附錄 C
JavaScript 語法手冊

chapter 1

瞭解 JavaScript 的語法

著作權屬旗標公司，侵害必究．旗標2003

一個 JavaScript 程式長得像什麼樣子呢? 下面是一個簡單的例子:

```
<script language="JavaScript">
<!--
document.write(" 嗨! 本行文字由 JavaScript 所建立.")
//-->
</script>
```

注意, Javascript 會分辨英文字元的大小寫 (Case-sensitive), 也就是說, 同樣字元的大寫和小寫是不一樣的, 所以請讀者務必按照上例中的大小寫格式來鍵入程式, 以免造成錯誤。

JavaScript 語言的基本架構

在一篇文章中, 通常會有章、節、段落、註解、參考文獻等等; 同樣的, 在 JavaScript 的程式架構中, 也有許多類似的元件, 以下我們將為您詳細介紹。

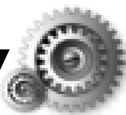
1-1 敘述

在一篇文章中, 最小的獨立單位是『句子』, 每一個句子都會表達出一些簡單的意思。同樣的, 在 JavaScript 的程式中也有最小的獨立單位, 那就是『敘述』 (Statement), 每一個敘述都會要求電腦做一件簡單的事情。每一個敘述原則上都單獨寫在一行, 若想在同一行內寫入數個敘述, 則每個敘述間就必須以一個分號 ';' 作分隔。

敘述是 JavaScript 程式中最小的可執行單位。 JavaScript 的敘述大致可分為下列幾種:

❑ 定義 / 宣告敘述 (Declaration statement): 如 var i, j;

著作權屬旗標公司，侵害必究・旗標 2003



- ❑ 運算式敘述 (Expression statement): 如 i= j+5;
- ❑ 條件選擇敘述 (Selection statement): 如 if (...)
- ❑ 迴圈敘述 (Iteration statement): 如 for (...)、 while (...)
- ❑ 跳躍敘述 (Jump statement): 如 break 、 continue 、 return
- ❑ 複合敘述 (Compound statement): 將多個敘述用『{ }』組合成一個敘述。

以上各項目看不懂也沒關係，稍後我們會一一說明。底下先來看個簡單的例子:

```
var a ◄────────── 定義變數 a
a = 1 ◄────────── 將 a 設定為 1
document.write(a) ◄──── 將 a 的值輸出到螢幕
```

在上面的程式中，定義了一個整數變數 a，並將 1 指定給 a，最後將 a 的值輸出到螢幕上。這 3 行都是敘述，我們只要將各種不同的敘述組織起來，就可以產生許多有用的程式。

複合敘述 (Compound statement): { }

我們可以將許多敘述組合起來，放入一對大括弧 { } 之中，則可形成一個敘述的集合，稱為複合敘述 (Compound Statement)，亦可稱為程式區塊 (Block) 。例如:

```
{
var i
i = 2 + 3
 document.write(i)
}
```


著作權屬旗標公司，侵害必究・旗標 2003

同時，複合敘述中還可以有其他的複合敘述，如:

```
{
{ var i; i = 5                    }
{ var j; j = 1+3; document.write(j) }
{ var k  { var m; m = 1 }  i = 2   }
}
```

因此，JavaScript 的敘述可大可小，可簡單也可複雜，完全看你怎麼去用它。在程式區塊之後 (即 '}' 之後) 不需要加分號，如果加了則該分號被視為空敘述，所以也不會有任何影響。以後我們會再慢慢為您介紹有關複合敘述的各項特性及使用時機。

1-2 註解: //... 與 /*...*/

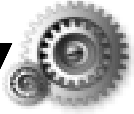
凡是以 "//" 開頭或是夾在 "/*" 和 "*/" 之間的文字都是註解，註解的目的只是用來提供一些說明。電腦在執行 JavaScript 程式時會自動略過這些文字，所以不會影響程式的正常運作。

```
// 這行純粹是註解，不會被電腦執行到...嘻嘻
```

在 "//" 符號開始，直到整行的結束，中間的文字都是註解，不會被電腦所執行。

在 JavaScript 敘述裡較為特殊的用法是，使用 HTML 的註解標籤 "**<!--**" 與 "**-->**" 搭配 JavaScript 註解符號 "//"，將 JavaScript 敘述包起來:

著作權屬旗標公司，侵害必究・旗標 2003



```
<script>
<!--
  JavaScript 敘述......
  JavaScript 敘述......
  JavaScript敘述......
  JavaScript 敘述......
  JavaScript 敘述......
//-->
</script>
```

使用這種混合型的註解將程式包起來

由於早期版本的瀏覽器並不支援 JavaScript 語言，使用這種混合型的註解將程式包起來，可以防止早期版本的瀏覽器將我們的 JavaScript 程式敘述顯示在網頁上。

1-3 流程控制: if

if 是 JavaScript 的流程控制指令，其格式可為單一的 if，或與 else 指令合用，請看下表:

格式	範例

if (條件)
    動作 1
	例：
	if (a > 0)
	    documnrt.write("a > 0")

或
if (條件)
    動作 1
else
    動作 2
	例：
	if (a > b)
	    max = a
	else
	    max = b

著作權屬旗標公司，侵害必究・旗標 2003

如果 if 的條件成立, 則動作 1 會被執行, 否則執行動作 2 (如果有 else 的話)。動作 1 和動作 2 可以爲單一敘述, 亦可爲複合敘述, 如果是複合敘述的話, 則須以 { } 括起來:

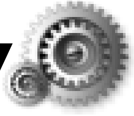
```
if (a > b)
  {
  max = a
  document.write("a 大於 b")
  }
```

if 的條件是一個運算式, 我們一般常用『比較算符』來計算條件是否成立。下面是 JavaScript 的比較算符說明列表:

算符	用途	實 例
>	大 於	a > b
>=	大於或等於	a >= 3
<	小 於	a < 5 + 4
<=	小於或等於	a <= b
==	等 於	a == b
!=	不等於	a != 9

如果比較的條件成立, 則結果爲 true, 否則爲 false 。

著作權屬旗標公司，侵害必究．旗標 2003



1-4 迴圈控制: while 與 for

while 的格式和 if 類似:

<table>
<tr><td>while(條件)
　　動作

註: 如果動作為複合敘述, 則須加 { }</td><td>例：
i = 0
while(i < 5)
　{
　document.write(i)
　i = i + 1
　}</td></tr>
</table>

只要條件為真, 動作就會一遍又一遍地做下去, 直到條件變為不成立為止。

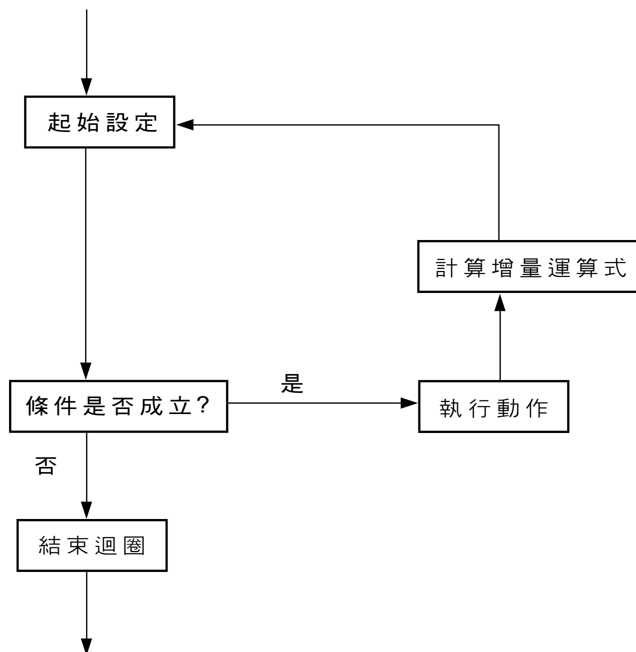
for 指令可以設定迴圈執行的次數, 其格式如下:

<table>
<tr><td>for (起始設定 ; 條件 ; 增量運算式)
　　動作
註: 如果動作為複合敘述, 則須加 { }</td></tr>
<tr><td>例：
　var i;
　for (i=0; i<10; i++)
　　{
　　documrnt.write("Loop: ")
　　documrnt.write(i)
　　documrnt.write("\n")
　　}

[註] ++ 是一個遞增算符, i++ 表示 i 本身的值加一之意 (即 i = i + 1 之意)。</td></tr>
</table>

著作權屬旗標公司，侵害必究・旗標 2003

起始設定只會做一次，就是在 for 迴圈剛開始的時候。整個迴圈的執行順序如下所示:

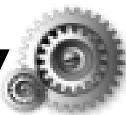


和 while 迴圈一樣，只要條件一直為真，動作就會一遍又一遍地做下去，直到條件變為假時才結束迴圈。

1-5 自由格式

JavaScript 程式的寫法是採用自由格式 (Free Format): 雖然每個敘述原則上均單獨寫在一行，但您還是可以將數個敘述寫在同一行，只要彼此間以分號 ';' 隔開即可。除此之外沒有別的限制，所以我們可以用非常自由的格式來書寫程式。例如將多行敘述合併成一行，或在敘述中任意插入空格。例如下面的寫法都是合法的:

著作權屬旗標公司，侵害必究・旗標 2003



```
var a
a       =      1
 document.write(a)
```

或

```
var a; a=1; document.write(a)
```

事實上, JavaScript 的分隔字元可包括空白字元和定位字元 (Tab), 所以多插入幾個分隔字元或減少幾個都不會影響程式的正確性。然而, 像上述的寫法雖然合法, 但降低了程式的可讀性, 所以我們建議您還是使用一個敘述一行的寫法較好。

1-6 函式

如果把能夠完成某項工作的各個敘述包裝起來, 製作成一個功能單元, 再給予一個名稱, 就形成了所謂的函式 (Function)。我們可以把常用的敘述群集合成為函式, 然後加以呼叫使用。例如, 我們把一段顯示時間的程式包裝起來成為一個名叫 clock() 的函式, 那麼每次只要呼叫 clock() 便可顯示時間, 而不必再重寫該段程式了。

一個函式的定義可分為三部份: 函式名稱、參數列和函式的內容:

```
function 函式名稱  (參數 1, 參數 2, ...)
 {
   ...
    (敘述的集合)
   ...
 }
```

例:

```
   function output (i, j)
      {
      document.write(i)
      document.write(j)
      }
```


著作權屬旗標公司，侵害必究・旗標 2003

1. **函式名稱：**可由英文字母、數字或底線 '_' 組成，而且開頭第一個字不可爲數字。

2. **參數列：**可以接收多個參數。所有的參數都是放在函式名稱之後的一對小括弧之內，如果沒有設定參數，這個小括弧亦不可省略。

4. **函式內容：**就是被包在一對大括弧 { } 之內的所有敘述。函式的內容可以很複雜，也可以空無一物。下面是一個最簡單的 JavaScript 函式：

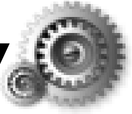
```
function nothing () ◄── 沒有參數
   {} ◄──────────── 內容空無一物
```

在函式的執行中，如果遇到 return 敘述，則會立刻結束函式，並傳回緊跟在 return 後面的值。例如:

```
function test()
  {
          ....
          return 2;          函式立刻終止並傳回整數 2
          ....
  }

  a = test()
```

若 test() 函式內沒有 return 敘述或 return 之後沒有東西，則不傳回任何數值。如此一來我們就不能寫出 a = test() 這樣的敘述了。

著作權屬旗標公司，侵害必究・旗標 2003

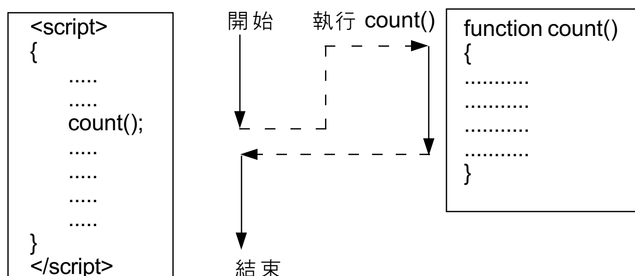


注意，每個函式都是獨立的個體，所以函式之中不可以再有其他的函式定義。如果以 JavaScript 程式的結構層次來看，則每一層次與層次之間都是獨立而不互相干擾的：

```
// 檔案 (一個網頁文件內可以有多個 <script>...</script> 區段
  <script>
    // 函式 (一個 <script>...</script> 區段間可以有多個函式)
      function kkk()
      {
      // 敘述 (一個函式內可以有多個敘述)
      ......
      ......
      }
      ......
      ......
  </script>
......
......
```

1-7 JavaScript 的程式組成

讀者可以把函式想像成是具有某種特殊功能的大型複合敘述，而 JavaScript 程式的主幹即是由這些大型敘述 (函式) 所構成的。至於程式執行的順序則如下圖所示：



著作權屬旗標公司，侵害必究・旗標 2003

當然了, 在一個 HTML 檔案內並不限定您只能有一段用 <script>...</script> 標籤包起來的 JavaScript 程式敘述, 當一個 HTML 檔案內有好幾段用 <script>... </script> 標籤包起來的 JavaScript 程式敘述, 則瀏覽器會由第一段 JavaScript 程式敘述開始執行起, 一直到最後一段 JavaScript 程式敘述。

這時候要注意到的是, 由於 JavaScript 是由上而下逐一執行各段的 JavaScript 程式敘述, 如果您將某個函式的定義放在後面, 但卻在前面的程式敘述中呼叫到這個函式, 則會發生錯誤:

```
<script>
    ...... (JavaScript敘述)
   ......
   count()  ◄────────  程式執行到這裡便會發生錯誤, 因爲
</script>              呼叫到一個尚未定義的函式
......
...... (HTML部份)
......
<script>
  function count()
   {
   ......
    ...... (JavaScript敘述)
   ......
   }
</script>
```

若整個網頁都載入完畢後, 再由 <BODY onLoad=”count()”>啓動運作, 自然不會出現這樣的錯誤。

在程式中呼叫函式的方法, 就是寫出函式名稱, 然後再加上一對小括弧 () 即可。小括弧內可以有多個參數, 分別對應到函式定義時的參數列上。

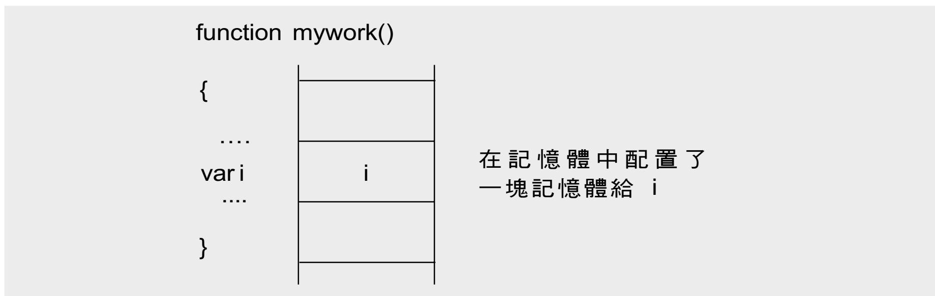
著作權屬旗標公司，侵害必究・旗標 2003

1-8 定義變數與函式

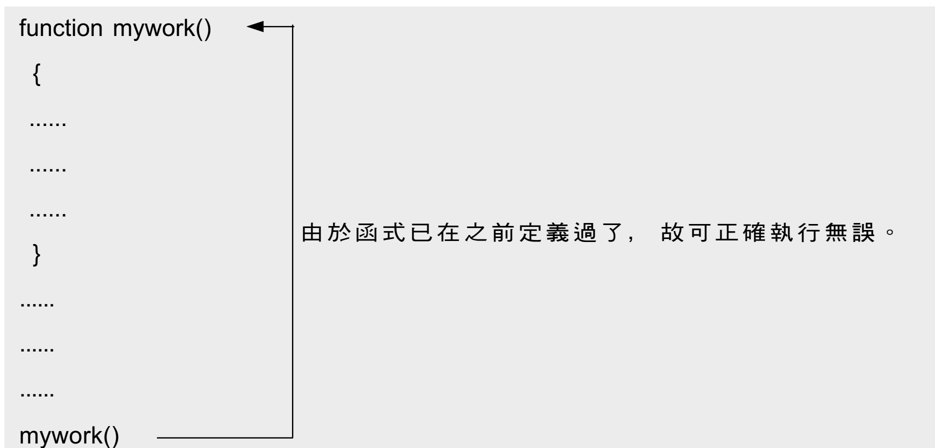
在電腦程式中，最重要的 2 項資源就是資料儲存及可執行的程式碼區塊。若將之對映到 JavaScript 語言上，則是變數及函式。

電腦中	資料儲存	可執行的程式碼區塊
JavaScript 語言	變數	函式 (敘述的組合)

以變數來說，當我們定義了一個變數，那麼程式在執行時就會要求電腦配置一小塊記憶體給這個變數。例如：



函式在使用 (呼叫) 之前必須先加以定義，如此電腦才能找到正確的函式來執行。



著作權屬旗標公司，侵害必究・旗標 2003

在 JavaScript 裡, 我們用 function 敘述定義函式, 用 var 敘述定義變數。

1-9 註解的使用技巧

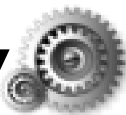
註解只是用來對程式的內容做一些說明, 讓人較容易閱讀而已, 電腦在執行 JavaScript 程式時會自動將註解的部份除去, 對程式的執行並無任何影響。

> JavaScript 有 2 種形式的註解:
>
> 1. 以 "/*" 和 "*/" 包起來的文字資料。
>
> 2. 由 "//" 開頭, 到該行結束爲止之間的文字資料。

以 "/*" 和 "*/" 包起來的註解, 可以放在程式中任何允許有空白的地方, 並可以延續好多行。例如:

```
  /*======================
  Script Name: MyScript
  Created By : FongHsinTz
  ======================*/

function  rain()
  { /* This is the function rain */
    var i           /* define a variable */
     /* following program is just for demo */
    var j           /* define a variable */
    /* assign 4 to i */   i = 4
    /* assign 5 to j */   j = 5
  }  /* end of function rain */
```


著作權屬旗標公司，侵害必究・旗標 2003



以上都是合法的註解，但是過多的註解反而會和程式混淆，造成不易閱讀的困擾。同時，這種註解也不可以有重疊的情形，例如下面的註解是錯誤的:

```
/* Comment /* can't */ be nested */
```

由 "//" 開頭的註解，其有效範圍是到一行的結束，而且不會有重疊的問題:

```
// This nested comment // is allowed.
// This /* combined comments */ is allowed.
```

JavaScript 看到第一個 "//" 時，會直接跳到下一行繼續編譯，所以即使這個註解中還有其他的 "//" 存在，也不會造成 JavaScript 的困擾。

JavaScript 的註解	起始符號	終止符號	註解內容中不可有
第一種	/*	*/	/* 或 */
第二種	//	換行符號	換行符號

另外，註解符號的 2 個字元間不可有空白，下面都是錯誤的例範:

```
/ *  This is an error comment */
/ /  This is also an error comment
```

較好的註解風格

註解的主要目的，就是要增加程式的可讀性，或是提供一些額外的資訊，請看下面例子的註解部份:

```
var i                    // 作為暫時儲存最大值之用
i = MaxValue(5, 6)       // MaxValue() 的函式主體在前段程式敘述中
```


著作權屬旗標公司，侵害必究・旗標 2003

如果註解的內容只是重複敘述程式本身的動作，那麼就沒有什麼意義了 (除非做爲教學之用)，如下面的例子:

```
var i                  // 定義一個整數變數 i
i = MaxValue(5, 6)     // 求出 4 和 5 的最大值, 並存入 i 中
```

對一個懂得 JavaScript 的人來說，上面這 2 行註解根本是多餘的，我們應該避免這種用法。事實上，註解加得多並不一定好，必須用得恰當才行。

程式的自我詮釋技巧

假如我們能提高程式本身的可讀性，那麼就可降低對註解的依賴了，下面是同一個程式的 2 種寫法:

不易閱讀的程式	較好的寫作風格
var i	var MaxVal
i = mv(4, 5)	MaxVal = MaxValue(4, 5)

上例中的第 2 種寫法，由於程式本身即具備了自我詮釋的功能，所以這 2 行根本不需要加任何的註解。

一個好的程式，除了要增加程式本身自我詮釋的能力外，還是免不了要加上一些註解，而且註解應集中在解釋程式在做什麼、爲什麼要這樣做，以及提供額外的必要資訊上。我們應養成在程式中加上適當註解的習慣，以便日後供自己或他人閱讀之需。

著作權屬旗標公司，侵害必究・旗標 2003

資料型別、常數、變數與陣列

在電腦的記憶體中，資料是以位元 (bit) 逐一存放的。如果要我們直接去處理像 01011... 這種資料，那實在是太困難也太辛苦了。幸好 JavaScript 都已將這些位元資料做了比較高階一點的抽象化，讓我們可以直接使用如字串、數值等較爲熟悉的資料單元，而這些抽象的資料單元，就是所謂的『資料型別』(Data type)，一般簡稱爲『型別』。

2-1 資料型別

JavaScript 的基本資料型別有 4 種:

1. **數值型別:** 如 1、2、3、4、600、1000、3.14159 等等的數字資料。

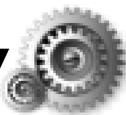
   數值還可細分爲**整數**與**浮點數**兩種。所謂的**浮點數**是指像 1.234 這樣小數點後面還有數字的數值。而**整數**當然就是不含小數點，以及小數點後面的數字! 例如 25。

2. **字串型別:** 如 "ABC"、"風信子"、"F-14" 等非純數字的字元資料。
3. **布林型別:** 只有 true (眞) 或 false (假) 兩種值，用於程式的流程控制。
4. **空 (Null) 型別:** 當我們定義一個變數，而尚未設定該變數的初始值，則該變數的型別即爲空 (Null) 型別。

2-2 常數

所謂『常數』(Constant)， 在科學上的定義是『不會隨時間而改變的數』，像圓周率、水銀的比重、光速等；同樣的，對程式語言來說，常數就是『不會在程式執行過程中改變的數』。

著作權屬旗標公司，侵害必究・旗標 2003



當我們在程式中使用如 1 、'a' 或 24.51 時, 這樣的資料就是常數。每一個常數都會有一個型別, 例如 1 、 24.51 爲數值型別, JavaScript 看到這些常數時, 會自動爲其設定適當的型別, 所以不必我們操心。

整數常數

整數類型的常數可以用 10 、 8 或 16 進位的方式來表示, 例如 18 可用下列三種方式來表達:

```
18          // 10 進位
022         //  8 進位
0x12        // 16 進位
```

凡是以 0 (零) 開頭的常數均會被 JavaScript 解譯成 8 進位, 而以 0x 或 0X 開頭的則視爲 16 進位。

浮點常數

浮點常數的表示, 可以用一般小數點的寫法, 也可以用科學表示法, 而科學符號中的 e 可爲小寫或大寫。例如:

```
456.567     // 一般小數點的寫法
-2.22e8     // 科學表示法, 等於 -222,000,000
3.5E-5      // 科學表示法, 等於 0.000035
```


著作權屬旗標公司，侵害必究・旗標 2003

字串常數

字串在 JavaScript 中是很特殊的一種資料，它既被視爲是一種包含字元資料的型別，也被視作是一個獨立的物件 (這樣的用法第 5 章會詳述)。字串常數的表示法則是將一連串的字元放在一對雙引號『"』或一對單引號『'』之間，JavaScript 會將引號內的資料視作字串資料。

```
'This is a book'
"我的名字是風信子"
```

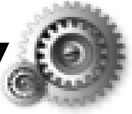
至於不可見的字元，則需用反斜線 '\' 後面跟著 1 個可見字元 (此方式稱爲 Escape sequence)，代表某些特殊的控制字元:

符號	用途
\b	退位
\f	換頁
\n	換行
\r	游標至行首
\t	水平定位
\"	雙引號

此外，我們也可以將一個長字串分成數個較小的字串，其間以 '+' 號分隔，則 JavaScript 會自動將之連起來:

```
MyNote = "String " + "can "
    + "connect " + "to " + "next "
    + "line."
```


著作權屬旗標公司，侵害必究・旗標 2003



2-3 代名、識別名稱與保留字

我們都知道，無論是資料或是程式碼都是存放在記憶體中，而記憶體則是按照所謂的位址 (Address) 來編排的，這就好像我們的門牌地址一樣，每戶都有唯一的地址，如此信件往來才能順利的進行。

代名

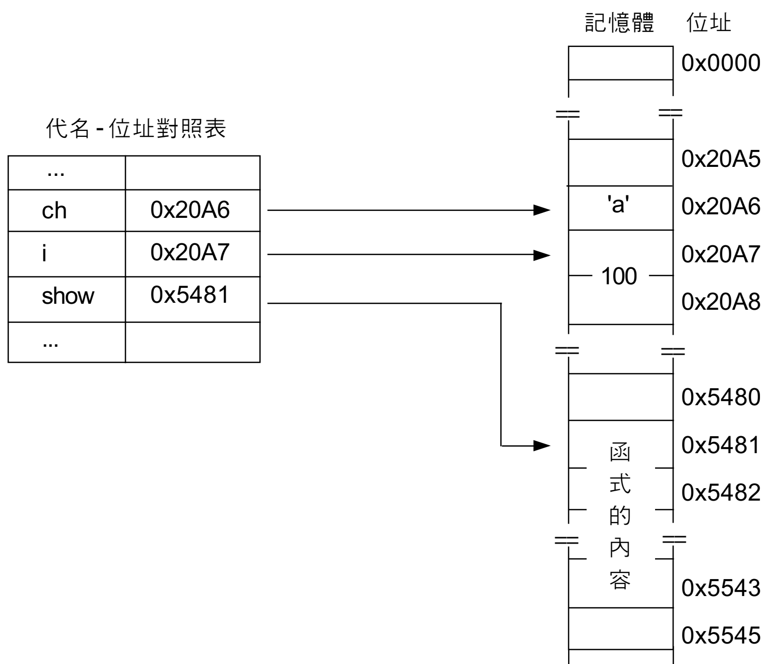
記憶體的最小單位是字元 (byte)，因此每一個字元都有其專屬的位址編號。然而，無論是資料或函式，其所佔的記憶體大小並不固定，因此我們都是以它們的起始位址來做代表。爲了使用上的方便，一般程式語言均允許我們以較明顯易懂的文字名稱來代表位址編號，而這個文字名稱就叫做『代名』 (Name)。

在 JavaScript 中，變數和函式的名稱都是代名，例如下面的程式中，ch 、i 和 show 都是代名:

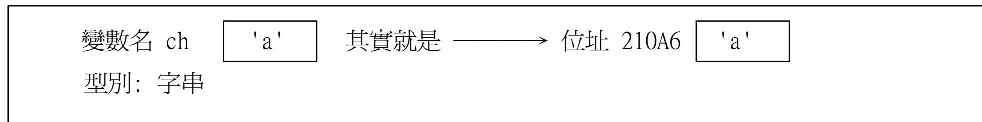
```
function show()
 {
   var ch
   var i
   ch = 'a'
   i  = 100
  ......
  ......
 }
```

JavaScript 在執行程式時會建立一個『代名 - 位址對照表』，並自動將代名轉換成它們真正代表的相對位址:

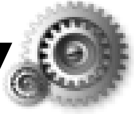
著作權屬旗標公司，侵害必究・旗標 2003



在抽象的概念裡，我們也可以把代名想像成在某『位址』上的一塊記憶體；而這塊記體的大小，以及對其內容的詮釋方式則依代名的型別而定。至於如何將代名轉換成位址，這是 JavaScript 的工作，不用我們去操心。



著作權屬旗標公司，侵害必究・旗標 2003



識別名稱及其命名規則

在 JavaScript 中, 變數和函式的代名都是識別名稱的一種, 所謂『識別名稱』 (Identifier) , 就是可以用來加以辨識的名稱, 這就好像每個人都要有一個名字一樣。

識別名稱必須由大小寫的英文字母, 數字或底線符號 ('_') 所構成, 而且第一個字不可爲數字。下面是一些合法及非法的例子:

合法的識別名稱	非法的識別名稱
abc	abc$ // 不可有 $
MaxValue	9plus // 不可用數字開頭
No0274	my book // 中間不可有空白
_my_book	

同時, 大小寫的英文字是不一樣的, 例如 Abc 和 abc 會被當成是不同的識別名稱。在命名時要注意其可讀性, 一般均以小寫來命名, 並且使用較有意義的字眼, 例如:

```
var  age, tax, salary, date;
```

若爲複合字, 則我們可以用底線來分開各單字, 或將各單字的第一個字元以大寫表示:

```
int is_error, draw_line, new_tax;
int isError,  DrawLine,  newTax ;
```


著作權屬旗標公司，侵害必究・旗標 2003

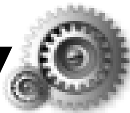
保留字

JavaScript 預先保留了一些識別名稱以供語言本身使用，這就是所謂的保留字 (Keyword)。我們在命名時必須避開這些保留字，以免造成錯誤。下面是 JavaScript 的保留字:

abstract	boolean	break	byte	case
catch	char	class	const	continue
debugger	default	delete	do	double
else	enum	export	extends	false
final	finally	float	for	function
goto	if	implements	import	in
instanceof	int	interface	long	native
new	null	package	private	protected
public	return	short	static	super
switch	synchronized	this	throw	throws
transient	true	try	var	void
volatile	while	with		

當然，有些保留字在目前的 JavaScript 版本內並沒有用到，但這些保留字有可能會在未來的 JavaScript 版本內 (否則就不會事先列入保留字了)，所以爲了確保您的 JavaScript 程式能順利地在未來的瀏覽器軟體上執行，命名變數時請避開這些保留字。

著作權屬旗標公司，侵害必究・旗標 2003



2-4 變數

變數 (Variable) 和常數一樣，都在記憶體中佔有一塊空間，並且其內可以存放一個值；然而，變數的值是可以任由我們改變的，爲了做到這一點，我們必爲它取一個識別名稱，然後再以這個名稱來進行各種操作。

```
var i, j
var a, b, c
var f
```

設定變數的初始值

在定義一個變數的時候，JavaScript 會配置一塊記憶體這個變數，如果這時候我們不設定變數的初始值，則變數的型別便是空型別。

我們可以在定義變數的時候設定其初始值，把初始放在等號之後。下面是一些例子：

```
var i = 1,    j = 5
var a = 100.2, b = 0.9999
```

另外，我們也可以用運算式來做爲變數的初始值：

```
var i = 2, j = i - 6;   // j = -4
var k = i + j          // k = -2
```


著作權屬旗標公司，侵害必究・旗標 2003

變數的作用範圍

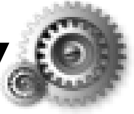
當我們使用 var 這個關鍵字來定義一個變數時, 若 var 敘述放置在某個函式裡面, 則該變數便只能在該函式內使用, 稱做區域 (local) 變數; 若將 var 敘述放置在函式外, 則該變數便能被整個程式 (包括所有的函式) 所共同使用, 稱做全域 (global) 變數。若是使用一個變數前沒有先以 var 敘述定義, 則不管該敘述在函式內或函式外, 一律都視爲全域變數:

```
<script>
var i = 1 ←全域變數
......
......
function show(dat)
 {
 var k = dat ←區域變數
 ......
 ......
 w = 1024 ←—全域變數 (因爲 w 變數之前尚未定義過)
 ......
 ......
 }
</script>
```

變數的多樣性

JavaScript 比起其他語言較爲特殊的是變數的多樣性。例如:

著作權屬旗標公司，侵害必究・旗標 2003



```
var i
i = 100
document.write(i)    // 印出數值 100
i = 'I am the law.'
document.write(i)    // 印出字串 'I am the law.'
```

同一個變數 i, 在前一刻可以是一個數值型別的數字, 但下一刻馬上就可以搖身一變, 成為一個不折不扣的字串。

隱藏式的型別轉換

另外, 當字串與數值混合運算時, JavaScript 會試著先將數值轉換成字串後, 再參與運算:

```
var k = 'We Are The World.'
m = 100 + k              //結果 m='100We Are The World.'
n = k + 100              //結果 n='We Are The World.100'
```

但是如果使用到數學函式, 需要一個數值時, 則 JavaScript 也會試著先將字串轉換成數字:

```
var k = '-100'
m = Math.abs(k)   //Math.abs(x) 是用來計算數字的絕對值。
```

Math 為 JavaScript 內建的數學函式物件, 詳細使用方法請看第 5 章介紹。

以上面這個例子來說, 變數 k 原本為字串, 但在參與運算的過程中, JavaScript 會先試著將 k 由字串 "-100" 轉換成數字 -100, 再傳送給 Math.abs(), 結果使 m = 100 。

著作權屬旗標公司，侵害必究・旗標 2003

以這個例子來說, 由於變數 k 其所包含字串 "-100" 能順利的轉換成數值 -100, 所以程式能正確執行。若變數 k 其所包含字串無法順利的轉換成數值 (如 k= "abc"), 則程式不免就會發生錯誤了。

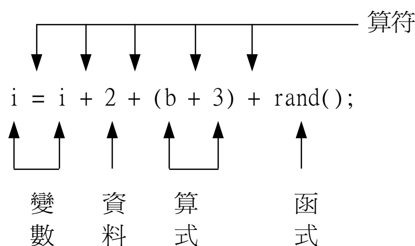
著作權屬旗標公司，侵害必究．旗標 2003

運算式與算符

對於基本型別的資料, JavaScript 已預先定義好了一組算符(Operators), 可以讓我們很方便地來作各種運算; 我們將詳細討論這些算符的操作方式。

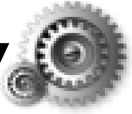
3-1 運算式的組成元件

運算式 (Expression) 是算符與資料的合法組合。算符的運作對象是資料, 因此所有的變數、能傳回資料的函式, 以及算式本身均能由算符來加以運作。所以更明確的說, 算式是算符與資料、變數、函式或其他算式的合法組合:



所有的運算式經過運算後都會產生一個結果值, 這個結果值就代表該運算式本身的值。以上面的運算式 (b + 3) 為例, JavaScript 會先算出 b 加 3 的和做為其值, 然後再用這個值來參與其他的運算。若運算式內有函式, 則該函式必須傳回一個值, 而這個值則會被用來繼續參與其他的運算。

著作權屬旗標公司, 侵害必究・旗標 2003



3-2 算符的優先順序與結合順序

當一個運算式中有超過一個以上的算符時，例如：a=b+c*d，我們就必須考慮到每個算符執行的『優先順序』(Priority)了。由經驗(先乘除後加減) 知道，c*d 會先被運算，然後其結果再和 b 相加，最後才將加後的結果透過 '=' 算符設給變數 a，所以我們可以說 '*' 算符的優先程度較 '+' 算符還高。

對於 a=b+c*d 這個運算式有另一方面也是要考慮的，這就是執行的方向，雖然 '+' 和 '*' 算符是無左右方向之分 (即 a+b 與 b+a 結果一樣),可是其他算符如除法或餘數算符就會有左右之分了，例如 a-b-c，是要先做 a-b 呢？還是先做 b-c？對於這種執行方向的判別,我們稱其為『結合順序』(Associativity)，大部份算符的結合順序都是由左到右。

```
a-b-c;  ◄──────  相當於 (a-b)-c; ('-' 是由左到右)
```

下面是由簡單到複雜的各種運算式，它們都是合法的例子：

```
a = a + 23;
a = a + 23 + ( a = b = max(a, b) + b + 23);
```

3-3 算符種類

JavaScript 提供了許多預先定義好的算符 (Operator)，其中有一些只作用在單一的資料上，如正負號(+,-)、遞增(++)、遞減(--)等，稱之為『一元算符』(Unary operator)；其他的算符則必須放在二個資料之間才能運作，我們稱之為『二元算符』(Binary operator)。

著作權屬旗標公司，侵害必究・旗標 2003

還有一些符號可身兼一元算符及二元算符的功能, 如 '-' 可以做為負號, 也可做為減號, 算符的功能完全視其在運算式中前後文的狀況而定。

接下來我們會先為您介紹各個算符的功能及特性, 然後介紹它們之間的優先順序和結合順序, 最後再說明不同型別資料的相互轉換方式。

3-4 數學算符

最常用的算符大概就是 +、-、*、/、% (加、減、乘、除、求餘數) 等 5 個數學算符 (Arithmetic Operator)。 '%' 是取餘數的算符, a%b 的結果會得到 a 除以 b 的餘數。請注意, 參與 % 運算的運算元必須是整數。我們以底下的表格來說明:

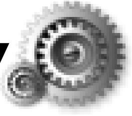
算符	使用例	說　明
+	a+b	把 a 的值和 b 的值相加
-	a-b	以 a 的值減去 b 的值
*	a*b	把 a 的值乘以 b 的值
/	a/b	把 a 的值除以 b 的值
%	a%b	取 a 除以 b 的餘數

3-5 遞增 (++) 與 遞減 (--) 算符

遞增算符 (Increment Operator) 可以使變數的值加一, 遞減算符 (Decrement Operatro) 可以使變數的值減一, 其使用法如下表所示:

算符	使用例	說明	相當於
++	i++ 或 ++i	i 的值加一	i=i+1
--	i-- 或 --i	i 的值減一	i=i-1

著作權屬旗標公司，侵害必究・旗標 2003



++ 與 -- 可以放在變數的前面或後面, 都是用來使變數值加一或減一, 不過若與其它算符合併使用時, 則意義是不相同的。 ++ 或 -- 在變數前, 表示 ++ 或 -- 是在其它算符之前執行的, 而 ++ 或 -- 若在變數之後, 則表示 ++ 或 -- 是在其它算符之後執行的。例如:

```
i=1;
j=++i; // 先將 i 加 1, 然後再取出 i 的值來參與其他的運算
```

運算過程是 i 值加 1 變爲 2, 然後再設給 j,結果是 i=2 、 j=2 。而

```
i=1;
j=i++; // 先取 i 的值來參與其他的運算, 最後再將 i 加 1
```

運算過程是 i 的值先設給 j, 然後再加 1 變爲 2,結果是 i=2 、 j=1, 所以結果不同。 ++ 與 -- 只能對單一變數運算, 不能對複合運算式或常數做運算。例如:

```
(i + 3)++;  // 錯誤
```

++ 與 -- 具有相當高的運算優先順序, 比如 x*y-- 等於是 (x)*(y--) 而不是 (x*y)--,並且請記得 (x*y)-- 也是不合法的。所以:

```
a = (b - c--)/2;
```

是整個式子算完了, 然後 c 再 -- 。因爲 ++ 、 -- 的優先性是對單一變數而言的。

另外, 在算式中使用 ++ 、 -- 的變數最好不要重複出現, 如:

```
y=3;
x = y/2 - (++y + 3);  ←——— 出現兩個 y
```


著作權屬旗標公司，侵害必究・旗標 2003

這樣到底是先算 y/2 呢? 還是先算 ++y 後再算 y/2 ? 老實說, 一眼看過去很難分辨得出, 而且實際運算結果也不易預料, 所以要特別小心去避免。

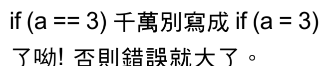
3-6 比較算符

JavaScript 語言共有 6 個做爲比較用的比較算符 (Relational Operator), 其運算結果只傳回眞 (true) 或假 (false)。其使用法如下表所列:

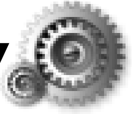
算 符	使 用 例	說 明
>	i > j	比較 i 是否大於 j
<	i < j	比較 i 是否小於 j
==	i == j	比較 i 是否等於 j
>=	i >= j	比較 i 是否大於或等於 j
<=	i <= j	比較 i 是否小於或等於 j
!=	i != j	比較 i 是否不等於 j

由於只做比較而不做設定, 所以執行後各運算元的值不變。例如:

```
if (a == 3)   // 比較 a 是否等於 3
  { 敘述1 }   // 如果 a 等於 3 則執行敘述1
```



著作權屬旗標公司，侵害必究・旗標 2003



3-7 邏輯算符

JavaScript 提供有 3 個邏輯算符 (Logical Operator), 邏輯算符是取運算元的眞假值來參與運算, 運算結果也只傳回眞 (true) 或假 (false)。其使用法如下表所示:

算符	使用例	說明
!	!i	對 i 做邏輯的 NOT
&&	i && j	i 與 j 做邏輯的 AND
||	i || j	i 與 j 做邏輯的 OR

所謂的『邏輯的 AND 、 OR 、 NOT』是由底下的眞値表所定義:

AND	眞	假
眞	眞	假
假	假	假

OR	眞	假
眞	眞	眞
假	眞	假

NOT	眞	假
	假	眞

若是 "&&", 二者都必須爲眞結果才會爲眞, 否則爲假。若是 "||", 二者只要有一個爲眞結果即爲眞, 否則爲假。若是 '!' , 則眞變假, 假變眞。

比較或邏輯運算所產生的結果只有 true 或 false 二種可能。此結果可以用作判斷之用 (如 if, while)。例如:

```
for ( a==1 && b==2 ) { 敘述1 }
```

則當 a==1 與 b==2 兩者同時成立 (兩者都爲 true) 時, JavaScript 才會執行**敘述 1** 。

著作權屬旗標公司，侵害必究・旗標 2003

3-8 位元算符

JavaScript 的特徵之一就是能對資料做相當低階的處理。它總共提供了 6 種位元算符 (Bitwise Operator), 大大的提高了 JavaScript 的細部處理能力。位元算符會把資料看成是 bit 的集合, 其運算方式也是位元逐一地來處理:

算符	使用例	説明
<<	i << j	把 i 的位元左移 j 個 BIT
>>	i >> j	把 i 的位元右移 j 個 BIT (i 有正、負之分)
>>>	i >>> j	把 i 的位元右移 j 個 BIT (i 無正、負之分)
~	~i	把 i 的每一位元反相
|	i | j	i 與 j 對應的位元做 OR
&	i & j	i 與 j 對應的位元做 AND
^	i ^ j	i 與 j 對應的位元做 XOR

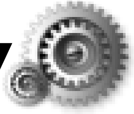
其中 XOR 的眞值表如下:

XOR	眞	假
眞	假	眞
假	眞	假

◆ 恰有一爲眞則結果爲眞, 否則爲假。

位元算符再運算之前會將運算元一律轉成 32 bits 的整數型態來運算。下例的註解內均以二進位來表示:

著作權屬旗標公司，侵害必究・旗標 2003



```
char i = 8;      //   i=    00000000 00000000 00000000 00001000

i = i >> 1;      //         00000000 00000000 00000000 00001000
                 //   >>1   0000000 00000000 00000000 00000100
                 //   --------------------------------------
                 //   i=    00000000 00000000 00000000 00000100

i = i << 3       //         00000000 00000000 00000000 00000100
                 //   <<3 00000 00000000 00000000 00000100
                 //   --------------------------------------
                 //   i=    00000000 00000000 00000000 00100000

i = ~i;          //   ~     00000000 00000000 00000000 00100000
                 //   --------------------------------------
                 //   i=    11111111 11111111 11111111 11011111

i = 14 & 23      //     00000000 00000000 00000000 00001110 (14)
                 //   & 00000000 00000000 00000000 00010111 (23)
                 //   --------------------------------------
                 //   i=00000000 00000000 00000000 00000110 (6)

i = 14 | 23      //     00000000 00000000 00000000 00001110
                 //   | 00000000 00000000 00000000 00010111
                 //   --------------------------------------
                 //   i=00000000 00000000 00000000 00011111

i = 14 ^ 23      //     00000000 00000000 00000000 00001110
                 //   ^ 00000000 00000000 00000000 00010111
                 //   --------------------------------------
                 //   i=00000000 00000000 00000000 00011001
```


著作權屬旗標公司，侵害必究．旗標 2003

在右移 ">>" 或左移 "<<" 時, 移轉出去的位元將被捨棄, 而空出的的部份則自動補 0 。但若該資料爲負數, 那麼右移後左邊的空白部份將補 1 。

事實上, 左移 n 即是乘上 2 的 n 次方, 而右移 n 則是除以 2 的 n 次方, 例如:

```
5 << 1   即是 5 * 2 = 10
5 >> 2   即是 5 / 4 = 1
```

由於乘、除法會佔用較多的 CPU 時間, 在某些情況下我們可以改用右、左移來運算, 以加快執行速度。(但可讀性則會降低)

">>" 或 ">>>" 均是右移, 兩者的差別在於 ">>>" 不管所處理的數值是正數或負數, 移轉出去的位元一律被捨棄, 而空出的 的部份則自動補 0 。而當 ">>" 遇到所處理資料爲負數, 那麼右移後左邊的空白部份將補 1 。

3-9 指定算符 '='

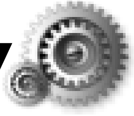
'=' 號是指定算符 (Assignment Operator), 它會把 '=' 右邊算式的值設給左邊的變數。其使用方式我們都已經很熟悉了, 但要注意 '=' 的左邊必須是一個可以存入資料的變數。

```
5 = a + b;     // 錯誤
(a+b) = 6;     // 錯誤
fun() = 6;     // 錯誤
```

當我們把許多等號串接起來時, 其結合順序是由右往左:

```
a = b = c = 5;  // 相當於: a = (b = (c = 5));
```


著作權屬旗標公司，侵害必究・旗標 2003



'=' 可以和算數或位元算符合併使用，而以 "op=" 表示 (op 表合用的算符)，此時 "op=" 的左邊也只能是變數，而右邊可以是任何算式。茲將所有可以合併的算符列表如下：

算符	使用例	相當於
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b
>>=	a >>= b	a = a >> b
>>>=	a >>>= b	a = a >>> b
<<=	a <<= b	a = a << b
|=	a |= b	a = a | b
&=	a &= b	a = a & b
^=	a ^= b	a = a ^ b

如果 "op=" 右邊爲複合運算式，則該複合運算式會先計算，然後再以 "op=" 來操作，例如：

```
a *= b + c;
```

是 a = a * (b + c); 之意，而非 a = a * b + c (乘法的優先順序較加法爲高）。

著作權屬旗標公司，侵害必究・旗標 2003

3-10 條件算符: '?:'

JavaScript 提供一個『條件算符』(Conditional operator) 可以代替簡單的 if-else 指令, 其使用格式為:

算符	使用例	說明
?:	i ? j : k	若 i 為真, 則取 j, 否則取 k

例如:

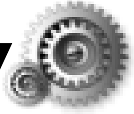
```
int a = b>c ? b:c;          // 將 b 和 c 的最大值存入 a 中
a<10 ? fun1():fun2();       // 若 a<10, 執行 fun1(),
                            // 否則執行 fun2()
```

3-11 運算式中的優先順序與結合順序

以下就是 JavaScript 算符的優先順序及結合順序表:

優先順序	符號	功能及使用場合	結合性
1. 逗號算符	,	分隔開變數或算式	由左到右
2. 指定及複合算符	=	指定	由右到左
	+=	相加後再存入左邊的變數	
	-=	相減後再存入左邊的變數	
	*=	相乘後再存入左邊的變數	
	/=	相除後再存入左邊的變數	
	%=	取餘數再存入左邊的變數	
	>>=	右移後存入左邊的變數	
	<<=	左移後存入左邊的變數	
	&=	位元 AND 後存入左邊的變數	
	^=	位元 XOR 後存入左邊的變數	
	|=	位元 OR 後存入左邊的變數	

著作權屬旗標公司，侵害必究・旗標 2003



<table>
<tr><td>3. 條件算符</td><td>?:</td><td>Exp1 ? Exp2 : Exp3
檢查 Exp1 是否爲眞，眞則
執行 Exp2，僞則執行 Exp3</td><td>由右到左</td></tr>
<tr><td>4. || 邏輯算符</td><td>||</td><td>運算式做 "OR" 運算</td><td>由左到右</td></tr>
<tr><td>5. && 邏輯算符</td><td>&&</td><td>運算式做 "AND" 運算</td><td>由左到右</td></tr>
<tr><td>6. | 位元算符</td><td>|</td><td>每個位元做 "OR" 運算</td><td>由左到右</td></tr>
<tr><td>7. ^ 位元算符</td><td>^</td><td>每個位元做 "XOR" 運算</td><td>由左到右</td></tr>
<tr><td>8. & 位元算符</td><td>&</td><td>每個位元做 "AND" 運算</td><td>由左到右</td></tr>
<tr><td>9. 邏輯等值
算符</td><td>==
!=</td><td>是否相等
是否不相等</td><td>由左到右</td></tr>
<tr><td>10. 比較關係
算符</td><td><
<=
>
>=</td><td>小於
小於等於
大於
大於等於</td><td>由左到右</td></tr>
<tr><td>11. 移位(shift)
算符</td><td>>>
>>>
<<</td><td>右移運算(考慮符號)
右移運算(不考慮符號)
左移運算</td><td>由左到右</td></tr>
<tr><td>12. 加法算符</td><td>+
-</td><td>加法運算
減法運算</td><td>由左到右</td></tr>
<tr><td>13. 乘法算符</td><td>*
/
%</td><td>乘法運算
除法運算
餘數運算</td><td>由左到右</td></tr>
<tr><td rowspan="2">14. 一元算符</td><td>!
~</td><td>否定
取 1 的補數</td><td>由右到左</td></tr>
<tr><td>++
--</td><td>增 1 (Increment)
減 1 (Decrement)</td><td>由右到左或
由左到右</td></tr>
<tr><td>15. 特殊算符</td><td>( )
[ ]</td><td>函式呼叫用的小括弧
陣列元素</td><td></td></tr>
</table>

算符的優先順序及結合順序

在同一欄內的各算符具有相等的優先順序，若這些算符同時出現在運算式中，則可依其結合順序來決定是從左邊或右邊開始運算。

著作權屬旗標公司，侵害必究・旗標 2003

我們可以用小括弧來更改運算的優先順序及結合順序，凡是用小括弧括起來的算式會優先運算。若有巢狀的小括弧，則較裏面的小括弧先算：

```
b = 3
c = 7
d = 9
a = b * c + d      //結果 a = 30
a = b * ( c + d )  //結果 a = 48
```


著作權屬旗標公司，侵害必究・旗標 2003

流程控制與函式

所謂的流程控制 (flow control) 是以控制程式的執行方向,進而達到掌握程式動態爲目的。流程控制是 JavaScript 程式的靈魂, 它包含:條件判斷、迴圈控制兩大類:

一、**條件判斷控制**: 判斷條件的眞僞,然後程式依眞僞的情形至指定的地方去執行程式。這方面的指令有:if...else 。

二、**迴圈控制** : 程式依指定的條件做判斷,若條件成立則進入迴圈執行迴圈內的動作。 每執行完一次迴圈內動作便再回頭做一次條件判斷, 直到條件不成立後才結束迴圈, 屬於這方面的流程控制指令有 : while 、 for 兩種。

4-1 條件式的眞假

爲什麼我們稱條件式而不簡單的稱條件呢? JavaScript 的條件判別式其實就是一個運算式, 而所謂條件的眞假就是以此算式的運算結果來判別的。例如:

條件式	說明
a == b	當 a 與 b 相等時, 則結果爲眞 (true)
a==5 && b==4	當 a 值爲 5 並 b 值爲 4 時, 則結果爲眞(true)
a==5 || b==4	當 a 值爲 5 或 b 值爲 4 時, 則結果爲眞(true)
a>=5 || a<=10	當 a 的值介於 5 ~ 10 之間, 則結果爲眞(true)
!a	當 a 值爲眞(true), 則結果爲假(false) 當 a 值爲假(true), 則結果爲眞(false)

著作權屬旗標公司,侵害必究・旗標 2003

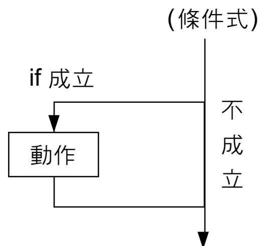
4-2 條件判斷 if...else... 結構變化

if 指令

if 是最常用的條件判斷指令, 其格式為:

```
if (條件式)
  {動 作}
```

意義如下圖:



執行 if 敘述時,若條件式為眞, 則執行 { } 內的動作, 若條件式不成立 ,則略過該動作而繼續往下執行。比如:

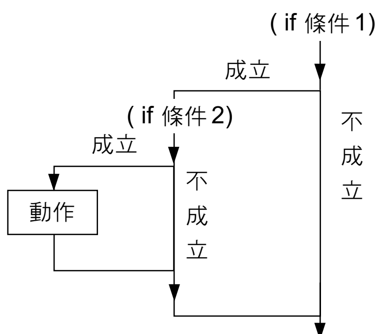
```
.....
if (c=='q')
  alert("Good-Bye!")
.....
```

當 c 的值是 'q' 時, 則印出『Good-Bye!』, 否則略過而往下執行。

多重的 if 指令

我們可以用多重的 if 來做多重的條件限制:

著作權屬旗標公司，侵害必究・旗標 2003




```
if (a=='O')
  if (b=='K')          ←— 第一個 if 條件成立後才會被執行
     alert("You are OK!")   ←— 兩個 if 都成立後才執行
```

請注意! 前 2 行只不過是決定一個敘述是否被執行而已。必須兩個 if 的條件都成立, alert("You are OK!") 動作才會被執行。所以多重 if 可以看成是條件式的交集。當然了, 您也將這段程式改寫成這樣:

```
if (a=='O'&& b=='K')
 alert("You are OK!")   ←—— 兩個條件都成立後才執行
```

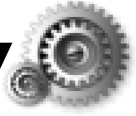
效果是一樣的。

if...else... 指令

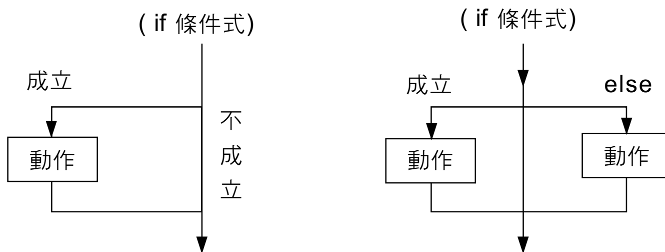
if 是對某一動作做選擇。如果我們想對兩種動作做選擇,則可以使用 if...else...:

```
if ( 條件式 )
  { 動作 1 }   /* 條件式成立時, 執行動作 1 */
else
  { 動作 2 }   /* 否則, 執行動作 2 */
```


著作權屬旗標公司，侵害必究・旗標 2003



這個敘述是說, 當條件式成立時, 則執行動作 1,然後略過 else 的部份 (動作 2), 接著往下執行。當條件式不成立時, 則略過 if 後的部份 (動作 1) 而執行 else 的動作 2, 然後再往下執行。也就是說, 動作 1 與動作 2 只會因條件式的眞假由二者選一來執行。



if... 與 if...else... 的差異

下面是一個 if...else... 的使用例子:

```
if (c=='q')
   alert('c 的值是q')
 else
   alert('c 的值不是q')
```

巢狀的 if...else...

if...else... 的動作部份可以是任何合法的敘述, 所以也可以是另一個 if...else...。也就是說我們可以在 if...else... 中再插入 if...else..., 或是說把 if 或 else 的動作部份以另一組 if...else... 來取代。這就是所謂 if...else... 的變形。首先我們來看這種情形:

著作權屬旗標公司，侵害必究・旗標 2003

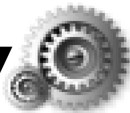
```
if (a > 0)
  if(b > 0)
    alert("a>0 and b>0")
  else
    alert("a>0 and b<=0")
else
  if(b > 0)
    alert("a<=0 and b>0")
  else
    alert("a<=0 and b<=0")
```

這種寫法是以 if...else... 來取代掉原來 if 或 else 的動作部份,通常稱為『巢狀的 if...else...』。這是最容易引起誤解的寫法, 我們要掌握的閱讀原則是『else 要與最接近且未配對的 if 配成一對』。這種寫法也可能產生變形, 比如 if 比 else 還多的情形, 但我們的原則還是以最接近的 if 來與 else 來配對。 if...else... 的關係會被區段 { } 中止, 所以上述的 if...else... 配對規則只限於同一區段內部。

```
if (a > 0)
  if(b > 0)
    alert("a>0 and b>0")
else
  alert("a<=0")
```

上述的寫法乍看之下, else 好像是與第一個 if 配對, 但事實上卻是與第二個 if 配對 (與最接近且「未配對」的 if 配對)。因此, 我們應加上一個 { } 以使 else 和第一個 if 配對:

著作權屬旗標公司，侵害必究・旗標 2003



```
if (a > 0)
  {
  if(b > 0)
    alert("a>0 and b>0")
  }
else
  alert("a<=0")
```

另外一種連續判斷式的 if...else... 是:

```
if (條件式1)
 { 動作 A }     /* 條件式 1 成立時 */
else if (條件式 2)
 { 動作 B }     /* 條件式 2 成立時 */
else if (條件式 3)
 { 動作 C }     /* 條件式 3 成立時 */
else
 { 動作 D }     /* 其它 */
```

這種寫法是以 if...else...來取代原來的 else 動作部份, 這是所謂過濾式或多重選擇式的寫法。以所得稅的計算爲例, 其稅率是依照年收入 (income) 的多寡而有所不同:

```
if (income > 5000000)
  tax = income * 0.20
else if (income > 1000000)
  tax = income * 0.15
else if (income > 300000)
  tax = income * 0.10
else if (income > 150000)
```


著作權屬旗標公司，侵害必究・旗標 2003

```
    tax = income * 0.05
  else
    tax = 0
```

注意事項

使用 if...else... 敘述有下列幾點要小心的:

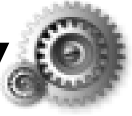
1. 若有多個 if else 比較時,這些 if else 可以寫成鋸齒狀的表示式，使程式可讀性提高, 這時最好每個 if 都能對應一個 else, 否則最好加上 { } 以免出錯。

```
  if(a > 0)
    {
   if(b > 0)
     if(c > 0)
       alert("a,b,c all > 0")
      else
       alert("a,b>0 and c<=0")
    }
  else
   alert("a <= 0")
```

2. else 會與最近的 if 配對, 除非我們以 { } 來加以區隔。下例中若不加 { }, 則 else 會與第二個 if 配對:

```
         if(n > 0)                    // 第一個 if 敘述
           ┌─{
 加上      │  if(a > b)               // 第二個 if 敘述
 大括弧    │     max = a;
           └─}
```


著作權屬旗標公司，侵害必究・旗標 2003



```
else                    // else 與誰配對？
  max = b;
```

3. 其實 if 的條件式不一定要是一個運算式,也可以是一個變數 (記得,單一變數也是合法的算式), 所以:

```
範例 1:  if (a)      // 若 a 為假(false) 則結果為假, 否則結果為真
    .....
範例 2:  if (!a)     // 若 a 為假(false) 則結果為真, 否則結果為假
    .....
```

4-3 迴圈控制 while 、 for 的結構變化

預先判斷式迴圈: while

while 為預先判斷式的迴圈。即先檢查條件是否為真 (true), 若為真, 則執行迴圈內的動作, 然後跳回條件式上再檢查,如此一直執行到條件不成立為止, 即條件為僞時才離開迴圈。預先判斷式迴圈的寫法如下:

```
while    (條件式) ── 先測試條件
         {動作}   ── 再做動作
```

假使條件式恆真或執行動作後仍不能使條件變為假, 比如:

```
while (true) { putchar(7) }
```

則這個迴圈就永遠出不來了, 有時甚至會碰到當機。寫程式時往往就會碰到這種情況, 至於如何避開這種情形呢? 只有靠多寫程式來磨鍊了。當然有時這種無限迴圈是我們故意製造的, 就不在此限了。

著作權屬旗標公司，侵害必究・旗標 2003

固定次數的迴圈: for

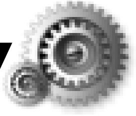
for 迴圈和 BASIC 的 FOR 指令類似,但其能力比 BASIC 的 FOR 更強,其寫法是:

```
for  (算式 1; 算式 2; 算式 3)
     { 執行動作 }
```

for 中的三個算式分別以 ';' 分號隔開, 彼此之間可以互相有關係或是根本沒有關係。其中:

- ❑ **算式 1**: 通常是起始值的設定, 此算式只有在第一次進入迴圈時才會被執行。
- ❑ **算式 2**: 通常是條件判斷式, 如果條件爲眞時, 則執行 {} 內的動作, 否則離開 for 的迴圈。
- ❑ **算式 3**: 通常是步進運算式, { } 內的動作執行完後, 必須再到這裡做此運算, 然後到算式 2 做判斷。

著作權屬旗標公司,侵害必究・旗標 2003



for 迴圈的使用頻率較 while 和 do-while 都還要來得高,雖然我們也可以用 if 和 while 來完成相同的動作, 可是還是數 for 最具有彈性了。 for 中的算式 1 、 2 或 3 均可省略。以下是兩個使用 for 的程式例:

```
int cnt
for(cnt=0; cnt<3; cnt++)   ←—— 要執行 3 次
        {
        document.write("error!")
        }
```

執行結果:

error!error!error!

```
var n
var m
for(n=1; n<=7; n++)
    {
    for(m=0; m<n; m++)
            { document.write("*") }        巢狀迴圈
    document.write("<BR>")
    }
```

執行結果:

```
*
**
***
****
*****
******
*******
```


著作權屬旗標公司，侵害必究・旗標 2003

各種迴圈的使用時機

❑ **while** ：無初始設定及步進運算的時候，如:

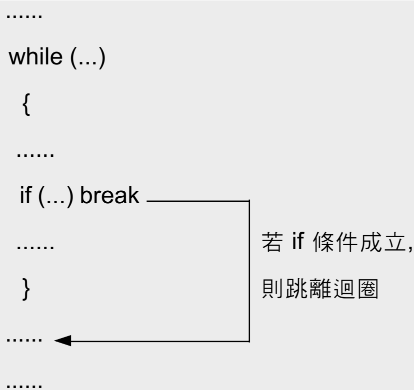
```
while (c==' ' || c=='\n') { }      //用來略過空格及 '\n'
```

❑ **for** ：有初始設定及步進運算時使用。

```
for(cnt=0; cnt<3; cnt++) { document.write("error!") }
```

迴圈控制中的 break

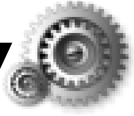
當 while 或 for 迴圈在執行中，若想立即跳出迴圈的話，有二種辦法可以做得到，一是使判斷的條件不成立，這樣自然就離開 Loop 了。另一個方法就是使用 break 指令,命令程式立即離開迴圈。通常 break 都和 if 連用,當某些條件成立後，就跳離開迴圈，break 只能跳出一層迴圈，若程式欲離開的地點被好幾個 {} 包圍住時，就得逐次 break，才能使程式離開。



例如：

```
var  KeyESC = 7
var  key    = 0
while(1)            //永遠的迴圈
```


著作權屬旗標公司，侵害必究・旗標 2003

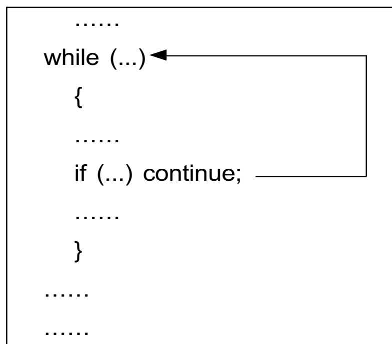


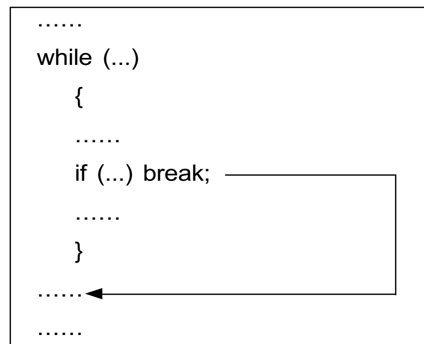
```
{ key++
document.write (key, '<BR>')
if (key == KeyESC) break;          // 用 break 跳出迴圈
}
document.write("Break Success!")
```

```
執行結果:
1
2
3
4
5
6
7
Break Success!
```

continue

當程式執行 while 或 for 時，若不想執行某次迴圈，則可以使用 continue 敘述來略過該次迴圈，使程式由下一次的迴圈開始。通常 continue 是用在迴圈剩餘部份非常複雜的情形下，為了使程式清晰明瞭，才使用的。continue 並不跳離迴圈，只是**忽略**掉了 continue 後面的敘述，而由下一次迴圈繼續執行，這是 continue 與 break 不同之處。





著作權屬旗標公司，侵害必究・旗標 2003

```
var i
var sum = 0
var sss = new Array(7)
sss[0]  = 1
sss[1]  = -2
sss[2]  = 3
sss[3]  = -4
sss[4]  = 5
sss[5]  = -6
sss[6]  = 7
for(i=0; i<7; i++)
        {
        if(sss[i] <= 0) continue     // 若負數則略過
        sum=sum+s[i]
        }
        document.write("SUM = ", sum)
```

執行結果:

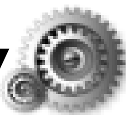
```
SUM = 16
```

上面這個程式讀入一個整數矩陣, 並將矩陣中的正整數相加起來,當 if 碰到負值時, 就使用 continue 敘述使負的值不加到 sum 中。

4-4 函式的使用

所謂的函式 (Function) 就是『敘述的集合』, 並且以一個函式名稱來代表此敘述集合。通常一個函式可以完成某項功能, 所以我們可以把函式想成是一種對資料的的操作方法, 它和算符在本質上是一樣的, 只不過是用不同的方式來表達而已:

著作權屬旗標公司,侵害必究・旗標 2003



操作法的比較	函式	算符
名稱 例如：	函式名稱 add	算符名稱 +
操作的資料 例如：	以參數方式傳入 add(3, 8)	放在算符的左右二邊 3 + 8
結果值 例如：	函式的傳回值 i=add(3, 5)-1;	最後一次運算的結果 i = (3+5) -1;
操作法的定義 例如：	使用者自定函式 add(a,b) { return a+b }	由 JavaScript 預先定義好的。

我們知道, 整個 JavaScript 程式就是由各個函式所構成的, 每個函式都是一個完整的單元, 由固定的入口進, 由簡單的出口出, 所以其內部的資料及運算過程可以做最佳的封裝。函式的入口即函式的開頭, 出口則是函式的結尾或 return 之處, 這種特性使得函式的行爲容易掌握, 所設計出來的程式就更強固耐用 (robust) 了。使用函式可以讓程式更具結構化、更能相互支援、也更容易除錯及維護。並可達到功能獨立、資訊隱藏的目的。

4-5 函式的定義、宣告與呼叫方式

函式的定義方式:

```
function (參數 1, 參數 2, ...)
{
    敘述的集合...     (函式內容)
    .....
}
```

例：

```
function max(a, b)
{
        return (a>b ? a : b );
}
```


著作權屬旗標公司，侵害必究・旗標 2003

函式的呼叫方式:

```
函式名稱 (參數 1, 參數 2, ...)
[註] 若函式有傳回值, 則可將此傳回值存入變數中, 或用來參與其他的運算。
```

例：

```
i = max(5, b)
j = i + max(i, j) * 3;
```

任何的函式在呼叫之前都必須定義過，否則會產生執行時的錯誤。我們一般都會將函式的定義部份放在 Web 文件的 <HEAD>...</HEAD> 標籤之間, 以確保往後 JavaScript 程式執行時, 所有會用到的函式均已事先定義過了。

在函式的定義或宣告中，參數列內的各參數是用來接收傳入資料的, 我們稱爲『形式參數』(Formal arguments), 而在呼叫函式時實際傳過去的參數, 則稱爲『實際參數』(Actual arguments)。

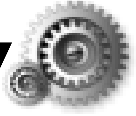
在定義函式時, 我們通常會爲『形式參數』取一些比較有義意的名稱來增加可讀性，例如:

```
PickUp (locate, string)
```

上式若寫成 PickUp(a, b), 則令人很難了解各參數的作用。

在呼叫函式時, JavaScript 會嚴格地檢查各實際參數及傳回值是否和定義的樣子一致 (Strong type checking), 若參數的數目不相符, 則立刻產生錯誤。

著作權屬旗標公司，侵害必究・旗標 2003



```
function max(a, b)
 {
 return (a>b ? a : b )
 }
ans = max(3, 5)        // ans = 5
ans = max(3, '4')      // ans = 4
ans = max("abc", 5)    // 錯誤: "abc" 無法轉為數值型別
ans = max(3)           // 錯誤: 參數數目不對, 可能會造成不可預期的錯誤。
```

函式的定義是各自獨立的，我們不能在函式定義的內部再定義函式。整個 JavaScript 程式就是由一個個的函式定義所組成的。當我們的程式呼叫別的函式時，會把控制權交給該函式，而被呼叫的函式也是依定義的內容去執行，一直到函式結束為止，再將控制權轉移回原呼叫之處。

函式的參數傳遞，可包括傳入的參數列和傳出的傳回值，我們稱之為『公用界面』(Public interface)。一個好的函式設計，應該只使用公用界面來與外界溝通，這樣做的好處是當函式需要修改時，只要公用的界面不變，就不會影響到其他的函式運作。

著作權屬旗標公司，侵害必究・旗標 2003

著作權屬旗標公司，侵害必究・旗標 2003

內建的物件與函式

5-1 eval() 函式

只要將整行運算式傳給 eval() 函式，它就能傳回計算結果。例如:

```
<script>
var mathString = '1 * 2 + 3 * 4 + 5 * (6 + 7)'
var sum =eval(mathString)
document.write(mathString)
document.write(' = ')
document.write(sum)
</script>
```

執行結果:

1 * 2 + 3 * 4 + 5 * (6 + 7) = 79

Ex0c-01.htm

其實 eval 函式的能爲尚不只如此，凡是所有的 JavaScript 敘述、指令，都可以使用 eval 函式來執行。例如:

```
<script>
var myString = 'document.write(" 生日快樂 ")'
eval(myString)
</script>
```

執行結果:

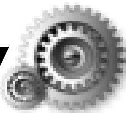
生日快樂

5-2 parseInt() 函式

JavaScript 內建的 parseInt() 函式，可以將一個字串轉換成整數數值。例如:

```
myString = '123'  //此時 myString 是個字串
```


著作權屬旗標公司，侵害必究・旗標 2003



則 parseInt(myString) 就會傳回 123 這個數值。由於 parseInt() 函式對小數點以下的數字一律捨棄，所以:

```
FloatString = '123.544'  // 此時 FloatString 是個字串
```

而 parseInt(myString) 還是傳回 123 這個數值。

當然了，並非所有的字串都能順利地轉換成一個整數數值，這時候 parseInt() 函式便會耍一點小聰明。例如:

```
myString = "31My Name is Fong"

var a = parseInt(myString)
```

則 parseFloat(myString) 傳給 a 的數值便是 31 。也就是說, 當 parseFloat() 函式遇到無法轉換成數字的英文字母，就會在此停住。將字母之前的字串轉換成數值。要是字串的第一個字元就無法轉換成數字, 則傳回 "NaN" (表示 Not a Number)。

5-3 parseFloat() 函式

JavaScript 內建的 parseFloat() 函式, 可以將一個字串轉換成浮點數值。例如:

```
myString = '123.45'  // 此時 myString 是個字串
```

則 parseFloat(myString) 就會傳回 123.45 這個數值。

所謂的浮點數值, 是指像 1.234 這樣小數點後面還有數字的數值。而整數數值, 當然就是不含小數點, 以及小數點後面的數字! 例如 25。

當然了，並非所有的字串都能順利地轉換成一個浮點數值，這時候 parseFloat() 函式便會耍一點小聰明。例如:

著作權屬旗標公司，侵害必究・旗標 2003

```
myString = "3.14159My Name is Fong"

var a = parseFloat(myString)
```

則 parseFloat(myString) 傳給 a 的數值便是 3.14159 。也就是說，當 parseFloat() 函式遇到無法轉換成數字的英文字母，就會在此停住。將字母之前的字串轉換成數值。要是字串的第一個字元就無法轉換成數字，則傳回 "NaN" (表示 Not a Number)。

5-4 Date() 日期物件

JavaScript 內建了幾個基本的物件，其中一個便是 Date 物件。Date 物件讓我們在網頁內能掌控時間資訊並加以控制。

建立 Date 物件

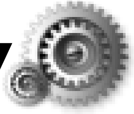
當我們要使用 Date 物件，必須先使用 new 指令建立一個 Date 物件，例如：

```
today = new Date();
```

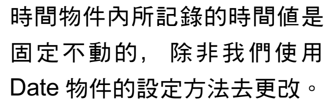
如此即建立了一個名爲 today 的 Date 物件。當我們建立 Date 物件時，若無給予任何參數，則所建立的 Date 物件內會記錄著目前的時間。如果我們在建立 Date 物件時給予時間參數，則這個 Date 物件就會依參數值來調整其內部時間值：

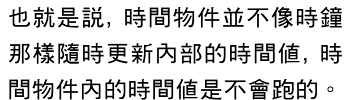
Date() 的三種用法	範 例
物件名稱 =new Date()	today = new Date()
物件名稱 =new Date(" 月 日,年 時:分:秒 ")	count = new Date("May 24,1996")
物件名稱 =new Date(年,月,日,時,分,秒)	NewDay = new Date(1995,11,20)

著作權屬旗標公司，侵害必究・旗標 2003



如果我們使用後兩種格式輸入時間, 卻忽略掉時、分、秒的設定 (如範例所示), 則 JavaScript 會視作 00:00:00 。





Date 物件的使用方法

建立好 Date 物件後, 接著讓我們來看幾個 Date 物件的方法 (Method):

取得時間的方法		設定時間的方法	
取得 " 年份 "	getYear()	設定 " 年份 "	setYear()
取得 " 月份 "	getMonth()	設定 " 月份 "	setMonth()
取得 " 日期 "	getDate()	設定 " 日期 "	setDate()
取得 " 星期 "	getDay()	----------	--------
取得 " 小時 "	getHours()	設定 " 小時 "	setHours()
取得 " 分鐘 "	getMinutes()	設定 " 分鐘 "	setMinutes()
取得 " 秒鐘 "	getSeconds()	設定 " 秒鐘 "	setSeconds()

著作權屬旗標公司，侵害必究・旗標 2003

使用取得時間的方法，我們可以從 Date 物件上取得時間值；而使用設定時間的方法，則可爲 Date 物件設定時間值。例如:

```
today      = new Date()
Hours      = today.getHours()
Minutes    = today.getMinutes()
Seconds    = today.getSeconds()
```

便可以讓我們以 Hours 、 Minutes 、 Seconds 這三個變數取得現在時間的 " 時 " 、 " 分 " 、 " 秒 " 資訊。而下面這個例子:

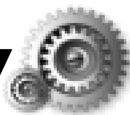
```
today = new Date()
today.setHours(14)
today.setMinutes(30)
today.setSeconds(20)
```

則讓我們以將 today 這個物件內部所記錄的時間改設定成下午 2:30:20。

時間的格式

儘管這樣的使用方法十分地方便，但是有些傳回值與設定值還是值得我們注意一下:

著作權屬旗標公司，侵害必究・旗標 2003



使用方法	意義	數值	數值意義
Year()	年	1970～nnnn	以四位數的數字表現實際年代
Month()	月	0 ～ 11	0=Jan(一) ～ 11=Dec(十二)
Date()	日	1 ～ 31	
Day()	週	0 ～ 6	0=Sun(日) ～ 6=Dat(六)
Hours()	時	0 ～ 23	
Minutes()	分	0 ～ 59	
Seconds()	秒	0 ～ 59	

在月份值上，我們必須再加上 1，才與日常習慣用法相吻合。

另外還有一點，雖然我們可以使用 getDay() 傳回的星期**幾**的資訊，但是卻無法使用 setDay() 來設定星期**幾**。因爲只要『年、月、日』 確定後，電腦便能自動算出星期**幾**，無須我們操心。

輸出時間的簡便方法

經過前面對 Date 物件的介紹，我們現學現賣，立刻寫一個顯示時間的程式：

```
<script>
today = new Date()
Year = today.getYear()
Month = today.getMonth() + 1    ←── 月份數記得要加 1 !
Date = today.getDate()
Hour = today.getHours()
Min = today.getMinutes()
Sec = today.getSeconds()
document.write(Year,"/",Month,"/",Date," ",Hour,":",Min,":",Sec)
</script>
```

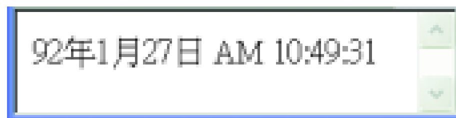
Ex0c-02.htm

著作權屬旗標公司，侵害必究・旗標 2003

然而要是您使用 Date 物件只是要列印出時間，其實還有更容易的方法：

```
<script>
today = new Date()
  document.write(today.toLocaleString())
</script>
```

Ex0c-03.htm



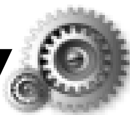
顯示出日期與時間

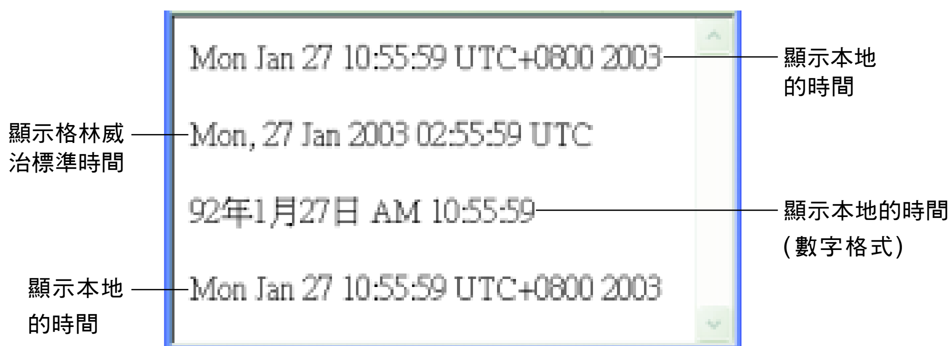
toLocaleString() 會直接將 Date 物件內含的時間值轉換成表示時間的字串，直接使用 toLocaleString() 有時候也蠻方便的。當然了，將時間值轉換成表示時間字串的方法還有 toString()、 toGMTString() 與 toLocaleString():

```
<script>
today = new Date()
document.write(today.toString())
document.write('<P>')
document.write(today.toGMTString())
document.write('<P>')
document.write(today.toLocaleString())
document.write('<P>')
document.write(today)
</script>
```

Ex0c-04.htm

著作權屬旗標公司，侵害必究．旗標 2003





從這個例子我們可以看出 toGMTString() 會傳回格林威治標準時間, 而 toString() 則會傳回本地的標準時間。toLocaleString() 也會傳回本地時間, 只是改採用數字格式來表示。至於如果我們直接輸出 today 這個 Date 物件, 電腦會自動以 toString() 方法取得傳回值, 顯示出與 today.toString() 相同的值。

5-5 String 字串物件

在眾多 JavaScript 內建的物件中, String 字串物件算是使用得最平常的了, 只是我們常省略了定義的步驟而已:

```
document.write('歡迎來到風信子首頁')
```

事實上改寫成這樣結果是一樣的:

```
var MyString = '歡迎來到風信子首頁' //定義 MyString 這個字串物件
document.write(MyString)
```


著作權屬旗標公司，侵害必究・旗標 2003

String 物件的 length 屬性

String 物件的 length 屬性, 會顯示出字串長度。例如當我們定義一個字串物件:**var MyName = " 風信子 "** 則 **documrnt.write(MyName.length)** 就會顯示出 6 。當然了, String 物件的 length 屬性是唯讀的, 我們只能由它得知字串的長度, 而無法直接設定其值。

String 物件的方法

String 物件的操作方法很多, 在此我們先介紹有關字串運算的各種方法:

方 法	意 義
indexOf(' 字元 ')	由字串首算起, 找出字串中的某字元的位置
lastIndexOf(' 字元 ')	由字串尾算起, 找出字串中的某字元的位置
toLowerCase()	將字串中的英文字母全轉成小寫
toUpperCase()	將字串中的英文字母全轉成大寫
charAt(索引值)	取出字串中的第幾個字元
substring(起始,終點)	取出字串中的某部份 (一或多個字元)

字母大小寫轉換

例如當我們定義了一個 Happy 字串物件:

```
var Happy = "Happy Birthday and Merry Christmas !"
```

Happy.toLowerCase() 會傳回 "happy birthday and merry christmas !" 而 Happy.toUpperCase() 則會傳回 "HAPPY BIRTHDAY AND MERRY CHRIST-MAS !"

著作權屬旗標公司，侵害必究・旗標 2003

字元的搜尋與讀取

例如當我們定義了一個 myURL 字串物件:

```
var myURL = "http://www.flag.com.tw"
            ↑                     ↑
            位置0                 位置21
```

則 myURL.indexOf('w') 會傳回 7, 表示從 myURL 字串頭開始找起, 在第 7 個 (由 0 開始數起) 字元上找到 'w' 字元。

而 myURL.lastIndexOf('w') 會傳回 21, 表示從 myURL 字串尾開始找起, 在 myURL 的第 21 個字元上找到 'w' 字元。

注意! JavaScript 對字串
是從 0 開始算起的呦!

例如當我們定義了一個 myWord 字串物件:

```
var myWord = 'To be, or not to be.'
```

則 myWord.charAt(0) = 'To be, or not to be.' 就是 'T' !

如果我們定義一個 MyString 的字串物件, 則它的第一個字元就是 MyString.charAt(0), 而最後一個字元則是 MyString.charAt(MyString.length - 1)

著作權屬旗標公司，侵害必究・旗標 2003

取出子字串

例如當我們定義了一個 myAddress 字串物件:

```
var myAddress = '台北市杭州南路一段 15-1 號 11 樓'
```

則 myAddress.substring(6,14) 的值就是 '杭州南路' (中文字爲雙位元)

只要我們的**終點**值大於**起始**值,例如前例中的 myAddress.substring(6,14), 便能夠正確地傳回 '杭州南路'。要是**終點**值等於或小於**起始**值, 則會傳回一個空字串。

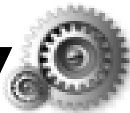
字串的運算

字串物件除了本身的使用方法外, 還能參與加法運算:

```
myName1 = "風信子"
myName2 = "黃金葛"
myName = myName1 + myName2
```

在這個例子裡 myName 的值就等於 myName1 加上 myName2, 也就是 "風信子黃金葛"。

著作權屬旗標公司，侵害必究・旗標 2003



使用 String 物件產生 HTML 標籤的方法

String 物件另外有一組使用方法是用來產生 HTML 標籤的, 例如:

```
var myString  = "風信子"
var newString = myString.anchor("NewStart")
```

則 newString 的值就會變成『<A NAME="NewStart">風信子</A>』字串。下面列出這類使用方法的輸出結果:

方法	輸出結果
myString.anchor(x)	<A NAME=x>風信子</A>
myString.link(x)	<A HREF=x>風信子</A>
myString.fontcolor(x)	<FONT COLOR="x">風信子</FONT>
myString.fontsize(x)	<FONTSIZE="x">風信子</FONTSIZE>
myString.big	<BIG>風信子</BIG>
myString.blink	<BLINK>風信子</BLINK>
myString.italics	<I>風信子</I>
myString.bold	<B>風信子</B>
myString.fixed	<TT>風信子</TT>
myString.sub	_{風信子}
myString.sup	^{風信子}
myString.small	<SMALL>風信子</SMALL>
myString.strike	<STRIKE>風信子</STRIKE>

著作權屬旗標公司，侵害必究・旗標 2003

5-6 Math() 數學物件

JavaScript 本身內建的物件, 除了 Date (時間) 物件與 String (字串) 物件外, 還有一個 Math (數學) 物件。 Math 物件是當瀏覽器啓動後便已經建立完成的, 這與 Date 、 String 物件不同。當我們要使用 Date 、 String 物件前, 我們必須先建立該物件:

```
var today  = new Date()
var MyName = "風信子"
```

然而 Math 物件則不用, 當我們想讓數值變數 myPi 等於**圓周率**時, 只須直接讀出 Math 物件的 PI 屬性即可:

```
myPi = Math.PI      //myPi = 3.141592653589
```

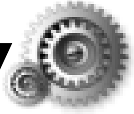
當然了, 也由於 Math 物件在瀏覽器啓動後便已經建立完成了, 所以整個系統就只有一個 Math 物件, 名字就叫 Math, 我們不能也毋須再建立第二個 Math 物件。

Math 物件的屬性

Math 物件提供了一些相當實用的屬性 (都是唯讀的), 這些屬性都是數學運算上用得到的『常數』。

屬性	説明	近似值
E	Euler 常數, 爲自然對數的底數	2.718281828459
LN2	2 的自然對數	0.693147180559
LN10	10 的自然對數	2.302585092994
LOG2E	以 2 爲底數, E 爲基數的 Log 值	1.442695040888
LOG10E	以 10 爲底數, E 爲基數的 Log 值	0.434294481903
PI	圓周率	3.141592653589
SQRT1_2	二分之一的平方根	0.707106781186
SQRT2	二的平方根	1.414213562373

著作權屬旗標公司，侵害必究・旗標 2003



就讓我們練習使用這些數學常數吧:

```
<script>
document.write("Math.E="          , Math.E          , "<BR>")
document.write("Math.LN2="        , Math.LN2        , "<BR>")
document.write("Math.LN10="       , Math.LN10       , "<BR>")
document.write("Math.LOG2E="      , Math.LOG2E      , "<BR>")
document.write("Math.LOG10E="     , Math.LOG10E     , "<BR>")
document.write("Math.PI="         , Math.PI         , "<BR>")
document.write("Math.SQRT1_2="    , Math.SQRT1_2    , "<BR>")
document.write("Math.SQRT2="      , Math.SQRT2      , "<BR>")
</script>
```

Ex0c-05.htm.htm

```
Math.E=2.718281828459045
Math.LN2=0.6931471805599453
Math.LN10=2.302585092994046
Math.LOG2E=1.4426950408889633
Math.LOG10E=0.4342944819032518
Math.PI=3.141592653589793
Math.SQRT1_2=0.7071067811865476
Math.SQRT2=1.4142135623730951
```


著作權屬旗標公司，侵害必究・旗標 2003

Math 物件的使用方法

Math 物件所提供的使用方法，都是計算用的數學函式:

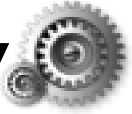
一般用途函數	
abs(x)	求 x 的絕對值
round(x)	求 x 的四捨五入值
ceil(x)	求大於或等於 x 的最小整數
floor(x)	求小於或等於 x 的最大整數
sqrt(x)	求 x 的平方根
pow(x, y)	求 x 的 y 次方值
exp(x)	以 E 的 x 次方值，為 log(x) 的反函數
max(x, y)	求出 x 與 y 中，較大的值
min(x, y)	求出 x 與 y 中，較小的值
random()	傳回一個介於 0 與 1 之間的亂數
三角函數與對數	
sin(x)	以度為單位，求 x 的 sin 函數
cos(x)	以度為單位，求 x 的 cos 函數
tan(x)	以度為單位，求 x 的 tan 函數
asin(x)	以度為單位，求 x 的 arc_sin 函數
acos(x)	以度為單位，求 x 的 arc_cos 函數
atan(x)	以度為單位，求 x 的 arc_tan 函數
log(x)	求 x 的自然對數， 為 exp(x) 的反函數

對於想要撰寫 JavaScript 遊戲的讀者來說，也許就會常用到 random() 和 floor() 方法來取得亂數值。舉例來說，想讓 myChar 讀入一個介於 1 到 26 之間的亂數值，可使用下面的運算敘述:

```
myChar = Math.floor(Math.random() * 26 + 1)
```

有了 Math 物件，大大地擴增 JavaScript 數學運算的能力。

著作權屬旗標公司，侵害必究・旗標 2003



with 敘述

當我們在 JavaScript 程式中用到 Math 物件時, 常常都是在一些複雜的數學運算上:

```
theta  = Math.PI / 6
result = 4 * Math.cos(theta) + 3 * Math.pow (Math.sin(theta), 3)
```

短短的一段運算式子就有四個『Math.』字樣, 不但煩人, 而且容易出錯。這時候我們就可以使用 with 關鍵字, 將敘述簡化:

```
with  (math)
 {
 theta  = PI / 6
 result = 4 * cos(theta) + 3 * pow (sin(theta), 3)
 }
```

如此式子更加簡潔明瞭了。

5-7 Array 陣列物件

我們在撰寫程式時, 每當需要將一個資料輸入到記憶體中, 便需要使用一個變數來儲存, 因此, 若有 100 個資料即需要定義 100 個變數來儲存。遇到大量 (或數量變動不定) 的資料時, 例如輸入 100 位學生的姓名資料, 利用簡單變數來處理, 便成了一項工程浩大的麻煩事。

這時候我們可以使用 Array 陣列來幫我們的忙, 只要建立一個具有 100 個元素的 student 陣列, 便可輕鬆地掌握這 100 個學生的資料:

著作權屬旗標公司，侵害必究．旗標 2003

```
var student = new Array(100)
 student[0] = "張民人"
 student[1] = "風信子"
 student[2] = "練良聖"
  ......
  ......
 student[98]= "王小明"
 student[99]= "陳大丙"
```

使用 Array 陣列物件後, 對於資料的存取可也有更大的彈性, 例如當您需要存取第 10 位學生的資料, 則可由 student[9] 取得(因為 student 代表第 1 位學生)。此外, 我們甚至可使用其它的變數 (例如 i) 來存取不特定學生 (例如: student[i]) 的資料呢!

著作權屬旗標公司，侵害必究・旗標 2003

網頁內建的物件

JavaScript 可用的物件一種是由瀏覽器內部提供, 諸如 String (字串物件)、 Date (日期物件)、 Math (數學物件) 等。另一種 JavaScript 可用的物件是由我們的網頁文件所提供, 請看以下的範例:

6-1 存在於網頁中的物件

舉例來說, 當我們寫了一個簡單的網頁:

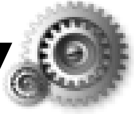
```
<H3>Star Trek 電影介紹</H3>
<IMG SRC="StarTrek1.gif    " NAME="TOS">
<A  HREF="StarTrek1.htm    " NAME="MOA">拯救未來</A><P>
<A  HREF="StarTrek2.htm    " NAME="MOB">終極先鋒</A><P>
<A  HREF="StarTrek4.htm    " NAME="MOC">日換星移</A><P>
<IMG SRC="StarTrek2.gif    " NAME="TNG">
```

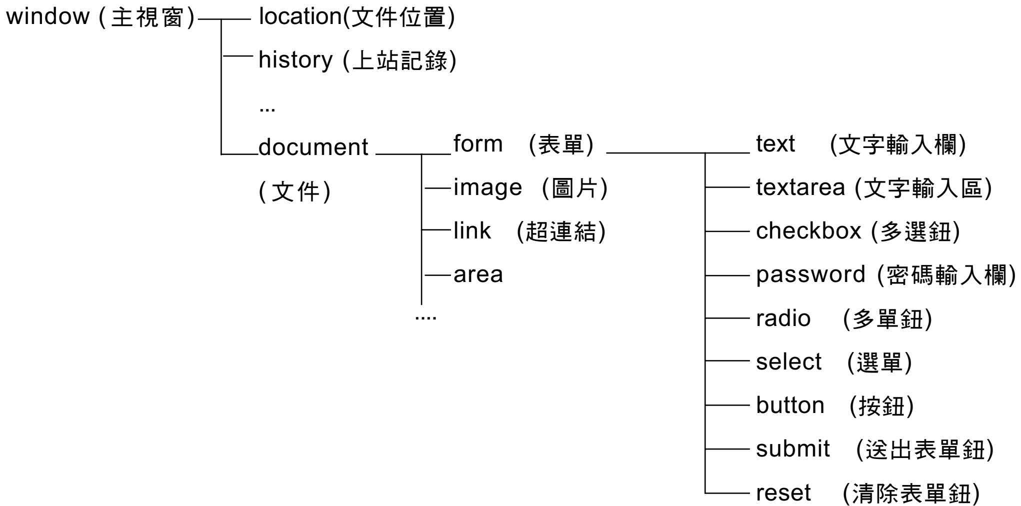
則這個簡單的文件中便擁有兩個 image (圖片) 物件與三個 link (超連結) 物件。

6-2 各物件之間的階層關係

當瀏覽器顯示出一個網頁時, 會同時產生許多許多的物件, 這些物件之間都具有階層性 (包含) 的關係, 例如最上層的是 window (主視窗) 物件; window 物件內又含有 document (文件) 物件、 location (文件位置) 物件...; 而 document 物件內則又有若干的 form (表單) 物件、 image (圖片) 物件與 link (超連結) 物件等等。其架構簡示如下:

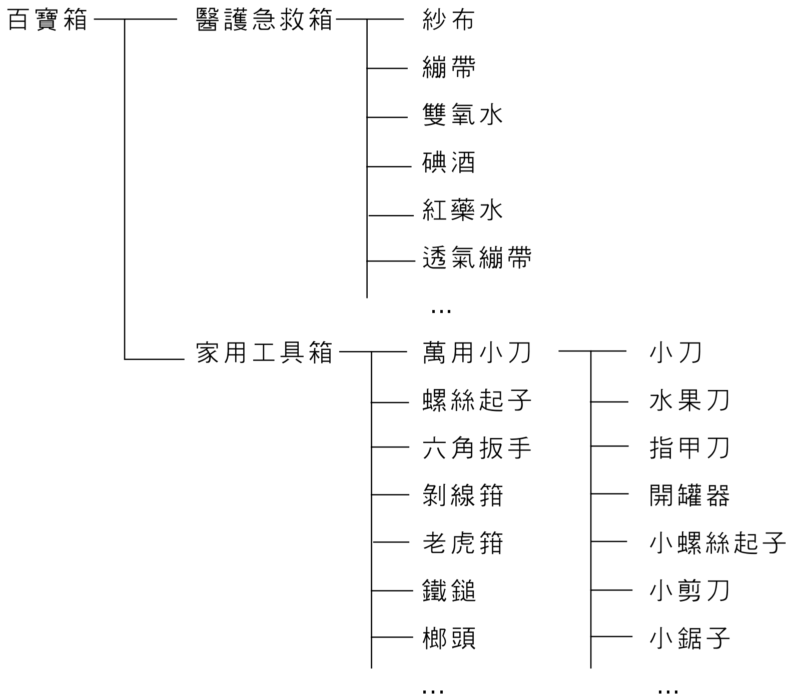
著作權屬旗標公司，侵害必究・旗標 2003





物件中的物件?

想想日常生活上的例子,一個家庭用的百寶箱裡,也放滿了各種實用的物件:



著作權屬旗標公司,侵害必究・旗標 2003

以 NAME 屬性指定物件

以接下來的網頁爲例，我們便可以使用物件的名字 (以 NAME 屬性所指定的) 來稱呼它。以下程式指定 2 張圖片與 2 個表單物件的名字：

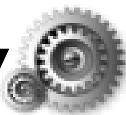
```
<IMG SRC="StarTrek1.gif"  NAME="TOS">

<FORM METHOD=GET ACTION="GetCode.asp"  NAME="MOA">
...
</FORM>

<IMG SRC="StarTrek2.gif"  NAME="TNG">

<FORM METHOD=GET ACTION="chkForm.asp"  NAME="MOB">
...
</FORM>
```

以上程式即可用以下方式來指定物件：

- ☐ document.TOS 代表圖片 StarTrek1.gif
- ☐ document.TNG 代表圖片 StarTrek2.gif
- ☐ document.MOA 代表第 1 個表單
- ☐ document.MOB 代表第 2 個表單

在 JavaScript 程式內是會分辨字母的大小寫的, 不可打錯。

著作權屬旗標公司，侵害必究・旗標 2003



以物件陣列來指定物件

除了直接透過 document 物件來指定網頁衍生物件外，我們還可以依其物件類別，透過 document 物件所附屬的子物件陣列來指定這些物件:

document.images["TOS"]	第一張圖片
document.images["TNG"]	第二張圖片
document.forms["MOA"]	第一個表單
document.forms["MOB"]	第二個表單

以物件陣列內的順序來指定物件

要是我們剛剛在撰寫 Web Page 時忘記爲他們命名，還是可以使用物件陣列 (編號) 來稱呼它們:

document.images[0]	第一張圖片
document.images[1]	第二張圖片
document.links[0]	第一個超連結
document.links[1]	第二個超連結

物件的陣列與編號

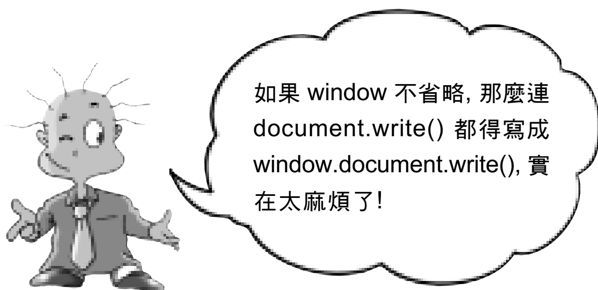
陣列是由多個性質相同的物件組合而成，並依加入的先後順序予以編號。瀏覽器在顯示文件時，會將文件中讀到的物件加入各相關陣列中。

例如讀到第一個 image 物件時，便加到 **document** 的 **images** 陣列中，而成爲 document.images[0]，其餘以此類推。

著作權屬旗標公司，侵害必究・旗標 2003

6-3 物件的方法 (Method)

在前面我們所學到的 write(), 便是 document 物件使用方法, 所以寫成了 document.write()。而 alert() 則是 window 物件的使用方法, 可以寫成 window.alert(), 只是因爲網頁內的所有物件都根植於 window 物件, 所以就乾脆省略寫成 alert() 即可。



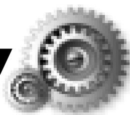
6-4 物件的屬性 (Property)

每一種物件都會有一到多個屬性, 就像電鍋有顏色、形狀、容量等屬性一樣。以一個簡單的網頁文件而言, 其 document 物件至少都會有下列的屬性:

document.title	文件的標題
document.fgColor	文件的前景顏色
document.bgColor	文件的背景顏色

改變 document.fgColor 、 document.bgColor 屬性值, 便能改變文件的前景、背景顏色。但 document.title 屬性值卻是唯讀的, 我們只能藉由它取得文件的標題, 但不能改變它的值, 否則就會產生錯誤。

著作權屬旗標公司，侵害必究・旗標 2003



另外，任何一個 window 物件也都會有下列的屬性:

```
windows.status          顯示在狀態列上的訊息
windows.defaultStatus 顯示在狀態列上的訊息 (當瀏覽器無所事事時)
```

這兩個屬性值可以為我們所改變，讓狀態列顯示出我們想要告訴使用者的訊息。

6-5 物件的驅動事件 (Event)

除了將 JavaScript 敘述直接嵌入 HTML 檔案各處外，我們還可以用網頁內建物件的驅動事件來執行 JavaScript 敘述，以一個簡單的 HTML 檔為例:

```
<title>我的範例</title>
<a href="profile.htm">
歡迎光臨風信子個人小檔案!
</a>
```

在這個 HTML 檔裡，有一組 <A>...</A> 的超連結標籤，讓我們在這個超連結 <A> 標籤內加上 onMouseOver 事件驅動屬性，如下所示:

```
<title>我的範例</title>
<a href="profile.htm" onMouseOver="alert(' 入內請勿吸菸!')">
歡迎光臨風信子個人小檔案!
</a>
```

則在網頁上，只要您讓游標經過這個超連結，電腦便會立刻開啓一個警告訊息:

著作權屬旗標公司，侵害必究・旗標 2003

按下**確定**鈕即可關閉這個交談窗

一個 HTML 的標籤通常看起來是這個樣子的:

```
<HTML 標籤 HTML 屬性... HTML 屬性...>
```

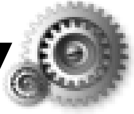
而將其加入事件屬性後，則變成下面這個樣子:

```
<HTML 標籤 HTML 屬性... HTML 屬性... 事件驅動屬性 ="JavaScript 敘述 ">
```

下面列出可以啓動 Javascript 程式的事件:

事件的名稱	事件驅動屬性	事件啓動的時機
mouseOver	onMouseOver	當滑鼠游標經過這個物件時
mouseOut	onMouseOut	當滑鼠游標移離這個物件時
abort	onAbort	當使用者停止 (abort) 載入影像時
load	onLoad	當使用者載入這頁網頁時
unload	onUnload	當使用者結束這頁網頁時
error	onError	當使用者載入這頁文件 (或影像) 發生錯誤時
focus	onFocus	當使用者將游標滑鼠移入表單元件時
blur	onBlur	當使用者將游標滑鼠移開表單元件時
change	onChange	當使用者改變了輸入欄位內的資料時
select	onSelect	當使用者選取了表單輸入元件時
submit	onSubmit	當使用者送出 (submit) 表單時
reset	onReset	當使用者重設 (reset) 表單時
click	onClick	當滑鼠點選這個物件時

著作權屬旗標公司，侵害必究・旗標 2003



由於這些事件驅動屬性是包含在 HTML 標籤內的，所以不分大小寫，因此 "onClick" 、 "onclick" 或 "OnClick" 的意思都是一樣的。下面列出所有事件與其可驅動物件的對照表：

驅動事件名稱	可驅動的物件名稱					
onMouseOver	area	link				
onMouseOut	area	link				
onAbort	image					
onLoad	image	window				
onUnload		window				
onError	image	window				
onFocus		window	select	text	textarea	
onBlur		window	select	text	textarea	
onChange			select	text	textarea	
onSelect				text	textarea	
onSubmit	form					
onReset	form					
onClick	link	button	checkbox	radio	reset	submit

也就是說每種物件，都限定只有幾種驅動事件能使用：

物件名稱	可以設定的驅動事件				
window	onError	onFocus	onBlur	onLoad	onUnload
Image	onError	onLoad			onAbort
Area	onMouseOut		onMouseOver		
Link	onMouseOut		onMouseOver		onClick
form	onSubmit	onReset			
select	onChange	onFocus	onBlur		
text	onChange	onFocus	onBlur	onSelect	
textarea	onChange	onFocus	onBlur	onSelect	
button					onClick
checkbox					onClick
radio					onClick
reset					onClick
submit					onClick

著作權屬旗標公司，侵害必究・旗標 2003

6-6 window 視窗物件

物件介紹

window 視窗物件是所有網頁文件中最頂層的物件。要開啓一個新的視窗 (也就是開啓一個新的瀏覽器視窗) ,我們可以使用 window.open 方法, 如下所示:

```
windowVar = window.open("URL", "windowName" [,"windowFeatures"])
```

windowVar 是新視窗的物件名稱, windowName 是新視窗的視窗名稱。在 JavaScript 內, 我們可以使用兩種名稱的其中一種來指定該視窗; 但由於 HTML 標籤 " 看 " 不到視窗 " 物件 ", 所以無法使用視窗的物件名, 而只能以**視窗名**來指定某個分割視窗, 例如:

```
<A HREF="URL" TARGET="視窗名">
```

附屬子物件

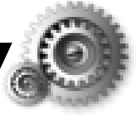
window 視窗物件之下, 可能包含下列三種物件:

```
document 文件物件
history 紀錄物件
location 位置物件
```

事件

window 視窗物件可使用的驅動物件有:

著作權屬旗標公司，侵害必究・旗標 2003



```
onLoad
onUnload
onBlur
onError
onFocus
```

要使用這些事件, 我們必須在網頁的 <BODY> 或 <FRAMESET> 標籤上設定:

```
<BODY
    ...
    [onLoad="handlerText"]
    [onUnload="handlerText"]
    [onBlur="handlerText"]
    [onError="handlerText"]
    [onFocus="handlerText"]>

</BODY>
```

```
<FRAMESET
    ROWS="rowHeightList"
    COLS="columnWidthList"
    [onLoad="handlerText"]
    [onUnload="handlerText"]
    [onBlur="handlerText"]
    [onError="handlerText"]
    [onFocus="handlerText"]>
    [<FRAME SRC="locationOrURL" NAME="frameName">]
</FRAMESET>
```


著作權屬旗標公司，侵害必究・旗標 2003

方法

window 視窗物件可使用的操作方法有:

alert(x)	顯示一個警告視窗，視窗內填入 x 訊息,和一個**確定**鈕。
confirm()	顯示一個確定視窗，視窗內填入 x 訊息、一個**確定**鈕和**取消**鈕。
prompt()	顯示一個輸入視窗，視窗內填入 x 訊息，並供使用者輸入資料。
open()	開啟一個新的瀏覽器視窗。
close()	關閉一個瀏覽器視窗。
setTimeout(z,x)	設定電腦經過 x 毫秒後，執行 z 敘述
clearTimeout()	取消某個 setTimeout() 的設定

由於 window 物件是最頂層的物件，所以我們省略物件名稱，例如 window.alert(" 歡迎光臨 ") 我們就省略寫成 alert(" 歡迎光臨 ")。

屬性

window 物件可使用的屬性有:

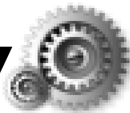
defaultStatus	顯示在視窗訊息列上的訊息 (當瀏覽器無所事事時)
status	顯示在視窗訊息列上的訊息
length	顯示電腦上開啟的瀏覽器視窗數
name	該視窗的**視窗名稱**

6-7 document 文件物件

附屬子物件

document 物件之下，可能包含下列物件:

著作權屬旗標公司，侵害必究・旗標 2003



history	記錄 history（上站記錄）的物件
anchor	文件內含的 anchor 物件
image	文件內含的 image 物件
area	文件內含的 area 物件
link	文件內含的 link 物件
form	文件內含的 form 物件

屬性

document 物件可使用的屬性有:

forms	該文件內所包含的 form 物件陣列
anchors	該文件內所包含的 anchor 物件陣列
links	該文件內所包含的 link 物件陣列
lastModified	顯示文件最後一次修訂的日期
bgColor	顯示文件的 BGCOLOR 屬性值
fgColor	顯示文件的 TEXT 屬性值
alinkColor	顯示文件的 ALINK 屬性值
linkColor	顯示文件的 LINK 屬性值
vlinkColor	顯示文件的 VLINK 屬性值
referrer	顯示呼叫本文件的上份文件的 URL
title	顯示文件的 TITLE
URL	顯示文件的 URL

方法

document 物件可使用的操作方法有:

document.open()	開啓一個文件區段
document.close()	關閉一個文件區段
document.write(x...)	輸出 x... 訊息到文件區段
document.writeln(x...)	輸出 x... 訊息到文件區段（自動換行）

著作權屬旗標公司，侵害必究・旗標 2003

6-8 form 表單物件

屬性

form 物件之下，可使用的屬性：

length	顯示該表單物件下的所有輸入元件的總數
elements	表示該表單物件下的所有輸入元件陣列
encoding	顯示該表單物件的 ENCTYPE 屬性值
action	顯示該表單物件的 ACTION 屬性值
method	顯示該表單物件的 METHOD 屬性值
target	顯示該表單物件的 TARGET 屬性值

方法

form 物件之下，可使用的操作方法只有一種：submit() 。如果我們有一個 myForm 輸入表單，則 myForm.submit() 方法就能將表單資料送出。

事件

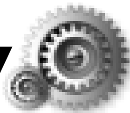
form 物件之下，可使用的驅動事件：

```
onSubmit
onReset
```

附屬子物件

form 物件之下，可能包含下列輸入物件：

著作權屬旗標公司，侵害必究・旗標 2003



button	使用者自訂的按鈕元件
hidden	隱藏式輸入元件
text	文字輸入欄
password	密碼輸入欄
textarea	多行式文字輸入欄
checkbox	單選鈕
radio	多選鈕
select	列表選擇元件
reset	清除表單鈕
submit	送出表單鈕

著作權屬旗標公司，侵害必究・旗標 2003

著作權屬旗標公司，侵害必究・旗標 2003