Tabla de Contenidos

- @Input
- Comunicación entre componentes
- @Input vs @Output
- @Input vs @Output y su ejecución
- Otros artículos relacionados

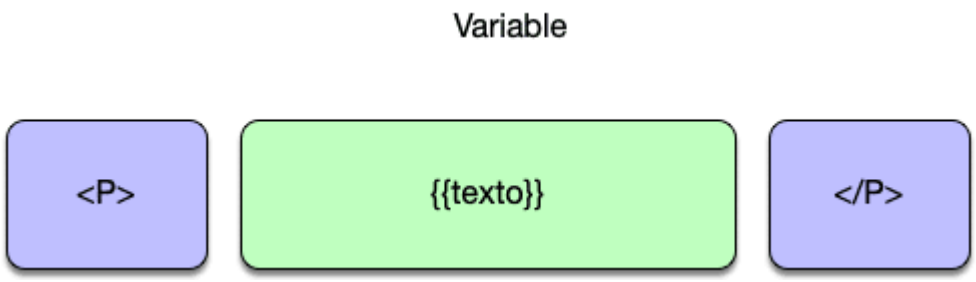
@Input vs @Output son dos de los decoradores más típicos de Angular y que nos permiten Comunicar componentes entre ellos . Habitualmente una página de Angular esta compuesta por varios componentes y en ocasiones estos componentes necesitan comunicarse para pasarse información entre ellos .

@Input

Vamos a ver un ejemplo sencillo de como hacer este tipo de operaciones. Lo primero que vamos a ver es el contenido del componente hijo a nivel de HTML.

```
<p>hijo soy el componente hijo</p>
<p> el mensaje del padre es :{{texto}}</p>

<input type="button" value="enviarPadre" />
```

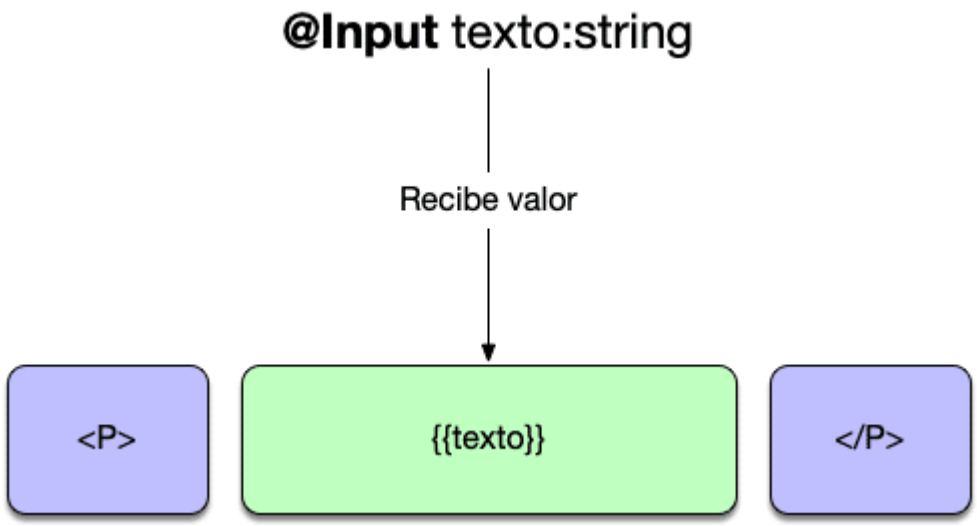
Este componente dispone de su texto y concretamente de una variable {{texto}} que se situa en un parrafo.



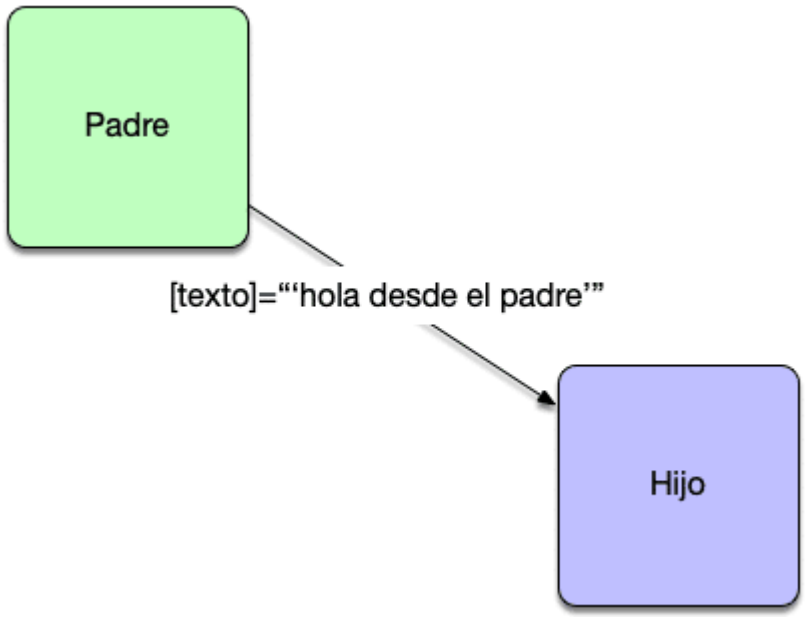
Esta variable va a ser rellenada desde otro componente ya que la vamos a publicar a nivel del código de JavaScript.

```
export class HijoComponent implements OnInit {

  @Input()
  texto: string;

  constructor() { }

  ngOnInit(): void {
  }


}
```

La variable texto dispone de un decorador @Input que le indica que será una variable recibida de otro componente y este componente pasará el valor:



Este valor se asignará a través de un property binding en el conponente padre que es el que contiene al hijo.



Es momento de ver el contenido del componente Padre a nivel de HTML

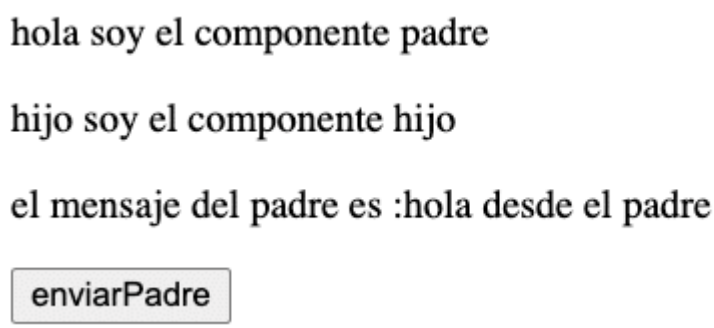
```
<p> hola soy el componente padre {{mensajeHijo}} </p>
<app-hijo [texto]="'hola desde el padre'"  > </app-hijo>
```

Comunicación entre componentes

Como vemos el componente padre pasa un texto con “hola desde el padre hacia el hijo y declara también una variable {{mensajeHijo}} para que el hijo pueda mandar mensajes al padre. Si revisamos el código de TypeScript del Padre nos encontraremos :

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

@Component({
  selector: 'app-padre',
  templateUrl: './padre.component.html',
  styleUrls: ['./padre.component.css']
})
export class PadreComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

El componente esta prácticamente vacío ya que no necesita nada especial para pasar información al Hijo ya que se la pasamos como propiedad. Si ejecutamos la aplicación de Angular podremos ver el resultado .El componente hijo recibe el mensaje del padre.



@Input vs @Output

Ahora nos queda de ver cómo el componente hijo emite un evento y envía información hacia el padre . Esto se realiza a través de la gestión de eventos y de la etiqueta @Output. Lo primero que tenemos que hacer es añadir un evento click al boton enviar al padre con Angular. Esto es sencillo

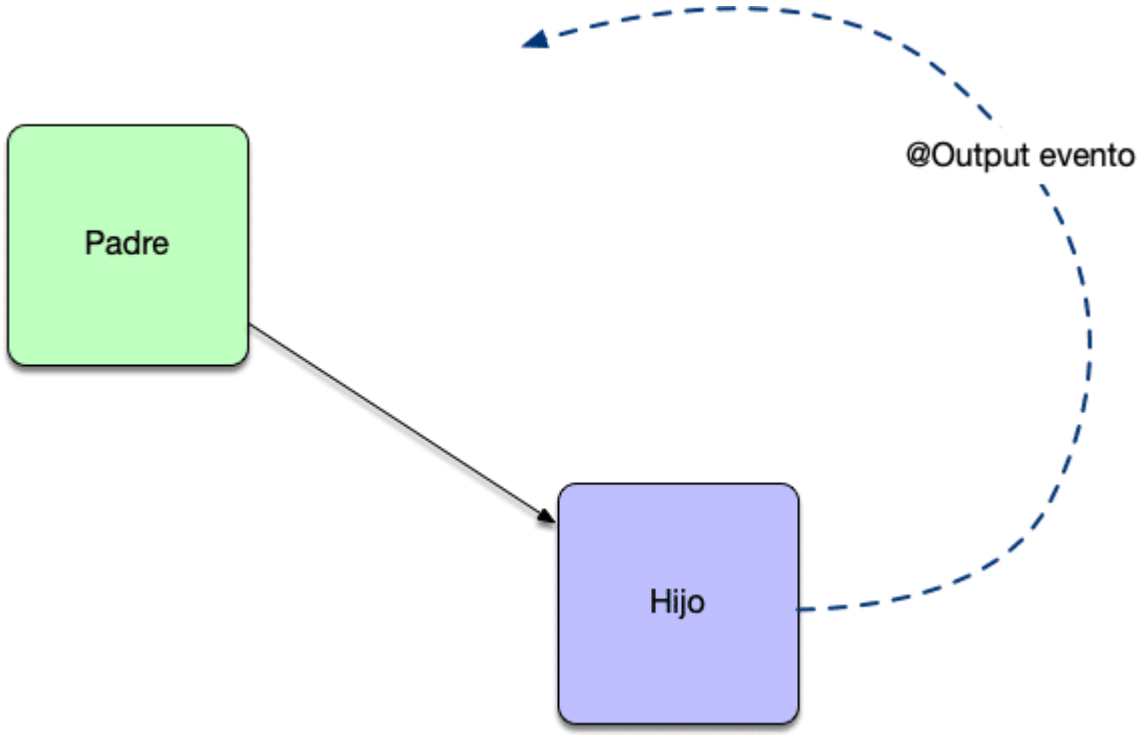
```
<p>hijo soy el componente hijo</p>
<p> el mensaje del padre es :{{texto}}</p>

<input type="button" (click)="enviarPadre()" value="enviarPadre" />
```

Acabamos de crear el evento click y le asociamos a una función concretamente la función enviarPadre .Esta función esta evidentemente declarada en el componente Hijo.

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

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

```
  @Input()
  texto: string;
  @Output()
  eventoHijo = new EventEmitter<string>();

  constructor() { }

  ngOnInit(): void {
  }

  enviarPadre() {
    this.eventoHijo.emit("evento hijo!!!!")
  }
}
```

Como vemos el componente ha quedado modificado y tiene una nueva variable “eventoHijo” esta variable esta anotada con @Output de tal forma que declaramos un evento que el componente Hijo “emite”. Eso si el tipo de declaración es EventEmitter usando String pero se podría haber usado cualquier otro objeto complejo.



Una vez tenemos declarado el evento que se emite nos queda configurar el padre de tal forma que cuando reciba un evento pueda reaccionar a él.

```
<p> hola soy el componente padre {{mensajeHijo}} </p>
<app-hijo [texto]="'hola desde el padre'"
(eventoHijo)="onMensajeHijo($event)"> </app-hijo>
```

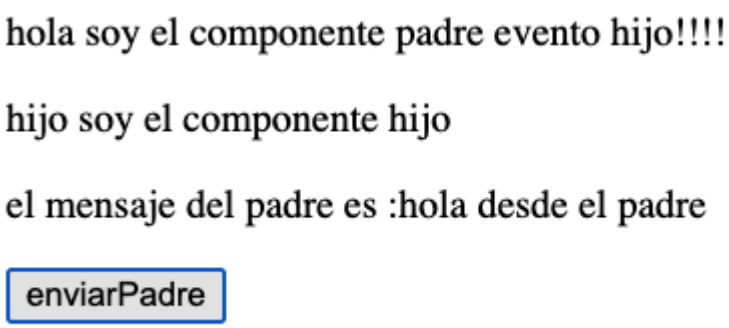
Cómo vemos ahora el componente padre registra el evento que le viene del hijo con (eventoHijo). Es momento de ver el código de TypeScript del Componente.

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

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

```
  mensajeHijo:string
  constructor() { }

  ngOnInit(): void {
  }
  onMensajeHijo(mensaje) {

   this.mensajeHijo=mensaje;
  }

}
```

@Input vs @Output y su ejecución

Como vemos estamos asociando la función a una variable mensajeHijo que se rellena con el "mensaje" que recibimos del evento en nuestro caso "evento hijo!!!!". Es momento de probar la aplicación y ver que la gestión de envío de eventos entre componentes funciona. Nos será suficiente con pulsar el botón enviarPadre y el componente padre cambiará su interface de usuario y añadirá el texto "evento hijo"



Todo funciona correctamente acabamos de ver la diferencia entre @Input y @Output con Angular a la hora de gestionar la comunicación entre componentes.

Otros artículos relacionados

- Angular ngModel y two way databindings
- Angular select y todas sus opciones
- Angular ngFor la directiva y sus opciones
- Curso Angular