**Angular - Controlando componentes hijos con ViewChild y ViewChildren**

Se va a revisar cómo controlar los componentes o directivas que se tiene en el *template* de nuestro componente usando los [decoradores](https://www.typescriptlang.org/docs/handbook/decorators.html) ViewChild y ViewChildren.

Durante el artículo vamos a utilizar como ejemplo un componente al que llamaremos SystemMessage que mostrará un mensaje al usuario cuando la aplicación lo requiera.

|  |
| --- |
| @Component({  selector: 'system-message',  template: `  <div \*ngIf="active" class="system-message">  <p>{{message}}</p>  </div>  `  })  export class SystemMessageComponent {  @Input() message: string;  @Input() active: boolean;  show () {  this.active = true;  }  hide () {  this.active = false;  }  } |

**ViewChild**

Tanto ViewChild como ViewChildren son dos decoradores que, utilizados sobre una propiedad de la clase que representa a un componente, permiten obtener las instancias de elementos nativos, directivas y componentes que estén en el *template* del mismo.

*Configurar una consulta de la vista.*

*Puedes utilizar ViewChild para obtener el primer elemento o la directiva que coincida con el selector del DOM de la vista. Si el DOM de la vista cambia, y un nuevo hijo coincide con el selector, la propiedad se actualizará.*

*Las consultas de la vista se establecen antes de que se llame al callback ngAfterViewInit.*

Es importante remarcar que las propiedades decoradas con ViewChild se resuelven justo antes de ejecutarse el *hook* ngAfterViewInit. Esto quiere decir que si intentamos acceder a la propiedad en un *hook* previo, como por ejemplo en el *hook* ngOnInit, esta tendrá como valor undefined.

Para realizar una *query* utilizando ViewChild podemos utilizar principalmente dos tipos de selectores:

* El tipo del componente. Se obtendrá el primer componente que se encuentre en la vista con el tipo informado
* El nombre de una *template variable*

**Usando un tipo**

Usar ViewChild utilizando como selector el tipo del componente o directiva que queremos obtener nos permite realizar búsquedas en la vista sin necesidad de crear una *template variable*.

Hay que tener en cuenta que en caso de haber más de un elemento con el tipo que se haya usado como selector el framework nos devolverá la primera ocurrencia de la búsqueda.

**app.component.ts**

@Component({

selector: 'my-app',

template: `

<h1>My App</h1>

<system-message></system-message>

`

})

export class AppComponent implements AfterViewInit {

@ViewChild(SystemMessageComponent) message: SystemMessageComponent;

ngAfterViewInit () {

// Ahora puedes utilizar el componente hijo

}

}

**Usando una *template variable***

Si queremos obtener una instancia en concreto entre varias posibilidades podemos usar una *template variable* para utilizar su nombre como selector.

**app.component.ts**

@Component({

selector: 'my-app',

template: `

<h1>My ViewChild Demo</h1>

<system-message #firstMessage></system-message>

<system-message #secondMessage></system-message>

`

})

export class AppComponent implements AfterViewInit {

@ViewChild('firstMessage') message: SystemMessageComponent;

ngAfterViewInit () {

// Ahora puedes utilizar el componente hijo

}

}

La ventaja de este método es que podemos obtener también instancias de elementos nativos:

**app.component.ts**

@Component({

selector: 'my-app',

template: `

<h1 #header>My ViewChild Demo</h1>

<system-message></system-message>

`

})

export class AppComponent implements AfterViewInit {

@ViewChild('header') header: ElementRef;

ngAfterViewInit () {

// Ahora puedes utilizar el componente hijo

}

}

**Utilizando el API con ViewChild**

Hasta ahora hemos aprendido a obtener las instancias de componentes, directivas y elementos nativos que existan en el *template* de nuestro componente pero, ¿y cómo hacemos para comunicarnos con ellas?.

¡Muy sencillo! El objeto recuperado es una instancia de la clase del elemento que hemos obtenido y por lo tanto todos sus métodos y propiedades están accesibles desde el mismo.

Lo único que hay que tener en cuenta es que solo se pueden utilizar después de la ejecución del *hook* ngAfterViewInit.

**app.component.ts**

@Component({

selector: 'my-app',

template: `

<h1>My ViewChild Demo</h1>

<system-message></system-message>

<button (click)="toggleMessage()">Toggle message</button>

`

})

export class AppComponent implements AfterViewInit {

@ViewChild(SystemMessage) systemMessage: SystemMessage;

ngAfterViewInit () {

this.systemMessage.message = 'Hello World!';

}

toggleMessage () {

this.systemMessage.active ? this.systemMessage.hide() : this.systemMessage.show();

}

}

**ViewChildren**

El decorador ViewChildren nos permite obtener todos los elementos que existan en nuestro *template* que sean del **tipo** que usamos como selector. El objeto devuelto es del tipo QueryList, el cual es resuelto antes del *hook* ngAfterViewInit. Podemos subscribirnos al Observable que nos proporciona la propiedad changes para obtener los resultados de la búsqueda. Cada vez que se añada, elimine o mueva un elemento hijo, QueryList se actualizará, y el observable de cambios de la lista de consultas emitirá un nuevo valor.

**app.component.ts**

@Component({

selector: 'my-app',

template: `

<h1 #header>My App</h1>

<system-message message="First message"></system-message>

<system-message message="Second message"></system-message>

<system-message message="Third message"></system-message>

`

})

export class AppComponent implements AfterViewInit {

@ViewChildren(SystemMessageComponent) query: QueryList<SystemMessageComponent>;

ngAfterViewInit () {

this.query.changes.subscribe((items: Array<SystemMessageComponent>) => {

messages.forEach((item: SystemMessageComponent) => console.log(item.message));

});

}

}

Como sucede con las variables locales, los decoradores @ViewChild y @ViewChildren no se limitan a los componentes, sino que también permiten recuperar las referencias del elemento (ElementRef), para acceder a sus propiedades.