

Angular 13 開發實戰：進階開發篇

深入了解 Angular Form 表單機制

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 支援兩種表單開發模型

以範本為主的表單開發模式 (Template-Driven Form)

[表單範例](#)

- 要匯入 **FormsModule**
`import { FormsModule } from '@angular/forms';`
- 都用 **宣告(Declarative)** 的方式建立表單
- 使用 **ngModel** 指令 (Directive)
- 在範本中**宣告驗證規則**
- 建立 **directive** 進行表單驗證
- 資料為**可變的** (Mutable)
- 對底層 API 增加一層抽象層

以模型為主的表單開發模式 (Model-Driven Form)

[表單範例](#)

- 要匯入 **ReactiveFormsModule**
`import { ReactiveFormsModule } from '@angular/forms';`
- 都用 **編程(Imperative)** 的方式建立表單
- 使用 **formControlName** 屬性
- 在**元件**中**宣告驗證規則**
- 建立 **function/class** 進行表單驗證
- 資料為**不可變的** (Immutable)
- 直接存取底層 API

<https://angular.tw/guide/forms-overview#key-differences>

Angular 內建兩種表單模組

- **FormsModule (預設)**

- Directives
 - NgModel
 - NgModelGroup
 - NgForm

- **ReactiveFormsModule**

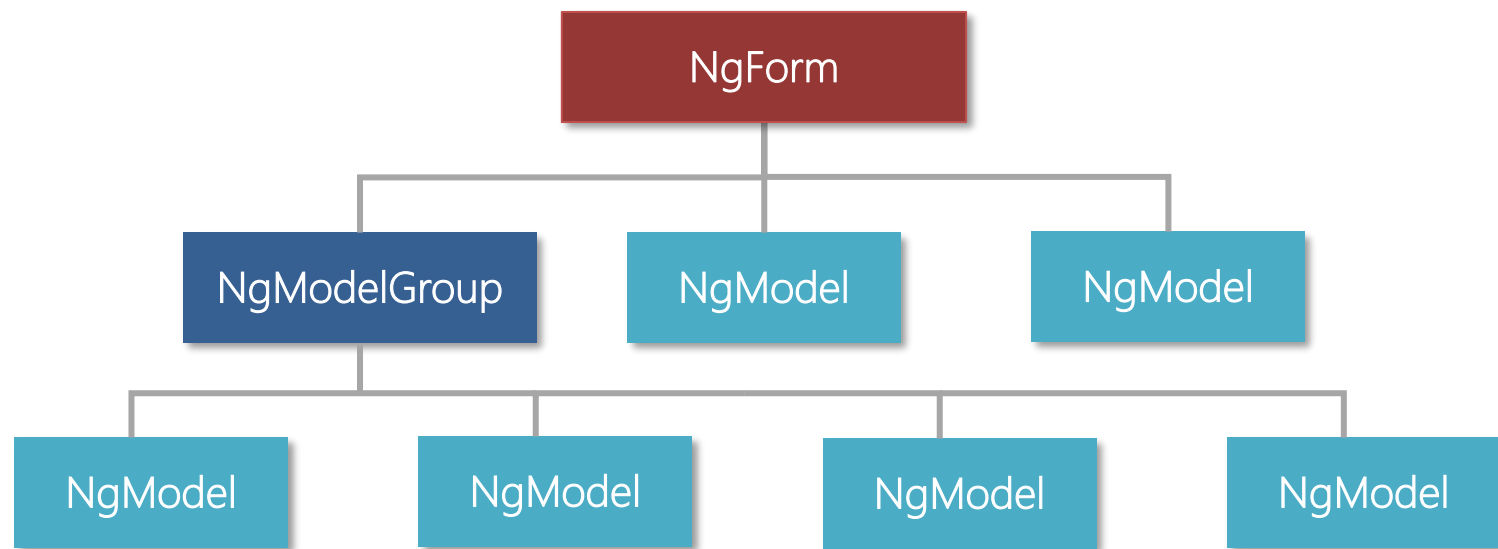
- Directives
 - FormControlDirective
 - FormGroupDirective
 - FormControlName
 - FormGroupName
 - FormArrayName
- Providers
 - FormBuilder



Template-Driven Forms

範本驅動表單開發模式

範本驅動表單物件架構 (表單模型)



重新認識 NgModel 指令 (Directive)

- 初學者的觀念
 - [(ngModel)]="name"
 - 建立一個「**雙向繫結**」綁定元件中的特定屬性
- 重新建立觀念
 - NgModel 主要用來建立一個 **表單控制項** (FormControl) 實體
 - **表單控制項**的主要用途
 - 用來追蹤使用者在表單欄位輸入或選取的**值** (value)
 - 用來追蹤使用者在表單欄位上的**互動狀態** (pristine, dirty, touched, untouched)
 - 用來追蹤表單欄位的**驗證狀態** (errors)
 - 用來維持與元件中 Model 進行同步 (**單向屬性繫結**或**雙向繫結**)

如何建立表單控制項實體 (instance)

- 不綁定直接建立實體 **ngModel**

```
<input name="username" ngModel>
```

- 單向綁定 (one-way binding) 使用 **[ngModel]**

```
<input name="username" [ngModel]="username">
```

- 雙向綁定 (two-way binding) 使用 **[(ngModel)]**

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

如何建立並取得表單控制項實體

- 不綁定直接建立實體 **ngModel**

```
<input name="username" ngModel  
      #mUsername="ngModel">
```

- 單向綁定 (one-way binding) 使用 **[ngModel]**

```
<input name="username" [ngModel]="username"  
      #mUsername="ngModel">
```

- 雙向綁定 (two-way binding) 使用 **[(ngModel)]**

```
<input name="username" [(ngModel)]="username"  
      #mUsername="ngModel">
```


如何賦予表單欄位預設值

- 直接跟元件中的屬性 (Model) 進行資料綁定即可
 - 單向綁定 (one-way binding) 使用 `[ngModel]`

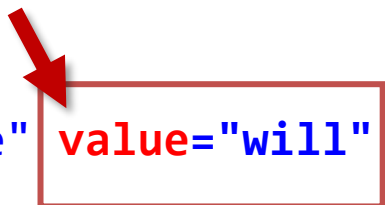
```
<input name="username" [ngModel]="username">
```

- 雙向綁定 (two-way binding) 使用 `[(ngModel)]`

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

- 錯誤示範

```
<input name="username" value="will" [ngModel]="username">
```



表單欄位名稱的必要性

- 沒有表單的輸入欄位（沒有 `<form>` 包覆的欄位）

```
<input type="text" class="form-control rounded"  
      [(ngModel)]="username" #f1="ngModel">
```

- 放在 `<form>` 裡面的輸入欄位則必須要有 `name` 屬性

```
<input type="text" class="form-control rounded"  
      ngModel #f1="ngModel" name="username">
```

```
<input type="text" class="form-control rounded"  
      ngModel #f1="ngModel"  
      [ngModelOptions]="{standalone: true}">
```

內建的驗證器 (Validators)

- FormsModule 預設提供的 [Validators](#) 有：

驗證屬性	用途說明
required	表單控制項為必填欄位
minlength	表單控制項的值不可少於幾個字元
maxlength	表單控制項的值不可大於幾個字元
pattern	表單控制項的值必須符合給予的 Regex 正規表示式
email	表單控制項的值必須符合 E-mail 樣式

```
<form>
  <input name="title"      ngModel required>
  <input name="summary"    ngModel minlength="20">
  <input name="longdesc"   ngModel maxlength="300">
  <input name="mobile"     ngModel pattern="\d{4}-\d{6}">
  <input name="email"      ngModel email>
</form>
```

表單驗證狀態的 CSS 樣式處理

- **表單驗證機制**會自動根據**驗證狀態**提供 CSS 樣式類別
- 會根據**驗證狀態**提供 CSS 樣式類別的 Directives 有：
 - NgModel
 - NgModelGroup
 - NgForm

範例程式

欄位狀態	true	false
欄位是否曾經被碰過 (f.touched / f.untouched)	ng-touched	ng-untouched
欄位值是否有改變過 (f.dirty / f.pristine)	ng-dirty	ng-pristine
是否有通過欄位驗證 (f.valid / f.invalid)	ng-valid	ng-invalid

常見的表單驗證 CSS 樣式

- 有效的欄位

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

- 無效的欄位 (排除 form 標籤)

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

在範本中取得驗證結果的方法

- 直接在輸入欄位上套用**驗證屬性** (validation attributes)
`<input name="title" [ngModel]="title" required>`
- 如果欄位**驗證成功**
 - NgModel 物件的 **errors** 屬性會回傳 **null**
- 如果欄位**驗證失敗**
 - NgModel 物件的 **errors** 屬性會包含所有驗證狀態 (如下圖示)
 - 取得必填的錯誤狀態：`f1['errors']?.required` (?只能用在範本裡)

Rounded Corners

```
<input type="text" name="title" [ngModel]="title" #f1="ngModel">
```



```
{  
  "required": true      {{ f1.errors | json }}  
}
```

取得 NgModel 實體的常見屬性

屬性名稱	型別	用途說明
name	string	欄位名稱
value	any	欄位值
valid	boolean	有效欄位 (通過欄位驗證)
invalid	boolean	無效欄位 (欄位驗證失敗)
errors	{[key: string]: any}	當出現無效欄位時，會出現的錯誤狀態
dirty	boolean	欄位值是否曾經更動過一次以上
pristine	boolean	欄位值是否為原始值 (未曾被修改過)
touched	boolean	欄位曾經經歷過 focus 事件
untouched	boolean	欄位從未經歷過 focus 事件
disabled	boolean	欄位設定為 disabled 狀態
enabled	boolean	欄位設定為 enabled 狀態 (預設啟用)
formDirective	NgForm	取得目前欄位所屬的 NgForm 表單物件
valueChanges	EventEmitter	可用來訂閱欄位值變更的事件
statusChanges	EventEmitter	可用來訂閱欄位狀態變更的事件

認識 NgForm 指令 (Directive)

- 重要觀念
 - [NgForm](#) 主要用來建立一個 **最上層** 的 **表單群組** ([FormGroup](#)) 實體
 - **表單群組** ([FormGroup](#)) 的主要用途
 - 用來追蹤**群組內**所有**表單控制項**的**欄位值**與**驗證狀態**
 - 也可以從**表單群組**物件實體取得所有**表單控制項**的**欄位值**與**驗證狀態**
 - FormsModule 預設會將所有的 `<form>` 標籤宣告為 NgForm 指令
 - 因此 `<form>` 標籤不需要額外宣告 **ngForm** 指令 (Directive)
 - 要從範本取得 **ngForm** 物件實體，可以透過**範本參考變數**完成
`<form name="form1" #f="ngForm">`
 - 比較 **ngModel** 的用法（建立表單控制項一定要用 **ngModel** 宣告過）
`<input name="txt" ngModel #mTxt="ngModel">`

使用 NgForm 的 ngSubmit 事件

- 搭配 **type="submit"** 按鈕與 (**ngSubmit**) 事件 (建議用法)

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">  
  <input name="account" ngModel required>  
  <button type="submit">Submit</button>  
</form>
```

- 搭配 **type="button"** 按鈕與 (**click**) 事件 (不會觸發 ngSubmit 事件)

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">  
  <input name="account" ngModel>  
  <button type="button" (click)="onSubmit(f)">  
    Submit  
  </button>  
</form>
```

表單事件範例

取得 NgForm 實體的常見屬性

屬性名稱	型別	用途說明
value	any	表單內所有欄位值 (以物件型態呈現)
valid	boolean	所有欄位是否皆為有效欄位
invalid	boolean	是否有任意一個欄位為無效欄位
errors	請注意：在 NgForm 物件下有此屬性，但無資料！	
dirty	boolean	任意欄位是否曾經更動過一次以上
pristine	boolean	任意欄位欄位是否為原始值 (未曾被修改過)
touched	boolean	任意欄位曾經經歷過 focus 事件
untouched	boolean	任意欄位從未經歷過 focus 事件
disabled	boolean	所有欄位設定為 disabled 狀態
enabled	boolean	任意欄位為 enabled 狀態就為 true
valueChanges	EventEmitter	可用來訂閱任意欄位值變更的事件
statusChanges	EventEmitter	可用來訂閱任意欄位狀態變更的事件
submitted	boolean	判斷該表單是否經歷過 ngSubmit 事件

取得表單所有欄位值

- 範例表單

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
  <input name="account" ngModel #mAccount>
  <button type="submit">Submit</button>
</form>
```

- 取得表單欄位值的注意事項

- 預設透過 **f.value** 即可取得表單所有欄位值 (物件型態)
- 如果表單中有**部分欄位**被設定為 **disabled** 狀態
 - 在 **f.value** 物件將會找不到 **disabled** 欄位的值
- 如果表單中**所有欄位**被設定為 **disabled** 狀態
 - 在 **f.value** 物件將會顯示所有欄位的值 (包含 **disabled** 的欄位)
 - 但此時的 **f.valid** 屬性將會改為 **false**

認識 NgModelGroup 指令 (Directive)

- 主要用途
 - 在現有表單 (NgForm) 建立額外的欄位群組
 - 只能使用在有 **<form>** 標籤的表單範圍內宣告
- 重要觀念
 - [NgModelGroup](#) 主要用來建立一個 表單群組 ([FormGroup](#)) 實體
 - 表單群組 ([FormGroup](#)) 的主要用途
 - 用來追蹤群組內所有表單控制項的欄位值與驗證狀態
 - 用來取得群組內所有表單控制項的欄位值與驗證狀態
- 使用範例

```
<div ngModelGroup="group1" #group1="ngModelGroup">  
  <input name="fName" [(ngModel)]="model.firstName">  
  <input name="lName" [(ngModel)]="model.lastName">  
</div>
```

取得 NgModelGroup 實體的常見屬性

屬性名稱	型別	用途說明
value	any	群組內所有欄位值 (以物件型態呈現)
valid	boolean	群組內所有欄位是否皆為有效欄位
invalid	boolean	群組內是否有任意一個欄位為無效欄位
errors	請注意：在 NgModelGroup 物件下有此屬性，但無資料！	
dirty	boolean	任意欄位是否曾經更動過一次以上
pristine	boolean	任意欄位欄位是否為原始值 (未曾被修改過)
touched	boolean	任意欄位曾經經歷過 focus 事件
untouched	boolean	任意欄位從未經歷過 focus 事件
disabled	boolean	所有欄位設定為 disabled 狀態
enabled	boolean	任意欄位為 enabled 狀態就為 true
valueChanges	EventEmitter	可用來訂閱任意欄位值變更的事件
statusChanges	EventEmitter	可用來訂閱任意欄位狀態變更的事件



Model-Driven Forms (Reactive Forms)

模型驅動表單開發模式

認識 Reactive Form 表單開發模式

- 模型驅動表單 (**Model-driven Forms**) 又稱 **Reactive Forms**
 - 在同一個 Angular 應用程式中可以混合使用兩種模型
- 完全由 **表單模型** (Model) 集中管理/設定所有表單欄位
 - 從元件內部進行**表單模型**與**資料模型**的對應與設計
 - 給予欄位**預設值**與**驗證規則**
- 範本還是需要跟元件模型進行綁定
 - AppModule 需匯入 **ReactiveFormsModule** 模組才有以下 Directives
 - FormControlDirective [formControl]="ctrl1"
 - FormGroupDirective [formGroup]="form"
 - FormControlName formControlName="title"
 - FormGroupName formGroupName="group1"
 - FormArrayName formArrayName="array1"

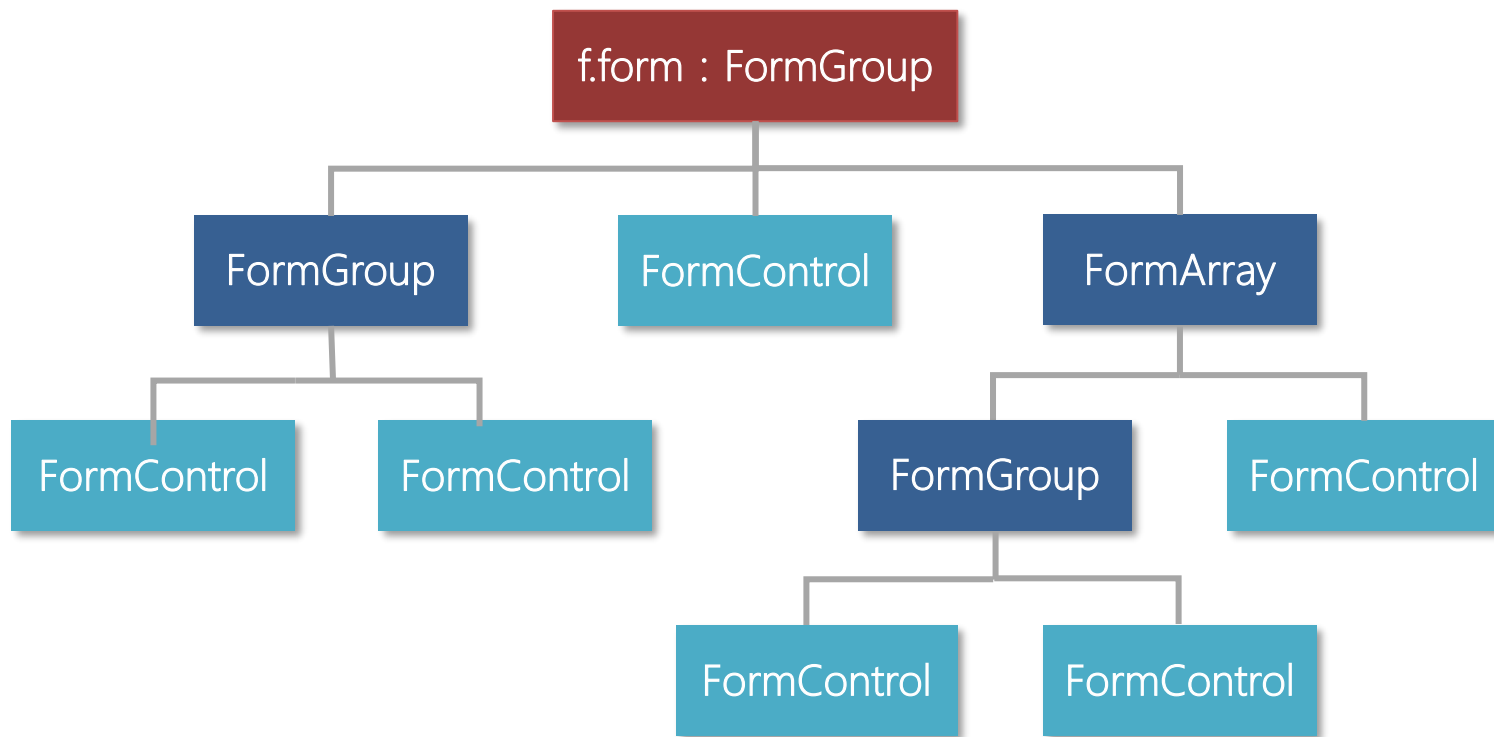
匯入 ReactiveFormsModule 模組

```
app.module.ts x
1 import { BrowserModule } from '@angular/platform-browser';
2 import { ReactiveFormsModule, FormsModule } from '@angular/forms';
3 import { NgModule } from '@angular/core';
4 import { HttpClientModule } from '@angular/http';
5
6 import { AppComponent } from './app.component';
7
8
9 @NgModule({
10   declarations: [
11     AppComponent,
12   ],
13   imports: [
14     BrowserModule,
15     FormsModule,
16     ReactiveFormsModule,
17     HttpClientModule
18   ],
19   providers: [],
20   bootstrap: [AppComponent]
21 })
22 export class AppModule { }
23
```


認識幾個 Reactive Forms 重要類別

- [AbstractControl](#)
 - 這是一個抽象類別。它是 Reactive Forms 架構中 3 個表單類別共同的抽象基底類別 (abstract base class)，這 3 個表單控制項就是 FormControl, FormGroup 與 FormArray 類別。這裡定義所有表單控制項共用的屬性與方法，有一部分屬性為 Observable 型別
- [FormControl](#)
 - 用來追蹤某一個表單控制項的欄位值與驗證狀態
- [FormGroup](#)
 - 用來追蹤一群表單控制項的欄位值與驗證狀態
 - 該群組下的欄位值包含所有子控制項與子群組的所有集合
 - 該群組會以「物件」的方式進行群組（欄位之間沒有順序性，索引值為字串）
- [FormArray](#)
 - 用來追蹤一群表單控制項的欄位值與驗證狀態
 - 該群組下的欄位值包含所有子控制項與子群組的所有集合
 - 該群組會以「陣列」的方式進行群組（欄位之間的索引值為數值）

模型驅動表單物件架構 (表單模型)



使用 FormBuilder 建立表單模型

- 宣告表單模型與注入 FormBuilder 服務元件

```
form: FormGroup;
```

```
constructor(private fb: FormBuilder) {  
}
```

- 建立表單模型物件（頂層表單群組 + 子表單控制項）

```
ngOnInit() {  
  this.form = this.fb.group({  
    title: 'This is title'  
  });  
}
```

表單控制項
(FormControl)

欄位預設值

設計範本的表單介面

- 重要觀念
 - 通常一個表單都擁有多個欄位
 - 多個欄位意味著需要一個群組
 - 所以表單最上層就是一個 FormGroup 表單群組
 - 模型驅動表單欄位不需要 **name** 屬性 (在範本驅動表單則為必要條件)
- 表單介面最上層必須套用 **[formGroup]** 繫結表單群組物件

```
<form [formGroup]="form" (ngSubmit)="submit()">
```

```
  <input type="text" formControlName="title">
```

```
</form>
```

比較兩種不同的表單模型建立方法

- 使用 FormBuilder 建立表單元件

```
this.form = this.fb.group({  
  name: this.fb.group({  
    first: 'Will',  
    last: 'Huang'  
  })  
});
```

- 使用 FormControl, FormGroup, FormArray 建立表單元件

```
this.form = new FormGroup({  
  name: new FormGroup({  
    first: new FormControl('Will'),  
    last: new FormControl('Huang')  
  })  
});
```

設計多層表單群組的表單 (模型部分)

- 模型物件

```
this.form = this.fb.group({  
  
  title: 'This is title',  
  
  name: this.fb.group({  
    firstName: 'Will',  
    lastName: 'Huang'  
  })  
  
});
```

設計多層表單群組的表單 (範本部分)

- 範本介面

```
<form [formGroup]="form">
  <input type="text" formControlName="title">
  <div formGroupName="name">
    <input type="text" formControlName="firstName">
    <input type="text" formControlName="lastName">
  </div>
</form>
```

```
<pre>{{ form.value | json }}</pre>
```

設計多層表單陣列的表單 (模型部分)

- 模型物件

```
this.form = this.fb.group({  
  title: 'This is title',  
  name: this.fb.array([  
    this.fb.control('Will 1', Validators.required),  
    this.fb.control('Will 2', Validators.required)  
  ])  
});
```

- 注意事項

- 傳入 `this.fb.array` 的是一個陣列
- 陣列內的 `formControlName` 將會是一個陣列索引值 (數值)
- 陣列內每個元素可以是任意表單物件
(`FormControl`, `FormGroup`, `FormArray`)

設計多層表單陣列的表單 (範本部分)

- 範本介面
 - 以下範例可以看出，**表單陣列群組**非常適合用來建置**動態表單**

```
<form [formGroup]="form">
  <input type="text" formControlName="title">
  <div formArrayName="name">
    <input type="text"
      *ngFor="let item of form.controls.name.controls;
        let i=index"
      [formControlName]="i">
  </div>
</form>
```

```
<pre>{{ form.value | json }}</pre>
```

套用欄位驗證

- 匯入驗證器 ([Validators](#))

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

- 套用預設值與驗證器

```
this.form = this.fb.group({  
  title: ['The Will Will Web', [  
    Validators.required,  
    Validators.minLength(3)  
  ]  
});
```

- 範本介面

```
<input type="text" formControlName="title">
```

表單物件的常見屬性

- [FormControl](#) 物件
 - 屬性繼承自 `AbstractControl` 類別
 - 屬性與 `NgModel` 完全一樣
 - 有許多自訂的方法 (詳見 [FormControl API](#) 文件)
- [FormGroup](#) 物件
 - 屬性繼承自 `AbstractControl` 類別
 - 屬性與 `NgModelGroup` 完全一樣
 - 有許多自訂的方法 **可動態增減** 子控制項 (詳見 [FormGroup API](#) 文件)
- [FormArray](#)
 - 屬性繼承自 `AbstractControl` 類別
 - 屬性與 `NgModelGroup` 完全一樣
 - 有許多自訂的方法 **可動態增減** 子控制項 (詳見 [FormArray API](#) 文件)

資料模型 v.s. 表單模型

- 名詞定義
 - 資料模型
 - 通常代表著從伺服器回傳的資料模型物件
 - 表單模型
 - 由 FormControl, FormGroup, FormArray 組成的表單模型物件
 - 可透過 FormBuilder 來建立，也可以自行 new 出新物件
- 重要觀念
 - 通常我們會將**資料模型**複製到**表單模型**中
 - 這意味著開發人員**必須了解資料模型如何對應到表單模型**中
 - 在表單上所進行的**資料異動**，只會改到**表單模型**，不會動到**資料模型**
 - **表單模型與資料模型**之間的關係**不需要 1 對 1 對應**
 - 通常我們只需要將資料模型的部分資料對應到表單上
 - 若資料模型與表單模型接近，兩者之間的關係會更加清楚（可維護性）

setValue vs. patchValue vs. reset

- 將資料模型寫入到表單模型中（會覆寫完整物件）
 - 傳入 `setValue()` 的物件，必須完全與 `FormGroup` 中的物件結構相同！
- 將資料模型部分更新到表單模型中（僅更新部分屬性）
 - 傳入 `patchValue()` 的物件，如果比對不到屬性，不會出現錯誤（會自動忽略）！

```
this.form.setValue({  
  name:    this.hero.name,  
  address: this.hero.addresses[0] || new Address()  
});
```

```
this.form.patchValue({  
  name: this.hero.name  
});
```

重置表單內容 (reset)

- 重置表單的 reset() 方法
 - 表單控制項或表單群組的狀態會被重置為 **pristine**
 - 表單控制項或表單群組的狀態會被重置為 **untouched**

- 清空所有內容 (欄位值都會變成 **null** 值)

```
this.form.reset();
```

- 重置為預設內容

- 傳入 reset() 的物件，如果比對不到屬性，不會出現錯誤(會自動忽略)！

```
this.form.reset({  
  firstName: 'Will',  
  lastName: 'Huang'  
});
```

FormArray 與 FormGroup 開發技巧

- 善用 [map\(\)](#) 函式，將資料模型轉換成表單陣列物件

```
setAddresses(addresses: Address[]) {  
    // 先將資料模型轉換為表單群組（每筆資料都轉換成一個群組）  
    const addressFGs =  
        addresses.map(address => this.fb.group(address));  
    // 再將表單群組加入到表單陣列中  
    const addressFormArray = this.fb.array(addressFGs);  
    // 最後將表單陣列取代 form 下的 'secretLairs' 表單控制項  
    this.form.setControl('secretLairs', addressFormArray);  
}
```

取得表單群組或表單陣列中的控制項

- 取得子表單控制項

```
return this.form.get('email') as FormControl;
```

- 取得子表單陣列

```
return this.form.get('group1') as FormArray;
```

- 取得子表單群組

```
return this.form.get('array1') as FormGroup;
```

- 取得多層群組下的表單控制項

```
return this.form.get('group1.email') as FormControl;
```


自訂錯誤驗證器

- 驗證器範例程式

```
export function forbiddenNameValidator(control: AbstractControl) {  
  const nameRe = /Will/;  
  const name = control.value;  
  const no = nameRe.test(name);  
  return no ? { 'forbiddenName': true } : null;  
};
```

- 驗證器回傳型別

- **驗證通過**要回傳 `null`，**驗證不通過**要回傳一個**驗證錯誤物件**
- 驗證錯誤物件 (validation error object)
 - 通常都包含一個屬性
 - 屬性名稱會是驗證控制器的名稱
 - 屬性的值則是任意值
 - 通常為布林值/也可設成當下的值或錯誤訊息

自訂錯誤驗證器 (帶參數的驗證器)

- 驗證器範例程式

```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {  
  return (control: AbstractControl): { [key: string]: any } => {  
    const name = control.value;  
    const no = nameRe.test(name);  
    return no ? { 'forbiddenName': true } : null;  
  };  
}
```

- 驗證器回傳型別

[範例程式](#)

- **驗證通過**要回傳 `null`, **驗證不通過**要回傳一個**驗證錯誤物件**
- 驗證錯誤物件 (validation error object)
 - 通常都包含一個屬性
 - **屬性名稱**會是**驗證控制器的名稱**
 - **屬性的值**則是**任意值**
 - 通常為**布林值**/也可設成**當下的值**或**錯誤訊息**

自訂錯誤驗證器 (帶參數的驗證器)

- 驗證器範例程式

```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {  
  return function(control: AbstractControl) {  
    const name = control.value;  
    const no = nameRe.test(name);  
    return no ? { 'forbiddenName': true } : null;  
  };  
}
```

- 驗證器回傳型別

[範例程式](#)

- **驗證通過**要回傳 `null`, **驗證不通過**要回傳一個**驗證錯誤物件**
- 驗證錯誤物件 (validation error object)
 - 通常都包含一個屬性
 - **屬性名稱**會是**驗證控制器的名稱**
 - **屬性的值**則是**任意值**
 - 通常為**布林值**/也可設成**當下的值**或**錯誤訊息**

撰寫完整的表單驗證器 (Reactive + Template)

- 先建立 Directive 元件
 - `ng g directive twid-validator`
- 再透過 Code Snippet 產生程式碼骨架
 - `ng-validator`
- 加入主要的驗證邏輯
 - **驗證成功**回傳 `null`, **驗證失敗**回傳 `object`
 - [taiwan-id-validator2](#) (台灣身分證字號驗證器)
- 加入 directive 到範本中的 `ngModel` 表單欄位
 - `twid`

自訂非同步錯誤驗證器 (AsyncValidator)

- 每一個表單控制項的 status 屬性包含 4 種可能的狀態：
 - VALID 欄位是**有效的**
 - INVALID 欄位是**無效的**
 - PENDING 欄位**正在進行檢查**中（非同步驗證器才有這個狀態）
 - DISABLED 欄位是**無效的**
- 相關連結與程式範例
 - [Angular - AsyncValidator](#)
 - [Creating Angular Synchronous and Asynchronous Validators for Template Validation](#)
 - [Implementing Async Validators in Angular Reactive Forms](#)
 - [AsyncValidator interface](#)

如何選擇兩種截然不同的表單模型

- 範本驅動表單開發模式 (Template-Driven Forms)
 - 採用**宣告**的方式建立表單 (**較為簡單**) (維護較為容易)
 - 適合**固定欄位數量**的表單
 - 會員註冊、登入、線上下單、修改會員資料、...
 - **無法**對表單進行**單元測試** (Unit Testing)
 - 因為所有表單邏輯都定義在範本中，無法對元件進行單元測試
 - 只能透過 E2E 的方式測試表單
- 模型驅動表單開發模式 (Model-Driven Forms)
 - 採用**編程**的方式建立表單 (**較為繁瑣**) (程式碼維護較為麻煩)
 - 適合**動態欄位數量**的表單 (動態表單)
 - 由後台定義的動態問卷系統、變動選項的投票系統、...
 - 可以直接針對表單進行**單元測試** (Unit Testing)
 - 因為所有表單邏輯都已宣告為物件類型的表單模型

相關連結

- [Angular - Introduction to forms](#)
- [Angular - Template-driven forms](#)
- [Angular - Reactive Forms](#)
- [Angular - Form Validation](#) / [Async Validation](#)
- [Angular - Dynamic Forms](#)
- [\[Angular\] Custom Validator](#)
- [Custom Validators in Angular by thoughtram](#)
- [Custom **Async Validators** in Angular](#)



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



多奇·教育訓練

THANK YOU!

Q&A