**Data binding 資料繫結**

<https://angular.tw/guide/architecture-components#data-binding>

主要分三種:內嵌{{}},屬性[],事件()

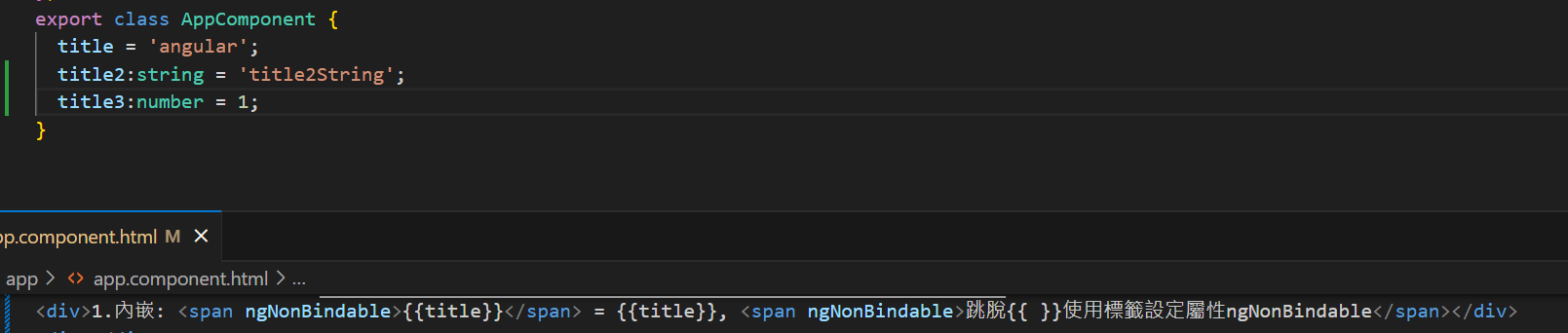


1. 內嵌(interpolation) {{value}} :

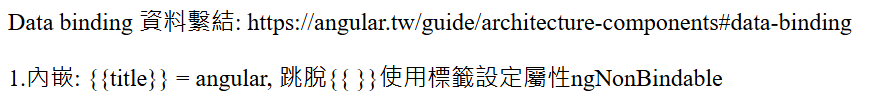
執行時,把屬性value的值帶入{{value}}替換

備註:如果本身要使用{{ xxx }} 符號當字串,

需要於標籤屬性中增加ngNonBindable



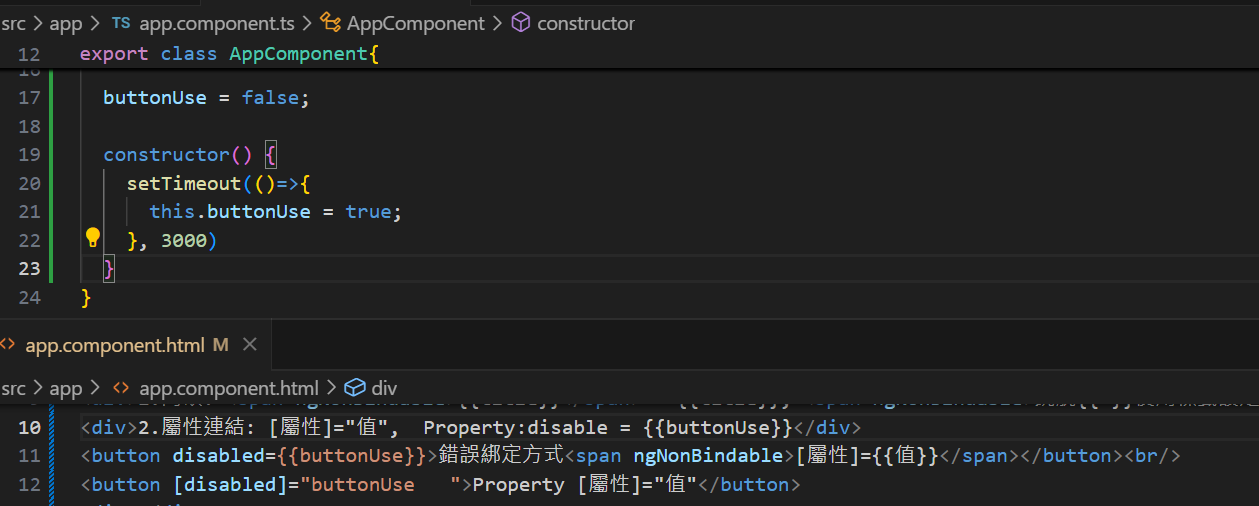
網頁顯示:



1. 屬性繫結(property binding) [property] = "value" :

https://angular.tw/guide/property-binding

執行時,屬性值的變化直接影響Property。ex: src,disable,….都是標籤的Property



剛進入網頁顯示:



進入網頁三秒後:



1. 屬性繫結(attribute binding) [attr.property] = "value":

與property binding差異在於property是使用標準html的屬性時使用,而attribute binding使用在非標準ex: aria-\*(無障礙),data-\*(自訂義儲存) => [attr.data-xxx]="xxx"

1. 類別繫結(class binding) [class.Class名稱]="真假值" 或 [class]=" classExpression":

**單一類別 [class.Class名稱] = "真假值":**

依真假值判斷是否加上class

ex:<style>red{color:red}</style>…<div [class.red]="hasColor"></div>

extend class XXXComponent{hasColor:boolean = true…}

此如果值為true會顯示class,如果為false不會顯示class

**多個類別 [class]="classExpression":**

* 空格分隔字串名稱:

ex: <div [class]="clazz">xxx</div>

export class ClassCompontent{

clazz: string = 'classA classB classC';

}

* 用物件,其中key為class名稱,value為真假值

ex: <div [class]="clazzObj">xxx</div>

export class ClassCompontent{

clazzObj = {

classA : true,

classB : false

};

}

* 用陣列:

ex: <div [class]="clazzArr">xxx</div>

export class ClassCompontent{

classArr: string[] = ['classA', 'classB', 'classC'];

}

1. 樣式繫結(binding to a single style):

**單一style [style.Css屬性] = "值":**

ex: <div [style.color]="textColor">xxx</div>

export class StyleCompontent{

textColor: string = 'red';

}

**多個style [style]="styleExpression":**

* + 空格分隔字串名稱:

ex: <div [style]="styleString">xxx</div>

export class StyleCompontent{

styleString: string = 'font-size: 1.5em; text-decoration: underline; color: blue;';

}

* + 用物件,其中key為class名稱,value為真假值

ex: <div [style]="styleObj">xxx</div>

export class StyleCompontent{

styleObj = {

border: '1px solid #999',

borderRadius: '30px',

backgroundColor: '#ccc'

}

}

備註:如果css屬性有使用 – 要換掉並且改為駝峰

* + style不能使用陣列宣告和class不同

1. 事件繫結(event binding) (event) = "方法(..)" :

當發生event時呼叫TypeScript的方法

ex: <button (click)="onClick()">xxx</button>

export class EventCompontent{

onClick() {

alert('click!!');

}

}

如果要自訂事件需要使用@ Directive宣告

ex:自訂義myClick這個事件

<button type="button" (myClick)="clickMessage=$event" clickable>xxx</button>

<div>{{clickMessage}}</div>

class內容見Directive

1. 被動事件繫結 passive events binding (event.pasive)="'方法(..)":

…目前感受不太到和直接使用scroll差異….

版本舊沒有pasive要手動處理

提高效能,適合高頻觸發ex:滾動

非被動事件,會預設使用event.preventDefault()

被動事件,不會使用event.preventDefault()

ex: 創src/zone-flags.ts

編輯檔案加入(window as any)['\_\_zone\_symbol\_\_PASSIVE\_EVENTS'] = ['scroll'];

在main.ts中import './zone-flags.ts';import 'zone.js';

<div (scroll)="onScroll($event)">xxx</div>

export class PasiveComponent{

onScroll(event : Event): void {

console.log('scroll', event);

}

}

1. 鍵盤事件繫結 keyboard events binding:

鍵值表: <https://developer.mozilla.org/en-US/docs/Web/API/UI_Events/Keyboard_event_key_values>

鍵碼表: <https://developer.mozilla.org/en-US/docs/Web/API/UI_Events/Keyboard_event_code_values>

* + 使用鍵值表(鍵盤事件.按鍵)="方法(..)" :單一鍵觸發

ex: <input (keydown.v)="onKeydown($event)" placeholder="keydownV"/>

export class KeyEventComponent {

onKeydown(event : Event):void {

console.log('keydown', event);

}

}

* + 組合鍵: (鍵盤事件.按鍵.按鍵)="方法(..)" : 同時多個鍵按時觸發

ex: <input (keydown.shift.v)="onKeydown($event)" placeholder="keydown Shift + V"/>

export class KeyEventComponent {

onKeydown(event : Event):void {

console.log('keydown', event);

}

}

* + 使用鍵碼表(鍵盤事件.code.按鍵.按鍵)="方法(..)" : 使用code方式

ex: <input (keydown.code.KeyV)="onKeydown($event)" placeholder="keydown V"/>

export class KeyEventComponent {

onKeydown(event : Event):void {

console.log('keydown', event);

}

}

1. 雙向繫結(two-way binding) [(小孩屬性)]="父屬性" 或 [(ngModel)="屬性":

第一種方式:

元件自己和html互動,使用ngModel

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

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

class中宣告屬性即可(上面範例為宣告title)

第二種方式: (需要先看@Input和@Output這兩個裝飾器)

實際上為[小孩屬性]="父屬性",(小孩事件)="父屬性=$event"

小孩屬性名稱:input如果是xxx,output需要設定xxxChange

ex: 父元件的html

<two-way-binding [(size)]="fontSize"></two-way-binding>

<div [style.font-size.px]="fontSize">parent size: {{fontSize}}px</div>

父class

fontSize=16;

小孩元件的html

<div>

    <div>

        <button type="button" (click)="dec()" title="smaller">-</button>

        <span [style.font-size.px]="size">字體大小,size:{{size}} px</span>

        <button type="button" (click)="inc()" title="bigger">+</button>

    </div>

</div>

小孩class (output一定要是xxxChange)

@Input()  size!: number | string;

@Output() sizeChange = new EventEmitter<number>();

resize(delta : number) {

//"+"this.size =>string to number

this.size = Math.min(40, Math.max(8, +this.size + delta));

this.sizeChange.emit(this.size);

}

dec(){this.resize(-1);}

inc(){this.resize(+1);}