前言

即使這是我的第二本 JavaScript 書籍，我還是會對自己扮演 JavaScript 專家與傳道者的角色感到有點驚訝。與許多程式員一樣，我在 2012 年之前仍然對 JavaScript 抱著強烈的偏見，現在我對自己有 180 度大改變仍然覺得很驚奇。

我的偏見與許多人一樣：當時，我認為 JavaScript 只是一種玩具語言（因為沒有真正瞭解，所以無知），它是危險、邋遢、未受過訓練的程式員才會使用的語言。其實大家的想法是有一些道理的，首先，ES6 的開發速度很快，就連它的創造者 Brendan Eich 都承認，剛開始的時候，有一些東西是不正確的，但是當他發現時，已經有許多人開始使用有問題的功能了，所以他沒辦法有效率地修正（不過，哪個語言沒有這種問題？）。其次，大家很容易就會用到 JavaScript 程式，這是因為大家都有瀏覽器，除此之外，他們只要隨便做一些事情，就可以讓 JavaScript 快速地在網路上散播。大家都會透過試誤法來學習，他們會閱讀彼此的程式碼，所以很容易就抄襲因為沒有充分理解語言而寫出的爛程式。

我很開心自己已充分瞭解 JavaScript，知道它不是一種玩具語言，而是具備非常雄厚的基礎，並且強大、有彈性，且富有表現力的語言。我也很開心自己可以擁抱 JavaScript 帶來的易用性。我對業餘者沒有偏見：每個人都必須從某個地方開始，編寫程式是一種可以賺錢的技術，而且將寫程式當成職業有很多好處。

對於程式設計新手、業餘者，我必須說：身為業餘玩家，並不可恥。**持續**當一個業餘玩家並不可恥。如果你想要練習編寫程式，就去**練習**它。盡可能地使用所有資源來學習一切事物。保持開放的心態，且最重要的，或許是質疑每一件事，質疑每一位專家、質疑每一個有經驗的程式員，不斷地問“為什麼？”。

在多數情況下，我會試著讓本書忠於 JavaScript，但不可能完全沒有自己的意見。當我

發表意見時，你就當它們只是個意見，你當然可以不同意，我也鼓勵你尋求其他有經驗的開發者的意見。

你正處於一個令人振奮的 JavaScript 時期學習。Web 已經離開初期階段了（技術上來說），且 Web 已經不再是 5 或 10 年前的那個令人困惑、複雜的蠻荒時代了。HTML5 與 ES6 等標準，可讓你更輕鬆地學習 web 開發，也更容易開發高品質的應用程式。Node.js 已將 JavaScript 延伸到瀏覽器之外，現在已經是系統指令碼、桌上型應用程式開發、後端 web 開發，甚至嵌入式應用程式的選擇。自從我在 1980 年代開始寫程式以來，從來沒有感覺這麼有趣過。

JavaScript 簡史

JavaScript 是 Brendan Eich 在 1995 年開發的，當時他是 Netscape Communications Corporation 的開發者。它的初期發展得十分快速，所以大部分的批評，都是來自它缺乏遠程的規劃。但是，Brendan Eich 不是一個半調子的人－他有很雄厚的電腦科學基礎，所以將非常複雜且有遠見的想法加入 JavaScript，它有許多領先的技術，其他的主流開發者花了 15 年才追上這個語言所提供的精密性。

JavaScript 一開始叫做 Mocha，有一小段時間稱為 LiveScript，1995 年版的 Netscape Navigator 將它正式命名為 JavaScript。JavaScript 的 Java 字眼不是偶然，但很容易讓人誤解：它與 Java 都繼承同一個語法祖先，但是它與 Self（一種 Xerox PARC 在 80 年代中期開發的原型（prototype-based）語言）及 Scheme（Guy Steele 與 Gerald Sussman 在 1970 年代開發的語言，它深深地受到 Lisp 與 ALGOL 的影響）的相似程度都比 Java 還要高。Eich 很熟悉 Self 以及 Scheme，所以使用它們的一些前瞻性思維來開發 JavaScript。之所以使用 JavaScript 這個名稱，部分的原因是基於市場考量，想搭上當時很成功的 Java 的順風車 [1]。

在 1996 年 11 月， Netscape 宣佈它們已經將 JavaScript 提交至 Ecma，Ecma 是個私人、國際性的非營利標準組織，可對科技與通訊產業產生重大的影響。Ecma International 發表第一版的 ECMA-26 規格其實就是 JavaScript。

Ecma 的規格描述的是一種稱為 ECMAScript 的語言，它與 JavaScript 的關係主要是學術性的。技術上來說，JavaScript 是 ECMAScript 的作品，但就實際的目的而言，你可以交互使用 JavaScript 與 ECMAScript。

1 在 2014 年，Eich 在一場面談中承認這件事，來對討厭 JavaScript 的 Sun Microsystems 表達他的不屑。

ECMAScript 的最後一個主要版本是 5.1（通常稱為 ES5），在 2011 年 6 月發表。市面上不支援 ECMAScript 5.1 的老舊瀏覽器已經少之又少，所以我們可以放心地說，目前 ECMAScript 5.1 已經是 Web 通用語言了。

本書討論的 ECMAScript 6 (ES6) 是 Ecma International 在 2015 年 6 月發表的。在發表前，規格使用的名稱是 Harmony，所以你會聽到有人將 ES6 稱為 Harmony、ES6 Harmony、ES6、ES2015 與 ECMAScript 2015。這本書會單純將它稱為 ES6。

ES6

既然 ES5 是目前的 Web 通用語言，細心的讀者或許會想，為什麼這本書要把重心放在 ES6 ？

ES6 代表 JavaScript 語言一個很重要的進步，有一些 ES5 的主要缺失都已被 ES6 解決了。我想，你將會發現 ES6 用起來更令人開心，而且功能更加強大（不過對初學使用者來說，ES5 已經是種令人愉快的語言了）。另外，拜轉譯器之賜，你也可以使用 ES6 來編寫程式，再將它轉譯成與 web 相容的 ES5。

當 ES6 發表時，支援它的瀏覽器將會快速地成長，屆時，你就不需要為了迎合廣大的觀眾而使用轉譯器（我不會傻到隨便預測這會在什麼時候發生）。

我們可以確定的是，ES6 代表 JavaScript 未來的發展，如果你現在投資時間來學習，就可以為將來預做準備，何況現在有轉譯器可用，讓我們免於犧牲相容性。

但是，並不是每一位開發者都可以奢侈地編寫 ES6。很有可能你目前正在處理既有的龐大 ES5 程式群，將它轉換成 ES6 需要付出十分昂貴的代價。有一些開發者並不想要付出額外的成本來轉換程式。

本書除了第一章之外，都會討論 ES6，而不是 ES5。在適當的情況下，我會指出 ES6 與 ES5 不同的地方，但是如果 ES6 的方法比較好，我不會使用範例程式或大篇幅的文字，來說明如何使用 ES5 來做同樣的事情。無論什麼原因，如果你必須持續使用 ES5，這本書應該不適合你（但我希望你將來可以閱讀這本書！）。

我們費盡心思地將重心放在 ES6。因為 ES6 的改善很多，我們很難維持一個明確的教學框架。總之，一本同時想要討論 ES5 與 ES6 的書籍，將無法兩全其美。

本書適用對象

這本書的主要對象是已經具備一些程式設計經驗的讀者（即使只參加過介紹性的程式設計課程，或線上教學）。如果你是程式設計新手，也可以從這本書受益，但你可能還要參考一些基本文章或課程。

對已經有一些 JavaScript 經驗的讀者來說（特別是僅止於 ES5 的），你會看到實際且深入的重要觀念。

來自其他語言的讀者，在閱讀這本書時，應該會感到如沐春風。

這本書的目的，是全面討論語言功能、相關工具、技術，與驅動現代 JavaScript 發展的典範。因此，這本書會從簡單的部分（變數、控制流程、函式）討論到複雜且深入的主題（非同步編程、正規表達式）。按照你的經驗，你可能會發現有些章節比其他章節有挑戰性：初學者當然需要多次閱讀其中的一些內容。

不適合本書的對象

這本書不是全面性的 JavaScript 或相關程式庫的參考書籍。Mozilla Developer Network（MDN）維護一個傑出、詳盡、隨時更新且免費的 JavaScript 參考網站（*https://developer.mozilla.org/en-US/docs/Web/JavaScript*），本書會大幅地引用它。如果你比較喜歡閱讀實體書本，David Flanagan 的 *JavaScript: The Definitive Guide* 相當完整（不過當我寫到這裡時，它還沒有討論 ES6）。

本書編排慣例

本書使用的字型、字體慣例，如下所示：

斜體字（*Italic*）
: 用來表示檔名、副檔名、路徑、網址和電子郵件。對於初次提到或重要的詞彙，中文以楷體字呈現，其對應的英文則以斜體表示。

定寬字（`Constant width`）
: 用來表示程式列表，也用在文章段落中表示程式元素，例如變數或函式的名稱、資料庫、資料類型、環境變數、敘述，以及關鍵字。

第一章

你的第一個應用程式

通常最好的學習方式，就是直接動手做，所以我們會先編寫一個簡單的應用程式。這一章的重點，不是解釋每一個地方發生的事情，所以你會對一些地方感到陌生與困惑，建議你先放輕鬆，不要逼自己瞭解每一件事。第一章的目的，是讓你開始練習。你只要享受這個過程，當你讀完這本書時，就可以充分瞭解這一章的所有內容了。

如果你沒有很多程式設計經驗，一開始會讓你感到挫折的事情，就是電腦這個東西。我們的頭腦可以輕鬆地處理不準確的輸入訊號，但電腦很不擅長處理這件事。如果我犯了一個語法錯誤，你或許會對我的編寫能力產生質疑，但你應該可以理解我的意思。JavaScript 與所有程式語言一樣，無法處理模糊的輸入，所以大小寫、拼寫與單字的順序，都相當重要。如果你遇到問題，請確保你已經正確地複製所有東西：你沒有把分號打成冒號，或把逗點打成句點，你沒有混合使用單引號與雙引號，而且程式都使用正確的大小寫。當你具備一些經驗之後，你就可以知道哪邊可以用自己的方式來做事，哪邊必須完全符合規則，但現在，你遇到的挫折比較少，只要輸入完全一致的範例就可以了。

根據慣例，程式書籍一開始都會有一個範例，稱為 Hello, World，它會在你的終端機上面印出 hello world 這句話。或許你有興趣知道，這個傳統源自 1972 年的 Brian Kernighan，他是 Bell Labs 的電腦科學家。這個例子第一次出現在書中，是在 1978 年的 *C Programming Language*，由 Brian Kernighan 與 Dennis Ritchie 所著。直到今日，*The C Programming Language* 仍然被普遍認為是最好的，也是最有影響力的程式語言書籍，我在寫這本書時，也使用一些來自那本書的靈感。

雖然 Hello, World 對愈來愈成熟的程式設計學生世代來說有點老套，但時至今日，這句話隱含的意義仍然與 1978 年一樣強大：它們是你準備投入精力的對象說的第一句話，它證明你是普羅米修斯，偷取諸神的火種，將神的名字刻到泥偶的神士，如同 Frankenstein 博士（科學怪人的創造者）將精力投入它的創作[1]。我之所以投入程式設計領域，就是因為這種創造力。或許有一天，你會讓某個人造物種具有生命，他說的第一句話，很可能就是 hello world。

在這一章，我們會平衡 Brian Kernighan 在 44 年前開創的傳統文化，與今日程式員的成熟。我們會在螢幕上看到 hello world，但與原本的大相徑庭，不是在 1972 年，用螢光粉顯示的文字。

要從哪裡開始

在這本書，我們將會討論 JavaScript 的各種變化的應用（伺服端、指令碼、桌上、瀏覽器，與其他），但出於歷史與實際的原因，我們會先從瀏覽器程式開始。

其中一種從瀏覽器範例開始的原因，就是我們可以透過它來輕鬆地操作圖形程式庫。人類是視覺動物，將程式概念與視覺元素結合起來，是一種強大的學習工具。在這本書的開頭，我們會花很多時間從許多行文字開始，但在那之前，我們先來看一些比較有趣的視覺化物件。我選擇這個範例還有一個原因，就是它可以介紹一些相當重要的概念，例如事件驅動程式設計，這可以讓你更容易理解接下來的章節。

工具

木匠沒有鋸子就很難做出桌子，我們沒有工具也無法寫出軟體。幸運的是，我們這一章需要的工具很少，只有瀏覽器與文字編輯器。

我很開心地宣布，當我寫到這裡時，世上沒有瀏覽器無法支援你手上的工作。就連程式員長久以來的眼中釘—Internet Explorer 都已經有所改善，現在已經不亞於 Chrome、Firefox、Safari 與 Opera 了。不過，我選擇的瀏覽器是 Firefox，我會在內容說明可以在程式設計旅程中協助你的 Firefox 功能。其他的瀏覽器也有那些功能，但我會用 Firefox 的功能來說明它們，所以當你讀完這本書之後，最順暢的路徑，就是使用 Firefox。

1 我希望你對作品的同情心比 Dr. Frankenstein 還要好，並做出更好的作品。

你需要一個文字編輯器來編寫程式。文字編輯器的選擇，幾乎與選擇宗教一樣。廣義來說，文字編輯器的種類有文字模式編輯器與視窗型編輯器。最受歡迎的兩種文字模式編輯器是 vi/vim 與 Emacs。文字編輯器有一個很大的優點，就是你除了可以在自己的電腦使用它們之外，也可以透過 SSH，也就是遠端連結到某台電腦，用你熟悉的編輯器來編輯你的檔案。視窗式編輯器感覺上比較現代，會加入一些有用（你也比較熟悉）的使用者介面元素。但是，畢竟你編輯的是文字，所以視窗式編輯器並沒有比文字模式編輯器還要好。目前熱門的視窗式編輯器有 Atom、Sublime Text、Coda、Visual Studio、Notepad++、TextPad 與 Xcode。如果你已經熟悉其中一種編輯器，就沒必要改用其他的編輯器。但是，如果你目前還在 Windows 上使用 Notepad，我強烈建議你升級為較先進的編輯器（對 Windows 使用者來說，Notepad++ 是簡便且免費的選擇）。

本書不會說明編輯器的所有功能，但你得學會一些功能：

語法醒目提示

語法醒目提示會使用顏色來分辨程式中的語法元素。例如，它可能會用一種顏色來表示常值，另一種顏色表示變數（你很快就會知道這些名詞的意思！）。這個功能可以讓你更容易看到程式中的問題。大部分的現代文字編譯器都預設啟用語法醒目提示功能，如果你的程式碼不是彩色的，請參考編輯器的文件，來瞭解如何啟用它。

括弧配對

大部分的程式語言都會大量使用括弧、大括弧與方括弧（以下統稱為括弧）。有時括弧的內容有好幾行，甚至超出螢幕範圍，而且括弧裡面通常有其他的括弧，通常是其他的類型。括弧的配對非常重要，如果沒有配對好，你的程式就無法正確地運作。括弧配對功能可以讓你看到括弧開始與結束的地方，並且協助你看到沒有成對的括弧所造成的問題。不同的編輯器會用不同的方法來處理括弧配對，從很細微的線索，到很明顯的提示。沒有成對的括弧，經常是初學者感受挫折的原因，所以我強烈建議你學會使用編輯器的括弧配對功能。

程式碼折疊

折疊與括弧配對有一些關係。程式碼折疊的意思是暫時隱藏與你目前處理的地方沒有關係的程式碼，可讓你把重心放在手上的工作。這個概念，來自將一部分的紙張折起來，以隱藏不重要的部分。如同括弧配對，各種編輯器會用不同的方式來處理程式碼折疊。

自動完成

自動完成（也稱為**文字完成**（*word completion*）或**智慧感知**（*IntelliSense*）[2]）是一種方便的功能，它會試著在你完成打字之前，先猜出你要打的字。它有兩個目的，第一個是節省打字時間，例如，你可以打出 **enc**，接著從清單中選擇 encodeURIComponent，而不需要打完 **encodeURIComponent**。第二個是**發現**其他項目，例如，如果你想要使用 encodeURIComponent 而打出 **enc**，會發現還有一個函式稱為 encodeURI。依編輯器而定，你甚至有可能會看到一些註解，來區分這兩種選項。在 JavaScript，自動完成比其他語言還難做，因為它是寬鬆型態語言（loosely typed language），另一個原因是它的範圍規則（稍後你會看到）。如果自動完成對你來說是很重要的功能，可能得自己去尋找符合需求的編輯器，有一些編輯器絕對是其中的佼佼者，有些其他的編輯器（例如 vim）也提供很強大的自動完成功能，但需要先做一些設定。

註解的註解

JavaScript 與大部分的程式語言一樣，都有可在程式碼中標示**註解**的語法。註解會完全被 JavaScript 忽略，它們是讓你或同事使用的。它們可讓你用自然語言來解釋某些難懂的地方發生什麼事情。在這本書中，我們會廣泛地在範例程式中使用註解來解釋發生的事情。

JavaScript 有兩種註解：行內註解與區塊註解。行內註解的開頭是兩個斜線（//），它的範圍到該行結束的地方為止。區塊註解的開頭是一個斜線與一個星號（/*），結尾是一個星號與一個斜線（*/），它的範圍可達很多行。以下這個範例說明這兩種註解：

```
console.log("echo");    // 在主控台印出 "echo"
/*
   在上一行中，雙斜線之前的都是 JavaScript 程式，
   它必須是有效的語法。雙斜線會開始一個註解，
   它會被 JavaScript 忽略。

   這段文字在一個區塊註解裡面，
   它也會被 JavaScript 忽略。我們將這一段註解縮排，來方便你閱讀，
   但你不一定要使用縮排。
*/
/* 媽，你看，沒有縮排！ */
```

2 Microsoft 的說法。

我們即將看到的階層式樣式表（Cascading Style Sheets，CSS）也使用 JavaScript 的區塊註解語法（CSS 不提供行內註解）。HTML（與 CSS 一樣）沒有行內註解，它的區塊註解與 JavaScript 不一樣，是用笨拙的 <!-- 與 --> 來包住註解：

```
<head>
   <title>HTML and CSS Example</title>
   <!-- 這是 HTML 註解 ...
            它可以延伸好幾行。 -->
   <style>
      body: { color: red; }
      /* 這是 CSS 註解 ...
            它可以延伸好幾行。. */
   </style>
   <script>
      console.log("echo"); // 回到 JavaScript...
      /* ... 行內與區塊註解
            都可以使用。 */
   </script>
</head>
```

動工

首先，我們要建立三個檔案：一個 HTML 檔，一個 CSS 檔，與一個 JavaScript 原始檔。我們可以在 HTML 檔案裡面做任何事情（HTML 裡面可以嵌入 JavaScript 與 CSS），但將它們分開有一些好處。如果你是程式新手，強烈建議你遵循接下來的步驟，我們將會在這一章採取相當有探索性、漸進式的做法，這將有助於你的學習過程。

或許乍看之下，我們做了很多事情，只是為了完成一個很簡單的工作，但過程中有許多道理。我當然可以創造一個範例，用很少的步驟來做相同的事情，但這種做法是在**教你壞習慣**。你即將看到的步驟，將會重複出現，或許你現在覺得它太過複雜，但至少你可以確保自己正在學習**正確**的做事方式。

這一章最後一件重要的事情：這是本書唯一使用 ES5 語法，而不是 ES6（Harmony）來編寫範例程式的章節。這是為了確保範例程式可以執行，即使你使用未支援 ES6 的瀏覽器也可以。在接下來的章節，我們會討論如何用 ES6 來寫程式，以及轉譯它，讓它可以在舊版的瀏覽器上執行。探討這個主題後，本書其餘的內容將會使用 ES6 語法。本章的範例程式很簡單，所以使用 ES5 沒有任何問題。

在這個練習中，你必須確保所建立的檔案都在同一個資料夾或目錄裡面。我建議你建立一個新資料夾或目錄來供這個範例使用，以免將它們混入其他檔案，而無法找到。

我們從 JavaScript 檔案開始。使用文字編輯器，建立一個名為 *main.js* 的檔案。我們先在檔案裡面加入一行程式就好：

```
console.log('main.js loaded');
```

接著建立 CSS 檔，*main.css*。目前還沒有東西需要放入這個檔案，所以我們先加入一個註解，讓它不是一個空檔案：

```
/* 以下是樣式。 */
```

接著建立一個檔案，稱為 *index.html*：

```
<!doctype html>
<html>
   <head>
      <link rel="stylesheet" href="main.css">
   </head>
   <body>
      <h1>My first application!</h1>
      <p>Welcome to <i>Learning JavaScript, 3rd Edition</i>.</p>

      <script src="main.js"></script>
   </body>
</html>
```

雖然這本書不討論 HTML 與 web 應用程式開發，但有很多人是為了它們而學習 JavaScript 的，所以我們會指出一些與 JavaScript 開發有關的 HTML 內容。HTML 文件有兩個主要的部分：**標頭**與**內文**。標頭裡面的資訊不會**直接在你的瀏覽器上顯示**（不過它會影響瀏覽器顯示的東西）。內文包含會被顯示在瀏覽器的網頁內容。重點是，標頭內的元素永遠不會被顯示在瀏覽器上，但內文的元素會（有一些元素類型不會被顯示，例如 `<script>`，而且有些 CSS 可以隱藏內文元素）。

在標頭中，我們有一行 `<link rel="stylesheet" href="main.css">`；它會將目前空的 CSS 檔連結到你的文件。接著，在內文結尾，有一行 `<script src="main.js"></script>`，它也會將 JavaScript 檔連結到你的文件。或許你會對標頭有一行程式，內文的結尾也有一行程式覺得奇怪。我們也可以在標頭放入 `<script>` 標籤，但將它放在內文結尾，是出於效能與複雜度的考量。

在內文中，有 `<h1>My first application!</h1>`，它是第一級標題文字（代表網頁中最大、最重要的文字），之後是一個 `<p>`（段落）標籤，它裡面有一些文字，其中的一些文字是斜體（以 `<i>` 標籤來標記）。

接著在你的瀏覽器中載入 *index.html*。在大部分的系統中，最簡單的做法是在檔案瀏覽器中，按兩下檔案（你也可以將檔案拉到瀏覽器視窗裡面）。你將會看到 HTML 檔案的內文內容。

本書有許多範例程式。因為 HTML 與 JavaScript 檔案都有可能變大，我不會每次都展示整個檔案，我會解釋範例程式該放到檔案的哪個地方。這可能會造成初學者的麻煩，但瞭解程式組合的方式很重要，這是必要的過程。

JavaScript **主控台**

我們已經寫了一些 JavaScript 了：`console.log('main.js loaded')`。它會做什麼事？*console*（**主控台**）是一種純文字工具，可協助程式員診斷他們的作品。這本書將會大量地使用主控台。

不同的瀏覽器會用不同的方式來打開主控台。因為你以後會經常做這件事，我建議你瞭解一下鍵盤快速鍵。Firefox 是 Ctrl-Shift-K（Windows 與 Linux）或 Command-Option-K（Mac）。

在載入 *index.html* 後的網頁，開啟 JavaScript 主控台，你會看到“main.js loaded”文字（如果你沒有看到它，請試著重新載入網頁）。`console.log` 是一種方法[3]，它會在主控台印出你要的東西，它是有益於除錯與學習的工具。

主控台有許多好用的功能，其中，你除了可以看到程式的輸出之外，也可以**直接在主控台輸入** ***JavaScript***，來做測試、學習 JavaScript 功能，甚至暫時修改程式。

3 你將會在第九章學到函式與方法之間的差異。

jQuery

我們將要在網頁中加入一種極受歡迎的用戶端指令碼程式庫，稱為 *jQuery*。雖然這不是必要的，甚至與目前的工作無關，但它是一種無處不在的程式庫，通常是你在網頁程式中第一個加入的程式庫。就算這個範例也可以不使用它，但你愈早習慣看到 jQuery，就對你愈有利。

在內文的結尾加入我們自己的 *main.js* **之前**，我們要將 jQuery 連結進來：

```
<script src="https://code.jquery.com/jquery-2.1.1.min.js"></script>

<script src="main.js"></script>
```

你可以看到，我們使用一個 Internet URL，代表你的網頁如果沒有連上 Internet，就無法正確工作。我們從一個公開的**內容傳遞網路**（*content delivery network*，CDN）連入 jQuery，它有效能上的優點。如果你之後會離線編寫專案，就必須下載檔案，並且連結電腦裡面的版本。現在我們要來修改 *main.js* 檔，以使用其中一種 jQuery 的功能：

```
$(document).ready(function() {
   'use strict';
   console.log('main.js loaded');
});
```

如果你沒有 jQuery 的經驗，這看起來有點像胡言亂語。有一些東西要到後面你才會明白。jQuery 在這裡為我們做的，就是確保瀏覽器在執行我們的 JavaScript 之前（它目前只是一個 `console.log`），已載入所有的 HTML。當我們編寫瀏覽器 JavaScript 時，會用這種做法：任何我們編寫的 JavaScript 都會位於 `$(document).ready(function() {` 與 `});` 之間。請注意 `'use strict';` 這一行是之後會學到的東西，但基本上，它會要求 JavaScript 解譯器更嚴格地看待你的程式。雖然這乍聽之下不是件好事，但它確實可以幫助你寫出更好的 JavaScript，避免常見且難以偵測的問題。我們當然想在這本書中學習編寫相當嚴謹的 JavaScript。

繪製圖形元件

HTML5 有許多好處，其中一種是標準圖形介面。HTML5 *canvas* 可讓你繪製圖形元件，例如正方形、圓形與多邊形。直接使用 canvas 是件痛苦的事情，所以我們要使用一種圖形程式庫，稱為 Paper.js（*http://paperjs.org*），來使用 HTML5 canvas。

Paper.js 並不是唯一的 canvas 圖形程式庫：KineticJS（*http://kineticjs.com*）、Fabric.js（*http://fabricjs.com*） 與 EaselJS（*http://www.createjs.com/#!/EaselJS*）都是相當熱門且強大的替代方案。我用過以上所有程式庫，它們都有很高的品質。

在我們開始使用 Paper.js 來繪製東西之前，需要一個 HTML canvas 元素，才可在上面畫圖。在內文中加入以下的程式（你可以將它放在任何一個地方，例如在簡介的段落之後）：

```
<canvas id="mainCanvas"></canvas>
```

留意，我們指派一個 `id` 屬性給 canvas：所以之後可以在 JavaScript 與 CSS 裡面輕鬆地參考它。如果我們現在就載入網頁，將不會看到任何不同，我們不但還沒有在 canvas 裡面畫任何東西，它在空白的網頁裡面，也只是個空白的 canvas，沒有寬度與高度，所以你很難看到它。

每一個 HTML 元素都可以擁有一個 ID，為了讓 HTML 是有效的（格式是正確的），每個 ID 都必須是獨一無二的。因為我們已經建立一個 canvas，它的 `id` 是 mainCanvas，我們就不能重複使用那個 ID。所以，建議你謹慎地使用 ID。我們之所以在這裡使用它，是因為對初學者來說，一次處理一件事情比較容易，一個 ID 只代表網頁中的一個東西。

我們來修改 *main.css*，讓 canvas 在網頁中浮現。如果你還不熟悉 CSS，不用擔心－這個 CSS 只會設定 HTML 元素的寬與高，並讓它有一個黑邊[4]：

```
#mainCanvas {
   width: 400px;
   height: 400px;
   border: solid 1px black;
}
```

現在載入網頁就可以看到 canvas 了。

現在要在上面畫一些圖案，我們會連結 Paper.js 來協助我們畫圖。在連結 jQuery 的程式後面，在連結我們自己的 *main.js* 程式前面，加入這一行：

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/paper.js/0.9.24/ ↩
paper-full.min.js"></script>
```

4 如果你想進一步瞭解 CSS 與 HTML，我推薦 Codecademy 免費的 HTML & CSS track（*https://www.codecademy.com/tracks/web*）。

留意，與 jQuery 一樣，我們使用一個 CDN 來將 Paper.js 加入專案。

你可能已經開始發現，連結東西的順序很重要。我們之後會在 *main.js* 裡面使用 jQuery 與 Paper.js，所以必須先連結它們。它們彼此之間沒有依賴關係，所以將哪一個放在前面都沒有關係，但我習慣先加入 jQuery，因為在 web 開發領域中，很多東西都與它有關。

現在我們已經連結 Paper.js 了，我們必須稍微設定一下 Paper.js。這種在做事情之前需要重複編寫的程式稱為**樣板**（*boilerplate*）。在 *main.js* 裡面的 `'use strict'` 之後加入以下程式（如果你想要的話，可以移除 `console.log`）：

```
paper.install(window);
paper.setup(document.getElementById('mainCanvas'));

// TODO

paper.view.draw();
```

第一行程式會在全域範圍（第七章會介紹）安裝 Paper.js。第二行程式會將 Paper.js 指派給 canvas，準備 Paper.js 來繪圖。中間的 `TODO` 是實際做些有趣的事情的地方。最後一行會要求 Paper.js 在螢幕上實際畫一些東西。

現在可以放下樣板，我們來畫些東西！我們要先在 canvas 中間畫出一個綠色的圓圈。將“TODO”註解換成以下幾行：

```
var c = Shape.Circle(200, 200, 50);
c.fillColor = 'green';
```

重新整理瀏覽器，不出所料，你會看到一個綠色的圓圈。我們已經寫出第一個真正的 JavaScript 了。其實這兩行程式做了許多事情，但現在你只需要知道幾件重要的事情。第一行會建立一個圓圈**物件**，它是用三個**引數**來做這件事的：圓心的 *x* 與 *y* 座標，與圓的半徑。我們的 canvas 有 400 像素寬與 400 像素高，所以 canvas 的中心點是 (200, 200)。半徑 50 讓圓是 canvas 的寬高的八分之一。第二行程式會設定**填充**顏色，它與輪廓顏色不同（Paper.js 的說法稱之為 *stroke*）。你可以任意修改這些引數來試試。

自動執行重複的工作

假如你不是只想要加入一個圓，而是想用它們來填滿 canvas，以格狀來排列它們，要怎麼做？如果你讓每個圓小一些，直徑是 50 個像素，就可以在 canvas 放入 64 個。當然你可以複製已經寫好的程式 63 次，並且親自修改所有座標，讓它們以格狀排列，但這聽起來是項繁複的工作，不是嗎？幸運的是，電腦的專長正是這種重複的工作。我們來看一下，如何畫出均勻排列的 64 個圓。將畫出一個圓的程式換成：

```
var c;
for(var x=25; x<400; x+=50) {
   for(var y=25; y<400; y+=50) {
      c = Shape.Circle(x, y, 20);
      c.fillColor = 'green';
   }
}
```

重新整理瀏覽器之後，你就會看到 64 個綠色的圓了！如果你是程式新手，可能不瞭解剛才寫的程式，但你可以看到它比親手寫 128 行程式來做同樣的事情還要好。

我們剛才使用所謂的 `for` 迴圈，它是一種控制流程語法，第四章會詳細說明。你可以在 `for` 迴圈指定一個初始條件（25），一個結束條件（少於 400），與一個遞增值（50）。我們在一個迴圈裡面加入另一個迴圈，來完成 x 軸與 y 軸。

我們可以用很多種方式來編寫這個範例。我們編寫的方式，是讓 *x* 與 *y* 座標變成最重要的資訊：我們明確地指定圓開始的地方，以及它們之間的間隔。我們可以用另一種做法來解決這個問題：我們可以指明我們要的圓圈數目（64），讓程式計算如何排列它們，將它們放在 canvas 上。之所以採用這種做法，是因為它比較可以與剪下貼上圓圈程式 64 次，並親自找出排列方法的做法比較。

處理使用者輸入

到目前為止，我們寫的程式並沒有要求使用者輸入任何東西。使用者可以按下圓圈，但它不會有任何效果，試著拖曳圓圈也沒有效果。我們來加入一些互動性，讓**使用者**可以選擇繪製圓圈的位置。

熟悉使用者輸入的**非同步**性質是很重要的。**非同步事件**是你完全無法控制發生時機的事件。使用者按下滑鼠是一種非同步事件：你無法知道他們何時按下按鍵。當然你可以提示他們按下滑鼠，但他們實際按下滑鼠的時機，以及是否按下，都是他們自己決定的。

使用者輸入產生的非同步事件是很直觀的，但之後的章節會討論較不直觀的非同步事件。

Paper.js 使用一種稱為 *tool* 的專案來處理使用者輸入。如果你不明白它為什麼要使用這個名字，你是對的：我同意，也不明白為何 Paper.js 的開發者要使用這個名字[5]。或許你可以在心中將“工具”換成“使用者輸入工具”。將畫出格狀排列圓圈的程式換成以下程式：

```
var tool = new Tool();

tool.onMouseDown = function(event) {
   var c = Shape.Circle(event.point.x, event.point.y, 20);
   c.fillColor = 'green';
};
```

這段程式的第一個步驟是建立 tool 專案。完成後，我們可以指派一個**事件處理器**給它。這裡的事件處理器稱為 onMouseDown。當使用者按下滑鼠時，**就會呼叫我們指派給這個處理器的函式**，這是很重要的地方，你一定要瞭解。在之前的程式中，這段程式會馬上執行：當我們重新整理瀏覽器時，綠圓就會自動出現。但這個範例不會出現這種事情，它不會在螢幕的某處畫出一個綠色圓圈。在 function 後面的大括號裡面的程式，**只會在使用者在 *canvas* 上按下滑鼠時執行**。

事件處理器會為你做兩件事：它會在滑鼠被按下**時**執行你的程式，並告訴你滑鼠被按下的**地方**。滑鼠位置會被存在一個引數的屬性：event.point，它有兩個屬性：x 與 y，說明滑鼠被按下的地方。

我們可以稍微節省一點打字數，直接將 point 傳給圓（而不是分別傳遞 *x* 與 *y* 座標）：

```
var c = Shape.Circle(event.point, 20);
```

5 技術校閱 Matt Inman 認為 Paper.js 的開發者或許曾經用過 Photoshop，並熟悉“hand tool”、“direct selection tool”等等。

這說明一個非常重要的 JavaScript 觀念：它可以知道被傳入的變數的資訊。在之前的案例中，如果它看到三個連續的數字，就會知道它們分別代表 *x*、*y* 座標與半徑。如果它看到兩個引數，就會知道第一個是 point 物件，第二個是半徑。我們會在第六章與第九章進一步學習這個部分。

Hello, World

最後，我們來重現 Brian Kernighan 在 1972 年的範例。我們已經完成所有粗重的工作了，接下來只要加入文字。在你的 `onMouseDown` 處理器之前，加入：

```
var c = Shape.Circle(200, 200, 80);
c.fillColor = 'black';
var text = new PointText(200, 200);
text.justification = 'center';
text.fillColor = 'white';
text.fontSize = 20;
text.content = 'hello world';
```

我們建立另一個圓，它是文字的背景，接著創造文字物件（`PointText`），指定繪製它的地方（螢幕中央），與一些其他的屬性（對齊、顏色與大小），最後，指定實際的文字內容（“hello world”）。

這不是我們第一次用 JavaScript 來發送文字，我們之前曾經用 `console.log` 第一次做這件事。我們當然可以將那個文字改成“hello world”。這種做法有許多地方比較像 1972 年的做法，但這個範例的目的，不在文字或它的呈現方式，而是要讓你建立一種自動化、有明顯效果的東西。

使用這段程式，重新整理瀏覽器之後，你就會看到歷史悠久的“Hello, World”範例。如果這是你的第一個“Hello, World”，歡迎你加入俱樂部。如果不是，希望這個範例可以讓你稍微瞭解 JavaScript。