
ANGULAR 15

CONTENIDO

- Angular.
- Instalación.
- Commandos.
- Estructura (Scaffolding).
- Components.
- Routing.
- Directives.
- Interfaces.
- Pipes.
- Guards.
- Data Binding.
- Comunicación entre componentes.
- Manipulación del DOM.
- Services.
- Interceptors.
- async-await.
- RxJS.

Data Binding

El Data Binding (enlace de datos) mantiene automáticamente actualizada la página en función del estado de la aplicación. La vinculación de datos se utiliza para especificar elementos como la fuente de una imagen, el estado de un botón o datos para un usuario en particular.

Tipos de data binding

- One-way binding.
  - View to Source .
  - Source to View.
- Two-way binding.

Data Binding

Source to View (One-way binding)

El enlace se realiza desde la fuente (una variable o una función) hacia la vista. Cualquier modificación en la fuente actualizará la vista automagicamente.

Uso

Se realiza mediante **Interpolación** incluyendo el nombre de una propiedad o expresión dentro de llaves dobles '{{ }}'

```
<h1>{{ title }}</h1>
```

Se realiza mediante **Property Binding** incluyendo el nombre de una propiedad o expresión dentro de corchetes '[]'

```
<img [src]="itemImageUrl">

export class AppComponent {
  title = 'Hola Angular';
  itemImageUrl = 'www.imagen.com/123432';
}
```

Data Binding

View to Source (One-way binding)

El enlace se realiza desde la vista (un input, un evento) hacia la fuente. Cualquier entrada en la vista actualizará la fuente automagicamente.

Uso

Se realiza mediante **eventos**

```
<button (click)="myFunction()"> ... </button>


...

export class MiComponenteComponent {

      myFunction() {
            // cualquer acción
      }
}
```

Data Binding

Two-way binding

Se utiliza [()] para enlazar en una secuencia bidireccional de vista a fuente a vista.

Uso

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

//property binding
<mat-select [(value)]="name"> ... </mat-select>


...

export class MiComponenteComponent {

      name = 'Juan'
}
```

Data Binding

Resumen

- Usar **[]** para vincular desde la fuente a la vista.
- Usar **()** para vincular desde la vista a la fuente.
- Usar **[()]** para enlazar en una secuencia bidireccional de vista a fuente a vista.

Comunicación entre Componentes

Refiere al mecanismo a través del cual los diferentes componentes de una aplicación Angular intercambian datos y coordinan acciones. Esta interacción entre componentes es esencial para crear aplicaciones web dinámicas e interconectadas.

Formas de comunicación entre componentes

- Property sharing: @Input.
- Event (EventEmitter): @Output.
- Services.
- Routing.

Comunicación entre Componentes

Property sharing - Parent to Child

Utilizamos el decorador @Input que nos permite recibir un valor desde el componente padre.

Uso

```
...

export class MiComponenteComponent {

      @Input() nombre: string;
      @Input() objeto: Object;
      @Input() lista: Array<any>;

      constructor() {
      }

}
```

Comunicación entre Componentes

Uso

```
...

@Component({
      selector: 'app',
      template:  `<bank-account name="RBC" account-id="4747"></bank-account>`
})

export class MiComponenteComponent {

      @Input('account-id') id: string;

      constructor() {
      }

}
```

Comunicación entre Componentes

Uso

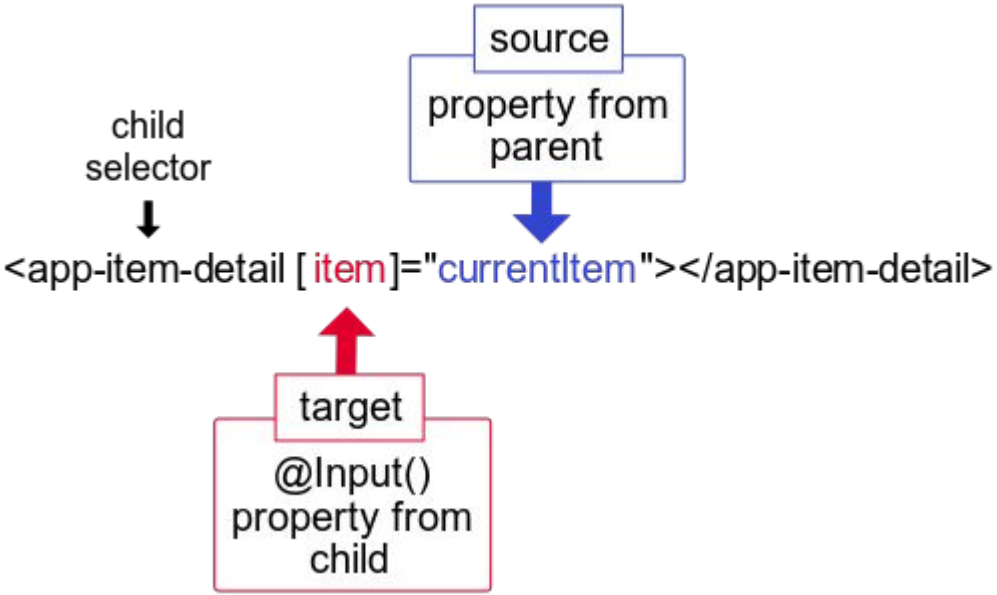
```
<app-padre-component>
      <app-mi-componente [nombre]="Juan" [lista]="[1, 2, 3]"></app-mi-componente>
</app-padre-component>

<app-padre-component>
      <app-mi-componente [lista]="miLista"></app-mi-componente>
</app-padre-component>


...

export class PadreComponent {
      miLista: string[] = ['Juan', 'María', 'Pedro', 'Ana'];
}
```

Comunicación entre Componentes



Comunicación entre Componentes

EventEmitter - Child to Parent

Utilizamos el decorador @Output que nos permite enviar un valor al componente padre como eventos a ser escuchados por este.

Uso

```
...

export class MiComponenteComponent {

      @Output() nuevoElemento = new EventEmitter<string>();

      constructor() {
      }

      addElemento(value: string) {
            this.nuevoElemento.emit(value);
      }
}
```

Comunicación entre Componentes

Uso

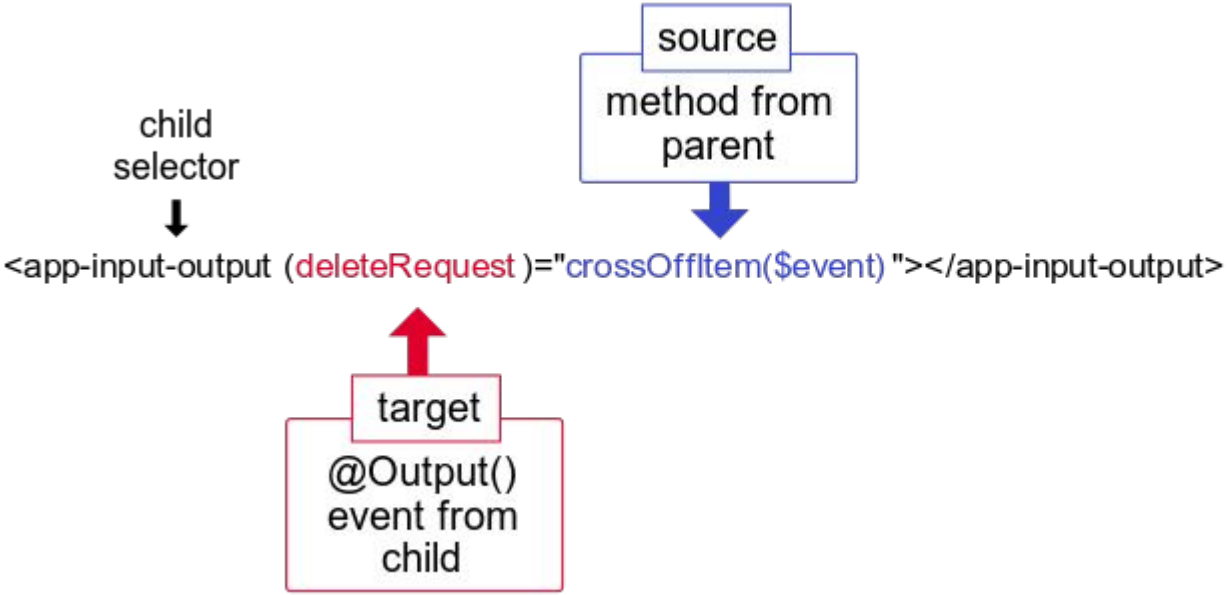
```
<app-padre-component>
      <app-mi-componente (nuevoElemento)="elementoAgregado($event)"></app-mi-componente>
</app-padre-component>
```

```
...

export class PadreComponent {
      elementos: string[] = [];

      elementoAgregado(elemento: string) {
            this.elementos.push(elemento);
      }
}
```

Comunicación entre Componentes



Manipulación del DOM

El DOM es una interfaz de programación que nos permite crear, cambiar, o remover elementos del documento. El DOM visualiza el documento de HTML como un árbol de tres nodos. Un nodo representa un documento de HTML. La manipulación del DOM permite interactuar con la estructura, el estilo y el contenido de las páginas web y modificarlos.

Formas

- Nativa.
- jQuery.
- Decoradores
  - @ViewChild / @ViewChildren
  - @ContentChild / @ContentChildren

Manipulación del DOM

@ViewChild / @ViewChildren

Decorador de propiedades que configura una consulta de vista. El detector de cambios busca el primer elemento o la directiva que coincida con el selector en el DOM de la vista. Si el DOM de la vista cambia y un nuevo elemento secundario coincide con el selector, se actualiza la propiedad.

Sintaxis

```
@ViewChild('myElement') myElement!: ElementRef;
@ViewChild(SomeComponent) someComponent!: SomeComponent;

@ViewChild('myElements') myElements!: QueryList<ElementRef>;
@ViewChild(SomeComponent) someComponents!: QueryList<SomeComponent>;
```

Manipulación del DOM

Uso

```
<h1 #headerElement>Hola, Angular!</h1>

export class AppComponent implements AfterViewInit {
  @ViewChild('headerElement') header!: ElementRef;

  printHeader() {
      console.log(this.header.nativeElement.textContent);
  }

  changeHeader() {
      this.header.nativeElement.textContent = 'Encabezado Cambiado!';
  }
}
```

Manipulación del DOM

Uso

```
<input #inputElement type="text" />

export class AppComponent implements AfterViewInit {
  @ViewChild('inputElement') input!: ElementRef;

      ngAfterViewInit() {
            this.input.nativeElement.value = 'Texto Inicial';
      }
}
```

Manipulación del DOM

Uso

```
<h3 #header>Encabezado 1</h3>
<h3 #header>Encabezado 2</h3>
<h3 #header>Encabezado 3</h3>

export class AppComponent implements AfterViewInit {
  @ViewChildren('header') headers!: QueryList<ElementRef>;

      ngAfterViewInit() {
            this.headers.forEach((header, index) => {
                  console.log(`Encabezado ${index + 1}: ${header.nativeElement.textContent}`);
            });
      }
}
```

Manipulación del DOM

Uso

```
<app-child></app-child>
<app-child></app-child>
<app-child></app-child>

export class AppComponent implements AfterViewInit {
  @ViewChildren(ChildComponent) children!: QueryList<ChildComponent>;

      ngAfterViewInit() {
            this.children.forEach(child => child.greet());
      }
}
```