

## Pipe 用法

### 一、新增一個 Pipe 用法

甲、輸入 `Ng g p [名稱]`

```
D:\todomvc\todomvc>ng g p state-filter
CREATE src/app/state-filter.pipe.spec.ts (208 bytes)
CREATE src/app/state-filter.pipe.ts (211 bytes)
UPDATE src/app/app.module.ts (670 bytes)
```

二、因為我有切 Service，所以在 `data.service.ts` 加入一個變數，不然也可以在 `app.component.ts` 層加入變數即可

```
export class DataService {
  state;
}
```

三、在新建的 `state-filter.pipe.ts` 中寫入想要的方法

甲、第一個 `value: any` 是回傳值

乙、`state?: any` 是參數，要傳多個參數可以用逗點的方式繼續加下去

```
export class StateFilterPipe implements PipeTransform {
  transform(value: any, state?: any): any {
    if (state === 'active') {
      return value.filter(x => x.isCompleted === false);
    }
    if (state === 'completed') {
      return value.filter(x => x.isCompleted === true);
    }
    return value;
  }
}
```

四、在 `state-filter.pipe.ts` 設定

甲、上方有加入 `pure: false` 就代表即時更新

```
@Pipe({
  name: 'stateFilter',
  pure: false
})
```

五、之後在 `app.component.html` 中使用剛剛建立的 Pipe

甲、將 `dataService.todos` 回傳內容

乙、轉換為 stateFilter: dataService.state 的型態

```
<li *ngFor=" let todo of (dataService.todos | stateFilter:
dataService.state); let idx = index " [ngClass]="{ completed:
todo.isCompleted }" >
```

六、最後就是要在 section-footer.component.html 加入 click 事件

甲、加入 Click 事件(click)="dataService.state = 'active'"

乙、同時加入 CSS 樣式，[class.selected]="dataService.state === 'active'"

丙、上方跟[ngClass]="{ selected: dataService.state === 'active' }"同樣意思

```
<ul class="filters">
  <li>
    <a [class.selected]="dataService.state === '' href="#"
(click)="dataService.state = ''">All</a>
  </li>
  <li>
    <a href="#/active" [class.selected]="dataService.state ===
'active'" (click)="dataService.state = 'active'" >Active</a>
  </li>
  <li>
    <a href="#/completed" [class.selected]="dataService.state ===
'completed'" (click)="dataService.state = 'completed'" >Completed</a>
  </li>
</ul>
```

## 重建專案失敗解決方案

- 一、如果輸入 `ng serve` 時有遇到以下問題
- 二、可以輸入 `npm install` 重新安裝 module
- 三、可能是電腦一開始沒有幫我們建立到 module 這部份

```
D:\callApi>ng serve
Could not find module "@angular-devkit/build-angular" from "D:\\callApi".
Error: Could not find module "@angular-devkit/build-angular" from "D:\\callApi".
    at Object.resolve (C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\node_modules\angular-devkit\core\node\resolve.js:111)
    at Observable.rxjs_1.Observable [as _subscribe] (C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\node_modules\angular-devkit\architect\src\architect.js:132:48)
    at Observable._trySubscribe (C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\node_modules\rxjs\internal\Observable.js:25)
    at Observable.subscribe (C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\node_modules\rxjs\internal\Observable.js:38:2)
    at C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\node_modules\rxjs\internal\Observable.js:99:19
    at new Promise (<anonymous>)
    at Observable.toPromise (C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\node_modules\rxjs\internal\Observable.js:97:1)
    at ServeCommand.initialize (C:\Users\Lenovo\AppData\Roaming\npm\node_modules\angular\cli\models\architect-command.js:91:94)
    at process._tickCallback (internal/process/next_tick.js:68:7)
    at Function.Module.runMain (internal/modules/cjs/loader.js:744:11)

D:\callApi>npm install
```

## 建立第一個 API

一、在 app.module.ts 加入 import

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

二、同時在下方 imports 加入 HttpClientModule

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

三、此時就可以在 app.component.ts 引用

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

四、Call API 的方式

甲、Constructor 一個剛剛引用的 HttpClient

乙、使用 http.get 方法，寫入對應 URL

丙、透過 Subscribe 回傳 console.log(data)

```
export class AppComponent {  
  constructor(private http: HttpClient) {  
    http  
      .get('https://jsonplaceholder.typicode.com/posts')  
      .subscribe(data => {  
        console.log(data);  
      });  
  }  
}
```

## 透過 MAP 方式做欄位對應

一、在 app.component.ts 引用 MAP

```
import { map } from 'rxjs/operators';
```

二、使用 MAP 的方式

甲、使用.pipe 的方式用 MAP 欄位對應

乙、然後只 return id 跟 title

```
constructor(private http: HttpClient) {  
  http  
    .get('https://jsonplaceholder.typicode.com/posts')  
    .pipe(  
      map((posts: any[]) => {  
        return posts.map((post: any) => {
```

```

        return { id: post.id, title: post.title };
    });
  })
)
.subscribe(data => {
  console.log(data);
});
}

```

### 三、重構程式碼

甲、建立 posts 陣列變數

乙、改成只註冊 constructor

丙、建立 apiSource 直接 Call API 取值

丁、利用 loadData 處理方法回傳結果

戊、未來可以不要寫在 component.ts，應該存在 Service 層

```

export class AppComponent {
  posts = [];
  constructor(private http: HttpClient) {}

  apiSource =
this.http.get('https://jsonplaceholder.typicode.com/posts').pipe(
  map((posts: any[]) => {
    return posts.map((post: any) => {
      return { id: post.id, title: post.title };
    });
  })
);

loadData() {
  this.apiSource.subscribe(data => {
    console.log(data);
    this.posts = data;
  });
}
}

```

使用 Router 代替 HTML href

一、在 app.module.ts 內寫入 import Route 內容

甲、必須同時引用 RouterModule,Route

乙、而 Page1Component 是我們透過 ng g c 新建的 Component 位置

```
import { RouterModule,Route } from '@angular/router';
import { Page1Component } from '../page1/page1.component';
```

二、且在 imports 下方 @Module 中間，新增常數 routes(第三步會解釋)

```
import { Page1Component } from '../page1/page1.component';
6 import { Page1Component } from '../page1/page1.component';
7
8 const routes: Route[] = [
9   { path: 'page1', component: Page1Component },
10  { path: '**', redirectTo: '/page1', pathMatch: 'full' }
11 ];
12
13 @NgModule({
14   declarations: [AppComponent, Page1Component],
15   imports: [BrowserModule, HttpClientModule, RouterModule.forRoot(routes)
```

三、修改常數內容為對應連結

甲、Path 是前端輸入的 routerLink 連結名稱

乙、Component 是剛剛引用的名稱

丙、Path=\*\*代表如果都不是上方輸入的情況

丁、redirectTo 代表轉址到的地點

戊、pathMatch 是代表位置要全部相符

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

四、在 Module 加入中加入對應內容

甲、Declarations 中需要加入要連結的 Component

乙、Imports 有分 forRoot 和 forChild，通常使用 forRoot 即可

```
@NgModule({
  declarations: [AppComponent,Page1Component],
  imports: [BrowserModule, HttpClientModule,
RouterModule.forRoot(routes)],
  providers: [],
  bootstrap: [AppComponent]
})
```

五、最後就可以在 app.component.html 中設定

甲、routerLink 轉址

乙、並加入 router-outlet，即可更改部份內容

```
<ul>
```

```

<li><a routerLink="/page1">Page1</a></li>
</ul>
<router-outlet></router-outlet>

```

加入子畫面項目

一、在 app.module.ts 內寫入 import Route 內容

甲、必須同時引用 RouterModule,Route

乙、而 Page1, Page2,Page2-detail 是透過 ng g c 新建的 Component 位置

```

import { RouterModule, Route } from '@angular/router';
import { Page1Component } from './page1/page1.component';
import { Page2Component } from './page2/page2.component';
import { Page2DetailComponent } from
'./page2-detail/page2-detail.component';

```

二、且在 imports 下方 @Module 中間，加入 routes 常數內容如下

甲、這裡建立 page2 時，有在項下多建立 children

乙、若有子項目時，需要多設定傳入的內容，例如設定 path:'id'

```

const routes: Route[] = [
  { path: 'page1', component: Page1Component },
  {
    path: 'page2',
    component: Page2Component,
    children: [{ path: ':id', component: Page2DetailComponent }]
  },
  { path: '**', redirectTo: '/page1', pathMatch: 'full' }
];

```

三、在 Module 加入中加入對應內容

```

@NgModule({
  declarations: [
    AppComponent,Page1Component,Page2Component,Page2DetailComponent
  ],
  imports: [BrowserModule, HttpClientModule,
RouterModule.forRoot(routes)],
  providers: [],
  bootstrap: [AppComponent]
})

```

四、最後就可以在 app.component.html 中設定跳轉內容

甲、也可以像下面在 routerLink 後面可以加入 id

```

<ul>

```

```
<li><a routerLink="/page1">Page 1</a></li>
<li><a routerLink="/page2">Page 2</a></li>
<li><a routerLink="/page2/1">Page 2 Detail with ID 1</a></li>
<li><a routerLink="/page2/2">Page 2 Detail with ID 2</a></li>
<li><a routerLink="/page2/3">Page 2 Detail with ID 3</a></li>
</ul>
```

## 轉址的方式 2

一、在 page2.component.ts 中加入引用

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

二、在 page2.component.ts 寫入 gotoPage1 方法

```
export class Page2Component implements OnInit {
  constructor(private router: Router) {}

  gotoPage1() {
    this.router.navigate(['/page1']);
  }
}
```

三、最後在畫面上寫入 click

```
<button (click)="gotoPage1()">Go to Page1</button>
```



## 表單(開啟 Form 練習檔)

一、在 app.module.ts 中 import 表單

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

二、且在 app.module.ts 下方 import 部份也要加入 FormsModule

```
imports: [  
  FormsModule,  
],
```

三、接下來在 signup.component.html 指定 form 表單

```
<form #f="ngForm">
```

四、且修改姓名 input 內的內容

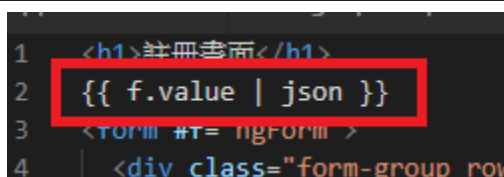
甲、加入 ngModel

乙、給予一個 name，例如此處為 personName

```
<input name="personName" ngModel type="text" class="form-control"  
placeholder="姓名 (必填)" required/>
```

五、將剛剛設定的#f 表單內容，轉換為 Json 格式顯示在畫面上

```
{{ f.value | json }}
```



```
1 <h1>註冊畫面</h1>  
2 {{ f.value | json }}  
3 <form #f="ngForm">  
4 <div class="form-group row
```

## 表單欄位驗證

一、在 signup.component.html 修改姓名的 input 內容

甲、加入#personName="ngModel"

```
<input name="personName" ngModel #personName="ngModel" type="text"  
class="form-control" placeholder="姓名 (必填)" required/>
```

二、並在下面寫下，如果驗證失敗將會顯示 false

```
{{ personName.errors | json }}  
<div>invalid: {{ personName.invalid }}</div>  
<span *ngIf="personName.hasError('required')">必須輸入姓名欄位</span>
```

## 表單全部合法驗證

一、全部要驗證的欄位都設定完後，可以在送出時搭配 f.invalid

```
<button type="submit" class="btn btn-primary" [disabled]="f.invalid">  
送出</button>
```

## 重新表單內容

二、可以透過 `reset` 方法讓表單內容初始化

甲、使用 `f.reset()` 就可以初始化

乙、也可以在裡面放 `Object`，將值初使化代入

```
<button (click)="f.reset({ personName: 'hello' })">reset</button>
```

取得表單資訊

一、在 `signup.component.ts` 中引用 `ViewChild`

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

二、可以用 `ViewChild` 方式取得 `form`

```
export class SignupComponent implements OnInit {  
  //form 盡量不要用 ViewChild 的方式取得  
  @ViewChild('f') form;  
  submitForm(f) {  
    // 使用 ViewChild 取得 Form 實體  
    // console.log(this.form);  
    // 透過傳值的方式取得 Form 實體  
    console.log(f);  
  }  
}
```

三、前端 `signup.component.html` 加入

```
<button type="submit" class="btn btn-primary" [disabled]="f.invalid"  
(click)="submitForm(f)" >
```

或在 `form` 表單上面加入 `click` 事件

```
<form #f="ngForm" (ngSubmit)="submitForm(f)">
```

四、之後將內容顯示在畫面上看是否成功

```
{{ f.form.getRawValue() | json }}
```

選擇核選方塊

一、在核選方塊中加入屬性

甲、加入 `ngModel`

乙、給予一個 `name`

丙、並且指定 `#send=ngModel`

```
<input class="form-check-input" type="checkbox" value=""  
id="defaultCheck1" ngModel name="send" #send="ngModel" />
```

二、之後若要勾選後將某區域顯示/隱藏時可透過 `if` 判斷 `send.value`

```
<ng-container *ngIf="send.value">  
  <div class="form-group row">  
    <label for="staticEmail" class="col-form-label">地址</label>  
    <div ><input type="text" class="form-control" /></div>
```

```
</div>
</ng-container>
```

## 解析 JSON 資料

一、首先看一下資料樣子，但我們希望只得到縣市就好

```
▶ 南投縣: {南投市: "540", 中寮鄉: "541", 草屯鎮: "542", 國姓鄉: "544", 埔里鎮: "545", ...}
▶ 台中市: {中區: "400", 東區: "401", 南區: "402", 西區: "403", 北區: "404", ...}
▶ 台北市: {中正區: "100", 大同區: "103", 中山區: "104", 松山區: "105", 大安區: "106", ...}
▶ 台南市: {永康區: "710", 歸仁區: "711", 新化區: "712", 左鎮區: "713", 玉井區: "714", ...}
▶ 台東縣: {台東市: "950", 綠島鄉: "951", 蘭嶼鄉: "952", 延平鄉: "953", 卑南鄉: "954", ...}
```

二、因此這時候我們需要建立一個變數在 `signup.component.ts`

```
cityTownAreaSource;
```

三、並且製作一個方法

```
loadData() {
  this.http.get('/assets/data/cityarea.json').subscribe(data => {
    console.log(Object.keys(data));
    this.cityTownAreaSource = Object.keys(data);
  });
}
```

四、最後在去令初使化時執行該方法

```
ngOnInit() {
  this.loadData();
}
```

五、而在 `signup.component.html` 畫面上可以使用 `*ngFor` 來撰寫

```
<select id="inputState" class="form-control">
  <option selected>請選擇縣市</option>
  <option *ngFor="let city of cityTownAreaSource">{{city}}</option>
</select>
```

## 使用 ReactiveForms 做表單

一、要在 `app.module.ts` 引用 `ReactiveFormsModule`(`FormsModule` 可以不用)

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

二、並且在下方 `import` 註冊

```
imports: [
  ReactiveFormsModule,
],
```

三、同時到需要用到的地方(如: `signup-advance.component.ts`)內引用

```
import { FormGroup, FormControl, FormBuilder, Validators} from
 '@angular/forms';
```

四、最後就可以繼續將我們要驗證的條件寫入 formData

甲、FormControl 內第一個參數是初始值

乙、後面是放其他條件

```
export class SingupAdvanceComponent implements OnInit {
  formData = new FormGroup({
    name: new FormControl('12333', [Validators.required]),
    email: new FormControl(),
    confirmEmail: new FormControl({ value: '222', disabled: true })
  });
}
```

或下方兩者選一即可

```
formData = this.fb.group({
  name: ['12333', [Validators.required]],
  email: '',
  confirmEmail: [{ value: '222', disabled: true }]
});
```

五、最後在 singup-advance.component.html 畫面部份加入我們設定的表單

甲、透過[formGroup]屬性即可設定

```
<form [formGroup]="formData">
```

六、將 Form 內的驗證規則導入 input 中

甲、在要加驗證的地方使用 formControlName 屬性

乙、等於我們剛剛在 formData 中設定的名稱

```
<input type="text" class="form-control" placeholder="姓名 (必填)"
formControlName="name" />
```

使用 Group 驗證方式 – 例如二次驗證信箱

一、先加入比對方法

甲、可以寫在最外層，但要用 function

```
FormBuilder,  
Validators  
} from '@angular/forms';  
  
function ConfirmEmailCheck(group) {  
  if (group.get('email').value !== group.get('confirmEmail').value) {  
    return { emailNotMatch: true };  
  }  
  return null;  
}  
  
@Component({  
  selector: 'app-singup-advance',  
  templateUrl: './singup-advance.component.html',  
})
```

二、方法內容是比對 email 跟 confirmEmail 是否相等的方法

```
function ConfirmEmailCheck(group) {  
  if (group.get('email').value !== group.get('confirmEmail').value) {  
    return { emailNotMatch: true };  
  }  
  return null;  
}
```

三、在 formData 中寫各種驗證條件

甲、但這邊要注意的是 emailGroup 是可以包含了 email 和 confirmEmail

乙、而驗證條件 validator 就是第二步驟寫的[ConfirmEmailCheck]

```
export class SingupAdvanceComponent implements OnInit {  
  formData = this.fb.group({  
    name: ['', [Validators.required]],  
    emailGroup: this.fb.group(  
      {  
        email: '',  
        confirmEmail: [{ value: '', disabled: false }]  
      },  
      { validator: [ConfirmEmailCheck] }  
    )  
  });  
}
```

四、此時前端 singup-advance.component.html

甲、前端只需要用 formGroupName="emailGroup"來指定 Group

乙、其他內容部份就不用做多餘的更改，因為是在後端 Group 的

```
<div formGroupName="emailGroup">  
  <div class="form-group row">
```

```

        <label for="staticEmail" class="col-sm-2
col-form-label">E-mail</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" placeholder="Email (必
填)" formControlName="email"/>
        </div>
    </div>
    <div class="form-group row">
        <label for="staticEmail" class="col-sm-2 col-form-label"
        >Confirm E-mail</label>
        >
        <div class="col-sm-10">
            <input type="text" class="form-control" placeholder="Email (必
填)" formControlName="confirmEmail"/>
        </div>
    </div>
</div>

```

五、顯示部份就可以使用

```
{{ formData.get('emailGroup').errors | json }}
```