Tema 4 - Angular

Anexo
Directivas.

Desarrollo web en entorno cliente
IES Pere Maria Orts I Bosch

Índice

Directivas

Las directivas son elementos o atributos creados por nosotros en Angular y que añaden o modifican funcionalidad a la plantilla HTML. Hay 3 tipos de directivas:

Directivas de componente

Una directiva de componente se define por la propiedad selector en el decorador @Component. Cuando un elemento con el mismo nombre se crea en la plantilla HTML, se instancia la clase del componente asociado y su plantilla se inserta dentro del mismo. Estas directivas son las que hemos estado creando hasta ahora.

```
@Component({
  selector: 'star-rating',
  templateUrl: './star-rating.component.html',
  styleUrls: ['./star-rating.component.css']
})
export class StarRatingComponent implements OnInit {
  ...
}
```

```
<star-rating *ngIf="product" [rating]="product.rating"
             (ratingChanged)="changeRating($event)"></star-rating>
```

Directivas de atributo

Las directivas de atributo son atributos creados para modificar el comportamiento de un elemento en la plantilla. Las directivas de Angular NgClass y NgStyle son ejemplos de esto.

Vamos a crear una directiva de este tipo. En este caso servirá para cambiar el color de fondo de un elemento cuando hagamos click en ella. Para generar la directiva, creamos un directorio llamado directives y ejecutamos dentro:

```
ng g directive setColor
```

```
@Directive({
  selector: '[setColor]'
})
export class SetColorDirective {
  constructor() { }
}
```

La clase de la directiva que hemos creado (SetColorDirective) debe incluirse en la sección declarations del módulo de la aplicación (o en otro módulo como veremos más adelante). Una vez creada se la podemos aplicar a cualquier elemento HTML. Por ejemplo:

```
<h1 [setColor]="'yellow'">{{title}}</h1>
```

Cuando colocamos el atributo de la directiva (en este caso [setColor]), esta se aplica al elemento en cuestión. Podemos pasarle parámetros extras y producir

eventos igual que hacíamos con los componentes anidados (@Input, @Output). Para procesar los eventos ('click', 'mouseenter', etc.) sobre el elemento que tiene la directiva asignada, usamos el decorador @HostListener.

app.component.ts

```
@Component({
  ...
})
export class AppComponent {
  title = 'app works!';
  color = 'yellow';
}
```

app.component.hmtl

```
<h1 [setColor]="color">{{title}}</h1>
<select [(ngModel)]="color">
  <option value="yellow">Yellow</option>
  <option value="red">Red</option>
  <option value="#06F">Blue</option>
</select>
```

set-color.directive.ts

```
@Directive({
  selector: '[setColor]'
})
export class SetColorDirective {
  private isSet: boolean;
  @Input('setColor') color: string;

  constructor(private elem: ElementRef) {
    this.isSet = false;
  }

  @HostListener('click') onClick() {
    if(this.isSet) {
      this.elem.nativeElement.style.backgroundColor = "";
    } else {
      this.elem.nativeElement.style.backgroundColor = this.color;
    }
    this.isSet = !this.isSet;
  }
}
```

https://angular.io/docs/ts/latest/guide/attribute-directives.html

Directivas estructurales

Las directivas estructurales se utilizan para modificar la estructura del DOM. Normalmente indican si un elemento debe aparecer en el DOM o no. No es muy común crear directivas estructurales, pero puede sernos bastante útil.

NgSwitch

Ejemplos de este tipo de directivas son NgIf → *ngIf y NgFor → *ngFor. Existe otra llamada NgSwitch → [ngSwitch] que no hemos visto y que permite crear una estructura switch en la plantilla HTML.

```
<span [ngSwitch]="property">
  <span *ngSwitchCase="'val1'">Value 1</span>
  <span *ngSwitchCase="'val2'">Value 2</span>
  <span *ngSwitchCase="'val3'">Value 3</span>
  <span *ngSwitchDefault>Other value</span>
</span>
```

NgIf (else)

La directivas estructurales necesitan del elemento <ng-template> para funcionar. Sin embargo, con el asterisco(*) delante no es necesario ponerlo, ya que Angular lo genera por nosotros. Este elemento <ng-template> desaparece cuando la directiva se procesa. Estas dos formas de escribir ngIf son equivalentes:

```
<div *ngIf="condition" >{{somevalue}}</div>
```

```
<ng-template [ngIf]="condition">
  <div>{{somevalue}}</div>
</ng-template>
```

Si queremos crear un bloque else, necesitamos otro elemento ng-template, sólo que esta vez no lo podemos omitir, ya que está fuera del elemento al que afecta la directiva ngIf. Necesitamos crear una referencia para este nuevo elemento, poniendo la almohadilla (#referencia), y usarla para definir la condición else.

```
<div *ngIf="show; else elseBlock">La condición es verdadera</div>
<ng-template #elseBlock>La condición es falsa</ng-template>
```

Creando una directiva estructural

Vamos a crear una directiva estructural que repita un elemento n veces. La directiva se llamará repeatTimes:

```
ng g directive repeat-times
```

Así es como la utilizaremos una vez implementada:

```
<p *apRepeatTimes="3; let n = current">
  {{n}} welcome works!
</p>
```

Para obtener el valor recibido (3) dentro de la directiva, tenemos que crear un *setter* que tenga el mismo nombre que la misma. Dentro de ese método debemos llamar a createEmbeddedView para dibujar el elemento que contiene la directiva (en este caso con un bucle lo dibujamos *num* veces). También podemos pasar valores a la plantilla dentro de un objeto, en este caso current. En la plantilla recogemos este valor en una variable llamada n con let n = current.

```
import { Directive, TemplateRef, ViewContainerRef, Input } from '@angular/core';

@Directive({
  selector: '[repeatTimes]'
})
export class RepeatTimesDirective {
  @Input()
  set repeatTimes(num: number) {
    this.viewContainer.clear(); // Si cambian los datos de entrada borramos

    for (let i = 0; i < num; i++) {
```

```
      this.viewContainer.createEmbeddedView(this.templateRef, {
        current: i + 1
      });
    }
  }

  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef
  ) {}
}
```

1 welcome works!

2 welcome works!

3 welcome works!

Vamos a crear otra directiva estructural. Esta vez muy similar a ngFor, pero que también acepta una función para filtrar el array de elementos:

```
<p *apForFilter="let person from persons by filter">
  {{person | json}}
</p>
```

Este es el filtro que aplicamos:

```
filter = (p) => p.age >= 18;
```

{ "name": "Clara", "age": 22 }

{ "name": "John", "age": 36 }

Tenemos 2 diferencias con la anterior directiva. Los datos de entrada (el array persons y filter) vienen precedidos de las palabras from y by. Para obtenerlos debemos crear setters con dichos sufijos (forFilterFrom y forFilterBy). La otra diferencia es como enviamos cada persona a la plantilla cuando recorremos el array. En este caso, con esta sintaxis en la plantilla, se envía en un valor llamado $implicit.

```
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';

@Directive({
  selector: '[forFilter]'
})
export class ForFilterDirective {
  @Input()
  set forFilterFrom(array: any[]) {
    this.items = array;
  }

  @Input()
  set forFilterBy(filter: (item: any) => boolean) {
    this.viewContainer.clear(); // Si cambian los datos de entrada borramos
    this.items.filter(filter).forEach(elem => {
      this.viewContainer.createEmbeddedView(this.templateRef, {
        $implicit: elem
      });
    });
  }

  items: any[] = [];

  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef
  ) {}
}
```

ng-container

El elemento ng-container sirve para usar directivas estructurales. La diferencia

frente a usarlas con un elemento normal, es que este desaparece cuando ha sido procesada. Es útil cuando quieres combinar 2 o más directivas estructurales ya que esto no se puede hacer en un mismo elemento.

```
<ng-container *ngFor="let person of persons"> <!-- Desaparece -->
  <ng-container *ngIf="person.age >= 18"> <!-- Desaparece -->
    <p>{{person | json}}</p> <!-- Sólo quedará este elemento -->
  </ng-container>
</ng-container>
```