前言

我們經常高估或低估特定事件與計劃的影響。我深信我在 Google 的最後一個專案 Google Feedback 最終將完全改變公司與用戶互動的方式。我也認為 Angular（當時稱為 AngularJS）只是 Feedback 專案使用的管理介面框架。

事後來看結局完全相反。雖然許多 Google 產品還在使用 Feedback，但 Angular 從一個內部小專案變成世界各地的公司與數千個開發者使用的框架。這要歸功於 Misko、Igor、與整個團隊致力於改善我們開發網頁應用程式的方式。

它從兩個人的專案變成最大的開源社群以及數千個專案所使用的架構。關於 Angular 有很多書、教學、文章，且採用數量與資源每天都在增加。

Angular 的第一個版本具有一些超前時代的概念（例如資料連結、關注點分離、相依性注入等），如今已經是新框架普遍採用的功能。

AngularJS 生態系最大的變化是新版本（最初稱為 Angular 2.0，現在稱為 Angular）。它大幅的改變、不向後相容幾乎導致整個社群的分裂，但透過開放社群參與將可能是災難性的行動轉變成網頁開發的新時代。

真正讓 Angular 成功的是它的社群——對核心框架做出貢獻或開發外掛與日常使用它的人們。

身為社群的一份子，我很高興能以這本書作為我對社群的貢獻。

本書讀者

這本書寫給想要開始使用 Angular（2.0 或後續版本）並已經熟悉 JavaScript 與 HTML 的人，而學習 Angular 只需基本認識 JavaScript 就夠了，無需學習過 AngularJS 1.0。

我們也會使用 TypeScript，它是以 Angular 進行開發的推薦方式，對其有基本認識就足夠。

我們會逐步說明，因此可以放輕鬆跟著我愉快的學習。

為什麼要寫這本書

Angular 框架發展出很多功能，而背後的大社群提供了很多有幫助的資源。但這些資源不是專注於某個特定部分不然就是對初學者沒有幫助。

這本書致力於逐步指引 Angular 的上手，所有概念都有條不紊的循序加入。由於技術變化很快，這本書不打算討論所有面向而是專注於核心部分的仔細研究，以讓讀者能自行探索其餘部分。

讀完本書，你應該會熟悉 Angular 框架並能以 Angular 開發應用程式。

關於現在的網路應用程式開發

JavaScript 一路走來成為現在最常見的程式設計語言。如今開發者不太需要擔心瀏覽器間的不一致，而這是 jQuery 等框架出現的主要原因。

使用框架（例如 Angular 與 React）是開發前端的常見選擇，現在很少人會不用框架進行開發。

框架有很多優點，包括減少模板程式和提供一致的結構與佈局等。它的主要目標是減少浪費時間並專注於主要功能。若能跨瀏覽器（與平台，例如 Android、iOS、桌面等）則更好。

Angular（以及其他框架）透過下列基礎核心達成這樣的能力：

- 宣告式程式設計驅動的模板語法
- 模組化與分離關注點

- 資料連結與資料驅動程式設計
- 可測試性與測試支援
- 導向與導航
- 支持其他功能，包括伺服器端繪製與撰寫原生行動應用程式等！

Angular 讓我們能專注於建構良好體驗並管理複雜性。

本書內容安排

這本書逐步指引開發者學習 Angular。介紹新概念的章節後面會接著一章如何進行單元測試的說明，大致安排如下：

- 第 1 章 *Angular* **介紹**，介紹 Angular 以及它的概念與如何開始撰寫 Angular。
- 第 2 章 *Hello Angular*，逐步建構一個很簡單的 Angular 應用程式，並解釋每個部分的合作方式。它還介紹 Angular 的 CLI。
- 第 3 章**使用** *Angular* **內建指令**，討論基本的 Angular 內建指令（包括 `ngFor`、`ngIf` 等）與使用方式。
- 第 4 章**認識與使用** *Angular* **元件**，更深入的討論 Angular 元件與各種選項。它還討論元件的基本生命週期掛鉤。
- 第 5 章**測試** *Angular* **元件**，介紹如何使用 Karma 與 Jasmine 以及 Angular 測試框架進行單元測試。
- 第 6 章**使用模板驅動表單**，討論在 Angular 中建構與操作表單，特別是模板驅動的表單。
- 第 7 章**使用反應式表單**，討論定義與操作反應式表單的方式。
- 第 8 章 *Angular* **服務**，討論 Angular 服務，包括使用 Angular 內建的服務與自訂服務。
- 第 9 章**從** *Angular* **發出** *HTTP* **呼叫**，討論伺服器通訊並深入 HTTP 呼叫以及攔截器等進階概念。
- 第 10 章**單元測試服務**，回頭再度討論單元測試，但這一次專注於單元測試服務。這包括測試簡單的服務與非同步流程等進階主題。
- 第 11 章 *Angular* **的導向**，深入討論 Angular 應用程式的導向與導向模組。

- 第 12 章**製作 *Angular* 應用程式**，整合所有概念並討論應用程式的開發以及相關的各種概念與技術。

所有程式碼都放在 GitHub，若你不想自己打字或要確保取得最新最正確的範例，可以上網去抓（*https://github.com/shyamseshadri/angular-up-and-running*）。

所有程式範例均使用 AngularJS 5.0.0 版。

線上資源

下面是很棒的 AngularJS 開發者線上資源：

- Angular API 的官方文件（*https://angular.io/api*）
- 官方的 Angular 快速入門指南（*https://angular.io/guide/quickstart*）
- Angular Heroes Tutorial 應用程式（*https://angular.io/tutorial*）

本書編排慣例

本書使用以下的編排規則：

斜體字（*Italic*）

代表新的術語、URL、電子郵件地址、檔案名稱及副檔名。中文以楷體表示。

定寬字（`Constant width`）

代表程式，也在文章中代表程式元素，例如變數或函式名稱、資料庫、資料類型、環境變數、陳述式，與關鍵字。

定寬粗體字（**`Constant width bold`**）

代表指令，或其他應由使用者逐字輸入的文字。

定寬斜體字（*`Constant width italic`*）

代表應換成使用者提供的值，或依上下文而決定的值。

第一章

Angular 介紹

我們現在覺得網路應用程式（桌上與行動）的執行能力應該與原生應用程式相同。如今網路應用程式的規模與複雜度與桌上原生應用程式相同，對開發者來說也是一樣複雜。

不僅如此，單頁應用程式（Single-Page Application，SPA）變成常見的前端體驗，因為它有很好的速度與反應。程式載入用戶的瀏覽器後，互動只需載入更多的資料而無需從伺服器重新載入整個網頁。

AngularJS 本來的設計目的是將單頁應用程式結構化與一致化，並能快速開發可擴大與可維護的應用程式。自它發佈後，網頁與瀏覽器進步的很快，AngularJS 要解決的問題已不存在。

然後新版本的 Angular 針對新世代網頁而重寫。它利用模組與元件等新技術改善原有的 AngularJS 功能，例如相依性注入與模板。

接下來的內容中，AngularJS 指原始的 AngularJS 框架 1.0 版，Angular 指 2.0 版。主要是因為 Angular 2.0 不只使用 JavaScript 還支援 TypeScript。

為何使用 Angular

Angular 框架利用新技術且提供團隊開發者共通的結構。它讓我們開發可維護的大型應用程式。接下來的章節會深入討論這些功能：

用戶元件

Angular 可讓你建構自訂的宣告式元件，將功能與繪製邏輯包裝在可重複使用的小程式段中。它還可與網頁元件合作。

資料連結

Angular 能讓你將資料從 JavaScript 程式移動到視圖並反應事件而無需自行撰寫連結程式。

相依性注入

Angular 可讓你撰寫模組化服務並注入到所需的任何地方，如此能大幅改善可測試性與可重複使用性。

測試

測試是第一級公民，且 Angular 在設計時就考慮到測試。你可以（且應該要）測試應用程式的每個部分。

功能完整

Angular 是功能完整的框架，提供伺服器通訊、導向等立即可用的解決方案。

新版本 Angular 會採用語意化版本編號。此外，核心團隊規劃每六個月釋出一個主要改版。然後原來的 Angular 2 改稱為 Angular，因為我們不會稱它們為 Angular 2、Angular 4、Angular 5 等。

也就是說，不像 AngularJS 對 Angular，Angular 的版本升級（例如 2 到 4）是逐步上去的，且很少會是小改版。因此你無需擔心每隔幾個月就要大幅改寫程式。

本書討論範圍

雖然 Angular 是個大框架，但其社群更大。很多好功能都來自這個社群。這讓作者很難選擇要寫什麼給 Angular 開發者看。

因此，雖然 Angular 能以多種方式擴充，例如使用 Angular 撰寫原生行動應用程式（見 NativeScript，*https://www.nativescript.org/*）、在伺服器繪製 Angular 應用程式（見 Angular Universal，*https://universal.angular.io/*）、在 Angular 中使用 Redux 作為第一級選項（多種選項；見 ngrx，*https://github.com/ngrx*）等，本書第一版只專注於 Angular 核心與其功能，且更聚焦於常見狀況而非 Angular 的每一個功能，否則這樣的一本書會需要數千頁。

本書的目標是專注於對所有 Angular 開發者必要且有用的部分，而不是特定目的的功能。

開發環境

Angular 需要你在電腦上完成基本開發設定，接下來的討論必須先安裝好的軟體。

Node.js

雖然你無需寫 Node.js 程式，但 Angular 以 Node.js 作為開發環境。因此，要使用 Angular 就必須在環境中安裝 Node.js。安裝的方式有很多種，更多資訊見 Node.js 下載頁（*https://nodejs.org/en/download/*）。

在 macOS 上以 Homebrew 安裝 Node.js 會有一些問題，若遇到問題可嘗試直接安裝。

你必須安裝 6.9.0 或以上版本的 Node.js 與 3.0.0 或以上版本的 npm。安裝後可以使用下列命令確認版本：

```
node --version
npm --v
```

TypeScript

TypeScript 在我們寫的程式中加入一組型別使程式更容易理解、看懂、與追蹤。它確保最新提出的 ECMAScript 功能可供我們運用。你的 TypeScript 程式碼最終會編譯成可在任何環境執行的 JavaScript。

開發 Angular 應用程式不一定要使用 TypeScript，但強烈建議這麼做，因為這樣比較容易寫且程式更好維護。本書使用 TypeScript 開發 Angular 應用程式。

TypeScript 以 NPM 套件安裝，因此可使用下列命令安裝：

```
npm install -g typescript
```

要確保安裝 2.4.0 或以上版本。

雖然我們會討論使用到的 TypeScript 功能與概念，但閱讀 TypeScript 文件（*https://www.typescriptlang.org/docs/home.html*）來學習它是個好主意。

Angular 的 CLI

不像 AngularJS 很容易引入相依檔案來執行，Angular 的設定更複雜。因此 Angular 團隊寫了一個命令列介面（command-line interface，CLI）工具幫助開發 Angular 應用程式。

在它的幫助下可讓開發程序更簡單，我建議使用它直到你可以自行處理為止。本書會同時討論 CLI 命令與它在底下實際執行的動作以讓你認識所有必要的改變。

執行下列命令以安裝最新版本（目前是 1.7.3）：

```
npm install -g @angular/cli
```

上面的 Angular 套件版本命名慣例使用 NPM 中稱為範圍套件的新語法，它能讓多個套件安排在單一目錄下。更多資訊見 *https://docs.npmjs.com/misc/scope*。

安裝後可執行下列命令確認：

```
ng --version
```

取得程式碼

本書範例與練習題都放在 Git 程式庫。雖然不一定要下載，但下載後可作為參考或執行範例。你可以用下列命令複製此 Git 程式庫：

```
git clone https://github.com/shyamseshadri/angular-up-and-running.git
```

它會在你目前的工作目錄下建構 *angular-up-and-running* 目錄，此目錄下有依章節安排的子目錄。

總結

我們已經設定好開發環境並準備好開發 Angular 應用程式。我們安裝了 Node.js、TypeScript、Angular 的 CLI 並知道其作用。

下一章會開始建構第一個 Angular 應用程式，並認識一些 Angular 的基本詞彙與概念。

第二章

Hello Angular

前一章簡短的討論過 Angular 與其功能以及設定 Angular 開發環境。這一章會從頭建構一個非常簡單的應用程式以討論 Angular 應用程式的不同部分。我們以此應用程式討論一些基本的術語與模組、元件、資料和事件連結、元件資料傳遞等概念。

我們從一個非常簡單的股市應用程式開始，它可以讓我們看到股票名稱、代號、與價格。過程中會討論如何將股票資訊包裝在獨立與可重複使用的元件中，以及如何使用 Angular 事件與資料連結。

啟動你的第一個 Angular 專案

如前述，我們會大量依靠 Angular 的 CLI 來開發應用程式。我假設你已經根據前一章的指令在開發環境中安裝好 Node.js、TypeScript、Angular 的 CLI。

執行下列命令建構新應用程式：

```
ng new stock-market
```

執行此命令時，它會在 *stock-market* 目錄下自動產生應用程式的骨架與一些檔案，並安裝 Angular 應用程式必要的相依檔案。這可能需要花一點時間，但最終你應該會看到下列訊息：

```
Project 'stock-market' successfully created.
```

可喜可賀，你剛剛建構了你的第一個 Angular 應用程式！

我們使用 Angular 的 CLI 建構了第一個應用程式，`ng new` 命令的參數能讓你設定選項，包括：

- 是否使用 CSS、SCSS、或其他 CSS 框架（例如 `ng new --style=scss`）
- 是否產生導向模組（例如 `ng new --routing`）；第 11 章會深入討論
- 是否需要行內樣式 / 模板
- 元件是否需要前綴（舉例來說，`ng new --prefix=acme` 對所有元件前綴 `acme`）

還有其他參數，可在更熟悉 Angular 後執行 `ng help`，以探索是否需要設定其他選項。

認識 Angular 的 CLI

我們建構了第一個應用程式，但 Angular 的 CLI 不只是產生骨架而已。事實上，它可在開發過程中執行各種工作：

- 輔助啟動
- 服務應用程式
- 執行測試（單元與完整）
- 建置與發佈
- 產生新元件、服務、導向等

每個工作有相對應的 CLI 命令，接下來遇到時會解釋。命令可設定參數與選項使 CLI 能夠負責各種任務。

執行應用程式

我們已經產生了應用程式，下一步是執行以從瀏覽器觀察。技術上有兩種執行方式：

- 以開發模式執行，由 Angular 的 CLI 編譯並更新 UI
- 以上線模式執行，經最佳化編譯成靜態檔案

現在我們以開發模式從專案的根目錄 *stock-market* 執行：

```
ng serve
```

經過一陣子處理與編譯後你應該會看到下列訊息：

```
** NG Live Development Server is listening on localhost:4200,
   open your browser on http://localhost:4200/ **
Date: 2018-03-26T10:09:18.869Z
Hash: 0b730a52f97909e2d43a
Time: 11086ms
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 17.9 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 549 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 41.5 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 7.42 MB [initial] [rendered]

webpack: Compiled successfully.
```

上面的輸出是 Angular 的 CLI 產生以供執行應用程式的檔案，包括轉譯出的 *main.bundle.js*、包含第三方函式庫與框架（包含 Angular）的 *vendor.bundle.js*、編譯後的 CSS 樣式表 *styles.bundle.js*、在舊版瀏覽器中支援新功能（例如 ECMAScript 功能）的 *polyfills.bundle.js*、啟動應用程式所需的 *inline.bundle.js*。

ng serve 在本機 4200 埠啟動開發伺服器。從瀏覽器打開 *http://localhost:4200* 會看到如圖 2-1 所示的 Angular 應用程式。

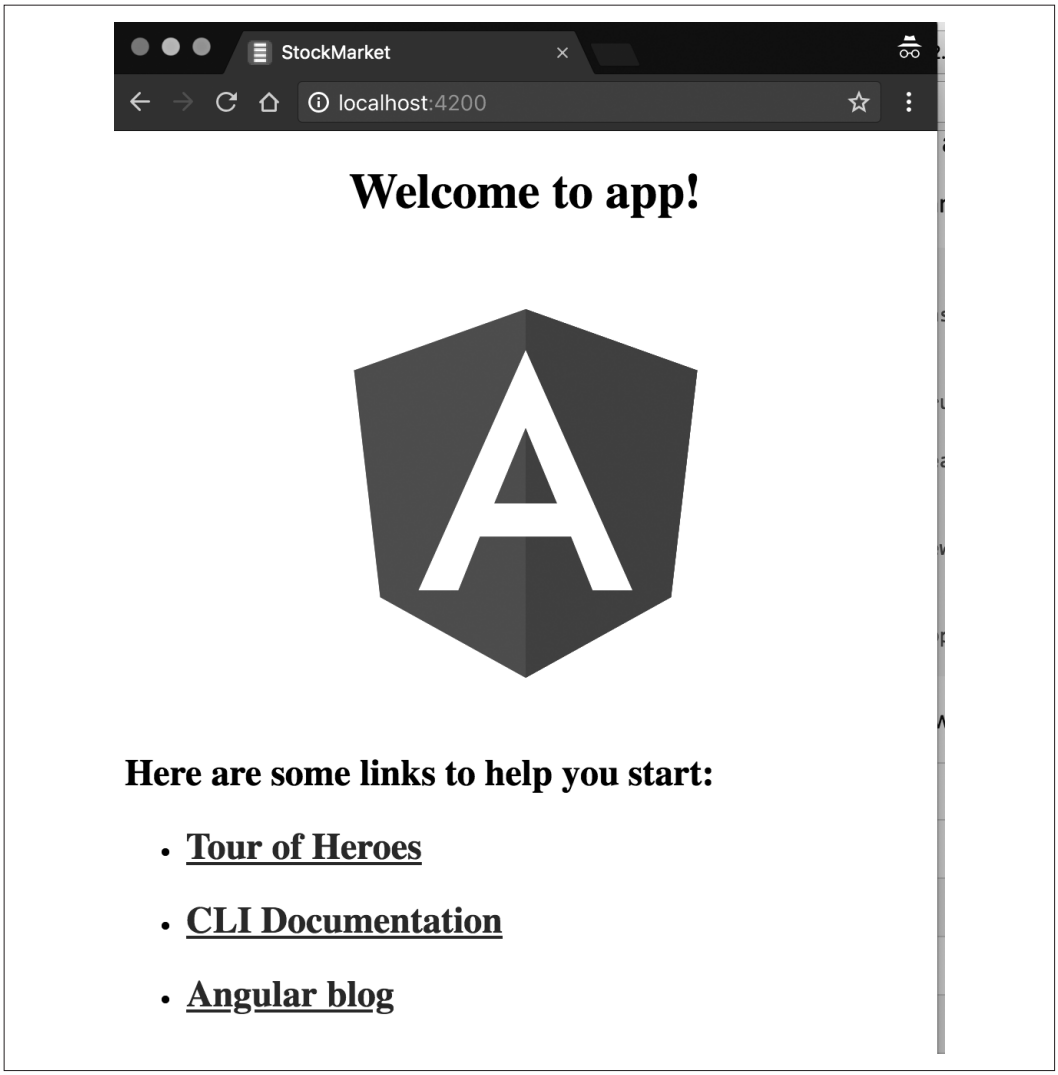

圖 2-1　瀏覽器中的 Hello Angular 應用程式

你可以讓 ng serve 命令在終端機中持續執行並繼續修改。若有應用程式在瀏覽器中開啟，它會在儲存時自動的更新。這樣會讓開發過程更方便。

接下來進一步討論產生出的 Angular 應用程式如何運作與各個組成部分。

Angular 應用程式的基礎

Angular 應用程式的核心是單頁應用程式（Single-Page Application，SPA），因此載入是由一個對伺服器的請求觸發。從瀏覽器開啟一個 URL 時會發出第一個對伺服器（此例中以 `ng serve` 執行）的請求。第一個請求會回傳一個 HTML 網頁，然後它載入必要的 JavaScript 以載入 Angular 與我們的程式碼和模板。

注意雖然我們以 TypeScript 開發 Angular 應用程式，但此應用程式被轉譯成 JavaScript。`ng serve` 命令負責將 TypeScript 轉譯成瀏覽器載入的 JavaScript。

Angular 的 CLI 產生的結構如下：

```
stock-market
+----e2e
+----src
     +----app
          +----app.component.css
          +----app.component.html
          +----app.component.spec.ts
          +----app.component.ts              ❶
          +----app.module.ts                 ❷
     +----assets
     +----environments
     +----index.html                         ❸
     +----main.ts                            ❹
 +----.angular-cli.json                      ❺
```

❶ 根元件

❷ 主要模組

❸ 根 HTML

❹ 進入點

❺ Angular CLI 組態

stock-market 目錄下還有其他檔案，但上面列出的是這一章要討論的檔案。此外，第 5 章、第 10 章、第 12 章還有單元測試、端至端測試、資源檔案、各種環境（開發、產品）的組態、與其他通用組態。

根 HTML—index.html

檢視 *src* 目錄下的 *index.html* 檔案，你會注意到它很簡單與乾淨，沒有參考任何腳本或相依檔案：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>StockMarket</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>                    ❶
</body>
</html>
```

❶ Angular 應用程式的根元件

上面的程式中唯一值得注意的是 `<app-root>` 元件，它是載入應用程式碼的標記。

載入核心 Angular 腳本與應用程式碼的部分呢？它由 `ng serve` 命令在執行期動態的插入，結合所有廠商函式庫、應用程式碼、樣式表、放在個別檔案的行內模板，並於瀏覽器繪製該頁時插入到 *index.html* 中。

進入點—main.ts

第二個重要部分是 *main.ts* 檔案。*index.html* 檔案決定載入什麼檔案。另一方面，*main.ts* 識別應用程式啟動時要載入什麼 Angular 模組（接下來會討論）。它也可以改變第 12 章會討論的應用程式層級組態（例如使用 `enableProdMode()` 旗標關閉框架層級的斷言與檢驗）：

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
```

```
}

platformBrowserDynamic().bootstrapModule(AppModule)          ❶
  .catch(err => console.log(err));
```

❶ 啟動 AppModule

main.ts 檔案中的大部分程式是通用的，通常無需修改此進入點檔案。它的主要目的是對 Angular 框架指出應用程式的核心模組並從該點觸發其餘應用程式原始碼。

主要模組—app.module.ts

這是應用程式原始碼的啟動位置。應用程式模組檔案可視為應用程式的核心組態，載入相關與必要的相依檔案、宣告使用的元件、標記應用程式的主要進入點元件：

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component';


@NgModule({                              ❶
  declarations: [
    AppComponent                         ❷
  ],
  imports: [                             ❸
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]              ❹
})
export class AppModule { }
```

❶ NgModule 這個 TypeScript 的標記指出此類別定義為一個 Angular 模組

❷ 宣告應用程式中使用的元件與指示

❸ 匯入其他功能模組

❹ 啟動應用程式的進入點元件

這是我們第一次處理 TypeScript 專屬的功能，它是修飾子（可以視為注釋）。修飾子能讓我們以注釋與屬性以及元功能修飾類別。

Angular 大量運用此 TypeScript 功能，例如使用模組與組件的修飾子。

更多 TypeScript 修飾子的資訊見官方文件（*http://bit.ly/2IDQd1U*）。

接下來的章節會深入這些部分，但其核心是：

`declarations`

`declarations` 區塊定義此模組中可用於此 HTML 範圍內的所有元件。所有元件必須在使用前宣告。

`imports`

你不會建構應用程式使用的每一個功能，`imports` 陣列可匯入其他 Angular 應用程式與函式庫模組，並利用這些元件、服務、與其他已經寫在這些模組中的功能。

`bootstrap`

`bootstrap` 陣列定義應用程式的進入點元件。若未將主元件加入，則應用程式不會啟動，因為 Angular 不知道要從 *index.html* 中找什麼元素。

加入新元件、服務、函式庫、模組時通常需要修改這個檔案（若沒有使用 CLI）。

根元件—AppComponent

它是真正提供應用程式功能的 Angular 程式碼，此例中的 `AppComponent` 是主要（且唯一）的元件，其程式碼如下：

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',                    ❶
  templateUrl: './app.component.html',     ❷
  styleUrls: ['./app.component.css']       ❸
})
export class AppComponent {
  title = 'app';                           ❹
}
```

❶ 此 DOM 選擇器會被轉譯成此元件的一個實例

❷ 此元件的 HTML 模板——此例中為指向它的 URL

❸ 元件專屬的樣式表，同樣指向另一個檔案

❹ 元件類別與成員和函式

Angular 中的**元件**只是 TypeScript 類別，以一些屬性與元資料修飾。此類別封裝元件的所有資料與功能，而修飾子指定如何轉譯成 HTML。

應用程式的選擇器是 Angular 找出 HTML 網頁中特定元件的 CSS 選擇器。雖然我們使用元素選擇器（上面範例中的 `app-root`，它會轉譯成尋找 HTML 中的 `<app-root>` 元素），但它可以是從 CSS 類別到屬性等任何 CSS 選擇器。

`templateUrl` 是繪製此元件的 HTML 的路徑。我們也可以使用行內模板而非如範例一樣指定 `templateUrl`。此例中指向的模板是 *app.component.html*。

`styleUrls` 是對應模板的樣式表，它封裝此元件的所有樣式表。Angular 確保樣式表被封裝，因此無需擔心一個元件的 CSS 類別會影響其他元件。與 `templateUrl` 不同，`styleUrls` 是個陣列。

元件類別本身最終封裝元件的所有功能，可將此元件類別的責任視為兩個部分：

- 載入並保存繪製此元件的所有資料
- 處理元件中任何元素可能會發出的事件

類別中的資料會驅動元件的顯示。讓我們看一下此元件的模板：

```
<h1>
  {{title}}                    ❶
</h1>
```

❶ 資料連結的標題

元件的 HTML 非常簡單。它只有一個元素，連結元件類別中的一個欄位。雙大括弧（`{{ }}`）語法指示 Angular 替換相對應類別變數。

此例中，應用程式載入並繪製後，`{{title}}` 會替換成 `app works!`。第 19 頁“認識資料連結”一節會更深入討論資料連結。