

# Angular 9 開發實戰：進階開發篇

深入了解 Angular Router 路由機制

多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

<https://blog.miniasp.com>





Angular Routing: Getting Started

Angular 路由快速上手

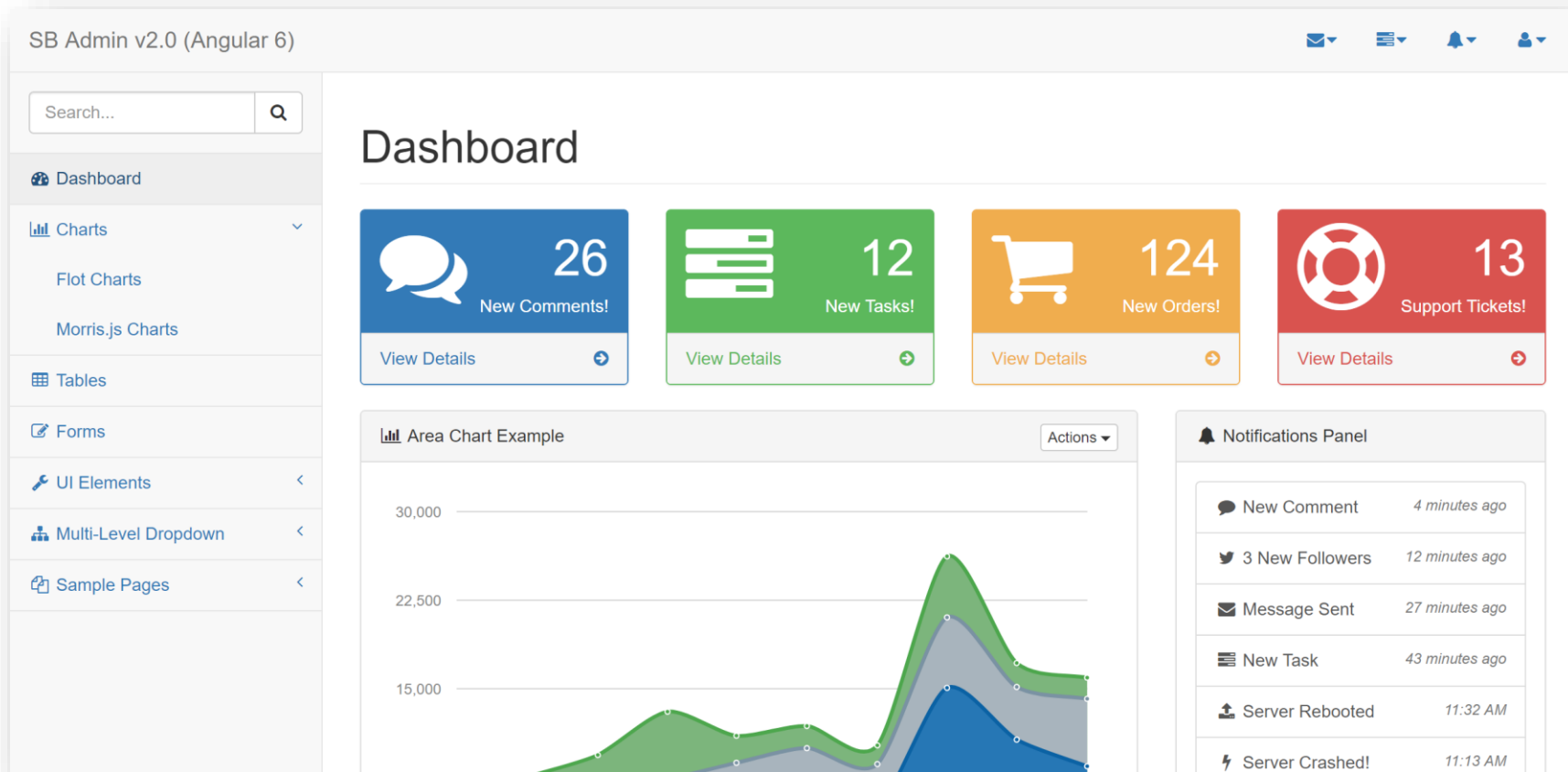
# 關於 Angular 的元件架構

- Angular 網站就是一整套由**元件**組成的**應用程式**
  - 最上層 UI 元件又稱為「根元件」( **AppComponent** )
  - **每個元件**可以再切割成多個不同的**子元件**(可重複利用的 UI 元件)
  - 整體是以一個**樹狀結構**的**元件架構**組成完整的應用程式
- Angular 是一套 **SPA** (Single Page Application) 框架
  - 在多頁面的網站架構下**每個頁面**就是一個**子元件**
- 提問：在 **SPA** 架構下**頁面與頁面**之間如何切換？

# 關於 Angular 路由機制

- 關於 Angular 路由的主要用途
  - 依據網址路徑來重新配置頁面中應該呈現的 UI 元件
  - 透過網址路徑與查詢字串來保存某些路由狀態 (RouterState)
  - 路由狀態最重要的就是紀錄網址路徑與元件之間的關係
- 關於 Angular 內建的路由策略
  - 使用 **HashLocationStrategy** 路由策略
    - 網址結構：`http://localhost:4200/#/dashboard/`
  - 使用 **PathLocationStrategy** 路由策略
    - 網址結構：`http://localhost:4200/dashboard/`

# 示範 Angular Router 的效果



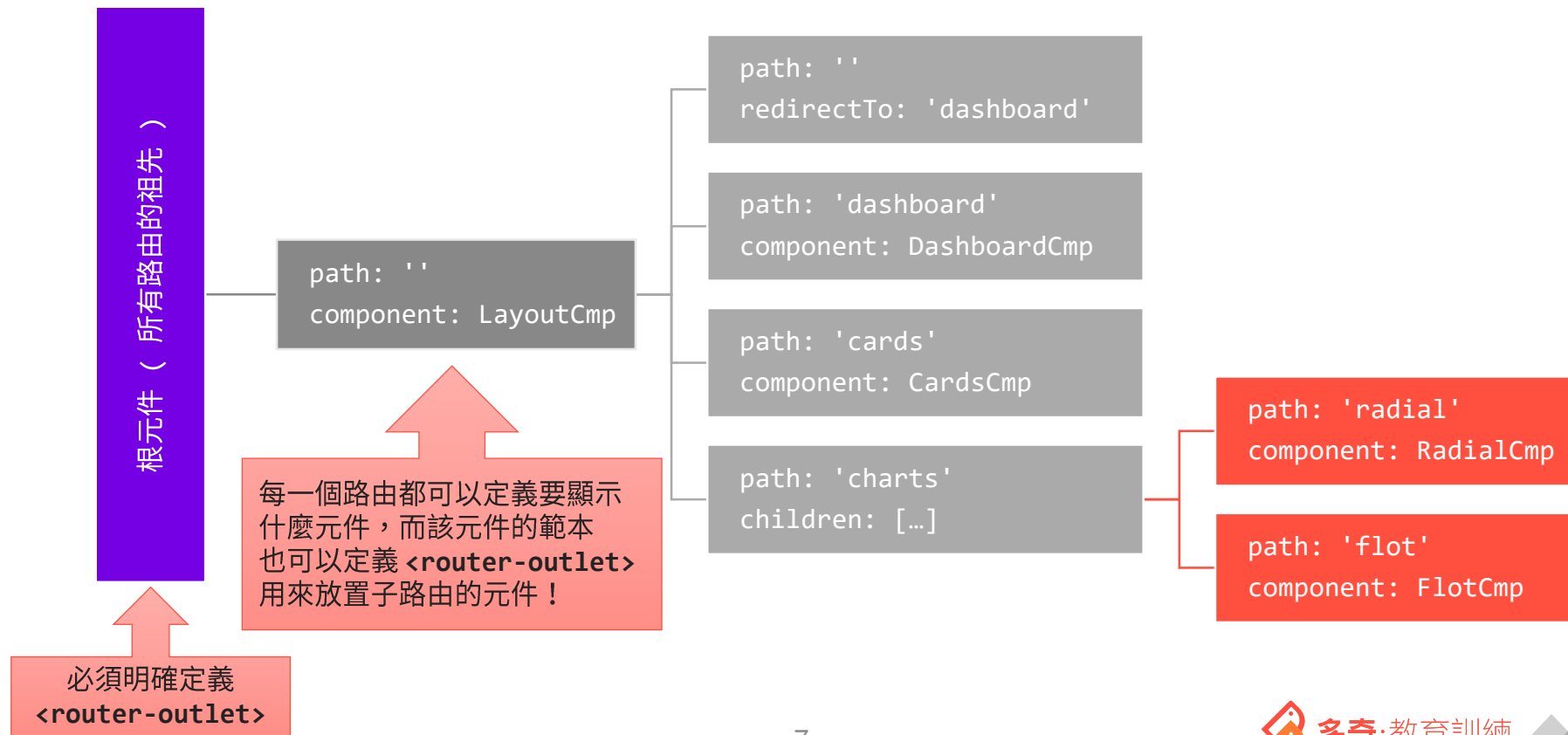
# Angular 路由設定範例

- 路由定義也是一種樹狀結構，並預先定義路由狀態

```
{
  path: '', component: LayoutComponent,
  children: [
    { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
    { path: 'dashboard', component: DashboardComponent },
    { path: 'cards/:type', component: CardsComponent },
    { path: 'charts',
      children: [
        { path: 'flot/:dots', component: FlotComponent }
      ]
    }
  ]
}
```

# 認識 <router-outlet> 路由插座

- 路由插座（<router-outlet>）就是每一頁放置元件的地方



# 實作簡單的第一層路由 - 檔案結構

- 先查看幾個重要檔案

- **main.ts** Angular 程式的進入點
- **app.module.ts** Angular 主要啟動模組
  - 從外部匯入 `AppRoutingModule` 模組
- **app-routing.module.ts** Angular 最上層路由定義模組
  - 必須在 `@NgModule()` 設定 **imports** 把路由定義載入
    - `RouterModule.forRoot(routes)`
- **app.component.ts** Angular 應用程式根元件（必要）
- **app.component.html** Angular 應用程式根元件範本（必要）
  - 這裡會安插一個路由插座（**<router-outlet></router-outlet>**）



# 使用 PathLocationStrategy 的注意事項

- 絕對網址

`https://domain/path/to/file?key=value`

- 相對網址

`//domain/path/to/file?key=value`

相對於**目前通訊協定**的網址

`/path/to?key=value`

相對於**目前域名**的路徑

`path/to?key=value`

相對於**目前路徑**的路徑

- 基底網址

- **路由機制**通常都會搭配 HTML 的 **<base href="/">** 進行宣告
- 路由時都是透過 URL 進行切換，因此**基底網址**非常重要！
- 通常設定於 **src/index.html** 網頁中（首頁）

# PathLocationStrategy 路由策略

- 預設 **app-routing.module.ts** 路由定義

```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],
```

- 網址結構

- **http://localhost:4200/dashboard/**

- 採用技術

- [HTML5 History API](#) ( [MDN](#) )

- 支援瀏覽器：**IE10+**, Safari 5+, iOS 4, Firefox 4+, Google Chrome

- [The State of the HTML5 History API](#)

- [部署 Angular 應用程式到 IIS 網站伺服器的說明](#)

# HashLocationStrategy 路由策略

- 修改 `app-routing.module.ts` 路由定義

```
@NgModule({  
  imports: [RouterModule.forRoot(routes, {  
    useHash: true  
  })],
```

- 網址結構
  - `http://localhost:4200/#/dashboard/`
- 採用技術
  - HTML4 標準 Hash 網址格式 (IE6+)

# 實作簡單的第一層路由

- 建立兩個新元件
  - ng g c page1
  - ng g c page2
- 修改 `app-routing.module.ts` 路由定義

```
const routes: Routes = [  
  { path: 'page1', component: Page1Component },  
  { path: 'page2', component: Page2Component }  
];
```

- 開啟 <http://localhost:4200/> 進行預覽
  - 開啟 <http://localhost:4200/page2> 進行預覽
  - 開啟 <http://localhost:4200/error> 進行預覽

# 在路由之間建立超連結 - 常量繫結

- app.component.html
  - 路由連結 (不用**中括號**就只能繫結**固定的字串**資料, **不能繫結變數**)

```
<ul>  
  <li><a routerLink="/">Home</a></li>  
  <li><a routerLink="/page1">Page1</a></li>  
  <li><a routerLink="/page2">Page2</a></li>  
</ul>
```
  - 路由插座
    - `<router-outlet></router-outlet>`
  - 注意事項
    - 不能寫成 `<router-outlet />` 喔！

# 在路由之間建立超連結 - 物件繫結

- app.component.html

- 路由連結 (加上**中括號**可繫結一個**陣列物件**)

```
<ul>
```

```
  <li><a [routerLink]="['/']">Home</a></li>
```

```
  <li><a [routerLink]="['/page1']">Page1</a></li>
```

```
  <li><a [routerLink]="['/page2']">Page2</a></li>
```

```
</ul>
```

- 路由插座

- <router-outlet></router-outlet>

- 注意事項

- 不能寫成 <router-outlet /> 喔！

# 在路由連結套用 routerLinkActive 樣式

- app.component.css

```
.active { background-color: darkblue; }
```

- app.component.html

- 路由連結 (不用**中括號**就只能繫結**固定的字串**資料)

```
<ul>
```

```
  <li><a routerLink="/"
    routerLinkActive="active">Home</a></li>
```

```
  <li><a routerLink="/page2"
    routerLinkActive="active">Page2</a></li>
```

```
</ul>
```

- 查看 <http://localhost:4200/page2> 網址, 看是否有套用 **active** 類別

# 路由連結如何套用 routerLinkActive 樣式

- 路由定義是樹狀結構資料
  - 當網址路由在比對 routerLink 時，會從根路由開始比對起！
  - 假設我們有兩個不同路由的頁面
    - /
    - /page2
  - 網址路由停在 / 的時候 routerLinkActive 會這樣比對路由：
    - 比對 / 路由連結，比對成功
    - 比對 /page2 路由連結，無法比對到目前網址 ( / )，比對失敗
  - 網址路由停在 /page2 的時候 routerLinkActive 會這樣比對路由：
    - 比對 / 路由連結 (由於 / 可以成功比對 /page2 網址)，比對成功
    - 比對 /page2 路由連結，比對成功
- 核心概念
  - 預設 routerLinkActive 會套用到頁面的上層路徑
  - 此特性非常適合用在選單類型的樣式類別套用



# 避免套用 routerLinkActive 到上層路徑

- 解決方案

- 只要在路由連結（routerLink）套用以下 Directive 即可

**[routerLinkActiveOptions]="{exact: true}"**

- 使用範例

```
<ul>
  <li><a routerLink="/" routerLinkActive="active"
    [routerLinkActiveOptions]="{exact: true}">
    Home</a>
  </li>
  <li><a routerLink="/page1" routerLinkActive="active">
    Page1</a>
  </li>
</ul>
```

# 定義預設路由 (default route)

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

import { Page1Component } from './page1/page1.component';
import { Page2Component } from './page2/page2.component';

const routes: Routes = [
  { path: '', component: Page1Component },
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component }
];
```

# 定義轉向路由 (redirect route)

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

import { Page1Component } from './page1/page1.component';
import { Page2Component } from './page2/page2.component';

const routes: Routes = [
  { path: '', redirectTo: '/page1', pathMatch: 'full' },
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component }
];
```

- 注意事項

- 使用 **redirectTo** 時，一定要加上 **pathMatch** 屬性
- 轉向路由最多 **只能轉向一次**，**不能**從 / 轉到 /page1 然後再轉向 /page2

# 定義萬用路由 (wildcard route)

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

import { Page1Component } from './page1/page1.component';
import { Page2Component } from './page2/page2.component';

const routes: Routes = [
  { path: '', redirectTo: '/page1', pathMatch: 'full' },
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component },
  { path: '**', redirectTo: '/page1', pathMatch: 'full' }
];
```

- 注意事項
  - 使用萬用路由時，一定要放在最後一個路由定義中！

# 定義獨立的 Route 物件

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

```
import { Page1Component } from '../page1/page1.component';
```

```
import { Page2Component } from '../page2/page2.component';
```

```
const fallbackRoute: Route = {  
  path: '**', redirectTo: '/page1', pathMatch: 'full'  
};
```

```
const routes: Routes = [  
  { path: '', redirectTo: '/page1', pathMatch: 'full' },  
  { path: 'page1', component: Page1Component },  
  { path: 'page2', component: Page2Component },  
  fallbackRoute  
];
```

# 實戰演練：套用多頁面路由與元件

- 將以下兩個靜態頁面轉成 Angular 元件
  - charts.html
  - tables.html
- 將新元件設定為兩條全新路由並設定選單連結
  - /charts
  - /tables
- 新增預設路由 / 轉向到 /dashboard 路由
  - 需新增 DashboardComponent 並將 AppComponent 範本移到此元件
- 新增萬用路由並建立 NotFoundComponent 顯示查無此頁
  - 404.html

## 定義 /utilities 子路由 ( children )

```
const routes: Routes = [  
  { path: 'dashboard', component: DashboardComponent },  
  { path: 'tables', component: TablesComponent },  
  { path: 'charts', component: ChartsComponent },  
  { path: 'utilities', // 無元件路由 (僅包含子路由)  
    children: [  
      <<放置 Route 路由設定>>  
    ]  
  },  
  fallbackRoute  
];
```

## 定義 /utilities 子路由

```
const routes: Routes = [  
  { path: 'utilities', // 無元件路由 (僅包含子路由)  
    children: [  
      { path: 'animations', component: AnimationsComponent },  
      { path: 'borders', component: BordersComponent },  
      { path: 'colors', component: ColorsComponent },  
      { path: 'other', component: OtherComponent }  
    ],  
  },  
  fallbackRoute  
];
```



# 實戰演練：套用 /utilities 子路由

- 將以下頁面都轉成 Angular 元件與設定好路由
  - utilities-animation.html → /utilities/animations
  - utilities-border.html → /utilities/borders
  - utilities-color.html → /utilities/colors
  - utilities-other.html → /utilities/other
- 設定 /utilities 也可以正常顯示頁面
  - 預設顯示 OtherComponent 頁面
- Angular CLI 參考命令
  - `ng g c utilities/animations`



Angular Routing Parameters

Angular 路由參數

# 路由參數區分兩種

- **必要參數**（若不傳入此參數則路由會比對失敗）
  - 設定在 path 的參數
  - 參數宣告方式為 **:type**
  - 路由範例：`{ path: 'color/:type', component: ColorComponent }`
  - 網址範例：`/utilities/color/1`
- **非必要參數**（若不傳入此參數也不影響路由比對）
  - 矩陣參數 ([matrix URL notation](#))
    - 網址範例：`/utilities/color?type=1;size=2`
  - 查詢字串 ([Query String](#))
    - 網址範例：`/utilities/color?type=1&size=2`

# 如何從 UI 元件中取得路由參數

```
import { ActivatedRoute, Router, ParamMap } from '@angular/router';

export class CardsComponent implements OnInit {

  constructor(private router: Router,
               private route: ActivatedRoute) { }

  ngOnInit() {
    this.route.snapshot.paramMap.get('type'); // 取得當下的參數值
    this.route.paramMap.subscribe((params: ParamMap) => {
      this.type = +params.get('type'); // 參數變動時取得新值
    });
  }
}
```

※ 注意：this.route.paramMap 是一個 Observable 物件！

# 關於 Router 服務元件

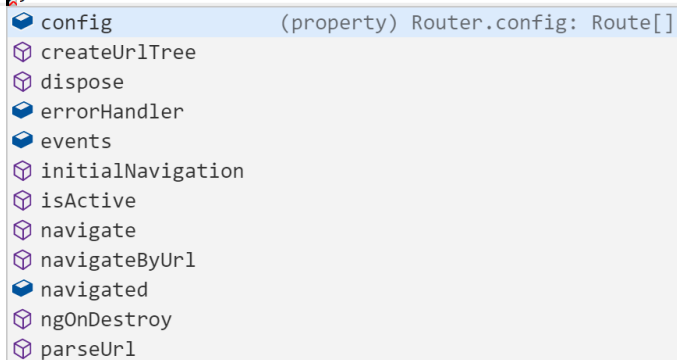
- 用來取得完整的路由資訊

- config
- events
- navigated

- 可透過 API 進行導覽

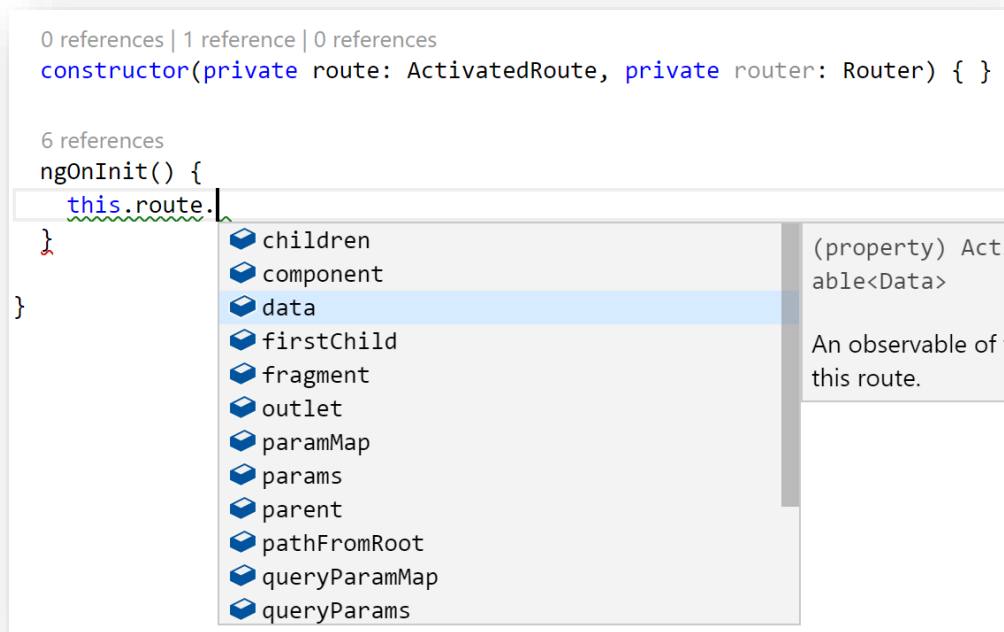
- navigate()
- navigateByUrl()

```
export class CardsComponent implements OnInit {  
  
  constructor(private router: Router, private route: ActivatedRoute) { }  
  
  ngOnInit() {  
    console.log(this.router.);  
  }  
}
```



# 關於 ActivatedRoute 服務元件

- 代表**目前正被啟用**的路由物件
- 可以從這個物件取得相當豐富的路由資訊（包含路由參數）



早期 Angular 會用 params 取得路由參數，但官網建議今後都改用 paramMap 來取得參數值！

<https://angular.io/guide/router#activated-route>

# 實戰演練：在頁面之間傳遞參數

- 調整 /utilities/color 路由
  - 可從網址路徑設定 type 參數
  - 若網址為 /utilities/color/1 則只會顯示第一個內容區塊 (div)
  - 若網址為 /utilities/color/2 則只會顯示第一個內容區塊 (div)
  - 若網址為 /utilities/color 則顯示所有內容
- 調整 /tables 頁面
  - 可傳入 num 參數，並可限制頁面中表格資料的呈現筆數
    - /tables?num=15
  - 資料來源：<http://www.mocky.io/v2/5c9e523f3000005500ee97cf>
- 調整選單連結，確保可以正確開啟頁面！

# 透過程式進行路由導覽 (Navigation)

- 絕對位址導覽

```
this.router.navigateByUrl('/home/1');    // 傳入字串
```

```
this.router.navigate(['/home/1']);    // 傳入陣列物件
```

- 相對位址導覽 (相對於**目前路由**的**網址路徑**)

```
this.router.navigate(['../'], { relativeTo: this.route });
```

```
this.router.navigate(['../2'], { relativeTo: this.route });
```



# 使用 navigationExtras 設定導覽參數

```
let navigationExtras: NavigationExtras = {  
    queryParamsHandling: 'preserve', // 轉向時保留當前查詢字串  
    preserveFragment: true // 轉向時保留當前 Hash 片段  
};
```

```
let redirect = '/utilities/other';
```

```
this.router.navigate([redirect], navigationExtras);
```

# 導覽時加入 Hash 片段

- `http://localhost:4200/tables#main`

- 準備 `NavigationExtras` 物件

```
let navigationExtras: NavigationExtras = {  
  fragment: 'main'  
};
```

- 程式寫法

```
this.router.navigate(['/tables'], navigationExtras);
```

- 範本寫法

```
<a routerLink="/tables" fragment="main">Price</a>
```

# 導覽時加入查詢字串

- `http://localhost:4200/tables?page=1&size=20`

- 準備 [NavigationExtras](#) 物件

```
let navigationExtras: NavigationExtras = {  
  queryParams: { page: 1, size: 20 }  
};
```

- 程式寫法

```
this.router.navigate(['/tables'], navigationExtras);
```

- 範本寫法

```
<a routerLink="/tables" [queryParams]="{ page: 1, size: 20 }">  
  Tables</a>
```

## 導覽時加入矩陣參數

- `http://localhost:4200/tables?page=1;size=20`
- 傳入到 `navigate` 方法第一個陣列參數的最後一個元素
- 程式寫法

```
this.router.navigate(['/tables', {  
  page: 1,  
  size: 20  
}]);
```

- 範本寫法

```
<a [routerLink]="['/tables', { page: 1, size: 20 }]">  
Tables</a>
```

# 實戰演練：各種導覽功能撰寫

- 調整 /dashboard 頁面邏輯
  - 點擊第一個 div 區塊可以透過範本中的 routerLink 外加 [queryParams] 連結到 /tables 頁面，並傳入 num 參數（設定每頁顯示筆數）
  - 點擊第二個 div 區塊可以透過 (click) 事件呼叫 this.router.navigate() 連結到 /tables 頁面，並傳入 num 參數（設定每頁顯示筆數）

# 取得路由參數的開發技巧 (switchMap)

- 透過 RxJS 的 [switchMap](#) 將一個 Observable 轉成另一個

```
import { switchMap } from 'rxjs/operators';
constructor(private router: Router, private service: Svc,
             private route: ActivatedRoute) {
  // 從第一個 Observable 得到的值傳給 switchMap
  // switchMap 會先接到事件資料，並回傳一個新的 Observable 物件
  this.route.paramMap
    .pipe(
      switchMap((params: ParamMap) => {
        return this.service.getHero(+params.get('id'))
      })
    ).subscribe((hero: Hero) => this.hero = hero);
}
```

# 使用 ActivatedRoute 的注意事項

- 大部分的 Observable 物件在**訂閱**之後都需要執行**取消訂閱**，只有少數例外不需要！
- 在 ActivatedRoute 物件中所有的 Observable 屬性，都不需要特別執行**取消訂閱**這個動作。
- ActivatedRoute 會隨著元件建立時建立新的物件實體並注入，也會隨著元件在摧毀時自動消失，因此你在訂閱像是 `this.route.paramMap` 這個 Observable 之後，**不需要取消訂閱**。



Lazy Loading Feature Modules

# Angular 延遲載入功能模組



# 建立具有延遲載入能力的功能模組

- 建立功能模組
  - `ng g m pages --routing`
- 建立元件並同時註冊進功能模組
  - `ng g c pages/blank`
- 定義功能模組下的子路由設定
  - `pages-routing.module.ts`
- 設定 `AppRoutingModule` 動態載入功能模組

```
{ path: 'pages', loadChildren:  
  () => import('./pages/pages.module').then(mod => mod.PagesModule)}
```

(強型別語法是 Angular 8.0.0 之後才有的特性)

# 模組延遲載入的優點

- 使用者需要甚麼功能，就只會下載特定功能的程式回來
  - 在大型網站應用程式專案可**有效降低頁面下載的大小**
- 加快頁面的載入速度，瀏覽器僅需下載必要的模組回來
  - 不用讓使用者在**第一頁**下載整個網站的所有模組、元件原始碼
- 每次載入特定功能模組，僅下載遺漏的模組程式碼
  - 大幅優化每個功能模組的載入速度

# 啟用模組延遲載入的必要條件

- 原本的 **AppModule** 不能跟功能模組有任何相依關係

```
6
7 import { AppComponent } from './app.component';
8 import { CardsComponent } from './cards/cards.component';
9 import { DashboardComponent } from './dashboard/dashboard.component';
10
11 - import { ChartsModule } from './charts/charts.module';
12 -
13 @NgModule({
14   declarations: [
15     AppComponent,
16     CardsComponent,
17     DashboardComponent
18   ],
19   imports: [
20     BrowserModule,
21     FormsModule,
22     HttpClientModule,
23 -   ChartsModule,
24     AppRoutingModule
25   ],
26   providers: [],
27   bootstrap: [AppComponent]
28 })
29 export class AppModule { }
```

# 啟用模組預先載入機制 (非同步載入)

- 找到最上層路由模組 ( app-routing.module.ts )

```
import { PreloadAllModules } from '@angular/router';
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    enableTracing: true,
    preloadingStrategy: PreloadAllModules
  })],
  exports: [RouterModule],
  providers: []
})
export class AppRoutingModule { }
```

# 模組載入機制

## 1. 全部載入 (預設)

- 透過 webpack 會將**所有模組**打包成 1 個 JS 檔案

## 2. 延遲載入

- 透過 webpack 將**部分模組**拆解成**可延遲載入的程式片段** (chunks)
- 使用 webpack 最為知名的 [code splitting](#) 技術
- 當使用者點開網頁，所有**可延遲載入的程式片段**並**不會預先載入**
- 主動擊進入特定路由，延遲載入的模組才會即時載入

## 3. 預先載入

- 透過 webpack 將**部分模組**拆解成**可延遲載入的程式片段** (chunks)
- 使用 webpack 最為知名的 [code splitting](#) 技術
- 當使用者點開網頁，所有**可延遲載入的程式片段**會**立即預先載入**
- **預先載入的過程透過非同步背景下載**，不影響畫面顯示或使用者操作

# 實戰演練：建立子路由模組與延遲載入

- 建立子路由模組（包含路由模組）
  - PagesModule
- 設定子路由模組
  - pages-routing.module.ts
- 設定根路由模組透過 **延遲載入** 機制來載入 PagesModule 模組
  - app-routing.module.ts
- 驗證延遲載入運作正常
  - 透過 F12 開發者工具的 Network 頁籤進行檢查

# forRoot() 與 forChild()

- forRoot() 定義**根路由**的路由設定
  - 包含整個應用程式的路由導覽規則設定
  - 整份 Angular 應用程式只能有一個 forRoot() 定義
- forChild() 定義**子路由**的路由設定
  - 只包含特定功能模組下的路由設定
  - 整份 Angular 可以定義**無限階層**的 forChild() 子路由設定



Route Guards

Angular 路由守門員



# 建立登入頁面

- 遭遇問題
  - 登入頁面的版型與其他頁面的版型不同！
- 解決方案
  - 在根元件與其他頁面之間多插入一層 Layout 元件
- 注意事項
  - 多階層版面須在每一層都加上 `<router-outlet></router-outlet>`
  - 記得調整 `<body>` 元素的 HTML 屬性 (HTML Attributes)
- 建立頁面
  - `ng g c login`

# 實戰演練：建立登入頁面

- 建立 Login 元件
- 建立 Layout 元件
- 調整版面與路由結構，讓所有頁面都能正常顯示

# 建立 AuthGuard 服務元件

- 建立 AuthGuard 服務元件並實作 CanActivate 介面

ng g g auth → 選擇 CanActivate 範本

ng g g auth --implements=CanActivate

- 實作必要的介面方法即可

```
canActivate(  
  next: ActivatedRouteSnapshot,  
  state: RouterStateSnapshot): Observable<boolean | UrlTree> |  
Promise<boolean | UrlTree> | boolean | UrlTree {  
  return true;  
}
```

# 實作 CanActivate 介面的注意事項

- 回傳型別
  - **Observable**<boolean | UrlTree>
  - **Promise**<boolean | UrlTree>
  - **boolean**
  - **UrlTree** (這是 Angular 7.1.0 之後才有的功能) ([DEMO](#))
- 身分認證失敗的 canActivate() 實作範例

```
if (!localStorage.getItem('apikey')) {  
    return this.router.parseUrl('/login');  
} else {  
    return true;  
}
```

# 啟用 Route Guards 設定

- 套用 canActivate 屬性到路由設定中

```
const routes: Routes = [
```

```
{
```

```
  path: '',
```

```
  component: LayoutComponent,
```

```
  canActivate: [AuthGuard],
```

```
  children: [
```

```
    { path: ''
```

```
    { path: ''
```

```
    { path: ''
```

```
  ]
```

```
},
```

```
{
```

```
  path: 'logi
```

```
}
```

```
canActivate
```

```
canActivateChild
```

```
ng-route-guard-canactivate
```

```
ng-route-guard-canactivatechild
```

```
a-guard-can-activate
```

```
a-guard-can-activate-child
```

```
canDeactivate
```

```
ng-route-guard-candeactivate
```

```
a-guard-can-deactivate
```

(property) Route.canActivate?: any[]

An array of dependency-injection tokens used to look up `CanActivate()` handlers, in order to determine if the current user is allowed to activate the component. By default, any user can activate.

# 實戰演練：透過 AuthGuard 驗證登入

- 建立 AuthGuard 服務元件
- 在路由設定 `canActivate: [AuthGuard]`
- 實作登入機制（登入時寫入 `localStorage['apikey']` 即可）
- 驗證 AuthGuard 正常運作

# 實作 CanDeactivate 介面的注意事項

- 回傳型別
  - **Observable**<boolean | UrlTree>
  - **Promise**<boolean | UrlTree>
  - **boolean**
  - **UrlTree** (這是 Angular 7.1.0 之後才有的功能) ([DEMO](#))
- [canDeactivate\(\)](#) 傳入參數
  - component: YourComponent 指定的元件
  - currentRoute: ActivatedRouteSnapshot 當前啟用的路由
  - currentState: RouterStateSnapshot 當前停留的路由狀態快照
  - nextState: RouterStateSnapshot 即將前往的路由狀態快照

# CanDeactivateLogin 實作範例

```
export class CanDeactivateLogin implements CanDeactivate<LoginComponent> {  
  canDeactivate(  
    component: LoginComponent,  
    currentRoute: ActivatedRouteSnapshot,  
    currentState: RouterStateSnapshot,  
    nextState: RouterStateSnapshot  
  ): Observable<boolean|UrlTree>|Promise<boolean|UrlTree>|boolean|UrlTree {  
  
    console.log(component);  
    console.log(currentRoute);  
    console.log(currentState);  
    console.log(nextState);  
  
    if (component.form.dirty) { return false; }  
    else { return true; }  
  
  }  
}
```



# 實戰演練：實作 CanDeactivateLogin

- 調整 LoginComponent 元件
  - @ViewChild(NgForm) form: NgForm;
- 實作 CanDeactivateLogin 服務元件
  - 可判斷 `component.form.dirty` (表單已填寫) 是否為 `true`

# 可使用的 Route Guards 清單

- [CanActivate](#) 是否可以進入指定的路由
- [CanActivateChild](#) 是否可以進入指定路由的子路由
- [Resolve](#) 在進入路由前預先取得資料
- [CanLoad](#) 是否可以延遲載入功能模組
- [CanDeactivate](#) 是否可以離開目前路由



Router Events

路由事件

# 啟用路由追蹤

- 修改 `app-routing.module.ts` 路由定義

```
@NgModule({  
  imports: [RouterModule.forRoot(routes, {  
    enableTracing: true  
  })],  
  exports: [RouterModule],  
  providers: []  
})  
  
export class AppRoutingModule { }
```

- 按下 **F12** 進入 Console 畫面即可查看路由追蹤資訊！

## 訂閱 router.events 屬性

- 注入 Router 服務元件後，可訂閱 events 屬性查看所有路由事件

```
export class AppComponent {  
  constructor(private router: Router) {  
    this.router.events.subscribe(event => {  
      console.log(event);  
    });  
  }  
}
```

▶ NavigationStart {id: 2, url: <code>"/products"</code> , navigationTrigger: <code>"imperative"</code> , restoredState: null}	<a href="#">app.component.ts:14</a>
▶ RouteConfigLoadStart {route: {...}}	<a href="#">app.component.ts:14</a>
▶ RouteConfigLoadEnd {route: {...}}	<a href="#">app.component.ts:14</a>
▶ RoutesRecognized {id: 2, url: <code>"/products"</code> , urlAfterRedirects: <code>"/products"</code> , state: RouterStateSnapshot}	<a href="#">app.component.ts:14</a>
▶ GuardsCheckStart {id: 2, url: <code>"/products"</code> , urlAfterRedirects: <code>"/products"</code> , state: RouterStateSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationStart {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationStart {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationStart {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationStart {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationStart {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationStart {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
	<a href="#">app.component.ts:14</a>
▶ GuardsCheckEnd {id: 2, url: <code>"/products"</code> , urlAfterRedirects: <code>"/products"</code> , state: RouterStateSnapshot, shouldActivate: true}	<a href="#">app.component.ts:14</a>
▶ ResolveStart {id: 2, url: <code>"/products"</code> , urlAfterRedirects: <code>"/products"</code> , state: RouterStateSnapshot}	<a href="#">app.component.ts:14</a>
▶ ResolveEnd {id: 2, url: <code>"/products"</code> , urlAfterRedirects: <code>"/products"</code> , state: RouterStateSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ ChildActivationEnd {snapshot: ActivatedRouteSnapshot}	<a href="#">app.component.ts:14</a>
▶ NavigationEnd {id: 2, url: <code>"/products"</code> , urlAfterRedirects: <code>"/products"</code> }	<a href="#">app.component.ts:14</a>
▶ Scroll {routerEvent: NavigationEnd, position: null, anchor: null}	<a href="#">app.component.ts:14</a>

# 路由事件列表 (1)

事件名稱	說明
NavigationStart	當路由導覽開始時觸發
RouterConfigLoadStart	讀取 lazy loading 模組的路由設定前觸發
RouterConfigLoadEnd	讀取 lazy loading 模組的路由設定後觸發
RoutesRecognized	當解析到正確的路由設定對應時觸發
GuardsCheckStart	進入路由守門員 (Route Guard) 階段前觸發
ChildActivationStart	當要啟動子路由前觸發
ActivationStart	當要啟動目前路由前觸發
GuardsCheckEnd	路由守門員檢查完後觸發
ResolveStart	進入到路由解析 (Resolve) 階段前觸發
ResolveEnd	路由解析完成後觸發
ChildActivationEnd	子路由啟動完成後觸發
ActivationEnd	路由啟動完成後觸發
NavigationEnd	路由導覽結束後觸發

## 路由事件列表 (2)

事件名稱	說明
NavigationCancel	當路由守門員回傳 false 時，路由導覽會被取消，之後觸發此事件
NavigationError	當路由過程中發生未預期的錯誤時觸發
Scroll	路由可以進行卷軸滾動時觸發



# 路由事件的應用 (1)

- 在每頁導覽時加上過場特效 ([DEMO](#))

```
export class AppComponent {  
  loading = false;  
  constructor(private router: Router) {  
    router.events.subscribe(event => {  
      if(event instanceof NavigationStart) {  
        this.loading = true;  
      }  
      if(event instanceof NavigationEnd) {  
        this.loading = false;  
      }  
    })  
  }  
}
```

## 路由事件的應用 (2)

- 取得網址並產生麵包屑 ([DEMO](#))

```
export class AppComponent {  
  mapping = new Map<string, string>([  
    ['/page-a', 'Page A'], ['/page-b', 'Page B']  
  ]);  
  breadcrumbs = '';  
  constructor(private router: Router) {  
    router.events  
      .pipe(  
        filter(e => e instanceof NavigationEnd),  
        map((e: NavigationEnd) => e.url)  
      )  
      .subscribe(url => {  
        console.log(url);  
        this.breadcrumbs = this.mapping.get(url);  
      })  
  }  
}
```



Advanced Router Tips

## 進階路由應用

# 使用 Resolver 預先取得資料

- 建立 Resolver 服務元件 ([DEMO](#))

```
export class TodoResolver implements Resolve<any> {  
    constructor(private todoService: TodoService) { }  
  
    resolve(  
        route: ActivatedRouteSnapshot,  
        state: RouterStateSnapshot) {  
  
        return this.todoService.getTodo(route.paramMap.get('id'));  
    }  
}
```

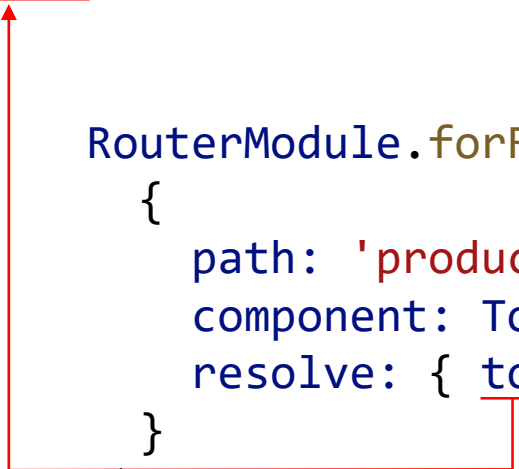
# 在路由中設定 Resolver (DEMO)

```
RouterModule.forRoot([  
  {  
    path: 'todo/:id',  
    component: TodosComponent,  
    resolve: { todos: TodoResolver }  
  }  
])
```

# 在元件中取得 Resolver 解析後的資料

```
export class TodosComponent implements OnInit {  
  todos: any;  
  constructor(private route: ActivatedRoute) { }  
  ngOnInit() {  
    this.route.data.subscribe(data => {  
      this.todos = data.todos;  
    })  
  }  
}
```

```
RouterModule.forRoot([  
  {  
    path: 'product/:id',  
    component: TodosComponent,  
    resolve: { todos: TodoResolver }  
  }  
])
```



A red arrow originates from the `todos` property access in the `RouterModule.forRoot` configuration (specifically from `todos: TodoResolver`) and points upwards to the `this.todos` assignment in the `ngOnInit` method of `TodosComponent`. This illustrates how the resolved data is passed to the component.

# 當導覽到相同網址時，強制重新整理

- 只能設定在 RouterModule.forRoot() 內
- 如果導覽至相同路由，可設定強制重新重載路由（並非重整頁面）
  - 只有 **router.events** 會被重新執行一次
  - 整個 Component 並不會重新建立（意即 `ngOnInit` 並不會重新執行）
  - 可搭配**路由規則**設定 **runGuardsAndResolvers: 'always'**
    - 當路由事件重新觸發時再執行一次 Route Guards 與 Resolvers
- 範例程式：<https://stackblitz.com/edit/onsameurlnavigation>

```
RouterModule.forRoot(routes, {  
  onSameUrlNavigation: 'reload'  
}))
```

# 路由卷軸滾動器 (Router Scroller)

- Angular 6.1 增加了路由卷軸滾動器的功能，解決了兩個問題
  - 使用瀏覽器回上一頁功能時，保留上一頁的位置
  - 當導覽網址包含雜湊片段 (#) 時，能夠前往正確的元素



# scrollPositionRestoration

- <https://angular.io/api/router/ExtraOptions#scrollPositionRestoration>
- 路由導覽時，決定是否要還原卷軸的位置
  - top: 永遠回到最上面
  - enabled: 回到上一頁時還原卷軸的位置
  - disabled: 回到上一頁時部還原卷軸位置 (預設值)

```
RouterModule.forRoot(routes, {  
  scrollPositionRestoration: 'enabled'  
})
```



# anchorScrolling

- <https://angular.io/api/router/ExtraOptions#anchorScrolling>
- 當網址包含 Hash 片段時，是否可以移動到目標位置
  - disabled: 不移動到目標位置 (預設值)
  - enabled: 移動到目標位置

```
RouterModule.forRoot(routes, {  
  anchorScrolling: 'enabled'  
})
```

# scrollOffset

- <https://angular.io/api/router/ExtraOptions#scrollOffset>
- 當產生卷軸滾動時，加上一個額外的間距
  - [number, number]: 分別沿著 x 軸, y 軸加上額外間距
  - () => [number, number]: 呼叫指定的方法, 回傳指定的間距

```
RouterModule.forRoot(routes, {  
  anchorScrolling: 'enabled',  
  scrollOffset: [0, 30]  
})
```



# 聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

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

Facebook

Will 保哥的技术交流中心

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

Twitter

[https://twitter.com/Will\\_Huang](https://twitter.com/Will_Huang)



多奇·教育訓練

**THANK YOU!**

Q&A