附錄 A

給 C 語言家族程式設計師的 Javascript 教學

Javascript的正式名稱是ECMAScript，歐洲電腦製造商協會（European Computer Manufacturers Association, ECMA）主導了ECMAScript的定義及發展。在撰寫本書的同時，ECMAScript現行的版本是ECMA-262版本5.1，你可以在http://ecmascript.org這個網站上找到這個版本的規格書。

Javascript或許是這個世界上最容易取得的程式語言，因為在最新的個人電腦、智慧型手機及平板電腦的瀏覽器上，都配有Javascript。在本附錄中，我們將會看到，微軟在Internet Explorer及Windows市集應用程式上所實作的Javascript成果。這個附錄的內容並無法取代Javascript在這15年內的所有的演變、標準化、文件化及社群討論的內容。我強烈地建議你一定要看過這個附錄的內容，來引導您進入狂放奔騰的Javascript世界！[1]

1 由Marijn Haverbeke所撰寫的Eloquent Javascript這本書是另一個不錯的參考資料。這本書由No Starch Press公司所出版。在http://eloquentjavascript.net/這個網站上有互動式的版本。另外，Underscore函式庫文件（http://documentcloud.github.com/underscore/docs/underscore.html，短網址：http://tinysells.com/281）也是非常棒的參考資料

Hello, World

Javascript的語法基礎是來自於C語言（例如C++、C#、Objective-C，還有Java），不過比較鬆散一些。這些C語言家族的成員，在區塊架構、以堆疊為基礎的函數、陳述的語法及基本的運算子（例如，`if`指令、數學表示式等等）都是相同的。不過就算你已經熟悉了C語言的標準，但是在使用Javascript的詞彙時，也有可能會出錯。所以就讓我們從頭開始吧！

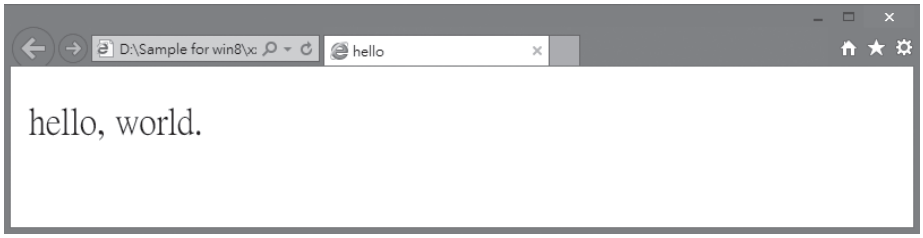
為了要執行Javascript，我們需要一個執行環境（host），而最容易取得的執行環境就是瀏覽器了，只要寫些簡單的HTML，就可以讓Javascript在瀏覽器中執行：

```
<!DOCTYPE html>
<!-- default.html -->
<html>
<head><title>hello</title></head>
<body><div id="output"></div></body>
<script>output.innerHTML += "<p>hello, world.</p>";</script>
</html>
```

在這個範例中，我們所使用的語言是HTML5，你可以從`DOCTYPE`中得知文件的格式（你可以從附錄B得到更多的資訊）。具有「Output」這個`id`的`div`元素，是用來處理程式中的輸出內容。我們有`innerText`或`innerHTML`這兩個選擇，但是我們選了`innerHTML`，因為它可以讓我們很方便地用段落元素來加入新的一行（如果你想弄得更花俏一些，還可以把文字改成粗體或斜體）。

如果你在檔案總管中用滑鼠雙擊這個檔案（如圖A.1所示），或是把它擺到Windows市集應用程式中來執行這段程式碼，就會出現你所期望的結果。

只要你的Javascript基礎執行環境開始運行後，你就不必關掉它之後再重新啟動它—相反的，你只要在瀏覽器中按[F5]鍵（或者是功能表列中的「檢視」→「重新整理」），或者是在微軟的Visual Studio 2012中按下[Ctrl]+[Shift]+[R]（或者是功能表列中的「偵錯」→「重新整理Windows應用程式」）就可以重新執行同一支Javascript程式。



圖A.1 Javascript 的輸出結果"hello, world"

同樣的，如果你的應用程式需要偵錯的話，你可以在Internet Explorer 10中按[F12]鍵，就可以帶出偵錯工具組；或者你也可以很輕易地在Visual Studio偵錯器中執行你的應用程式；這兩種方式都可以設定中斷點、在執行的時候更換變數，還有執行程式碼等等。

將關注的事分開處理

在進行更進一步的教學之前，我們應該先談談Javascript要放在何處，才是最適合的。最差的方式是在HTML中插入Javascript的程式碼。最好的方式是把Javascript程式碼分別放在一個或多個檔案中，然後在HTML中設定參考這些檔案：

```
<!DOCTYPE html>
<!-- default.html -->
<html>
<head>
  <title>hello</title>
  <script src="default.js"></script>
</head>
<body><div id="output"></div></body>
</html>

// default.js
output.innerHTML += "<p>hello, world.</p>";
```

現在，除了將實際上執行的Javascript程式碼放在`<script>`元素中之外，我們使用了`src`屬性來指向另一個檔案。`src`屬性中的檔案名稱格式可以是相對路徑的檔案名稱（就如同範例中的內容），或者是絕對的URL，這種方式可以讓你很方便地將外部函式庫導入（例如，你可以透過`http://code.`

圖A.2 在Javascript中存取未定義的HTML元素

jquery.com/jquery-1.8.0.js，將最常用的jQuery函式庫版本1.8導入）。此外，另一種最常見的方式，是把所有的Javascript檔案，放在HTML最頂端的head元素中。然而在我們的範例中，當一開始執行這個程式碼後，就會遇到像圖A.2一樣的問題。

這個問題是當瀏覽器一遇到HTML標籤時，就會立即驗證這些標籤，這意謂著<script>元素（及元素中所參考的Javascript程式碼），會在 輸出結果建立前先被執行。在這個範例中，比較保險的方式是等到HTML都載入之後，再執行Javascript，我們可以透過DOMContentLoaded事件來達到此目的：

```
// default.js
document.addEventListener("DOMContentLoaded", function () {
  output.innerHTML += "<p>hello, world.</p>";
}, true);
```

document物件屬於全域物件，它所代表的是現在已載入HTML檔案的進入點。DOMContentLoaded是這個物件的事件之一（DOM，Document Object Model（文件物件模型）的縮寫），它所表示的是HTML檔案已經完整地載入，這也意謂著此時存取 輸出是不會有問題的。我們把程式碼包在一個沒有命名的函式中，而且把它當做是一個參數傳送，這是因為Javascript將函式當做是第一級物件，稍後我們會在本附錄中詳細地討論這個功能。

將id當做物件

有一個實用的技巧，就是把 元素的id直接當做Javascript的物件來使用：

```
<!-- default.html -->
...
<div id="output"></div>
...

// default.js
...
//直接透過div的id來使用它
output.innerHTML += "<p>hello, world.</p>";
...
```

在這個範例中，微軟web平台的實際執行方式是，在已載入的HTML中，把具有id的每一個物件，轉換為具有相同名稱的Javascript物件。如果我要把這段程式碼寫得更詳細的話，大概會寫成這樣：

```
...
//明確地取得HTML元素
var output = document.getElementById("output");
output.innerHTML += "<p>hello, world.</p>";
...
```

這種簡潔的寫法可能沒辦法在每一款瀏覽器上執行，但是在微軟的Javascript實作環境上卻能夠運作得很順暢。

啟動WinJS

在HTML元素可以使用時，會觸發已經打包為事件處理程序的初始程式碼。而這個程式碼當然也能如你所願，在瀏覽器及Windows市集應用程式中執行。不過，在Windows市集應用程式中的結構通常會稍微冗長一些，就如你在第一章「哈囉，Windows 8」所看到的：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>hello</title>

    <!—WinJS 參考 -->
```

```
    <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
    <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

    <!-- hello references -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>
</head>
<body>
    <div id="output"></div>
</body>
</html>
```

除了參考default.js之外，由Visual Studio內建範本所產生的Windows市集應用程式還可以使用特殊的URL格式，來導入base.js及ui.js這兩個檔案，它們是用來定義專屬Javascript的視窗函數庫（又稱為 WinJS）。而相關的default.js檔案則會遵循這個模式：

```
(function () {
    "use strict";

    var app = WinJS.Application;
    app.onactivated = function (args) {
      output.innerHTML += "<p>hello, world.</p>";
    };

    app.start();
})();
```

一般來說，Javascript的事件處理方式採取了兩種形式。第一種，也就是我們稍早所看到的addEventListener，可允許針對單一物件的相同事件，加入多個事件處理器。而另一個onEventXxx，則稍微簡單些，不過只允許針對單一物件的特定事件加入一個事件處理器。但每個人的喜好不同，而導致陪審團分裂，甚至連本書的兩位作者都各有所好（Brandon喜歡第一種，不過Chris喜歡第二種）。這兩種形式皆可在瀏覽器及Windows市集應用程式中執行，所以你只要挑適合你需求的形式來使用就好。

在這個模式中，我們可以看到變數app被指定為WinJS命名空間中的application物件，而變數app內的事件activated，則是整個被包裝成可自我執行的匿名函數，並標記成可嚴謹的執行。這模式內所有的內容需要更進一步的解釋，就讓我們開始吧！

值與資料型態

在Javascript中，你可以用關鍵字var來宣告變數：

```
var u = undefined;                      // undefined
var v;                                  // undefined
var z = null;                           // object
var b = true;                           // boolean
var s = "a string";                     // string
var i = 42;                             // number
var n = 3.14159;                        // number
var pi = Math.PI;                       // number
var o = { name: "Marcus", age: 16 };    // number

var a = [1, "two", 3];                  // object(array)
var d = new Date();                     // object(date)
var re = /str/;                         // object(RegExp)
var matches = re.test(s);               // boolean
var f = function () { return 2 + 2; }; // object
```

Javascript的變數都有特定的資料型態（你可以使用typeof運算子來取得，而輸出的結果則顯示在前段程式碼的註解中）。但Javascript不同於其他C語言的地方是，在變數的生命週期內，資料型態是可以改變的。例如，下面的範例可以順利地執行：

```
var b = true; // b 是boolean(布林)型態
b = 1; // 現在b是數字(number)了
```

Undefined常數[2]代表的是值還未設定。而null值代表的意義與undefined值不同，它所指的是參考的物件是沒有值的，這個與其他C語言中的指標或參考是相同的。

Javascript中的布林與字串值，與其他高階語言（如C#、Java）所代表的意義非常相似，而且可以看你當天的心情如何，再決定要把字串放在單引號或雙引號內（或者你只需要一個簡單的方式在字串中嵌入引號）：

```
var s = "a string";
s = 'another string';
s = 'a \'string\' within a string';
s = 'a "string" within a string';
```

2 技術上來說，undefined常數實際上並不是常數。不過我會把這個極端的案例留給你當做未來的測驗。

在Javascript中，整數與浮點數的是沒有區分的。Javascript反而是採用IEEE754二進位雙精確度64位元的格式來代表數字型態。

如同我先前提及的，函數是第一級的物件，它可以解釋typeof的結果；我們馬上就會更詳細地討探函數的作用。

運算子

Javascript同樣具有標準C語言家族的運算式、運算子及註解型式：

```
var x = 1 + 2 - 3 * 4 / 5 + Math.pow(6, 7) + Math.sqrt(4); // 279938.6
var b = (x == 42) || (x != 452) && (x < 12); // false
var b2 = ("x" == "42"); // false
var b3 = "x".localeCompare("42"); // false
var sameObject = (x === x); // true
var b4 = (x | 4 ) & b3; // false
var y = (x == 1 ? 2 : 3); // 3

if (x) { /* 執行某些動作 */ } else { /* 執行其他的動作 */ }

while (false) { /* 執行動作 */ }

do { /* 執行某些動作 */ } while (false);

for (var i = 0; i < 10; i++) { /* 執行某些動作 */ }

for (var key in { one: 1, two: 2 } ) { /* 執行某些動作 */ }

try {
  switch (x) {
    case 1: /* 執行動作 */ break;
    case 2: /* 無法執行 */
    case 3: /* 執行某些動作 */ break;
    default: /* 錯誤防止機制 */ throw new Error("oops");
  }

  switch ("string") {
    case "one": break;
    case "two": break;
  }
}
catch (ex) { /* oops! */ }
```

我想你能理解這其中的大部份內容[3]，但比較有趣的是三個等號（`===`），它代表的意義是「**絕對相等**」，雙等號（`==`）則是「**型態轉換的相等**」。舉例來說，因為Javascript在型態的轉換上沒那麼嚴謹，所以某些相等的式子會出乎你的意料之外：

```
var b1 = ("1" == 1); // 此結果為真，因為使用了型態轉換
var b2 = ("1" === 1); // 此結果為偽，因為避免了型態轉換
```

當我們專注在這個議題上時，你會碰到在Javascript中的「仿真」的概念，這個概念通常用來做快速的等式檢查：

```
if (false) { /* 偽 */ }
if (null) { /* 偽 */ }
if (undefined) { /* 偽 */ }
if ("") { /* 偽 */ }
if (0) { /* 偽 */ }

if (true) { /* 真  */ }
if (1) { /* 真 */ }
if ("a") { /* 真 */ }
if ("false") { /* 真 */ }
if ([]) { /* 真 */ }
if ({}) { /* 真 */ }
```

在Javascript中，如果你合理地「期待」某些事情為「真」（`true`），那就是「真」的，例如，`null`、`undefined`、空字串及0，這些都是「偽」（`false`）；而非零的數字及非空字串，都是「真」（甚至連字串`"false"`也是「真」的）。但對於空的陣列及空的物件的真偽判斷就沒那麼直覺了，這兩者都為「真」。我猜有可能是，就算它們兩者都是空的，他們仍是非null值的物件參考。

物件

接著我們要提到物件。在Javascript中，宣告一組成對的「名稱：值」的集合，就是取得一個物件的最簡單的方式，就如同以下的例子：

3 你可以從下列的網址http://msdn.microsoft.com/en-us/library/b272f386（v=vs.94）（http://tinysells.com/238）來瞭解Math（數學）物件。

```
var o = { name: "Marcus", age: 16 }; // 物件宣告
var name = o.name;
o.age += 1; // 生日快樂！（多了一歲）
```

在預設情況下，所有的物件可透過附加屬性來擴展，例如：

```
o.favoriteColor = "green"; // 擴展物件
```

這些屬性也可以是函數：

```
o.birthday = function () { this.age += 1; }
o.birthday(); // 生日快樂！
```

與其他的C語言家族（如C++、C#及Java）一樣，關鍵字this是用來參考物件本身，用來呼叫函數，存取與這個物件有關的屬性及函數。

你也可以使用中括號的語法來存取物件屬性：

```
o["age"] += 1; // 生日快樂!
```

雖然這個語法看起來不太像其他的C語言，但它卻是Javascript動態本質的關鍵之一，就像將字串值指定給一個變數，你不必把值直接寫死：

```
var prop = "age";
o[prop] += 1; // 生日快樂！
```

你也可以依照自己的期望來建立巢狀物件：

```
o.mom = { name: "Michelle", age: 29 };
var momAge = o.mom.age; // 29
```

在Javascript中，用物件來儲存一組命名的值（屬性與物件）是常見的方式。Javascript的資料型態就建立在這些原則之上，而另外一個簡單的內建資料型態則是日期。

日期

日期（Date）的資料型態是用來呈現日期與時間資料。你可以透過建立新的日期（Date）物件，來取得現在的日期和時間：

```
var now = new Date();
var s = now.toString(); // "Wed Aug 22 16:22:16 PDT 2012 （譯註：這是日期資料的顯示格式，代表的是2012年8月22日星期二16:22:16 太平洋日光時區）"
```

toString方法適用於所有物件，並且能夠適當的呈現每個物件。你可以在日期物件上使用像getDay及getFullYear的這些方法，能讓你在特定日期

及時間的取得上，獲得更多更控制權。而為了建立某個特定的日期，你可以解析某個字串，或者導入特定的年、月、日及相似的值：

```
var bday = new Date("1969/6/2 1:37pm PDT");
s = bday.toString(); // "Mon Jun 2 13:37:00 PDT 1969"
var bday = new Date(1969, 5, 2, 13, 37);
s = bday.toString(); // "Mon Jun 2 13:37:00 PDT 1969
```

這裏有日期物件的參考資料，其中列出了日期物件的屬性與方法[4]。不過，我要指出在Javascript的日期資料型態中，雖然年、日、小時、分鐘、秒和毫秒都是從1開始計算，但是「月」則是從0開始算（換言之，1969年就是1969，但六月的話，就變成5了），幸運的是，當解析以字串為基礎的日期資料時，這個日期就是從0開始算了；這點你必須非常小心，否則很容易出錯。

正規表示式

Javascript為正規表示式提供了內建且廣泛的支援：

```
var s1 = "yet another string";
var re = /t/g; // 在整個字串中尋找t這個字
var re = new RegExp("t", "g"); // 跟上一行程式的功能相同
var matches = re.test(s1); // 利用re這個條件，檢查字串中有沒有符合的字元
var s2 = s1.replace(re, "T"); // 將小寫的t，取代為大寫的T
// s2 == "yeT anoTher sTring"
```

Javascript為了呈現正規表示式，不但準備了RegExp物件，同時也內建了常數表示式的語法（以反斜線「\」表示）。我已經示範了如何使用正規表示式，來進行簡單的搜尋與取代，但是Javascript對正規表示式有更進一步的支援，如果你喜歡它們，就讓它們來協助你[5]。

---

4 這個網址（http://msdn.microsoft.com/en-us/library/cd9w2te4（v=vs.94）（http://tinysells.com/239））內有日期物件的定義。

5 就像Netscape的Jamin Zawinski所寫的：「某些人，當遇到一個問題時，會想"我知道，我要用正規表示式”，現在他們就遇到兩個問題了」即使如此，我發覺小規模地使用正規表示式還是很有用的，建議你參考以下的連結以獲得更多訊息：http://msdn.microsoft.com/zh-tw/library/h6e2eb7w（v=VS.94）.aspx

陣列

陣列幾乎跟Javascript中的每件事都一樣，也是一個物件。建立一個陣列是很容易的：

```
var a = [1, "two", 3]; // 建立一個具有3個項目的陣列
a[1] = 2; // 陣列的索引起點是從0開始
var sum = a[0] + a[1] + a[2]; // sum == 6
```

陣列具備的屬性，適合用來操作陣列：

```
var length = a.length; // length == 3
```

length這個屬性並不是你所想的那樣，它並不是陣列中的項目數量，而是陣列最大的索引值再加1：

```
a[42] = "the meaning of life";
var length = a.length; // length == 43
```

如果你知道Javascript中的陣列是**「疏鬆」陣列**（**sparse array**）的話，那Javascript計算陣列長度的方式就能夠說得通了。「疏鬆」陣列意謂著，它只存放已設定的數值，不管它最高的索引值有多大。另外，就跟物件一樣，陣列的索引也可以是文字：

```
var a = [1, 2, 3];
a[42] = "the meaning of life";
a["life"] = "the universe and everything";
var length = a.length; // length == 43
```

當遇到計算陣列長度時，字串索引並不會被計入（就如你所見）。如果你對陣列中完整的值集合有興趣的話，你可以使用for-in迴圈：

```
var indices = "";
var values = "";
for (var i in a) {
  indices += i.toString() + ", ";
  values += a[i] + ", ";
}
// indices == "0, 1, 2, 42, life, "
// values == "1, 2, 3, the meaning of life, the universe and everything, "
```

For-in迴圈會將陣列中的索引從頭到尾列出來，所以你可以用來索引陣列內部的資料。

如果只要列出數值索引的集合，for迴圈是最常用的方式：

```
var a = [1, 2, 3];
a["life"] = "the universe and everything";
var indices = "";
var values = "";
for (var i = 0; i < a.length; i++) {
  indices += i.toString() + ", ";
  values += a[i] + ", ";
}
// indices == "0, 1, 2, "
// values == "1, 2, 3, "
```

用來列出陣列中所有內容的第三種方式，就是使用陣列本身的forEach方法（雖然它還是會跳過字串索引）：

```
var a = [1, 2, 3];
a["life"] = "the universe and everything";
var indices = "";
var values = "";
for (var i = 0; i < a.length; i++) {
  indices += i.toString() + ", ";
  values += a[i] + ", ";
}
// indices == "0, 1, 2, "
// values == "1, 2, 3, "
```

在這個forEach版本中，最有趣的事情是它把函數當做一個值，而forEach的實作會呼叫這個函數，並且以值及索引值當做引數。使用已傳入的索引也是選項之一。

```
var a = [1, 2, 3];
var sum = 0;
a.forEach(function (item) { sum += item; });
// sum == 6
```

forEach函數只是陣列類別中幾個有用的方法之一，你可以查看陣列的參考資料來瞭解所有的方法[6]。在C#中，如果你喜好LINQ風格、以集合為基礎的方式來寫程式，那你應該看看Javascript的陣列方法，像filter、map、reduce，還有與.NET方法相同的Where、Select、Aggregate及Any方法[7]。

6 你可以在這個網址中找到陣列物件的定義：http://msdn.microsoft.com/en-us/library/k4h76zbx（v=vs.94）（http://tinysells.com/241）。

7 你可以在這個網頁「Fluent-Style Programming in Javascript」（網址：http://sellsbrothers.com/posts/Details/12692（http://tinysells.com/242））中更進一步瞭解這種模式的寫法。

物件原型（類別）

在Javascript中，陣列（Array）、數學（Math）、日期（Date）、正規表示式（RegExp）、字串（String）及數字（Number）等資料型態，都是內建資料型態的範例之一。而物件（Object）這個資料型態本身，就如你在先前的範例所看到的，能夠支援即時加入新的屬性及方法的能力。

建構子

把在執行時刻「擴展」物件的能力，結合建立建構子函數的能力，你就能夠自己建立新的資料型態：

```
// Person constructor
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.birthday = function () { this.age++; }
}
```

在此我們已經建立了一個建構子，它是一個函數，你可以透過使用new運算子來呼叫它。我們也可以運用Javascript物件「expando」的特性來增加屬性及方法，因此我們可依照你的要求來建立並運用Person物件：

```
var marcus = new Person("Marcus", 16);
var age = marcus.age; // 16
marcus.birthday();
age = marcus.age; // 17
```

如你所見，你可以直接在Javascript的物件中新增屬性及方法。不過，我們還有其他不那麼直接的方式，可在物件中加入屬性及方法。

原型（prototype）

除了建立與每個物件有關的方法之外，還有更優化的方式，那就是Javascript支援每次只針對某個資料型態設定方法，這種方法是透過使用建構子中的特別屬性prototype：

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
```

```
// 把birthday函數加到所有的person物件中
Person.prototype.birthday = function () { this.age++; }
```

使用方式都一樣（換言之，我們仍用相同的方式建立了Person物件的實體，birthday方法仍然以相同的方式運作）；唯一不同的地方是，我們只用了prototype一次來設定方法，但所有Person物件的實體都具備了這個方法。這種方式有效的原因，是因為當在解析某個物件中的名稱時，無論它們是屬性或方法，只要名稱不存在於這個物件中，Javascript就會檢查prototype。

你也可以使用這種方式來設定預設的屬性值：

```
function Person(name, age) {
   if (name) { this.name = name; }
   if (age) { this.age = age; }
}

// 把這些成員加到所有的 Person 物件中
Person.prototype.birthday = function () { this.age++; }
Person.prototype.name = "";
Person.prototype.age = 0;
```

改寫後的Person建構子，在開始建立物件時，會檢查name和／或age參數是否有傳入值，如果有值傳入，而且這個屬性的值已由prototype設定，才會修改這個值（譯註：這個修改的值會寫入到新的記憶體空間；否則的話，這些由Person建構子建構的所有物件中，相同的屬性值（未修改前）會共用同一個記憶體空間），這種方式提供了些許記憶體使用的最佳化。這些在prototype中的值會一直被使用，直到被更新為止（又稱為寫入時複製（**copy-on-write**））：

```
var marcus = new Person("Marcus", 16);
var baby = new Person(); // 這會使用prototype的值
var age = baby.age; // 0, 這個值來自prototype
baby.age = 1; // 寫入時複製
var babyAge = baby.age; // 1, 來自物件
var marcusAge = marcus.age; // 16, 來自物件
```

在Javascript中，這種模式跟C++、C#或Java的「類別—物件」模式（在編譯時，物件實體的類別是會被凍結的）是不同的。這是「原型—物件」模式，可讓我們模擬出一些類別實體的語意，這就像繼承，但也有一些有趣的特點是，全部的物件都會擁有這些內容。舉例來說，我們可以在Person資料類型的prototype上加入或取代方法，甚至連現存的Person物件都可以得到這些功能：

```
var marcus = new Person("Marcus", 16);

// 每個Person物件都可以存取這個方法
Person.prototype.sayHi = function () { return "Hi from " + this.name; };

var hi = marcus.sayHi(); // "Hi from Marcus"
```

看到這樣的結果，會出現兩派人士的看法：一派人士會認為：「哇，這真有趣！」，但另一派人會覺得：「我的天呀，維護這些東西將會是可怕的夢魘！」從結果上來看，兩方都沒錯。然而，我最喜愛的功能是，可以在內建的資料型態中附加功能的能力。舉例來說：

```
Array.prototype.sum = function () {
  return this.reduce(function (current, item) { return item + current; }, 0);
};

var nums = [1, 2, 3];
var sum = nums.sum(); // 6
```

在這個例子中，我透過使用reduce函數，在Array（陣列）的資料型態上實作了sum函數，它非常實用，因為我永遠不用記得引數在累加函數中的順序，累加函數已經把reduce函數視為是引數了。對於那些熟悉C#的人來說，在現有資料型態中加入成員的能力，是Javascript與C#的擴充方法相同的地方，但卻沒有那麼令人討厭的編譯型態安全檢查。

而對於那些希望在自訂型態中，能具備某些安全性的人來說，當在使用Object.create方法時，可被允許做某些事情，例如像是關閉在prototype上換裝方法的能力：

```
function Person(name, age) {
  if (name) { this.name = name; }
  if (age) { this.age = age; }
}

// 使用Object.create來產生prototype
Person.prototype = Object.create(null, {
  birthday: { value: function () { this.age++; } },
  name: { value: "", writable: true },
  age: { value: 0, writable: true },
  _favoriteColor: { value: "blue", writable: true },
  favoriteColor: {
```

```
    get: function () { return this._favoriteColor; },
    set: function (value) { this._favoriteColor = value; },
  },
});
```

Object.create函數會需要一個初始值，來代表一個現有的使用物件為原型。然而對於簡單的建立prototype來說，我們會傳入一組屬性名稱值，在這裏，值是每一個屬性的設定資訊，在我們的例子中，我們已經將birthday變成一個方法；name、age及_favoriteColor變成了讀／寫屬性，而favoriteColor是已計算（calculated）屬性。按照慣例，名稱前面有底線的話，意謂著_favoriteColor是Person資料型態內部的實作細節，外部的任何利用是不會碰觸到的（雖然沒有真正的方式來執行）。「私有（private）」_favoriteColor屬性是用來做為favoriteColor存取函數的內部儲存，這些函數可以在你需要的時候，再實作它們（例如，執行實際的計算功能）。

任何屬性的可寫入（writable）屬性傳到Object.create的預設值是false，如果任何人嘗試修改不能寫入的值，就會造成執行階段錯誤。使用方式就如你所期望的：

```
var marcus = new Person("Marcus", 16);
var age = marcus.age; // 16
marcus.birthday();
age = marcus.age; // 17
var color = marcus.favoriteColor; // "藍色"
marcus.favoriteColor = "azure";
color = marcus.favoriteColor; // "azure(蔚藍色)"

marcus.birthday = function () { /* 劫持 */ }; // 錯誤 (無法寫入)
```

屬性與已計算屬性兩者的使用方式與先前的相同。不過，如果要嘗試取代任何唯讀的值，不管是屬性還是方法，結果都會是執行階段錯誤。

原型繼承（Prototypal Inheritance）

在Javascript中，除了使用建構子函數來模擬類別之外，你也可以透過覆寫資料型態的prototype屬性本身的預設值，來模擬繼承。這種方式稱為**原型繼承（Prototypal Inheritance）**，請看以下的範列：

```
// "父"類別
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.birthday = function () { this.age++; };
}

// "子"類別
function Student(name, age, id) {
  Student.prototype.call(this, name, age); // initialize the "base"
  this.id = id;
}

// 從student繼承person
Student.prototype = Person;

var marcus = new Student("Marcus", 16, 452);
var name = marcus.name; // "Marcus" (Person.name)
var age = marcus.age; // 16 (Person.age)
var id = marcus.id; // 452 (Student.id)
marcus.birthday(); // (Person.birthday)
age = marcus.age; // 17 (Person.age)
```

這段程式碼最關鍵的部份是，使用了call函數來呼叫Student資料型態的「父」類別（稍後我們會討論這部份），而且將預設（物件）的prototype屬性更改至Person，這提供我們一個在Student類別中，「繼承」Person所有方法的效果。

靜態成員

靜態屬性或函數可以很容易地附加至建構子函數，變為其中的成員：

```
function Person(name, age) {
  if (name) { this.name = name; }
  if (age) { this.age = age; }
  Person.people++; // 追踪所建立的people數量
}

Person.prototype = Object.create(null, {...});

// 建立靜態屬性
Person.people = 0;

var marcus = new Person("Marcus", 16);
var tom = new Person("Tom", 17);
```

```
var john = new Person("John", 18);
var people = Person.people; // 3
```

一般來說，在Javascript中物件的運用非常地有彈性，以致於它們可以被用來實作所有類型的功能，而這些並不會直接出現在這個程式語言中。不過，一但弄清楚了這些功能和模式，你便可以很容易地在寫程式時運用它們。

透過WinJS定義物件

為了一個稍微簡單的模式，可以用來隱藏Javascript物件vs.靜態方法的實作細節，還有擴充建構子的原型（prototype）vs.擴充建構子自身的函數，WinJS（為Javascript建立的Windows函式庫）一包括了預設在Windows商店中所有的Javascript專案一提供了`Class.define`方法：

```
WinJS.Class.define(constructor, instanceMembers, staticMembers)
```

你可以在這個輔助函數中，一次宣告並定義自訂「類別」的所有三個元素：

```
var Person = WinJS.Class.define(
  /* constructor */
  function (name, age) {
    if (name) { this.name = name; }
    if (age) { this.age = age; }
    Person.people++;
  },
  /* instance members */
  {
    birthday: { value: function () { this.age++; } },
    name: { value: "", writable: true },
    age: { value: 0, writable: true },
    _favoriteColor: { value: "blue", writable: true },
    favoriteColor: {
      get: function () { return this._favoriteColor; },
      set: function (value) { this._favoriteColor = value; },
    },
  },
  /* 靜態成員 */
  {
    people: { value: 0, writable: true },
  }
);
```

屬性與方法的定義其實是相同的—唯一不同的地方是，Class.define方法的微妙之處是把所有的東西都一起給你。如果你想實驗模擬單一或多重繼承的話，你可以透過Class.derive及Class.mix方法來達成。

以這種方式產生的資料類型，其使用方式跟以前一樣：

```
var marcus = new Person("Marcus", 16);
...
```

就算WinJS都已經把基礎的資料型態準備好了，就如同你親自完成的一樣，但對於那些想要在Javascript中，模擬C++、C#及Java所具備的「類別」概念的人來說，Class.define方法是非常有用的輔助功能。

函式（function）

就如你所看到的，Javascript的函式是第一類的物件，以下的函式宣告方式結果都是一樣的：

```
// 定義一個名稱為"foo"的函式
function foo() { /* my function */ }

//定義一個名稱為"foo"的函式
var foo = function () { /* my function */ };
```

你也可以把函式當做參數，傳給其他的函式：

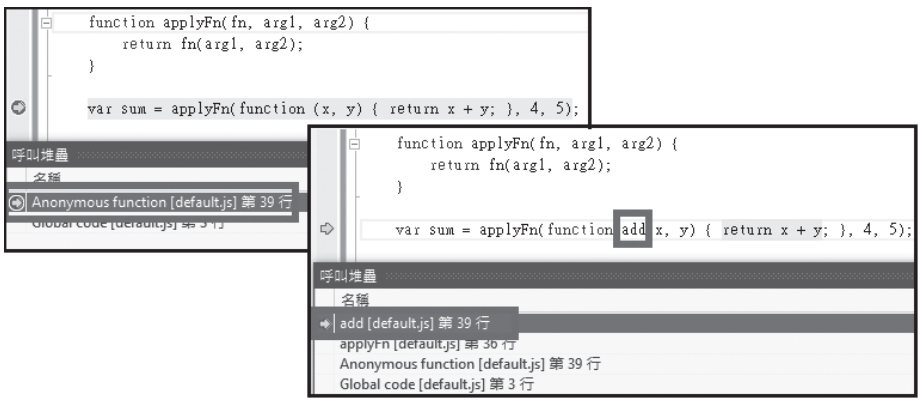
```
function applyFn(fn, arg1, arg2) {
  return fn(arg1, arg2);
}

function add(x, y) { return x + y; }
function multiply(x, y) { return x * y; }

var a = applyFn(add, 4, 5); // 9
var b = applyFn(multiply, 4, 5); // 20
```

當然，這個例子有些刻意，但是它展現了將函式當作物件的兩種面向。在Javascript中，就好比在LINQ導向的C#（它將匿名函式稱為lambdas），這是常見的用法，就如同你已經在這整篇附錄中所看到的。

當你使用匿名函式語法時，你可能會發現，雖然這個函式不需要命名，但幫它取個名字或許會更有用：

```
// 定義一個名為 "foo"的函數
var foo = function foo() { /* my function */ };
```

在這個例子中，這個函式，還有擁有這個函式的變數，兩者都取名為「foo」。在使用Visual Studio 2012建立Windows市集應用程式時，這種方式特別地有用，因為函數名稱會出現在呼叫堆疊中，如圖A.3所示：



圖A.3 定義匿名函式的名稱，會讓它出現在Visual Studio 2012的呼叫堆疊中

如果你更深入地瞭解WinJS程式碼時，你會瞭解正是這個原因，才需要給匿名函式取名。

函式的參數

無論你在參數清單中定義了多少參數，函式都可以處理：

```
function Person(name, age) {
  if (name) { this.name = name; }
  if (age) { this.age = age; }
}
```

檢查參數是否已經設定是常見的做法，因為在Javascript中，所有的參數不一定要設定值：

```
var unknown = new Person(); // 這樣可以
var babyMarcus = new Person("Marcus"); // 這樣也可以
var marcus = new Person("Marcus", 16); // 這樣也沒問題
var moreMarcus =
  new Person("Marcus", 16, "cheese", "pizza", "video games"); // 這樣也OK
```

以上這四種呼叫方法，在Javascript中都沒問題。如果你想要找出那些額外傳入函數的參數，可以使用arguments集合：

```
function Person(name, age) {
  if (name) { this.name = name; }
  if (age) { this.age = age; }

  // check for other arguments
  if (arguments.length > 2) {
    this.likes = Array.prototype.slice.call(arguments, 2);
  }
}
```

不幸的是，arguments集合並不是陣列，所以我們無法直接呼叫slice方法。不過我們可以使用call方法，就算argument變數不是陣列，它會強迫Javascript把arguments變數視為是陣列型態[8]。

呼叫和繫結

就像Javascript中的其他物件一樣，函數是非常具有彈性的。例如，請回想起之前我們在Array原型中加入了一個函數，因此就擴展了這個資料型態：

```
// 擴展陣列的資料型態
Array.prototype.sum = function () {
  return this.reduce(function (current, item) { return item + current; }, 0);
};

// 使用你自訂的陣列方法
var nums = [1, 2, 3];
var sum = nums.sum(); // 6
```

在這種方式中，你可能會認為這種加入功能的想法，並不是你寫程式的習慣，尤其是如果你擔心有人也使用了相同的方式，就會覆寫掉你的方法。不過，你可以建立一個函數，把陣列當作是一個參數：

```
// 取得一個陣列參數
function sum(array) {
  return array.reduce(function (current, item) { return item + current; }, 0);
};
```

8 單元測試會變成為任何實際的Javascript應用程式發展程序的必備部份。

```
// 取得一個陣列參數
var nums = [1, 2, 3];
var sum = sum(nums); // 6
```

不過，你可能已經注意到，對成員函數來說，都有一個this變數提供給它們。如果你想要的話，可以使用call函數來設定this變數：

```
function sum() {
  // 假設"this"變數是一個陣列
  return this.reduce(function (current, item) { return item + current; }, 0);
};

// 調用sum函數，彷彿它看來就像是一個Array方法
var nums = [1, 2, 3];
var sum = sum.call(nums); // 6
```

在這個例子中，我們調用了sum函數，就好像它是Array資料型態的成員一樣，而不用實際地去擴展Array資料型態。因為我想鼓勵你去運用它，所以我不會展示給你看；我更希望你能在Javascript程式碼中詳細的閱讀它，而且用心地去理解它（就像稍早之前，我們用Student建構子呼叫父類別Person的建構子，還有在arguments物件上使用Array.slice方法的技巧）。

在Javascript較常用到的是bind函數，它會使用this變數來建立新的函數物件，而這個this變數可繫結至任何你指定的物件。對於要將物件繫結至函數來說，這是非常有用的，就像在C++中的成員函數指標，或是C#中的委派：

```
WinJS.UI.Pages.define("/html/postsPage.html", {
  ready: function (element, options) {
    this.section = document.querySelector("section[role=main]");
    ...

    // 對包含"this"的downloadError方法建立回呼
    WinJS.xhr(..., downloadError.bind(this));
  },
  ...,
  // 使用 "this" 變數
  downloadError: function (feed) {
    this.section.innerHTML = "<p>error</p>";
  }
});
```

請回想在第一章中，我們定義了一個可以下載訊息來源的瀏覽網頁。這個網頁本身就是由Pages.define方法所定義的物件，Pages.define本身只是個Class.define方法的外覆類別。我們以ready還有downloadError（及諸如此類的）方法做為結束，兩者都使用了this變數來引用頁面狀態，包含了在下載錯誤事件中的Section元素，其中有我們所撰寫的HTML狀態。

對我們的目的而言，這個故事的重要部份是在downloadError方法上調用bind時，傳入了this變數。當取得訊息來源的錯誤事件發生時會調用bind，這種方式實際上會傳回新的函數，這函數將會設定this變數和呼叫downloadError方法。事實上，在邏輯上你可以想像是透過call函數來實作bind函數，就像這樣：

```
Function.bind = function (obj) {
  // 神奇的是省略傳遞特定參數……
  return function () { this.call(obj); }
};
```

所有這一切的結果，代表的是如果你要你的物件方法可以完整無缺地隨著this變數被回呼的話，你就需要使用bind方法。在Javascript程式碼中，這種方式使用得很普遍，而且本書範例的許多部份也使用這種方式。

閉包（Closures）

最後壓軸的是在Javascript中，函式做得最有用的一件事是，捕捉到圍繞在它們週圍的變數，這種函式稱之為閉包。

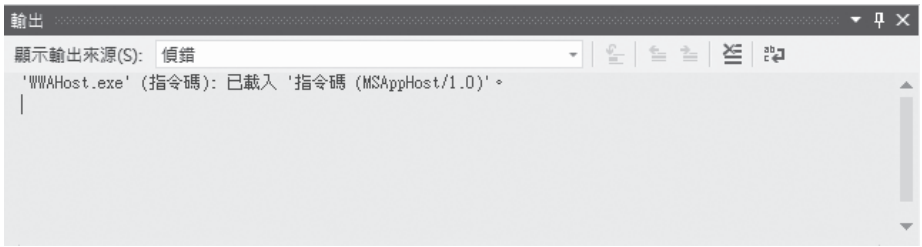
```
var say = "hello"; // 定義一個變數

var p1 = document.querySelector("p"); // 捉取文件的第一段
p1.onclick = function () {
  // 捕捉一個變數
  Debug.writeln(say); // "hi" 輸出到除錯主控台
};

// 更改變數內容
say = "hi";
```

在這段程式碼中，我們建立了一個變數，當它在滑鼠點擊事件中被捕捉到之後，就會更改它的值。當點擊事件被調用時，它會取得這個變數現

在的值。這種行為就跟在C#及新的C++ 11標準中的匿名（lambda）函式一樣。

偵錯輸出

因為我賣弄了`Debug`物件的`writeln`方法，所以你應該知道，當你在Visual Studio 2012的偵錯模式下執行應用程式時，你可以點選功能表上的「偵錯」→「視窗」→「輸出」，會看到偵錯內容的輸出（還有`write`方法的輸出，如圖A.4所示）。



圖A-4 正在執行中的Visual Studio 2012輸出視窗

大部份情況下，我發現我都使用Visual Studio 2012的中斷點、資料提示方塊，還有Javascript主控台（點選「偵錯」→「視窗」→「Javascript主控台」），來取代傾印字串到偵錯輸出中，但對於記錄控制流程來說，這個功能還蠻方便的。

有效範圍宣告（Scoping）

在Javascript中，有效範圍（scope）有兩種：全域（Global）與函式（function）。全域範圍是在函式外面定義的任何事。

```
var x = 0; // 全域範圍
function foo() {
  var y = 1; // 函式範圍
}
```

假設程式正在瀏覽器或Windows市集應用程式環境中執行，任何在全域範圍中定義的東西，都能自動地加入全域window物件，成為屬性：

```
window.x = 0; // 全域範圍
function foo() {
  var y = x; // 存取全域 x
  window.z = 2; // 這也是全域範圍的屬性
}
```

如果你要明確地將某些東西宣告為全域範圍的話，window物件是非常有用的。

提昇（hoisting）

如果你過去在C語言家族中，經常用語彙範圍（lexical scoping）來宣告（也就是說，在某個函式中最早定義的變數，後面就不能再定義了；或者是在迴圈外部定義的變數，就不能定義成為迴圈的一部份）的話，那你在寫Javascript的程式時，就把這些規則都忘了吧：

```
function bar() {
  var a = 1;
  var x = b; // ok (雖然b還未定義)
  var y = c; // 錯 (c從未宣告)
  for (var i = 0; i < 10; i++) { }
  var b = i; // i = 10, b = 10, x = undefined(未定義)
}
```

從本質上來說，你可以想做是Javascript函式中的每個變數，都「提昇」到函式一開頭處宣告，然後當執行到某些設定值的程式碼時才定義變數。區域性的參照規則並不適用。

模組

函式通常用來定義有效範圍，因為它要執行的工作不會影響到全域命名空間：

```
function init() {
  var a = 1;
  var b = 2;
  // 初始化...
}

init();
```

在這個範例中，**init**函式被定義要執行某些工作，然後在定義之後馬上被呼叫。這使得任何暫時性的變數在工作完成後，就不會存在於有效範圍內了。事實上，這種模式非常常見，而且可以簡化成這樣：

```
// 匿名的自我執行函式，亦稱為Javascript模組
(function () {
  var a = 1;
  var b = 2;
  // 初始化...
})();
```

因為Javascript沒有模組的正式定義，而模組又可以在不影響全域命名空間的情況下工作，於是Javascript社群就採用了這種函式，並結合了函式所提供的有效區域特性，它可以匿名、可以在執行時定義，而且可以被馬上調用。任何在這樣的函式（就像在任何函式中）中所定義的事，它們的有效區域就被規範在函式中。

命名空間（Namespace）

如果你要以同樣的方式來繼續工作，卻不會影響到全域命名空間的話，你也會想要在命名空間中，能夠堆疊自訂的資料型態。雖然命名空間在Javascript中，沒有像在C++、C#及Java中有正式的定義，我們還是能使用物件來建立：

```
var Book = Book || {}; // 建立全域命名空間
Book.Samples = Book.Samples || {}; // 建立巢狀的命名空間
Book.Samples.Person = function (name, age) { ... } // 加入一個建構子
```

第一行程式碼是用來檢查Book物件是否存在；如果不存在就建立一個新的空Book物件。這會形成我們自訂的根命名空間。第二行程式碼是用來檢查巢狀的Sample物件是否存在，以形成巢狀命名空間。在這個巢狀命名空間中，我們可以加入成員，不論它們是資料型態的建構子，或是會變成命名空間元素的命名空間屬性值。從命名空間所建立的資料型態實體，將會依照你所認為的命名空間的運作方式來運作：

```
// 建立一個資料型態的實體
var marcus = new Book.Samples.Person("Marcus", 16);
```

如果你要建立更多的命名空間，你也可能要選擇WinJS來協助你做這件事。

WinJS命名空間

當使用WinJS建立命名空間時，它可以協助你明確地定義：

```
//建立Book.Samples命名空間及巢狀的Person資料型態
WinJS.Namespace.define("Book.Samples", {
  Person: function (name, age) { ... },
});

var marcus = new Book.Samples.Person("Marcus", 16);
```

使用WinJS命名空間建立協助功能的主要原因是，它知道如何為你解析句點，而且能夠很聰明地沿著已經建立的道路以避免侵犯命名空間。正如你所想的，命名空間內部的define類別充分運用了WinJS協助功能：

```
//建立Book.Samples命名空間及巢狀的Person資料型態
WinJS.Namespace.define("Book.Samples", {
  Person: WinJS.Class.define(function (name, age) { ... }, ...),
});

var marcus = new Book.Samples.Person("Marcus", 16);
```

整組的WinJS命名空間及類別已經使用這些協助功能（還有它們的變形）定義完成。

Strict模式

如果你讀到由Visual Studio 2012的Windows市集應用程式專案範本所產生的程式碼，你會看到我在這個附錄，還有其他的章節中所示範的嚴格模式用法：

```
// default.js
(function () {
  "use strict";
  ...
})();
```

儘管這個看起來像會被Javascript解析器當作未指定或未使用的字串常數，而不去處理它，但它的確是Javascript Strict模式的宣告。Javascript Strict模式的目的是將某些常見的錯誤捕捉器開啟。在某個範例中，如果你在Strict模式宣告的區域之外，使用未宣告的變數是沒有問題的：

```
// default.js
(function () {
  x = 1; // OK
})();
```

在前面的程式碼中，Javascript假設你已經宣告了名為x的全域變數，而且已經設定值。當然這個問題是，錯誤的拼字會使我們很容易地忽略這些明顯的錯。為了避免這個問題，就要開啟Strict模式：

```
// default.js
(function () {
  "use strict";

  x = 1; // err
})();
```

在Windows市集應用程式中，當你嘗試使用一開始未宣告的變數，就會出現錯誤訊息，如圖A.5所示：

圖A.5 在Strict模式中的未定義變數，會造成執行時刻例外

在這裏列出了許多使用Strict模式的好處[9]，我建議你使用它。

序列化（Serialization）

本附錄並未將Javascript語言的所有功能介紹給你，不過也應該足夠幫助你開始撰寫了。但還有一件事，我會在本節介紹內建於Javascript中，非常有用的序列化技術，稱為Javascript物件標記法[10]（Javascript Object Notation, JSON）。它的概念是，你可以取得一個Javascript物件的內容，然後把它轉成字串來儲存，或是反過來，從字串轉為Javascript物件：

```
var marcus = {
  name: "Marcus",
  age: 16,
  likes: ["cheese", "pizza", "video games"],
};

var s = JSON.stringify(marcus); // 將Marcus轉成字串
// s = '{"name":"Marcus","age":16,"likes":["cheese","pizza","video games"]}'
var m2 = JSON.parse(s); // 取回Marcus
```

JSON格式就是更嚴謹、而且是Javascript物件初始化格式的合法形式。JSON中的`stringify`及`parse`方法，就是分別將物件內容轉換為字串，或從字串轉回物件內容。

9 不過，我還是鼓勵你去這個網址看看Strict模式的限制清單：http://msdn.microsoft.com/zh-tw/library/br230269（v=VS.94）.aspx

10 取得詳細JSON資訊的最佳下注地點，永遠是：http://json.org