前言

我的 TypeScript 之旅不是直接速成學會的。一開始在校園時，主要撰寫 Java，然後是 C++，就如同許多學習靜態型別語言的開發人員一樣，認為 JavaScript「只是」一種隨性在網站上執行的小型指令稿語言。

在這個語言中，我的第一個實質性專案是純用 HTML5/CSS/JavaScript 對原作**瑪利歐兄弟**電視遊戲進行的重製。並且，在許多早期專案中，是典型的一團糟。在專案開始時，打從心底不喜歡 JavaScript，它帶有奇怪的靈活性，並且缺乏防護的機制。直到專案接近尾聲時，才真正開始尊重 JavaScript 的特性和古靈精怪：它作為一種靈活性的語言，串連許多小功能的能力，以及在幾秒鐘內讓使用者瀏覽器載入網頁的**工作能力**。

當我完成第一個專案時，已經愛上 JavaScript。

諸如 TypeScript 之類的靜態分析（無須執行即可分析程式碼的工具）一開始也讓我感到不舒服。*JavaScript* **是如此輕鬆流暢，為什麼要讓自己陷入僵化的結構和型態之中？**是不是又回到了 Java 和 C++ 世界？

回到剛剛的專案，我花了 10 分鐘的時間努力閱讀舊有、令人思索的 JavaScript 程式碼，才理解如果沒有靜態分析，事情會變得多麼混亂。清理程式的行為表現出，可以從某種結構中獲得益處，進而套用在所有地方。從那時起，我就開始迷上靜態分析，並盡可能在專案中加入。

距離第一次使用 TypeScript 已經快十年了，我一如既往喜歡它。這個語言仍在不斷發展新的功能，並且為 JavaScript 提供**安全**和**結構**方面的特性，比過去來得更加有用。

希望透過閱讀《TypeScript 學習手冊》，使用者可以像我一樣學會欣賞 TypeScript。這不僅是一種搜尋錯誤和錯別字的方法，當然也不是對 JavaScript 做任何實質性程式碼的變更；而是作為帶有型別的 JavaScript：一個更完整的系統，用來宣告 JavaScript 應該如何工作，並協助我們堅持下去。

誰應該讀這本書

如果讀者撰寫 JavaScript 程式碼，可以在終端介面中執行基本命令，並且有興趣了解 TypeScript，那麼本書適合你。

也許你曾聽說過 TypeScript 可以幫助使用者撰寫大量 JavaScript 並減少錯誤（沒錯！），或妥善記錄使用者的程式碼來供其他人閱讀（這也是正確的！）。也許讀者已經注意到，現在有很多 TypeScript 的職缺，或者讀者也正在這個職位之中。

無論出於何種原因，只要理解 JavaScript 的基礎知識 —— 變數（Variable）、函數（Function）、閉包（Closure）/ 作用域（Scope）和類別（Class），這本書將帶領讀者從沒有 TypeScript 知識，到掌握這個語言的基礎知識及最重要的特性。讀完本書會明白：

- TypeScript 在「原生」的 JavaScript 之上有何歷史和背景
- 型別系統如何打造模組程式碼
- 型別檢查如何分析程式碼
- 如何使用開發型別註記來知會型別系統
- TypeScript 如何與 IDE（整合開發環境）一起提供程式碼檢索和重構工具

而讀者將能夠：

- 闡述說明 TypeScript 的優勢及其型別系統的一般特徵。
- 在程式碼中在使用的地方，增加型別註記。
- 使用 TypeScript 的內建推斷和新語法，來妥善表示複雜的型態。
- 使用 TypeScript 協助本地端開發重構程式碼。

為什麼寫這本書

TypeScript 是開放原始碼和產業界都廣受歡迎的語言：

- 觀察在 GitHub 平台中，該程式語言，在 2017 年排名第十，在 2019 和 2018 年排名第七，在 2021 和 2020 年之間，躍升到第四。
- 在 StackOverflow 的 2021 年開發者調查中，將其列為世界第三大最受歡迎的程式語言（擁有 72.73% 的使用者）。
- 2020 年 JS Survey Status 網站顯示，以 TypeScript 作為建構工具和 JavaScript 的變體程式，一直保有很高的滿意度和使用量。

TypeScript 對於前端開發人員，在所有主要的 UI 程式庫與框架中，都得到很好的支援，強烈建議使用 TypeScript 的包括 Angular，以及 Gatsby、Next.js、React、Svelte 和 Vue。TypeScript 對於後端開發人員，可產生 JavaScript 並在 Node.js 原生環境中執行；而同樣是 Node 作者所開發的 Deno，則在執行時強調，直接支援 TypeScript 檔案。

然而，儘管受到如此多的專案歡迎與支持，但作者第一次學習這門語言時，對於缺乏良好的線上內容性介紹，感到相當失望。許多線上文件資源，並沒有妥善地解釋什麼是「型別系統」或如何使用它。它們通常假設讀者擁有大量 JavaScript 和強型別語言的先行知識，或者只是粗略的撰寫程式碼範例。

這幾年以來沒有看到一本 O'Reilly 的書籍，用可愛的動物封面介紹 TypeScript，這很令人失望。雖然在本書之前，已經有很多其他出版商（包括 O'Reilly）出版關於 TypeScript 的書籍，但卻找不到一本完全符合作者想要以語言為基礎的書：針對它的工作方式、核心功能，如何協同工作進行討論。這本書從一開始對語言的基礎做解釋，然後逐一增加語言特性。身為作者，很高興能夠為還不熟悉 TypeScript 原理的讀者，做清晰而全面地介紹 TypeScript 語言基本知識。

瀏覽本書

學習 TypeScript 有兩個目的：

- 讀者可以很透徹理解整個 TypeScript。
- 接著，可以將其作為 TypeScript 語言入門的實用參考。

本書從概念到實際使用，分為三個部分：

- 第一部分，「概念」：TypeScript 為 JavaScript 增加了什麼？以及 TypeScript 以型別系統（*type system*）為基礎建立後，是如何產生 JavaScript？
- 第二部分，「特點」：緊實的型別系統是如何在編輯 TypeScript 程式碼時，使用 JavaScript 的主要部分進行切換。
- 第三部分「使用」：既然讀者理解，構成 TypeScript 語言的功能後，那麼如何在現實世界中使用它們，來改善程式碼閱讀和編輯體驗。

我已經在最後的第四部分「額外學分」中介紹，較少使用但偶爾仍會用到的 TypeScript 功能。讀者無須深入理解它們，即可將自己視為 TypeScript 開發人員。但它們都是有用的概念，將 TypeScript 用於實際專案時可能會出現。一旦完成前三個部分的理解後，我強烈建議讀者學習額外的部分。

每章都以一段俳句作為開始，用來深入說明其內容的精神，並以一段雙關語做結尾。整個 Web 開發和其中的 TypeScript 社群，以熱情和歡迎新人的參與而聞名。我試圖讓這本書，對於不喜歡冗長枯燥文字的學習者來說，讀起來會很愉快。

範例和專案

與許多其他介紹 TypeScript 資源不同的地方，本書透過顯示單一新資訊的獨立範例，來介紹語言特性，而不是深入研究中大型專案。我喜歡這種教學方法，因為它首先將焦點放在 TypeScript 語言上。TypeScript 在如此多的框架和平台上，都很有用——其中許多會進行定期 API 更新；我並不想在本書中含有特定於任何框架或平台的內容。

即便如此，在學習程式編譯語言，採用導入概念後立即練習它們，會顯得非常有效果。強烈建議在每個章節之後休息一下，多加練習其中的內容。每章的結尾都有建議參考的部分，可在 *https://learningtypescript.com* 網站上完成其中條列的範例和專案。

本書編排慣例

本書使用下列的編排方式：

斜體字（*Italic*）

表示專業用語、URL、電子郵件地址、檔案名和檔案副檔名稱。（中文使用楷體字）

第一章

從 JavaScript 到 TypeScript

今天的 *JavaScript*
支援瀏覽器數十年
網路的美好

在談論 TypeScript 之前，我們需要先理解它的來源：就是 JavaScript！

JavaScript 的歷史

JavaScript 是 1995 年由 Netscape 的 Brendan Eich，在 10 之天內設計出來的，它平易近人且容易使用於網站之中。從那以後，一直有開發人員在取笑它古怪與明顯的缺點。我將在下一節中，介紹其中的一些。

不過，自 1995 年以來，JavaScript 已經發生了巨大的變化！自 2015 年以來，其 TC39 指導委員會，每年都會發布新版本的 ECMAScript（JavaScript 的基礎語言規範），並使 JavaScript 具有與其他現代語言保持一致的新功能。令人印象深刻的是，即便使用正規新版本的語言，JavaScript 幾十年來也設法在不同的環境中保持向下相容性，包括瀏覽器、嵌入式應用程式和伺服器執行時。

如今，JavaScript 是一種非常靈活的語言，具有很多優勢。人們應該意識到，雖然 JavaScript 有其古怪，但也有助於實現 Web 應用程式和網際網路的驚人能力。

> 給我完美的程式編譯，我將呈現一種沒有侷限的語言。
>
> —Anders Hejlsberg, TSConf 2019

原生 JavaScript 的陷阱

開發人員通常將沒有使用任何重要的 JavaScript 語言擴充功能或框架，稱之為「原生」：意指接近原始的味道。我們很快就會討論到，為什麼 TypeScript 會添加正確的風格，來克服這些主要的特殊陷阱；理解為什麼它們會讓人痛苦，是很有用的。所有這些弱點都會隨著專案的規模越大、壽命越長而變得越明顯。

昂貴的自由

不幸的是，許多開發人員對 JavaScript 最大的不滿是它的主要特性之一：JavaScript 對程式碼結構幾乎沒有任何約束限制。這種自由使得以 JavaScript 開始專案時，變得非常有趣！

但是，隨著擁有越來越多的檔案，很明顯這種自由可能會造成破壞。從一個虛構的繪圖應用程式中擷取以下片段：

```
function paintPainting(painter, painting) {
  return painter
    .prepare()
    .paint(painting, painter.ownMaterials)
    .finish();
}
```

在沒有任何前後文的情況下閱讀這樣的程式碼，我們只能對如何呼叫 `paintPainting` 函數有模糊的想法。也許如果曾在周圍的相關程式碼中處理過，我們可能會記得 `painter` 應該是某個 `getPainter` 函數回傳的內容。甚至可以幸運的猜出 `painting`，所意指的是一個字串。

但是，即便這些假設是正確的，對程式碼之後的修改也可能會使它們失效。或許，`painting` 字串修改為其他資料型態，也或者可能被一個或多個畫家的實作方法，而被重新命名。

若是其他擁有編譯器的程式語言，為了確保程式碼正確性，可能會當機，而某些其他語言可能會拒絕執行程式碼。而動態型別語言並非如此，因此不檢查程式語言的正確性，而執行那些程式碼可能導致當機，例如 JavaScript。

當我們希望安全地執行程式碼時，會讓 JavaScript 如此自由有趣的程式碼，變成了真正的折磨。

鬆散的文件

JavaScript 語言規範中，並沒有任何內容可以正式描述程式碼中，函數的參數、回傳、內部變數或其他結構的含義。許多開發人員採用了一種稱為 JSDoc 的標準，來使用區塊註解描述函數及變數。JSDoc 的描述規範了如何編寫文件註解，這些註解直接放在函數和變數等結構之上，並以標準方式格式化。以下是一個只取出部分內容的範例：

```
/**
* Performs a painter painting a particular painting.
*
* @param {Painting} painter（畫家）
* @param {string} painting（作畫）
* @returns {boolean} 畫家是否畫了這幅畫
*/
function paintPainting(painter, painting) { /* ... */ }
```

JSDoc 有一些關鍵議題，這些議題常常是存在大型程式碼的資料中，造成使用上的不愉快：

- 沒有什麼能阻止 JSDoc 描述錯誤的程式碼。
- 即使我們的 JSDoc 描述之前是正確的，但在程式碼重構期間，也很難找出與現在我們所有修改相關的無效 JSDoc 註解。
- 描述複雜對象，顯得既笨重且冗長，需要多個獨立的註解，來定義型別及其關係。

跨越幾十個檔案，維護 JSDoc 註解是不會占用太多時間，但是跨數百甚至數千個，不斷更新的檔案可能是一件真正的苦差事。

較弱的開發人員工具

由於 JavaScript 沒有提供辨識型別的內建方法，而且程式碼很容易與 JSDoc 註解產生分歧，因此很難自動對程式碼資料進行大規模的修改或深入理解程式碼。JavaScript 開發人員經常驚訝地看到，在 C# 和 Java 等型別化語言，允許開發人員執行類別成員重新命名或快速轉跳到宣告參數型別位置的功能。

讀者可能會提出異議，現代 IDE（例如 VS Code）確實提供一些開發工具，例如對 JavaScript 的自動重構。沒錯，但是在許多 JavaScript 功能的底層中，使用 TypeScript 或等效的工具；並且這些開發工具在大多數 JavaScript 程式碼中，不如定義良好的 TypeScript 程式碼中，來得可靠、強大。

TypeScript！

TypeScript 在 Microsoft 內部於 2010 年初所建立，然後於 2012 年發佈並開放原始碼。開發的負責人是 Anders Hejlsberg，他還領導開發流行的 C# 和 Turbo Pascal 語言。TypeScript 通常被描述為「JavaScript 的超集合」或「帶有型別的 JavaScript」。但什麼是 TypeScript？

TypeScript 有四大特性：

程式編譯語言（*Programming language*）

一種包含所有既定 JavaScript 語法的語言，以及用於 TypeScript 中，定義和使用型別的特定語法。

型別檢查（*Type checker*）

一個程式，它接收一組用 JavaScript 或 TypeScript 編輯的檔案，分析所有建立的結構（變數、函數……），並讓我們知道是否在設定上有任何的不正確。

編譯器（*Compiler*）

它是一個執行型別檢查的程式，匯報任何問題，然後輸出等效的 JavaScript 程式碼。

程式語言服務（*Language service*）

一個使用型別檢查通知編輯器（例如 VS Code），為開發人員提供有用的資訊及工具。

TypeScript Playground 入門

到目前為止，我們已經閱讀了大量有關 TypeScript 的內容。動手開始做吧！

主要的 TypeScript 網站裡頭，包含一個「遊樂場」的編輯器 *https://www.typescriptlang.org/play*。可以在常用的編輯器中，輸入程式碼並觀看結果、提出相關建議，這如同一些本地端完整 IDE（Integrated Development Environment，整合開發環境）中，使用 TypeScript 時，所看到的近似功能。

本書中的大多數程式片段都是刻意很小而且獨立的，以至於使用者可以在 Playground 中輸入，並加以修改後，獲得一些趣味性。

TypeScript 實戰

看看以下這個程式碼片段：

```
const firstName = "Georgia";
const nameLength = firstName.length();
//                           ~~~~~~
// 無法呼叫此運算式。
```

程式碼是用一般的 JavaScript 語法所撰寫的——目前還沒有介紹 TypeScript 的特定語法。如果讀者要在這段程式碼上執行 TypeScript 型別檢查，它會利用已知字串的長度屬性，會是一個數字而不是一個函數，並顯示出一些建議與評論，提供使用者參考。

若將此程式碼貼在 Playground 或編輯器中，語言服務會告訴使用者，在 `length` 下加註一條紅色的波浪線，表示 TypeScript 對使用者的程式碼有所異議。

將滑鼠游標停留在波浪符號的程式碼上，會出現一些文字訊息（如圖 1-1）。

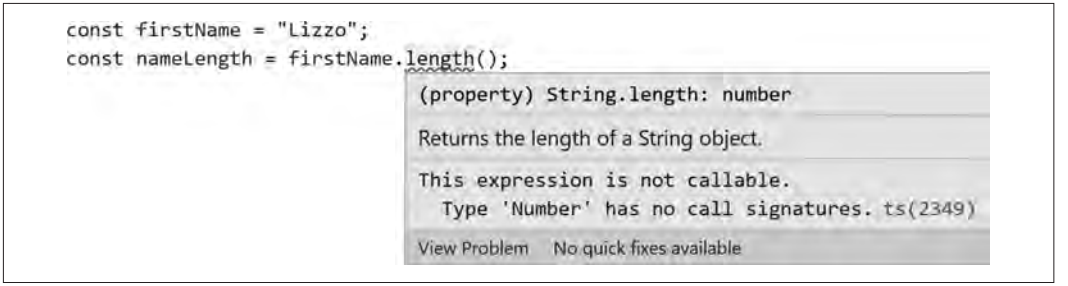

圖 1-1　TypeScript 報告關於字串長度無法呼叫的錯誤

在編輯器輸入時被告知這些簡單錯誤，總是比等到執行時遇到特定程式碼，並拋出錯誤要愉快得多。如果是在 JavaScript 中嘗試執行這樣的程式碼，會直接當機！

透過約束來獲得自由

TypeScript 允許我們可以指定參數和變數提供哪些型別的數值。一些開發人員發現，首先必須在程式碼中，限制明確寫出特定區域是如何運作。

但我認為這種被「約束」的方式，實際上是一件好事！我們透過程式碼約束，限制只能以所指定的方式來操作；TypeScript 可以讓使用者確信在某個程式碼區域中所做的修改，不會破壞其他使用到它的部分。

例如，如果我們修改函數所需參數的數量，TypeScript 會在忘記更新呼叫該函數的位置通知我們。

在下面的範例中，`sayMyName` 從原先接受兩個參數修改為接受一個參數，但是對應呼叫它的地方，仍使用兩個字串沒有更新，因此觸發 TypeScript 錯誤訊息：

```
// 之前：sayMyName(firstName, lastNameName) { ...
function sayMyName(fullName) {
  console.log(`You acting kind of shady, ain't callin' me ${fullName}`);
}

sayMyName("Beyoncé", "Knowles");
//                   ~~~~~~~~~
// 應有 1 個引數，但得到 2 個。
```

這段程式碼在 JavaScript 中執行不會當機，但其輸出與預期不同（其中不包括「`Knowles`」）：

```
You acting kind of shady, ain't callin' me Beyoncé
```

使用數量錯誤的參數，呼叫函數正是 TypeScript 約束 JavaScript 那種短視近利的自由。

精確的文件

再看一下先前的 `paintPainting` 函數，它的 TypeScript 版本。雖然還沒有詳細討論，善於記錄型別的 TypeScript 語法細節，但以下程式碼片段仍然可以非常精確地將程式碼記錄下來：

```
interface Painter {
  finish(): boolean;
  ownMaterials: Material[];
  paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean { /* ... */ }
```

第一次閱讀此程式碼的 TypeScript 開發人員可以瞭解，`painter` 至少具有三個屬性，其中兩個是方法。透過語法來描述物件的「形狀」，TypeScript 提供了一個優秀、強制的系統來描述物件的外觀。

更強大的開發人員工具

TypeScript 的型別允許使用 VS Code 等編輯器，更深入地瞭解我們的程式碼。然後，可以在輸入時，使用這些經過分析後所提出的智慧建議。這些建議對開發者而言相當有用。

如果你使用 VS Code 編輯過 JavaScript，可能已經注意到，當使用內建型別的物件（如字串）編輯程式碼時，它會提出建議，並「自動完成（autocompletion）」。例如，若開始輸入已知字串的成員文字，TypeScript 會顯示建議字串的所有成員（圖 1-2）。

圖 1-2　TypeScript 在 JavaScript 中，為字串提供自動補齊的建議

加入 TypeScript 的型別檢查來分析程式碼，可以為我們在撰寫過程時，提供這些有用的建議。在 `paintPainting` 函數中輸入 `painter.` 時，TypeScript 會知道 `painter` 參數是 `Painter` 型別，並且 `Painter` 型別具有以下成員（圖 1-3）。

圖 1-3　TypeScript 在 JavaScript 中，為字串提供自動補齊的建議

我們將在第十二章「使用 IDE 功能」中，大量介紹其他有用的編輯器功能。

編譯語法

TypeScript 的編譯器允許輸入 TypeScript 語法，並對其進行型別檢查，產生同等效果的 JavaScript 程式碼。為方便起見，編譯器還可以採用現代 JavaScript 語法，並將其編譯為較舊版的 ECMAScript 等效內容。

如果要將以下 TypeScript 程式碼，貼到 Playground 中：

```
const artist = "Augusta Savage";
console.log({ artist });
```

在 Playground 的右側螢幕會顯示編譯器等效 JavaScript 的程式碼輸出（圖 1-4）。

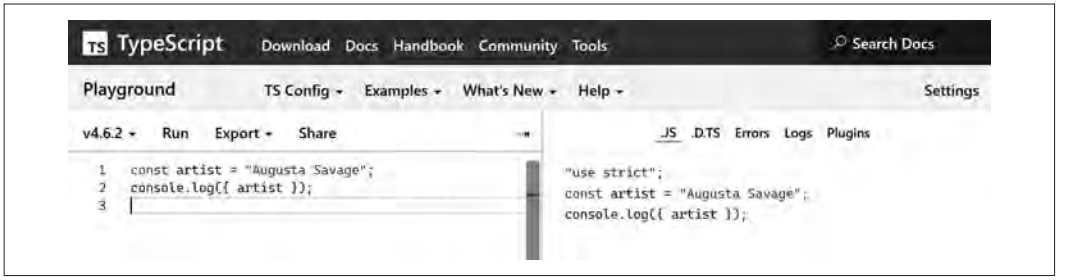

圖 1-4　Playground 將 TypeScript 程式碼編譯成等效的 JavaScript

TypeScript Playground 是呈現原始 TypeScript 如何成為轉換輸出成 JavaScript 的絕佳工具。

許多 JavaScript 專案使用個別的轉譯器，例如 Babel（*https://babeljs.io*）。而 TypeScript 是將自己的原始碼轉譯成可執行的 JavaScript。我們可以在 *https://learningtypescript.com/starters* 上，找到常見專案起始框架的列表。

本機執行入門

只要在本機電腦上安裝 Node.js，就可以在執行 TypeScript。如果要全域安裝最新版本的 TypeScript，請執行以下命令：

```
npm i -g typescript
```

現在，我們能夠使用 tsc（**T**ype**S**cript **C**ompiler）編譯器命令。嘗試使用 --version 確認設定配置：

```
tsc --version
```

無論安裝是否為 TypeScript 的最新版本，應該列印出類似版本編號 X.Y.Z 的內容：

```
$ tsc --version
Version 4.7.2
```

在本機端執行

現在已經完成安裝 TypeScript，接下來讓我們在建立一個需要執行 TypeScript 程式碼的資料夾。在電腦中的某處建立資料夾後，並執行以下命令，會建立一個新的 *tsconfig.json* 配置設定檔案：

```
tsc --init
```

tsconfig.json 檔案說明 TypeScript 在解析程式碼時所使用的設定。這個檔案中的大多數選項，在本書中並不會逐一介紹（程式語言在編譯中有許多不常見的特殊情況需要解決！）。我們將在第 13 章「配置設定選項」中介紹它們。目前重要的是我們可以執行 tsc，來告訴 TypeScript 編譯該資料夾中的所有檔案，TypeScript 將參考 *tsconfig.json*，獲得任何配置選項。

嘗試添加一個名為 *index.ts* 的檔案，其內容如下：

```
console.blub("Nothing is worth more than laughter.");
```

然後，執行 tsc 並提供 *index.ts* 檔案名稱：

```
tsc index.ts
```

此時應該會收到一個如下所示的錯誤：

```
index.ts:1:9 - error TS2339: 型別 'Console' 沒有屬性 'blub'。

1 console.blub("Nothing is worth more than laughter.");
          ~~~~

找到 1 個錯誤。
```

實際上，console 中不存在 blub。

在我們修正程式碼之前，請注意 tsc 為我們建立了一個 *index.js*，其內容包括 console.blub。

這是一個重要的概念：即使我們的程式碼中有一個型別錯誤，在語法上仍然是完全有效的。TypeScript 編譯器仍會從輸入檔案生成 JavaScript，而不管任何型別錯誤。

修正 *index.ts* 中的程式碼，正確呼叫 `console.log`，並再次執行 `tsc`。在終端介面中應該沒有任何錯誤訊息，並且產生更新後的程式碼檔案 *index.js*：

```
console.log("Nothing is worth more than laughter.");
```

強烈建議讀者在閱讀本書各章節片段時，可在 Playground 上或在支援 TypeScript 的編輯器中，透過執行 TypeScript 語言服務操作它們。獨自練習小型及大型專案，有助於學習上的體驗，在 *https://learningtypescript.com* 上亦能學到的知識。

編輯器功能

建立 *tsconfig.json* 檔案的另一個好處是，當編輯器打開特定資料夾時，會將該資料夾辨識為 TypeScript 專案。例如，在資料夾中以 VS Code 打開，它會分析其中 TypeScript 程式碼的設定，將根據資料夾中的 *tsconfig.json* 檔案中的所有內容。

回顧本章中的程式碼片段，並在編輯器中輸入它們作為練習。當讀者輸入名稱時，應該會看到建議補齊名稱的下拉選單，尤其是對於諸如 `console` 的 `log` 之類成員。

倘若正在使用 TypeScript 語言服務來幫助自己撰寫程式碼，將非常令人高興。你正在成為一名 TypeScript 開發人員！

VS Code 提供很好的 TypeScript 支援，並且它本身是在 TypeScript 中所建構的。我們並非一定要在 VS Code 中使用 TypeScript ——幾乎所有現代編輯器對 TypeScript 都具有出色的支援，無論是內建或透過外掛程式來提供相關功能——但還是建議至少在閱讀本書後，嘗試在 VS Code 中使用 TypeScript。如果讀者使用其他編輯器，建議開啟所專屬的 TypeScript 支援功能。我們將在第 12 章「使用 IDE 功能」中，更深入地介紹編輯器功能。

哪些是 TypeScript 所無法處理的？

我們既然已經看到 TypeScript 的美妙之處，也必須警告讀者其中的一些限制。每個工具都在某些領域中有出色的表現，而在其他領域則存在某種局限性。

錯誤程式碼的補救措施

TypeScript 可以幫助我們建構 JavaScript，但除了強制型別安全之外，它不會提出任何關於該結構應該是什麼樣的內容提出意見。

TypeScript 是一種讓每個人都應該能夠使用的程式語言，而不是針對單一目標的小眾框架。我們可以在 JavaScript 中，使用的任何架構模式撰寫程式碼，TypeScript 亦將支援它們。

如果有人試圖告訴你 TypeScript 強迫使用類別，或者是很難寫出好的程式碼，又或者有任何程式碼風格的差異，請給他們一個嚴肅的表情，並告訴他們閱讀這本《TypeScript 學習手冊》。TypeScript 不強制執行程式碼樣式選項，例如是否使用類別或函數，也不與任何特定的應用程式框架（如 Angular、React 等）有任何相關連動。

作為 JavaScript 的（大部分）擴充

TypeScript 的設計目標，明確指出它應該：

- 與目前與未來的 ECMAScript 提案保持一致
- 保留所有 JavaScript 程式碼在執行時的行為動作

TypeScript 根本不會嘗試改變 JavaScript 的工作方式。創建者非常努力地避免會額外增加到 JavaScript 或造成 JavaScript 衝突的新功能程式碼。這樣的任務是 TC39 的範疇，TC39 是負責 ECMAScript 本身的技術委員會。

TypeScript 中有一些是多年前增加的舊有功能，來反映 JavaScript 程式碼中的常見案例。這些特性中，大多數要不是相對不常見到，就是已經失去原有的需求性，這在第 14 章「語法擴充」中會簡要說明。建議在大多數情況下避免使用它們。

截至 2022 年，TC39 正在研究為 JavaScript 增加型別註記的語法。最新的提議是將它們作為一種註解形式，在執行時不會影響程式碼，並且僅限用於 TypeScript 等系統開發時。在 JavaScript 中增加型別註記或類似的東西，還需要很多年，所以本書在其他地方不會提及它們。

執行速度比 JavaScript 慢

有時在網際網路上，我們可能會聽到一些固執己見的開發人員抱怨，TypeScript 在執行時比 JavaScript 慢。這種說法通常是不準確且具有誤導性。TypeScript 對程式碼所做的唯一修改是，如果要求它將程式碼編譯為早期版本的 JavaScript，用來支援較舊的執行環境時，例如 Internet Explorer 11。許多生產框架根本不會使用 TypeScript 的編譯器，而是使用單獨的工具進行轉譯（將原始碼從一種程式編譯語言轉換為另一種），並且讓 TypeScript 僅用於型別檢查。

然而 TypeScript 確實會增加一些時間來建構程式碼。在大多數環境（例如瀏覽器和 Node.js）在執行之前，必須將 TypeScript 程式碼編譯為 JavaScript。大多數建構管道的設定，通常會造成效能損耗可以忽略不計，並且較慢的 TypeScript 某些功能（例如分析程式碼，用來搜尋可能的錯誤）與生成可執行的應用程式碼檔案，可分開完成。

即使是看似允許直接執行 TypeScript 程式碼的專案，例如 ts-node 和 Deno，它們本身也會在執行之前，將在內部的 TypeScript 程式碼轉換為 JavaScript。

完成進化

網路的演進開發還未完成，而 TypeScript 也沒有。TypeScript 語言不斷收到錯誤修復與功能增加，來滿足 Web 社群不斷變化的需求。我們將在本書中，學習 TypeScript 的基本原則將保持不變，但錯誤訊息、更進階的功能和編輯器的整合，將隨著時間的演進而改變。

事實上，雖然這本書的最新版本是 TypeScript 4.7.2 版本，但當我們開始閱讀它時，可以確定已經發佈了一個更新的版本。在本書中的一些 TypeScript 錯誤訊息，甚至可能已經過時了！

總結

在本章中，將瞭解 JavaScript 的一些主要弱點，以及 TypeScript 的作用和如何開始使用 TypeScript：

- JavaScript 的簡史
- JavaScript 的缺陷：代價高昂的自由、鬆散的文件與功能較弱的開發工具

- 什麼是 TypeScript：程式編譯語言、型別檢查、編譯器和程式語言服務
- TypeScript 的優勢：透過約束、精確的文件和更強大的開發工具來實現自由
- 開始在 TypeScript Playground 和本機端電腦上撰寫 TypeScript 程式碼
- 哪些是對 Typescript 的誤解：對不良程式碼的補救、作為 JavaScript（大部分）擴充、執行速度比 JavaScript 慢，或是已完成的進化版本

現在我們已經閱讀完本章，在 *https://learningtypescript.com/from-javascript-to-typescript* 上，練習所學到的內容。

如果在執行 *TypeScript* 編譯器時發現錯誤？
最好動手抓住它們！