

Angular 2初探

2016-12-11

預備知識

- Typescript
- ES6
- npm

optional

- webpack

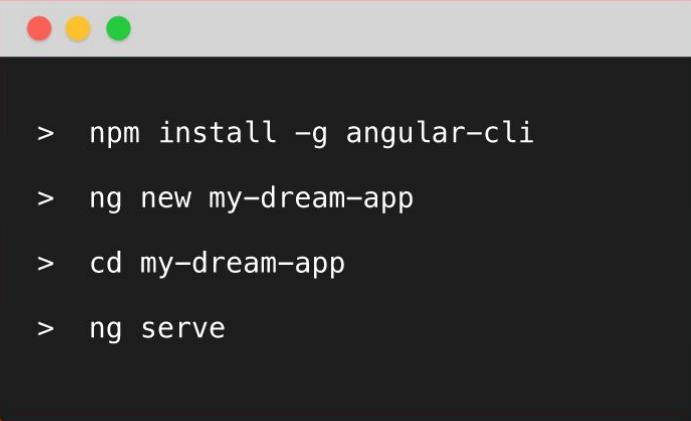
作業系統

- Windows 7 以上版本 (更新到最新 Service Pack 版本)
- Mac OS X 10.6 以上版本

npm 安裝Angular CLI (要裝5~10分鐘)

官網 :<https://cli.angular.io/>

GitHub :<https://github.com/angular/angular-cli>



The image shows a screenshot of a Mac OS X terminal window. The window has a dark background and three colored window control buttons (red, yellow, green) at the top. Inside the terminal, four command-line instructions are listed:

```
> npm install -g angular-cli  
> ng new my-dream-app  
> cd my-dream-app  
> ng serve
```

To the right of the terminal window, there is a large orange rectangular area containing the Angular CLI logo and description.

Angular CLI
A command line interface for Angular

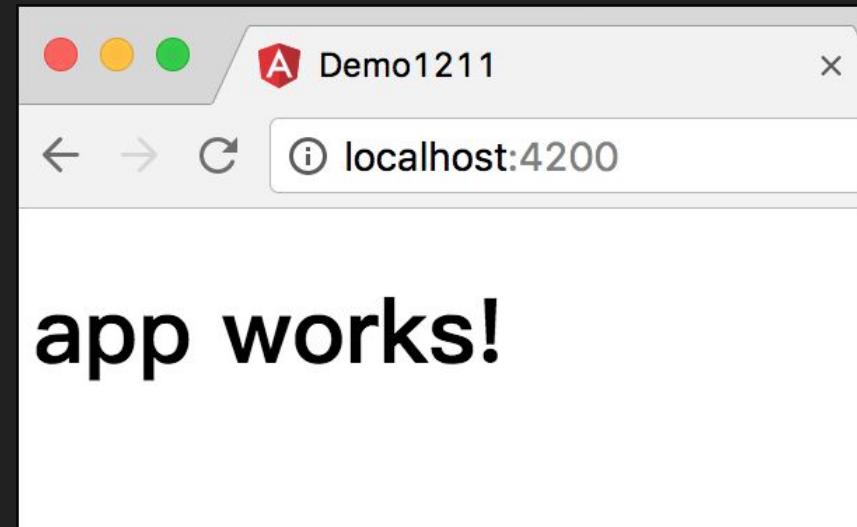
[GET STARTED](#)

建專案用yarn去裝dependencies快很多

- yarn 100%與npm相容 速度快5-20倍
- npm install -g angular-cli
- npm install -g yarn
- ng new demo1211 --skip-npm
- cd demo1211
- yarn
- ng serve

<http://localhost:4200/>

看網頁是否有run起來



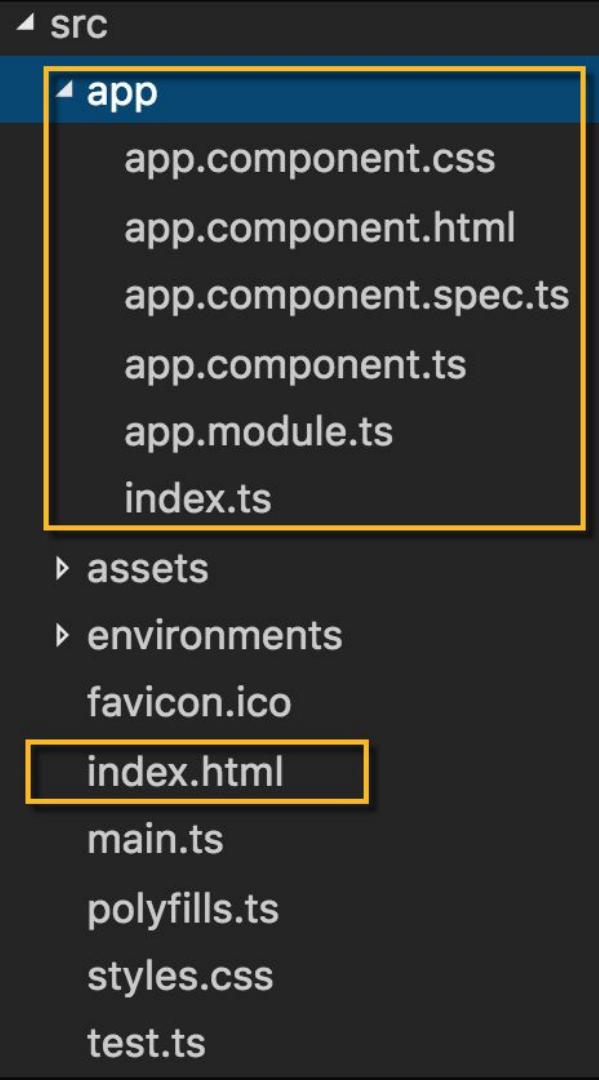
有安裝過angular cli要升級

有error可加上sudo npm uninstall -g angular-cli

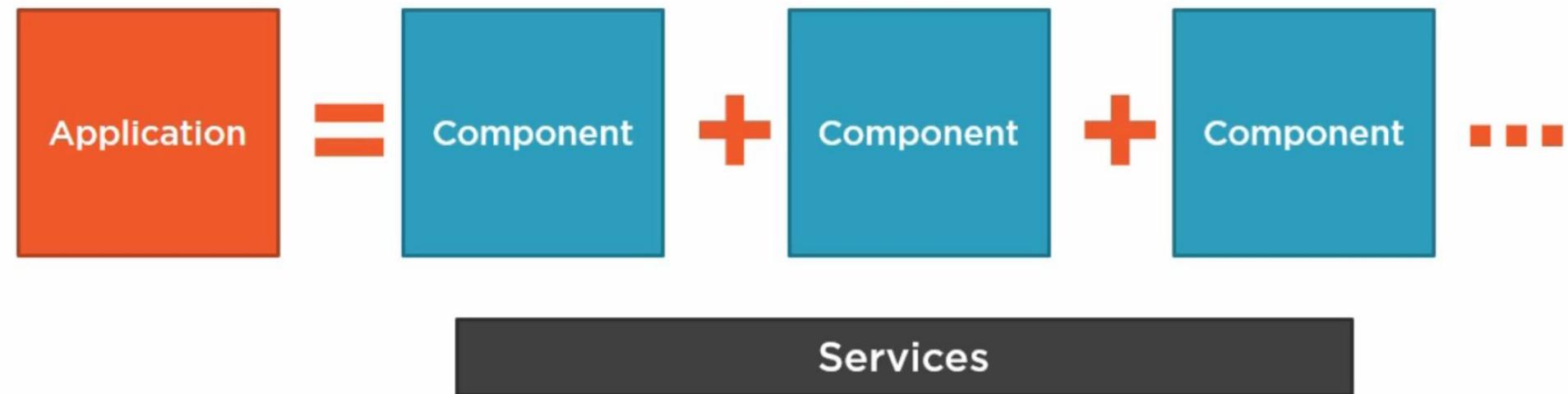
```
npm uninstall -g angular-cli  
npm cache clean  
npm install -g angular-cli
```

專案結構

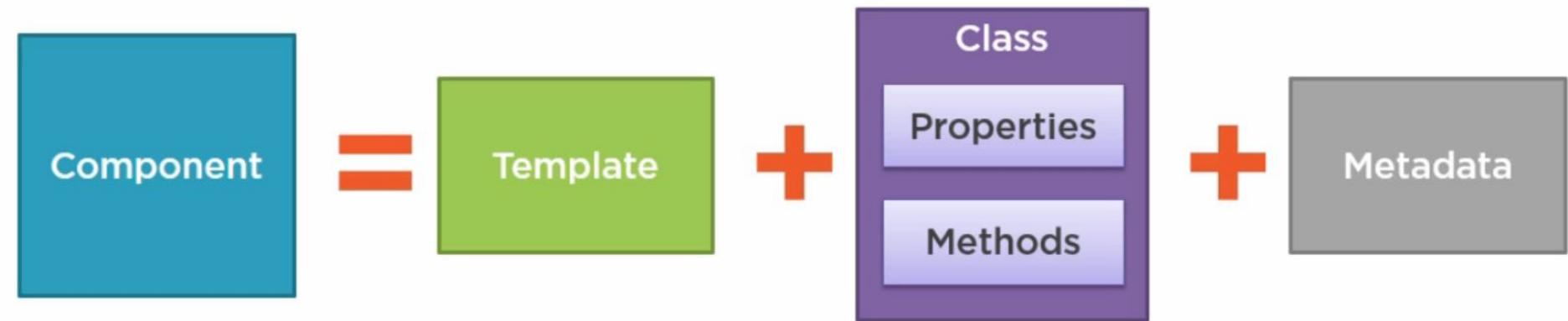
- 主要看app folder



Angular App的結構

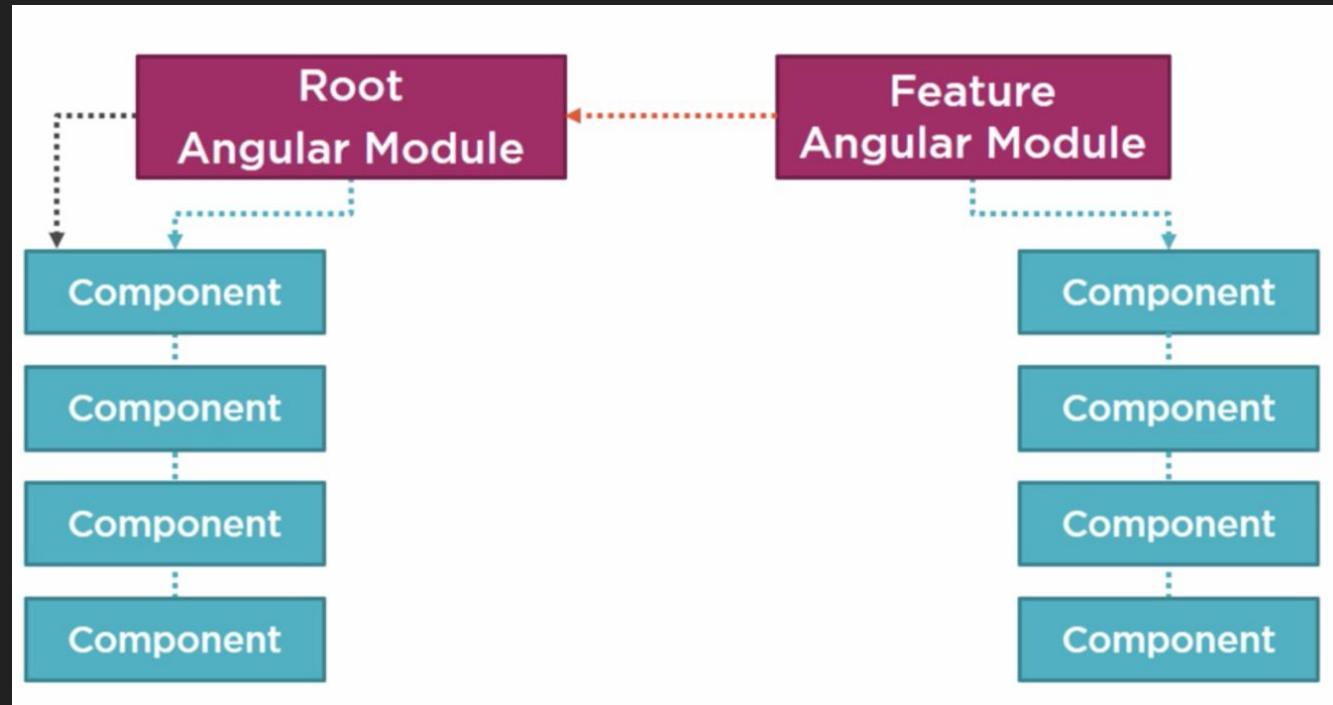


Component



Angular Module

至少會有一個module - Root



Demo

Acme Product Management Home Product List

Product List

Filter by:

Hide Image

Product	Code	Available	Price	5 Star Rating
 Leaf Rake	GDN-0011	March 19, 2016	\$19.95	★★★
 Garden Cart	GDN-0023	March 18, 2016	\$32.99	★★★★
 Hammer	TBX-0048	May 21, 2016	\$8.9	★★★★★
 Saw	TBX-0022	May 15, 2016	\$11.55	★★★★
 Video Game Controller	GMG-0042	October 15, 2015	\$35.95	★★★★★



filter

Product List

Filter by:

Filtered by: am

Show Image	Product	Code	Available	Price	5 Star Rating
	Hammer	TBX-0048	May 21, 2016	\$8.9	★★★★★
	Video Game Controller	GMG-0042	October 15, 2015	\$35.95	★★★★★

product detail

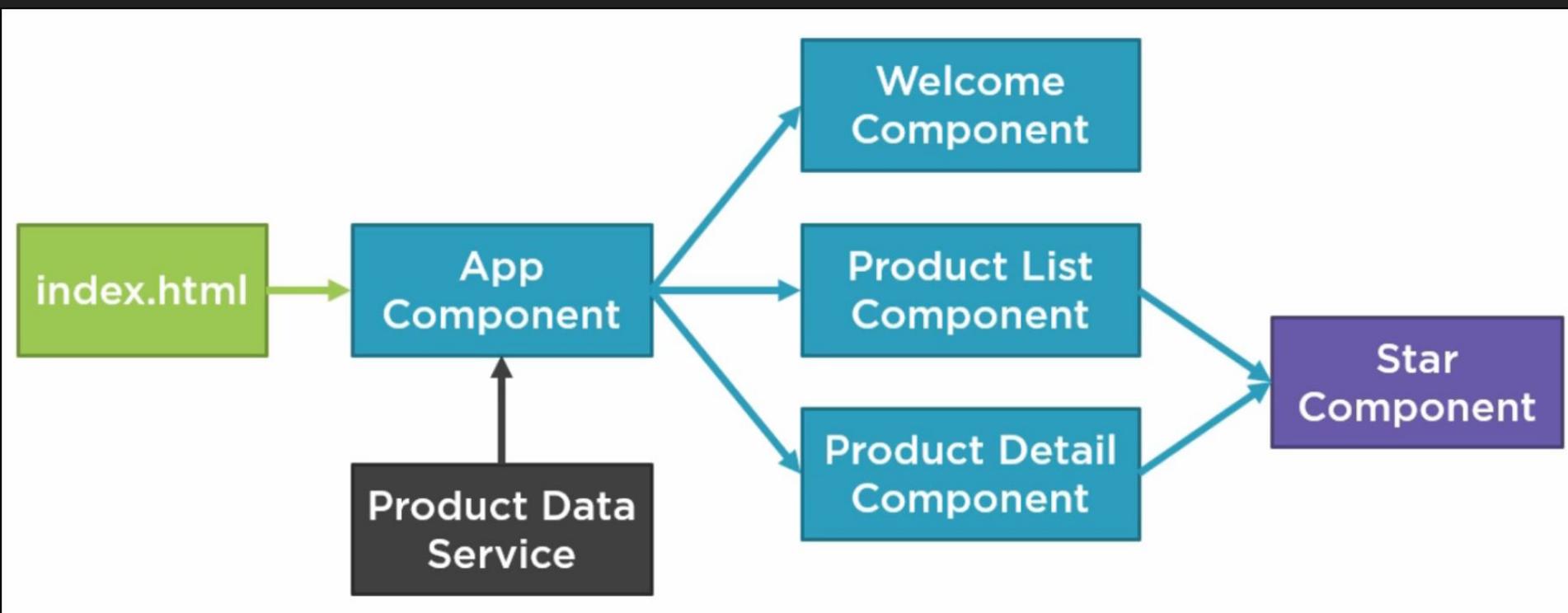
Product Detail: Video Game Controller

Name: Video Game Controller
Code: GMG-0042
Description: Standard two-button video game controller
Availability: October 15, 2015
Price: \$35.95
5 Star Rating: ★★★★☆



[◀ Back](#)

App架構



JS有namespaces跟code organize問題

ES2015把file都當成modules

所以不用另外定義module

只要直接寫code, 然後export/import,

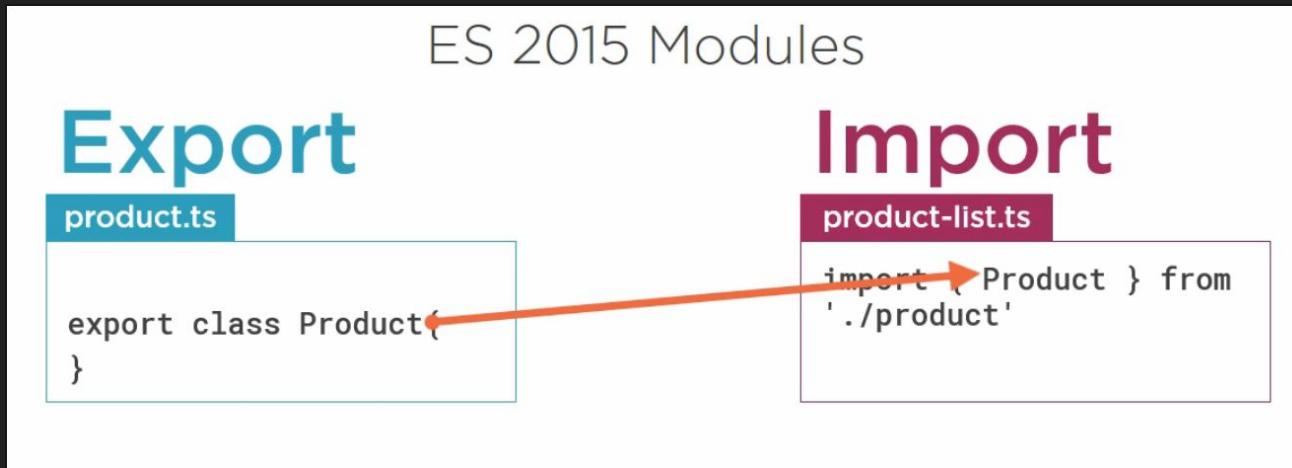
這個file就成了module

Ag2使用此法

所以當我們建立code files, import/export某些東西

就是建立modules

ES 2015 Modules



建立product.ts 並export 一個class Product, 此file就成了module

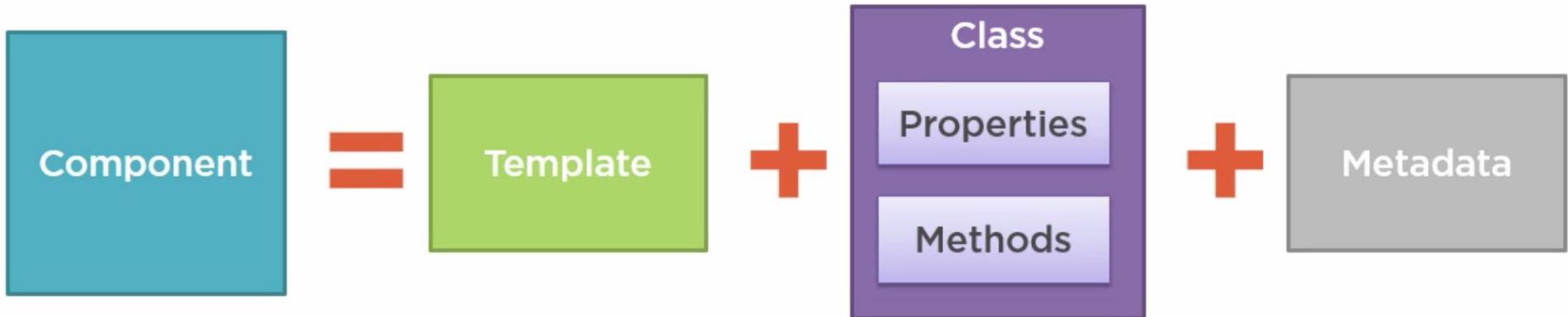
因為class被export後, 其他module就可以import它

在product-list.ts去import此class { Product }, 因為此file有import, 所以也變成了module

Introduction to Components

Angular 2就是一堆component組合起來





- View layout
- Created with HTML
- Includes binding and directives
- Code supporting the view
- Created with TypeScript
- Properties: data
- Methods: logic
- Extra data for Angular
- Defined with a decorator

Component

app.component.ts

```
import { Component } from 'angular2/core';

@Component({
  selector: 'pm-app',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
  `
})
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```

Metadata &
Template

Class

最下面是Class: 定義View要使用的 property跟method
export使這個檔案變成module

@Component: View+metadata

template定義HTML

Component

app.component.ts

```
import { Component } from 'angular2/core';

@Component({
  selector: 'pm-app',
  template:
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
  )
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```

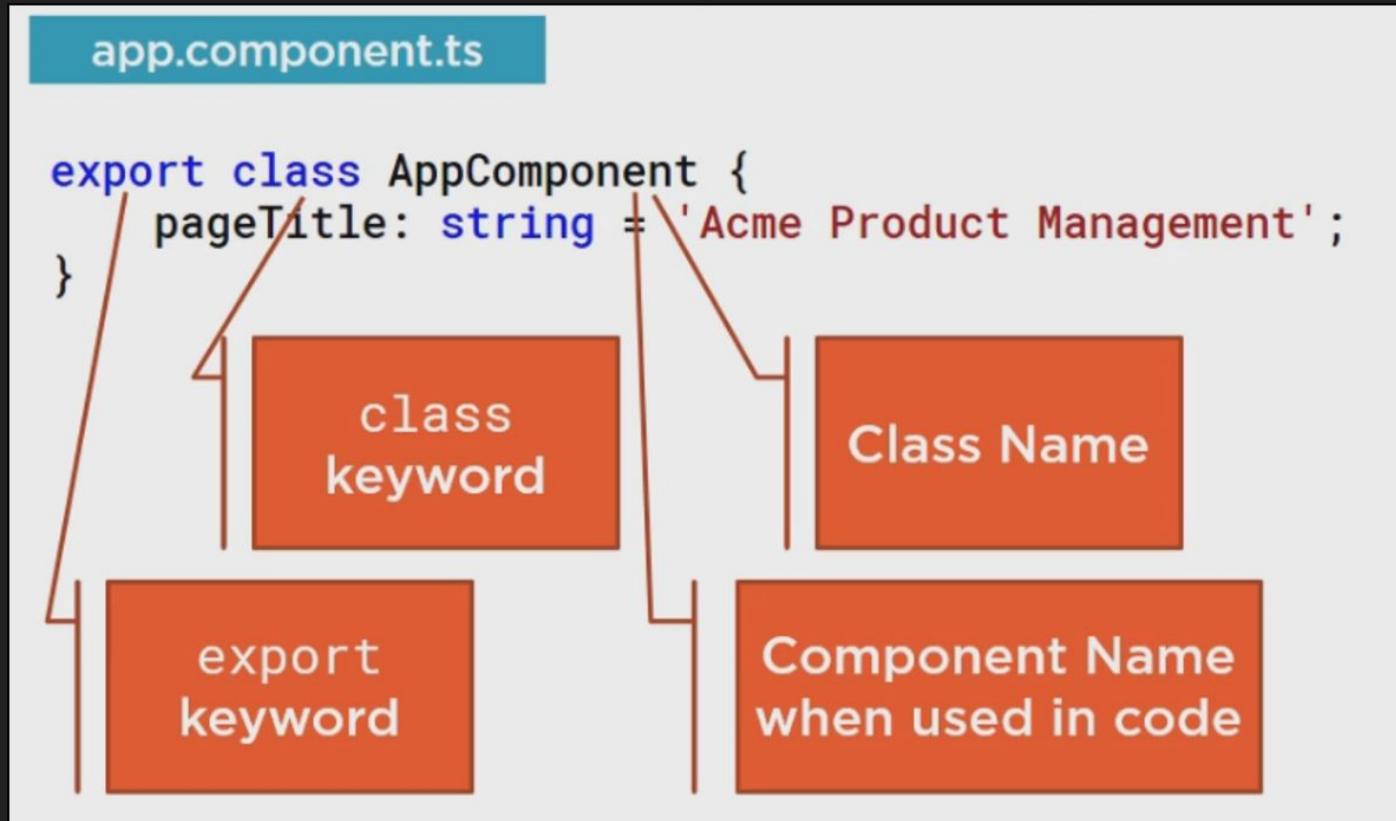
Import

Metadata &
Template

Class

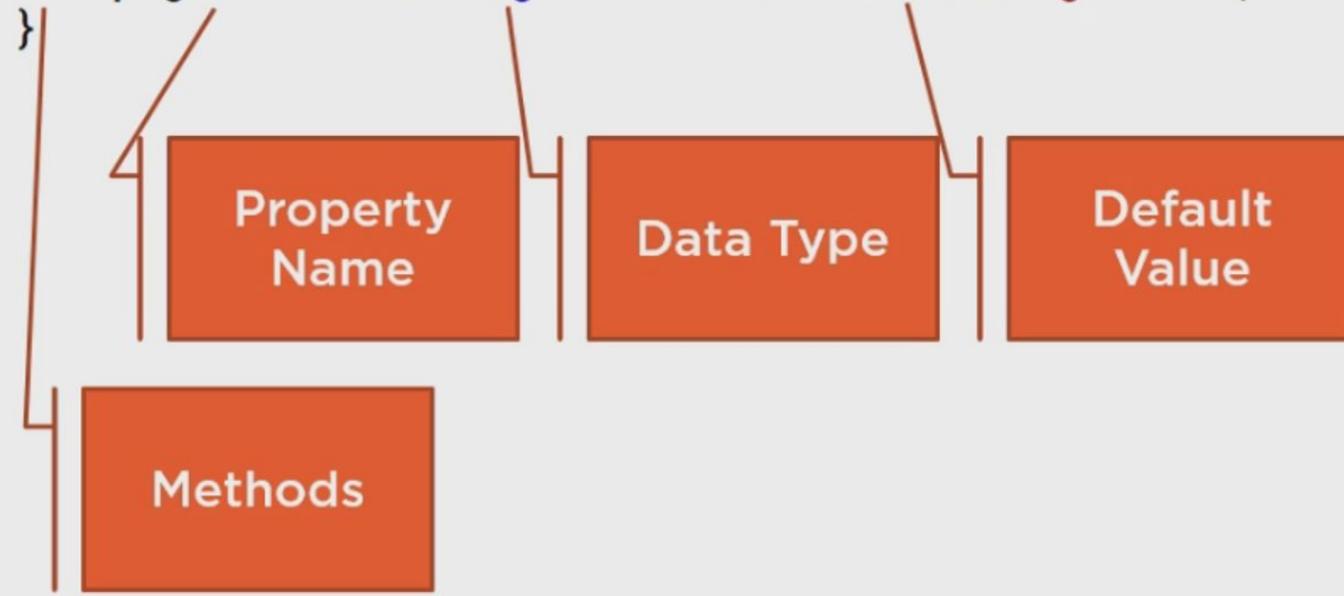
最上面是import
import需要的module

建立Component Class



app.component.ts

```
export class AppComponent {  
  pageTitle: string = 'Acme Product Management';  
}
```



Decorator

-JS feature

-在TypeScript實作

-在ES2016被提出

- 是function

-scope為其decorate的feature

decorator可加metadata到

(1)class

(2)class members

(3)class method arguments

* decorator都是以@開頭

@Component為Ag內建decorator

Angular有許多內建decorator

也可以自己建

位置

decorator都會放在要

decorate的feature前面
(注意沒有;)

```
@Component({
  selector: 'pm-app',
  template:
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
  )
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```



這邊我們要decorate的是一個class: AppComponent

Defining the Metadata

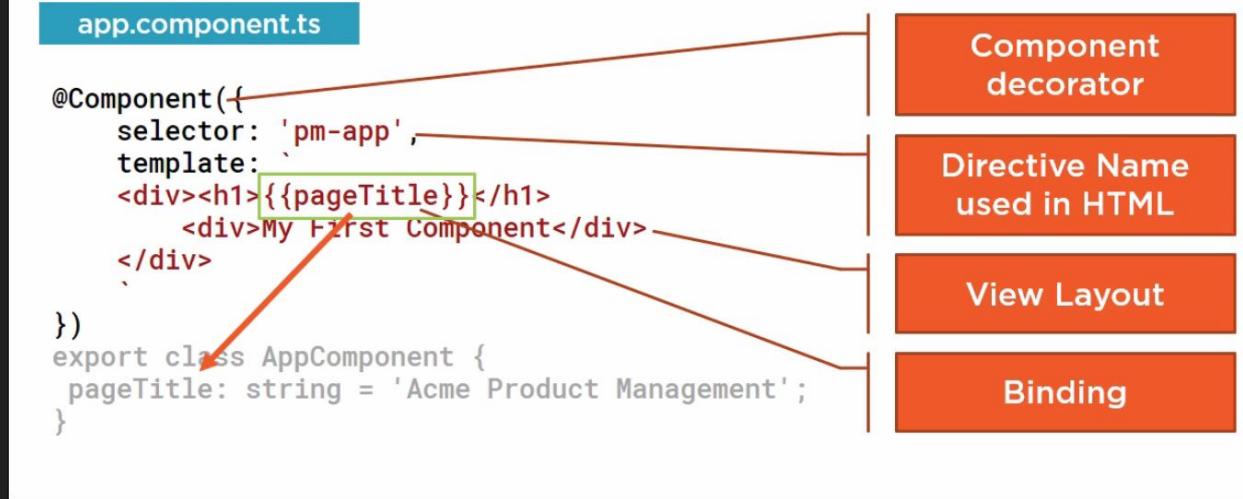
app.component.ts

```
@Component({  
    selector: 'pm-app',  
    template:  
        <div><h1>{{pageTitle}}</h1>  
            <div>My First Component</div>  
        </div>  
}  
)  
export class AppComponent {  
    pageTitle: string = 'Acme Product Management';  
}
```

Component
decorator

@Component是function=> @Component()

Defining the Metadata



傳入設定的物件 {}, 包含一些properties

`selector: 'pm-app'` Component的directive Name(HTML custom tag)

`template`: 使用此directive會載入的HTML, 裡面的{{ }} 有data-binding
會綁定class內的東西

* 當directive在HTML使用時, AG會解析該component的template

* Component一定要寫template(在裡面定義View)

import

- ES2015 feature
- Typescript 實作

1. 讓我們使用 exported members from external modules
2. 可 import 3-rd party / 自己的 / 或 Angular 的 modules

import from Angular

Angular Is Modular

@angular/
core

@angular/
animate

@angular/
http

@angular/
router

<https://www.npmjs.com/~angular>

使用import告訴module loader, 到 angular2/core去找到
Component這個function

Importing What We Need

app.component.ts

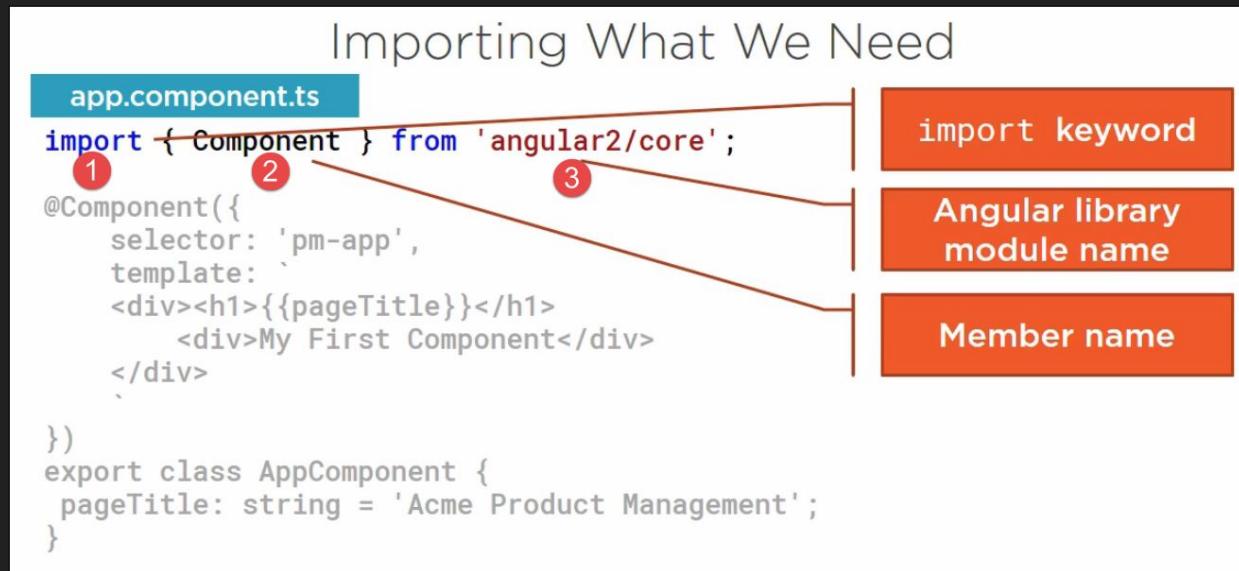
```
import { Component } from 'angular2/core';


@Component({
  selector: 'pm-app',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
  `

})
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```

1. import keyword
2. 要import的 member name (property/funtion)
3. 包含此memeber的module

* 如果要import多個membebr, 用comma (,)分隔



完整的Component

app.component.ts

```
import { Component } from 'angular2/core';

@Component({
  selector: 'pm-app',
  template: `
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
  `

})
export class AppComponent {
  pageTitle: string = 'Acme Product Management';
}
```

建立root Component

命名 convention

Feature name :

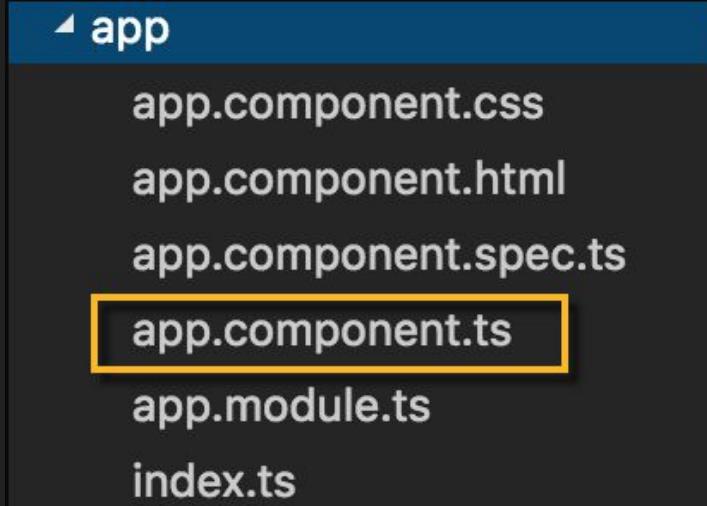
app-代表root application

Type of module :

.component-說明這個module是component

副檔名 :

typescript- **.ts**



先寫class

再寫@Component({ }) => 會有錯誤提示

所以要import Component member

```
app.component.ts x
1  @Component({})
2  export class AppComponent {
3      pageTitle: string = 'Acme Product Management';
4  }
```

```
app.component.ts app
1  @Component({
2      Cannot find name 'Component'.
3  } any
4  export class AppComponent {
5      pageTitle: string = 'Acme Product Management';
6  }
```

加上selector與template

```
app.component.ts ✘  
1 import { Component } from '@angular/core';  
2  
3 @Component({  
4   selector: 'pm-app',  
5   template: ` ->  
6     <div><h1>{{pageTitle}}</h1>  
7       <div>My First Component</div>  
8     </div>  
9   ` ->  
10 })  
11 export class AppComponent {  
12   pageTitle: string = 'Acme Product Management'  
13 }
```

Back Ticks
NOT Quotes

建立順序

注意：

template要用 `包圍
(tab上方那個按鈕)

```
app.component.ts app
1 import { Component } from 'angular2/core';
2
3 @Component({
4   selector: 'pm-app',
5   template: `
6     <div><h1>{{pageTitle}}</h1>
7       <div>My First Component</div>
8     </div>
9   `
10 })
11 export class AppComponent {
12   pageTitle: string = 'Acme Product Management';
13 }
```

3

4

2

1

Host the App

index.html

```
<body>
  <pm-app>Loading App ...</pm-app>
</body>
```

app.component.ts

```
import {Component} from 'angular2/core';

@Component({
  selector: 'pm-app',
  template:
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
```

當我們在HTML中使用directive 時，要用selector定義的名稱

即在index.html中使用<pm-app> (對AG為directive,對HTML為custom element)

當root element被load時，會先出現Loading App...

load結束時，directive template會被插入替換Loading App

index.html

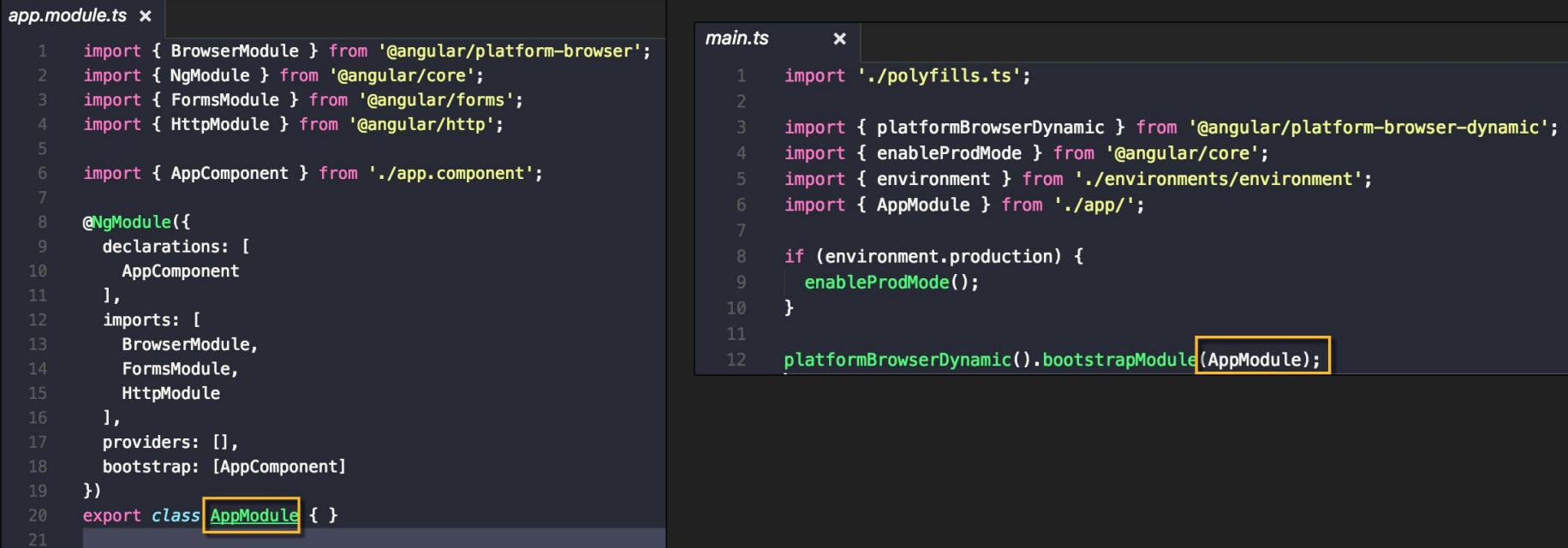
```
<body>
  <pm-app>Loading App...</pm-app>
</body>
```

app.component.ts

```
import {Component} from 'angular2/core';

@Component({
  selector: 'pm-app',
  template:
    <div><h1>{{pageTitle}}</h1>
      <div>My First Component</div>
    </div>
})
export class AppComponent {
```

main.ts 會去啟動 AppModule



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

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

```
app.module.ts ✘
```

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

declarations 設定屬於這個 module 的 component (可多個)

imports 設定 external modules

BrowserModule 是每個 browser app 必須的，有註冊重要的 service provider
如 error handling

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 works!';
10 }
```

app.module.ts ×

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

app component的selector
會在index.html內使用

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 works!';
10 }
```

index.html ×

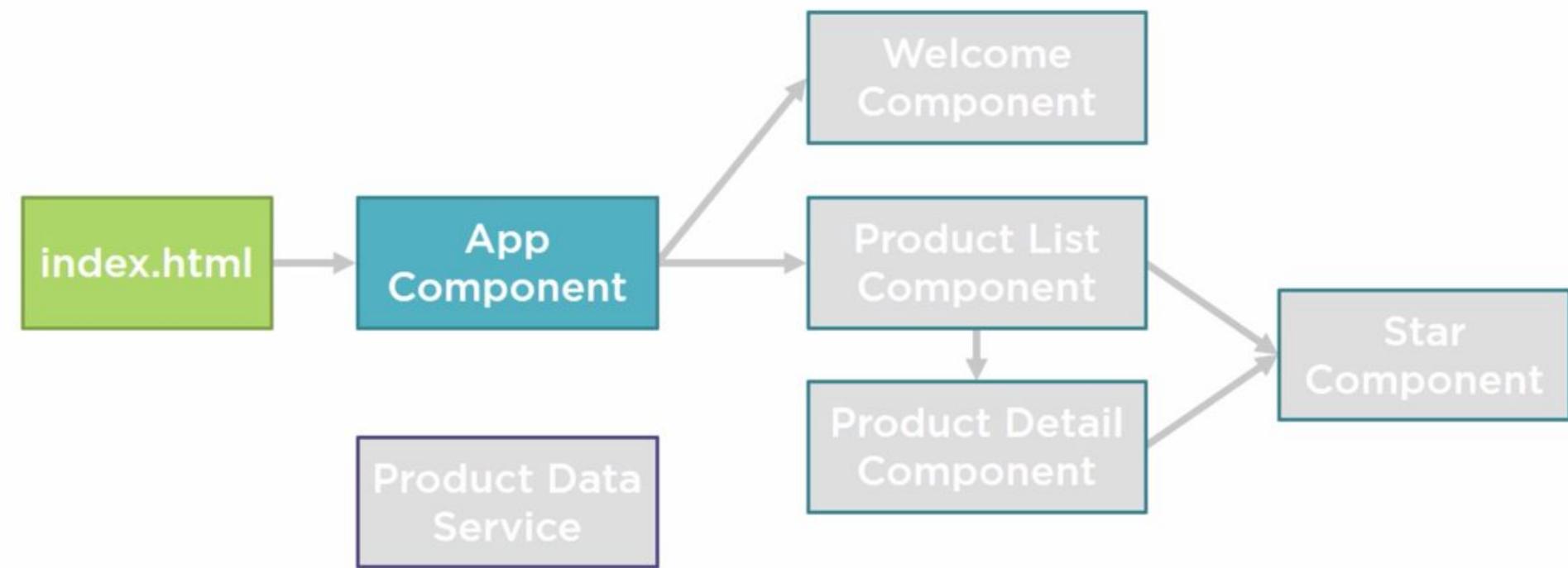
```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>Demo1211</title>
6   <base href="/">
7
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root>Loading...</app-root>
13 </body>
14 </html>
15
```

修改title 看看

The screenshot shows a Mac OS X desktop environment. At the top, there are two browser tabs: "Demo1211" and "Ng2FrontEnd". Below the tabs, the address bar displays "localhost:4200". The main window contains a browser view showing the text "Angular2 is awesome!!" and a code editor window titled "app.component.ts". A large yellow arrow points from the text in the browser to the "title" variable in the code editor.

```
app.component.ts x
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 = 'Angular2 is awesome!!';
10
11 }
```

Application Architecture



Templates
Interpolation
Directives

Data-binding :

顯示資訊+回應user動作

Directive :

加logic到HTML

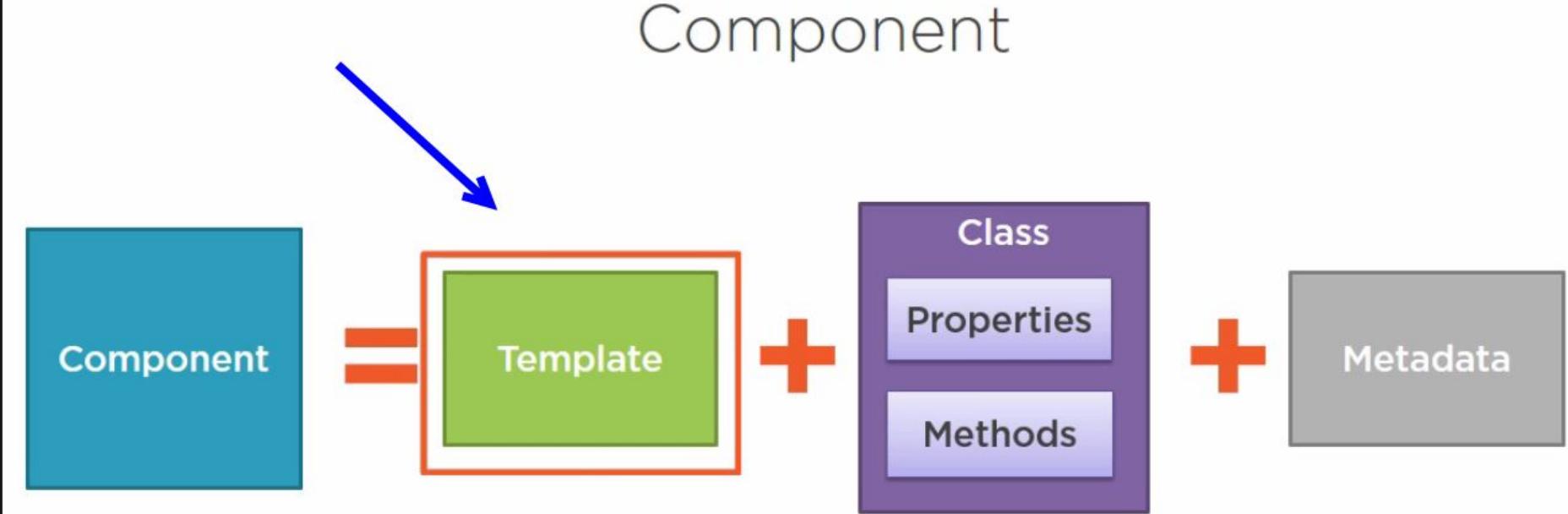
(if statement / for loops)

Component :

建立nested user interface fragments

(image rotator/rating stars)

皆下來先focus Template



Linked Template: 改用templateUrl

Defining a Template in a Component

Inline Template

```
template:  
"<h1>{{pageTitle}}</h1>"
```

1

Inline Template

```
template:  
<div>  
  <h1>{{pageTitle}}</h1>  
  <div>  
    My First Component  
  </div>  
</div>
```

2

ES 2015
Back Ticks

Linked Template

```
templateUrl:  
'product-list.component.html'
```

3

Product List component

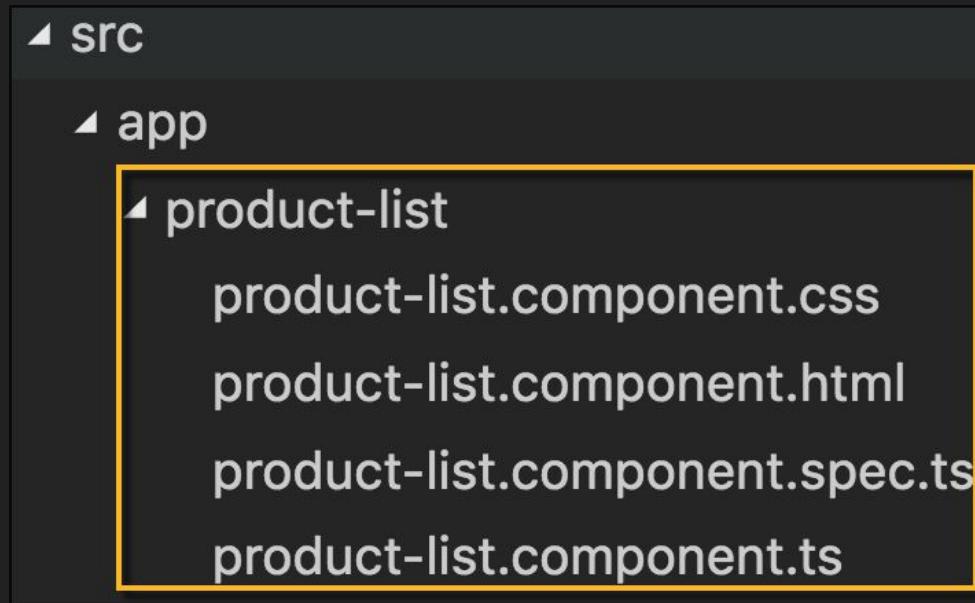
慣例會在app folder下

為各別功能建立資料夾

裡面建立html template

輸入 **ng generate component product-list**

自動幫你建好component所需的檔案



且自動在 AppModule 幫你 import 及 declare

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

Product List

建立product list

利用BS的panel / panel-primary / panel-heading

Filter by:

product-list.component.html ✘

```
1  <div class="panel panel-primary">
2      <div class="panel-heading">Product List</div>
3  </div>
4
5
```

panel-body

col-md-2

col-md-4

col-md-6

```
<div class='panel panel-primary'>
  <div class='panel-heading'>
    Product List
  </div>
  <div class='panel-body'>
    <div class='row'>
      <div class='col-md-2'>Filter by:</div>
      <div class='col-md-4'>
        <input type='text' />
      </div>
    </div>
    <div class='row'>
      <div class='col-md-6'>
        <h3>Filtered by: </h3>
      </div>
    </div>
  </div>
</div>
```

```
        <th>Available</th>
        <th>Price</th>
        <th>5 Star Rating</th>
    </tr>
</thead>
<tbody>
    </tbody>
</table>

```

```
14             <h3>Filtered by: </h3>
15         </div>
16     </div>
17     <div class='table-responsive'>
18         <table class='table'>
19             <thead>
20                 <tr>
21                     <th>
22                         <button class='btn btn-primary'>
23                             Show Image
24                         </button>
25                     </th>
26                     <th>Product</th>
27                     <th>Code</th>
28                     <th>Available</th>
29                     <th>Price</th>
30                     <th>5 Star Rating</th>
31                 </tr>
32             </thead>
33             <tbody>
34
35             </tbody>
36         </table>
37     </div>
```

```
</div>
</div>
<div class='table-responsive'>
  <table class='table'>
    <thead>
      <tr>
        <th>
          <button class='btn btn-primary'>
            Show Image
          </button>
        </th>
        <th>Product</th>
        <th>Code</th>
        <th>Available</th>
        <th>Price</th>
```

step1: 到app.component.ts

insert <app-product-list></app-product-list>

```
product-list.component.ts x app.component.html x
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-product-list', ←
5   templateUrl: './product-list.component.html',
6   styleUrls: ['./product-list.component.css']
7 })
8 export class ProductListComponent implements OnInit
9
10 constructor() { }
11
12 ngOnInit() {
13 }
14
```

```
1 <h1>
2 {{title}}
3 </h1>
4 <app-product-list></app-product-list>
5
```

step2:建立product list View(with BS)

到index.html加上

<http://getbootstrap.com/getting-started/#download>

jQuery

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
```

從cloud9 copy view (v1)

product-list.component.html

有畫面了!!

Angular2 is awesome!!

Product List

Filter by:

Filtered by:

Show Image

Product

Code

Available

Price

5 Star Rating

Angular如何知道去哪邊找這些selector?

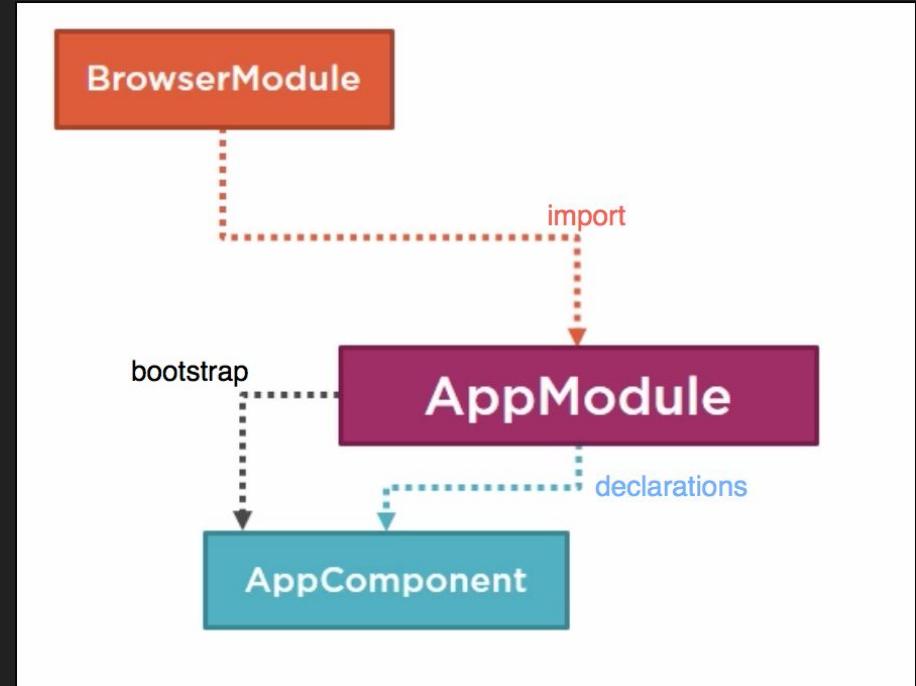
- 會去看擁有這個component的Module, 找那些看得到(visible)的directive
- 每個component都只會屬於一個Module
- 要讓ProductListComponent的selector可被AppComponent看到, 需要也在同一個Module去定義
- 如果要用的selector是在另一個module如BrowserModule一樣, 則要 import

每個angular app
至少要有一個module
(即root module-通常會命名為 **AppModule**)

每個component必須只屬於一個
Angular Module(透過 **AppModule**的 **declare**)

AppModule 啟動(**bootstrap**) **AppComponent**
所以這是第一個載入到我們app的component

AppModule有 **import BrowserModule**
(利用其功能讓app能在browser執行)



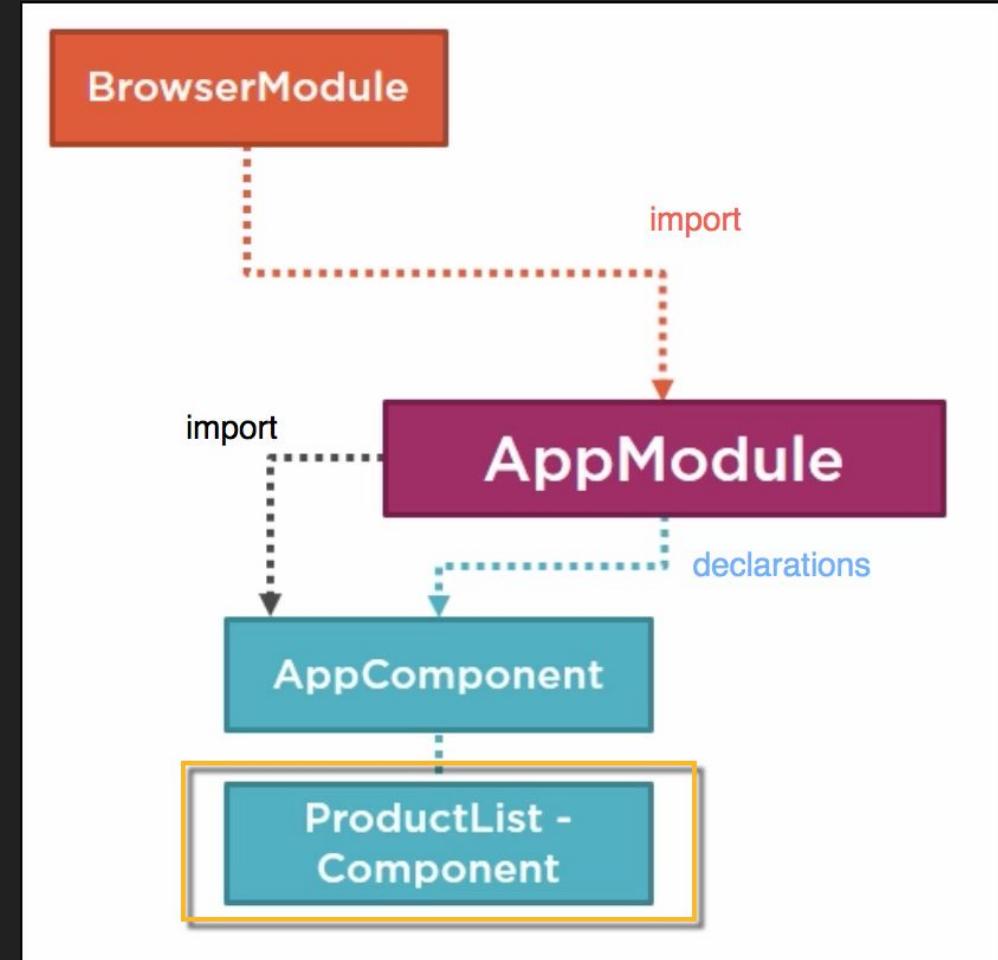
AppModule

有宣告AppComponent及
ProductListComponent

app.module.ts ×

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

Angular Module定義了
其宣告的component
能夠解析directive跟dependencies
的boundary(界線)跟context(情境)



例如在AppComponent內

(1)

有用到<app-product-list>

(2)

這個directive

(來自ProductListComponent
有設定selector)

(3)

Angular會去看 AppModule有
沒有宣告ProductListComponent

The screenshot shows a code editor with three tabs open:

- app.component.html**: Contains an `<h1> {{title}} </h1>` and a component tag `<app-product-list></app-product-list>`. A yellow box highlights the component tag, and a red circle with the number 1 is positioned above it.
- product-list.component.html**: An empty template file.
- app.module.ts**: An Angular module configuration. It includes imports for BrowserModule, NgModule, FormsModule, HttpModule, and RouterModule. The declarations array contains `AppComponent` and `ProductListComponent`, both highlighted with a yellow box. The imports array includes BrowserModule, FormsModule, HttpModule, and RouterModule. Providers and bootstrap arrays are also defined. A red circle with the number 3 is positioned above the ProductListComponent declaration.

The screenshot shows the `product-list.component.ts` file:

```
1 import { Component, OnInit } from '@angular/core'
2
3 @Component({
4   selector: 'app-product-list',
5   templateUrl: './product-list.component.html',
6   styleUrls: ['./product-list.component.css']
7 })
```

A red circle with the number 2 is positioned above the selector definition.

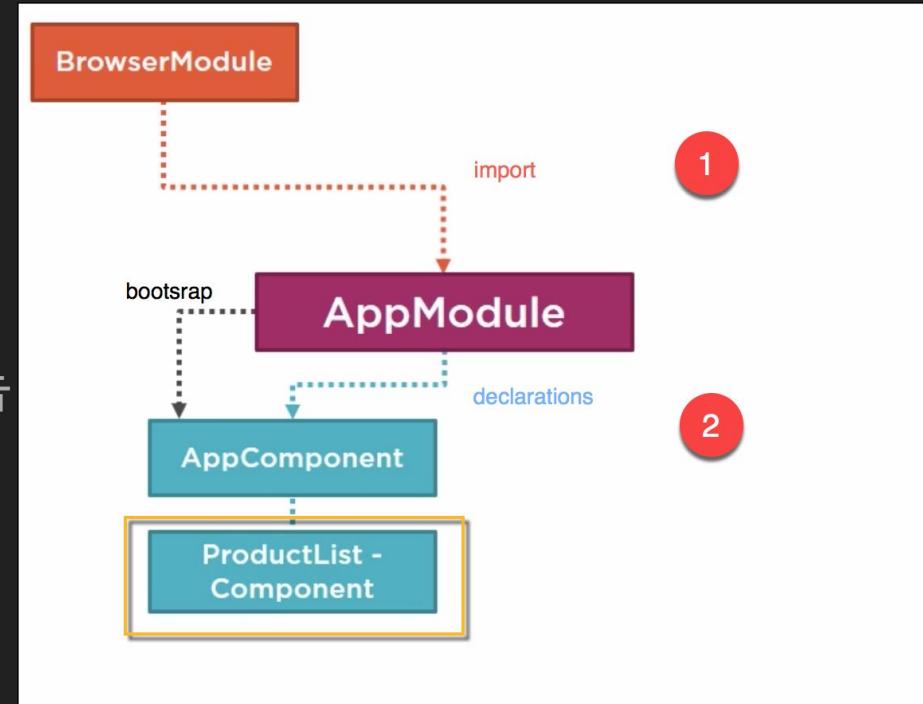
有2種方式可讓 AppModule 知道
有 directive 可用

(1) import

當 directive 已在另一個 module 有被宣告
直接匯入

(2) declarations

在自己本身有宣告



-list.component.ts

app.module.ts ×

app.component.ts

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

import { AppComponent } from './app.component';
import { ProductListComponent} from './products/product-list.component';

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ AppComponent,ProductListComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

1.在declarations加上 ProductListComponent

2. import 對應的component

* 注意這邊from path是對應此檔案

再看一次畫面

← → ⌂ ⓘ localhost:4200

Angular2 is awesome!!

Product List

Filter by:

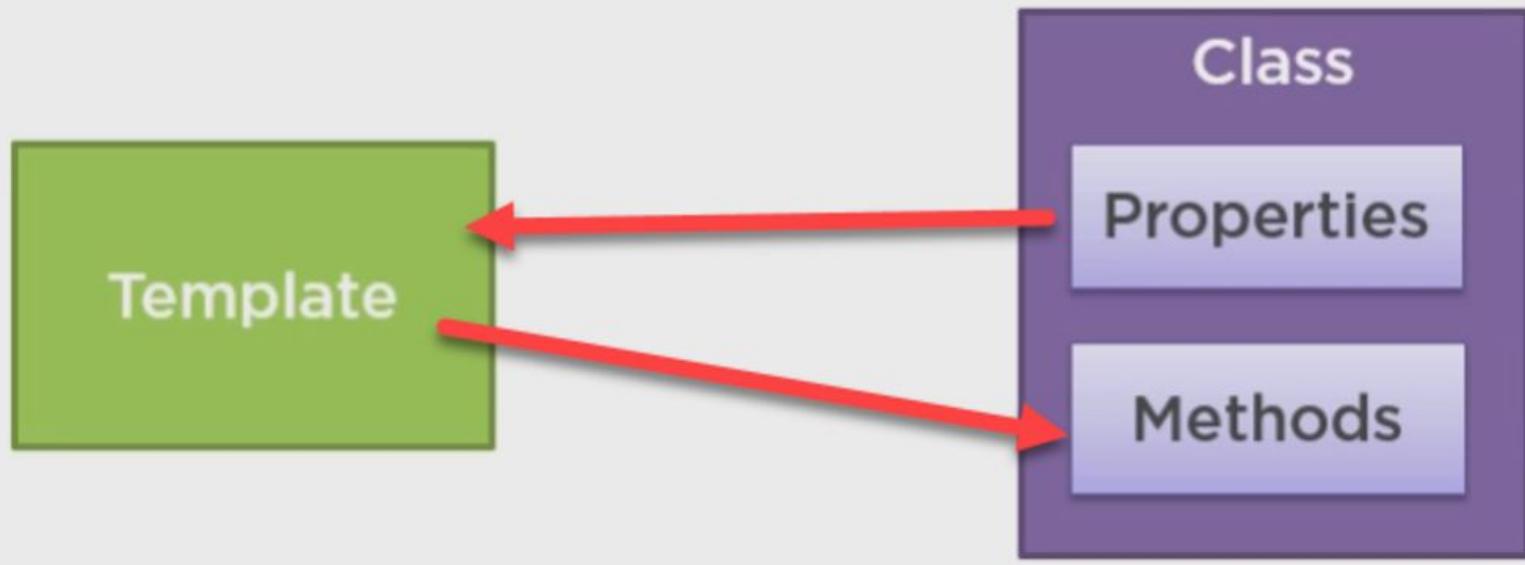
Filtered by:

Show Image	Product	Code	Available	Price	5 Star Rating
------------	---------	------	-----------	-------	---------------

Binding

什麼是Binding？

- 協調 component的class與其template之間的溝通
- 通常會傳遞資料



Interpolation

Template

```
<h1>{{pageTitle}}</h1>  
{{'Title: ' + pageTitle}}  
{ {2*20+1}}  
{{'Title: ' + getTitle()}}  
  
<h1 innerText={{pageTitle}}></h1>
```

1

2

3

4

5

Class

```
export class AppComponent {  
  pageTitle: string =  
    'Acme Product Management';  
  getTitle(): string {...};  
}
```

Template Expression {{ }}

Template

```
<h1>{{pageTitle}}</h1>  
{{'T' Template }}pageTitle}  
{ {2* $\theta+1$ }}
```

Template
Expression

調整product-list.component.html與.ts檔

product-list.component.ts x

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-product-list',
5   templateUrl: './product-list.component.html',
6   styleUrls: ['./product-list.component.css']
7 })
8 export class ProductListComponent implements OnInit {
9
10   pageTitle: string = "Product List";
11
12   constructor() { }
13
14   ngOnInit() {
15   }
16 }
```

product-list.component.html x

```
1 <div class='panel panel-primary'>
2   <div class='panel-heading'>
3     {{pageTitle}}
4   </div>
5   <!-- Filter the Products -->
6   <div class='panel-body'>
7     <div class='row'>
8       <div class='col-md-2'>Filter</div>
9       <div class='col-md-4'>
10         <input type='text' />
11       </div>
12     </div>
13     <div class='row'>
14       <div class='col-md-6'>
15         <h3>Filtered by: </h3>
16       </div>
```



Directives

Directive

客制化的 **HTML element** 或 **attribute**

用來強化或擴充HTML

- 可客製化(custom)
- 內建(built-in)

Custom Directive

```
app.component.ts          product-list.component.ts
@Component({
  selector: 'pm-app',
  template:
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    ,
  directives: [ProductListComponent]
})
export class AppComponent { }

@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent { }
```

在AppComponent的template中使用ProductListComponent的selector來呈現其template
也就是在A component中呈現 B component的template
(把component當成 directive)

Built-in Directives

**Structural
Directives**



***ngIf: If logic**

***ngFor: For loops**

*ngIf Built-In Directive

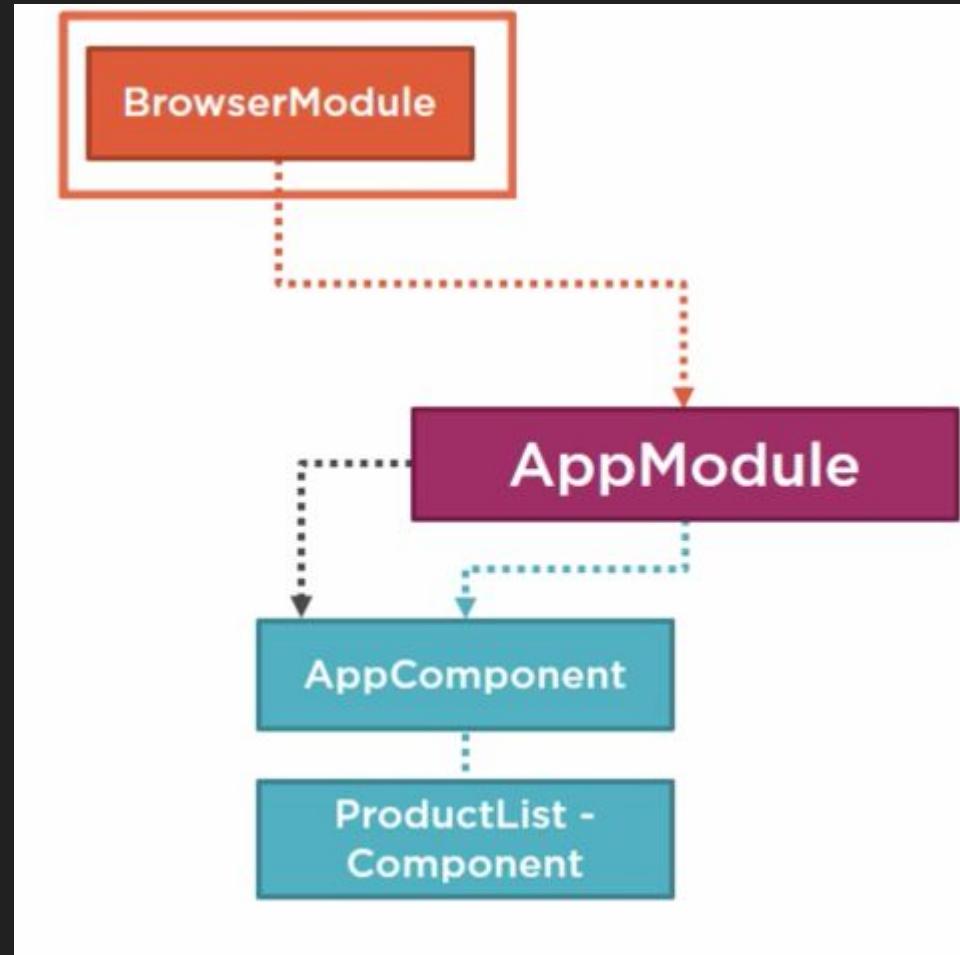
```
<div class='table-responsive'>  
  <table class='table' *ngIf='products && products.length'>  
    <thead> ...  
    </thead>  
    <tbody> ...  
    </tbody>  
  </table>  
</div>
```

```
...<!DOCTYPE html>  
<html>  
  <head>...</head>  
  <body>  
    <script id="__bs_script__" type="text/javascript">...</script>  
    <script async="" src="/browser-sync/browser-sync-client.2.11.2.js"></script>  
    <pm-app>  
      <div>  
        <h1>Acme Product Management</h1>  
        <pm-products>  
          <div class="panel panel-primary">  
            <div class="panel-heading">  
              Product List  
            </div>  
            <div class="panel-body">  
              ::before  
              <div class="row">...</div>  
              <div class="row">...</div>  
              <div class="table-responsive">  
                <!--template bindings={}-->  
                <div>  
                  ::after  
                </div>  
              </div>  
            </div>  
          </pm-products>  
        </div>  
      </pm-app>  
    </body>  
</html>
```

application去哪邊找到
*ngIf ?

BrowserModule 會 expose
*ngIf
*ngFor

所以宣告在 AppModule 的
component 都可以使用
*ngIf 跟 *ngFor



實作 : 要讓table依據有沒有products而顯示出來

因為不知道products array中的item type, 所以先放 any

(通常data會從後端來)

```
ngOnInit() {  
}  
  
pageTitle:string = "Product List!";  
products:any[] = |
```

貼上api內product資料

```
pageTitle:string = "Product List!";
products:any[]=[
{
    "productId": 1,
    "productName": "Leaf Rake",
    "productCode": "GDN-0011",
    "releaseDate": "March 19, 2016",
    "description": "Leaf rake with 48-inch wooden handle.",
    "price": 19.95,
    "starRating": 3.2,
    "imageUrl": "http://openclipart.org/image/300px/svg_to_png/26"
},
{
    "productId": 2,
    "productName": "Garden Cart",
    "productCode": "GDN-0023",
    "releaseDate": "March 18, 2016",
    "description": "15 gallon capacity rolling garden cart",
    "price": 32.99,
    "starRating": 4.5
}
]
```

*ngIf 後面要加上' '

table element 加上 *ngIf

如果products存在且有東西，就顯示table

```
<div class='table-responsive'>
  <table class='table' *ngIf='products && products.length'>
    <thead>
      <tr>
        <th>
          <button class='btn btn-primary'>Show Image
        </button>
```

有product資料時

Product List!						
Filter by:						
Filtered by:	Show Image	Product	Code	Available	Price	5 Star Rating

註解掉products時



*ngFor product of products

```
<tr *ngFor='let product of products'>
  <td></td>
  <td>{{ product.productName }}</td>
  <td>{{ product.productCode }}</td>
  <td>{{ product.releaseDate }}</td>
  <td>{{ product.price }}</td>
  <td>{{ product.starRating }}</td>
</tr>
```

Template
input variable

要在<tbody>顯示資料

```
    <th>Price</th>
    <th>5 Star Rating</th>
  </tr>
</thead>
<tbody>
</tbody>
</table>
</div>
</div>
```

對應th去產生td

第一個欄位是image

接著是Product

Code

Available

```
*ngIf='products && products.length'>
<thead>
  <tr>
    <th>
      <button class='btn btn-primary'>
        Show Image
      </button>
    </th>
    <th>Product</th>
    <th>Code</th>
    <th>Available</th>
    <th>Price</th>
    <th>5 Star Rating</th>
  </tr>
</thead>
<tbody>
  <tr *ngFor='#product of products'>
    <td></td>
  </tr>
</tbody>
```

```
</thead>
<tbody>
  <tr *ngFor='let product of products'>
    <td></td>
    <td>{{product.productName}}</td>
    <td>{{product.productCode}}</td>
    <td>{{product.releaseDate}}</td>
    <td>{{product.price}}</td>
    <td>{{product.starRating}}</td>
  </tr>
</tbody>
</table>
```

如何知道property name?

The image shows a code editor with two files open: `product-list.component.html` and `products.json`.

`product-list.component.html` contains the following code:

```
<thead>
  <tr>
    <th>Product</th>
    <th>Code</th>
    <th>Available</th>
    <th>Price</th>
    <th>5 Star Rating</th>
  </thead>
  <tbody>
    <tr *ngFor='let product of products'>
      <td></td>
      <td>{{ product.productName }}</td>
      <td>{{ product.productCode }}</td>
      <td>{{ product.releaseDate }}</td>
      <td>{{ product.price }}</td>
      <td>{{ product.starRating }}</td>
    </tr>
  </tbody>

```

`products.json` contains the following JSON data:

```
[{"productId": 1, "productName": "Leaf Rake", "productCode": "GDN-0011", "releaseDate": "March 19, 2016", "description": "Leaf rake with 4 wooden handles.", "price": 19.95, "starRating": 3.2, "imageUrl": "http://openclipart.org/image/240px_color_800x800px_264318/leaf-rake.png"}, {"productId": 2, "productName": "Garden Cart", "productCode": "GDN-0023", "releaseDate": "March 18, 2016", "description": "Garden cart with 2 wheels and a handle.", "price": 39.99, "starRating": 4.5, "imageUrl": "http://openclipart.org/image/240px_color_800x800px_264318/garden-cart.png"}]
```

A yellow arrow points from the highlighted `productName` in the `product-list.component.html` template to the `productName` property in the first object of the `products.json` data.

```
{  
    "productId": 2,  
    "productName": "Garden Cart",  
    "productCode": "GDN-0023",  
    "releaseDate": "March 18, 2016",  
    "description": "15 gallon capacity rol  
    "price": 32.99,  
    "starRating": 4.2,  
    "imageUrl": "http://openclipart.org/i  
,  
    s
```

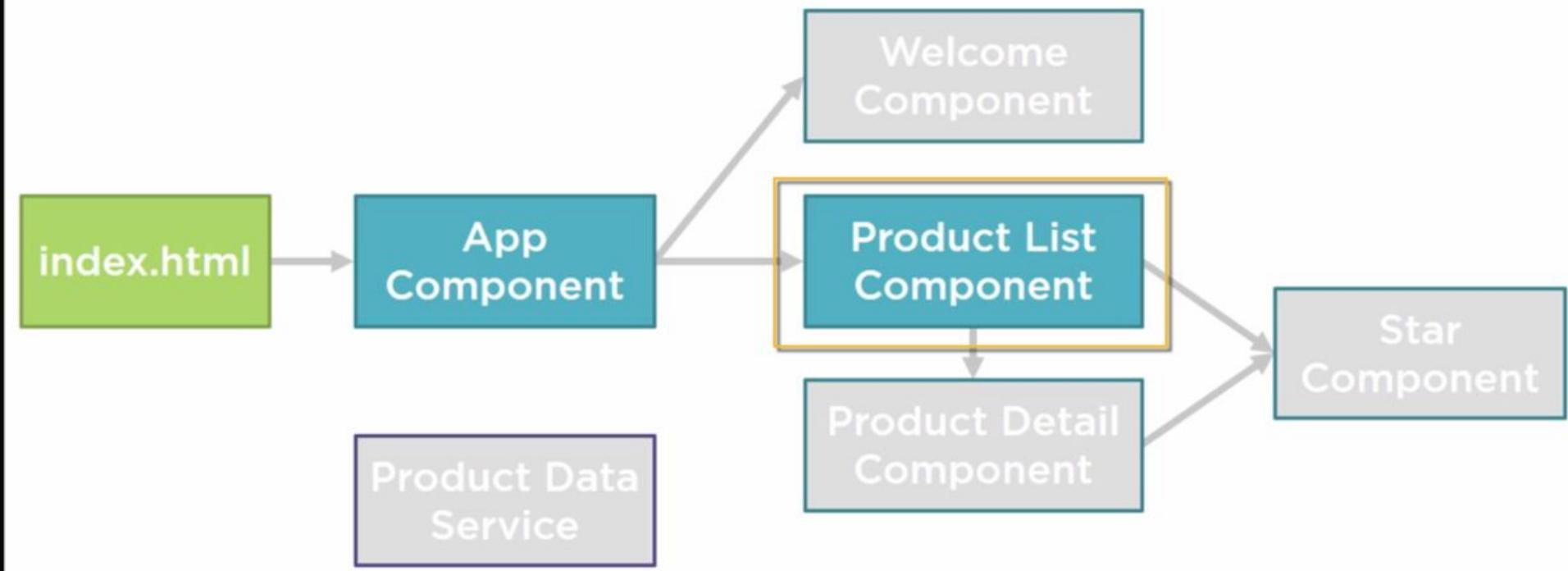
```
table class='table' *ngIf='products && products.length'><thead><tr><th><button class='btn btn-primary'>Show Image</button></th><th>Product</th><th>Code</th><th>Available</th><th>Price</th><th>5 Star Rating</th></tr></thead><tbody><tr *ngFor='let product of products'><td></td><td>{{product.productName}}</td><td>{{product.productCode}}</td><td>{{product.releaseDate}}</td><td>{{product.price}}</td><td>{{product.starRating}}</td></tr></tbody>
```

```
3      @Component({  
4          selector: 'pm-product-list',  
5          templateUrl: 'app/product-list/product-list.component.html'  
6      })  
7      export class ProductListComponent {  
8          pageTitle: string = 'Product List';  
9          products: any[] = [  
10              {  
11                  "productId": 1,  
12                  "productName": "T-shirt",  
13                  "productCode": "TS-001",  
14                  "releaseDate": "2023-01-01",  
15                  "description": "A simple t-shirt.",  
16                  "price": 10,  
17                  "starRating": 4.5,  
18                  "imageUrl": "https://example.com/t-shirt.jpg"  
19              },  
20              {  
21                  "productId": 2,  
22                  "productName": "Sweatshirt",  
23                  "productCode": "SW-002",  
24                  "releaseDate": "2023-01-01",  
25                  "description": "A comfortable sweatshirt.",  
26                  "price": 20,  
27                  "starRating": 4.8,  
28                  "imageUrl": "https://example.com/sweatshirt.jpg"  
29              },  
30              {  
31                  "productId": 3,  
32                  "productName": "Hoodie",  
33                  "productCode": "HO-003",  
34                  "releaseDate": "2023-01-01",  
35                  "description": "A cozy hoodie.",  
36                  "price": 30,  
37                  "starRating": 4.9,  
38                  "imageUrl": "https://example.com/hoodie.jpg"  
39              }  
40          ]  
41      }  
42  
```

產品出現

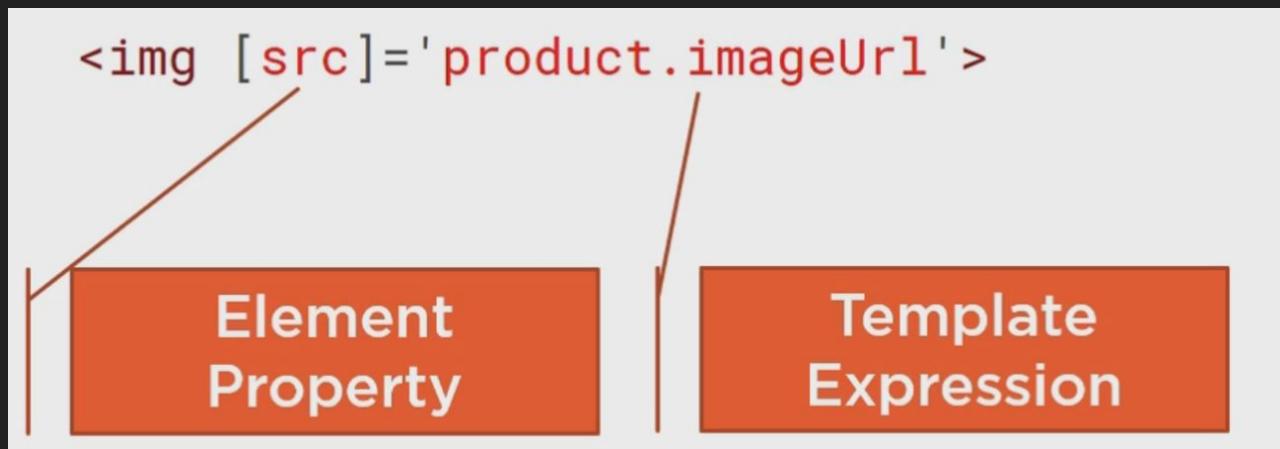
Filtered by:

Sgow Image	Product	Code	Available	Price	5 Star Rating
	Leaf Rake	GDN-0011	March 19, 2016	19.95	3.2
	Garden Cart	GDN-0023	March 18, 2016	32.99	4.2
	Hammer	TBX-0048	May 21, 2016	8.9	4.8
	Saw	TBX-0022	May 15, 2016	11.55	3.7
	Video Game Controller	GMG-0042	October 15, 2015	35.95	4.6



Data Binding & Pipes

Property Binding



把 `img element` 的 `source property` 繩定(bind) 到 `product` 的 `property imageUrl`
(把 `src` 的值 參考到 class 中的資訊)

```
<img [src]='product.imageUrl'>
```

[]

Binding Target

''

Binding Source

Binding Target 都要用[]，放在=的左邊 (property of the element)

Binding Source都要用'', 放在=的右邊 (Template Expression)

Property Binding

```
<img [src]='product.imageUrl'> 1
```

```
<img src={{product.imageUrl}}> 2
```

```
<img src='http://openclipart.org/{{product.imageUrl}}'> 3
```

(1) property binding 跟(2)interpolation 都是one way binding

(class 中的property => element 中 property)

可達成同樣結果時

一般建議用(1)

但如(3), 需要把template expression 放到比較大的expression中時, 就要用 interpolation

```
<th>Product</th>
<th>Code</th>
<th>Available</th>
<th>Price</th>
<th>5 Star Rating</th>

```

head>

body>

```
    <tr *ngFor='let product of products'>
        <td>
            <img [src]='product.imageUrl'
                  [title]='product.productName'>
        </td>
        <td>{{ product.productName }}</td>
        <td>{{ product.productCode }}</td>
    
```

```
10    pageTitle: string = "Product List";
11    products: any[] = [
12        {
13            "productId": 1,
14            "productName": "Leaf Rake",
15            "productCode": "GDN-0011",
16            "releaseDate": "March 19, 2016",
17            "description": "Leaf rake with 48-",
18            "price": 19.95,
19            "starRating": 3.2,
20            "imageUrl": "http://openclipart.or
21        },
22        {
23            "productId": 2,

```

圖太大怎麼辦？



```
</thead>
<tbody>
  <tr *ngFor='let product of products'>
    <td>
      <img [src]='product.imageUrl'
            [title]='product.productName'
            [style.width.px]='imageWidth'
            [style.margin.px]='imageMargin'>
    </td>
    <td>{{ product.productName }}</td>
    <td>{{ product.productCode }}</td>

```

8 export class ProductListComponent in
9
10 pageTitle: string = "Product Li:
11 imageWidth: number = 50;
12 imageMargin: number = 2;
13 products: any[] = [
14 {
15 "productId": 1,
16 "productName": "Leaf Ra
17 "productCode": "GDN-001:
18 "productDetail": "Manufac

[style.width.px]

設定寬度, 單位為px

[style.margin.px]

設定margin

注意這邊imageWidth前面

沒有product

因為imageWidth是

class property

Filtered by:

Show Image	Product	Code	Available
	Leaf Rake	GDN-0011	March 19, 2016
	Garden Cart	GDN-0023	March 18, 2016
	Hammer	TBX-0048	May 21, 2016

Event Binding

Handling Events with event binding

Template

```
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
<button (click)='toggleImage()'>
```

Class

```
export classListComponent {
  pageTitle: string = 'Product List';
  products: any[] = [...];
```

()
Target Event

''
Template Statement

發生Event時

當event發生時

template statement
會被執行

```
<button @click='toggleImage()'>
```

```
export class ListCom
  pageTitle: string =
  products: any[] = [
    toggleImage(): void
  }
```

DOM Events

<https://developer.mozilla.org/en-US/docs/Web/Events>

Mouse Events

Event Name	Fired When
<code>mouseenter</code>	A pointing device is moved onto the element that has the listener attached.
<code>mouseover</code>	A pointing device is moved onto the element that has the listener attached or onto one of its children.
<code>mousemove</code>	A pointing device is moved over an element. (Fired continuously as the mouse moves.)
<code>mousedown</code>	A pointing device button (ANY button) is pressed on an element.
<code>mouseup</code>	A pointing device button (ANY button) is released over an element.
<code>click</code>	A pointing device button (ANY button) has been pressed and released on an element.
<code>dblclick</code>	A pointing device button is clicked twice on an element.

實作show image button 功能

要有個flag去切換

```
10  constructor() { }
11
12  ngOnInit() {
13
14    pageTitle:string = "Product List";
15    imageWidth:number = 50;
16    imageMargin:number = 2;
17    showImage:boolean = false; // This line is highlighted with a yellow box
18    products:any[]=[
19      {
20        "productId": 1,
21        "productName": "Leaf Rake",
22        "productCode": "GDN-0011".
```

當button點下去時, flag要切換

用method來控制

通常method寫在所有property後面

* typescript的function不用寫function keyword

:void表示沒有回傳值

```
pageTitle: string = "Product List";
imageWidth: number = 50;
imageMargin: number = 2;
showImage: boolean = false;
products: any[] = [...];
];
```

```
toggleImage(): void {
    this.showImage = !this.showImage;
}
```

點button時觸發toggleImage()

用event binding綁定click event

```
<tr>
  <th>
    <button class='btn btn-primary'
      (click)='toggleImage()'>Show Image</button>
  </th>
  <th>Product</th>
  <th>Code</th>
  <th>Available</th>
```



```
65      ];
66
67      toggleImage(): void {
68        this.showImage = !this.
69      }
70
71      constructor() { }
```

要把image隱藏或顯示

```
<td>
  <img *ngIf='showImage'
    [src]='product.imageUrl' [title]='product.name'
</td>
```

測試

Filtered by:

Show Image

Product

Leaf Rake

Garden Cart

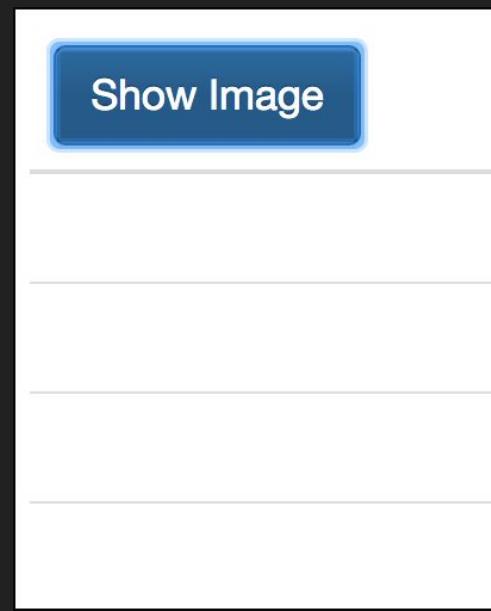
Hammer

Saw

Video Game Controller

調整button text

根據不同情況變化



JS condition operator

```
<th>
  <button class='btn btn-primary' (click)=
    {{showImage ?'Hide':'Show'}} Image
  </button>
</th>
<th>Product</th>
```

Handling Input
with
2-way binding

Two-way data binding

Template

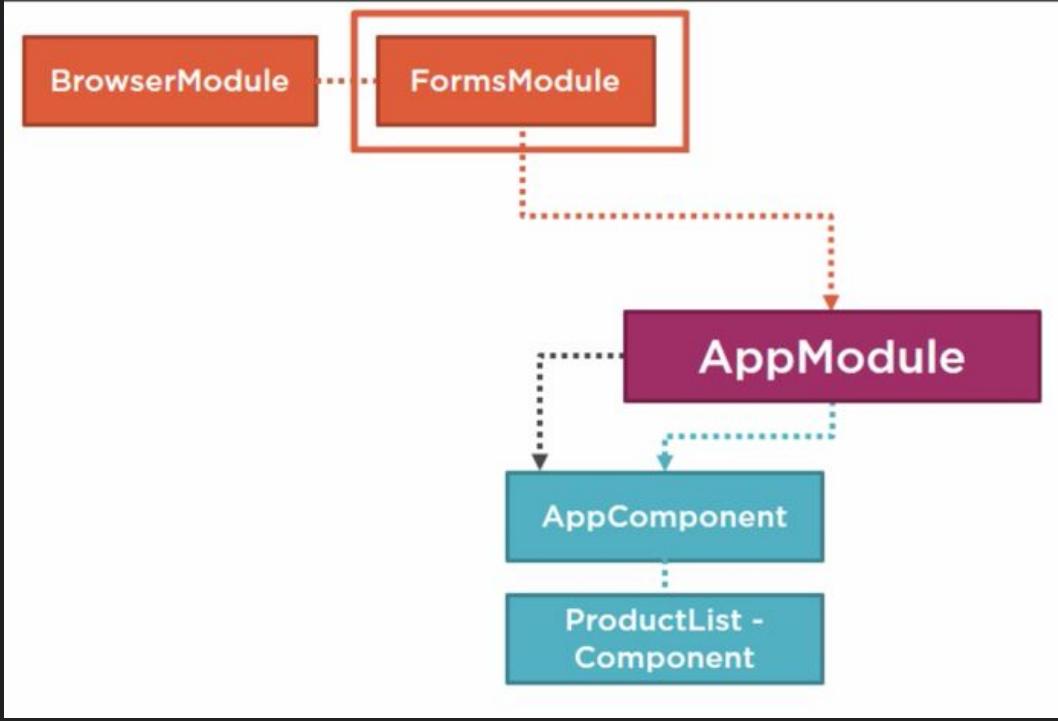
```
<input [(ngModel)]='listFilter'>
```

Class

```
export class ListComponent {  
  listFilter: string = 'cart';  
}
```

[0]
Banana in a Box





ngModel是FormsModule的一部份，需要import

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

import { AppComponent } from './app.component';
import { ProductListComponent } from './product-list.component';

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

```
<div class='col-md-2'>Filter by:</div>
<div class='col-md-4'>
  <input type='text' [(ngModel)]='listFilter' />
</div>
</div>
<div class='row'>
  <div class='col-md-6'>
    <h3>Filtered by: {{listFilter}}</h3>
  </div>
</div>
```



```
12
13
14
15 +
62
63
64
65
66
67
68
69
70
71
```

```
imageMargin: number = 2;
showImage: boolean = false;
listFilter: string = 'cart';
products: any[] = [...];
];

toggleImage(): void {
  // alert('click');
  this.showImage = !this.showImage;
}
```

新增 listFilter property
幫input綁定 ngModel
顯示 {{listFilter}}

測試

Product List

Filter by: cart123

Filtered by: cart123

Show Image	Product
	Leaf Rake
	Garden Cart

Transforming Data with Pipes

- 顯示綁定的properties前，先做轉換
- 內建pipes
 - Date, number, percent, currency
 - json, slice (處理object)
 - etc
- 客製化 pipes

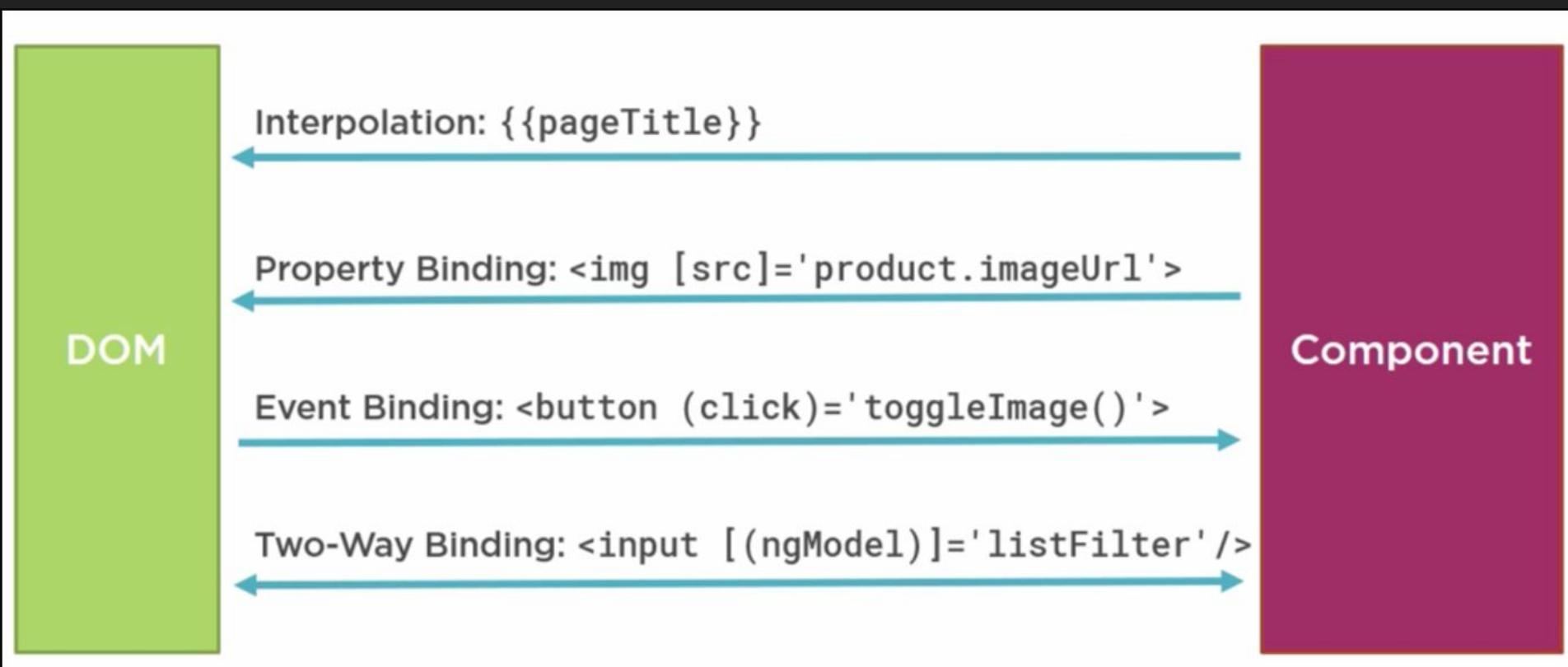
Pipe Example

```
 {{ product.productCode | lowercase }}  
  
<img [src]='product.imageUrl'  
      [title]='product.productName | uppercase'>  
  
 {{ product.price | currency | lowercase }}  
  
 {{ product.price | currency:'USD':true:'1.2-2' }}
```

```
<td>{{ product.productName }}</td>
<td>{{ product.productCode | lowercase }}</td>
<td>{{ product.releaseDate }}</td>
<td>{{ product.price | currency:'USD':true:'1.1-2' }}</td>
<td>{{ product.starRating }}</td>
</tr>
```

Code	Available	Price
gdn-0011	March 19, 2016	\$19.95
gdn-0023	March 18, 2016	\$32.99
tbx-0048	May 21, 2016	\$8.90
tbx-0022	May 15, 2016	\$11.55
gmg-0042	October 15, 2015	\$35.95

Data Binding 記憶圖



使用ngModel要記得import FormsModule

讓跟這個Module相關的Component都能使用ngModel
(AppComponent/ProductListComponent)

Checklist: ngModel

product-list.component.html

```
<div class='col-md-4'>
  <input type='text'
    [(ngModel)]='listFilter' />
</div>
```

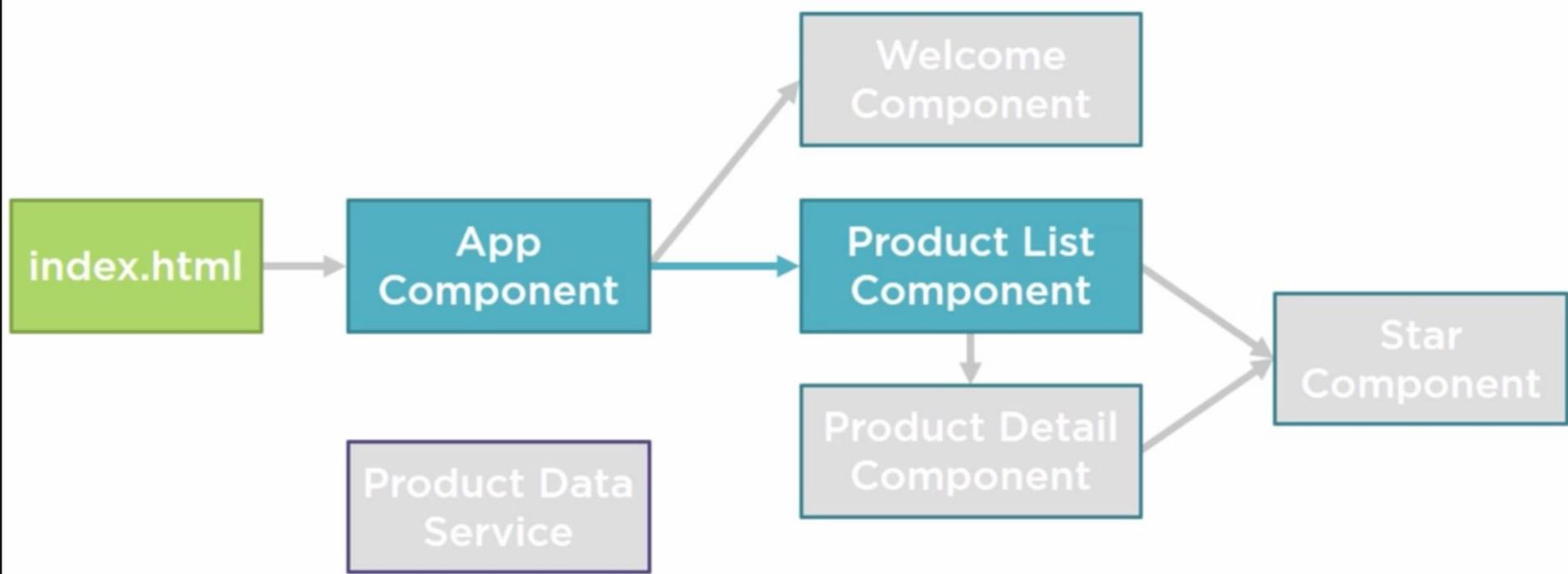
app.module.ts

```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule ],
  declarations: [
    AppComponent,
    ProductListComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

More on Components

- Strong typing & interfaces
- Encapsulating styles
- Lifecycle hooks
- Custom pipes

App 架構



Strong typing

可以有type的地方

- (1) property
- (2) method的return type
- (3) method的parameter type

但有時候property/method

會沒有適合的predefined type

如這裡的products array為any

要指定custom type, 要使用interface

```
export class productListComponent {  
    pageTitle: string = 'Product List';  
    showImage: boolean = false;  
    listFilter: string = 'cart';  
    message: string;  
  
    products: any[] = [...];  
  
    toggleImage(): void {  
        this.showImage = !this.showImage;  
    }  
  
    onRatingClicked(message: string): void {  
        this.message = message;  
    }  
}
```

Interface

一份定義properties/methods的規格

有實作(implement) Interface的class會支援這份規格

並可把interface當成data type來用

主要提供strong typing跟debug

ES5跟ES6不支援，但TypeScript支援

只是transpile後會消失，不會出現在JS檔內
(interface只有development time only)

TypeScript Interface範例

```
export interface IProduct {  
    productId: number;  
    productName: string;  
    productCode: string;  
    releaseDate: Date;  
    price: number;  
    description: string;  
    starRating: number;  
    imageUrl: string;  
    calculateDiscount(percent: number): number;  
}
```

export keyword

Interface Name

interface keyword

- **export** 讓Interface可以在app中到處被使用
- 命名通常會在最前面加上**I**代表Interface

定義Interface

```
export interface IProduct {  
    productId: number;  
    productName: string;  
    productCode: string;  
    releaseDate: Date;  
    price: number;  
    description: string;  
    starRating: number;  
    imageUrl: string;  
    calculateDiscount(percent: number): number;  
}
```

在interface中定義property跟method如上(含有data type)

使用Interface做為data type

- (1) import interface
- (2) 當成data type來用

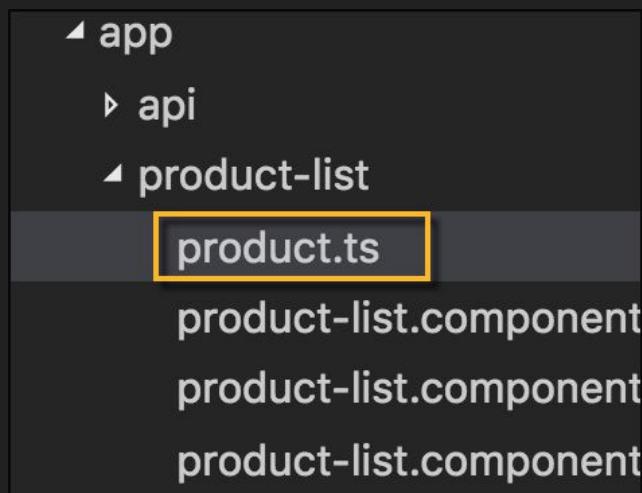
```
import { IProduct } from './product';  
  
export class ProductListComponent {  
    pageTitle: string = 'Product List';  
    showImage: boolean = false;  
    listFilter: string = 'cart';  
  
    products: IProduct[] = [...];  
  
    toggleImage(): void {  
        this.showImage = !this.showImage;  
    }  
}
```

新增product.ts

要修正any

新增Interface來定義product是什麼

新增product.ts



```
listFilter: string = 'cart';
products: any[] = [
  {
    "productId": 1,
    "productName": "Leaf Rake",
    "productCode": "LR01234",
    "releaseDate": "2021-09-23",
    "price": 99.99,
    "description": "A high-quality leaf rake for lawn and garden work. Made from durable steel and wood.",  
...
```

product.ts

```
export interface IProduct {
  productId: number;
  productName: string;
  productCode: string;
  releaseDate: string;
  price: number;
  description: string;
  starRating: number;
  imageUrl: string;
```

product-list.component.ts ×

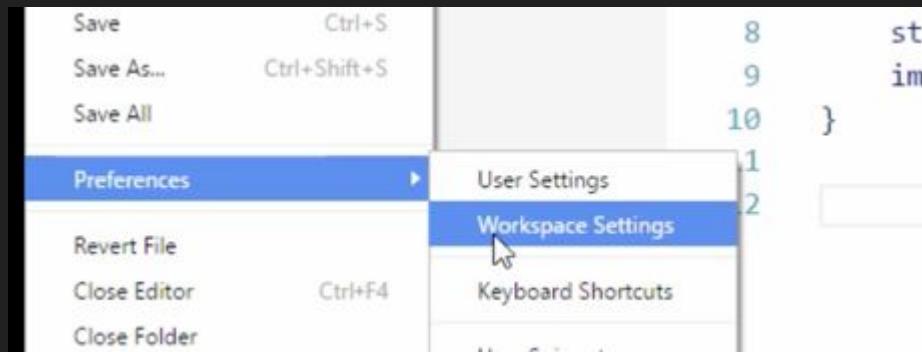
```
"price": 32.99,  
"starRating": 4.2,  
"imageUr": "http://openclipart.org/image/300px_800  
,  
{  
    imageUrl  
}
```

type checking

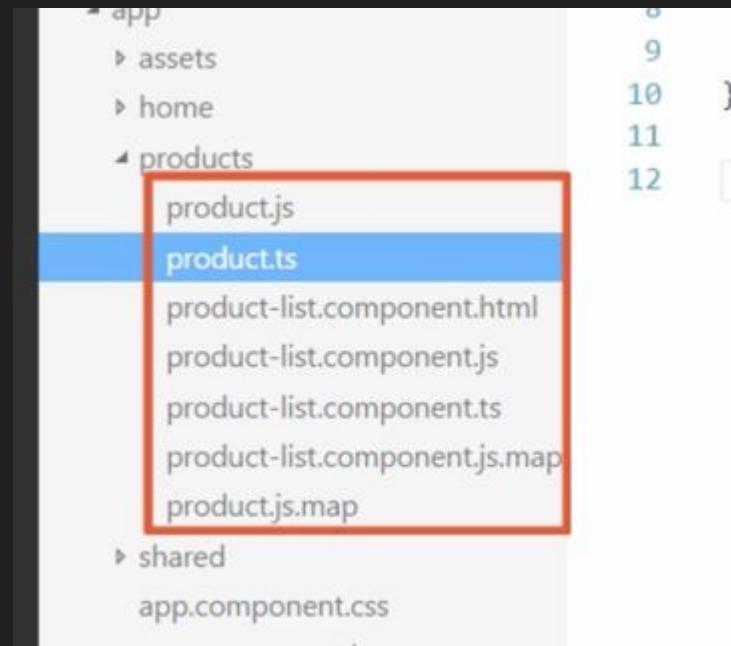
intellisense

```
1   import { Component, OnInit } from '@angular/core';  
2   import { IProduct } from './product';  
3  
4   @Component({  
5       selector: 'app-product-list',  
6       templateUrl: './product-list.component.html',  
7       styleUrls: ['./product-list.component.css']  
8   })  
9   export class ProductListComponent implements OnInit {  
10  
11       pageTitle: string = "Product List";  
12       imageWidth: number = 50;  
13       imageMargin: number = 2;  
14       showImage: boolean = false;  
15       listFilter: string = 'cart';  
16       products: IProduct[] = [  
17           {  
18               "productId": 1,  
19               "productName": "Leaf Rake",  
20               "productCode": "GDN-0011",  
21               "releaseDate": "March 19, 2016",  
22               "description": "Leaf rake with 48-inch wooden",  
23               "price": 19.94,  
24               "starRating": 3.2,  
25               "imageUrls": "http://openclipart.org/image/300px_800  
26           },  
27       ],  
28       filteredProducts: IProduct[] = []  
29   }  
30  
31   ngOnInit(): void {  
32       this.filteredProducts = this.products;  
33   }  
34  
35   applyFilter(filterValue: string): void {  
36       filterValue = filterValue.trim();  
37       filterValue = filterValue.toLowerCase();  
38       this.filteredProducts = this.products.filter((product) =>  
39           product.productName.toLowerCase().includes(filterValue)  
40       );  
41   }  
42  
43   addProduct(): void {  
44       const newProduct: IProduct = {  
45           productId: 5,  
46           productName: "Garden Trowel",  
47           productCode: "GDN-0012",  
48           releaseDate: "May 15, 2016",  
49           description: "Garden trowel with 36-inch wooden",  
50           price: 24.99,  
51           starRating: 4.5,  
52           imageUrls: "http://openclipart.org/image/300px_800  
53       };  
54       this.products.push(newProduct);  
55   }  
56  
57   deleteProduct(productId: number): void {  
58       const index = this.products.findIndex((product) =>  
59           product.productId === productId  
60       );  
61       if (index !== -1) {  
62           this.products.splice(index, 1);  
63       }  
64   }  
65  
66   updateProduct(productId: number, productName: string, productCode: string, releaseDate: string, description: string, price: number, starRating: number, imageUrls: string): void {  
67       const index = this.products.findIndex((product) =>  
68           product.productId === productId  
69       );  
70       if (index !== -1) {  
71           const updatedProduct: IProduct = {  
72               productId: productId,  
73               productName: productName,  
74               productCode: productCode,  
75               releaseDate: releaseDate,  
76               description: description,  
77               price: price,  
78               starRating: starRating,  
79               imageUrls: imageUrls  
80           };  
81           this.products[index] = updatedProduct;  
82       }  
83   }  
84  
85   sortProducts(sortOrder: string): void {  
86       this.products.sort((a, b) => {  
87           if (sortOrder === 'asc') {  
88               return a.productId - b.productId;  
89           } else {  
90               return b.productId - a.productId;  
91           }  
92       });  
93   }  
94  
95   filterProducts(filterValue: string): void {  
96       filterValue = filterValue.trim();  
97       filterValue = filterValue.toLowerCase();  
98       this.filteredProducts = this.products.filter((product) =>  
99           product.productName.toLowerCase().includes(filterValue)  
100      );  
101  }  
102  
```

把.js跟.map藏起來



```
"files.exclude": {  
    "**/.git": true,  
    "**/.DS_Store": true,  
    "**/*.js.map": true,  
    "**/*.js": {"when": "$(basename).ts"}  
}
```



Component Styles

```
import { Component, OnInit } from '@angular/core';
import { IProduct } from './product';

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

封裝component的style (@Component)

Encapsulating Component Styles

styles

```
@Component({
  selector: 'pm-products',
  templateUrl: 'app/products/product-list.component.html',
  styles: ['thead {color: #337AB7;}']})
```

styleUrls

```
@Component({
  selector: 'pm-products',
  templateUrl: 'app/products/product-list.component.html',
  styleUrls: ['app/products/product-list.component.css']})
```

幫header加上color

Show Image	Product	Code	Available
	Leaf Rake	gdn-0011	March 19, 2016

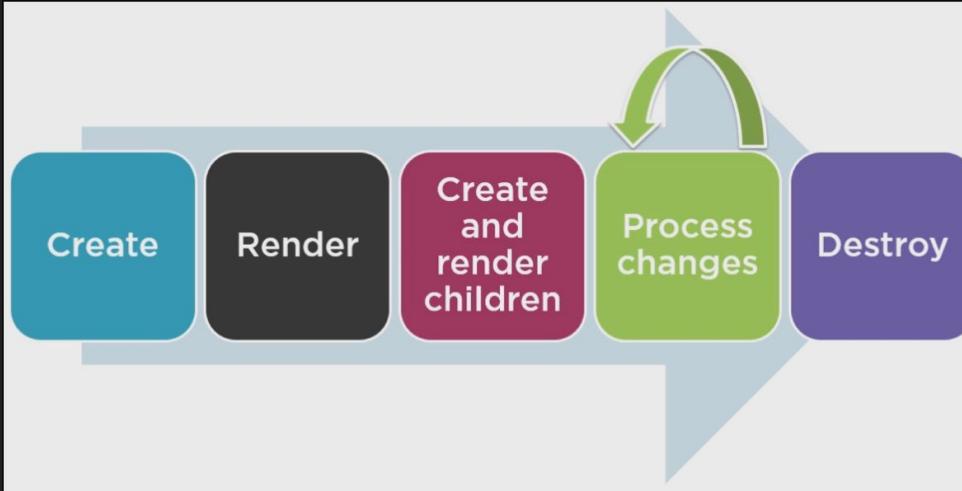
新增product-list css

可以直接寫thead{ }
因為style sheet有被封裝
所以不會影響其他部分的
component

```
product-list.component.html x ...
</div>
</div>
<div class='row'>
  <div class='col-md-6'>
    <h3>Filtered by: {{listFilter}}</h3>
  </div>
</div>
<div class='table-responsive'>
  <table class='table' *ngIf='products && products.l
    <thead>
      <tr>
        <th>
          <button class='btn btn-primary' (click)='t
            {{showImage ? 'Hide' : 'Show'}} Image
          </button>
        </th>
        <th>Product</th>
        <th>Code</th>
        <th>Available</th>
        <th>Price</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>Product 1</td>
        <td>P001</td>
        <td>In Stock</td>
        <td>$100</td>
      </tr>
      <tr>
        <td>2</td>
        <td>Product 2</td>
        <td>P002</td>
        <td>Out of Stock</td>
        <td>$200</td>
      </tr>
      <tr>
        <td>3</td>
        <td>Product 3</td>
        <td>P003</td>
        <td>In Stock</td>
        <td>$300</td>
      </tr>
      <tr>
        <td>4</td>
        <td>Product 4</td>
        <td>P004</td>
        <td>Out of Stock</td>
        <td>$400</td>
      </tr>
    </tbody>
  </table>
</div>
```

```
product-list.component.css x
1 thead{
2   color: #337AB7;
3 }
4 }
```

Component LifeCycle Hooks



component有生命週期(lifecycle), 由angular管理

- 建立
- 解析
- 建立及解析children
- 當data綁定的properties改變時要處理變動
- 把template從DOM移除前要destroy

看最常用的三個

OnInit:

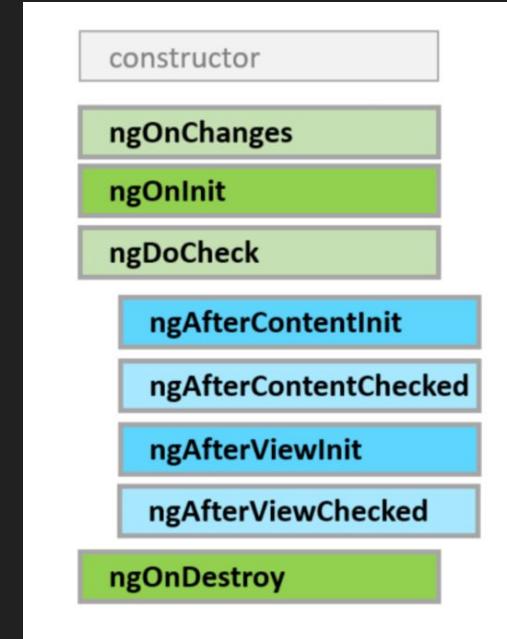
當angular初始化data-bound properties後
執行任何component初始化
=>適合從後端讀取資料給template

OnChange:

當data-bound input properties有改變時，可執行動作

OnDestroy:

destroy component前可進行清理動作



Angular有提供每個Lifecycle Hook對應的 interface供implement

```
2 import { Component, OnInit } from 'angular2/core';
1 export class ProductListComponent
    implements OnInit {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
  products: IProduct[] = [...];
3   ngOnInit(): void {
      console.log('In OnInit');
    }
}
```

每個對應的interface都只定義一個method, 名字為ngXXXX
如這邊的ngOnInit()

* LifeCycle interface不一定要實作, 因為transpile後會不見
除非有要在LifeCycle hook處理事情

EXPLORER

OPEN EDITORS

- app.component.ts src\app
- product-list.component.ts src\app\product...

APM

- e2e
- node_modules
- src
 - app
 - product-list
 - product-list.component.css
 - product-list.component.html
 - product-list.component.ts
 - shared
 - index.ts
 - app.component.css
 - app.component.html
 - app.component.ts
 - app.module.ts
 - index.ts
 - assets
 - environments
 - favicon.ico
 - index.html
 - main.ts
 - polyfills.ts
 - styles.css

app.component.ts product-list.component.ts x

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-product-list',
5   templateUrl: './product-list.component.html',
6   styleUrls: ['./product-list.component.css']
7 })
8 export class ProductListComponent implements OnInit {
9
10   constructor() { }
11
12
13
14   pageTitle:string = "Product List!";
15   imageWidth:number = 50;
16   imageMargin:number = 2;
17   products:any[]=[...]
18 ]
19
20   ngOnInit():void {
21     console.log('In OnInit');
22   }
23
24 }
```

2

1

3

驗證

Price	5 Star Rating
2016	19.95 3.2

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. A red arrow points from the text 'Angular is running in the development mode.' towards the word 'Angular'.

In OnInit [product-list.component](#)
Angular is running in the development mode. Call enableProdMode() to enable the prod mode.

Transforming Data with Pipes

需求：

使用者在input field輸入字串後

用來過濾product array

=>透過pipe實作

```
export class ProductFilterPipe
  implements PipeTransform {
  1
  transform(value: IProduct[],
  2   filterBy: string): IProduct[] {
  }
}
```

class implements PipeTransform interface

-只有一個 transform method

(1)要過濾的array

(2)使用者輸入的string

(3)回傳過濾後的array

加上@Pipe decorator，讓此class變成pipe

```
@Pipe({
  name: 'productFilter'
})
export class ProductFilterPipe
  implements PipeTransform {

  transform(value: IProduct[],
            args: string[]): IProduct[] {
  }
}
```

設定pipe的名稱：

name: productFilter是要在template中去使用

從@angular/core import

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'productFilter'
})
export class ProductFilterPipe
  implements PipeTransform {

  transform(value: IProduct[],
            filterBy: string): IProduct[] {
  }
}
```

是否與component很像？

angular提供一致性的style

使用piple

Template

```
<tr *ngFor ='let product of products | productFilter: listFilter'>
```

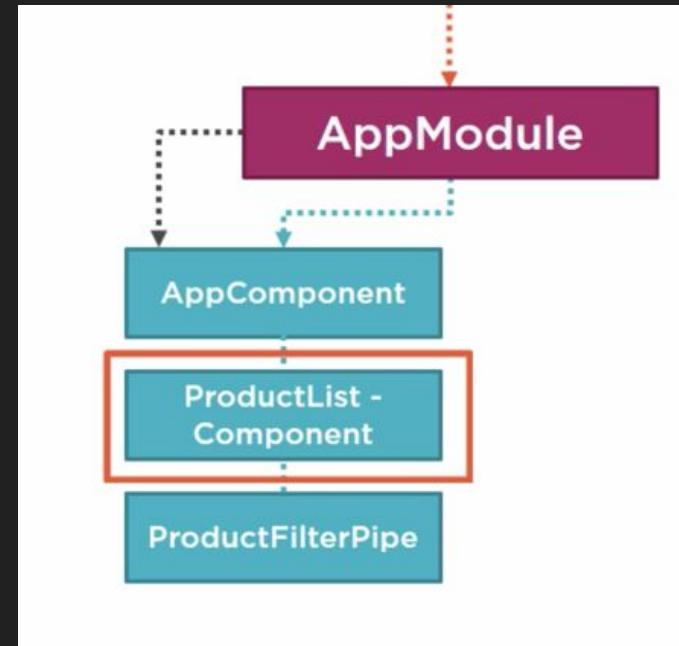
products會做為 productFiltrer中的transform method第一個參數
listFilter是transform method的第二個參數，對應user輸入的string

告訴angular去哪找這個pipe

因為是product list component要使用

其屬於AppModule

所以一樣去AppModule宣告



Module

```
@NgModule({  
    imports: [  
        BrowserModule,  
        FormsModule ],  
    declarations: [  
        AppComponent,  
        ProductListComponent,  
        ProductFilterPipe ],  
    bootstrap: [ AppComponent ]  
})  
export class AppModule { }
```

輸入ng g pipe product-filter

```
kuolundeMacBook-Pro:demo1211 kuolun$ ng g pipe product-filter
Could not start watchman; falling back to NodeWatcher for file
system events.
```

Visit <http://ember-cli.com/user-guide/#watchman> for more info.
installing pipe

```
create src/app/product-filter.pipe.spec.ts
create src/app/product-filter.pipe.ts
```

user輸入string後，用來過濾products array

```
product.ts          product-list.component.ts      product-filter.pipe.ts ×      product-list.componen
import { Pipe, PipeTransform } from '@angular/core';
import { IProduct } from './product';

@Pipe({
  name: 'productFilter'
})
export class ProductFilterPipe implements PipeTransform {
  transform(value: IProduct[], filterBy: string): IProduct[] {
    filterBy = filterBy ? filterBy.toLowerCase() : null;
    // 回傳那些符合搜尋條件的elements(新array)
    return filterBy ? value.filter((product:IProduct)=>
      product.productName.toLowerCase().indexOf(filterBy)!=-1):value
  }
}
```

```
transform(value: IProduct[], filterBy: string): IProduct[] {  
    filterBy = filterBy ? filterBy.toLocaleLowerCase() : null;  
    return filterBy ? value.filter((product: IProduct) =>  
        product.productName.toLocaleLowerCase().indexOf(filterBy) !== -1) : value;  
}
```

string.indexOf(searchString) 回傳第一個找到的位置

找不到則回傳-1

array.filter => 回傳符合callback fn內條件的那些elements組成的新array

```
the PRODUCTS -->
'panel-body'>
  s='row'>
    ass='col-md-2'>Filter by:</div>
    ass='col-md-4'>
      t type='text' [(ngModel)]=listFilter />
      
      s='row'>
        ass='col-md-6'>
          filtered by: {{listFilter}}</h3>

      s='table-responsive'>
        class='table' *ngIf='products && products.length'>
          d>
          >
          :th>
            <button class='btn btn-primary' (click)=toggleImage()>
              {{showImage ? 'Hide' : 'Show'}} Image
            </button>
          :/th>
          :th>Product</th>
          :th>Code</th>
          :th>Available</th>
          :th>Price</th>
          :th>5 Star Rating</th>
          ad>
          y>
            *ngFor='let product of products | productFilter:listFilter'>
              :td>
```

```
3
4
5 @Pipe({
6   name: 'productFilter'
7 })
8 export class ProductFilterPip
9
10  transform(value: IProduct[])
11    filterBy = filterBy ? filterBy : '';
12    return filterBy ? value.filter(item =>
13      item.productName.toLowerCase().includes(filterBy))
14    }
15
16 }
```

測試

Product List

Filter by:

cart

Filtered by: cart

Show Image

Product

Code

Garden Cart

gdn-0023

記得要到ngModuel import

```
import { ProductFilterPipe } from './product-filter.pipe';

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

用angular cli建立pipe時會自動幫我們加好

product-list.component.html

product-filter.pipe.ts x

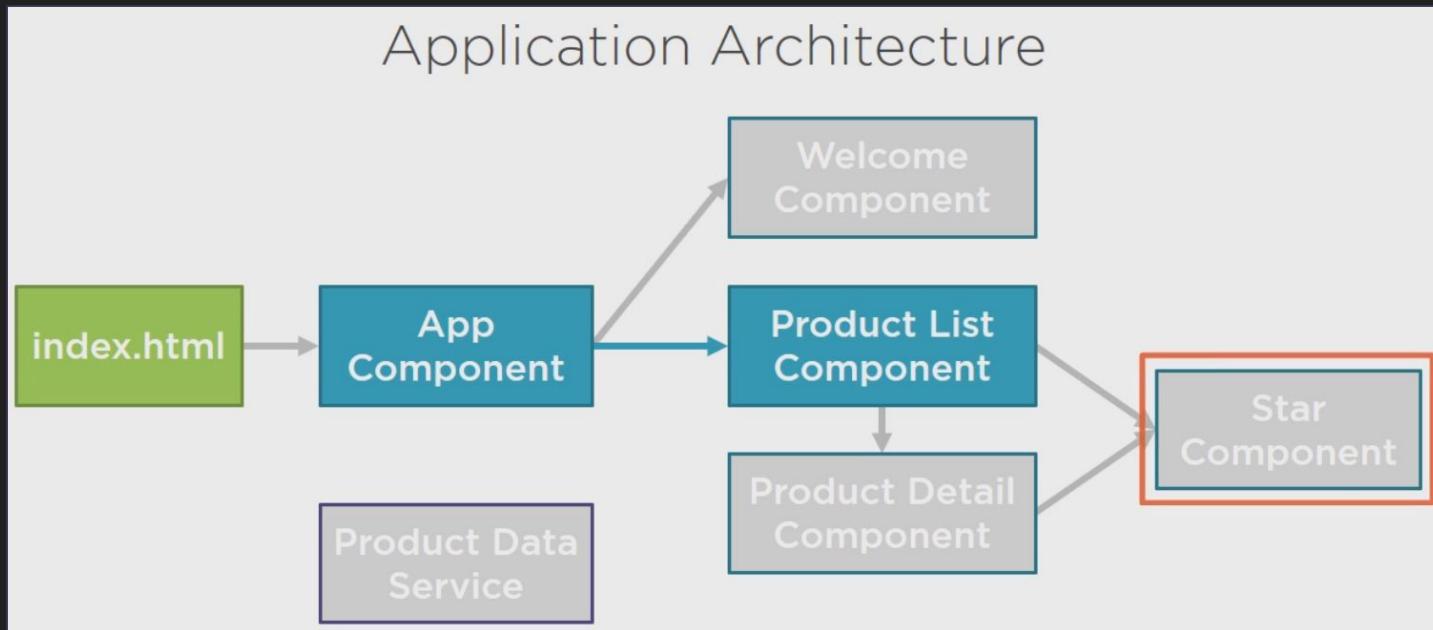
Extension: beautify

product



```
1 import { Pipe, PipeTransform } from '@angular/core';
2 import { IProduct } from './product';
3
4 @Pipe({
5   name: 'productFilter'
6 })
7 export class ProductFilterPipe implements PipeTransform {
8
9   transform(value: IProduct[], args: string): IProduct[] {
10    let filter:string = args?args.toLocaleLowerCase():null;
11    return filter ? value.filter((product:IProduct) =>
12      product.productName.toLocaleLowerCase().indexOf(filter) !== -1):value;
13  }
14
15 }
```

完成Product List Component



Service and Dependency Injection

Service

component很好用

但data/logic如果與view無關

或是我們想要跨component去分享時

要怎麼處理？

Service

有特定目的的class

有幾項特點：

- 獨立於其他component
- 提供共享的data/logic給其他component
- 封裝外部的互動(如取得資料)

程式容易test, debug, reuse

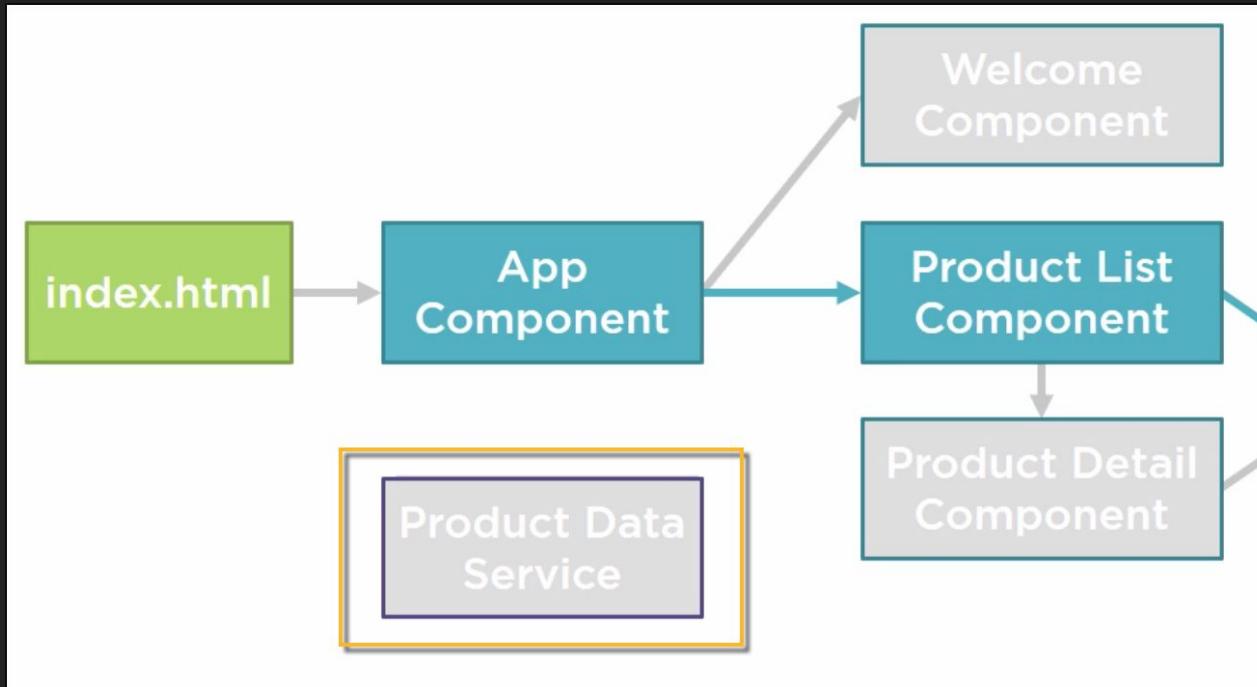
目標

要建立一個service

需要此service的component

要使用**dependency injection**

建立product data service



Service

```
export class myService {}
```

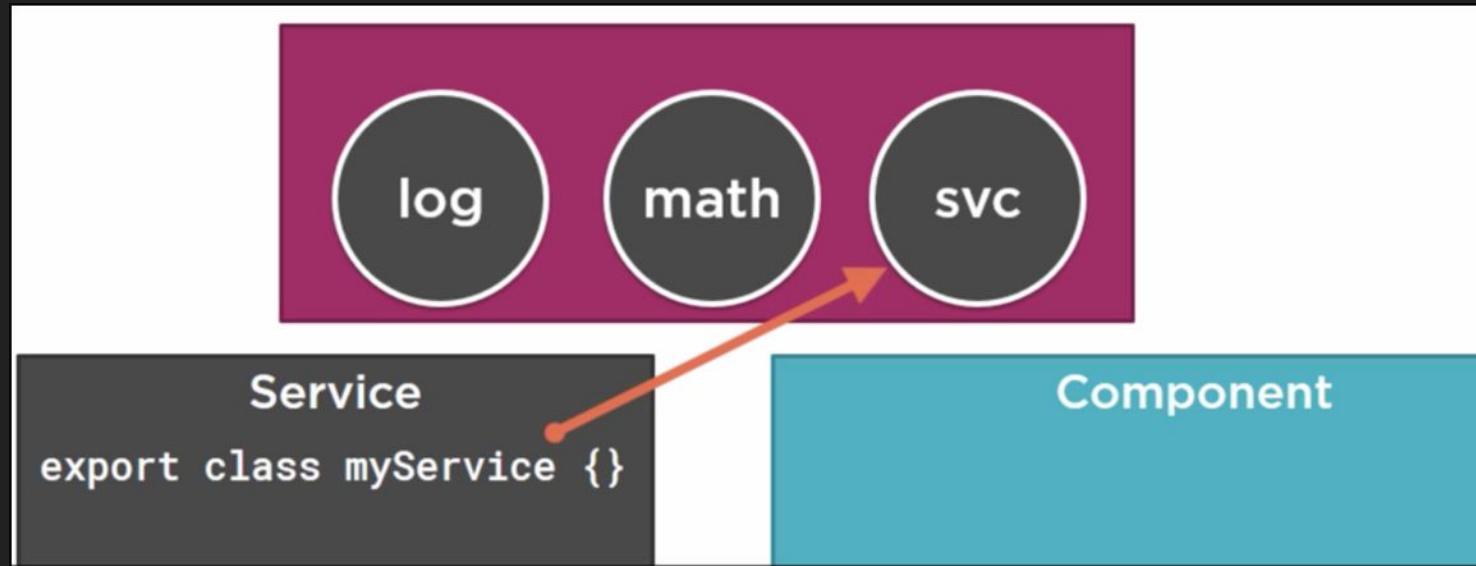
Component

```
let svc = new myService();
```



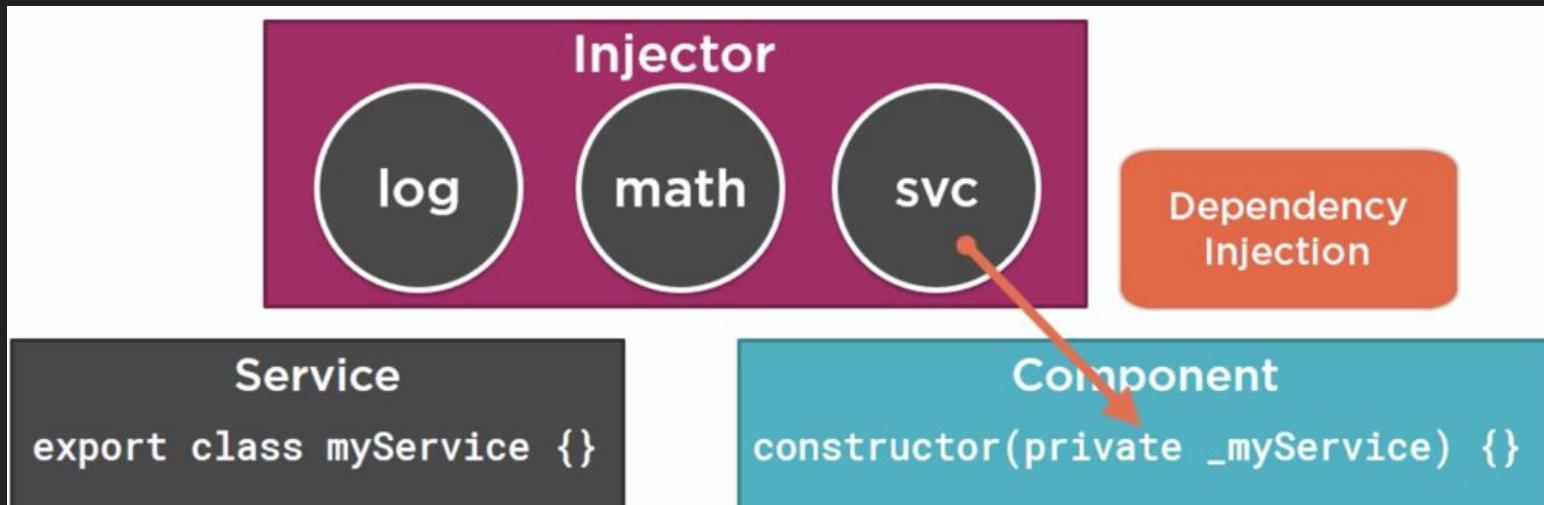
如果直接new一個service class
會拿到instance, 但是這會是local to the component

無法share data且難以測試



可以改向angular的injector註冊, angular會產生service的單一instance

稱為 singleton



Injector會管理這些instance，當Component有定義service做為dependency

在instantiate Component時，angular會inject service [即Dependency Injection]

因為由angular管理service instance，所以其內的data/logic可以分享給使用這個instance的 classes

Dependency Injection

一種coding pattern

class從外部來源接收所需的object instance
(稱為dependencies)

而不是自己建立

* 這邊外部來源是Angular Injector

建立一個Service

(1)建立service class

(2)用decorator定義metadata

(3)import

product.service.ts

```
import { Injectable } from '@angular/core'  
@Injectable()  
export class ProductService {  
    1  
    getProducts(): IProduct[] {  
        }  
    }  
}
```

- (1) export
- (2) @Injectable()
- (3) import

ng g service product

```
kuolundeMacBook-Pro:demo1211 kuolun$ ng g service product
Could not start watchman; falling back to NodeWatcher for file
system events.
Visit http://ember-cli.com/user-guide/#watchman for more info.
```

installing service

```
create src/app/product.service.spec.ts
```

```
create src/app/product.service.ts
```

幫service向injector註冊

向 provider 註冊

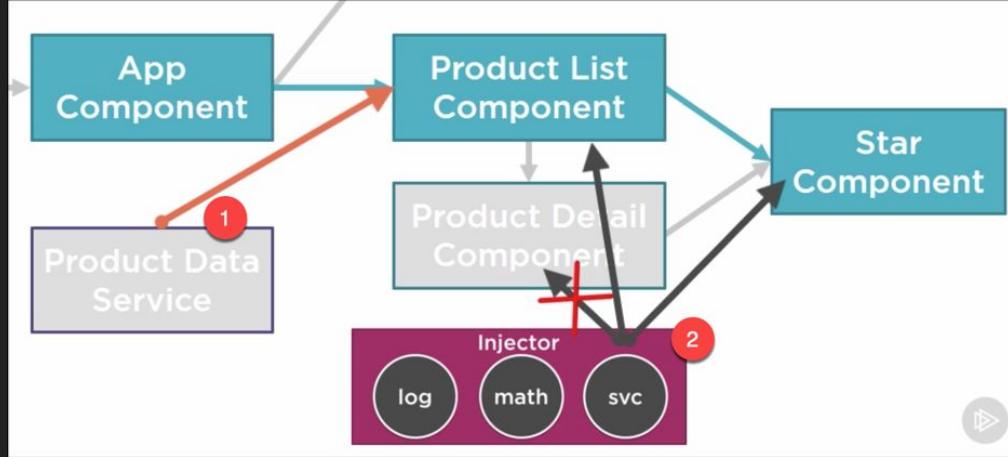
provider可以建立或回傳service

一般來說會是service class本身

定義在component內或Angular module metadata

註冊在component內 : 可以inject到component及其children

註冊在Angular module : 可以inject 到 app任何地方



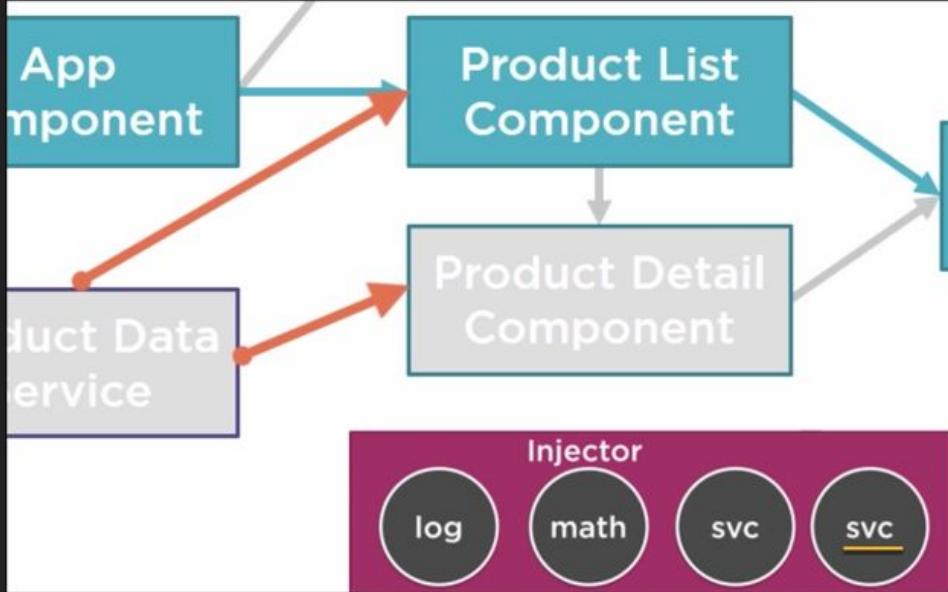
(1)如果service註冊在product list component

(2) 可以Inject到

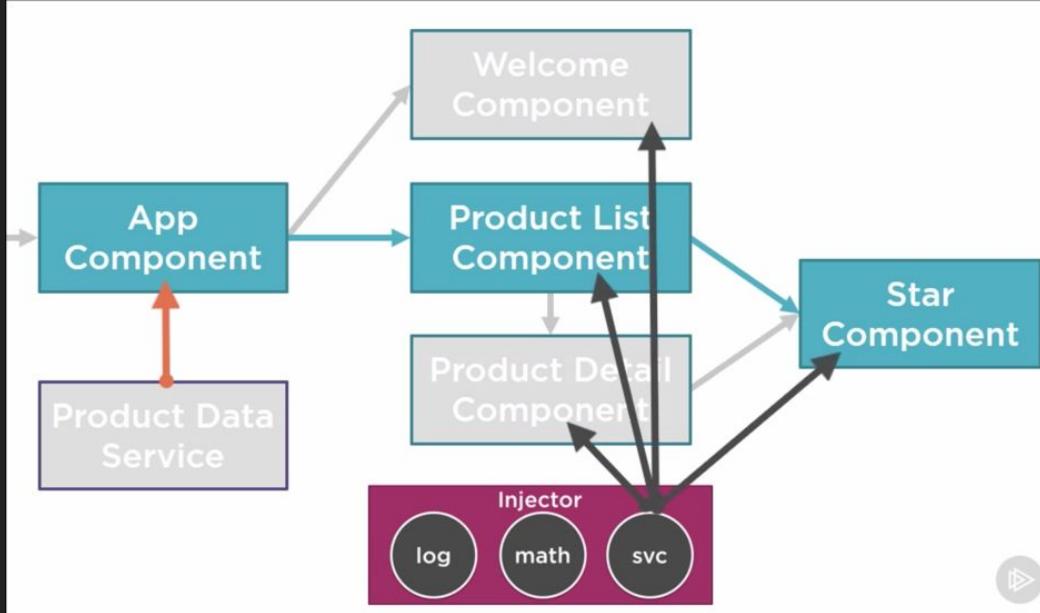
-product list

-star(children)

但不能inject到product detail (非children)



要讓service也可inject到product detail的話
可以用service也註冊到product detail
但這樣會產生兩個service instance, 就不會是singleton了 !



如果service註冊到root component (紅箭頭)

則instance可inject到任何component

以這個情況

我們要register到root component (AppComponent)

註冊provider

```
app.component.ts
...
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-app',
  template:
    <div><h1>{{pageTitle}}</h1>
      <pm-products></pm-products>
    </div>
    .
    .
    .
  providers: [ProductService]
})
export class AppComponent { }
```

在appComponent要註冊provider (用來建立service)

新增一個 **providers** property 為array

把要註冊的service放進去

(可註冊多個)

app.component.ts ×

```
1 import { Component } from '@angular/core';
2 import { ProductService } from './product.service';
3
4 @Component({
5   selector: 'app-root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css'],
8   providers: [ProductService]
9 })
10 export class AppComponent {
11   title = 'Angular2 is awesome!!';
12 }
```

在app.component.ts

import ProductService並新增一providers property

這樣service instance就可以inject到AppComponent及其child component了

Dependency Injection

Inject Service

constructor()

在class被建立實體時(component)
會執行

如果沒有定義
會自動執行隱含的constructor()

所以service可放在這邊去inject

* constructor是用來初始化 component

所以code越少越好，不要放會花時間的code

product-list.component.ts

```
@Component({
  selector: 'pm-products',
  templateUrl: 'product-list.component.html'
})
export class ProductListComponent {

  constructor() {
  }

}
```

```
templateUrl: 'product-list.component.html'
})
export class ProductListComponent {
private _productService;
constructor(productService: ProductService) {
  _productService = productService;
}
}
```

(1)定義 private variable

(2)constructor fn會接收productService參數

當constructor被執行時, injector會把參數(productService)設定為
要求的Service instance(ProductService的instance)

(3)把injected service instance(productService)指派給
local variable(_productServire)

就可在class內使用_productServire去存取service的properties/method

比較簡潔的寫法

```
export class ProductListComponent {  
  constructor(private _productService: ProductService) {}  
}
```

(Demo) product-list.component.ts

product list component不需寫provider

因為service已經註冊在app component (parent)

```
constructor(private _productService: ProductService) {  
}
```

所以只需要constructor fn (通常寫在所有prop之後)

```
import { IProduct } from './product';  
import { ProductService } from './product.service';
```

(1)inject product service

(2)import service

(3)使用service的method取資料

(4)把hard code的product拿掉

(5)把filter條件拿掉

* 抓資料的動作通常不會放在
constructor, 所以改放到
ngOnInit()

```
component.ts          product-list.component.ts x      app.module.ts
import { Component, OnInit } from '@angular/core';
import { IProduct } from './product';
import { ProductService } from '../product.service';  2

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {

  pageTitle: string = "Product List";
  imageWidth: number = 50;
  imageMargin: number = 2;
  showImage: boolean = false;
  listFilter: string; 5
  products: IProduct[]; 4

  toggleImage(): void {
    // alert('click');
    this.showImage = !this.showImage;
  }

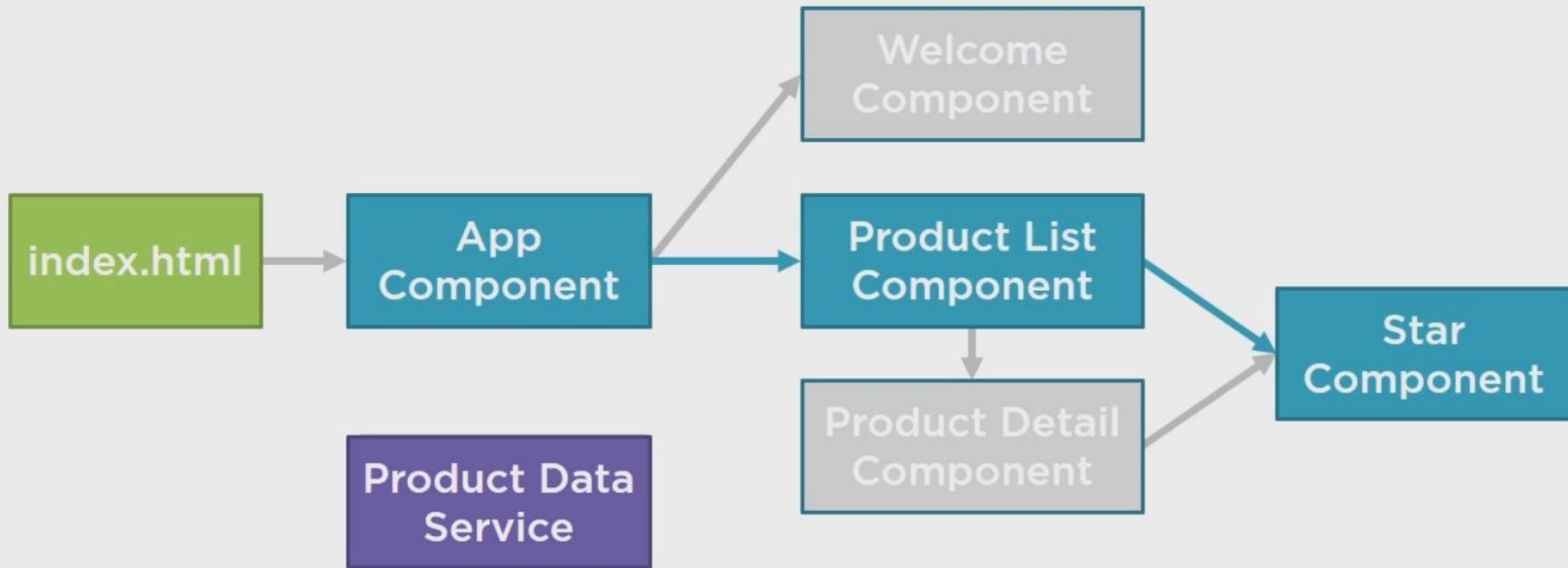
  constructor(private _productservice: ProductService) { }

  ngOnInit() {
    console.log('OnInit!');
    this.products = this._productservice.getProducts(); 3
  }
}
```

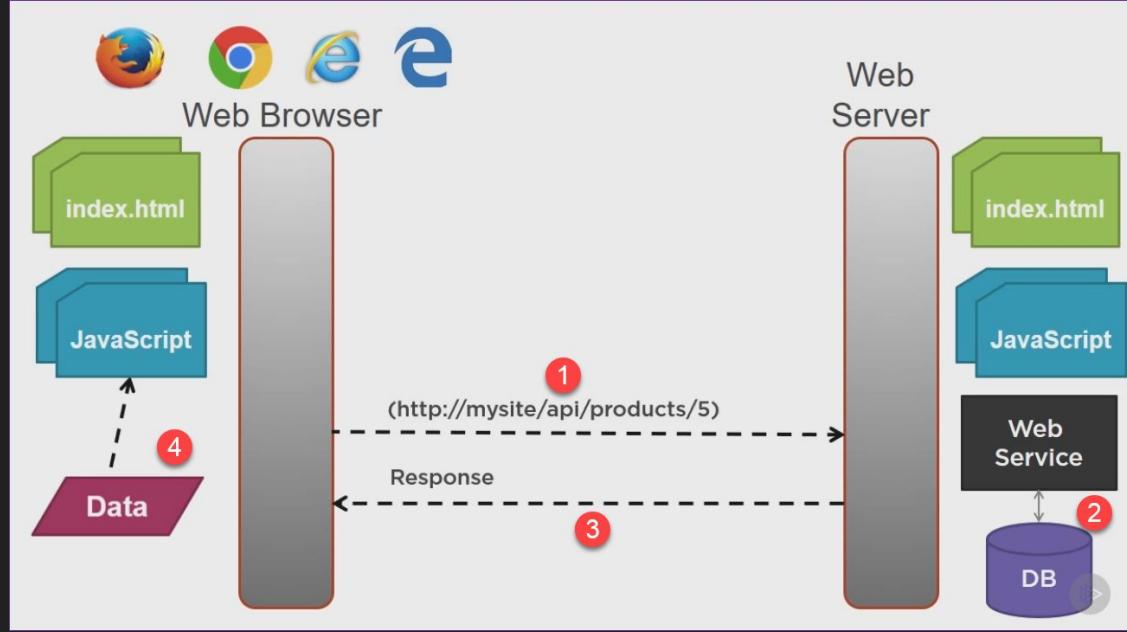
資料還是存在

Show Image	Product	Code
	Leaf Rake	gdn-0011
	Garden Cart	gdn-0023
	Hammer	tbx-0048
	Saw	tbx-0022
	Video Game Controller	gmg-0042

Application Architecture



Retrieving Data using HTTP



- (1) application 發出 get HTTP request 到 Web Service
- (2) 從 DB 取出資料後
- (3) 透過 HTTP response 回傳資料給 application
- (4) application 處理 data

Observable



Help manage asynchronous data

Treat events as a collection

- An array whose items arrive asynchronously over time

Are a proposed feature for ES 2016

Use Reactive Extensions (RxJS)

Are used within Angular

(1)幫助處理非同步資料

(2)把events視為collection

- array的item隨著時間非同步的抵達

(3)是ES7的功能

(4)使用RxJS

(5)Angular內部有使用(含ag的event system跟http client service)

Observable Operators

Methods on observables that compose new observables

Transform the source observable in some way

Process each value as it is emitted

Examples: map, filter, take, merge, ...

- (1) Operator就是在observable上的method, 用來產生新的observable
- (2) 把來源observable用某種方式轉換
- (3) 當value被emit時才處理
- (4) 如 map/ filter/ take /merge

Data sequence有很多形式

- backend service的response
- 一串系統的notifications
- 一串events (如user input)

RxJS把data sequence稱為observable sequence (簡稱 an observable)

下面圖包含三個elements

Interactive diagrams of Rx Observables



Observables

我們code裡面的method

可以訂閱(subscribe) observable

當有data非同步抵達時

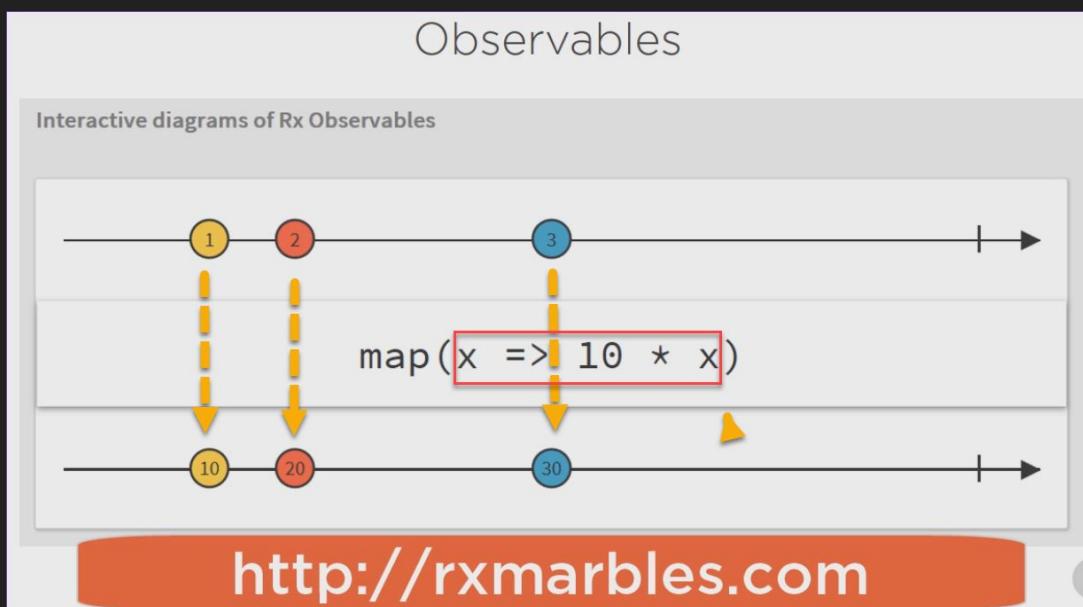
我們就會收到通知

method就可以對push給它的

data做出反應

(沒有更多data/或發生error時也會收到通知)

operator會對data item做出轉換(如把x都乘以10) 上圖稱為**marble diagrams**



Promise與Observable比較

Lazy: observable只有在被subscribe時才會emit

* 呼叫http時也可用promise(但這邊會用observable)

Promise	Observable
Provides a single future value	Emits multiple values over time
Not lazy	Lazy
Not cancellable	Cancellable
	Supports map, filter, reduce and similar operators

讓service從後端Web Server取資料

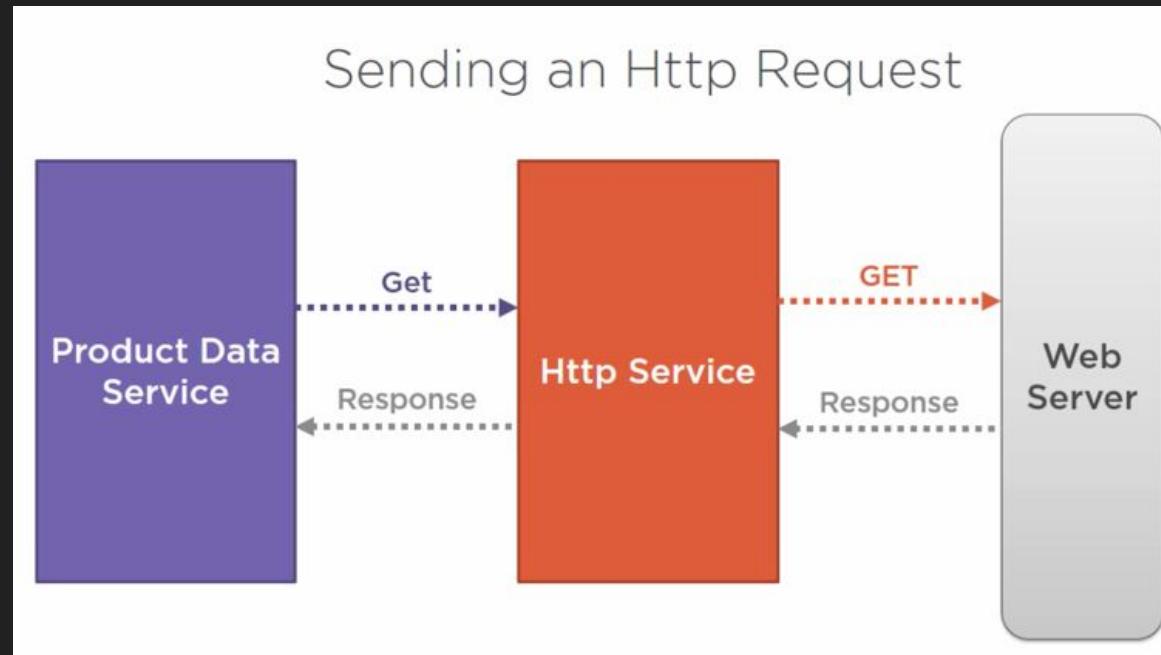
angular提供

HTTP service讓我們可以
跟後端server溝通

使用HTTP request/response
protocol

呼叫HTTP service的get method
會接著送出get request到
web server

web server response會被HTTP Service
以Observable形式回傳給product data service



送出HTTP Request

_productId

是要送出request的地方
(web server)

constructor

用來inject dependency
(必須先註冊provider)

inject Angular Http service

(建立_productId變數，並把Http service instance指派給它)

this._productId

使用_productId指向

傳進來service的
instance

product.service.ts

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

@Injectable()
export class ProductService {
  private _productId = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts() {
    return this._http.get(this._productId);
  }
}
```

使用Http需要import HttpModule

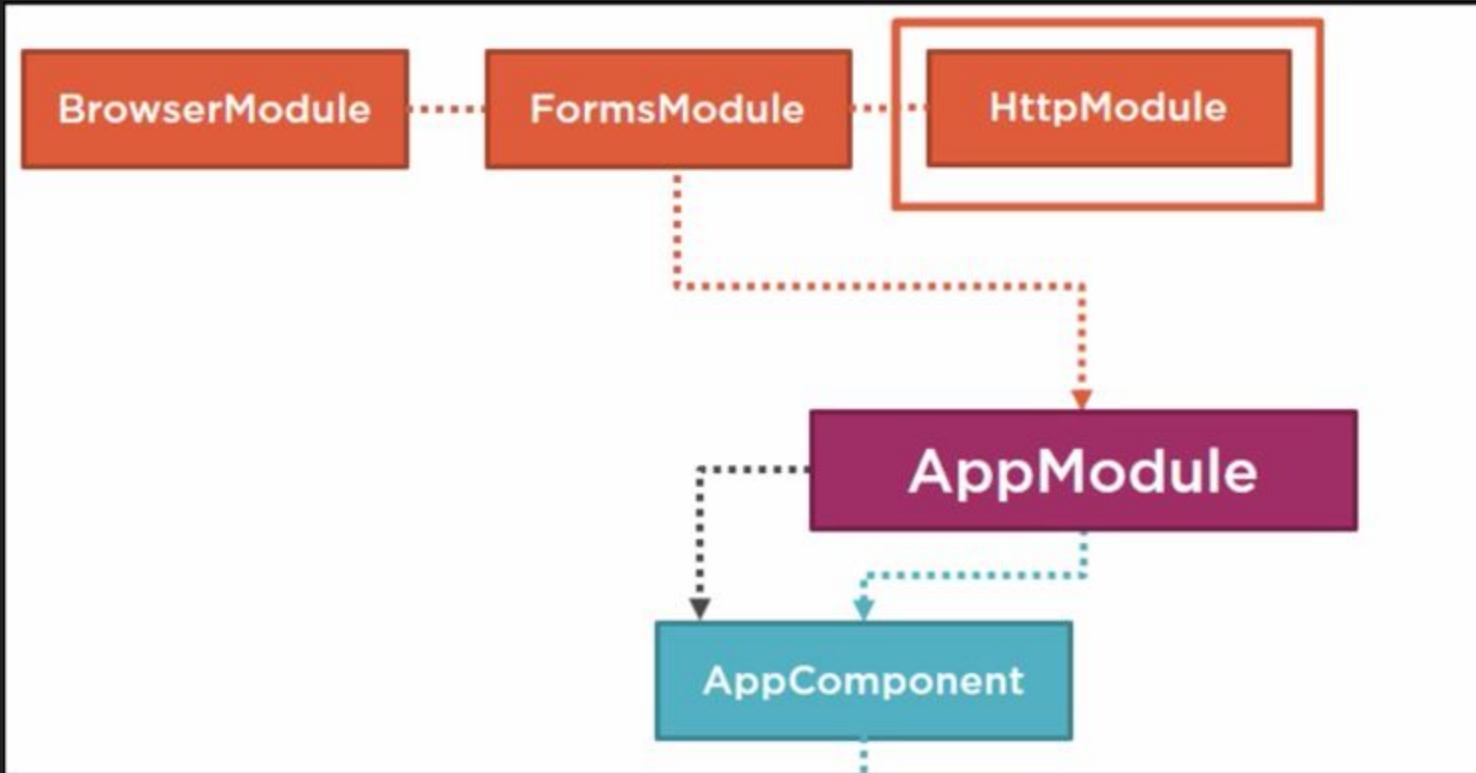
imports array加入
外部module
HttpModule

*imports array放的
是AppModule會用到
的外部module

app.module.ts

```
...
import { HttpModule } from '@angular/http';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule ],
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductFilterPipe,
    StarComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```



```
...  
import { Http, Response } from '@angular/http';  
import { Observable } from 'rxjs/Observable';  
  
private _productUrl = 'www.myWebService.com'  
  
constructor(private _http: Http) { }  
  
getProducts(): Observable<Response> {  
  return this._http.get(this._productUrl);  
}
```

getProducts()的回傳type為

Observable<Response>

(因為 _http.get 會得到 Observable, data type 為 HTTP response)

這邊的generic

定義 Observable 觀察 (Observing) 的 Observable sequence 中的 data type

這個 case 是 HTTP Response

* HTTP call 是一個 single async operation

(代表 observable sequence 只會含有一個 element: HTTP response object)

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

要import { Response } (從@angular/http)

及

Observable (從rxjs/Observable)

但getProducts()要得的不是http response

Product-list component是期望拿到list of products
=>轉換response obj成為array of products

product.service.ts

```
...
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map'; ①

@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts(): Observable<IProduct[]> { ②
    return this._http.get(this._productUrl)
      .map((response: Response) => <IProduct[]>response.json()); ③④
  }
}
```

但我們期望拿到的是 array資料
所以要用 map operator 來進行轉換

(1) import map (這邊是執行 JS library 去載入 map, 並沒真的 import 進來)

(3) _http.get 會拿到 observable, .map 把 emit 出的 http response obj 用 .json() 轉成 json object

(4) 再用 casting <IProduct[]> 把 json object 轉成 an array of products

*如果資料包在 data property 中, 則用 response.json().data 取得

.map 會拿到另一個 observable

(2) 這時回傳的資料就可改寫是 IProduct array => Observable<IProduct[]>

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
```

這種寫法的import 跟平常不太一樣

告訴module loader去載入library但是沒有import任何東西進來

載入library會執行js

這邊的js會載入map operator

step1 import HttpClientModule

(Recall)

Angular在HttpClient
內註冊其http service provider

現在可以inject HttpClient

到任何需要它的class

```
app.module.ts x product.service.ts
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-
3  import { FormsModule } from '@angular/forms';
4  import { HttpClientModule } from '@angular/http';
5
```

```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  declarations: [
    AppComponent,
    ProductListComponent,
```

step2- import Http , Response, Observable, map

import

@angular/http

Http - http client service

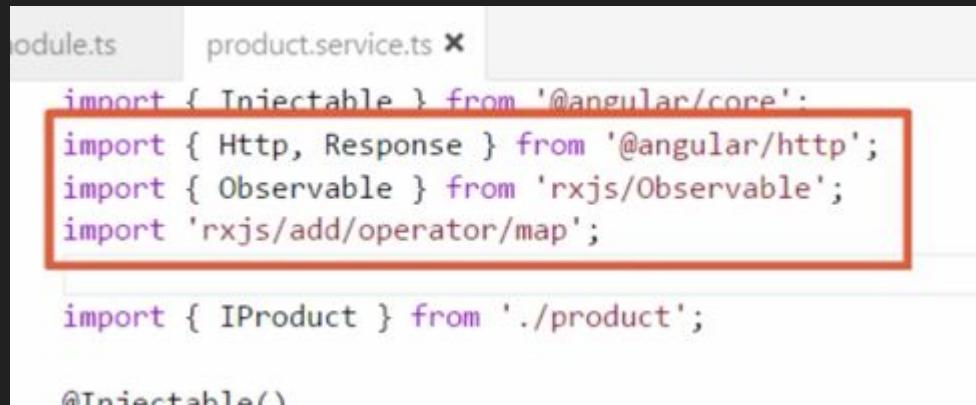
Response -http response

rxjs/Observable

Observable

map operator

利用特別語法載入operator, 但沒有import任何東西



```
module.ts      product.service.ts x
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';

import { IProduct } from './product';

@Injectable()
```

step3 - 建立constructor來取得http client service

我們需要http client service instance

所以透過inject來取得

```
@Injectable()
export class ProductService {
  private _productUrl = 'api/products/products.json';

  constructor(private _http: Http) {}

  getProducts(): IProduct[] {
```

這邊連到自己local的json (因為這樣不用建server)
如果要連web server, 就改這邊的URL

step4 - 使用operator轉換response object

但這邊寫完還不能測試，因為還沒有subscribe這個Observable

```
@Injectable()
export class ProductService {
  private _productUrl = 'api/products/products.json';

  constructor(private _http: Http){}

  getProducts(): Observable<IProduct[]> {
    return this._http.get(this._productUrl)
      .map((response: Response) => <IProduct[]> response.json());
  }
}
```

Exception handling: 可能有invalid request / lost connection

```
product.service.ts
...
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/catch';
...

getProducts(): Observable<IProduct[]> {
  return this._http.get(this._productUrl)
    .map((response: Response) => <IProduct[]>response.json())
    .do(data => console.log('All: ' + JSON.stringify(data)))
    .catch(this.handleError);
}

private handleError(error: Response) {
```

- (1) import do跟catch operator
- (2) do可以看傳回來的data =>可用來debug
- (3) .catch可以傳入錯誤處理fn, 有錯誤時, 會傳入error response object

handleError method

- (1)把error印出
- (2)回傳錯誤給呼叫的人

```
private handleError(error: Response) {  
    console.error(error);  
    return Observable.throw(error.json().error || 'Server error');  
}
```

do method

```
getProducts(): Observable<IProduct[]> {
    return this._http.get(this._productUrl)
        .map((response: Response) => <IProduct[]> response.json())
        .do(data => console.log('All: ' + JSON.stringify(data)))
        .catch(this.handleError);
}
```

可窺看response data

並用JSON.stringify美化

訂閱 Observable

訂閱(subscribe) Observable

```
x.then(valueFn, errorFn)          //Promise  
x.subscribe(valueFn, errorFn)      //Observable  
x.subscribe(valueFn, errorFn, completeFn) //Observable  
let sub = x.subscribe(valueFn, errorFn, completeFn)
```

第1行x是promise, 透過then可取得x的結果, 2個參數函數分別處理完成及錯誤情況

第2行x是observable, 只把then改成subscribe

因為observable隨時間處理多個value, 每當value emit時, 就會呼叫valueFn
有第3個optional completeFn, Observable完成時會呼叫

subscribe回傳的是subscription obeject, 可用來cancel subscribe (unsubscribe)

product-list.component.ts

```
ngOnInit(): void {
    this._productService.getProducts()
        .subscribe(
            products => this.products = products,
            error => this.errorMessage = <any>error);
}
```

在product-list component內inject productService

並呼叫其getProducts()取得Observable

因為

--Observable是lazy的，直到subscribe被呼叫前不會開始emit value

這邊會透過.subscribe觸發http get request並拿到回傳的array of products (非同步)

products就是emit出來的東西

因為http call是single async operation, 只會有一個item被emit(即http response obj)

如果observable fail, 第2個參數的fn會被呼叫處理error

(第3個completeFn不太使用在http request的狀況，

因為emit single response之後就會自動結束(complete))

新增errorMessage

```
export class ProductListComponent implements OnInit {
  pageTitle: string = 'Product List';
  imageWidth: number = 50;
  imageMargin: number = 2;
  showImage: boolean = false;
  listFilter: string;
  errorMessage: string;

  products: IProduct[];

  constructor(private _productService: ProductService) {}

  toggleImage(): void {
    this.showImage = !this.showImage;
  }

  ngOnInit(): void {
    this._productService.getProducts()
      .subscribe(products => this.products = products,
                error => this.errorMessage = <any>error);
  }

  onRatingClicked(message: string): void {
    this.pageTitle = 'Product List: ' + message;
  }
}
```

<any>errorMessage代表

把observable回傳的error
轉成<any> data type

結論: 建議使用HTTP的方式

- 封裝在service內
- 透過observable提供給需要呼叫的class
- class僅需透過subscribe就可獲取data

(補充) Observable vs promise

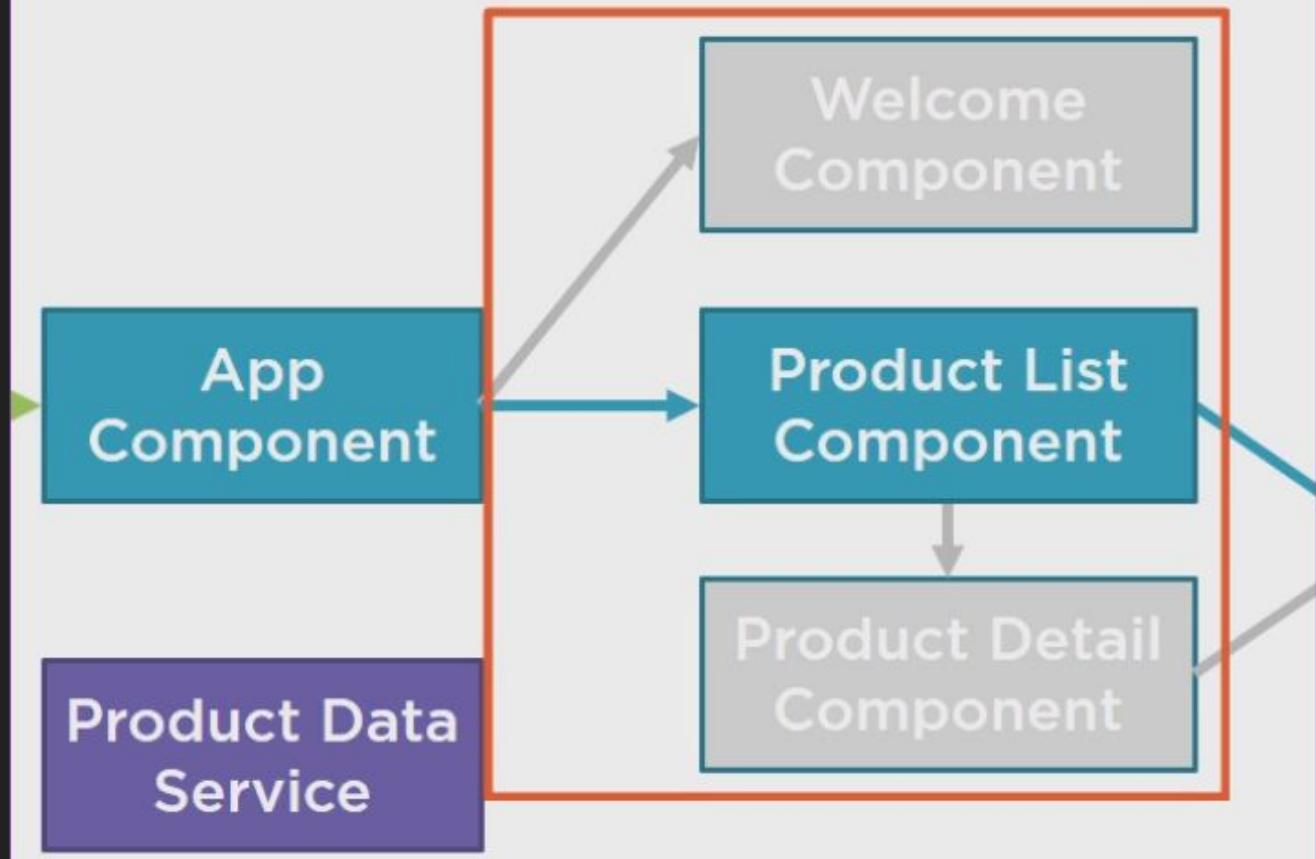
<https://angular.io/docs/ts/latest/guide/server-communication.html>

Angular http client回傳 Observable<Response>
但可轉成 Observable<Response>

* promise也許比較熟悉，但 Observable有許多優勢

Navigation and Routing Basics

Application Architecture



Route如何運作

1.替每個component設定route

2.定義options/actions

3.把每個option/action綁定route

4.根據user action啟動route

啟動route會顯示出component的view\

*option是選項的意思

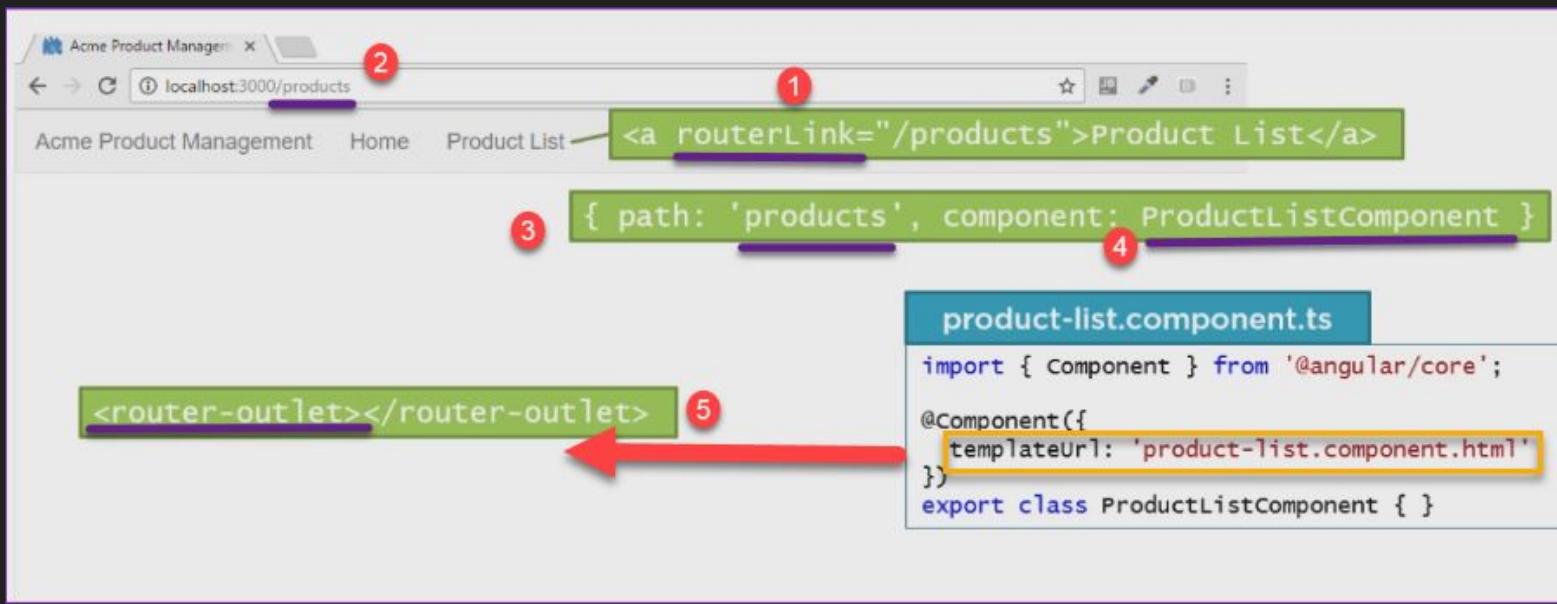
Configure a route for each component

Define options/actions

Tie a route to each option/action

Activate the route based on user action

Activating a route displays the
component's view



(1) 把menu option綁定內建的router directive: routerLink

當user點了這個option時, angular router會導向/products route

(2) 此時URL會改變 (沒有#的需要server設定URL rewriting)

* Angular預設使用HTML5 style URLs, 不會有#來指出local navigation

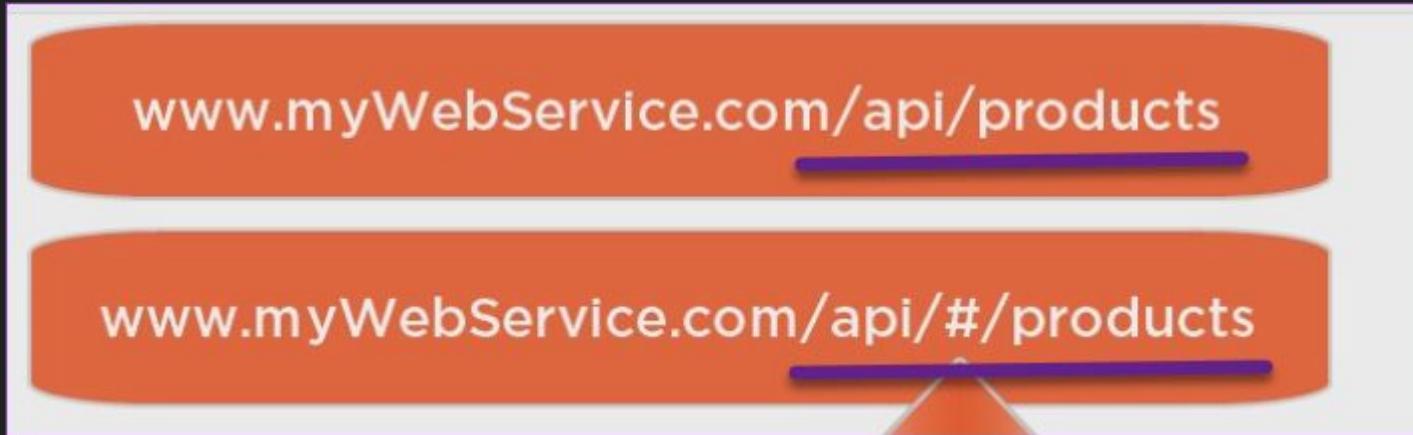
* 也支援#

(3) 當URL改變, angular router會找**route definition**, path 對應到products時 (/products)

(4) component對應會載入的ProductListComponent之template

(5) 載入template的點為 <router-outlet></router-outlet>

HTML5 style URLs

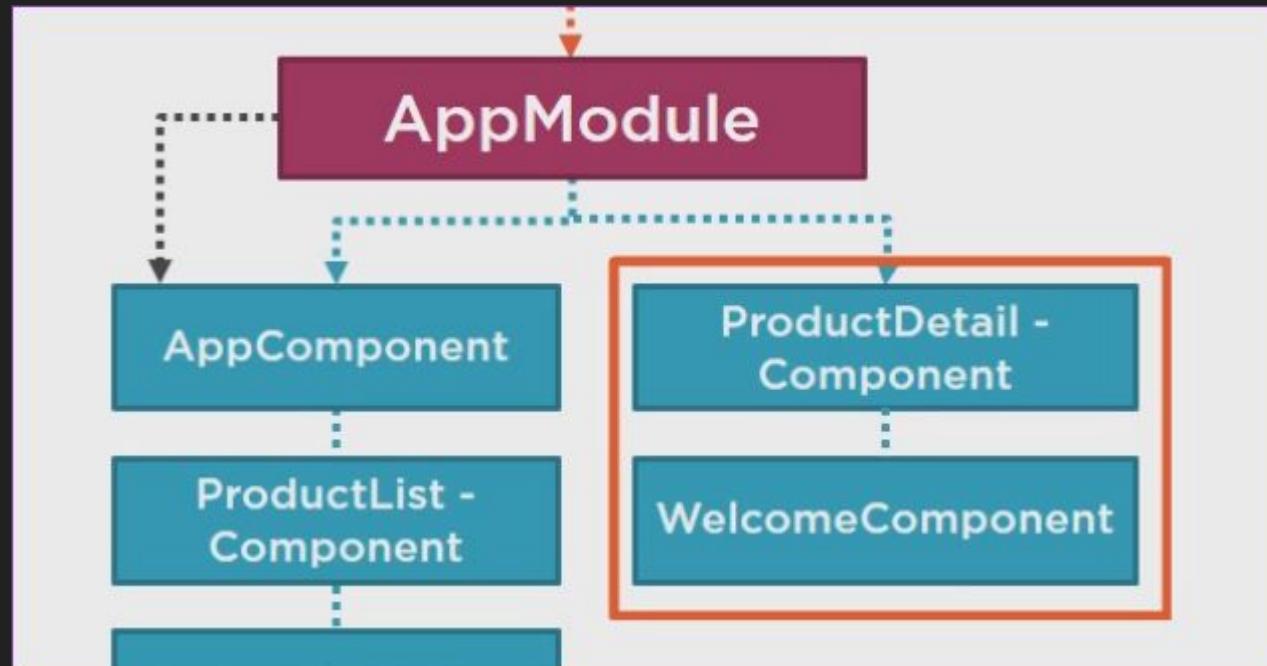


要做HTML 5 style URLs

必須要設定web server去執行URL rewriting

check web server for URL rewriting

每次新增component都要加到 AppModule上



Product Detail: Video Game Controller

Name: Video Game Controller
Code: GMG-0042
Description: Standard two-button video game controller
Availability: October 15, 2015
Price: \$35.95
5 Star Rating: ★★★★☆



◀ Back

新增product-detail.component.ts

The screenshot shows the Visual Studio Code interface with the following details:

- View Bar:** View, Go, Help.
- EXPLORER:** Shows the project structure:
 - OPEN EDITORS
 - APM
 - .vscode
 - api
 - app
 - assets
 - home
 - products
 - product.ts
 - product-detail.component.htm
 - product-detail.component.ts
 - product-filter.pipe.ts
 - product-list.component.css
 - product-list.component.html
- product-detail.component.ts Editor:** Contains the following code:

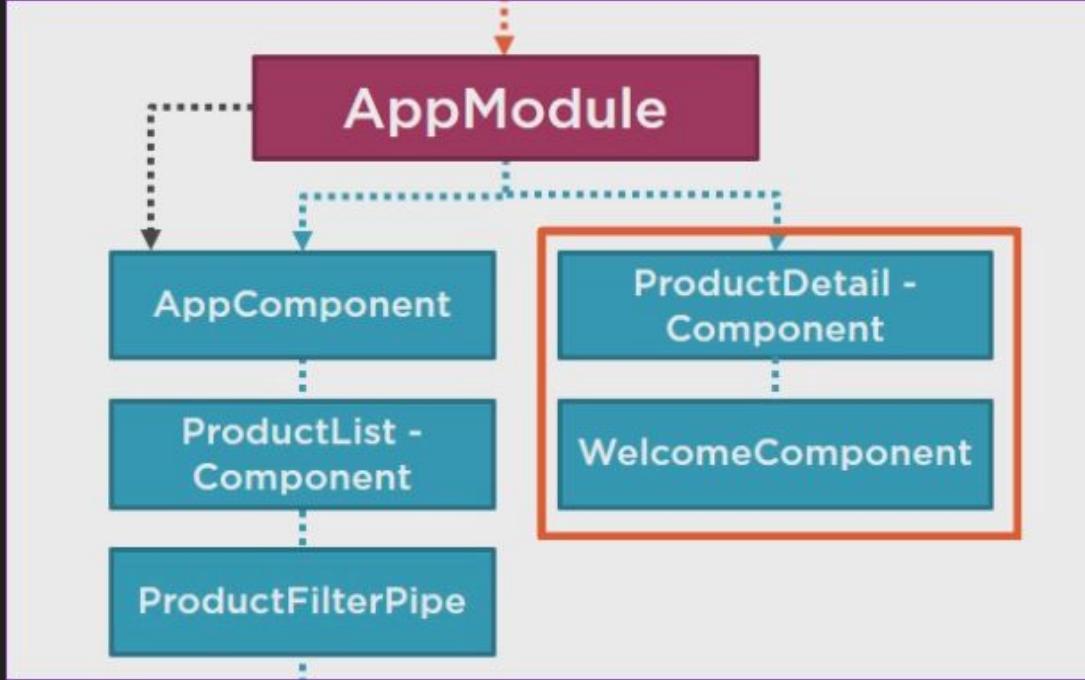
```
1 <div class='panel panel-primary'>
2   <div class='panel-heading'>
3     {{pageTitle}}
4   </div>
5 </div>
6 |
```
- product-detail.component.html Editor:** Currently empty.

product-detail.component.ts ✘ product-detail.component.html

```
1 import { Component } from '@angular/core';
2
3 import { IProduct } from './product';
4
5 @Component({
6   templateUrl: 'app/products/product-detail.component.html'
7 })
8 export class ProductDetailComponent {
9   pageTitle: string = 'Product Detail';
10  product: IProduct;
11 }
12
```

這邊沒有設定selector

因為那是要用在nest 這個compoent時才會用到



有新增component時，要加到Module

app.module

加入

WelcomeComponent

與

ProductDetailComponent

```
product-detail.component.ts | product-detail.component.html | app.module.ts x
```

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { FormsModule } from '@angular/forms';
4 import { HttpClientModule } from '@angular/http';
5
6 import { AppComponent } from './app.component';
7 import { WelcomeComponent } from './home/welcome.component';   
8
9 import { ProductListComponent } from './products/product-list.component';
10 import { ProductDetailComponent } from './products/product-detail.component';   
11 import { ProductFilterPipe } from './products/product-filter.pipe';
12 import { StarComponent } from './shared/star.component';
13
14 @NgModule({
15   imports: [
16     BrowserModule,
17     FormsModule,
18     HttpClientModule
19   ],
20   declarations: [
21     AppComponent,
22     WelcomeComponent,   
23     ProductListComponent,   
24     ProductDetailComponent,   
25     ProductFilterPipe,
26     StarComponent
27   ],
28   providers: []
29 })
30 export class AppModule { }
```

Router Service

Angular app會有一個router由 Angular's router **service**管理
使用service前，需要先在Angular Module註冊service provider

app.module.ts

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

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule
  ],
  declarations: [
    ...
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

- 1 Registers the router service
- 2 Declares the router directives
- 3 Exposes configured routes

在app.module.ts 中 import **RouterModule**

代表

- (1)註冊了router service
- (2)宣告了router directives
- (3)曝光設定過的route

如何讓routes可被App取用？

```
@NgModule({  
    imports: [  
        BrowserModule,  
        FormsModule,  
        HttpModule,  
        RouterModule.forRoot([])  
    ],  
    declarations: [
```

在navigate routes之前，必須確認routes是可被App取用

=>透過**RouterModule.forRoot()** 把 routes用array傳入

在App root建立 routes

如果要用URL中有帶#的，需加上設定

```
HttpModule,  
RouterModule.forRoot([], { useHash: true })  
],  
declarations: [
```

```
[  
 ①{ path: 'products', component: ProductListComponent },  
 ②{ path: 'product/:id', component: ProductDetailComponent },  
 ③{ path: 'welcome', component: WelcomeComponent },  
 ④{ path: '', redirectTo: 'welcome', pathMatch: 'full' },  
 ⑤{ path: '**', component: PageNotFoundComponent }  
]
```

設定routes

1. 對應到 /products
2. 對應 /product/5 類型, 會使用id
3. 對應/welcome
4. 當沒有東西對應時, **redirectTo**到welcom path
5. wildcard path-- 當1~4都不match時, 會被5捕捉到(適合404 page或redirect)

* 注意path前面不加上/ , routes的順序很重要, 先碰到的先套用
表示明確的route要放前面

Empty Path

(補)

Empty-path route configurations can be used to instantiate components that do not 'consume' any url segments. Let's look at the following configuration:

```
1.  [{  
2.    path: 'team/:id',  
3.    component: Team,  
4.    children: [  
5.      {  
6.        path: '',  
7.        component: AllUsers  
8.      },  
9.      {  
10.        path: 'user/:name',  
11.        component: User  
12.      }  
13.    ]  
14.  }]
```

team/11

team/11/user/jason

When navigating to `/team/11`, the router will instantiate the AllUsers component.

Empty path可有children

當導航到
team/11/user/jason時
會實體化一個
WrapperCmp component
(內包User component)

* empty path會繼承
parent的params跟data
(因為其無法有自己的
params)

The diagram illustrates a nested component structure. A red box highlights the first level of children under the 'Team' component. An orange box highlights the second level of children under the 'WrapperCmp' component. A red arrow points from the text 'team/11/user/jason' to the 'WrapperCmp' component, indicating the specific path being matched.

```
1.   [
2.     {
3.       path: 'team/:id',
4.       component: Team,
5.       children: [
6.         {
7.           path: '',
8.           component: WrapperCmp,
9.           children: [
10.             {
11.               path: 'user/:name',
12.               component: User
13.             }
14.           ]
15.         }
16.       ]
17.     }
18.   ]
```

team/11/user/jason

(補)Matching Strategy

Matching Strategy

By default the router will look at what is left in the url and check if it starts with the specified path (e.g., `/team/11/user` starts with `team/:id`).

We can change the matching strategy to make sure that the path covers the whole unconsumed url, which is akin to `unconsumedUrl === path` or `\$` regular expressions.

This is particularly important when redirecting empty-path routes.

router會看去check剩下的url部分是否是特定path起頭
如`:/team/11/user`是從`team/:id`起頭

[COPY CODE](#)

```
1.  [{  
2.    path: '',  
3.    pathMatch: 'prefix', //default  
4.    redirectTo: 'main'  
5.  },  
6.  {  
7.    path: 'main',  
8.    component: Main  
9.  }]
```

Since an empty path is a prefix of any url, even when navigating to '/main', the router will still apply the redirect.

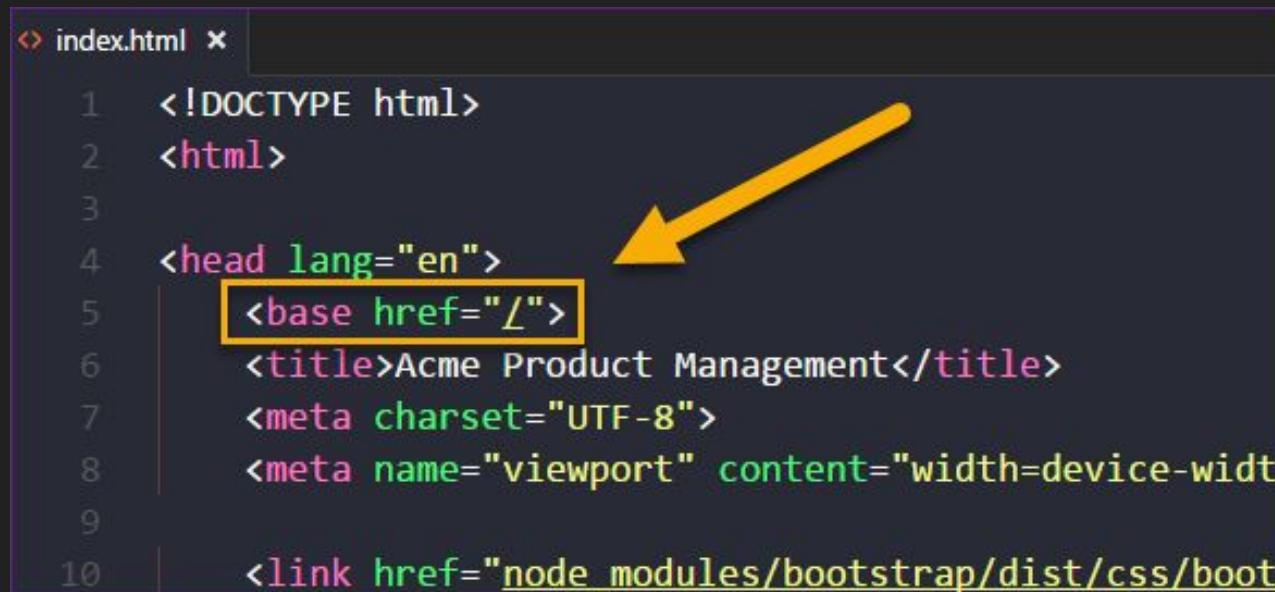
pathMatch:'prefix' 加上path: '' 時，表示empty path作為任何url的前綴

所以當導航到 /main 時，router 也會仍然套用 redirect (???)

step1:在index.html加上 base element

告訴router如何構成navigation URLs

因為app folder是application root



```
index.html
1  <!DOCTYPE html>
2  <html>
3
4  <head lang="en">
5    <base href="/">
6    <title>Acme Product Management</title>
7    <meta charset="UTF-8">
8    <meta name="viewport" content="width=device-width, initial-scale=1.0">
9
10   <link href="node_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet">
```

(補充) Set the <base href>

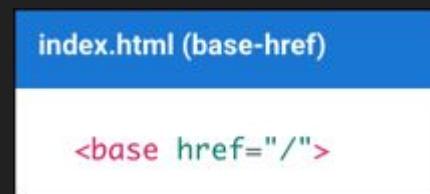
We must add a `<base href> element` tag to the `index.html` to make `pushstate` routing work. The browser also needs the base `href` value to prefix *relative URLs* when downloading and linking to css files, scripts, and images.

Add the base element just after the `<head>` tag. If the `app` folder is the application root, as it is for our application, set the `href` value in `index.html` exactly as shown here.

<https://angular.io/docs/ts/latest/guide/router.html#!#base-href>

必須加上 `<base href>` tag 到 `index.html` 來讓 `pushState` routing 作用
瀏覽器也需要 `href` value 來對相對 URL 路徑做前綴 (prefix)

如果 app folder 是應用程式的 root
則 href 就像右邊一樣設定



code/routes/basic/app/index.html

```
<!doctype html>
<html>
  <head>
    <base href="/" style="outline: 1px solid purple; border: 1px solid black; padding: 2px; margin-bottom: 5px;">
    <title>ng-book 2: Angular 2 Router</title>
```



```
{% for (var css in o.htmlWebpackPlugin.files.css) %} <link href="{%=o.htmlWebpackPlugin.files.css[css].url%}" type="text/css" rel="stylesheet">{% } %}
</head>
<body>
  <router-app></router-app>
  <script src="/core.js"></script>
```

(補充)

<https://angular.io/docs/ts/latest/guide/router.html#!#browser-url-styles>

HTML 5 style navigation is the Router default. Learn why "HTML 5" style is preferred, how to adjust its behavior, and how to switch to the older hash (#) style if necessary in the [Browser URL Styles](#) appendix below.

如果要切回 # style 請參考上方文章

step2: 加上RouterModule

import RouterModule

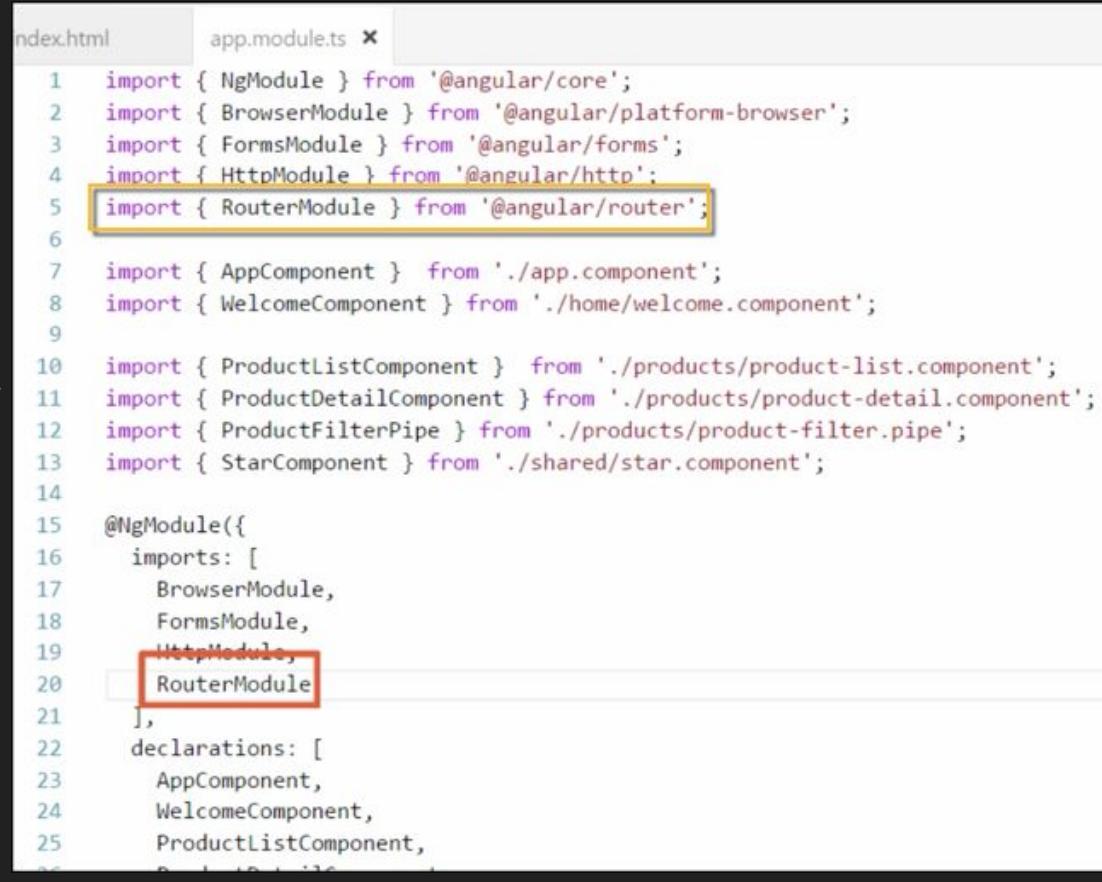
把RouterModule放到imports
array

=>

1.註冊了router's service provider

2.宣告router的directive

3.expose了configured routes



```
index.html      app.module.ts x
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms';
4  import { HttpClientModule } from '@angular/http';
5  import { RouterModule } from '@angular/router';
6
7  import { AppComponent } from './app.component';
8  import { WelcomeComponent } from './home/welcome.component';
9
10 import { ProductListComponent } from './products/product-list.component';
11 import { ProductDetailComponent } from './products/product-detail.component';
12 import { ProductFilterPipe } from './products/product-filter.pipe';
13 import { StarComponent } from './shared/star.component';
14
15 @NgModule({
16   imports: [
17     BrowserModule,
18     FormsModule,
19     HttpClientModule,
20     RouterModule
21   ],
22   declarations: [
23     AppComponent,
24     WelcomeComponent,
25     ProductListComponent,
```

RouterModule怎麼知道我們設定的routes?

將routes放在array後傳入forRoot()

```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot([
      { path: 'products', component: ProductListComponent },
      { path: 'product/:id', component: ProductDetailComponent },
      { path: 'welcome', component: WelcomeComponent },
      { path: '', redirectTo: 'welcome', pathMatch: 'full' },
      { path: '**', redirectTo: 'welcome', pathMatch: 'full' }
    ])
},
```

幫route綁上actions

```
app.component.ts
...
@Component({
  selector: 'pm-app',
  template:
    <ul class='nav navbar-nav'>
      <li><a [routerLink]=["['/welcome']">Home</a></li>
      <li><a [routerLink]=["['/products']">Product List</a></li>
    </ul>
  )
})
```

這邊把routerLink 綁上(bind) template expression
=>回傳link parameters array (即 ['/welcome']))

第一個參數是route的path(為string), 可加入其他element
代表額外的route parameters

step1 : 到app.component.ts增加routerLink

Home => 連到welcome

Product List=>連到/products

```
app.component.ts x
1 import { Component } from '@angular/core';
2
3 import { ProductService } from './products/product.service';
4
5 @Component({
6   selector: 'pm-app',
7   template: `
8     <div>
9       <nav class='navbar navbar-default'>
10         <div class='container-fluid'>
11           <a class='navbar-brand'>{{pageTitle}}</a>
12           <ul class='nav navbar-nav'>
13             <li><a [routerLink]=["'/welcome']>Home</a></li>
14             <li><a [routerLink]=["'/products']>Product List</a></li>
15           </ul>
16         </div>
17       </nav>
18     </div>
19   `,
20   providers: [ ProductService ]
21 })
```

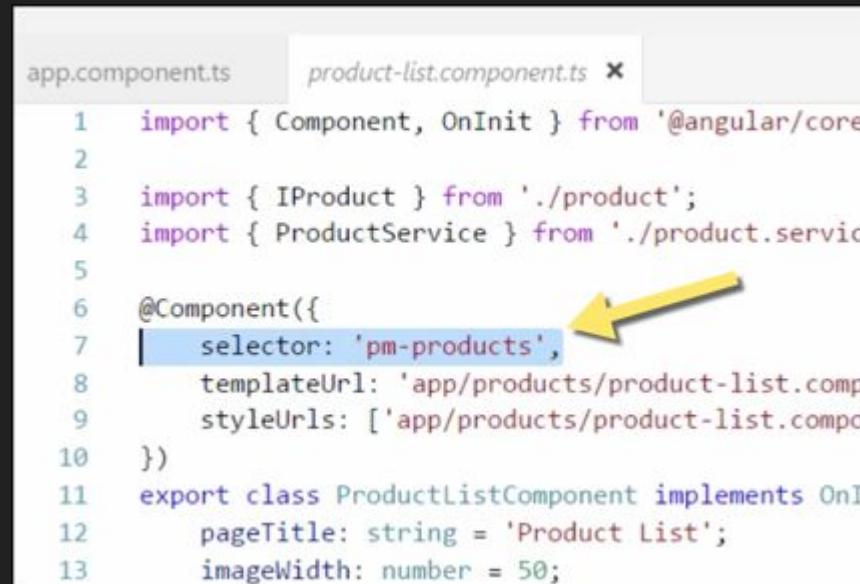
step2: product-list.component.ts

因為現在product-list component

要用router來切換

不再是nested component了

所以可把selector拿掉



```
app.component.ts          product-list.component.ts ×
1  import { Component, OnInit } from '@angular/core
2
3  import { IProduct } from './product';
4  import { ProductService } from './product.service'
5
6  @Component({
7    selector: 'pm-products',
8    templateUrl: 'app/products/product-list.comp
9    styleUrls: ['app/products/product-list.compo
10   })
11  export class ProductListComponent implements OnI
12    pageTitle: string = 'Product List';
13    imageWidth: number = 50;
```

啟動的view要顯示在哪？

用<router-outlet>

```
app.component.ts
...
@Component({
  selector: 'pm-app',
  template: `
    <ul class='nav navbar-nav'>
      <li><a [routerLink]="'[ '/welcome' ]'">Home</a></li>
      <li><a [routerLink]="'[ '/products' ]'">Product List</a></li>
    </ul>
    <router-outlet></router-outlet>
  `)
```

當不同的route被
activate之後

view會顯示在
<router-outlet>

```
@Component({
  selector: 'pm-app',
  template: `
    <div>
      <nav class='navbar navbar-default'>
        <div class='container-fluid'>
          <a class='navbar-brand'>{{pageTitle}}</a>
          <ul class='nav navbar-nav'>
            <li><a [routerLink]=["/welcome"]>Home</a></li>
            <li><a [routerLink]=["/products"]>Product List</a></li>
          </ul>
        </div>
      </nav>
      <div class='container'>
        <b><router-outlet></router-outlet></b>
      </div>
    </div>
  `,
  providers: [ ProductService ]
})
```

其
他

TypeScript error

Error The path d:\Very Important\Desktop\MEAN 2.0\node_modules\typescript\lib doesn't point to a valid tsserver install. TypeScript language features will be disabled.

使用較新版本的TypeScript

vs code預設使用最新的穩定版本

TypeScript會執行version checking(locally and globally)

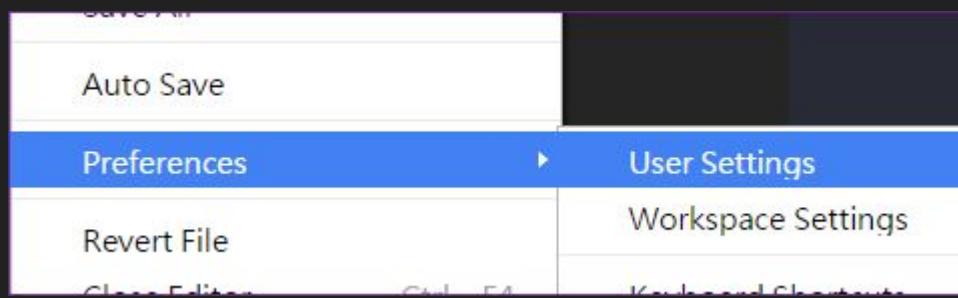
如果global version跟VScode的TypeScript version不一樣時會有警告

你可以讓VScode使用你workspace的typescript版本

設定typescript.tsdk指向包含tsserver.js的資料夾

```
{  
  "typescript.tsdk": "./node_modules/typescript/lib"  
}
```

改 user setting



```
45 ],
46 },
47 "editor.fontSize": 18,
48 "editor.formatOnSave": true,
49 "typescript.tsdk": "./node_modules/typescript/lib",
50 "window.zoomLevel": 1
51 }
```

A screenshot of a code editor displaying a JSON configuration file. The file contains several settings, including font size, save format, and TypeScript compiler paths. The 'typescript.tsdk' entry is highlighted with a yellow rectangular selection box. The code editor has a dark theme with syntax highlighting for the JSON keywords and values.