，就...
【流雲字幕組】進擊...
星际争霸战_在线观...
迪士尼動畫 天外奇...
其他書籤
檔案路徑：abc.txt
檔案不存在，您可修改本文後存檔！
送出

編輯不存在的檔案 abc.txt

上述網路記事本的完整程式碼如下所示：

檔案：NoteServer.js

```
var path = require('path');
var fs = require("fs");
var qs = require('querystring');
var express = require("express");
var app = express();
app.listen(80);

var noteTemplate = "";

fs.readFile("note.htm", "utf8", function(err, file) {
    noteTemplate = file;
});

fs.readFile("save.htm", "utf8", function(err, file) {
```

```
    saveTemplate = file;
});

var error = function(err, res) {
    if (err) {
        res.writeHead(404, {'Content-Type': 'text/plain'});
        res.end();
    }
}

var response = function(res, type, text) {
    res.writeHead(200, {'Content-Type': type});
    res.write(text);
    console.log(text);
    res.end();
}

app.post('/save/:path', function(req, res){
    console.log('save ' + req.params.path);
    var path = "./"+req.params.path;
    formData = '';
```

```
    req.on("data", function (chunk) {
        formData += chunk;
    });

    req.on("end", function () {
        form = qs.parse(formData);
        fs.writeFile(path, form.note, function (err) {
          if (!err)
            response(res, "text/html", saveTemplate.replace("[[?path?]]", req.params.path)
            .replace("[[?path?]]", req.params.path)
            .replace("[[?path?]]", req.params.path).replace("[[?fileText?]]", form.note));
        });
    });
});

app.get('/note/:path', function(req, res){
    console.log('note ' + req.params.path);
    var path = "./"+req.params.path;
    fs.readFile(path, "utf8", function(err, file) {
        if (err)
            response(res, "text/html", noteTemplate.replace("[[?path?]]", req.params.path)
```

```
            .replace("[[?path?]]", req.params.path).replace("[[?fileText?]]", "檔案不存在，您可修改本文後存檔！")
        else
            response(res, "text/html", noteTemplate.replace("[[?path?]]", req.params.path)
            .replace("[[?path?]]", req.params.path).replace("[[?fileText?]]", file));
    });
});

console.log('start NoteServer\n');
```

檔案：note.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
</head>
  <body>
    <form method="POST" action="/save/[[?path?]]">
      <label>檔案路徑：[[?path?]]</label><br>
      <textarea name="note" rows=30 cols=80>
[[?fileText?]]
```

```
    </textarea><br/>
    <input type="submit" value="送出"/>
  </form>
 </body>
</html>
```

檔案：save.htm

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
</head>
<body>
save:path=<a href="/note/[[?path?]]">[[?path?]]</a>
<pre>
[[?fileText?]]
</pre>
</body>
</html>
```

檔案：ifidie.txt

如果我死了！我不會在意自己被葬在哪裡，公墓裏有沒有一個骨灰罈或牌子！

但是我會在意自己的「照片、錄影、網誌、facebook、YouTube上的影片、電子書、以及我所創辦的程式

我不希望死後 facebook 上的資訊就被視為靜止戶而刪除，我不希望網誌上的資訊雜亂無章，我不希望數

我希望這些資訊可以變成我的日記、著作，有人幫我剪接這些影片，在喪禮上可以回顧我的一生，我希

如果我的老婆小孩經濟困難，我希望可以從我的數位遺產中找到有價值出版的作品，幫助他們度過難關

要完成這些事，就必須要有一個組織，提供「數位遺產管理」的服務。這需要有一些工具可以擷取出這

這是一個全新的領域 --「數位殯葬業 + 數位遺產管理事業」，當您抱怨網路行業都已經被 Google 、Face

結語

Web 可以說是 1990 年以來最重要的程式技術，但由於技術的演進呈現多元而紛亂的過程，因此總是讓初學者眼花撩亂。

您可以看到像 CGI, PHP, ASP, ASP.NET, JSP, Ruby on Rail, Python Dejango, Node.js 等技術，都可以用來 實作 Web 的伺服端程式，還好客戶端大致都統一在 HTML+CSS+JavaScript 的架構之下，雖然中間也有 Flash 技術的風行， 但 Flash 最近已經有逐漸被 HTML 5.0 取代的跡象。

在本書中，我們採用了一個相對較簡單的開發架構，也就是 HTML+CSS+JavaScript+Node.js 這樣的架構。在這種架構下， 只需要學會一種程式語言，也就是 JavaScript，希望這樣可以讓網站設計的學習者用最小的力氣，得到最大的效用。

但本書已經盡可能的涵蓋了筆者認為重要的部份，未來、如果有任何新的主題，筆者隨時會在補上，希望這樣的一本書 能對讀者有所幫助。

附錄

目錄（含教學影片）

主題	教學影片
第 1 章. Web 程式簡介	
第 2 章. HTML 網頁設計	
─ HTML 基礎 1	http://youtu.be/e0dQYn0vzeQ
─ HTML 基礎 2	http://youtu.be/DTRTWv0c-mY
第 3 章. CSS 版面設計	
─ CSS 樣式語言 1	http://youtu.be/FcFk4spEBlU
─ CSS 樣式語言 2	http://youtu.be/BBq2qcWe3Fs
第 4 章. JavaScript 程式設計	

第 5 章. JavaScript 的字串處理	
第 6 章. DHTML 動態網頁	
第 7 章. Node.js 命令列 - 讓 JavaScript 作為一般程式執行	
─ node.js 簡介	http://youtu.be/HfvgT6ow12E
─ node.js 基礎	http://youtu.be/l5S-855PKMk
- node.js 的字串	http://youtu.be/dtT3JgL9-7k
- node.js 正規表達式 1	http://youtu.be/hvF2iy3Ew4Y
- node.js 正規表達式 2	http://youtu.be/QPf0Gn57b3w
─ node.js 命令列 1	http://youtu.be/MF3faHnnWiA
─ node.js 命令列 2	http://youtu.be/G2V29mO5wo0
─ node.js 物件導向	http://youtu.be/he6-GzX8z54

─ node.js 檔案讀寫	http://youtu.be/1WxnW62XAqI
─ node.js 物件模組定義	http://youtu.be/XedEvcEZCXE
─ node.js 靜態模組定義	http://youtu.be/HfvgT6ow12E
─ node.js 匿名函數 Callback 1	http://youtu.be/0jC2TWIrMnU
─ node.js 匿名函數 Callback 2	http://youtu.be/Xg-3LKqrkN8
第 8 章. Node.js Web 程式	
─ node.js Hello Server	http://youtu.be/iohBu4gS770
─ node.js 簡易的 HtmlServer	http://youtu.be/NWcREc8l2mQ
─ node.js 簡易的 WebServer	http://youtu.be/K_tvilxm2I8
第 9 章. Node.js 表單	
─ node.js GET 參數取得與解析	http://youtu.be/z55rIcnhDis
─ node.js POST 參數的取得與處理	http://youtu.be/VWOQapnQfb4

ㄧ node.js 網路記事本 1	http://youtu.be/uOqW7TNMeFw
ㄧ node.js 網路記事本 2	http://youtu.be/8lFAcZLsQ9w
ㄧ node.js 網路記事本 3	http://youtu.be/6rduY-GDHB4

考題與解答

學校	科系	學期	課程	名稱	考題	解答
金門大學	資工系	101學年下學期	網路程式設計	期中考	101bWebExam1.pdf	101bWebExam1Answer.pdf
金門大學	資工系	101學年下學期	網路程式設計	期末考	101bWebExam2.pdf	101bWebExam2Answer.pdf