

Angular 7 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 ➔ **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+ ➔ **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

主要特色 (1) - 跨平台

- Progressive Web Apps ([PWA](#))
 - 結合**網頁**和**應用程式**優點於一身的絕佳體驗
 - [Angular - Getting started with service workers](#)
- 桌面應用程式 (Desktop Apps)
 - 可搭配 [Electron](#) 框架開發出跨越 Windows, Mac, Linux 的桌面應用程式
 - [Getting Started with Angular and Electron ← Alligator.io](#)
- 原生行動應用程式 (Native Apps)
 - 可搭配 [NativeScript](#), [React Native](#) 開發跨行動平台原生應用程式
- 混合式行動網站應用程式 (Hybrid Mobile Apps)
 - 可使用 [Ionic Native](#) 擴充行動版網站的原生支援能力

主要特色 (2) - 速度與效能

- 程式碼產生 ([AOT](#))
 - 將元件範本預先編譯成 JS 程式碼，對網頁應用程式進行優化
 - [Ahead-of-Time Compilation in Angular · Minko Gechev's blog](#)
 - 偵測變更：比 ng1 快 10 倍
 - 渲染速度：比 ng1 快 5 倍 (Render & Re-render)
- 伺服器端渲染 ([Angular 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
- 程式碼拆分 (Code Splitting)
 - 使用者瀏覽網頁時只須載入需要的程式碼，應用程式會自動分割程式碼
 - 內建的 **延遲載入** (Lazy Loading) 機制，可依照需要載入需要的程式碼
 - 透過全新的元件路由機制，Angular 可以實現快速載入每個頁面

主要特色 (3) - 生產力

- Angular CLI
 - 透過**命令列工具**快速建立專案範本、建立元件、執行單元測試與 E2E 測試
 - [CLI 命令參考手冊](#)
- 範本引擎 ([Templates](#))
 - 使用簡易與強大的範本語法提高開發效率
- 整合開發環境 (IDE)
 - 完整的 Angular 語言服務 ([Angular Language Service](#))
 - [Visual Studio Code](#)
 - [Sublime Text](#)
 - [WebStorm](#)
 - 讓你在範本中一樣享有**程式碼自動完成**與**即時錯誤檢查**
 - 可使用 **F12** 快速跳轉到參考的程式碼

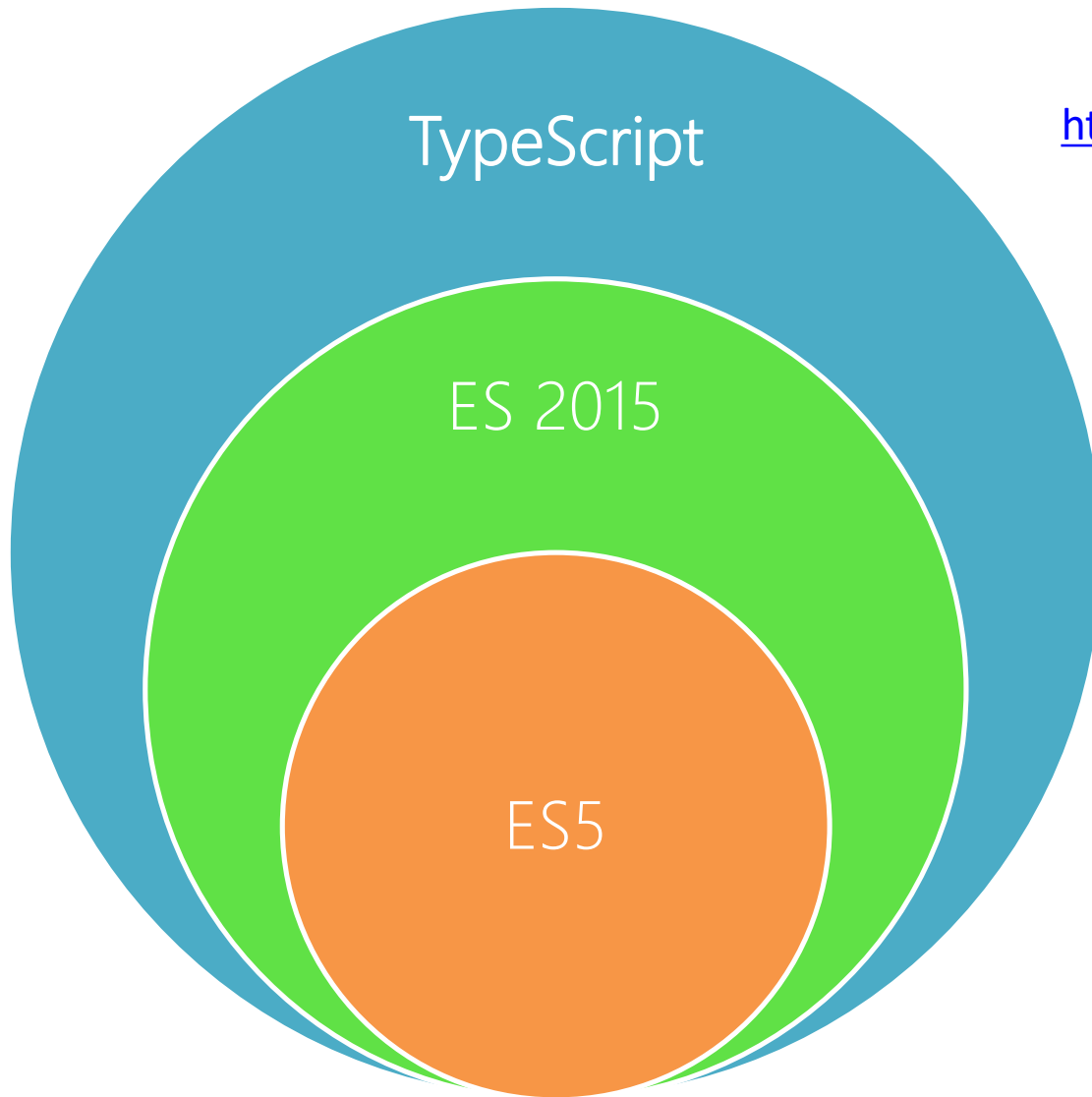
主要特色 (4) - 完整開發體驗

- 完整的測試開發體驗 ([Testing](#))
 - 採用 [Jasmine](#) 測試框架與 [Karma](#) 測試執行器
 - 結合 [Protractor](#) 執行各種 E2E 測試情境
- 完整的動畫開發體驗 ([Animations](#))
 - 透過 Angular 直觀的 API 完成複雜的頁面動畫效果
 - Angular 的動畫系統是基於 CSS 功能建構而成的，這意味著你可以 "動" 瀏覽器認為可動的任何屬性，其中包括位置、大小、變形、顏色、邊框等。
- 可及性設計 (Accessibility)
 - 透過支援 ARIA 的元件、開發者指南和內建的測試基礎架構，建立具有完備 [A11Y](#) 可及性的應用程式。

選擇 Angular 的理由

- 企業級前端框架 (Enterprise-grade Frontend Framework)
 - 使用強型別的 TypeScript 程式語言
 - 使用以類別為基礎的物件導向架構開發 Web 應用程式
 - 採用免費、高效、強大的 Visual Studio Code 開發工具
- 超高生產力 (Super High Productivity)
 - 內建程式碼產生器工具 ([Angular CLI](#)) 改善開發流程
 - 內建 [codelyzer](#) 靜態程式碼分析工具確保開發品質
 - 擁有強大的開發偵錯工具 [Augury](#) ([Change Logs](#))
 - 擁有 [AngularDoc](#) 全自動分析與產出開發文件
- 穩定的版本升級策略
 - [擁有穩定的 Angular 版本與釋出政策](#)
 - 每個版本升級皆可搭配 **ng update** 自動更新

開發程式語言



<http://www.typescriptlang.org/>



[AngularDart](#)

推薦的開發工具

- 免費的開發工具

- [Visual Studio Code](#)

免費的開源軟體

- 付費的商用開發工具

- [WebStorm](#)

地表最強 Angular 開發工具

- 免費的線上開發工具

- [StackBlitz](#)

與 Angular 整合的非常好

- [CodeSandbox](#)

另一套強大的免費線上開發工具

Angular 應用程式的組成

模組

- AppModule

元件

- App Component

元件

- Child Component

元件

- Service 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

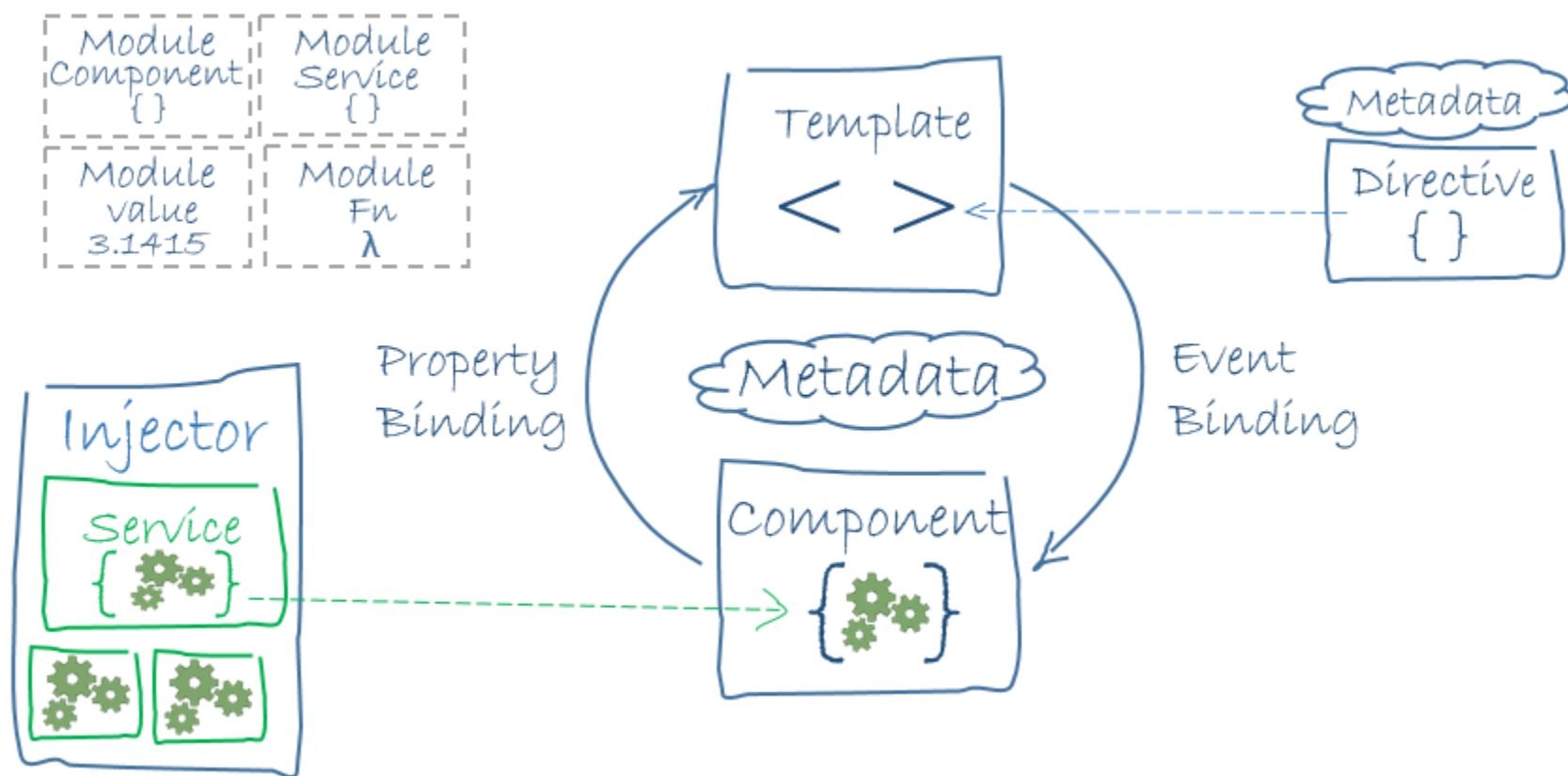
```
@Component({  
  selector: 'app-root',  
  templateUrl: 'app.component.html',  
  styleUrls: ['app.component.css']  
})
```

裝飾器

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

類別

剖析 Angular 架構 (圖示)



Setup your Angular development environment

建立 Angular 開發環境



準備 Angular 開發環境

- 請參考 [Angular 7 開發環境說明](#) 進行安裝設定
 - [Google Chrome](#)
 - [VSCode](#)
 - [Git](#)
 - [Node.js](#)
 - [Angular CLI](#)
- 建立第一個 Angular 專案
 - `ng new demo1 --routing --style css`
 - `cd demo1`
 - `npm start`
 - <http://localhost:4200/>

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](#) 至少 v10.9 以上版本
- 安裝方式
 - `npm install -g @angular/cli`
- 升級方法 (Global package)
 - `npm install -g @angular/cli`
- 升級方法 (Local project package)
 - 使用 `ng update` 顯示升級提示
 - `ng update @angular/cli`
- Angular CLI 版本變更紀錄：[CHANGELOG.md](#)

使用 Angular CLI 啟動開發伺服器

- 建立新專案並啟動開發伺服器
 - 執行 `npm start` 或 `ng serve` 皆可
 - 使用 `ng serve --open` 可自動開啟瀏覽器執行 (預設為 **4200** 埠號)
- 指定不同埠號啟動開發伺服器
 - `ng serve --port 3000`
- 在 Production 模式進行測試 (建置時間較長)
 - `ng serve --prod`
- 啟動含有 SSL 自簽憑證的 HTTPS 開發伺服器
 - `ng serve --ssl`
- 啟動可以讓外部連線的開發伺服器 (例如同網段下的其他行動裝置)
 - `ng serve --host 0.0.0.0 --disable-host-check`
- 在特定目錄下啟動開發伺服器
 - `ng serve --serve-path /app1/ --base-href /app1/`

使用信任的自簽憑證進行網站測試

- 使用 OpenSSL 建立自簽憑證
 - 請參考以下文件進行設定
<https://gist.github.com/doggy8088/9bc80c2872bd91399c257acddfc542c6>
- 啟動含有 SSL 自簽憑證的 HTTPS 開發伺服器
 - `ng serve --ssl --ssl-key server.key --ssl-cert server.crt`
- 注意事項
 - 請務必記得要將憑證匯入到「**受信任的根憑證授權單位**」
 - 匯入完成後 Google Chrome 瀏覽器可能不會立刻顯示這是個有效憑證 (快取的關係)，但你只要過一段時間重開 Chrome 瀏覽器，即可看見網址列的變化，不會再出現紅色不安全的提示。

使用 Angular CLI 快速產生程式碼

- 透過 Schematic 產生程式碼

- `ng generate <schematic> [options]`
- `ng g <schematic> [options]`

- 使用範例

- 產生 HeaderComponent 元件
 - `ng g component header`
 - `ng g c header`
 - `ng g c header --skipTests`
 - `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`
 - `ng g m member -m app`
- 查詢其他 Schematics 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 `charts` 模組

建立 `member` 模組並自動匯入 `AppModule`

顯示所有 Schematics 與用法說明

常見的 Angular CLI 程式碼產生器

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>
Guard	<code>ng g guard my-guard</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

- **ng build** 預設為 dev 開發模式
- **ng build --prod** 使用 production 模式進行建置

※ 建置的輸出目錄可在 **angular.json** 檔案的 "**outputPath**" 進行設定

- 建置專案的過程中，如果是 dev 模式會使用以下檔案
 - src/environments/environment.ts
- 建置專案的過程中，如果是 prod 模式會使用以下檔案
 - src/environments/environment.**prod**.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](#))
 - 由於 `package.json` 裡 "scripts" 有個 `lint` 命令，因此你也可以執行 **npm run lint** 執行相同命令！
- 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
 - 你只要安裝 [Angular Extension Pack](#) 就都有了！

Build your first Angular Application

建立 Angular 應用程式



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/styles.css` 全站共用的 CSS 樣式檔
 - `src/main.ts` 預設 Angular 主要程式進入點
 - `src/polyfills.ts` 增加瀏覽器相容性的 Polyfills 函式庫
- 公用檔案資料夾
 - `src/assets/` 網站相關的靜態檔案 (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 --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] ← 宣告啟動元件 (根元件)
17 })
18 export class AppModule { }
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);
  }
}
```

各種不同的命名規則

- lowercase
- UPPERCASE
- MiXeDCaSe
- camelCase
- PascalCase / UpperCamelCase
- Snake_Case
- lower_snake_case
- kebab-case / skewer-case

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 --skipTests`
 - 這個命令會建立 `HeaderComponent` 元件類別
- 會自動在 `app.module.ts` 匯入 `declarations` 宣告
 - `import { HeaderComponent } from './header/header.component';`
- 在任意元件範本中可使用 **Directives** 語法載入此元件
 - `<app-header></app-header>`

資料繫結的四種方法 (範本語法)

- 內嵌繫結 (interpolation)

`{{statement}}`

- 屬性繫結 (Property Binding)

`[propertyName]="statement"`

`[attr.attributeName]="statement"`

- 事件繫結 (Event Binding)

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

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

- 雙向繫結 (Two-way Binding)

`[(ngModel)]= 'property'`

關於事件過濾器 (Event Filters)

- 所有 [KeyboardEvent](#) 的 [key](#) 屬性值都可以使用
- 可用的 key 別名在此：[browser_adapter.ts](#)
- 主要程式碼：[getEventKey\(event: any\)](#)
- 使用範例
 - `(keyup.enter)="someMethod($event)"`
 - `(keyup.escape)="someMethod($event)"`
 - `(keyup.control.c)="someMethod($event)"`
 - `(keyup.control.alt.a)="someMethod($event)"`
 - `(keyup.control.meta)="someMethod($event)"`

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>`
 - `<ng-template [ngIf]="currentHero">`
 `<hero-detail [hero]="currentHero"></hero-detail>`
 `</ng-template>`
- 因此套用 * 星號其實是套用 `<ng-template>` 標籤的語法糖

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
 - 在範本中使用 **`$event`** 代表子元件傳出的資料 (可以是任意資料類型)
 - 推薦閱讀：[設計 @Output 時的命名注意事項](#)

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 可實作 OnChanges 介面 此方法會得到變更物件的目前值與先前的值
ngDoCheck	當 Angular 每次執行 變更偵測 時執行 (會影響效能)
ngOnDestroy	當 Angular 要摧毀元件時執行 可實作 OnDestroy 介面 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！(參考範例)

使用 Pipes 管線元件

- Angular 內建的 Pipes 元件
 - UpperCasePipe (uppercase) / LowerCasePipe (lowercase)
 - TitleCasePipe (titlecase) / PercentPipe (percent)
 - DecimalPipe (number) / DatePipe (date)
 - CurrencyPipe (currency)
 - {{ product.price | currency:'USD':true:'1.2-2' }}
 - 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)
 - KeyValuePipe (keyvalue)
- 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

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 --skipTests`

```
G:\Projects\demo1>ng g s hero --skipTests  
  
CREATE src/app/hero.service.ts (133 bytes)  
  
G:\Projects\demo1>
```

Angular 5 以前須註冊為全域提供者

- 編輯 app/app.module.ts 檔案

```
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  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
```

記得要 import 元件進來

服務元件必須註冊為「提供者」

Angular 6 以後預設就會註冊至全域

- 設定 `@Injectable({ providedIn: 'root' })`



The screenshot shows a VS Code editor window with two tabs: `app.component.ts` and `hero.service.ts`. The `hero.service.ts` tab is active, showing the following code:

```
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class HeroService {
7
8   constructor() { }
9 }
10
```

A red arrow points to the `providedIn: 'root'` property in the `@Injectable` decorator, with the text "設定服務元件自動註冊到全域提供者" (Set service component to automatically register to global provider) next to it.

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(private herosvc: HeroService) {  
  }  
}
```

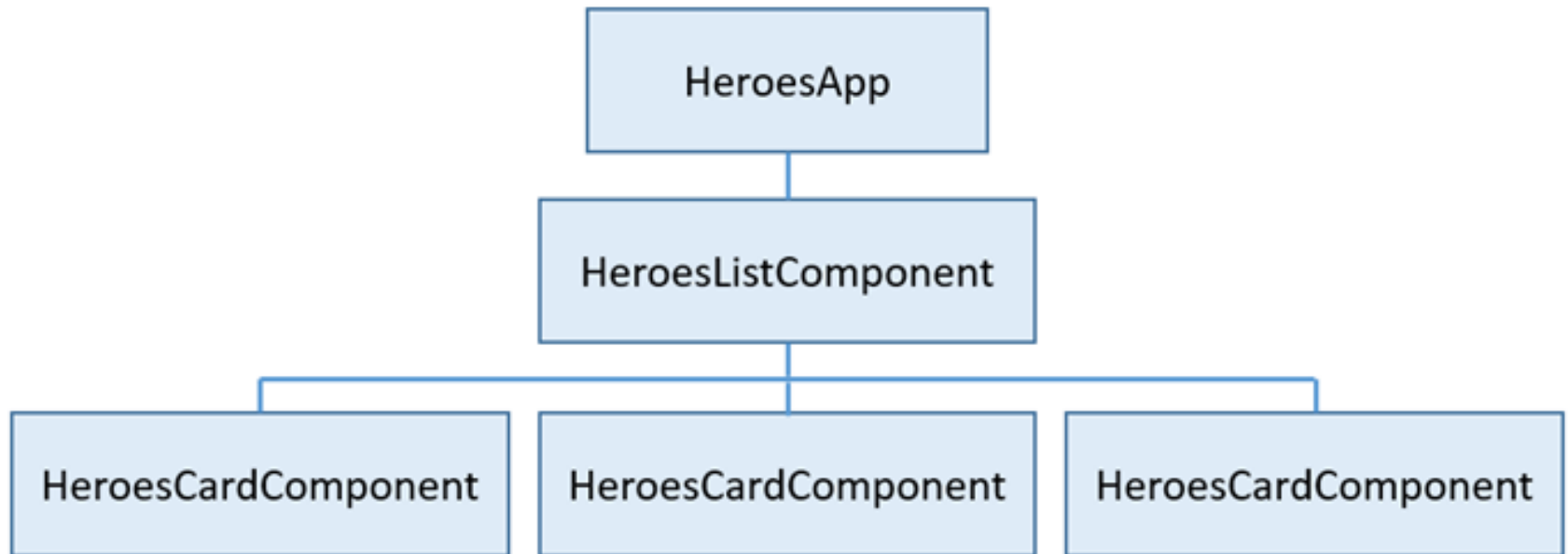

註冊為提供者 (僅提供給特定元件)

- 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)



[Angular - HttpClient](#)

使用 HttpClient 服務元件



準備工作 (從 AppModule 匯入 HttpClientModule 模組)

app.module.ts

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

記得要 import 模組物件進來

服務元件已經封裝在 HttpClientModule 模組內

從元件注入 Http 服務

- 匯入 Http 類別

- `import { HttpClient } from '@angular/common/http';`

- 注入 Http 服務

- `constructor (private http: HttpClient) {}`

- 發出 Http 要求 (GET)

- `this.http.get('/api/articles.json')`
 `.subscribe(data => {`
 `this.data = data;`
 `});`

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  
  );
```

HttpClient 最佳實務

- 將所有 API 操作集中在 Service 元件中管理
 - 所有方法一律回傳 Observable 型別
- 注入 HttpClient 的元件才能進行 subscribe 動作
 - 千萬不要在 Service 元件中執行 subscribe 動作
 - 要注意 JavaScript 原生非同步的特性
- 視情況使用 RxJS 處理複雜的 API 呼叫
 - 常用的 RxJS 運算子：[tap](#)、[map](#)、[share](#)、[switchMap](#)、[forkJoin](#)

發出 HTTP POST 要求與訂閱執行結果

```
import { HttpClient } from '@angular/common/http';

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

// 預設 data 物件會自動進行 JSON 序列化
this.http.post('http://localhost:3000/posts', data)
  .subscribe(
    (value) => { this.data = value; },
    (error) => { this.error = error; }
  );
```


發出 HTTP POST 要求 (自訂標頭)

```
import { HttpClient, HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json',
    'Authorization': 'my-auth-token'
  })
};

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

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

相關連結

- 官方資源
 - [Angular 官網](#) ([官方正體中文翻譯](#))
 - [Angular 風格指南](#)
 - [Angular 學習資源](#)
 - [Angular Release Notes](#) / [Angular CLI Release Notes](#)
 - [Angular Weekly Meeting Notes](#)
- 線上社群
 - [Angular Taiwan](#) (台灣 Angular 開發者社群)
 - [台灣 Angular 技術論壇](#)
- 部落格
 - [Angular | The Will Will Web](#)
 - [Angular | CK's Notepad](#)
 - [全端開發人員天梯](#)

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