

為什麼選擇 JavaScript ？

有數百種程式語言可供選擇，但以下因素讓 JavaScript 顯得獨一無二。最重要的是它與網頁瀏覽器的關係，例如 Google Chrome、Safari、Microsoft Edge 和 Firefox。JavaScript 幾乎可以在每個網頁瀏覽器上執行，這意味著在 JavaScript 編寫的程式碼，可以在任何能連上網頁瀏覽器的電腦執行，無需安裝任何額外特殊軟體。幾乎所有智慧型手機的瀏覽器也都能執行 JavaScript，所以可能現在你的口袋或包包裡，就有一個能支援 JavaScript 的瀏覽器。

由於它與網頁瀏覽器的關係，JavaScript 是網頁開發中極其重要的一部分。如果一個網站包含動態、互動式功能，它很可能就是用 JavaScript 建立的。例如，YouTube 使用 JavaScript 在你懸停在縮圖上時顯示影片預覽，Threads 使用 JavaScript 在你向下滾動頁面時載入更多貼文，Amazon 使用 JavaScript 來支援其「Look Inside」功能。

除了使用在網頁瀏覽器之外，JavaScript 還廣泛用於網站的後端，即在伺服器上執行的網站程式碼一部分，向使用者提供內容，相對於直接在使用者設備上執行的前端程式碼，實現方式的技術為 Node.js，許多知名網站都有一個 Node.js 後端，讓你可以網站前端和後端使用相同語言，甚至在兩者之間共享程式碼。

最後，JavaScript 已經成為各種應用程式中非常流行的指令碼語言，從可以自動化圖片處理的 Photoshop，到可以添加自動化整理電子郵件的 Gmail。有了一些 JavaScript 知識，你可以根據自己的需求客製化這些應用程式！

儘管這門語言在許多領域都能發揮作用，但本書將僅專注於以瀏覽器為主的 JavaScript。原因如下，首先，正如先前所提，在瀏覽器執行 JavaScript 的一個絕佳優勢，是無需安裝任何特殊軟體即可開始，我不想以在電腦上安裝 Node.js 這樣乏味的章節來開始這本書，因為很有可能寫完之後馬上就過時了。其次，儘管幾乎所有網站都可以使用 JavaScript 來進行前端開發，但 JavaScript 只是寫後端程式碼的眾多可能語言之一。總之，瀏覽器無疑是學習 JavaScript 最適合的環境。

一旦完成本書的相關學習並具有一些經驗後，我強烈建議你繼續了解 Node.js 和 JavaScript 的其他用途，這本書應該是你的 JavaScript 教育起點，而非終點。至於完成本書後的下一步，可參考後記。

PART I

語言

本書的第一部分會介紹 JavaScript 語言的基礎知識，帶你了解 JavaScript 程式的所有基本建構，包括許多對任何程式語言來說都息息相關的核心概念。

首先，我們將介紹兩種編寫和執行 JavaScript 程式碼的方式：在瀏覽器的控制台中或在文字編輯器中（第 1 章）。然後，你將學習使用變數和常數來表示個別資料片段，以及藉由陳述式和表達式來操作這些資料的方法（第 2 章）。接下來，你將看到使用 JavaScript 的複合資料型別，即陣列和物件，來將個別的值組合成更有意義的結構（第 3 章）。

本部分的後幾章將說明為程式碼加入結構的方法。首先，透過條件和迴圈將邏輯加入到程式中，這兩者能允許你做出決定，並控制程式碼執行的流程（第 4 章）。然後，你將學習使用函式（第 5 章）和類別（第 6 章），來重複使用和組織程式碼的技巧。

3

複合資料型別



前

一章討論了代表一個單一資料，例如數字或字串的 JavaScript 基本資料型別，現在要來看看 JavaScript 的複合資料型別 (*compound data types*)，陣列和物件，它們能將多個資料組合成一個單位，這是程式設計的重要組成部分，因為它能允許組織和處理任意大小的資料集合。你將學習建立和操作陣列及物件的方法，以及將它們組合成更複雜的資料結構辦法。

陣列

JavaScript 的陣列 (*array*) 是一種儲存有序的值清單的複合資料型別。陣列的元素可以是任何資料型別，不必全部都相同，但大部分時候是相同的。例如，一個陣列可以作為待辦事項清單，儲存一系列描述需要執行任務的字串，或者儲存一組表示從特定位置定期讀取的溫度讀數的數字集合。

陣列非常適合這類結構，因為它會將相關的值收集在一個地方，並且具有隨著值的增加或移除而增加和縮小的靈活性。如果你有固定數量的待辦事項，

比如說 4 個，則可能會使用單獨的變數來儲存它們，但使用陣列可以儲存無限數量的變化專案，同時保持固定順序。此外，一旦將元素放在一個陣列中，就可以編寫程式碼來有效地逐一操作陣列中的每個專案，這會在第 4 章介紹。

建立和索引

要建立一個陣列，請在一對中括號內列出其元素，並以逗號分隔：

```
let primes = [2, 3, 5, 7, 11, 13, 17, 19];  
primes;  
▶ (8) [2, 3, 5, 7, 11, 13, 17, 19]
```

這個陣列包含前 8 個質數，並儲存在變數 `primes` 中。輸入 `primes;` 時，Chrome 控制台應該顯示陣列的長度（8）以及它的元素。

每個陣列中的元素都有一個與之相關聯的索引編號。如同字串，陣列是從 0 開始索引，所以第 1 個元素位於索引 0，第 2 個位於索引 1，依此類推。要存取陣列的單個元素，請在陣列名稱後面的中括號中放入其索引號，例如以下存取 `primes` 陣列的第 1 個元素：

```
primes[0];  
2
```

由於陣列從 0 開始索引，因此陣列最後一個元素的索引會比陣列長度小一，因此，這個具有 8 元素的 `primes` 陣列，其最後 1 個元素位於索引 7：

```
primes[7];  
19
```

如果不知道陣列有多長，但又想取得最後一個元素，一開始可以使用點（dot）符號來存取其 `length`（長度）屬性，以尋找陣列長度，可見第 2 章的字串範例：

```
primes.length;  
8  
primes[7];  
19
```

或者，為了在一個陳述式中完成這一操作，可以簡單地從長度中減去 1 來取得最後一個索引的元素，如下所示：

```
primes[primes.length - 1];  
19
```

如果你使用的索引超出陣列範圍，JavaScript 會回傳 `undefined`：

```
▶1: (3) ['', '', '']
▶2: (3) ['0', '', '']
length: 3
▶[[Prototype]]: Array(0)
```

現在棋盤的左下角有一個 0。

總結一下，如果你想存取巢狀陣列中的元素，使用一組中括號選擇外層陣列中的元素，這將回傳內層陣列之一，然後再使用第二組中括號選擇內層陣列中的元素。

小試身手

3-1. 使用 `ticTacToe` 陣列與自己玩一場井字遊戲。記住，第一個索引號應該是棋盤的列，第二個索引號應該是欄。

陣列方法

JavaScript 提供幾個有用的方法來操作陣列，本節將介紹一些重要的方法。其中一些方法會修改陣列，稱為變異（*mutation*），相關例子包括增加或刪除陣列元素，或更改元素順序。其他方法則建立並回傳一個新陣列，而不改變原始陣列，這在仍需要原始陣列執行其他操作時非常有用。

了解使用的方法是否會變異陣列很重要。例如，假設有一個按時間順序列出的月份陣列，但程式中的某部分不小心更改了原始按時間順序排列的陣列，這會讓程式的其他部分誤以為 4 月是每年的第一個月。另一方面，如果有一個代表待辦事項清單的陣列，可能希望當增加或刪除任務時，都能更新本身的原始陣列，而不是建立一個新陣列。

向陣列加入元素

`push` 方法會將提供的元素加入到陣列尾端，以變異陣列，此方法的回傳值就是陣列的新長度。以下是使用 `push` 方法建構一個程式語言陣列的例子：

```
let languages = [];
languages.push("Python");
1
languages.push("Haskell");
2
languages.push("JavaScript");
3
languages.push("Rust");
```

PART II

互動式 JavaScript

在掌握這門語言的基礎之後，本書第二部分將展示使用 JavaScript 與網頁瀏覽器的互動方式。這將有效擴展語言的應用範圍，而能夠撰寫出互動性強、圖形化的網頁應用程式，例如遊戲、網上商店，甚至 Facebook 等。

首先，你將探索 HTML 和 CSS，這兩者是除了 JavaScript 以外，網頁開發的兩大支柱；也會看到使用 DOM 和 JavaScript 動態地修改網頁內容和外觀（第 7 章）。接著，你將練習編寫 JavaScript 程式碼，以回應瀏覽器中由使用者驅動的事件，如按鍵和滑鼠點擊（第 8 章）。最後，你將學習使用 JavaScript 和 Canvas API 產生靜態和動畫圖形（第 9 章）。

9

Canvas 元素



HTML 中，其中一個比較具互動性的元素是 `canvas`。這個元素就像畫家的畫布（`canvas`）一樣：能提供一個空間，可以使用 JavaScript 在瀏覽器視窗內繪製圖片。

更重要的是，透過不斷清除舊圖片並繪製新圖片，也可以在 `canvas` 上建立動畫。從這個意義上來說，`canvas` 元素更像電影院的大銀幕，圖片會每秒更新多次以創造運動的效果。

你將在本章學習建立 `canvas` 元素，以及使用 Canvas API 的方法，之後就可以藉由 JavaScript 使用 `canvas`，以編寫 JavaScript 的方式在 `canvas` 上繪製靜態圖片，然後，可以建構一個簡單的互動式繪圖應用程式，最後，會學習到在 `canvas` 上建立 2D 動畫的基礎知識。

建立 Canvas

要在網頁上包含 `canvas` 元素，只需將其加入到頁面 *index.html* 檔案中的 `body` 元素。你只需要起始和結束的 HTML 標籤 `<canvas></canvas>` 即可，因為 `canvas` 元素沒有任何必要的屬性。然而，幫 `canvas` 設定一個 `id` 會是個好主意，這樣就可以利用 JavaScript 輕鬆存取，同時，通常也會設定元素的 `width` 和 `height` 屬性，以訂下 `canvas` 的大小。

出現在 `canvas` 上的圖片是使用 JavaScript 而非 HTML 所產生的。任何在起始和結束標籤之間的 HTML 內容，只會在瀏覽器不支援 `canvas` 元素時顯示，因此可以作為舊版或僅限文字的瀏覽器備用方案。

現在來建立一個包含 `canvas` 元素的 HTML 檔案，並包含一個連結到 JavaScript 檔案的 `script` 元素，好在該檔案中編寫程式碼，以在 `canvas` 上產生圖片。這一整章都將使用相同的 HTML 檔案來繪製不同類型的圖片。建立一個名為 *chapter9* 的新目錄，並在該目錄中建立一個名為 *index.html* 的新檔案。輸入以下範例 9-1 的內容。

```
<!DOCTYPE html>
<html>
  <head>
    <title>Canvas</title>
  </head>
  <body>
    <canvas id="canvas" width="300" height="300"></canvas>
    <script src="script.js"></script>
  </body>
</html>
```

範例 9-1：包含 `canvas` 元素的 *index.html* 檔案

這是令人熟悉的 HTML 樣板，類似於前幾章建立的 *index.html* 檔案，只是這次用的是 `canvas` 而不是 `h1` 元素。`width` 和 `height` 屬性以像素為單位指定 `canvas` 尺寸，預設情況下，`canvas` 是透明的，所以載入頁面時，實際上不會看到任何東西。

製作靜態繪圖

有了 `canvas` 元素後，就可以準備使用 JavaScript 和 Canvas API 在其上繪圖。這裡從繪製實心矩形開始，再來將研究建立其他靜態繪圖的方式，在 *chapter9* 目錄中建立一個名為 *script.js* 的新檔案，並輸入範例 9-2 中顯示的程式碼。

與 Canvas 互動

若使用者可以與 `canvas` 互動時，`canvas` 會變得更加有趣。`canvas` 元素本身沒有內建的互動性，但是，可以編寫事件處理器，監聽某些使用者操作並觸發 Canvas API 方法來更新 `canvas`，從而增加互動性。

本節將使用帶有 `click`（點擊）處理器的 `canvas` 元素，建構一個非常基本的繪圖應用程式。處理器將監聽 `canvas` 上的點擊，並呼叫一個方法在點擊發生的位置繪製一個圓，還將建立一個滑桿，以便使用者可以設定圓的透明度，並加入一個按鈕來清除 `canvas`。

首先，加入必要的 HTML 元素來建立滑桿和按鈕，修改 `index.html` 如範例 9-7 所示。

```
<!DOCTYPE html>
<html>
  <head>
    <title>Canvas</title>
  </head>
  <body>
    <canvas id="canvas" width="300" height="300"></canvas>
    <div>
      <button id="clear">Clear</button>
      <input id="opacity" type="range" min="0" max="1" value="1" step="0.1">
      <label for="opacity">Opacity</label>
    </div>
    <script src="script.js"></script>
  </body>
</html>
```

範例 9-7：在 `index.html` 中加入一些額外的元素

這裡加入一個新的 `div` 元素，其中包含 3 個其他 HTML 元素。`div` 元素的作用，是將其中的元素分成一組，並將它們定位在 `canvas` 下方；如果沒有 `div`，它們就會出現在 `canvas` 的右側。

`div` 內的第一個元素是 `button` 元素，它建立一個可點擊的按鈕，任何位於起始和結束標籤之間的內容都將顯示為按鈕上的文字，因此按鈕將顯示 `Clear` 文字。稍後會編寫一個 JavaScript 函式，在使用者點擊按鈕時清除 `canvas` 上的所有圓。

`div` 內的下一個元素是 `input` 元素，用於從使用者那裡取得值。`input` 元素不允許任何子元素，因此它不需要結束標籤，在這種情況下，它的型別為 `range`，表示它將顯示為滑桿，並用於設定在 `canvas` 上繪製新的圓形透明度。它有幾個屬性定義其功能：`min` 定義滑桿將產生的最小值，`max` 定義最大值，`value` 定

然後用這個值更新 `opacity` 變數。由於 `drawCircle` 函式使用 `opacity` 的值作為 `RGBA` 顏色的 `alpha` 組件，因此任何新的圓，都將使用 `Opacity` 滑桿設定的最新值。

現在重新載入瀏覽器中的 `index.html`，應該有一個功能雖然基本但完整的繪圖應用程式！可以使用 `Opacity` 滑桿來更改新圓的透明度，並使用 `Clear` 按鈕清除 `canvas` 以重新開始繪製。試著將 `Opacity` 滑桿設定到一半，繪製重疊的圓，以查看它們的疊加方式。

小試身手

- 9-4.** 在 0 到 255 範圍內，加入控制顏色的 `R`、`G` 和 `B` 組件的滑桿。也可以加入一個 `Radius`（半徑）滑桿，來控制 `drawCircle` 函式中繪製的圓半徑。
- 9-5.** 建立一個名為 `drawSquare` 的新函式，在某個點上繪製一個以該點為中心的正方形，並在點擊處理器中呼叫該函式，而不是 `drawCircle` 函式。

動畫 Canvas

正如本章開頭所述，可以在 `Canvas` 上繪製每秒多次更新的圖片，來為 `Canvas` 增加動畫效果。本節就將編寫一個非常簡單的動畫來說明其基本原理。

動畫 `Canvas` 通常遵循以下基本模式：

1. 更新狀態
2. 清除 `Canvas`
3. 繪製圖片
4. 等待短時間
5. 重複

這裡的狀態（*state*），是指儲存有關動畫當前幀（*frame*）資訊的變數，這可能是移動物體的當前位置、物體移動的方向等等，在這裡的案例中，狀態將是圓的 `x` 和 `y` 座標。更新狀態時，將 `x` 和 `y` 座標分別遞增 1，意味著圓的位置將逐漸向右下方移動。繪製圖片將包括在更新的 `x` 和 `y` 座標上繪製一個小圓，在繪製圓之前清除 `Canvas`，以確保移除上一幀的圖片。最後兩步：等待和重複，將使用 `setInterval` 函式每 100 毫秒或每秒 10 次呼叫程式碼來完成。

PART III

專案

在學會 JavaScript 的基礎知識，和一些在網頁中使用 JavaScript 來建立互動應用程式的強大技術之後，是時候建構一些實際專案來運用這些技術了！

本書這部分會介紹 3 個重要專案，將幫助你練習所學的知識，說明開發和組織更複雜程式的方式，並體驗 JavaScript 所能實現許多令人興奮之事。這些專案之間沒有先後關係，因此不用按照一定順序，以下是每個專案的簡要預覽：

專案 1：建立遊戲

在這個專案中，你將製作經典遊戲 *Pong*（乓）的自製版本，擴充對資料結構、條件陳述式、函式和 Canvas API 的知識。你將開發兩個遊戲實現版本：一個使用獨立函式（第 10 章），另一個是使用類別和物件導向設計原則（第 11 章）；將展示不同的程式碼組織方式。

專案 2：製作音樂

這個專案介紹基本音訊程式和聲音合成技術（第 12 章），然後利用這些技術以 JavaScript 寫一首歌（第 13 章）。你將學習 Web Audio API 和 Tone.js 程式庫，這些工具能簡化聲音產生的過程。這將使你練習將第三方 JavaScript 程式庫整合到你的應用程式中。

專案 3：資料視覺化

這個專案將指導你使用流行的 D3 JavaScript 程式庫開發互動資料視覺化，帶你探索以可縮放向量圖形（SVG）繪圖的方式，並使用 D3 將 SVG 圖形組織成長條圖（第 14 章）。然後，使用藉由 GitHub 搜尋 API，從網際網路擷取的資料建構動態視覺化（第 15 章）。將 API 請求整合到應用程式中是一個極為常見的程式任務，因此這個專案將讓你準備好編寫與真實資料對接的程式碼。

你可以在 <https://codepen.io/collection/ZMjYLO> 找到所有這些專案的完整可下載程式碼，以及「小試身手」練習的解決方案。

13

寫一首歌



學會關於 Tone.js 和聲音合成的基本知識後，就足以寫出一首簡單的歌曲。這首歌曲可以由幾種樂器組成：前一章開發的鼓，喇叭取樣器，兩部分不同的合成貝斯，以及另一個合成器彈奏的一些和弦。

開始組織

這首歌曲將重複使用很多前一章的程式碼，但會重新組織，使其更容易理解歌曲建構方式。*index.html* 檔案將與第 12 章完全相同，但會從頭開始編寫一個新的 *script.js* 檔案，並將其分為 4 個邏輯部分：

樂器 (instruments) 用於實例化和設定樂器

排序 (sequencing) 用於建立要播放的迴圈音符序列

歌曲 (song) 用於安排每個序列開始和結束時間

事件處理 (event handling) 處理 click 事件以開始播放歌曲的程式碼

以下將使用多行註解來分隔這四個部分，讓 *script.js* 檔案更容易瀏覽。範例 13-1 說明這些註解的樣式，現在可以按照這個順序將它們加入到檔案中。

```
//////////  
// Instruments //  
//////////  
  
//////////  
// Sequencing //  
//////////  
  
//////////  
// Song //  
//////////  
  
//////////  
// Event Handling //  
//////////
```

範例 13-1：標示 *script.js* 主要部分的註解

在這一整章中，隨著歌曲的逐步建構，會將每個新程式碼片段加入到特定部分最後，這些註解就能幫忙快速找到新程式碼應該放的地方。

事件處理

從編寫 *script.js* 的事件處理部分開始。這段程式碼幾乎與上一章開頭編寫的程式碼相同：建立一個 `click` 事件監聽器，當使用者點擊按鈕時，切換播放（Play）按鈕和「播放中」（Playing）段落的樣式，並呼叫 *Tone.js* 以開始播放歌曲。將範例 13-2 的內容輸入到程式碼事件處理部分。

```
--snip--  
//////////  
// Event Handling //  
//////////  
  
let play = document.querySelector("#play");  
let playing = document.querySelector("#playing");  
  
play.addEventListener("click", () => {  
  // Hide this button  
  play.style = "display: none";  
  playing.style = "";  
  
  Tone.start();  
  
  // Modify this to start playback at a different part of the song  
  ❶ Tone.Transport.position = "0:0:0";
```

```
Tone.Transport.start();
});
```

範例 13-2：事件處理程式碼

與範例 12-2 的程式碼相比，這段程式碼有一個重要區別，❶ 使用 `Tone.Transport.position` 在呼叫 `Tone.Transport.start` 之前設定傳輸的起始位置。這裡將起始位置設為預設值 `"0:0:0"`，因此這個呼叫不具必要性。然而，若包含這行程式碼，這樣不想每次加入新元素時都要從頭聽整首歌曲，就可以輕鬆修改起始位置。例如，想跳過前 20 小節，可以將 `Tone.Transport.position` 的值更改為 `"20:0:0"`。

與上一章不同，所有建立樂器和序列的程式碼將位於事件處理器之外，這些程式碼可以在使用者按下播放（Play）按鈕之前執行。只有 `Tone.start` 呼叫必須位於處理器內，才能使歌曲正常運作。甚至可以將 `Tone.Transport` 行移到處理器之外，但在 `Tone.start` 之後的處理方式比較自然。

製作鼓聲

現在，來建立為歌曲鋪底的鼓聲，可使用在上一章建立的同樣高音鼓、小鼓和底鼓聲音。首先，宣告這些樂器，如範例 13-3 所示，將這段程式碼加入到 `script.js` 的樂器部分。

```
//////////
// Instruments //
//////////

❶ function mkDrums() {
  let reverb = new Tone.Reverb({
    decay: 1,
    wet: 0.3
  }).toDestination();

  let hiHatFilter = new Tone.Filter(15000, "bandpass").connect(reverb);

  let hiHat = new Tone.NoiseSynth({
    envelope: {
      attack: 0.001, decay: 0.1, sustain: 0, release: 0
    },
    volume: -6
  }).connect(hiHatFilter);

  class Snare {
    constructor() {
      this.noiseFilter = new Tone.Filter(5000, "bandpass").connect(reverb);
      this.noiseSynth = new Tone.NoiseSynth({
        envelope: {
```