
目錄

序	1.1
JavaScript字串常用屬性與函式	1.2
Regular Expression	1.3
日期資料	1.4

JavaScript Coding

這本書預計寫作有關於 JavaScript 程式設計的大小事。有空就寫一點，正陸陸續續完成，惟時間並不確定。

任何問題與建議，請寄 stdnt25@gmail.com，謝謝!

JavaScript 字串常用屬性與函式

length 字串字數

- *length* 屬性傳回字串字數
- 雖然每個中文字實際佔用三個位元組的記憶體，但 *length* 傳回單位為字數，也就是說，"中文字".*length* 長度為三。

```
var data = "中文字";  
var dataLength = data.length; // 傳回 3
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_length

indexOf() 搜尋特定文字

- 使用格式: "稻草堆".*indexOf*("針" [, 從哪一個位置找起])
- 傳回「針」的起始位置。傳回零，表示在開始第一個位置就找到了
- 傳回 -1 表示沒找到
- 省略第二個參數時，預設為零，也就是從最左邊開始找起

```
var data = "A secret makes a woman, woman."; // Vermouth (Detective Conan)  
var position = data.indexOf("woman"); // 傳回 17  
var nextPosition = data.indexOf("woman", position + 1); // 傳回 24
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_indexof

substr() 擷取字串的部分內容

- "字串內容".*substr*(開始位置 [, 長度])
- 省略「長度」時，將一直取至字串結尾

```
var str = "0123456789";  
var result = str.substr(2, 5); // 23456
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substr

Exercise

請想想看，如何以 `substr()` 取一個字串最右邊三個字？

`slice()` 擷取字串的部分內容

- "字串內容".`slice` (開始位置 [, 結束位置])
- 「開始位置」與「結束位置」大於等於零時，表示從開始位置取至結束位置之前(不含結束位置的那個字元)
- 省略「結束位置」時，將一直取至字串結尾

```
var str = "0123456789876543210";  
var result = str.slice(2, 5); // 234
```

- 「開始位置」與「結束位置」為負數時，表示從 `length + 「開始位置」`，取至 `length + 「結束位置」`
- 不妨想像成: 從右邊算起，只是，位置編號變成從一開始算起（JavaScript 習慣從零開始數起）。

```
var str = "0123456789876543210";  
//           012345678  
// 19 - 6 = 13  
// 19 - 1 = 18  
// 從 13 取至 18 (不含18),  
// 也可以想成: 右邊數來第六個，取到右邊第一個(不含)之前  
var result = str.slice(-6, -1); // 54321
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice

Exercise

請Google一下 `substring()` 的使用說明，該函式與 `slice()`、`substr()` 有何異同？

toUpperCase() ***toLowerCase()*** 轉大寫／小寫

- "string".*toUpperCase()* 轉大寫
- "string".*toLowerCase()* 轉小寫

```
// code
var data = "Hello!";
alert( data.toUpperCase() ); // HELLO!
alert( data.toLowerCase() ); // hello!
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_tolower

charAt() 傳回字串於指定位置的字元

- 相當於"string"[編號], 但是 *charAt* (編號) 比較安全
- 編號照例從零算起
- "string"[編號] 的編號超過範圍時, 會得到 *undefined*
- *charAt* (編號) 的編號超過範圍時, 傳回空字串

```
var str = "錢達智";

var result = str.charAt(2); // 智
var result = str[2];

var result = str.charAt(3); // "" 空字串
alert(result.length); // 0
var result = str[3]; // undefined
alert(result.length); // 程式發生錯誤
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_charat

charCodeAt() 查出字串指定位置的字元內碼

- *charCodeAt* (編號) 傳回字串指定位置的字元內碼
- 編號照例從零算起
- 指定的位置沒有字元時, 傳回 *NaN*

```
var str = "錢達智";
var result = str.charCodeAt(0).toString(16)
    + "," + str.charCodeAt(1).toString(16)
    + "," + str.charCodeAt(2).toString(16)
// 9322,9054,667a
```

Exercise

請用 `charCodeAt()` 查出你的中文名字的 Unicode 編號

`split()` 將字串拆解成陣列

- 格式: "字串".`split` ("分界字元")
- 用分界字元將字串斷成一個字串陣列

```
var str = "錢達智,B123456789,M";
var dataArray = str.split(",");
// ["錢達智", "B123456789", "M"]
console.log(JSON.stringify(dataArray));
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_split

Exercise

請想想看，如何取出 `c:\doc\faq.txt` 的檔名？

`search()` 以 Regular Expression 語法搜尋文字

- 與 `indexOf()` 相比，更有彈性（但也比較慢一些）。
- 下列程式的第一行，結尾處以 `i` 告知 JavaScript 引擎進行 `case-insensitive` 不區分大小寫文字比對。
- 找不到時，傳回 `-1`

```
var format = /ci-15\d/i;
var data = "Flight numbers: CI-123, CI-151, CI-156.";
var result = data.search(format);
console.log(result); // 24
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_search_regexp

Note

關於 Regular Expression，請參考 Regular Expression 這章的說明。

replace() 字串搜尋替換

- "在來源字串".**replace**(找什麼, "換成什麼")
- 第一個參數 "找什麼" 若為字串，會以區分大寫小寫的方式，換掉找到的第一個項目
- 第一個參數「找什麼」若為 Regular Expression，則依格式處理，例如下列區塊的最後一行程式
 - **i** : case-insensitive，不區分大寫小寫
 - **g** : global match，不只換掉一個，全部更換

```
var source = "abcABCabc";  
var result = source.replace("ABC", "$"); // abc$abc  
var result = source.replace(/abc/, "$"); // $ABCabc  
var result = source.replace(/abc/ig, "$"); // $$$
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_replace

Exercise

如何將 "0123456789" 的 8 移到字串最左邊呢?



Regular Expression

Regular Expression（正則表達式、正規表示法、正規運算式、規則運算式、常規表示法）是一套語法，這套語法在文字內容的比對與擷取等方面，功能十分強大。本章將說明Regular Expression的常用語法，並且應用於表單（Form）內容驗證及JavaScript程式。

基本入門：逐個字元定義格式

該怎麼使用Regular Expression呢？

舉例來說，華航的航班編號是 CI-XXX，其中XXX是數字，例如：CI-123。我們希望操作人員在輸入下列的表單資料時，能按照這個格式輸入。

```
<form method="post" action="echo.php">
  China Airline flight number:
  <input type="text" name="flightNumber" pattern=""> <br>
  <input type="submit" name="btnOK" id="btnOK" value="OK">
</form>
```

pattern 的屬性值可以這樣子處理：

1. 「CI-」是固定的，直接註記即可
2. 第一個 X，在一對中括號裏頭註記0-9，像這樣：CI-[0-9]
3. 第二個與第三個 X，也採用相同的作法，完成後的格式：CI-[0-9][0-9][0-9]，如下：

```
<form method="post" action="echo.php">
  China Airline flight number:
  <input type="text" name="flightNumber" pattern="CI-[0-9][0-9][0-9]"> <br>
  <input type="submit" name="btnOK" id="btnOK" value="OK">
</form>
```

Note

用一對中括號指定一個字元，在中括號裏頭註明可接受哪些字元，例如：[AX]O，就表示只有AO或XO符合格式，aO不行（a要大寫），CO也不可以（第一個字元只接受A或X）。

由於 [0-9] 數字很常用，於是Regular Expression特別設計一個簡寫的代替語法：\d，上述的華航的航班編號格式，可以改寫成

CI-\d\d\d。

其他常用的字元語法整理如下表：

格式	說明	相當於...
\d	數字	[0-9]
\w	英文字母、數字、底線	[0-9a-zA-Z_]
\s	空白字元	[\t\r\n\f]
\D	不要數字	[^0-9]
\W	不要這些：英文字母、數字、底線	[^0-9a-zA-Z_]

數量語法

接下來，我們來試試看**身份證字號**的格式該怎麼訂：

1. 第一個字元是大寫字母：**[A-Z]**
2. 第二碼不是一就是二，格式就變成：**[A-Z][12]**
3. 接下來有八個數字，寫成 **[A-Z][12][0-9][0-9][0-9][0-9][0-9][0-9][0-9]**，感覺不僅有點囉嗦，而且很容易數錯。
4. 改成這樣會好一點（恐怕還是會數錯）：**[A-Z][12]\d\d\d\d\d\d\d\d**
5. 這樣會不會更好：**[A-Z][12]\d{8}**

關於**{數字}**的語法說明：

- **X{N}**，N是數字，表示X的數量不多不少正好要是N個
- **{N, M}**，表示數量介於N到M之間(包括 N, M)，例如 **\d{3, 5}** 表示三到五個數字。
- **{N, }** 表示 N 個(含)以上
- **{, M}** 表示 M 個(含)以下

其他有關於數量的語法：

格式	說明	相當於...
?	零或一個	{0,1}
*	零個以上	{0, }
+	至少一個	{1, }

群組

學會「群組」，你的Regular Expression功力就更上一層。什麼是「群組」？簡單地說：用括號包起來的就算一組。

還是用例子來學比較有方向感。舉例來說，如何指定 e-mail 的格式呢？

1. e-mail 最好認的特徵就是 @，所以，先暫定成 `\w+@\w+`。也就是 @ 符號前後一定都要有字。
2. 有時候，在 @ 左邊可能會出現例如 `wolfgnag.ta-chih.chien@gmail.com` 含有句號、連字號的情形，但 `\w+` 並不接受句號、連字號，怎麼辦呢？
3. 這樣子如何: `\w+([.-]\w+)*@\w+([.-]\w+)+`

（驚！）別急一段一段慢慢來：

- `[.-]\w+` 用括號包起來，所以它們構成一組。這組的格式是說，句號或連字號的後頭一定要有字（像 `wolfgang-@gmail.com` 就不許，連字號後面一定要有字）。
- `([.-]\w+)` 這組又以 * 修飾，說明了這組可能完全不出現，也可能重複好幾次（每次重複都一定不能有句號、連字號結尾）。
- `\w+([.-]\w+)*@\w+([.-]\w+)+` 在 @ 右邊的情也類似，但是改用 + 符號設定至少要出現一次。

其他重要符號

符號	說明
	鍵盤 shift + \ 打出來的那個濾管符號，作用為：「或者」。例如郵遞區號有三碼也有五碼的數字，格式為： <code>\d{5} \d{3}</code>
\	倒斜線 = 「轉意符」、「逸脫字元」。例如，左括號在正則運算式已用來指定群組，但我們現在希望特定位置就是括號時，可利用 \ 來聲明那裏就是括號，不是群組。

JavaScript 與 Regular Expression

JavaScript 使用下列格式指定 Regular Expression:

```
/pattern/modifiers
```

例如：

```
var format = /ci-15\d/i;
var data = "Flight numbers: CI-123, CI-151, CI-156.";
var result = data.search(format);
console.log(result); // 24
```

上述程式的第一行，結尾處以 **i** 告知 JavaScript 引擎進行 case-insensitive 不區分大小寫文字比對。

test() 測試文字內容是否符合格式

- `/RegularExpression/.test("待驗內容")` 檢查「待驗內容」是否符合 Regular Expression 格式。
- 符合格式的話，`test()` 傳回 `true`，否則傳回 `false`

```
var format = /ci-15\d/i;  
var data = "Flight numbers: CI-123, CI-151, CI-156.";   
var result = format.test(data);  
console.log(result); // true
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_test

exec() 以 Regular Expression 擷取文字

- `/RegularExpression/.exec("文字內容")` 依 Regular Expression 格式擷取「文字內容」。
- 如果找到符合格式的文字，傳回找到的內容，否則傳回 `null`

```
var format = /ci-15\d/i;  
var data = "Flight numbers: CI-123, CI-151, CI-156.";   
var result = format.exec(data);  
console.log(result); // CI-151
```

如果要繼續找「下一個」，請留意下列兩個重點:

- `/pattern/modifiers`，modifiers 要加上 **g**，例如: `/ci-15\d/ig`
- 連續呼叫 `exec()` 方法。第二回合如果有找到的話就是第二個，第三回合找到的就是第三個，以下類推。例如:

```
var format = /CI-15\d/g;  
var data = "Flight numbers: CI-123, CI-151, CI-156.";   
while ( ( result = format.exec(data) ) !== null ) {  
    alert(result);  
}
```

Exercise

w3schools 線上練習網址:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_exec

JavaScript 日期資料

JavaScript 以 *Date* 物件處理日期方面的資料（年、月、日、時、分、秒、毫秒）。我們也可將日期視為是一種以格林威治標準時間 1970年元月一日零時零分零秒為基準點的毫秒差，例如 3000，即是1970年元月一日零時零分三秒。

建立 *Date* 物件時，有以下四種寫法（本文稍後會詳述）：

- `new Date()`，例如 `var dateData = Date();`
- `new Date(毫秒數)`，例如 `var dateData = Date(1489190400000);`
- `new Date("日期文字")`，例如 `var dateData = Date("2017-03-11T13:45:00Z");`
- `new Date(年月日等七個數字)`，例如 `var dateData = Date(2017, 2, 11, 13, 45, 0, 0);`

當下這一刻

下列程式建立 *Date* 物件時，沒有傳入任何參數，預設為「目前時間」。

所謂「目前時間」，詳細地說，是以瀏覽器的本地日期時間，配合瀏覽器所在作業系統設定的時區，換算成國際標準時間（以毫秒為單位的時間戳記），建立 *Date* 物件。

不過，*Date* 物件的資料在輸出成字串時，預設的作法會依照瀏覽器所在的時區，換算成本地時間。如果需要輸出國際標準時間的字串，應該改呼叫 `toUTCString()` 方法。程式示範如下：

```
var dateNow = new Date();
console.log(dateNow.getTime()); // 時間戳記(單位: 毫秒)
console.log(dateNow.toString()); // 輸出本地時間
console.log(dateNow.toUTCString()); // 輸出國際標準時間
```

將文字轉成日期

依照 ISO 8601 標準，日期格式為 `yyyy-mm-dd`，依此格式傳入日期文字可建立 *Date* 物件：

```
var dateData = new Date("2017-03-11");
console.log(dateData.toString());
// => Sat Mar 11 2017 08:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Sat, 11 Mar 2017 00:00:00 GMT
```

上述程式既沒有指定時間也沒有指定時區，預設為 00:00:00、格林威治國際標準時間。等同於下列寫法：

```
var dateData = new Date("2017-03-11T00:00:00Z");
console.log(dateData.toString());
// => Sat Mar 11 2017 08:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Sat, 11 Mar 2017 00:00:00 GMT
```

對於上述的輸出結果感到困惑嗎？請想想看，國際標準時3月11日零時，相當於台北上午八點鐘。

如果要指定時區，以台灣為例，請用下列任一格式：

- 2017-03-11T00:00:00+08:00
- 2017-03-11 00:00:00 (Taipei)
- 2017-03-11 00:00:00 (CST)

以下列程式為例，建立 **Date** 物件時，有特別註記台灣時間。請特別留意最後一行程式，台灣 3/11 00:00，倫敦時間為前一日的下午四點。

```
var dateData = new Date("2017-03-11T00:00:00+08:00");
console.log(dateData.toString());
// => Sat Mar 11 2017 00:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Fri, 10 Mar 2017 16:00:00 GMT
```

以七個日期數字建立 **Date** 物件

建立 **Date** 物件時，也可依序傳入年、月、日、時、分、秒、毫秒，建立特定日期時間的 **Date** 物件。但請留意第二個參數「月份」，月份在 JavaScript 是從零起跳的，也就是說，一月為0，二月為1，三月為2。

```
var dateData = new Date(2017, 2, 11, 0, 0, 0, 0);
console.log(dateData.toString());
// => Sat Mar 11 2017 00:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Sat Mar 11 2017 00:00:00 GMT+0800 (台北標準時間)
```

JavaScript 究竟如何儲存日期時間

時間戳記是個數字。JavaScript 的時間戳記是相對於 1970年元月一日零時零分零秒為基準點的毫秒差，例如 3000，即是1970年元月一日零時零分三秒。

不論我們指定哪一時區的時間資料，JavaScript 都會換算成國際標準時間，然後記下該時間的時間戳記。

以下列的程式來說，`getTime()` 抓出來的時間戳記值為零，因為筆者人在台灣，所以 `toString()` 顯示的是上午八點鐘。

```
var dateData = new Date("1970-01-01");
console.log(dateData.toString());
// => Thu Jan 01 1970 08:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Thu, 01 Jan 1970 00:00:00 GMT
console.log(dateData.getTime());
// => 0
```

請再看下列這一則例子：這回我們在建立 *Date* 物件時，有指定時區。雖然傳入的是上午八點鐘的資料，但是後來抓到的時間戳記值仍然是零（而非 $1000 * 60 * 60 * 8$ ），很明顯地，JavaScript *Date* 物件記住的是國際標準時間。

```
var dateData = new Date("1970-01-01T08:00:00+08:00");
console.log(dateData.toString());
// => Thu Jan 01 1970 08:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Thu, 01 Jan 1970 00:00:00 GMT
console.log(dateData.getTime());
// => 0
```

最後一則例子：

```
var dateData = new Date(0);
console.log(dateData.toString());
// => Thu Jan 01 1970 08:00:00 GMT+0800 (台北標準時間)
console.log(dateData.toUTCString());
// => Thu, 01 Jan 1970 00:00:00 GMT
console.log(dateData.getTime());
// => 0
```

上述程式，傳入 *Date* 建構程式的值為 0。這個零值的日期資料，在 `toString()` 時，傳回的是上午八點鐘，`getTime()` 的值為零。

存取日期的各項屬性

建立 *Date* 物件之後，可透過 *Date* 物件的 `getXXX()` 系列方法讀取年、月、日、時、分、秒、星期幾等資料（如下列程式），也可以透過 `setXXX()` 系列方法設定各個分項屬性。

使用這些方法時，請留意:

- 月份的數字從零起算，一月為0，二月為1。
- `getDay()` 傳回的是星期，星期天為0, 星期六為6
- `getDate()` 傳回的才是日期

```
var dateData = new Date("2017-03-11T13:45:00+08:00");
console.log(dateData.toString());
console.log(dateData.getMonth()); // 月份，三月為2
console.log(dateData.getDate()); // 日期，從一起算，3/1就是一，3/2傳回2
console.log(dateData.getDay()); // 星期，星期天為0
```

Exercise

請連往 [w3schools.com](https://www.w3schools.com/js/js_date_methods.asp) 關於日期各項方法的說明頁並試用其方法，

網址: https://www.w3schools.com/js/js_date_methods.asp

Exercise

如何將日期輸出成 yyyy-mm-dd 格式? 也就是將 2017/3/11 輸出成 2017-03-11