Angular 2

Ángel Villalba Fdez-Paniagua

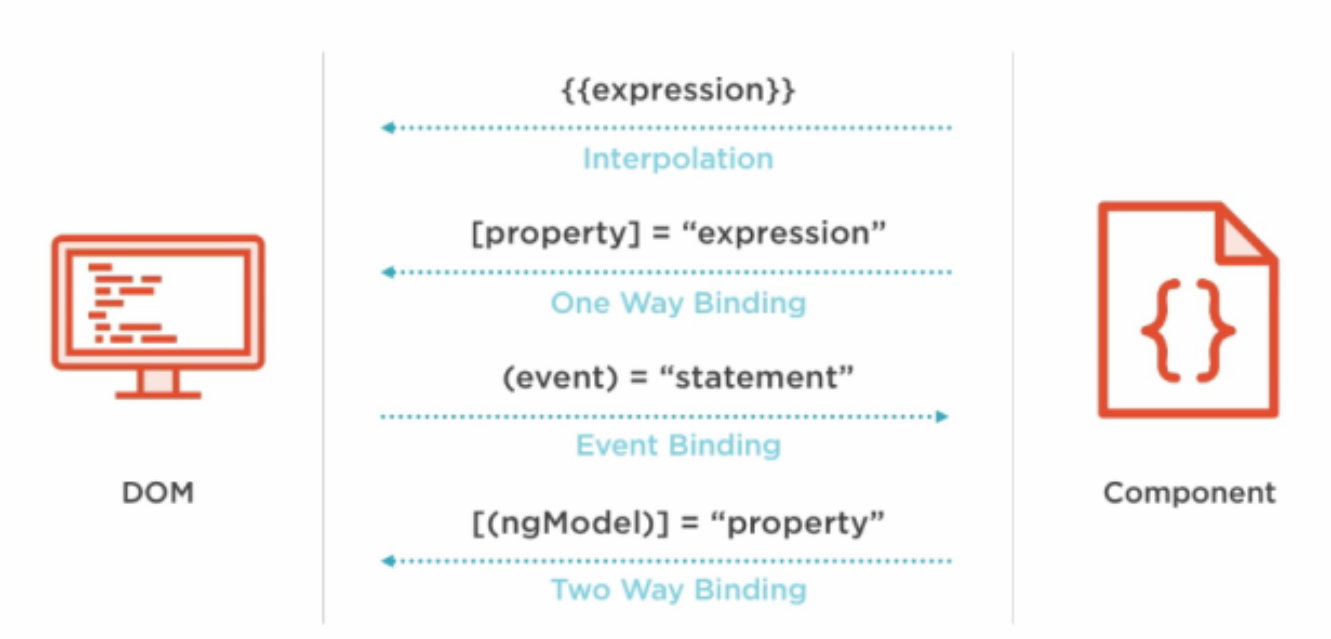
Table of Contents

1. Data Binding 1
   1.1. String Interpolation 1
   1.2. Property Binding 1
   1.3. Event Binding 3
   1.4. Two-Way Data Binding 4
   1.5. Variables de plantilla 5

1. Data Binding

El **data binding** consiste en la sincronización entre los datos del componente y la vista. En Angular 2 nos encontramos cuatro formas distintas de controlar el flujo en el que se mueven los datos:

- **String Interpolation**
- **Property Binding**
- **Event Binding**
- **Two-Way Binding**



1.1. String Interpolation

El `string interpolation` se usa para renderizar el valor de una variable en las plantillas de los componentes, como valor de las etiquetas o incluso de alguno de los atributos de estas etiquetas. Los datos que se usan son solo de lectura. Se usa con `{{ nombreVariable }}`.

```
nombre: string = 'Sara';
```

```
<h1>Mis datos</h1>
<p>Mi nombre es {{ nombre }}.</p> <!-- Mi nombre es Sara. -->
```

1.2. Property Binding

Las `property binding` se usan para enviar valores desde el componente a la plantilla. Se usa con `[propiedad]="expresión"`. La expresión tiene que devolver un valor del tipo que espera obtener la propiedad. También es de lectura. Se suele usar con las propiedades del DOM, de los componentes y de las directivas.

```
bordeRojo {
  border: 1px solid red;
}

letraAzul {
  color: blue;
}
```

```
datosValidos: boolean = false;

setStyles() {
  return {
    'font-size': '15px';
    'text-decoration': 'line-through';
  };
}
```

```
<p [ngClass]="{bordeRojo: true, letraAzul: false}">...</p> <!-- Añade la clase
bordeRojo a la etiqueta p. No añade la clase letraAzul porque tiene el valor false.
-->

<p [ngStyle]="setStyles()">...</p> <!-- Añade los estilos que devuelve la funcion
setStyles. -->

<button type="submit" [disabled]="!datosValidos">Enviar datos</button> <!--
Deshabilita el botón si datosValidos es false. -->
```

A los componentes se les puede pasar desde fuera del propio componente (por ejemplo el componente padre) algún valor para alguna de sus propiedades. Esto es posible por el decorator `@Input`.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'mi-componente',
  template: `
    {{result}}
  `,
  styles: []
})

export class PropertyBindingComponent {
  @Input() result: number = 0;
}
```

```
<mi-componente [result]="10"></mi-componente>
```

En el caso en que no se le pase la propiedad `[result]="10"`, result cogerá el valor con el que se ha inicializado dentro del componente.

1.3. Event Binding

Con el `event binding` le asignamos una función de un componente a un evento nativo como lo es el click, el mouseover, submit... Cuando un elemento detecte un evento sobre el que hay asignada una función, se ejecutará la función. En este caso la comunicación va desde la plantilla al componente. La sintaxis es `(evento)="expresión"` donde la expresíon suele se una función declarada en el componente.

```
saludar() {
  alert('Hola mundo!');
}
```

```
<button (click)="saludar()">Saludame</button>
```

Al igual que se puede crear un property binding, también se puede crear un event binding. Lo primero de todo es importar EventEmitter que permite disparar o emitir eventos con su método `.emit()`. También es necesario usar el decorator `@Output` para permitir que se pueda emitir datos hacia fuera del componente. Se crea una instancia de EventEmitter que lleva el @Output con un alias para poder captar el evento desde el componente que está escuchandolo. Y se crea una función que disparará el evento cuando se realice una acción (por ejemplo un click en un botón). En el siguiente caso, cuando se pincha en el botón, se llama a la función `onClicked()`, la cual emite el evento.

```
import { Component, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'mi-event-binding',
  template: `
    <button (click)="onClicked()">Haz click</button>
  `,
  styles: []
})
export class EventBindingComponent {
  @Output('clickable') clicked = new EventEmitter<string>();

  onClicked() {
    // Emite el evento clickable
    this.clicked.emit('This works!');
  }

}
```

En el componente que está escuchando el evento personalizado se añade un event binding que llamará a una función que se ejecutará cuando ocurra dicho evento.

```
// ...
onClicked(value: string) {
  alert(value);
}
// ...
```

```
<!-- Llamará a la función onClicked cuando se dispare el evento clickable -->
<mi-event-binding (clickable)="onClicked($event)"></mi-event-binding>
```

En el `$event` se envía lo que va como parámetro en el método `emit`.

Ponerle un alias al @Output para poder llamarlo desde fuera no es una buena practica. Se recomienda llamarlo como se llama internamente (en el caso anterior sería *clicked*).

1.4. Two-Way Data Binding

El `Two-Way binding` es la combinación de los dos casos anteriores, el *property binding* y el *event binding* y se refleja en la sintaxis. Recordad que se usaba `[]` para leer datos del componente en la plantilla y `()` para enviar datos desde la plantilla al componente. La sintaxis es `[(ngModel)]="propiedad"`, por lo que lee y envía datos. Este caso es el que se suele usar en los formularios. Comprobar que el `FormsModule` se encuentra importado en el archivo `app.module.ts`.

```
persona = {
  nombre: 'Robb'
  apellidos: 'Stark'
  edad: 28
}
```

```
<!-- Al modificar el nombre en el primer input envía los datos al componente y cambia el valor, y al cambiar el valor en el componente, cambia el valor en el resto de elementos que lo usan. -->
<input type="text" [(ngModel)]="persona.nombre" />
<input type="text" [(ngModel)]="persona.nombre" disabled />
<label>{{persona.nombre}}</label>
```

Este método de binding es costoso y lento.

1.5. Variables de plantilla

Las **variables de plantillas** son referencias a los elementos del DOM a los cuales se les pone una de estas. De esta forma podemos acceder a estos elementos para obtener el valor de algún atributo de los que tiene ese elemento.

Para poner una variable de plantilla solo hay que poner una variable en una etiqueta precedida por `#`.

```
<input type="text" #input>
<button type="button" (click)="mostrarValorInput(input)">Mostrar valor del input</button>
```

```
mostrarValorInput(elementoInput) {
  console.log(elementoInput.value);
  console.log(elementoInput.type);
}
```