Introducción a

Componentes

Fundamentos 4

Angular.io

Un *componente* controla un parche de pantalla llamado *vista* . Por ejemplo, los componentes individuales definen y controlan cada una de las siguientes vistas del Tutorial :

- **La raíz de la aplicación con los enlaces de navegación.**
- **La lista de héroes.**
- **El héroe editor.**

Usted define la lógica de aplicación de un componente, lo hace para admitir la vista, dentro de una clase. La clase interactúa con la vista a través de una API de propiedades y métodos.

Por ejemplo, HeroListComponent tiene una propiedad heroes que contiene una gran variedad de héroes. Su método selectHero() establece una propiedad selectedHero cuando el usuario hace clic para elegir un héroe de esa lista.

El componente adquiere los héroes de un servicio, que es una propiedad de parámetro TypeScript en el constructor. El servicio se proporciona al componente a través del sistema de inyección de dependencia.

src / app / hero-list.component.ts (class)

```
export class HeroListComponent
implements OnInit {

    heroes: Hero[];

    selectedHero: Hero;



    constructor ( private service:
  HeroService) { }

    ngOnInit() {

      this.heroes =
  this.service.getHeroes();

    }

    selectHero(hero: Hero) {
  this.selectedHero = hero; }

  }
```

Angular crea, actualiza y destruye componentes a medida que el usuario se mueve a través de la aplicación.

Su aplicación puede tomar medidas en cada momento de este ciclo de vida a través de ganchos opcionales del ciclo de vida , como ngOnInit().

@Component **metadatos**

de componente

El decorador @Component identifica la clase inmediatamente debajo de ella como una clase de componente y especifica sus metadatos.

En el código de ejemplo a continuación, puede ver que HeroListComponent es solo una clase, sin notación angular especial o sintaxis en absoluto. No es un componente hasta que lo marque como uno con el decorador @Component.

Los metadatos de un componente le indican a Angular dónde obtener los bloques de construcción principales que necesita para crear y presentar el componente y su vista. En particular, asocia una *plantilla* con el componente, ya sea directamente con código en línea o por referencia. Juntos, el componente y su plantilla describen una *vista* .

Además de contener o señalar la plantilla, los metadatos configuran, por ejemplo, cómo se puede hacer referencia al componente en HTML y qué servicios requiere

Aquí hay un ejemplo de metadatos básicos para HeroListComponent.

src / app / hero-list.component.ts (metadata)

```
@Component({
  selector:    'app-hero-list',
  templateUrl: './hero-
list.component.html',
  providers:  [ HeroService ]
})
export class HeroListComponent
implements OnInit {
/* . . . */
}
```

Este ejemplo muestra algunas de las opciones de configuración más útiles de @Component:

selector:

Un selector CSS que le dice a Angular que cree e inserte una instancia de este componente donde encuentre la etiqueta correspondiente en la plantilla HTML. Por ejemplo, si el HTML de una aplicación contiene <app-hero-list></app-hero-list>, Angular inserta una instancia de la vista de HeroListComponent entre esas etiquetas.

templateUrl:

La dirección relativa al módulo de la plantilla HTML de este componente. Alternativamente, puede proporcionar la plantilla HTML en línea, como el valor de la propiedad template. Esta plantilla define la *vista de host* del componente .

providers:

Una array de proveedores de servicios que requiere el componente. En el ejemplo, esto le dice a Angular cómo proporcionar la instancia HeroService que el constructor del componente utiliza para obtener la lista de héroes que se mostrará.

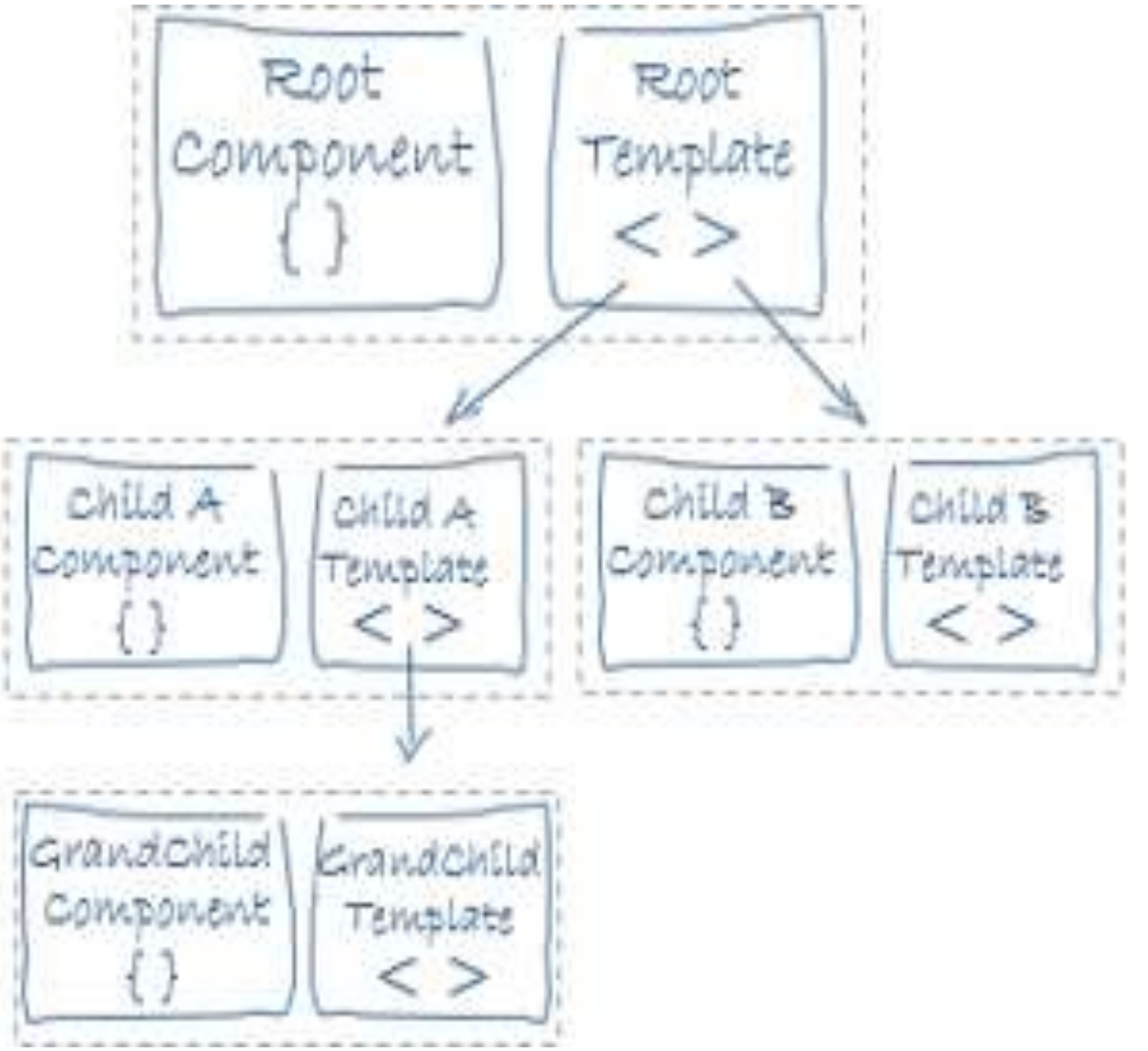
plantillas y vistas

Defina la vista de un componente con su plantilla complementaria. Una plantilla es una forma de HTML que le dice a Angular cómo representar el componente.

Las vistas generalmente se organizan jerárquicamente, lo que le permite modificar o mostrar y ocultar secciones o páginas de IU completas como una unidad.

La plantilla asociada inmediatamente con un componente define la *vista de host de* ese componente .

El componente también puede definir una *jerarquía de vistas* , que contiene *vistas incrustadas* , alojadas por otros componentes.


Root Component { }
Root Template < >
Child A Component { }
Child A Template < >
Child B Component { }
Child B Template < >
GrandChild Component { }
GrandChild Template < >

Una jerarquía de vistas puede incluir vistas de componentes en el mismo NgModule, pero también puede (y a menudo lo hace) incluir vistas de componentes que están definidos en diferentes NgModules.

sintaxis de Template

Una plantilla parece HTML normal, excepto que también contiene una sintaxis de plantilla angular , que altera el HTML en función de la lógica de su aplicación y el estado de la aplicación y los datos DOM.

Su plantilla puede usar *el enlace de datos* para coordinar la aplicación y los datos DOM, las *canalizaciones* para transformar los datos antes de que se muestren y las *directivas* para aplicar la lógica de la aplicación a lo que se muestra.

Por ejemplo, aquí hay una plantilla para el Tutorial HeroListComponent.

src / app / hero-list.component.html

```
<h2> Hero List </h2>

<p><i> Pick a hero from the list
</i></p>

<ul>

  <li *ngFor="let hero of heroes"
(click)="selectHero(hero)">

    {{hero.name}}

  </li>

</ul>

<app-hero-detail ngIf="selectedHero"
[hero]="selectedHero"></app-hero-
detail>
```

Esta plantilla utiliza elementos HTML típicos como <h2>y <p>, y también incluye elementos de template-sintaxis angular *ngFor, {{hero.name}}, (click)[hero], <app-hero-detail> .

Los elementos de sintaxis de plantilla le dicen a Angular cómo representar el HTML en la pantalla, utilizando la lógica y los datos del programa

- **La directiva *ngFor le dice a Angular que repita una lista**
- **{{hero.name}}, (click) y [hero] enlaza datos del programa hacia y desde el DOM, respondiendo a la entrada del usuario.**
- **La etiqueta <app-hero-detail> en el ejemplo es un elemento que representa un nuevo componente, HeroDetailComponent.**

- **HeroDetailComponent (el código no se muestra) define la vista secundaria de héroe-detalle de HeroListComponent. Observe cómo los componentes personalizados como este se mezclan perfectamente con HTML nativo en los mismos diseños.**

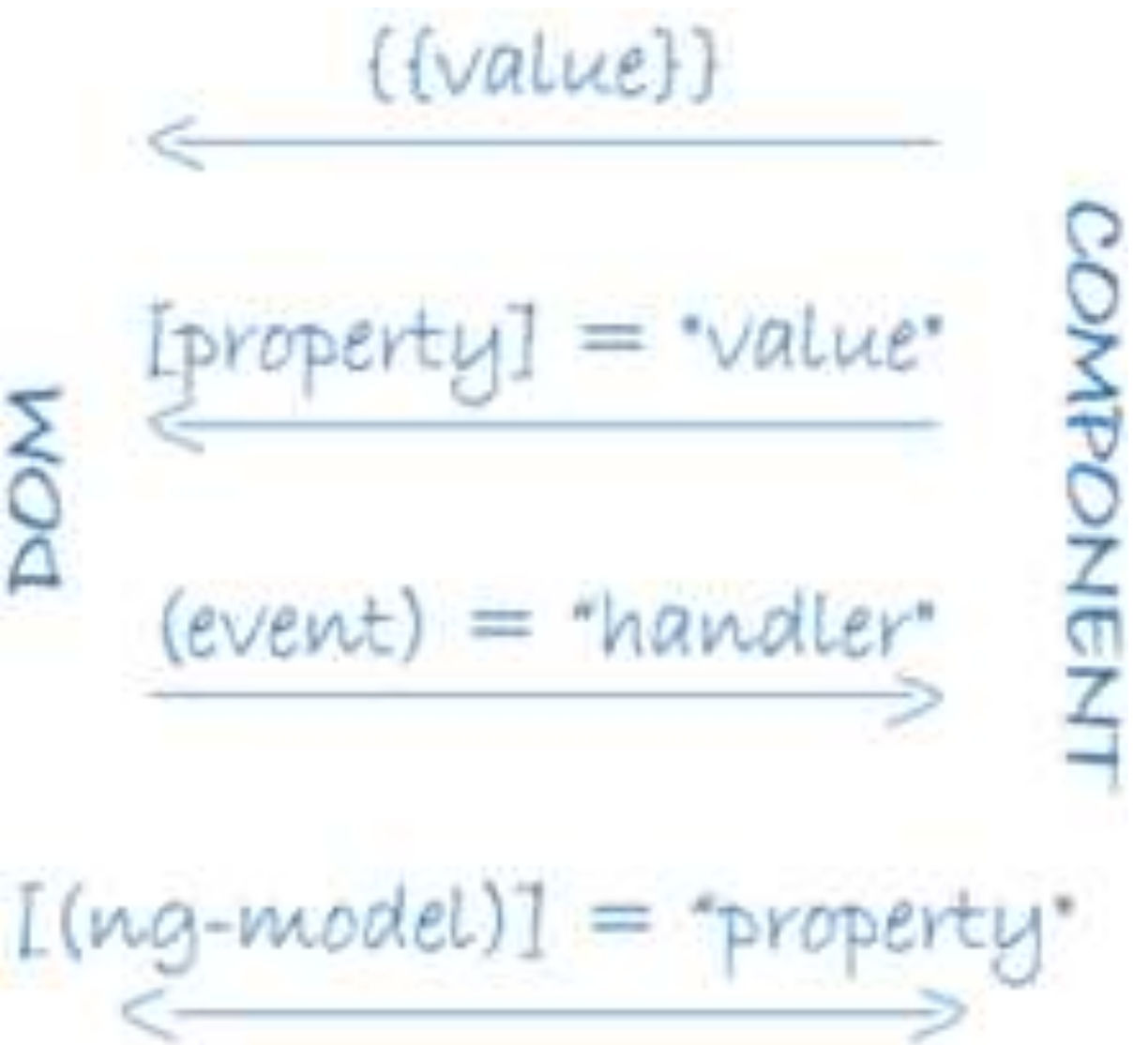
Data Binding

Sin un framework, usted sería responsable de insertar los valores de los datos en los controles HTML y convertir las respuestas de los usuarios en acciones y actualizaciones de valores. Escribir tal lógica es tedioso, propenso a errores y una pesadilla de lectura, como puede atestiguar cualquier programador de JavaScript front-end experimentado.

Angular admite el *enlace de datos bidireccional* , (*two-way data binding*), un mecanismo para coordinar las partes de una plantilla con las partes de un componente.

Agregue marcado de enlace (binding) al HTML de la plantilla para indicarle a Angular cómo conectar ambos lados.

El siguiente diagrama muestra las cuatro formas de marcado de enlace de datos. Cada formulario tiene una dirección: hacia el DOM, desde el DOM, o ambos.



Este ejemplo de

la HeroListComponentplantilla utiliza

tres de estos formularios.

src / app / hero-list.component.html

(binding)

**
{{hero.name}}
**

<app-hero-detail [hero] = "selectedHero">

</app-hero-detail>

**<li (click)="selectHero(hero)">
**

{{hero.name}}

Esta *interpolación* muestra el valor de propiedad del componente hero.name dentro del elemento < li >.

`[hero]` *property binding*

pasa el valor de selectedHero del padre HeroListComponent a la propiedad hero del niño HeroDetailComponent.

El `(click)` *event binding*

llama al selectHero método del componente `selectHero` cuando el usuario hace clic en el nombre de un héroe.

Two-way data binding, (el enlace de datos bidireccional)(utilizado principalmente en formularios basados en plantillas) combina el enlace de propiedad y evento en una sola notación.

Aquí hay un ejemplo del template HeroDetailComponent que usa enlace de datos bidireccional con la directiva ngModel.

src / app / hero-detail.component.html

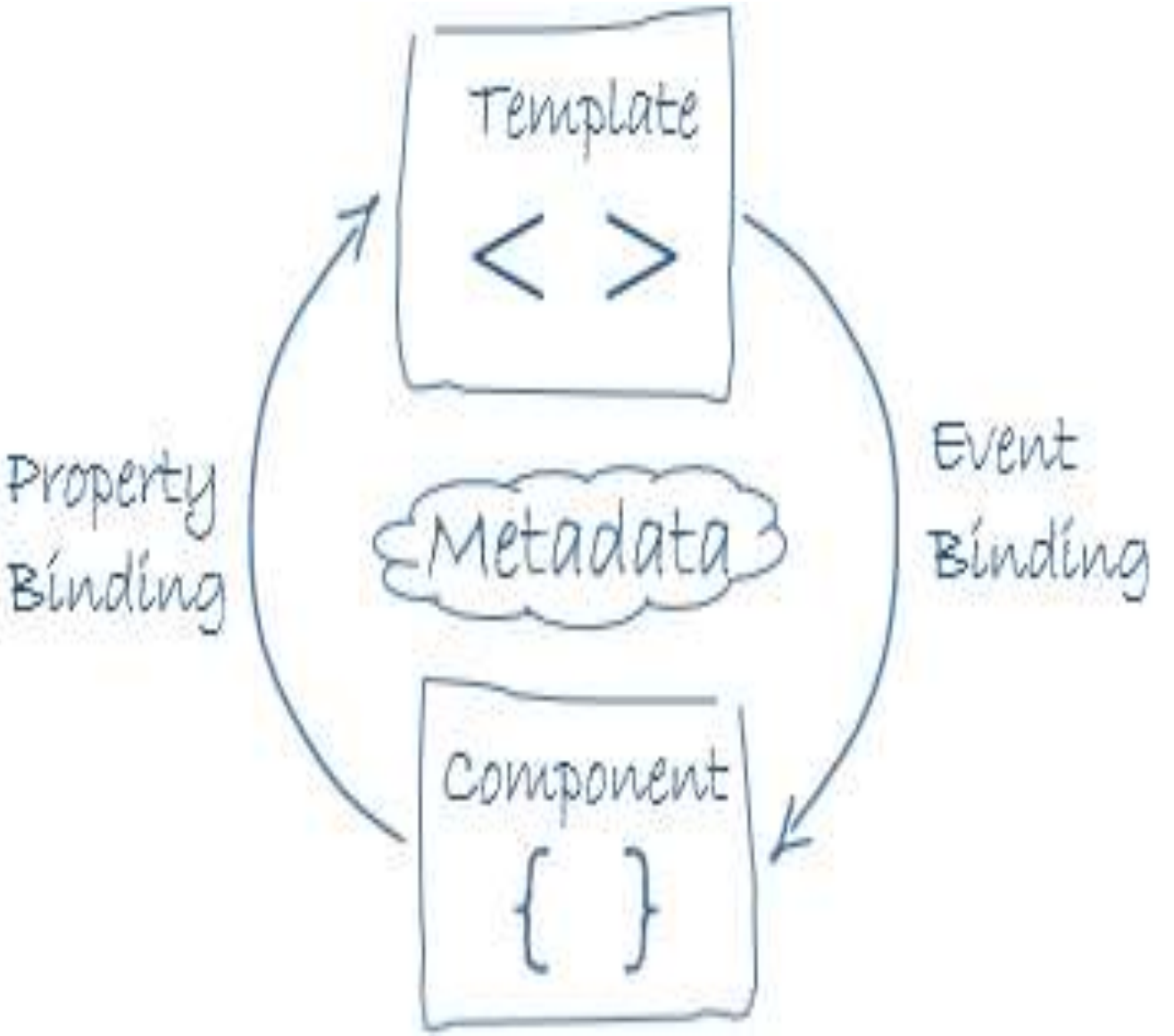
(ngModel)

```
<input [(ngModel)]="hero.name">
```

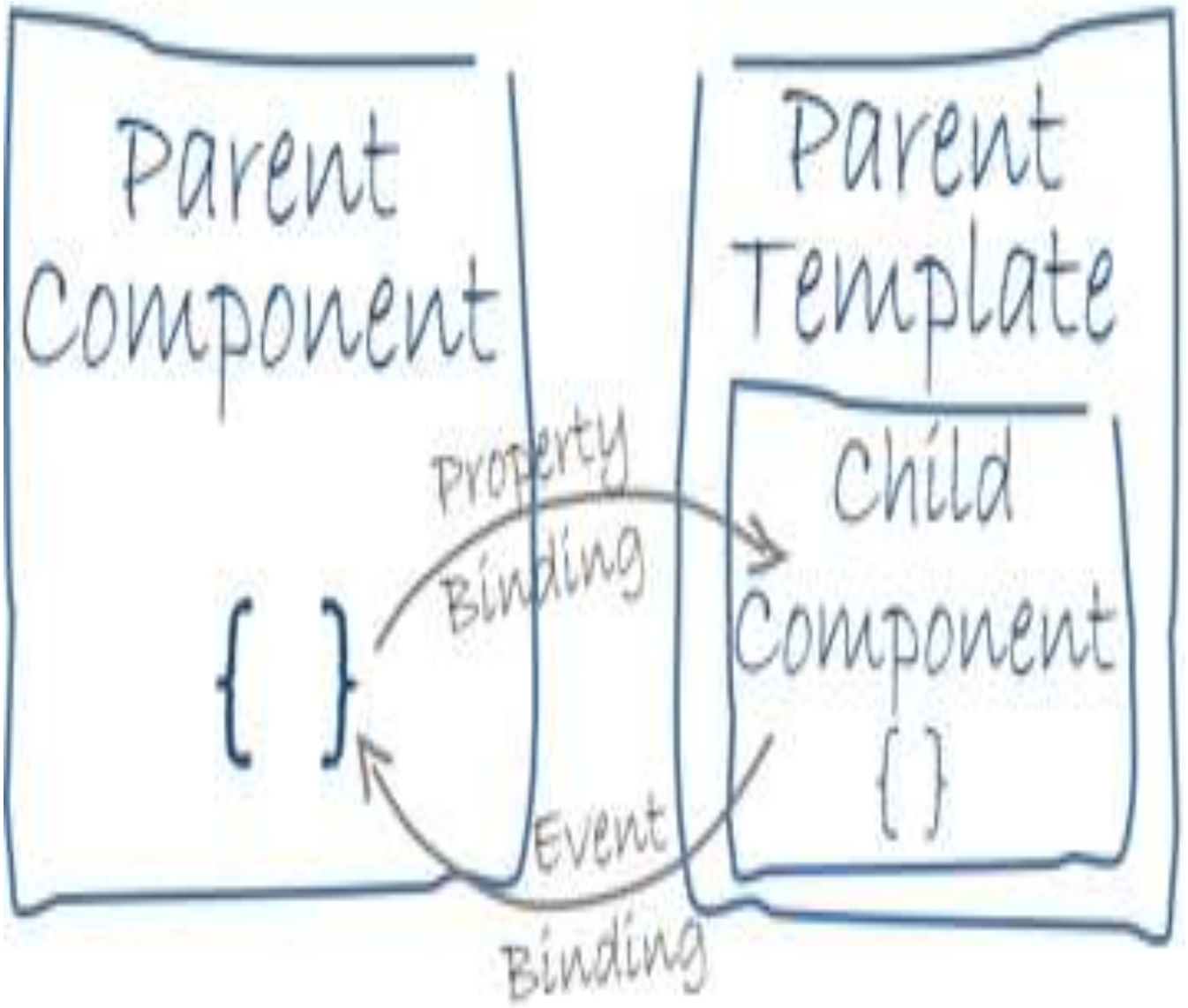
En el enlace bidireccional, (In two-way binding) , un valor de propiedad de datos fluye al cuadro de entrada desde el componente como con el enlace de propiedad.

Los cambios del usuario también vuelven al componente, restableciendo la propiedad al último valor, como con el enlace de eventos.

Angular procesa *todos* los enlaces de datos una vez para cada ciclo de evento de JavaScript, desde la raíz del árbol de componentes de la aplicación hasta todos los componentes secundarios.


Template
< >
Property
Binding
Metadata
Event
Binding
Component
{ }

El enlace de datos juega un papel importante en la comunicación entre una plantilla y su componente, y también es importante para la comunicación entre los componentes principales y secundarios.


Parent Component
{ }
Property Binding
Event Binding
Parent Template
Child Component
{ }

Pipes

Las pipes angular le permiten declarar transformaciones de valor de visualización en su plantilla HTML.

Una clase con el decorador @Pipe define una función que transforma los valores de entrada en valores de salida para mostrar en una vista.

Angular define varias pipes, como la pipe de fecha y la moneda .

Para obtener una lista completa, consulte la lista API de Pipes .

También puede definir nuevas pipes.

Para especificar una transformación de valor en una plantilla HTML, use el operador de pipe (|) .

{{interpolated_value | pipe_name}}

Puede encadenar pipes, enviando la salida de una función pipe para que sea transformada por otra función de pipe. Una pipe también puede tomar argumentos que controlan cómo realiza su transformación. Por ejemplo, puede pasar el formato deseado a la de pipea.

```
<!-- Default format: output 'Jun 15, 2015'-->

    <p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2015'-->

<p>The date is {{today | date:'fullDate'}}</p>

<!-- shortTime format: output '9:43 AM'-->

<p>The time is {{today | date:'shortTime'}}</p>
```

directivas

Las templates angular

son *dinámicas* . Cuando Angular los

procesa, transforma el DOM de acuerdo

con las instrucciones dadas por

las *directivas* . Una directiva es una

clase con un decorador @Directive()

Un componente es técnicamente una directiva. Sin embargo, los componentes son tan distintivos y centrales para las aplicaciones de Angular que Angular define el decorador @Component(), el qual extiende al decorador @Directive() con características orientadas a la template.

Además de los componentes, hay otros dos tipos de directivas:

- ***estructurales***
- **de *atributo* .**

Angular define una serie de directivas de ambos tipos, y puede definir las suyas utilizando el decorador @Directive()

Al igual que para los componentes, los metadatos de una directiva asocian la clase decorada con un selectorelemento que se usa para insertarlo en HTML.

En las plantillas, las directivas generalmente aparecen dentro de una etiqueta de elemento como atributos, ya sea por nombre o como destino de una asignación o un binding.

directivas

estructurales

Las directivas estructurales **alteran el diseño agregando, eliminando y reemplazando elementos en el DOM.**

La plantilla de ejemplo utiliza dos directivas estructurales integradas para agregar lógica de aplicación a cómo se representa la vista.

src / app / hero-list.component.html

(estructural)

```
<li *ngFor="let hero of heroes"></li>

    <app-hero-detail
    *ngIf="selectedHero"></app-hero-
    detail>
```

***ngFor**

Es un iterativo; le dice a Angular que elimine un por héroe en la lista heroes.

***ngIfes**

Es un condicional; incluye el componente HeroDetail solo si existe un héroe seleccionado.

directivas de atributos

Las directivas de atributos **alteran la apariencia o el comportamiento de un elemento existente.**

En las plantillas se ven como atributos HTML normales, de ahí el nombre.

La directiva ngModel, que implementa el enlace de datos bidireccional, es un ejemplo de una directiva de atributo.

ngModel modifica el comportamiento de un elemento existente (típicamente <input >) al establecer su propiedad de valor de visualización y responder a los eventos de cambio.

src / app / hero-detail.component.html (ngModel)

```
<input [(ngModel)]="hero.name">
```

Angular tiene más directivas predefinidas que alteran la estructura de diseño (por ejemplo, ngSwitch) o modifican aspectos de elementos y componentes DOM (por ejemplo, ngStyle y ngClass).

Glosario

directiva

es una clase con un decorador

@Directive()