



« Angular 4 開發實戰 »

新手入門篇



多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

部落格：<http://blog.miniasp.com/>

Angular: Introduction

ANGULAR 簡介



關於 AngularJS 與 Angular

- 全球領先的開源 JavaScript 應用程式框架
 - 由 Google 主導並擁有廣大社群共同參與框架發展
- Angular 1.x ➔ 正名：AngularJS
 - 擁有廣大開發社群 (最大的)
 - 透過嶄新的抽象化架構大幅簡化應用程式開發
- Angular 2.x / 4.x ➔ 正名：Angular
 - 重新打造的下一代 Angular 開發框架
 - 擁有更高的執行效率、更好的延展性架構
 - 透過全新的元件化技術建構現代化的開發框架

從框架轉向平台

| | | | |
|-----------|----------------------|-------------------|-----------|
| i18n | CLI | Language Services | Augury |
| Animation | Material | Mobile | Universal |
| Router | Compile | Change | Render |
| ngUpgrade | Dependency Injection | Decorators | Zones |

前端工程的夢幻逸品：Angular 2 開發框架介紹

Angular 主要特色 (1)

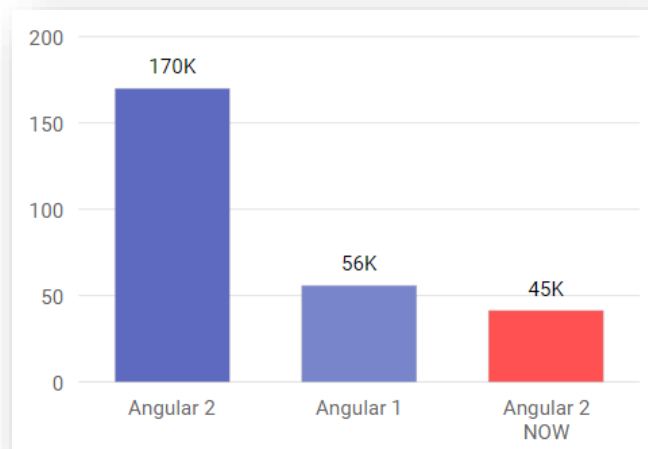
- 跨平台
 - [Progressive Web Apps](#) ([Angular Mobile Toolkit](#))
 - 結合網頁和應用程式優點於一身的絕佳體驗
 - Desktop Apps
 - 可搭配 [Electron](#) 框架開發出跨越 Windows, Mac, Linux 的桌面應用程式
 - Native Apps
 - 可搭配 [Ionic Native](#), [NativeScript](#), [React Native](#) 開發跨行動平台的原生應用程式
- 速度與效能
 - Code generation ([AOT](#))
 - 將元件範本預先編譯成 JS 程式碼
 - [Universal](#)
 - 將開啟頁面的首頁預先產生完整 HTML 與 CSS 原始碼，加快首頁載入速度
 - 可支援 Node.js, .NET, PHP 或任何其他伺服器端網頁架構
 - Code Splitting
 - 透過全新的元件路由機制，讓使用者只須載入需要的原始碼

Angular 主要特色 (2)

- 生產力提升
 - [Templates](#)
 - 使用簡易有強大的範本語法提高開發效率
 - [Angular CLI](#)
 - 透過命令列工具快速建模、新增元件、執行測試與發行部署
 - IDE
 - 在現有編輯器或開發工具中使用程式碼自動完成、即時錯誤提示與程式碼建議
- 完整的開發體驗
 - [Testing](#)
 - 結合 Karma 執行單元測試，結合 Protractor 執行各種 E2E 測試情境
 - [Animation](#)
 - 透過 Angular 直觀的 API 完成複雜的頁面動畫處理
 - Accessibility
 - 透過 ARIA-enabled 元件、開發者指引與內建的 [a11y](#) 測試基礎架構，建構具有可及性的應用程式

Angular 主要亮點

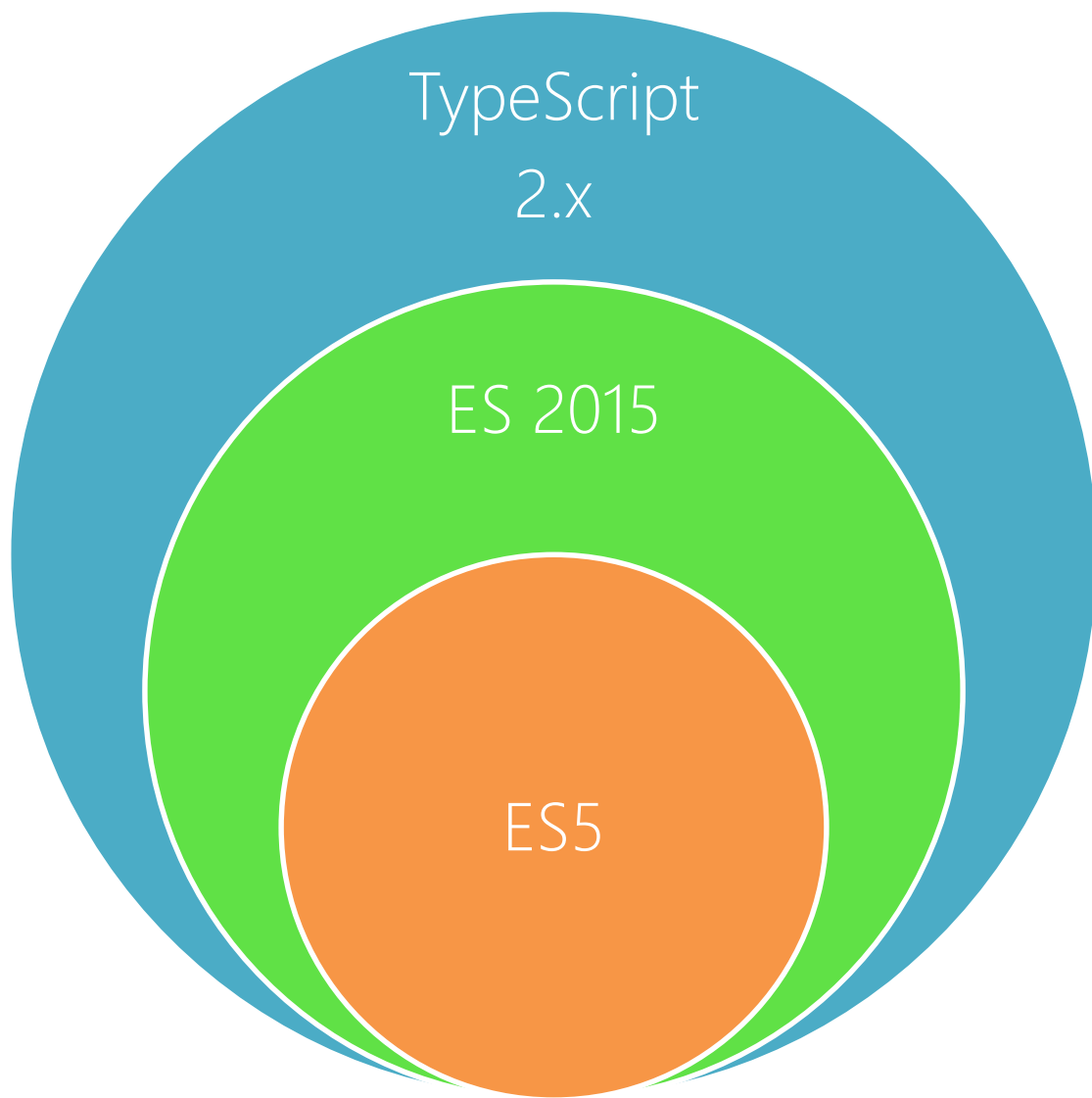
- 效能改進 (Performance)
 - 偵測變更：比 ng1 快 10 倍
 - 更小的 Library Size 與延遲載入機制
 - 範本編譯：支援 Template 預先編譯機制
 - 渲染速度：比 ng1 快 5 倍 (Render & Re-render)
 - 支援伺服器渲染機制 ([Node.js](#) & [ASP.NET](#))
- 高生產力 (Productivity)
 - 開發應用程式能夠用更簡潔的語法讓團隊更加容易上手跟維護
 - 更強大的開發工具 [Augury](#)
 - 移除超過 40+ 個 directives
- 多樣平台 (Versatility)
 - 支援 Browser, Node.js, NativeScript, and more ...



Angular 真正優勢

- **更熟悉的開發架構**
 - 採用 TypeScript 開發語言，使用以類別為基礎的物件導向架構開發 Web 應用程式，幫助 C#, Java, PHP, ...等開發人員快速上手全新架構。
 - 透過開發人員手邊現有的開發工具/編輯器，就可以開發 Angular 2 應用程式，並同時享有 IntelliSense、程式碼重構等工具支援。
- **更低的學習門檻**
 - 相較於 Angular 1 減少了許多抽象的架構與概念，對於剛入門的 Angular 開發者將更加容易上手
 - 例如 Angular 1 的 directives 就有非常多抽象概念
- **更好的執行效率與行動化體驗**
 - 不同行動裝置之間的各種特性皆考量在內，例如觸控、螢幕大小、硬體限制、...
 - 內建伺服器渲染技術 (server rendering) 與 Web Worker 技術改善頁面載入效率
 - 不僅僅做到預先產生 HTML 頁面，更能透過 NativeScript 或 Ionic 建立起網站框架與 Mobile App 之間的橋樑，開發速效率更好的行動瀏覽體驗。
- **更清晰的專案結構與更好的可維護性**
 - 使用 ES2015 模組管理機制，搭配 webpack 或 SystemJS 等工具即可立刻上手
 - 全新的元件模組化架構，更能夠幫助大家更快的了解程式碼結構，降低維護成本
 - 好的模組化架構更能降低開發工具的開發難度，也更適合開發大型的網站應用

Angular 的開發語言 (圖示)



Angular 的開發語言

- ES5
 - 傳統 JavaScript 程式語言 (IE9+)
- [ES 2015](#) (ES6)
 - 此版本為 ES5 的「超集合」
 - 具有新穎的 JavaScript 語言特性 (let, const, for-of, ...)
 - 可透過 [Babel](#) 轉譯器將瀏覽器不支援的語法轉為 ES5 版本
- [TypeScript](#)
 - 此版本為 ES 2015 的「超集合」
 - 具有強型別特性、內建 ES5 轉譯器 (Transpiler)、更好的工具支援
- [Dart](#)
 - 非 JavaScript 家族的程式語言
 - 具有強型別特性

Angular 的開發工具

- [Visual Studio Code](#) (免費的開源軟體) (推薦)
- [WebStorm](#) (商用軟體) (地表最強 Angular 開發工具)
- [Sublime Text](#)
- [Atom](#)
- [Plunker](#)
- Visual Studio 2017

Angular 應用程式的組成

模組

- AppModule

元件

- App Component

元件

- Child Component

元件

- Services Component

元件

- Pipe Component

Angular 頁面的組成

應用程式元件 + 樣板 + 樣式
(AppComponent)

頁首元件 + 樣板 + 樣式
(HeaderComponent)

子選單
元件 + 樣板 + 樣式
(AppComponent)

主要內容
元件 + 樣板 + 樣式
(ArticleComponent)

Angular 元件的組成

範本 (Template)

- HTML 版面配置
- HTML 部分片段
- 資料繫結 (Bindings)
- 畫面命令 (Directives)

類別 (Class)

- 建構式 (Constructor)
- 屬性 (Properties)
- 方法 (Methods)

中繼資料 (Metadata)

- 裝飾器 (Decorator)
 - 針對類別
 - 針對屬性
 - 針對方法
 - 針對參數

認識 Angular 元件的程式碼結構

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

匯入模組

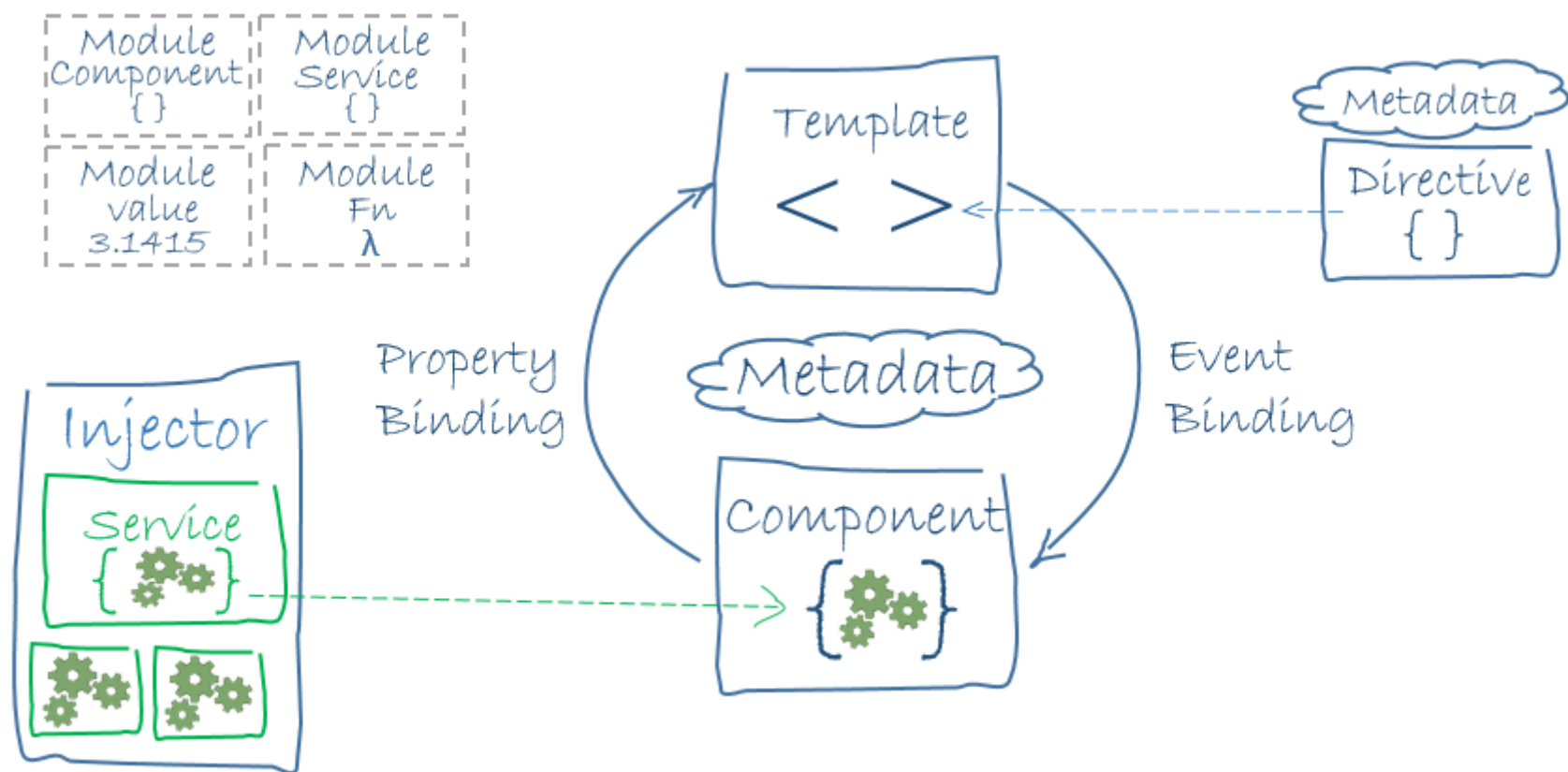
```
@Component({  
  selector: 'app-root',  
  templateUrl: 'app.component.html',  
  styleUrls: ['app.component.css']  
})
```

裝飾器

```
export class AppComponent {  
  title = 'app works!';  
}
```

類別

剖析 Angular 架構 (圖示)



剖析 Angular 架構

- [Modules](#) 應用程式被切分成許多「模組」
- [Components](#) 每個模組下有許多「元件」
- [Templates](#) 每個元件都可能有自己的「樣板」
- [Metadata](#) 每個元件都可以標示「中繼資料」
- [Data Binding](#) 樣板與元件屬性、方法可以進行綁定
- [Directives](#) 將 DOM 轉換為多功能的「宣告命令」
- [Services](#) 由「服務」集中管理資料與運算邏輯
- [Dependency Injection](#) 由「相依注入」機制管理物件生命週期



Setup your Angular development environment

建立 ANGULAR 開發環境

準備 Angular 開發環境

- 架設 Angular 4 開發環境說明文件
 - [Google Chrome](#)、[VSCode](#)、[Git](#)、[Node.js](#) 與 [Angular CLI](#) 工具
- 如何修改 Visual Studio Code 內建的 TypeScript 版本
 - Angular 4 預設採用 TypeScript 2.0+ 為主要開發語言！
- 關於 TypeScript 2.0 之後的模組定義檔 (*.d.ts)
 - 從 TypeScript 2.0 開始已改用 npm 來管理模組定義檔 ([@types](#))
 - 未來不再使用 [typings](#) 工具進行模組定義檔管理

安裝 Angular CLI 命令列工具

- 用來快速開發 Angular 應用程式的命令列工具 (Command Line Tools)
- 必備條件
 - 安裝 Node.js 6.9 以上版本
- 安裝方式
 - `npm install -g @angular/cli`
- 升級方法 (Global package)
 - `npm uninstall -g @angular/cli`
 - `npm cache clean`
 - `npm install -g @angular/cli`
- 升級方法 (local project package)
 - 完整刪除專案內的 `node_modules` 與 `dist` 資料夾
 - `npm install --save-dev @angular/cli@latest`
 - `npm install`
- Angular CLI 版本變更紀錄：[CHANGELOG.md](#)

使用 Angular CLI 建立專案架構

- 使用說明
 - `ng --help`
- 建立新專案並啟動開發伺服器
 - `ng new PROJECT_NAME`
 - `cd PROJECT_NAME`
 - `npm start` (執行 `ng serve` 也可以)
 - <http://localhost:4200>
- 啟動開發伺服器並自動開啟瀏覽器
 - `ng serve --open`
- 指定不同埠號啟動開發伺服器
 - `ng serve --port 4201 --live-reload-port 49153`
- 執行在 Production 模式
 - `ng serve --prod`

使用 Angular CLI 快速產生程式碼

- 透過 **藍圖** (blueprint) 產生程式碼
 - `ng generate 藍圖 元件名稱`
 - 透過 **藍圖** (blueprint) 產生程式碼 (簡寫)
 - `ng g 藍圖 元件名稱`
 - 使用範例
 - 產生 HeaderComponent 元件
 - `ng g component header`
 - `ng g c header`
 - `ng g c header --spec=false`
 - `ng g c charts/header`
 - 產生 DataService 服務元件
 - `ng g s data`
 - 產生 Charts 模組
 - `ng g m charts`
 - 查詢其他藍圖用法
 - `ng g --help`
- # 建立元件
簡寫版本 (`c = component`)
不要建立單元測試檔 (`*.spec.ts`)
在特定目錄(模組)下建立元件
- # 建立服務元件
- # 建立模組
- # 顯示所有藍圖與用法說明

常見 Angular CLI 產生器藍圖與範例

| 藍圖名稱 | 使用方式 |
|-----------|--|
| Component | <code>ng g component my-new-component</code> |
| Service | <code>ng g service my-new-service</code> |
| Module | <code>ng g module my-module</code> |
| Directive | <code>ng g directive my-new-directive</code> |
| Pipe | <code>ng g pipe my-new-pipe</code> |
| Class | <code>ng g class my-new-class</code> |
| Interface | <code>ng g interface my-new-interface</code> |
| Enum | <code>ng g enum my-new-enum</code> |

使用 Angular CLI 建置專案

- 建置專案 (預設為 dev 環境，可切換為 prod 模式)
 - **ng build**
 - **ng build --prod**
 - 預設會將現有應用程式建置後輸出到 **dist/** 目錄下
- 建置專案的注意事項
 - 建置專案的過程中，如果是 dev 模式 (**ng build**)
 - src/environments/environment.ts
 - 建置專案的過程中，如果是 prod 模式 (**ng build --prod**)
 - src/environments/environment.prod.ts
 - 你也可以自行定義不同的建置模式 (**ng build --env=NAME**)
 - .angular-cli.json
 - src/environments/environment.NAME.ts
 - 查詢其他的建置參數
 - **ng build --help**

使用 Angular CLI 執行測試

- 執行單元測試
 - **ng test**
 - 單元測試會在 ng build 執行完成後，透過 [Karma](#) 不斷執行
 - [Karma](#) 會自動偵測檔案變更，只要有變更就會自動在背景執行單元測試
 - 你可以執行 `ng test --watch=false` 只執行一次單元測試
 - 也可以執行 `ng test --build=false` 避免執行 `ng build` 建置動作
- 執行 E2E 測試 (End-to-end tests)
 - **ng e2e**
 - E2E 測試是透過 [Protractor](#) 來執行

使用 Angular CLI 檢查程式品質

- 執行 **ng lint** 即可進行自動化品質驗證
 - 該命令會在背景自動執行 [TSLint](#) 程式碼檢查 ([TSLint core rules](#))
 - 其實只是去執行 package.json 裡 "scripts" 內的 lint 命令而已
 - 因此你執行 npm run lint 也是完全相同的意思！
- 自動檢查專案是否符合官方的 [Angular Style Guide](#) 標準
 - 內建 [codelyzer](#) 分析工具 (<https://github.com/mgechev/codelyzer>)
 - 預設定義檔為 **tslint.json**
- 記得 Visual Studio Code 要安裝 [TSLint](#) 擴充套件



Build your first Angular Application

建立 ANGULAR 應用程式

從現有的 Angular 專案範本做起

- 使用 Angular CLI
 - `ng new demo1 --routing --skip-tests`
- 使用 Webpack
 - [johnpapa/angular-tour-of-heroes](https://github.com/johnpapa/angular-tour-of-heroes)
 - [AngularClass/angular-starter](https://github.com/AngularClass/angular-starter)
- 使用 SystemJS
 - [DanWahlin/Angular-JumpStart](https://github.com/DanWahlin/Angular-JumpStart)
 - [DanWahlin/Angular-BareBones](https://github.com/DanWahlin/Angular-BareBones)
- 使用 Gulp
 - [mgechev/angular2-seed](https://github.com/mgechev/angular2-seed)

了解 Angular CLI 建立的專案結構

- 首頁 HTML 與 Angular 主程式
 - src/index.html 預設網站首頁 (還是要有一份 HTML 網頁來載入 JS 執行)
 - src/style.css 預設全站共用的 CSS 樣式檔
 - src/main.ts 預設 Angular 程式進入點
- 公用檔案資料夾
 - src/assets/ 網站相關的靜態資源檔案 (CSS, Image, Fonts, ...)
- 應用程式原始碼
 - src/app/app.module.ts 應用程式的全域模組 (AppModule)
 - src/app/app.component.ts 根元件主程式 (AppComponent)
 - src/app/app.component.html 根元件範本檔 (HTML)
 - src/app/app.component.css 根元件樣式檔 (CSS)
 - src/app/app.component.spec.ts 根元件單元測試程式
- 共用的環境變數
 - src/environments/environment.ts 環境變數設定 (預設)
 - src/environments/environment.prod.ts 環境變數設定 (ng build --env=prod)

src/index.html

index.html x

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Demo1</title>
6   <base href="/">
7
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 </body>
14 </html>
15
```

咦？沒有載入任何 JavaScript 函式庫？

根元件的 directive 宣告

src/main.ts

TS main.ts x

```
1 import { enableProdMode } from '@angular/core';
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5 import { environment } from './environments/environment';
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 platformBrowserDynamic().bootstrapModule(AppModule);
12
```

啟用 Production 模式 (提升執行速度)

設定 AppModule 為啟動模組

src/app/app.module.ts

app.module.ts x

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6
7 @NgModule({
8   declarations: [ ← 宣告跟 View 有關的元件
9     AppComponent
10  ],
11   imports: [ ← 宣告要匯入此模組的外部模組
12     BrowserModule,
13     AppRoutingModule
14  ],
15   providers: [], ← 宣告要註冊的服務元件
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { } ← 宣告根元件
19
```


src/app/app.component.ts

app.component.ts ●

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'app';
10
11    constructor() {
12    }
13
14    changeTitle(title: string) {
15      this.title = title;
16    }
17  }
18
```

指令 (directive) 選擇器

元件網頁範本

元件 CSS 樣式

TypeScript 類別

類別中的屬性 (Property)

類別的建構式

類別中的方法 (Method)

認識 Angular 元件的命名規則

// 命名規則：PascalCase

```
export class AppComponent {
```

// 命名規則：camelCase

```
  pageTitle : string = "Hello World";
```

// 命名規則：動詞 + 名詞 with camelCase

```
  printTitle() {
```

```
    console.log(this.pageTitle);
```

```
  }
```

```
}
```

建立子元件 (Child Component)

- 建立子元件
 - 透過 `ng generate component header` 建立元件
 - 簡寫指令：`ng g c header --spec=false`
 - 會建立 `HeaderComponent` 元件類別
- 會自動在 `app.module.ts` 匯入 `declarations` 宣告
 - `import { HeaderComponent } from './header/header.component';`
- 在任意元件的範本中使用 Directives 語法載入元件
 - `<app-header></app-header>`

資料繫結的四種方法 (Binding syntax)

- 內嵌繫結 (interpolation)

`{{property}}`

- 屬性繫結 (Property Binding)

`[propertyName]="statement"`

`bind-propertyName="expression"`

`[attr.attributeName]="statement"`

`bind-attr.attributeName="expression"`

- 事件繫結 (Event Binding)

`(eventName)="someMethod($event)"`

`on-eventName="someMethod($event)"`

- 雙向繫結 (Two-way Binding)

`[(ngModel)]='property'`

`bindon-ngModel='property'`

範本參考變數 (Template reference variables)

- 在範本中任意 HTML 標籤套用 **#name** 語法
 - 會在範本內建立一個名為 **name** 的區域變數
 - 該 **name** 區域變數將只能用於目前元件範本中
 - 該 **name** 區域變數將會儲存該標籤的 DOM 物件
 - 你可以透過「事件繫結」將任意 DOM 物件中的任意屬性傳回元件類別中 (Component class)
- 以下這兩種是完全相等的語法 (使用 **#** 是語法糖)
 - **#name**
 - **ref-name**

三種 Angular 指令 (Directives)

- 元件型指令 (Component Directives)
 - 預設「元件」就是一個含有樣板的指令
- 屬性型指令 ([Attribute Directives](#))
 - 這種指令會修改元素的外觀或行為
 - [內建的屬性型指令](#)：[NgStyle](#) 或 [NgClass](#)
- 結構型指令 ([Structural Directives](#))
 - 這種指令會透過**新增**和**刪除** DOM 元素來**改變** DOM 結構
 - [內建的結構型指令](#)：[NgIf](#)、[NgFor](#) 或 [NgSwitch](#)
 - 請注意 `ngSwitch` 前面**不要**加上 * 星號
 - 請注意 `ngIf` 與 `ngFor` 與 `ngSwitchDefault` 與 `ngSwitchCase` 前面**要**加上 * 星號
 - 請善用 Visual Studio Code 的 Code Snippets 自動完成程式碼

關於 * 與 <template> 語法

- 當用到結構型指令時，以下三種寫法都是完全相等的
 - `<hero-detail`
 `*ngIf="currentHero"`
 `[hero]="currentHero"></hero-detail>`
 - `<hero-detail`
 `template="ngIf:currentHero"`
 `[hero]="currentHero"></hero-detail>`
 - `<template [ngIf]="currentHero">`
 `<hero-detail [hero]="currentHero"></hero-detail>`
 `</template>`
- 因此套用 * 星號其實是套用 <template> 標籤的語法糖
- 請注意上例中 ngIf 所傳入的 "currentHero" 其實是個字串，只要不是空字串都算 Truthy 值，因此不管語法怎麼寫都不可能發生例外！

Angular 元件之間的溝通方式

- 傳入屬性
 - `@Input() myProperty;`
 - 在外層元件請記得用「屬性繫結」傳入資料
- 傳出事件
 - `@Output() myEvent = new EventEmitter<any>();`
 - `this.myEvent.emit(data);`
 - 在外層元件請記得用「事件繫結」來接收傳出的資料
 - 在 Template 中使用 **`$event`** 代表子元件傳出的資料

指令元件的主要生命週期 Hooks

| Hook Method | 說明 |
|-------------|---|
| ngOnInit | 當 Angular 已經初始化過所有 @Input() 屬性後執行 可實作 OnInit 介面 |
| ngOnChanges | 當元件的任意 @Input() 屬性被設定後執行 此方法會得到變更物件的目前值與先前的值 可實作 OnChanges 介面 |
| ngDoCheck | 當 Angular 每次執行 <u>變更偵測</u> 時執行 (會影響效能) |
| ngOnDestroy | 當 Angular 要摧毀元件時執行 建議在此處取消訂閱觀察者物件或刪除先前註冊過的事件處理器，以避免記憶體洩漏問題發生！ 可實作 OnDestroy 介面 |

使用 [Pipes](#) 管線元件

- Angular [內建的 Pipes 元件](#)
 - [UpperCasePipe](#) (uppercase) / [LowerCasePipe](#) (lowercase)
 - [DecimalPipe](#) (number) / [PercentPipe](#) (percent)
 - `{{ product.price | currency:'USD':true:'1.2-2' }}`
 - [CurrencyPipe](#) (currency) / [DatePipe](#) (date)
 - 貨幣代碼必須使用 [ISO 4217](#) 定義的 `currency code` 標準格式
 - [JsonPipe](#) (json) / [SlicePipe](#) (slice) / [AsyncPipe](#) (async)
- Angular 2+ 並沒有 `FilterPipe` 與 `OrderByPipe` 的存在！
 - 在 AngularJS 1.x 的年代，這兩個經常被濫用且效能低落
 - 因為由於 JS 沒有「傳值」的特性，導致經常有 Bug 出現

Dependency Injection

ANGULAR 相依注入

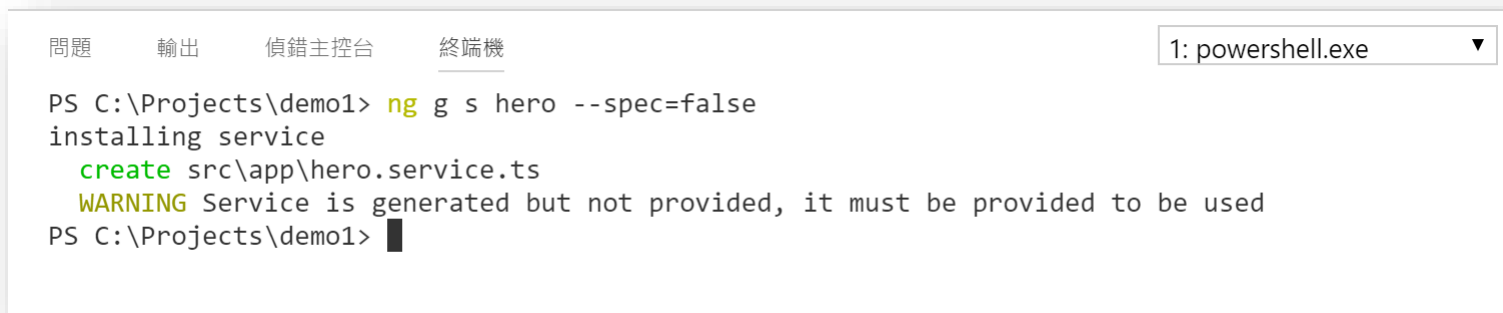


建立服務元件

- 主要用途
 - 可以在不同的元件之間共用相同的「資料」與「邏輯」

※ 注意：類別中只會有「屬性」與「方法」！

- 產生服務元件
 - `ng g s hero --spec=false`



```
問題 輸出 偵錯主控台 終端機 1: powershell.exe ▼
PS C:\Projects\demo1> ng g s hero --spec=false
installing service
  create src\app\hero.service.ts
  WARNING Service is generated but not provided, it must be provided to be used
PS C:\Projects\demo1> █
```

註冊為提供者 (全站共用的服務元件)

- 編輯 app/app.module.ts 檔案



The screenshot shows a code editor with the file 'app.module.ts' open. The code is as follows:

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { HeroService } from './hero.service';
7
8 @NgModule({
9   declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule
15   ],
16   providers: [
17     HeroService
18   ],
19   bootstrap: [AppComponent]
20 })
21 export class AppModule { }
22
```

Two red arrows point to specific parts of the code with Chinese annotations:

- An arrow points from the text **記得要 import 元件進來** (Remember to import the component) to the `import { HeroService } from './hero.service';` line (line 6).
- An arrow points from the text **服務元件必須註冊為「提供者」** (Service component must be registered as a 'provider') to the `HeroService` entry in the `providers` array (line 17).

注入服務元件到任意元件裡

- 在元件中注入一個**全站共用**的服務元件實體

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

```
import { HeroService } from '../hero.service';
```

```
@Component({  
  selector: 'app-header',  
  templateUrl: 'header.component.html'  
})
```

```
export class HeaderComponent {  
  constructor(herosvc: HeroService) {  
  }  
}
```

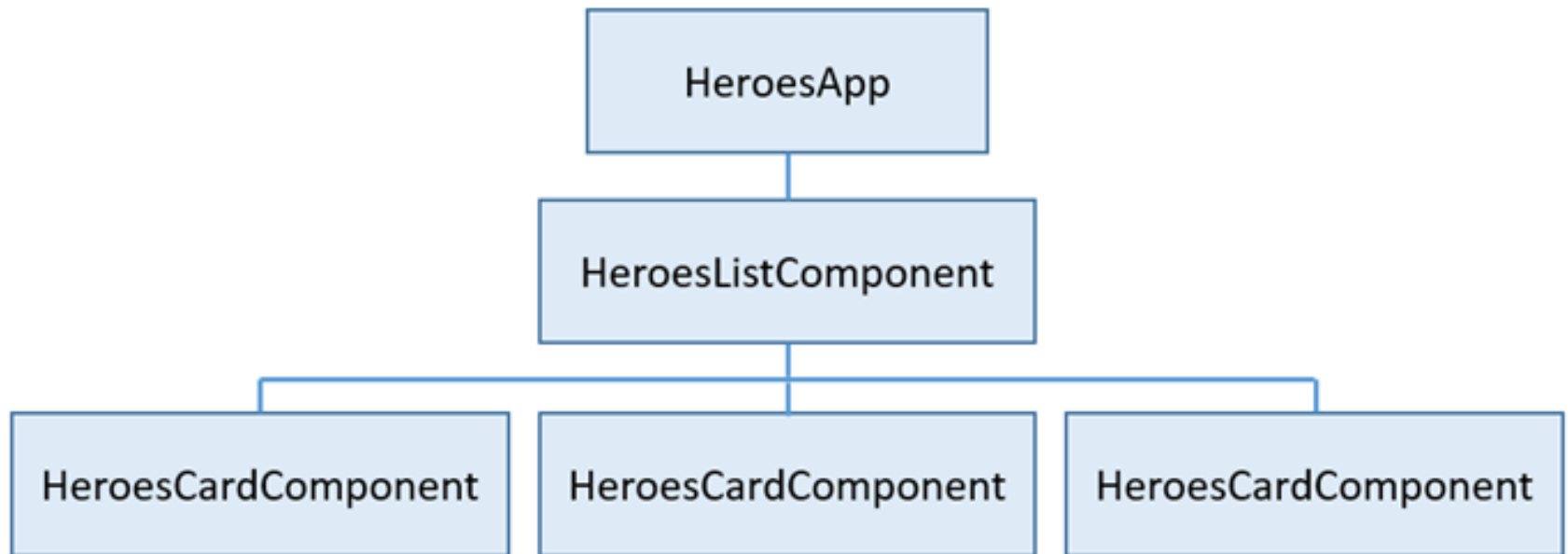
註冊為提供者 (僅提供給特定元件)

- 在元件中注入一個**全新的**服務元件實體

```
import { HeroService } from '../hero.service';

@Component({
  selector: 'my-heroes',
  templateUrl: 'some_template_url',
  providers: [HeroService],
  directives: [HeroListComponent]
})
export class HeaderComponent {
  constructor(herosvc: HeroService) {
  }
}
```

注入器的獨體模式 (Singleton)



HTTP CLIENT

使用 HTTP 服務元件



從 AppModule 匯入 HttpClientModule 模組

app.module.ts x

```
1 | import { HttpClientModule } from '@angular/http';
2 | import { BrowserModule } from '@angular/platform-browser';
3 | import { NgModule } from '@angular/core';
4 |
5 | import { AppRoutingModule } from './app-routing.module';
6 | import { AppComponent } from './app.component';
7 |
8 | @NgModule({
9 |   declarations: [
10 |     AppComponent
11 |   ],
12 |   imports: [
13 |     BrowserModule,
14 |     HttpClientModule,
15 |     AppRoutingModule
16 |   ],
17 |   providers: [],
18 |   bootstrap: [AppComponent]
19 | })
20 | export class AppModule { }
21 |
```

記得要 import 模組物件進來

Http 服務元件封裝在 HttpClientModule 模組內

從元件注入 Http 服務

- 匯入 Http 類別
 - `import { Http } from '@angular/http';`
- 注入 Http 服務
 - `constructor (private http: Http) {}`
- 發出 Http 要求 (GET)

```
this.http.get('/api/articles.json')  
  .subscribe(res => {  
    this.data = res.json(); // 將 JSON 轉為 JS 物件  
  });
```

發出 HTTP GET 要求與訂閱執行結果

```
import { Http, Response } from '@angular/http';

this.http.get('/api/articles.json')
  .subscribe(
    (value: Response) => {
      this.data = value.json();
    },
    (error: any) => {
      this.error = error;
    }
  );
```

[Angular2 - set headers for every request - Stack Overflow](#)

關於 .subscribe() 的用法

```
this.http.get('/api/articles.json')  
  .subscribe(  
    // 當 Observable 有資料出現時執行  
    nextFn,  
    // 當 Observable 有錯誤發生時執行  
    errorFn,  
    // 當 Observable 完成時執行  
    completeFn  
  );
```

發出 HTTP POST 要求與訂閱執行結果

```
import { Http, Response } from '@angular/http';

let data = { title: 'Angular 4 開發實戰', author: 'Will' };

this.http.post('http://localhost:3000/posts', data)
  .subscribe(
    (value: Response) => {
      this.data = value.json();
    },
    (error: any) => {
      this.error = error;
    }
  );
```

發出 HTTP POST 要求 (自訂標頭)

```
import { Http, Response } from '@angular/http';
import { Headers, RequestOptions } from '@angular/http';

let headers = new Headers({ 'Content-Type': 'application/json' });
let options = new RequestOptions({ headers: headers });

this.http.post('/api/article', data, options)
  .subscribe(
    (value: Response) => {
      this.data = value.json();
    },
    (error: any) => {
      this.error = error;
    }
  );
```

相關連結

- [Angular 官網](#) ([官方簡體中文翻譯](#))
- [Angular 風格指南](#) (官方版)
- [Angular 學習資源](#) (官方版)
- [Angular 學習資源](#) (社群版)
- [ng-conf 2016 - YouTube](#) / [ng-conf 2017 - YouTube](#)

- [Angular 2 Fundamentals | AngularClass](#) (免費 Angular 線上課程)

- [ReactiveX](#) ([RxJS on GitHub](#))
- [RxMarbles: Interactive diagrams of Rx Observables](#)

- [TypeScript - JavaScript that scales.](#) / [TypeScript Handbook \(中文版\)](#)

- [前端工程的夢幻逸品：Angular 2 開發框架介紹](#)
- [Angular User Group Taiwan](#) (台灣 Angular 開發者社群)

聯絡資訊

- The Will Will Web

記載著 Will 在網路世界的學習心得與技術分享

- <http://blog.miniasp.com/>

- Will 保哥的技術交流中心 (臉書粉絲專頁)

- <http://www.facebook.com/will.fans>



- Will 保哥的噗浪

- <http://www.plurk.com/willh/invite>

- Will 保哥的推特

- https://twitter.com/Will_Huang