

A dark blue vertical bar on the left side of the slide. A blue arrow points to the right from the bar, containing the date.

2018/5/2

Angular Fundamentals

Ultimate Angular: Todd

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

TONY

目錄

1. Architecture, setup, source files.....	6
1.1. Angular Architecture overview.....	6
1.1.1. Module	6
1.1.2. Component.....	6
1.1.3. Component.....	7
1.1.4. Directive	7
1.1.5. Service	7
1.1.6. Routing.....	8
1.2. Setup instructions	8
1.3. Source Files	8
1.4. Notable changes in v5	9
2. ES5 to ES6 and TypeScript refresher	9
2.1. Refresher video overview	9
2.2. Classes, Spread operator, Arrow functions, Immutability	9
2.3. Understanding import and exports	10
3. Getting started	10
3.1. Getting started	10
3.2. First component with @Component.....	11

3.3.	Root module with @NgModule	12
3.4.	Bootstrapping Angular.....	13
4.	Template fundamentals	14
4.1.	Interpolation and expressions.....	14
4.2.	Property binding.....	15
4.3.	Event binding	16
4.4.	Two-way databinding	17
4.5.	Template #ref variables	19
5.	Rendering flows	19
5.1.	ngIf, * syntax and <ng-template>	19
5.2.	ngFor and iterating collections	20
5.3.	ngClass and className bindings.....	21
5.4.	ngStyle and style bindings.....	22
5.5.	Pipes for data transformation.....	23
5.6.	Safe navigation operator.....	25
6.	Component Architecture and Feature Modules	26
6.1.	Smart and dumb components overview.....	26
6.2.	One-way dataflow overview	27
6.3.	Feature modules with @NgModule	28

6.4.	Creating a container (smart) component	29
6.5.	ngOnInit lifecycle hook	30
6.6.	Presentational (dumb) components	31
6.7.	Passing data into components with @Input	32
6.8.	Dynamic @Input values with *ngFor	32
6.9.	Emitting changes with @Output and EventEmitter	33
6.10.	Immutable state changes	35
6.11.	ngOnChanges lifecycle hook.....	36
7.	Services, Http and Observables.....	37
7.1.	Data Services and Dependency Injection	37
7.2.	Understanding @Injectable	39
7.3.	Http data fetching with Observables.....	39
7.3.1.	Update Http to HttpClient	41
7.4.	Http put, delete with immutable state	42
7.4.1.	使用 Http put 更新資料.....	42
7.4.2.	使用 Http delete 刪除資料.....	42
7.5.	Custom Headers and RequestOptions	43
7.6.	Http Promises alternative	43
7.7.	Observable.catch error handling.....	44

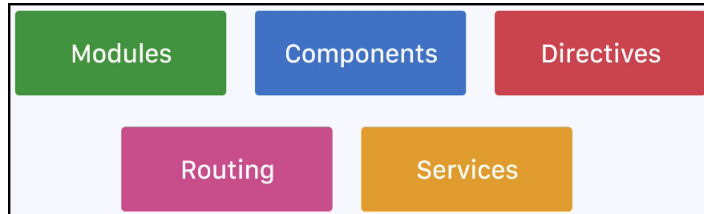
7.8.	HttpClient Retry	45
8.	Template-driven Forms, Inputs and Validation	45
8.1.	Forms container component	45
8.2.	Form stateless component	46
8.2.1.	修改 Passenger Interface and Json Server Data.....	46
8.2.2.	Create Form Component.....	46
8.2.3.	Passing Data To Form Component.....	46
8.2.4.	Import FormModule	46
8.3.	ngForm and ngModel	47
8.4.	Binding to radio buttons	48
8.4.1.	加入 radio buttons.....	48
8.4.2.	加入 radio buttons Event 去更動 TimeStamp	49
8.5.	Binding to checkboxes	50
8.6.	<select> option rendering, and ngValue.....	51
8.7.	Form validation and error states	52
8.7.1.	Input Field error states	52
8.7.2.	顯示錯誤訊息	54
8.7.3.	使用 dirty 及 touched 優化.....	54
8.8.	Dynamically disabling submit	55

8.9.	ngSubmit and stateless @Output	55
9.	Component Routing	56
9.1.	Base href and RouterModule	56
9.2.	Root module routes and outlet.....	57
9.3.	Wildcard routes for 404 handling	58
9.4.	Understanding routerLink.....	59
9.5.	Styling active routes	59
9.6.	Dynamic navigation with ngFor.....	60
9.7.	Feature-module routes with forChild()	61
9.8.	Child and dynamic routes	62
9.9.	Route params, data-fetching with switchMap	62
9.10.	Imperative routing API.....	63
9.11.	Hash location strategy	64
9.12.	Applying redirects.....	65

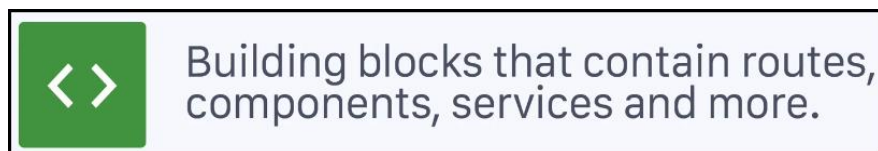
1. Architecture, setup, source files

1.1. Angular Architecture overview

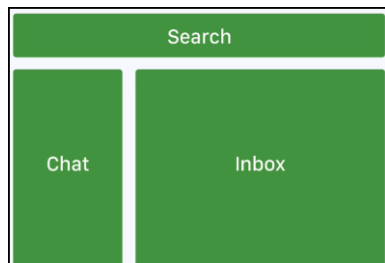
五個主要的功能



1.1.1. Module

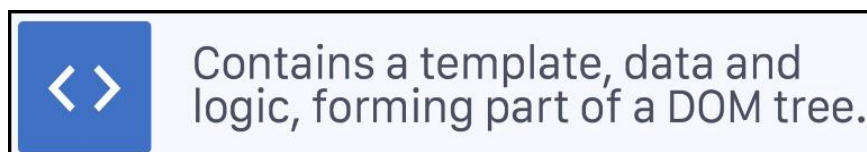


想像一下 mail 的 web, 有這三個 modules

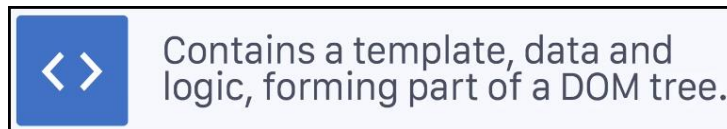


1.1.2. Component

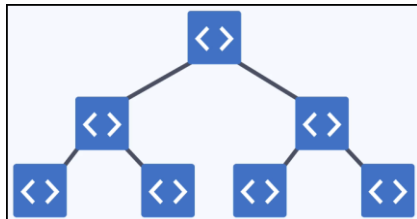
我自己的解釋 componet=view mode + template



1.1.3. Component



以 component 建立的 web, 就像一個 tree

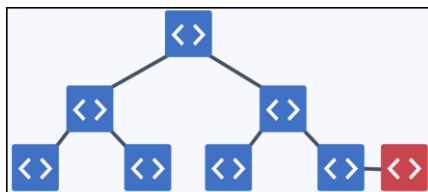


1.1.4. Directive

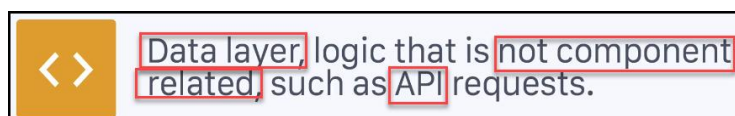
用來對 component 做一些附加的行為處理及 element 的處理.



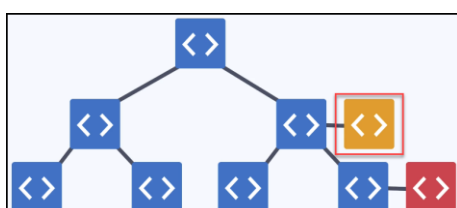
它的位置像這樣



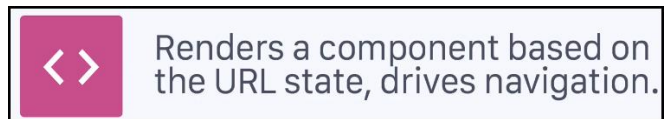
1.1.5. Service



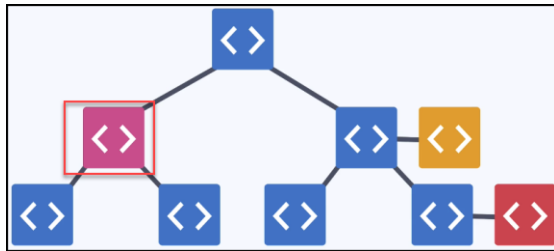
這張圖表示某一 component 需要一個 service 來處理資料



1.1.6. Routing



表示這個紫色的 routing 包含了 2 個普通的 component



1.2. Setup instructions

VS Code Plugin

1. [angular2-inline](#)
2. [Angular v5 Snippets](#)

Npm packages:

1. node-sass: `npm i -g node-sass`

1.3. Source Files

範例: [angular-fundamentals-seed](#)

[Source files](#)

1.4. Notable changes in v5

- `<template>` is now `<ng-template>`
- `Http` is now `HttpClient`
 - This change is almost identical but also comes with some additional typing options

2. ES5 to ES6 and TypeScript refresher

2.1. Refresher video overview

不要用 seed-project 來 study 影片中的範例, 因為 angular 版本不同, 會得到一些錯誤.

2.2. Classes, Spread operator, Arrow functions, Immutability

Clone seed-project 並安裝.

1. `npm i`
2. `npm start`

將 `main.ts` 清空, 開始在裡面練習 TypeScript.

程式中操作 `array` 的方式是 `Immutability`, 不會有繼承的問題.

2.3. Understanding import and exports

Export 函數

```
TS formatter.ts •
1  export function uppercase(str: string)
2    return str.toUpperCase();
3  }
```

Import 使用

```
import { uppercase } from './formatter'

let myName: string = 'Tony';

console.log(uppercase(myName));
```

3. Getting started

3.1. Getting started

使用 seed-project 會發生問題

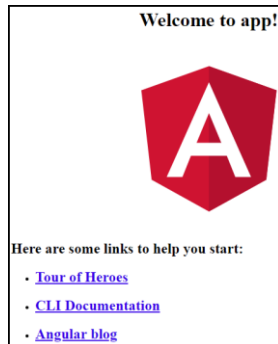
1. node-sass(node <V7)才能用
2. 需要 python2?

整個課程我想要用 angular-cli 重建.

1. 安裝 angular-cli
2. 建立 ng-fundamentals-seed

```
ng new ng-project-seed --skip-tests --skip-install --style scss
```

3. 回復套件: `npm i`
4. 執行: `ng serve -o`



3.2. First component with @Component

從零開始建立第 1 個 component.

1. 開啟 `app.ts`, 並刪掉全部的 code
2. 使用 angular v5 snippet 插件建立 component: `a-component`. 然後更名為 `AppComponent`.
3. 會看到使用 `@Component` 的 decorator 來修飾 `AppComponent` class.
4. 建立 `@Component` 中的元素. 並加入 `scss`.

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

export class AppComponent implements OnInit {
  title: string;
  constructor() {
    this.title = 'Ultimate Angular';
  }
}
```

`App.component.html`

```
<div class="app">{{title}}</div>
```

App.component.scss

```
.app {  
  background-color: red;  
}
```

然後就可在 browser 看到輸出了.

Ultimate Angular

3.3. Root module with @NgModule

Angular 使用 @NgModule decorator 來建立一個 module.

原型:

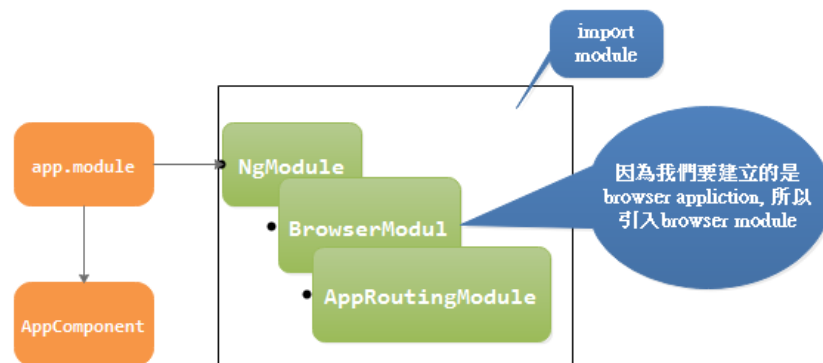
```
@NgModule({  
  providers?: Provider[]  
  declarations?: Array<Type<any> | any[]>  
  imports?: Array<Type<any> | ModuleWithProviders | any[]>  
  exports?: Array<Type<any> | any[]>  
  entryComponents?: Array<Type<any> | any[]>  
  bootstrap?: Array<Type<any> | any[]>  
  schemas?: Array<SchemaMetadata | any[]>  
  id?: string  
})
```

主要屬性及功能如下:

1. declarations: 宣告此 module 中所需要用到的 components. 也就是註冊 component 到 module.
2. imports: 指定一個模組列表, 列出哪些 directive/pipe 可以給這個 module 下的 template 使用
3. providers

4. bootstrap: 這個屬性只在 root module(convention 為 app.module)需要設定

Angular Root Module with NgModule



Tony

3.4. Bootstrapping Angular

建立好 root module(app.module)後, 要如何啟動 angular application?

因為我們使用 angular cli 的環境, 可以在 angular-cli.json 中看到啟動的設定:

```
"index": "index.html",  
"main": "main.ts",
```

也就是說第 1 個去載入的檔案會是 main.ts(也就是 main.js).

所以啟動程式必需在此檔案中去執行.

啟動器: 因為是 web application, 所以使用這個 module

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

利用啟動器的 bootstrapModule 來載入 root-module.

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

然後在 index.html 中的 app-root 會被 app.component 所渲染.

```
<body>  
  <app-root></app-root>  
</body>
```

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

整個應用程式就完成啟動.

4. Template fundamentals

4.1. Interpolation and expressions

[官方說明](#)

使用雙大括號來即時做插值.

```
src/app/app.component.html  
  
<p>My current hero is {{currentHero.name}}</p>
```

也可以在裡面運算

```
export class AppComponent {
  title: string;
  isHappy = false;
  numberOne = 1;
  numberTwo = 2;
}
```

```
<div>
  <p>{{numberOne}}</p>
  <p>{{numberTwo}}</p>
  <p>{{numberOne + numberTwo}}</p>
  <p>{{isHappy? '!' : ':(')}}</p>
</div>
```

=>

```
1
2
3
```

4.2. Property binding

[官方說明](#)

最常用的就是 HTML element 的 binding. 將 HTML element 的 property

binding 到 component property.

```
src/app/app.component.html
<img [src]="heroImageUrl">
```

```
src/app/app.component.html
<button [disabled]="isUnchanged">
```

範例: 二者輸出結果相同

```
<h1>{{title}}</h1>
<h1 [innerHTML]="title"></h1>
```

範例: img src property

```
logo = 'img/angular.svg';
```

```
<img [src]="logo" alt="" height="20%" width="20%">
```

範例: Input field, one-way binding. 即使改變 input value, 下方顯示的 name

也不會跟著變.


```
<input type="text" [value]="name">
<p [innerHTML]="name"></p>
```

```
export class AppComponent {
  name = 'Tony';
}
```

=>

Tonya1212

Tony

4.3. Event binding

[官方說明](#)

在 HTML element 中, 使用小括號來指定 event name, 並指派要呼叫的 component method.

```
src/app/app.component.html

<button (click)="onSave()" Save></button>
```

範例: \$event 是關鍵字, 表示在呼叫 handleBlur 時, 將目錄發生 blur 的 event 傳入 component method.

```
<input type="text"
[value]="name"
(blur)="handleBlur($event)">
```

```
handleBlur = (event: any): void => {
  console.log(event);
}
```

從 console 看到 \$event 是 FocusEvent

```
FocusEvent {isTrusted: true, relatedTarget: null, view: Win
nputDeviceCapabilities, ...}
```

而這個 input field 的值, 存在 target.value 中. 所以我們用它來更新 name 的

值, 所以在輸入完後做 blur, name 的值就會被更新

```
handleBlur = (event: any): void => {  
  this.name = event.target.value;  
  console.log(event);  
}
```

範例: 在 onInput event 發生後, 更新 name 的值. 所以就會一面打字一面即時

更新 name 的顯示.

```
<input type="text"  
  [value]="name"  
  (input)="handleInput($event)"  
  (blur)="handleBlur($event)">  
<p [innerHTML]="name"></p>
```

```
handleInput(event: any) {  
  this.name = event.target.value;  
}
```

範例: 使用 click event 強迫設定 name 的值.

```
<button (click)="handleClick()">強迫改名</button>
```

```
handleClick(event: any) {  
  this.name = '我的名字被改了!!!';  
}
```

4.4. Two-way databinding

Two way databinding - [官方說明](#)

ngModel two-way databinding – [官方說明](#)

也許你覺得用 event 來更新資料不好用, 想使用 two-way databinding.

但要使用前, 需先 module 中引入 FormModule. 以下在 app.module.ts 中引

入:

```
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
})
```

語法:

[()] = BANANA IN A BOX

範例: 直接使用[()]取代之前的 event 做法

```
<input type="text" [(ngModel)]="name">
```

但這種做法較沒有彈性. 官網是這麼說的

The [(ngModel)] syntax can only *set* a data-bound property. If you need to do something more or something different, you can write the expanded form.

建議的做法是使用 ngModel 的語法糖, 但是用 one way binding 的做法, 並取

代使用[value]及(input)的處理方式:

```
src/app/app.component.html

<input
  [ngModel]="currentHero.name"
  (ngModelChange)="setUppercaseName($event)">
```

範例: 使用 ngModel 做 one way data binding.(使用 ngModelChange)

```
<input type="text"
  [ngModel]="name"
  (ngModelChange)="handleChange($event)">
```

注意!! 傳入的是 string 而不是\$event

```
handleChange(event: string) {
  this.name = event;
}
```

4.5. Template #ref variables

[官方說明](#)

Angular 提供了一個在 html template 中的 input element 的 reference 功能.

只要在 input 中加入 #ref, 在此 html 中任何地方都可以用 #ref 來取值.

範例: 使用 #ref 方式更新 name 的值

```
<button (click)="handleClick(username.value)">異動姓名</button>
<input type="text" #username>
<p [innerHTML]="name"></p>
```

```
handleClick(newName: string) {
  this.name = newName;
}
```

5. Rendering flows

5.1. ngIf, * syntax and <ng-template>

[官方說明](#)

ngIf: 一個內建的 directive, 原型如下:

```
@Directive({ selector: '[ngIf]' })
class NgIf {
  constructor(_viewContainer: ViewContainerRef, templateRef: TemplateRef<NgIfContext>)
  set ngIf: any
  set ngIfThen: TemplateRef<NgIfContext> | null
  set ngIfElse: TemplateRef<NgIfContext> | null
}
```

語法:

```
<div *ngIf="show">Text to show</div>
```

範例: 如果輸入的名字長度大於 2, 下方即顯示 search for ...

```
<div *ngIf="name.length > 2">
  <p>Searching for... {{name}}</p>
</div>
```

也可以用 web component 的 template. 但自 ng5 之後, tag 要改為使用 ng-template, 而且要使用 property binding 的方式.

```
<ng-template [ngIf]="name.length > 2">
  <p>Searching for... {{name}}</p>
</ng-template>
```

5.2. ngFor and iterating collections

[官方說明](#)

ngFor 原型

```
@Directive({ selector: '[ngFor][ngForOf]' })
class NgForOf<T> implements DoCheck, OnChanges {
  constructor(_viewContainer: ViewContainerRef, _template: TemplateRef<NgForOfContext<T>>, _differs:
  IterableDiffers)
    ngForOf: NgIterable<T>
    ngForTrackBy: TrackByFunction<T>
    set ngForTemplate: TemplateRef<NgForOfContext<T>>
    ngOnChanges(changes: SimpleChanges): void
    ngDoCheck(): void
  }
```

語法:

```
<li *ngFor="let user of userObservable | async as users; index as i; first as isFirst">
  {{i}}/{{users.length}}. {{user}} <span *ngIf="isFirst">default</span>
</li>
```

範例: 使用 ngFor 建立旅客名單

```

<div>
  <h3>Airline Passengers</h3>
  <ul>
    <li *ngFor="let passenger of passengers; let i = index">
      {{i}}:{{passenger.fullname}}
    </li>
  </ul>
</div>

```

使用 web component

```

<ul>
  <ng-template ngFor let-passenger let-i="index" [ngForOf]="passengers">
    <li>
      {{i}}:{{passenger.fullname}}
    </li>
  </ng-template>
</ul>

```

5.3. ngClass and className bindings

[官方說明](#)

語法

```

<some-element [ngClass]="['first second']">...</some-element>

<some-element [ngClass]="['first', 'second']">...</some-element>

<some-element [ngClass]="{'first': true, 'second': true, 'third': false}">...
</some-element>

<some-element [ngClass]="stringExp|arrayExp|objExp">...</some-element>

<some-element [ngClass]="{'class1 class2 class3' : true}">...</some-element>

```

用 property binding 來對有 check-in 的客戶的樣式作格式化: 名字前顯示綠燈.

Scss 的樣式. 這裡用了一個 [Sass Nesting](#) 技巧, 可以方便的設定紅燈與綠燈.

相當於有 2 種樣式:

.status

.status.checked-in

```
.status {  
  width: 10px;  
  height: 10px;  
  background: #c0392b;  
  &.checked-in {  
    background: #2ecc71;  
  }  
}
```

使用 property binding, 用 checkedIn property 去設定 checked-in class. 它

的缺點是只能設定一個 class.

```
<li *ngFor="let passenger of passengers; let i = index">  
  <span class="status"  
    [class.checked-in]="passenger.checkedIn"></span>  
    {{i}}:{{passenger.fullname}}  
</li>
```

使用 ngClass 可以同時套用多個 css.

```
border-radius: 50%;  
&.checked-in {  
  background: #2ecc71;  
}  
&.checked-out {  
  background: #c0392b;  
}
```

```
<li *ngFor="let passenger of passengers; let i = index">  
  <span class="status"  
    [ngClass]="{  
      'checked-in': passenger.checkedIn,  
      'checked-out': !passenger.checkedIn  
    }"></span>  
    {{i}}:{{passenger.fullname}}  
</li>
```

5.4. ngStyle and style bindings

[官方說明](#)

語法

```
<some-element [ngStyle]="{'font-style': styleExp}">...</some-element>  
  
<some-element [ngStyle]="{'max-width.px': widthExp}">...</some-element>  
  
<some-element [ngStyle]="objExp">...</some-element>
```

範例: 使用 style property binding

```
<li *ngFor="let passenger of passengers; let i = index">
  <span class="status"
    [style.backgroundColor]="(passenger.checkedIn? '#2ecc71' : '#c0392b')"></span>
    {{i}}:{{passenger.fullname}}
</li>
```
















範例: 使用 ngStyle, 可以有多個 style

```
<li *ngFor="let passenger of passengers; let i = index">
  <span class="status"
    [ngStyle]="{
      'backgroundColor': (passenger.checkedIn? '#2ecc71' : '#c0392b'),
      'color': (passenger.checkedIn? 'gray' : 'blue'),
      'font-size': (passenger.checkedIn? '20px' : '10px')
    }">
    >Y</span>
    {{i}}:{{passenger.fullname}}
</li>
```

5.5. Pipes for data transformation

[官方說明](#)

[內建 pipe:](#)

 AsyncPipe	 CurrencyPipe	 DatePipe
 DecimalPipe	 DeprecatedCurrencyPipe	 DeprecatedDatePipe
 DeprecatedDecimalPipe	 DeprecatedPercentPipe	 I18nPluralPipe
 I18nSelectPipe	 JsonPipe	 LowerCasePipe
 PercentPipe	 SlicePipe	 TitleCasePipe
 UpperCasePipe		

1. 可以將 pipe 視為 function, 傳入一個值然後回傳一個全新不同的格式的值.

舉例來說, 就像在 JavaScript 中寫一個字串轉大寫的函數

```
function uppercase(string) {
  return string.toUpperCase();
}

var name = uppercase('todd');
console.log(name); // TODD
```

2. 還可以 chain pipe (date | date | uppercase)

範例: 使用 json pipe 顯示 component 的 data

```
<li *ngFor="let passenger of passen
  <span class="status"
    [style.backgroundColor]="(passeng
      {{i}}:{{passenger.fullname}}
      <pre>{{passenger | json}}</pre>
    </li>
```

範例: 使用 date pipe 將 unix timestamp 顯示成 date 字串

```
<p>{{passenger.checkInDate | date}}</p>
</li>
```

Airline Passengers

● 0:Stephen

```
{ "id": 1, "fullName": "Stephen", "checkedIn": true, "checkInDate": 1490742000000 }
```

Mar 29, 2017

加格式化

```
<p>Checked In Date: {{passenger.checkInDate | date: 'y/MMMM/d'}}</p>
</li>
```

● 0: Stephen

```
{ "id": 1, "fullName": "Stephen", "checkedIn": true, "checkInDate": 1490742000000 }
```

Check in date: March 29, 2017

範例: 若沒有 checkin date, 則輸出 Not checked in

```
<p>Checked In Date:
  {{passenger.checkedIn? (passenger.checkInDate | date: 'y/MMMM/d'):'
    'Not checked in'}}
</p>
```

● 0:Stephen

```
{ "id": 1, "fullName": "Stephen", "checkedIn": true, "checkInDate": 1490742000000 }
```

Checked In Date: 2017/March/29

● 1:Rose

```
{ "id": 2, "fullName": "Rose", "checkedIn": false, "checkInDate": null }
```

Checked In Date: Not checked in

範例: chain pipe

```
(passenger.checkInDate | date: 'y/MMMM/d' | uppercase)
```

```
● 0:Stephen
{ "id": 1, "fullName": "Stephen", "checkedIn": true }
Checked In Date: 2017/MARCH/29
```

5.6. Safe navigation operator

[官方說明](#)

為了避免在 html 中 binding 的 property 有 null 或 undefined 的值, 可以使用這個運算子來防止 error.

```
src/app/app.component.html
The current hero's name is {{currentHero?.name}}
```

範例: 沒有 safe navigation 時, 發生錯誤的語法

```
<div class="children">
  Children: {{passenger.children.length}}
</div>
```

```
▼ERROR TypeError: Cannot read
property 'length' of null
    at Object.eval [as updateRenderer]
```

範例: 加入 safe navigation operator

```
<div class="children">
  Children: {{passenger.children?.length}}
</div>
```

```
● 0:Stephen
{ "id": 1, "fullName": "Stephen", "checkedIn": true }
Checked In Date: 2017/MARCH/29
Children:
● 1:Rose
{ "id": 2, "fullName": "Rose", "checkedIn": false }
Checked In Date: Not checked in
Children: 2
```

範例: 加入 safe navigation operator 並使用類似 C# nullcheck 的語法, 若為 null 則用指定的值, 這裡設為 0

```
<div class="children">
  | Children: {{passenger.children?.length || 0}}
</div>
```

```
● 0:Stephen
{ "id": 1, "fullna
Checked In Dat
Children: 0
```

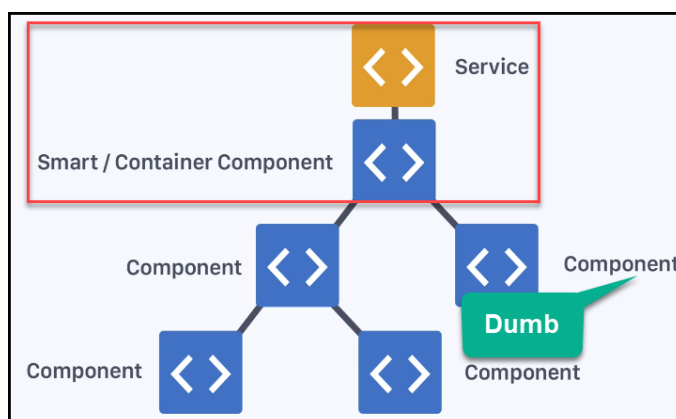
6. Component Architecture and Feature Modules

6.1. Smart and dumb components

overview

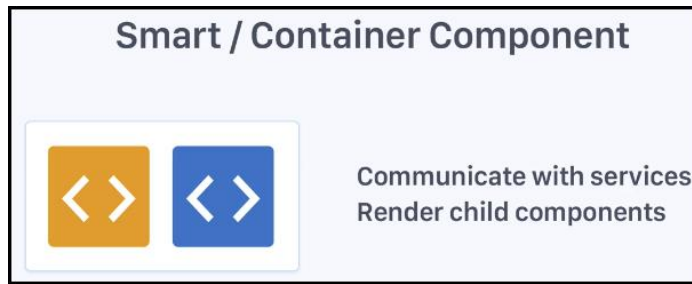
可以將 component 分為 2 類:

1. Smart/Container component
2. Dumb component



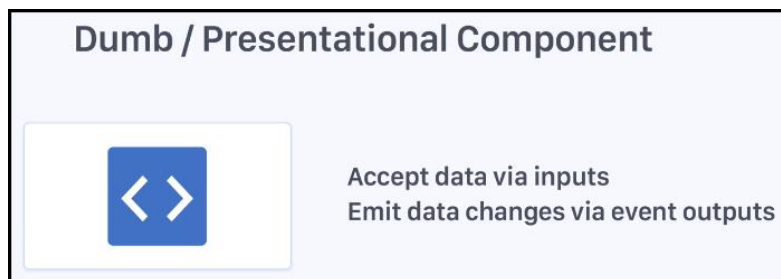
Smart/Container Component

主要用來與 service 溝通及繪製 child components



Dumb/Presentation Component

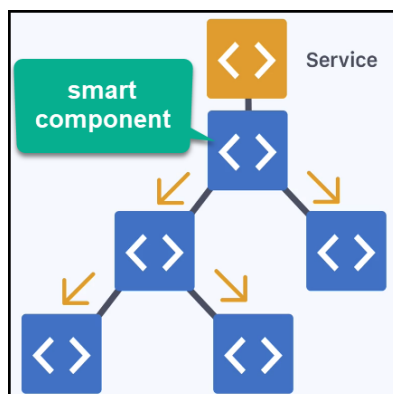
主要是接受由 Smart/Container Component 輸入的資料或是將本身有被使用者輸入或改變的資料用 event 的方式輸出到 Smart/Container component.



6.2. One-way dataflow overview

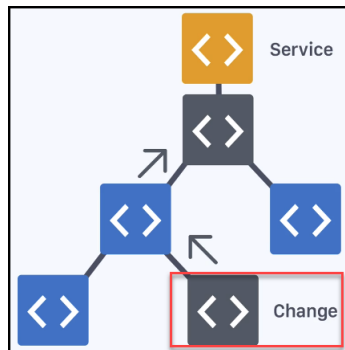
在設計上要使用 One way dataflow.

假設整個 DOM tree 如下, 資料是由 smart component 向下到 child components.

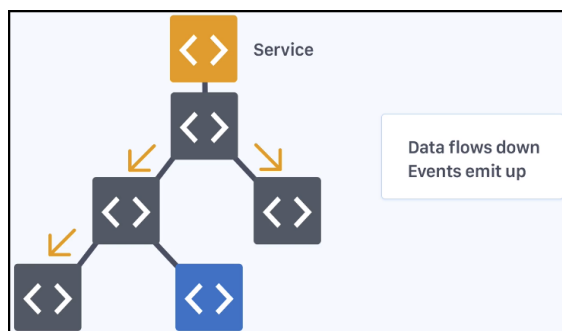


而當某一個 child component 有做資料異動, 舉例來說, 刪除了一筆資料, 它

會用 event 通知並傳送該筆被刪除的資料給 parent. 若 parent 不是 smart component, 則它會再送給它的 parent component, 循環下去直到 smart component.



而 smart component 收到 event 後, 透過 service 將該筆資料刪除, 並再次將資料更新後, child component 會再度 re-render.



6.3. Feature modules with @NgModule

接下來要開始實作一個 customer module. (客製化 module). 主要利用

@NgModule 建立一個獨立功能的模組, 再掛入 application 中.

1. 利用 CLI 建立 passenger-dashboard module.
2. 將此 module 加入 app.module

```
imports: [
  BrowserModule,
  AppRoutingModule,
  // Customer Modules
  PassengerDashboardModule
],
```

6.4. Creating a container (smart) component

Todd 在這部份的規劃是:

1. 一個 module 可能有一個以上的 smart/container component.
2. 所以需要建立一個 containers 的 folder

以 CLI 怎麼建立呢?

1. 建立 smart component: passenger-dashboard

```
ng g c passenger-dashboard/containers/passenger-dashboard
```

```

└─ app
  └─ passenger-dashboard
    └─ containers
      └─ passenger-dashboard
         ├── passenger-dashboard.component.html
         ├── passenger-dashboard.component.scss
         ├── TS passenger-dashboard.component.ts
         └── TS passenger-dashboard.module.ts
```

2. CLI 已自動將 passenger-dashboard component 加入 passenger-dashboard module 中, 但要將此 module 的 component 給其它 module 使用, 必需在 module 中的 exports 加入要開放的 components

```
@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [PassengerDashboardComponent],
  exports: [PassengerDashboardComponent]
})
```

3. 將原本在 `app.component.ts` 中的 `passenger array` 搬到 `passenger-dashboard.component.ts` 中.
4. 在 `app` 下建立 `models` 子目錄, 並在 `models` 目錄下建立 `passenger.interface.ts`, 然後將 `Passenger` 的 `interface` 搬到此檔中.
5. 將原本的 `.status` 的 `css` 搬入 `passenger-dashboard.component.scss` 中
6. 在 `app.component.html` 中加入 `app-passenger-dashboard` 的 `tag` 就可以正常運作了.

所有的資料, `template(HTML)`, `css` 全都搬入 `passenger-dashboard.component` 中, 完全是一個獨立的 `module` 及 `component` 可以給任何 `module` 來使用.

在本節學到

1. 建立 `component`
2. 將 `component` 加入 `module`
3. 將 `module` 中的 `component` 作 `export`, 可以給其它 `module` 使用
4. 建立 `component` 自己的 `scss`
5. 建立共用的 `models(interface)`

6.5. ngOnInit lifecycle hook

[官方說明](#)

在使用 `ngOninit hook` 時, 通常會與 `constructure` 搞混. 到底 2 者的差別在

哪? [Difference between Constructor and ngOnInit](#) 作了很好的解釋.

在 Angular framework 中, Constructor 通常只用來作 DI(相依性注入), 而

ngOnInit 用來做為 binding property 的初始化.

而 ngOnInit 會在 constructor 執行後才會被呼叫.

Angular 的 [life cycle hook](#) 有詳細的解釋, 可以再去細究其用法.

接下來要做的就是將 passengers[] 的初始化移入 ngOnInit() 中

6.6. Presentational (dumb) components

接下來要將 smart component 中的行為及畫面移出, 減輕它的負擔. 所以要建

立 2 個 presentation(dumb) components. 為了要區分 presentation

component 與 container component, 這 2 個 component 要建立在

components 子目錄下.

1. Passenger-count

2. Passenger-detail

```
ng g c passenger-dashboard/components/product-count -t -s
```

```
ng g c passenger-dashboard/components/product-detail -t -s
```

然後在 passenger-dashboard.component.html 加入這 2 個 component 的

selector, 就可看到它們的 template 內容.

6.7. Passing data into components with

@Input

Smart component 要怎麼將資料傳入 child component?

1. 在 smart component 的 html, 使用 property binding 將資料傳入
2. 在 child component, 使用 @Input decorator 宣告該 property.

```
<app-product-count  
  [items]="passengers">  
</app-product-count>
```

```
export class ProductCountComponent implements OnInit {  
  @Input() items: Passenger[];
```

在 passenger-count 中, 我們希望同時顯示已 check-in 的旅客.

```
checkedInCount(): number {  
  if (this.items) {  
    return this.items.filter((p) => p.checkedIn).length;  
  }  
}
```

```
<h4 class="text-green">  
  Total Checked-In: {{ checkedInCount() }} / {{ items?.length }}  
</h4>
```

6.8. Dynamic @Input values with *ngFor

本節要開發的是 passenger-detail.component.

1. 此 component 只顯示一個客戶的內容
2. 呼叫者使用 Dynamic *ngFor 來建立多個 passenger-detail
3. Passenger-detail 的樣式

呼叫端

```
<app-passenger-detail
  *ngFor='let passenger of passengers'
  [detail]="passenger"></app-passenger-detail>
```

Child component

```
export class PassengerDetailComp {
  @Input() detail: Passenger;
```

6.9. Emitting changes with @Output and EventEmitter

本篇要學的是怎麼從 child component 發 event 給 parent.

完成範例會使用到:

1. Input ref
2. Input event handler
3. @Input
4. [ngIf](#): 若要搭配 else 功能, 要使用 ng-template
5. @Output, EventEmitter

範例內容:

1. 新增 Edit button, trigger edit mode 可以修改 fullname
2. Edit button 的 text, 在 editing 時顯示 done, 否則顯示 Edit
3. 新增移除功能, 可以移除 passenger. 這要用到 Event emit 功能. 這與

knockout 使用 callback 來讓 child component 來通知 parent 處理資料變更的方式不同, angular 是使用 event.

4. 要在 parent 去設定 child component 的地方, 加上 remove event 的處理

```
<app-passenger-detail
  *ngFor='let passenger of passengers'
  [detail]="passenger"
  (remove)="handleRemove($event)">
</app-passenger-detail>
```

5. 在 child component 中, 要使用 @Output decorator 建立一個 output 變數, 並設為 EventEmitter 型別, 並 new 一個 EventEmitter 給 remove 變數

```
@Output() remove: EventEmitter<Passenger>;

constructor() {
  this.remove = new EventEmitter();
}
```

6. 設定 remove button 的 click event 的 callback: onRemove()去執行 remove.emit(this.detail). 這裡的 this.detail 就是 passenger 型別.
7. 在 parent 的 handleRemove 函數中, 使用 console.log 印出被點擊移除的 passenger 資料
8. 以同樣方式設計 child component 修改姓名的功能. 新增 edit 的 event emitter, 在 editing 被設為 done 後, 發出 edit 的 event 並帶 passenger 資料給 parent
9. 要在 parent 去設定 child component 的地方, 加上 edit event 的處理

```

<app-passenger-detail
  *ngFor='let passenger of passengers'
  [detail]="passenger"
  (edit)="handleEdit($event)"
  (remove)="handleRemove($event)">
</app-passenger-detail>

```

10. 在 parent 的 handleEdit 函數中, 使用 console.log 印出被編輯的 passenger 資料

6.10.Immutable state changes

接續上一節, 用 Immutable 方式重設資料.

1. 處理 handleRemove()

```

handleRemove(event: Passenger): void {
  this.passengers = this.passengers.filter((p => p.id !== event.id));
}

```

2. 處理 handleEdit()

```

handleEdit(event: Passenger): void {
  this.passengers = this.passengers.map((p) => {
    // 建立immutable object
    if (p.id === event.id) {
      p = { ...p, ...event };
    }
    return p;
  });
}

```

3. 確保修改名有成功, 用 log 印出來看做驗證,

```

▼ (5) [{...}, {...}, {...}, {...}, {...}] ⓘ
  ▼ 0:
    checkInDate: 1490742000000
    checkedIn: true
    children: null
    fullname: "TonyChen"
    id: 1

```

6.11.ngOnChanges lifecycle hook

從目前的設計來看, 會感覺由 parent 的資料傳到 child component 是 one way dataflow. 但實際上因為 JavaScript 的特性, 在 child component 修改姓名時, 實際上是改到同一個物件. 也就是說修改過程中, parent 的 passengers[] 的內容也被修改了. 這可以一個簡單的實驗來驗證. 在 parent 加上一個 ngFor 顯示 passenger 的 fullname, 然後在 passenger-detail component 中去修改姓名, 會看到 parent 的 fullname 會同步更新.

```
<div *ngFor="let passenger of passengers">
  {{passenger.fullname}}
</div>

<app-passenger-detail
  *ngFor='let passenger of passengers'
  [detail]="passenger"
  (edit)="handleEdit($event)"
  (remove)="handleRemove($event)">
</app-passenger-detail>
```

所以要做到真正的 one way dataflow, 必需使用 [ngOnChange life cycle hook](#).

它比 ngOnInit 更早被呼叫, 也就是在 model 與 template 做 binding 前執行. 所以可以利用它將 @Input 的資料做 immutable 的方式, 就可以解決這個問題.

原型如下:

```
export interface OnChanges {
  ngOnChanges(changes: SimpleChanges): void;
}
```

SimpleChanges 原型如下, 屬於弱型別的方式.

```
export interface SimpleChanges {
  [propName: string]: SimpleChange;
}
```

SimpleChange 型別如下

```
export declare class SimpleChange {
  previousValue: any;
  currentValue: any;
  firstChange: boolean;
  constructor(previousValue: any, currentValue: any, firstChange: boolean);
  /**
   * Check whether the new value is the first value assigned.
   */
  isFirstChange(): boolean;
}
```

若要存取@Input() detail 的 SimpleChange 參數, 要用 Object["name"]方式

```
ngOnChanges(changes: SimpleChanges): void {
  console.log(changes['detail'].currentValue);
}
```

打錯字就取不到值了.

然後開始在 ngOnChange 用 Immutable 方式重新建立 this.detail

```
ngOnChanges(changes: SimpleChanges): void {
  this.detail = {...changes['detail'].currentValue};
}
```

這樣就解決這個問題了. 雖然很麻煩...☹

7.Services, Http and Observables

7.1. Data Services and Dependency

Injection

要建立一個 passenger-dashboard.service 來處理 passenger 的資料.

用 cli 建立這個 Service

```
ng g s passenger-dashboard/passenger-dashboard
```

需要將這個 service 加入 passenger-dashboard.module 中的 providers.

```
providers: [PassengerDashboardService]
```

範例: 第一代的做法, 提供一個 getPassengers 的方法, 返回一個 array.

1. 要在 passenger-dashboard.component 中注入 service 到 component 中.

```
constructor(private passengerService: PassengerDashboardService) { }
```

好奇怪! 是誰去 new 一個 service 然後注入這個 component?

Core.js

```
function createClass(view, elDef, allowPrivateServices, ctor) {
  var len = deps.length; len = 1
  switch (len) {
    case 0:
      return new ctor(); ctor = f PassengerDashboard
    case 1:
      return new ctor(resolveDep(view, elDef, allowPr
```

注入這個物件

```
▼ 0:
  flags: 0
  token: f PassengerDashboardService()
  tokenKey: "PassengerDashboardService_61"
  __proto__: Object
length: 1
```

2. 在 ngOnInit() 中, 使用 Service 取得 passenger 的資料.

```
ngOnInit() {
  this.passengers = this.passengerService.getPassengers();
}
```

7.2. Understanding @Injectable

本節要使用 angular 的 Http Module 來進化資料的取得.

[Http Module](#)

[@Injectable](#)


1. 在 passenger-dashboard.module 中加入 HttpClientModule

```
import { HttpClientModule } from '@angular/http';
```

```
@NgModule({  
  imports: [  
    CommonModule,  
    HttpClientModule  
  ],  
})
```

2. 在 passengerDashboard.service 注入 Http(由@angular/http). 但這裡我們會看到它使用了@Injectable 的 decorator, 告訴 angular 這個 class 是可被注入的. 像是一般的 component 使用@Component decorator 一樣. 若沒有加入@Injectable(), 則無法注入及使用 Http. 反過來說, 若沒有要注入任何物到 constructor, 則不需要@Injectable()

```
@Injectable({  
  providedIn: 'root'  
})  
export class PassengerDashboardService {  
  
  constructor(private http: Http) { }  
}
```



7.3. Http data fetching with Observables

依教學去實作時(ng=v6, rxjs=v6), http.get().map()時, map 找不到. 找了一下

文章, 發現要改用 [pipeable-operators](#) (自 rxjs V5.5 後), 才可以使用 map. 用

法大致上是這樣(官網範例):

```
import { range } from 'rxjs/observable/range';
import { map, filter, scan } from 'rxjs/operators';

const source$ = range(0, 10);

source$.pipe(
  filter(x => x % 2 === 0),
  map(x => x + x),
  scan((acc, x) => acc + x, 0)
)
.subscribe(x => console.log(x))
```

用 rxjs(v6), 將 http.get 的 response 轉成 Observable<Passenger[]>.

Import map 的方式

```
import { map, filter, switchMap } from 'rxjs/operators';
```

實作

```
getPassengers(): Observable<Passenger[]> {
  return this.http
    .get(PASSENGER_API)
    .pipe(
      map<Response, Passenger[]>((response: Response) => response.json())
    );
}
```

Webapi server 採用 json-server.

1. 安裝 json-server

```
npm i --save-dev json-server
```

2. 建立 db.ts, routes.js

3. 在 package.json 建立 npm script: api

```
"scripts": {
  "api": "json-server db.js --routes routes.json"
```

4. 先產生 db.js

tsc db

5. 開啟另一個 terminal, 執行 npm run api
6. 程式中使用 localhost:3000 當成 webapi 主機
7. 程式使用 localhost:3000 當成 host

7.3.1. Update Http to HttpClient

自 Angular V4.3.0 後, HttpClientModule 的 Http 就已變成 deprecated, 並由新一代 module HttpClient 取代.

1. 在 passenger-dashboard.module 中改為引入 [HttpClientModule](#). 來源與 module 都不同了.

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

@NgModule({
  imports: [
    CommonModule,
    HttpClientModule
  ],
```

2. 在 passenger-dashboard.service 中注入 HttpClient
3. 使用 httpClient.get 並使用回傳資料的泛型型別(此為 Passenger[]), 即可以傳 Observable<Passenger[]>, 不需再做 pipe(map())

```
getPassengers(): Observable<Passenger[]> {
  return this.httpClient
    .get<Passenger[]>(PASSENGER_API);
}

(method) HttpClient.get<Passenger[]>(url: string, options?: {
  headers?: HttpHeaders | {
    [header: string]: string | string[];
  };
  observe?: "body";
  params?: HttpParams | {
    [param: string]: string | string[];
  };
  reportProgress?: boolean;
  responseType?: "json";
  withCredentials?: boolean;
}): Observable<Passenger[]> (+14 overloads)
```

7.4. Http put, delete with immutable state

7.4.1. 使用 Http put 更新資料

本節要做的事

1. 在 service 寫一個 updatePassenger 透過 httpClient.put 去更新 server 的資料, 更新完後回傳該筆更新後的資料.
2. 在 passenger-dashboar container 處理 handleEdit(), 透過 service 寫入變更的 fullname, 並更新 component 中 local 的 passengers 資料.

使用 put, json server 好像都會回傳更新後的資料, 很好用.

7.4.2. 使用 Http delete 刪除資料

本節要做的事

3. 在 service 寫一個 removePassenger 透過 httpClient.delete 去更新 server 的資料, 移除後, 並不會回傳該筆的資料.
4. 在 passenger-dashboar container 處理 handleRemove(), 透過 service 移除該筆資料, 並更新 component 中 local 的 passengers 資料.

使用 put, json server 好像都會回傳更新後的資料, 很好用.

補充: 如果參數使用 query parameter 型式, 而不是 rest 型式

```
public get(id: string): Observable<Hero> {  
    return this.httpClient.get<Hero>(this.URL, {  
        params: new HttpParams().set("id", id)  
    });  
}
```

7.5. Custom Headers and RequestOptions

某些資料伺服器在做 POST/PUT/PATCH 時, 必需指定 Content-Type:

application-json 的 header, 所以在設計 service 時, 若有需要用到

POST/PUT/PATCH 時, 最好指定 header.

來看一下 put 的原型:

```
put<T>(url: string, body: any | null, options?: {  
  headers?: HttpHeaders | {  
    [header: string]: string | string[];  
  });
```

所以要傳入一個 object, 含有 headers: HttpHeaders 物件的資料

```
const httpOptions = {  
  headers: new HttpHeaders({  
    'Content-Type': 'application/json'  
  })  
};
```

做 put 時傳入

```
updatePassenger(passenger: Passenger): Observable<Passenger> {  
  return this.httpClient  
    .put<Passenger>(`${PASSENGER_API}/${passenger.id}`, passenger, httpOptions);  
}
```

7.6. Http Promises alternative

如果不想用 observable 而要用 promise, 只要單純使用 [ES2015 promise](#) 即

可.

```
getPassengers(): Promise<Passenger[]> {  
  return this.httpClient  
    .get<Passenger[]>(PASSENGER_API)  
    .toPromise();  
}
```

```
ngOnInit() {
  this.passengerService
    .getPassengers()
    .then((data) => {
      this.passengers = data;
    });
}
```

7.7. Observable.catch error handling

Rxjs > 5.5+

將錯誤丟給呼叫者去處理

```
getPassengers(): Observable<Passenger[]> {
  return this.httpClient
    .get<Passenger[]>(PASSENGER_API)
    .pipe(
      catchError((err) => {
        console.log(err);
        return Observable.throw(err.json());
      })
    );
}
```

呼叫者

```
ngOnInit() {
  this.passengerService
    .getPassengers()
    .subscribe((data) => {
      this.passengers = data;
    }, (error) => {
      console.log(error);
    });
}
```

另一種方式, 使用 `caught`(就是取得資料或是 `retry` 取得的資料)

```
return this.httpClient
  .get<Passenger[]>(PASSENGER_API)
  .pipe(
    catchError((err, caught) => {
      console.log(err);
      return caught;
    })
  );
```

7.8. HttpClient Retry

可以加上 retry 及 error 處理

```
getPassengers(): Observable<Passenger[]> {  
  return this.httpClient  
    .get<Passenger[]>(PASSENGER_API)  
    .pipe(  
      retry(3),  
      catchError((err, caught) => {  
        console.log(err);  
        return caught;  
      })  
    );  
}
```

8. Template-driven Forms, Inputs and Validation

8.1. Forms container component

在 containers 目錄下, 建立一個 passenger-viewer component, 並可以 export 給 app.module 使用.

功能

1. 在 service 建立 getPassenger, 傳入 id 可以取回一筆 passenger
2. ngOnInit: 由 service 取得 passenger, 並存入 passenger property
3. template: 以 json 格式顯示 passenger 資料
4. 將 app.component.html 中的 passenger-dashboard 換成顯示 passenger-viewer.

8.2. Form stateless component

8.2.1. 修改 Passenger Interface and Json Server Data

接下來的範例要修改 passenger 的 model.

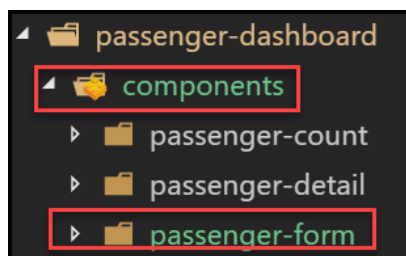
1. 加入 baggage: string, db.ts 中的資料全設成空字串
2. children 改成 optional

在 db.ts 及 models/passenger.ts 中都要改.

改完後要重建 db.ts: tsc db, 再重啟 json-server

8.2.2. Create Form Component

在 components 下建立 passenger-form component.



8.2.3. Passing Data To Form Component

然後在 passenger-viewer component 的 html 中使用 passenger-form, 並傳

入 passenger 給 form 的 detail 當 @Input()

8.2.4. Import FormModule

在 passenger-dashboard.module 中引入 FormsModule

```
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [
    CommonModule,
    HttpClientModule,
    FormsModule
  ],
```

引入 FormsModule 後, 就可以使用 ngModel 了.

8.3. ngForm and ngModel

[ngForm 官方文件](#)

[ngModel 官方文件](#)

ngForm 語法, 會產生一個 #detailForm ref 可以在 DOM 中使用

```
<form #detailForm="ngForm" novalidate>
</form>
```

開始使用 ngModel 搭配 ngForm 建立表單

1. 先建立一個顯示 detailForm ref 的 json 來做資料比對, 同時也建立一個

detail: Passenger 資料的顯示在最上方做比對

2. 使用 ngModel 建立 one-way data binding 的 detail.fullname 的 input.

甲、因為是 form, 所以 name 也要與資料名稱相同

乙、要做 safe navigation

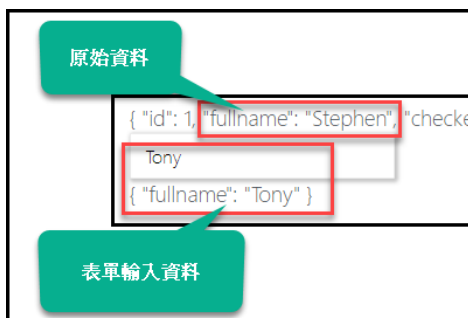
3. 看輸出


```

<form #detailForm="ngForm" novalidate>
  <div>
    <input type="text"
      name="fullname"
      [ngModel]="detail?.fullname"
    >
  </div>
  {{ detailForm.value | json }}
</form>

```

輸出: 因為是 one-way binding, 所以輸入的資料會暫存在#detailForm 中



4. 使用 ngModel 建立 one-way data binding 的 detail.id 的 input.

8.4. Binding to radio buttons

8.4.1. 加入 radio buttons

如何將 Radio button binding 到 form?

範例:

為何要用[value]="true"的 property binding? 因為要得到選擇的值是

true/false. 若是用 value="true", 則會得到字串 yes/no.

```

<div>
  <label for="checkedIn">
    <input
      type="radio"
      name="checkedIn"
      [value]="true"
      [ngModel]="detail?.checkedIn">
    Yes
  </label>
  <label for="checkedIn">
    <input
      type="radio"
      name="checkedIn"
      [value]="false"
      [ngModel]="detail?.checkedIn">
  </label>
</div>

```

8.4.2. 加入 radio buttons Event 去更動 TimeStamp

我們要在選擇 radio button 後, 希望去異動 checkInDate 到目前的時間. 利用 (ngModelChange)來處理.

在 yes, no 的 radio button 都加上 ngModelChange

```

[ngModelChange]="toggleCheckIn($event)">
Yes

```

Component

```

toggleCheckIn(checkIn: boolean): void {
  if (checkIn) {
    this.detail.checkInDate = Date.now();
  }
}

```

但 checkInDate 並沒有顯示在畫面上, 因為它並不在 form 裡. 要建立一個 checkInDate 的 input(number), 但是它要在 checkedIn=true 時才要顯示, 所以利用 ngIf 來判斷是否要有這個 input field.

這裡有用到 form reference 的技巧, 使用 ngForm 建立的 ref 來取 checkedIn 的值作 ngIf 的判斷.

```
<div *ngIf="detailForm.value.checkedIn">
  Check In Date:
  <input type="number"
    name="checkInDate"
    [ngModel]="detail?.checkInDate">
</div>
```

選 yes

☒ Yes ☐ No

Check In Date:

{ "fullname": "Stephen", "id": 1, "checkedIn": true, "checkInDate": 1490742000000 }

選 no

☐ Yes ☒ No

{ "fullname": "Stephen", "id": 1, "checkedIn": false }



為什麼選 no 時, checkInDate 不見了?

因為使用 ngIf, 會將 DOM 整個移除, 所以就不在 detailForm 裡, 當然就不見了.

8.5. Binding to checkboxes

將上節使用 radio button 的方式改為使用 checkbox, 並學習使用 checkbox binding.

```
<label for="checkedIn">
  <input
    type="checkbox"
    name="checkedIn"
    [ngModel]="detail?.checkedIn"
    (ngModelChange)="toggleCheckIn($event)">
</label>
```

8.6. <select> option rendering, and ngValue



[官方文件](#)

首先要建立可選的 options 的 array.

1. 建立 interface Baggage
2. 建立 baggage[]
3. Html 的<select> bind 到 detail.baggage
4. Html 的<option> bind 到 baggage[]

```
<div>
  Luggage:
    <select
      name="baggage"
      [ngModel]="detail?.baggage">
      <option *ngFor="let item of baggage"
        [value]="item.key">
        {{item.value}}
      </option>
    </select>
</div>
```

一旦選了其中一項, option 的 value 就會被 bind 到 detail?.baggage.

```
Luggage: Hand baggage ▼
{ "fullName": "Stephen", "id": 1, "checkedIn": true, "baggage": "hand-only"
}
```



如果要設定 select/option 的 default 值要怎麼做?

AngularV6 版有 2 種作法

1. 如果 option 的值是 string, 用[value]即可
2. 如果 option 的值是 object, 用[ngValue]

8.7. Form validation and error states



[官方文件](#)



所有的驗證都是用 `ngModel` 在 `html` 中完成



[如何加入 `css` 來強調錯誤訊息](#)



[ngForm](#) 及 [ngModel](#) 原型中, 定義了整個 `form`/單一輸入欄位的驗證

flags.

```
get value: any
get valid: boolean | null
get invalid: boolean | null
get pending: boolean | null
get disabled: boolean | null
get enabled: boolean | null
get errors: ValidationErrors | null
get pristine: boolean | null
get dirty: boolean | null
get touched: boolean | null
get status: string | null
get untouched: boolean | null
get statusChanges: Observable<any> | null
get valueChanges: Observable<any> | null
get path: string[] | null
reset(value: any = undefined): void
hasError(errorCode: string, path?: string[]): boolean
getError(errorCode: string, path?: string[]): any
```

我們要利用這些 flag 來 watch 整個驗證狀態.

8.7.1. Input Field error states

將 `fullname` 設為 `required`, 並且設定 `@fullname` ref 指向自己的 `ngModel`

```

<div>
  <input type="text"
    name="fullname"
    required
    #fullname="ngModel"
    [ngModel]="detail?.fullname"
  >
  {{fullname.errors | json}}
</div>

```

其結果如下

正常時

A screenshot of a form input field containing the text "Stephen". To the right of the input field, there is a red arrow pointing to the word "null".

空值時

A screenshot of an empty form input field. To the right of the input field, there is a red arrow pointing to the JSON object {"required": true}.

另外對整個 form 也加入 valid, invalid 的 watch

```

<div>Form Valid: {{ detailForm.valid | json }}</div>
<div>Form Invalid: {{ detailForm.invalid | json }}</div>

```

全部欄位正常

A screenshot of a form with all fields filled: "Stephen", "1", a checked checkbox, "1490742000000", and "Hand baggage". Below the form, the text "Form Valid: true" is highlighted with a red box, and "Form Invalid: false" is shown below it.

只要有一個欄位異常

A screenshot of the same form, but the first input field is empty. Below the form, the text "Form Valid: false" is highlighted with a red box, and "Form Invalid: true" is shown below it.

8.7.2. 顯示錯誤訊息

參考 [Template-driven validation](#) 及 [Control status CSS classes](#) 的示範, 使用

invalid 旗標搭配 ngIf 來顯示錯誤訊息.

1. 在 html 中加入錯誤訊息的顯示

```
<div *ngIf="fullname.invalid">
  Fullname is required
</div>
```

2. 在 passenger-form.scss 中加入 angular 在錯誤時會自動加入的 class 的

CSS

```
.ng-invalid {
  color: #a94442; /* red */
}

.ng-invalid:not(form) {
  border-left: 5px solid #a94442; /* red */
}

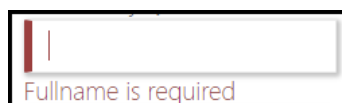
.ng-valid[required], .ng-valid.required {
  border-left: 5px solid #42A948; /* green */
}
```

正常



驗證失敗

```
<input _ngcontent-c2 name="fullname" required type="text"
ng-reflect-required ng-reflect-name="fullname" class="ng-
dirty ng-invalid ng-touched" ng-reflect-model="Stephen">
```



8.7.3. 使用 dirty 及 touched 優化

Dirty: 有在欄位輸入過任一字元, 會設為 true

Touched: 點過欄位, 再 blur, 會設為 true

我們希望有點入過欄位且有輸入過再做驗證

```
<div *ngIf="fullname.invalid && (fullname.dirty || fullname.touched)">
  Fullname is required
</div>
```

但 Todd 說它 prefer 用 dirty 就好.

8.8. Dynamically disabling submit



目標: 如何利用 form 驗證結果, 來 disable submit button

1. 用 property binding 控制 button disabled

```
<button type="submit" [disabled]="detailForm.invalid">
  Update Passenger
</button>
```

2. 修改 scss, 讓 disabled 時的樣式可以有禁用符號

```
button:disabled,
button[disabled]{
  cursor: not-allowed;
  border: 1px solid #999999;
  background-color: #cccccc;
  color: #666666;
}
```

8.9. ngSubmit and stateless @Output



目標: 使用 form submit event 將 detailForm 的資料送給呼叫者

1. 在 html form 中建立 submit event 呼叫 handleSubmit, 並傳入

detailForm ref 及 valid flag

```
<form
  (submit)="handleSubmit(detailForm.value, detailForm.valid)"
  #detailForm="ngForm" novalidate>
```

2. 在 component 中定義 update 的 event emitter, 並將 form 的資料送出


```

handleSubmit(passenger: Passenger, valid: boolean) {
  if (valid) {
    this.update.emit(passenger);
  }
}

```

3. 在使用端(passenger-viewer), 處理 update event, 並呼叫 onUpdate 傳入 passenger 的資料.

```

<div>
  <app-passenger-form
    [detail]="passenger"
    (update)="onUpdate($event)">
  </app-passenger-form>
</div>

```

4. 透過 service 將資料送至 server, 並將 form 傳入的 passenger 更新至 passenger-viewer 的 passenger(使用 immutable).

```

onUpdate(event: Passenger) {
  this.passengerService
    .updatePassenger(event)
    .subscribe((data) => {
      this.passenger = {...this.passenger, ...event};
    });
}

```

9. Component Routing

9.1. Base href and RouterModule

在 index.html 中, 一定要有 baseUrl="/", 不然使用 routing 會有問題

```

<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Ng6SeedProject</title>
  <base href="/">

```

9.2. Root module routes and outlet



[官方文件](#)



在 app.module 引入 RouterModule, 並建立 routing table.

在 RouterModule class 中有 2static method 及一個建構式

```
class RouterModule {  
  static forRoot(routes: Routes, config?: ExtraOptions): ModuleWithProviders  
  static forChild(routes: Routes): ModuleWithProviders  
  constructor(guard: any, router: Router)  
}
```

我們會先用到 [forRoot\(\)](#)

forRoot()需傳入 [Routes](#) 參數, 其實它是一個 [Route\[\]](#)

Routes

```
type Routes = Route[];
```

Route

```
interface Route {  
  path?: string  
  pathMatch?: string  
  matcher?: UrlMatcher  
  component?: Type<any>  
  redirectTo?: string  
  outlet?: string  
  canActivate?: any[]  
  canActivateChild?: any[]  
  canDeactivate?: any[]  
  canLoad?: any[]  
  data?: Data  
  resolve?: ResolveData  
  children?: Routes  
  loadChildren?: LoadChildren  
  runGuardsAndResolvers?: RunGuardsAndResolvers  
}
```

實際操作

1. 引入 RouterModule

```
import { RouterModule, Routes } from '@angular/router';
```

2. 在第一層建立 HomeComponent

```
ng g c home -it -is --flat
```

3. 在 app.module 中引入 HomeComponent(在 declarations[])
4. 建立 forRoot 需要的參數, Routes(Route[])

甲、Path: 這裡定義 root page, 所以就是"/", 所以填空白

乙、pathMatch: full -> 表示 path 要完全 match 才會使用這個 routing.

```
const routes: Routes = [  
  { path: '', component: HomeComponent, pathMatch: 'full' }  
];
```

5. 在 app.component.html 中, 建立<router-outlet>, 它就會將剛才定義的 root 的 component 取代這個 tag.

```
<div class="app">  
  <router-outlet></router-outlet>  
</div>
```

9.3. Wildcard routes for 404 handling



使用 router 的 wildcard(**)處理沒有對應到的 routing, 顯示 not found.

1. 建立 NotFound component
2. 建立 wildcard router

```
const routes: Routes = [  
  { path: '', component: HomeComponent, pathMatch: 'full' },  
  { path: '**', component: NotFoundComponent }  
];
```

3. 測試: 隨便打一個 routing, 會顯示 NotFound component 內容

9.4. Understanding routerLink

 使用 routerLink Directive 在 html 中開始頁面.



[官方文件](#)

```
@Directive({ selector: '[routerLink]' })
class RouterLink {
  constructor(router: Router, route: ActivatedRoute, tabIndex: string,
    renderer: Renderer2, el: ElementRef) {
    queryParams: {...}
    fragment: string
    queryParamsHandling: QueryParamsHandling
    preserveFragment: boolean
    skipLocationChange: boolean
    replaceUrl: boolean
    set routerLink: any[] | string
    set preserveQueryParams: boolean
    get urlTree: UrlTree
    onClick(): boolean
  }
}
```

array用來組url

string連route

範例

```
<div>
  <ul>
    <li><a routerLink="/">Home</a></li>
    <li><a routerLink="/oops">404</a></li>
  </ul>
</div>
```

9.5. Styling active routes

 學習使用 RouterLinkActive Directive 設定當 url match routerLink 時, 要

加入哪些 CSS class 來做 Active 頁面的 highlight 效果



[官方文件](#)

```

@Directive({
  selector: '[routerLinkActive]',
  exportAs: 'routerLinkActive'
})
class RouterLinkActive implements OnChanges, OnDestroy, AfterContentInit {
  constructor(router: Router, element: ElementRef, renderer: Renderer2,
    cdr: ChangeDetectorRef) {
    links: QueryList<RouterLink>
    linksWithHrefs: QueryList<RouterLinkWithHref>
    get isActive: boolean
    routerLinkActiveOptions: {...}
    set routerLinkActive: string[] | string
    ngAfterContentInit(): void
    ngOnChanges(changes: SimpleChanges): void
    ngOnDestroy(): void
  }
}

```

1. 設定 url match 時, 加入 active class
2. 使用 routerLinkActiveOptions property({exact: true}), 設定"/"要完全 match 才可加 active class

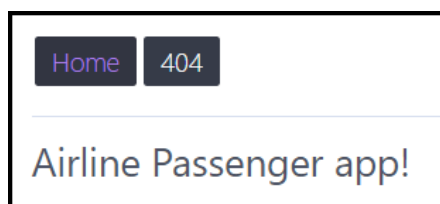
```

<nav class="nav">
  <a routerLink="/"
    routerLinkActive="active"
    [routerLinkActiveOptions]="{exact: true}">Home</a>
  <a routerLink="/oops"
    routerLinkActive="active">404</a>
</nav>
<router-outlet></router-outlet>

```

3. 在 ap.component.scss 加入 css 設定 link 樣式

只要 link match, 就會顯示紫色.



9.6. Dynamic navigation with ngFor



當系統變複雜時, 在 component 中建立 link 的 array 並搭配 ngFor 來建立頁面上的多個 link button

```
nav: Nav[] = [
  {
    link: '/',
    name: 'Home',
    exact: true
  },
  {
    link: '/oops',
    name: '404',
    exact: false
  },
];
```

```
<nav class="nav">
  <a *ngFor="let item of nav" [routerLink]="item?.link"
    routerLinkActive="active"
    [routerLinkActiveOptions]="{exact: item?.exact}">{{item?.name}}</a>
</nav>
```

9.7. Feature-module routes with forChild()



使用 RouterModule 的 forChild() 加入之前設計的 passenger module 功能模組.

1. 在 passenger-dashboard module 中, 設定 forChild() 的 routes.

```
const routes: Routes = [
  { path: 'passengers', component: PassengerDashboardComponent }
];

@NgModule({
  imports: [
    CommonModule,
    HttpClientModule,
    FormsModule,
    RouterModule.forChild(routes)
  ],
```

2. 在 passenger-dashboard module 中, 因為已使用 router, 所以可以將 exports[] 中的 component 移除.

3. 在 app.component 中, 加入 /passenger 的 router 設定

```
{
  link: '/passengers',
  name: 'Passenger',
  exact: true
},
```

9.8. Child and dynamic routes



將 forChild 的 routes 改為使用 children[] 來存取功能模組中所有的頁面.

1. 將上一節中的 path 對應的 component 移除
2. 以 children[] 建立更下一層的 routes
3. :id 表示為動態傳入的參數, example: /1, /2

```
const routes: Routes = [
  { path: 'passengers',
    children: [
      { path: '', component: PassengerDashboardComponent },
      { path: ':id', component: PassengerViewerComponent }
    ]
  }
];
```

4. 測試 passengers/1, passengers/2 會發現頁面資料都是相同. 下一節會解

決這個問題.

9.9. Route params, data-fetching with switchMap



使用 Router, ActivatedRoute



[ActivatedRoute](#)

Route 與被載入的 component 的資訊可由 ActivatedRoute 來取得.

1. 之前設計的 Passenger-Viewer 預設只載入第 1 筆資料

```
ngOnInit() {
  this.passengerService
    .getPassenger(1)
    .subscribe((data) => {
      this.passenger = data;
    });
}
```

2. 在 passenger-viewer 引入 ActivatedRoute

```
export class PassengerViewerComponent implements OnInit {
  passenger: Passenger;
  constructor(
    private route: ActivatedRoute,
    private passengerService: PassengerDashboardService) { }
```

3. 使用 rxjs 的 switchMap, 將 Params(Observable<Params>), 再映射到一

個回傳 Observable, 也就是我們的 service, 然後再用 subscribe 取得

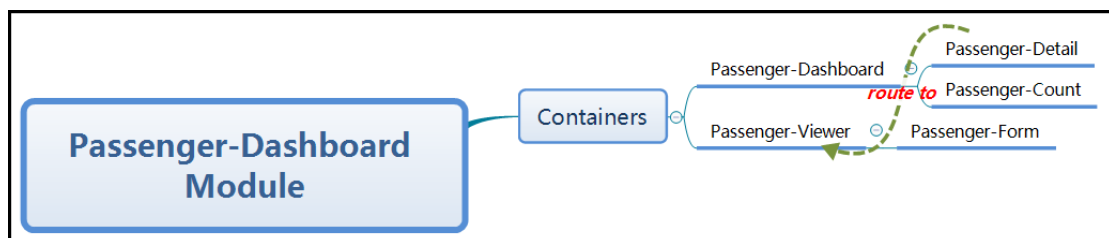
passenger.

```
ngOnInit() {
  this.route.params
    .pipe(
      switchMap((data: {id: number}) =>
        this.passengerService
          .getPassenger(data.id))
    )
    .subscribe((data) => {
      this.passenger = data;
    });
}
```

9.10.Imperative routing API

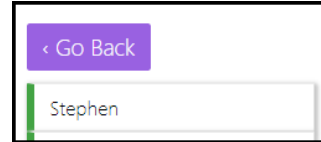
命令式的 routing.

先看一下架構



1. 在 passenger-viewer component 中, 加入 goBack, 使用 router 的 navigate 回到/passengers

```
goBack() {  
  this.router.navigate(['/passengers']);  
}
```



2. 在 passenger-viewer 中, 加入一個 button 可以呼叫 goBack()回到 /passengers
3. 在 Passenger-Detail 中, 加入 View button, 點選後直接換頁顯示 Passenger-Viewer, 不用像課程中用 EventEmitter 又發出 event 叫上一層的 Passenger-Dashboard 去換頁.

```
onView() {  
  // /passenger/1  
  this.routes.navigate(['/passengers', this.detail.id]);  
}
```

9.11.Hash location strategy

- useHash enables the location strategy that uses the URL fragment instead of the history API.

為何要用 hash?

You also may have noticed that we passed along `{ useHash: true }` as the second argument — by default Angular 2 uses HTML5 location based routing, but that would require us to have a server set up that's smart enough to handle it. For the sake of simplicity we told Angular 2 to use hash based routing instead as it's guaranteed to work in virtually any environment setup.

Angular V2+ 預設使用 HTML5 location, 也就是沒有用 hash, 但這樣的做法需要在 server side 做設定才會正常工作. 所以使用 hash 在大部份的環境都可以正常的工作, 除非要使用 Server Side Render 或是進階功能, 才不使用 hash. 設定方式是在 app.module 的 fRouterModule.forRoot 中, 使用第 2 個參數,

設定 useHash: true

```
RouterModule.forRoot(routes, { useHash: true })
```

localhost:4200/#/passengers/1

9.12.Applying redirects

假如我們希望在首頁就轉到 passenger-dashboard, 可以在 app.module 的

routes 中設定 redirectTo

原始的

```
const routes: Routes = [  
  { path: '', component: HomeComponent, pathMatch: 'full'},  
  { path: '**', component: NotFoundComponent }  
];
```

設定 redirectTo

```
const routes: Routes = [  
  { path: '', redirectTo: 'passengers', pathMatch: 'full'},  
  { path: '**', component: NotFoundComponent }  
];
```

輸入 localhost:4200, 會直接跳轉到 localhost:4200/\$/passengers