# **Resultado de imagen de simbolo de angular 8**

# **sintaxis de Template**

**https://angular.io/guide/template-syntax**

**sintaxis de Template**

**La aplicación Angular gestiona lo que el usuario ve y puede hacer, logrando esto mediante la interacción de una instancia de clase de componente (el componente ) y su plantilla orientada al usuario.**

**Puede estar familiarizado con la dualidad componente / plantilla de su experiencia con model-view-controller (MVC) o model-view-viewmodel (MVVM). En Angular, el componente desempeña el papel del controlador / modelo de vista, y la plantilla representa la vista.**

**Esta página es una referencia técnica completa del lenguaje de plantilla angular. Explica los principios básicos del lenguaje de plantilla y describe la mayor parte de la sintaxis que encontrará en otra parte de la documentación.**

**Muchos fragmentos de código ilustran los puntos y conceptos, todos ellos disponibles en el**[**Código de sintaxis de plantilla**](https://angular.io/generated/live-examples/template-syntax/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/template-syntax/template-syntax.zip)**.**

## **HTML en plantillas**

**HTML es el lenguaje de la plantilla angular. Casi toda la sintaxis HTML es una sintaxis de plantilla válida. El <script>elemento es una notable excepción; Está prohibido, eliminando el riesgo de ataques de inyección de script. En la práctica, <script>se ignora y aparece una advertencia en la consola del navegador. Vea la página de**[**Seguridad**](https://angular.io/guide/security)**para más detalles.**

**Algunos HTML legales no tienen mucho sentido en una plantilla. Las <html>, <body>y <base>los elementos no tienen ninguna función útil. Casi todo lo demás es un juego justo.**

**Puede ampliar el vocabulario HTML de sus plantillas con componentes y directivas que aparecen como nuevos elementos y atributos. En las siguientes secciones, aprenderá cómo obtener y establecer valores DOM (Modelo de objetos de documento) dinámicamente a través del enlace de datos.**

**Comience con la primera forma de enlace de datos (interpolación) para ver cuánto más rico puede ser el HTML de la plantilla.**

## **interpolación y expresiones de plantilla**

**La interpolación le permite incorporar cadenas calculadas en el texto entre etiquetas de elementos HTML y dentro de las asignaciones de atributos. Las expresiones de plantilla son lo que usa para calcular esas cadenas.**

**La interpolación**[**ejemplo en vivo**](https://angular.io/generated/live-examples/template-syntax/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/template-syntax/template-syntax.zip)**muestra todos los fragmentos de código y sintaxis descritos en esta sección.**

### **interpolación{{...}}**

**La interpolación se refiere a incrustar expresiones en texto marcado. Por defecto, la interpolación utiliza como delimitador de los tirantes dobles rizadas, {{y }}.**

**En el siguiente fragmento, {{ currentCustomer }}es un ejemplo de interpolación.**

**src / app / app.component.html**

**content\_copy<h3>Current customer: {{ currentCustomer }}</h3>**

**El texto entre llaves suele ser el nombre de una propiedad de componente. Angular reemplaza ese nombre con el valor de cadena de la propiedad del componente correspondiente.**

**src / app / app.component.html**

**content\_copy<p>{{title}}</p>**

**<div><img src="{{itemImageUrl}}"></div>**

**En el ejemplo anterior, Angular evalúa las titley itemImageUrlpropiedades y llena los espacios en blanco, primero se presentan algunos texto del título y luego una imagen.**

**En términos más generales, el texto entre llaves es una expresión de plantilla que Angular primero evalúa y luego convierte en una cadena . La siguiente interpolación ilustra el punto al sumar dos números:**

**src / app / app.component.html**

**content\_copy<!-- "The sum of 1 + 1 is 2" -->**

**<p>The sum of 1 + 1 is {{1 + 1}}.</p>**

**La expresión puede invocar métodos del componente host como getVal()en el siguiente ejemplo:**

**src / app / app.component.html**

**content\_copy<!-- "The sum of 1 + 1 is not 4" -->**

**<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>**

**Angular evalúa todas las expresiones en llaves dobles, convierte los resultados de la expresión en cadenas y las vincula con las cadenas literales vecinas. Finalmente, asigna este resultado interpolado compuesto a un elemento o propiedad directiva .**

**Parece que está insertando el resultado entre etiquetas de elementos y asignándolo a atributos. Sin embargo, la interpolación es una sintaxis especial que Angular convierte en un enlace de propiedad .**

**Si desea utilizar algo distinto de {{y }}, puede configurar el delimitador de interpolación a través de la opción de**[**interpolación**](https://angular.io/api/core/Component#interpolation)**en los**[**Component**](https://angular.io/api/core/Component)**metadatos.**

### **expresiones de plantilla**

**Una plantilla de expresión produce un valor y aparece dentro de las llaves dobles rizadas, {{ }}. Angular ejecuta la expresión y la asigna a una propiedad de un objetivo vinculante; el objetivo podría ser un elemento HTML, un componente o una directiva.**

**Las llaves de interpolación {{1 + 1}}rodean la expresión de la plantilla 1 + 1. En el enlace de propiedad, aparece una expresión de plantilla entre comillas a la derecha del  =símbolo como en [property]="expression".**

**En términos de sintaxis, las expresiones de plantilla son similares a JavaScript. Muchas expresiones de JavaScript son expresiones de plantillas legales, con algunas excepciones.**

**No puede usar expresiones de JavaScript que tengan o promuevan efectos secundarios, que incluyen:**

* **Asignaciones ( =, +=, -=, ...)**
* **Los operadores como new, typeof, instanceof, etc.**
* **Encadenamiento de expresiones con ;o,**
* **Los operadores de incremento y decremento ++y--**
* **Algunos de los operadores ES2015 +**

**Otras diferencias notables de la sintaxis de JavaScript incluyen:**

* **No hay soporte para los operadores bit a bit como |y&**
* **Nuevos**[**operadores de expresiones de plantilla**](https://angular.io/guide/template-syntax#expression-operators)**, como |, ?.y!**

### **contexto de expresión**

**El contexto de expresión es típicamente la instancia del componente . En los siguientes fragmentos, las recommendedllaves dobles dentro y las itemImageUrl2comillas se refieren a las propiedades de AppComponent.**

**src / app / app.component.html**

**content\_copy<h4>{{recommended}}</h4>**

**<img [src]="itemImageUrl2">**

**Una expresión también puede referirse a propiedades del contexto de la plantilla , como una variable de entrada de plantilla,**

**let customer, O una variable de referencia plantilla, #customerInput.**

**src / app / app.component.html (variable de entrada de plantilla)**

**content\_copy<ul>**

**<li \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let customer of customers">{{customer.name}}</li>**

**</ul>**

**src / app / app.component.html (variable de referencia de plantilla)**

**content\_copy<label>**[**Type**](https://angular.io/api/core/Type) **something:**

**<input #customerInput>{{customerInput.value}}**

**</label>**

**El contexto de los términos en una expresión es una combinación de las variables de plantilla , el objeto de contexto de la directiva (si tiene una) y los miembros del componente . Si hace referencia a un nombre que pertenece a más de uno de estos espacios de nombres, el nombre de la variable de plantilla tiene prioridad, seguido de un nombre en el contexto de la directiva y, por último, los nombres de los miembros del componente.**

**El ejemplo anterior presenta tal colisión de nombres. El componente tiene una customer propiedad y define una variable de plantilla.\***[**ngFor**](https://angular.io/api/common/NgForOf)**customer**

**El customeren {{customer.name}} se refiere a la variable de entrada plantilla, no la propiedad del componente.**

**Las expresiones de plantilla no pueden hacer referencia a nada en el espacio de nombres global, excepto undefined. No pueden referirse a windowo**[**document**](https://angular.io/api/platform-server/PlatformConfig#document)**. Además, no pueden llamar console.log()o Math.max()y están restringidos a hacer referencia a miembros del contexto de expresión.**

### **pautas de expresión**

**Cuando use expresiones de plantilla, siga estas pautas:**

* [**Sencillez**](https://angular.io/guide/template-syntax#simplicity)
* [**Ejecución rápida**](https://angular.io/guide/template-syntax#quick-execution)
* [**Sin efectos secundarios visibles.**](https://angular.io/guide/template-syntax#no-visible-side-effects)

#### **simplicidad**

**Aunque es posible escribir expresiones de plantilla complejas, es una mejor práctica evitarlas.**

**Un nombre de propiedad o una llamada al método debería ser la norma, pero una negación booleana ocasional !está bien. De lo contrario, limite la aplicación y la lógica empresarial al componente, donde es más fácil de desarrollar y probar.**

#### **ejecución rápida**

**Angular ejecuta expresiones de plantilla después de cada ciclo de detección de cambios. Los ciclos de detección de cambios se desencadenan por muchas actividades asincrónicas, como resoluciones prometedoras, resultados HTTP, eventos de temporizador, pulsaciones de teclas y movimientos del mouse.**

**Las expresiones deben terminar rápidamente o la experiencia del usuario puede arrastrarse, especialmente en dispositivos más lentos. Considere almacenar en caché los valores cuando su cálculo sea costoso.**

#### **No hay  efectos secundarios visibles**

**Una expresión de plantilla no debe cambiar ningún estado de aplicación que no sea el valor de la propiedad de destino.**

**Esta regla es esencial para la política de "flujo de datos unidireccional" de Angular. Nunca debe preocuparse de que leer el valor de un componente pueda cambiar algún otro valor mostrado. La vista debe ser estable en una sola pasada de representación.**

**Una expresión**[**idempotente**](https://en.wikipedia.org/wiki/Idempotence)**es ideal porque no tiene efectos secundarios y mejora el rendimiento de detección de cambios de Angular. En términos angulares, una expresión idempotente siempre devuelve exactamente lo mismo hasta que uno de sus valores dependientes cambia.**

**Los valores dependientes no deberían cambiar durante un solo giro del bucle de eventos. Si una expresión idempotente devuelve una cadena o un número, devuelve la misma cadena o número cuando se llama dos veces seguidas. Si la expresión devuelve un objeto, incluido un array, devuelve la misma referencia de objeto cuando se llama dos veces seguidas.**

**Hay una excepción a este comportamiento que se aplica a . tiene una funcionalidad que puede lidiar con la desigualdad referencial de los objetos al iterar sobre ellos. Ver**[**\* ngFor with**](https://angular.io/guide/template-syntax#ngfor-with-trackby)**para más detalles.\***[**ngFor**](https://angular.io/api/common/NgForOf)**\***[**ngFor**](https://angular.io/api/common/NgForOf)**trackBy**[**trackBy**](https://angular.io/guide/template-syntax#ngfor-with-trackby)

## **declaraciones de plantilla**

**Una declaración de plantilla responde a un evento generado por un objetivo vinculante, como un elemento, componente o directiva. Verá declaraciones de plantilla en la sección de**[**enlace de eventos**](https://angular.io/guide/template-syntax#event-binding)**, que aparecen entre comillas a la derecha del = símbolo como en (event)="statement".**

**src / app / app.component.html**

**content\_copy<button (click)="deleteHero()">Delete hero</button>**

**Una declaración de plantilla tiene un efecto secundario . Ese es el objetivo de un evento. Es cómo actualiza el estado de la aplicación desde la acción del usuario.**

**Responder a los eventos es el otro lado del "flujo de datos unidireccional" de Angular. Eres libre de cambiar cualquier cosa, en cualquier lugar, durante este giro del ciclo de eventos.**

**Al igual que las expresiones de plantilla, las declaraciones de plantilla usan un lenguaje que se parece a JavaScript. El analizador de sentencias de plantilla difiere del analizador de expresiones de plantilla y admite específicamente tanto la asignación básica ( =) como las expresiones de encadenamiento (con ;o ,).**

**Sin embargo, cierta sintaxis de JavaScript no está permitida:**

* **new**
* **operadores de incremento y decremento, ++y--**
* **asignación de operador, como +=y-=**
* **los operadores bit a bit |y&**
* **los**[**operadores de expresiones de plantilla**](https://angular.io/guide/template-syntax#expression-operators)

### **contexto de declaración**

**Al igual que con las expresiones, las declaraciones pueden referirse solo a lo que está en el contexto de la declaración, como un método de manejo de eventos de la instancia del componente.**

**El contexto de la declaración suele ser la instancia del componente. El deleteHero en (click)="deleteHero()"es un método del componente enlazado a datos.**

**src / app / app.component.html**

**content\_copy<button (click)="deleteHero()">Delete hero</button>**

**El contexto de la declaración también puede referirse a las propiedades del propio contexto de la plantilla. En los siguientes ejemplos, el $eventobjeto de plantilla , una**[**variable de entrada de plantilla**](https://angular.io/guide/template-syntax#template-input-variable)**( let hero) y una**[**variable de referencia de plantilla**](https://angular.io/guide/template-syntax#ref-vars)**( #heroForm) se pasan a un método de manejo de eventos del componente.**

**src / app / app.component.html**

**content\_copy<button (click)="onSave($event)">Save</button>**

**<button \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let hero of heroes" (click)="deleteHero(hero)">{{hero.name}}</button>**

**<form #heroForm (ngSubmit)="onSubmit(heroForm)"> ... </form>**

**Los nombres de contexto de plantilla tienen prioridad sobre los nombres de contexto de componentes. En lo deleteHero(hero)anterior, heroes la variable de entrada de plantilla, no la heropropiedad del componente .**

### **pautas de declaración**

**Las declaraciones de plantilla no pueden hacer referencia a nada en el espacio de nombres global. No pueden referirse a windowo**[**document**](https://angular.io/api/platform-server/PlatformConfig#document)**. No pueden llamar console.logo Math.max.**

**Al igual que con las expresiones, evite escribir declaraciones complejas de plantilla. Una llamada al método o una asignación de propiedad simple debería ser la norma.**

## **Sintaxis de enlace: un  resumen**

**El enlace de datos es un mecanismo para coordinar lo que ven los usuarios, específicamente con los valores de datos de la aplicación. Si bien podría insertar valores y extraer valores de HTML, la aplicación es más fácil de escribir, leer y mantener si convierte estas tareas en un marco vinculante. Simplemente declara enlaces entre fuentes de enlace, elementos HTML de destino y deja que el marco haga el resto.**

**Para ver una demostración de la sintaxis y los fragmentos de código en esta sección, consulte la publicación**[**ejemplo de sintaxis de enlace**](https://angular.io/generated/live-examples/binding-syntax/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/binding-syntax/binding-syntax.zip)**.**

**Angular proporciona muchos tipos de enlace de datos. Los tipos de enlace se pueden agrupar en tres categorías distinguidas por la dirección del flujo de datos:**

* **De la fuente a la vista**
* **De vista a fuente**
* **Secuencia bidireccional: vista a fuente a vista**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Sintaxis** | **Categoría** |
| **Propiedad de interpolación Atributo Clase Estilo** | **content\_copy{{expression}}**  **[**[**target**](https://angular.io/api/router/RouterLinkWithHref#target)**]="expression"**  **bind-target="expression"** | **Unidireccional desde la fuente de datos para ver el objetivo** |
| **Evento** | **content\_copy(**[**target**](https://angular.io/api/router/RouterLinkWithHref#target)**)="statement"**  **on-target="statement"** | **Unidireccional desde el objetivo de vista hasta el origen de datos** |
| **De doble sentido** | **content\_copy[(**[**target**](https://angular.io/api/router/RouterLinkWithHref#target)**)]="expression"**  **bindon-target="expression"** | **De doble sentido** |

**Vinculantes tipos distintos de interpolación tienen un nombre de destino a la izquierda del signo igual, ya sea rodeado de puntuacion, []o (), o precedido por un prefijo: bind-, on-,bindon- .**

**El objetivo de una unión es la propiedad o evento dentro de la puntuacion de unión: [], ()o [()].**

**Cada miembro público de una directiva fuente está automáticamente disponible para vinculación. No tiene que hacer nada especial para acceder a un miembro de la directiva en una expresión o declaración de plantilla.**

### **datos y  HTML**

**En el curso normal del desarrollo de HTML, crea una estructura visual con elementos HTML y modifica esos elementos estableciendo atributos de elementos con constantes de cadena.**

**content\_copy<div class="special">Plain old** [**HTML**](https://angular.io/api/core/SecurityContext#HTML)**</div>**

**<img src="images/item.png">**

**<button disabled>Save</button>**

**Con el enlace de datos, puede controlar cosas como el estado de un botón:**

**src / app / app.component.html**

**content\_copy<!-- Bind button disabled** [**state**](https://angular.io/api/animations/state) **to `isUnchanged` property -->**

**<button [disabled]="isUnchanged">Save</button>**

**Observe que el enlace es a la disabledpropiedad del elemento DOM del botón, no al atributo. Esto se aplica al enlace de datos en general. El enlace de datos funciona con propiedades de elementos DOM, componentes y directivas, no con atributos HTML .**

### **Atributo HTML versus  propiedad DOM**

**La distinción entre un atributo HTML y una propiedad DOM es clave para comprender cómo funciona el enlace angular. Los atributos están definidos por HTML. Se accede a las propiedades desde los nodos DOM (Modelo de objetos de documento).**

* **Algunos atributos HTML tienen una asignación 1: 1 a propiedades; por ejemplo id,.**
* **Algunos atributos HTML no tienen propiedades correspondientes; por ejemplo aria-\*,.**
* **Algunas propiedades DOM no tienen atributos correspondientes; por ejemplo textContent,.**

**Es importante recordar que el atributo HTML y la propiedad DOM son cosas diferentes, incluso cuando tienen el mismo nombre. En Angular, el único rol de los atributos HTML es inicializar el elemento y el estado de la directiva.**

**El enlace de plantilla funciona con propiedades y eventos , no con atributos .**

**Cuando escribe un enlace de datos, se ocupa exclusivamente de las propiedades y eventos DOM del objeto de destino.**

**Esta regla general puede ayudarlo a construir un modelo mental de atributos y propiedades DOM: los atributos inicializan las propiedades DOM y luego se hacen. Los valores de las propiedades pueden cambiar; los valores de los atributos no pueden.**

**Hay una excepción a esta regla. Los atributos se pueden cambiar por**[**setAttribute()**](https://angular.io/)**, lo que reinicia las propiedades DOM correspondientes.**

**Para obtener más información, consulte la**[**documentación de Interfaces MDN**](https://developer.mozilla.org/en-US/docs/Web/API#Interfaces)**que tiene documentos API para todos los elementos DOM estándar y sus propiedades. La comparación de los**[**<td>atributos de**](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/td)**atributos con las**[**<td>propiedades**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableCellElement)**proporciona un ejemplo útil para la diferenciación. En particular, puede navegar desde la página de atributos a las propiedades a través del enlace "Interfaz DOM" y navegar hasta la jerarquía de herencia HTMLTableCellElement.**

#### **Ejemplo 1: un <input>**

**Cuando se presenta el navegador <input type="text" value="Sarah">, crea un nodo DOM correspondiente con una valuepropiedad inicializada en "Sarah".**

**content\_copy<input type="text" value="Sarah">**

**Cuando el usuario ingresa "Sally" en el <input>, la value propiedad del elemento DOM se convierte en "Sally". Sin embargo, si observa el atributo HTML valueutilizando input.getAttribute('value'), puede ver que el atributo permanece sin cambios: devuelve "Sarah".**

**El atributo HTML valueespecifica el valor inicial ; La valuepropiedad DOM es el valor actual .**

**Para ver los atributos frente a las propiedades DOM en una aplicación en funcionamiento, consulte el**[**ejemplo en vivo**](https://angular.io/generated/live-examples/binding-syntax/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/binding-syntax/binding-syntax.zip)**especialmente para la sintaxis de enlace.**

#### **Ejemplo 2: un  botón deshabilitado**

**El disabledatributo es otro ejemplo. La disabled propiedad de un botón es falsepor defecto, por lo que el botón está habilitado.**

**Cuando agrega el disabled atributo , solo su presencia inicializa la disabled propiedad del botón para true que el botón esté deshabilitado.**

**content\_copy<button disabled>Test Button</button>**

**Agregar y eliminar el disabled atributo deshabilita y habilita el botón. Sin embargo, el valor del atributo es irrelevante, por lo que no puede habilitar un botón escribiendo <button disabled="false">Still Disabled</button>.**

**Para controlar el estado del botón, establezca la disabled propiedad ,**

**Aunque técnicamente podría establecer el [attr.disabled]enlace de atributo, los valores son diferentes en que el enlace de propiedad requiere un valor booleano, mientras que el enlace de atributo correspondiente depende de si el valor es nullo no. Considera lo siguiente:**

**content\_copy<input [disabled]="condition ? true : false">**

**<input [attr.disabled]="condition ? 'disabled' : null">**

**En general, use el enlace de propiedad sobre el enlace de atributo, ya que es más intuitivo (es un valor booleano), tiene una sintaxis más corta y tiene un mejor rendimiento.**

**Para ver el disabledejemplo del botón en una aplicación en funcionamiento, consulte el**[**ejemplo en vivo**](https://angular.io/generated/live-examples/binding-syntax/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/binding-syntax/binding-syntax.zip)**especialmente para la sintaxis de enlace. Este ejemplo muestra cómo alternar la propiedad deshabilitada del componente.**

## **tipos y objetivos de**

**El objetivo de un enlace de datos es algo en el DOM. Dependiendo del tipo de enlace, el destino puede ser una propiedad (elemento, componente o directiva), un evento (elemento, componente o directiva) o, a veces, un nombre de atributo. La siguiente tabla resume los objetivos para los diferentes tipos de enlace.**

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Objetivo** | **Ejemplos** |
| **Propiedad** | **Propiedad del elemento Propiedad del componente Propiedad directiva** | **src, heroy**[**ngClass**](https://angular.io/api/common/NgClass)**en lo siguiente:**  **content\_copy<img [src]="heroImageUrl">**  **<app-hero-detail [hero]="currentHero"></app-hero-detail>**  **<div [**[**ngClass**](https://angular.io/api/common/NgClass)**]="{'special': isSpecial}"></div>** |
| **Evento** | **Evento de elemento Evento de componente Evento de directiva** | **click, deleteRequesty myClicken lo siguiente:**  **content\_copy<button (click)="onSave()">Save</button>**  **<app-hero-detail (deleteRequest)="deleteHero()"></app-hero-detail>**  **<div (myClick)="clicked=$event" clickable>click me</div>** |
| **De doble sentido** | **Evento y propiedad** | **content\_copy<input [(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]="name">** |
| **Atributo** | **Atributo (la excepción)** | **content\_copy<button [attr.aria-label]="help">help</button>** |
| **Clase** | **class propiedad** | **content\_copy<div [class.special]="isSpecial">Special</div>** |
| **Estilo** | [**style**](https://angular.io/api/animations/style)**propiedad** | **content\_copy<button [style.color]="isSpecial ? 'red' : 'green'">** |

## **enlace de propiedad[property]**

**Utilice el enlace de propiedades para establecer propiedades de elementos de destino o decoradores de directivas . Para ver un ejemplo que muestra todos los puntos de esta sección, consulte la publicación @**[**Input**](https://angular.io/api/core/Input)**()**[**ejemplo de enlace de propiedad**](https://angular.io/generated/live-examples/property-binding/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/property-binding/property-binding.zip)**.**

### **unidireccional**

**El enlace de propiedad fluye un valor en una dirección, desde la propiedad de un componente a una propiedad de elemento de destino.**

**No puede usar el enlace de propiedad para leer o extraer valores de elementos de destino. Del mismo modo, no puede utilizar el enlace de propiedad para llamar a un método en el elemento de destino. Si el elemento genera eventos, puede escucharlos con un**[**enlace de evento**](https://angular.io/guide/template-syntax#event-binding)**.**

**Si debe leer una propiedad de elemento de destino o llamar a uno de sus métodos, consulte la referencia de API para**[**ViewChild**](https://angular.io/api/core/ViewChild)**y**[**ContentChild**](https://angular.io/api/core/ContentChild)**.**

### **ejemplos**

**El enlace de propiedad más común establece una propiedad de elemento en un valor de propiedad de componente. Un ejemplo es vincular la srcpropiedad de un elemento de imagen a la itemImageUrlpropiedad de un componente :**

**src / app / app.component.html**

**content\_copy<img [src]="itemImageUrl">**

**Aquí hay un ejemplo de enlace a la colSpanpropiedad. Tenga en cuenta que no está colspan, que es el atributo, escrito en minúscula s.**

**src / app / app.component.html**

**content\_copy<!-- Notice the colSpan property is camel case -->**

**<tr><td [colSpan]="2">Span 2 columns</td></tr>**

**Para obtener más detalles, consulte la documentación de**[**MDN HTMLTableCellElement**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableCellElement)**.**

**Otro ejemplo es deshabilitar un botón cuando el componente dice que isUnchanged:**

**src / app / app.component.html**

**content\_copy<!-- Bind button disabled** [**state**](https://angular.io/api/animations/state) **to `isUnchanged` property -->**

**<button [disabled]="isUnchanged">Disabled Button</button>**

**Otro es establecer una propiedad de una directiva:**

**src / app / app.component.html**

**content\_copy<p [**[**ngClass**](https://angular.io/api/common/NgClass)**]="**[**classes**](https://angular.io/api/core/DebugElement#classes)**">[[ngClass](https://angular.io/api/common/NgClass)] binding to the** [**classes**](https://angular.io/api/core/DebugElement#classes) **property making this blue</p>**

**Otro más es establecer la propiedad del modelo de un componente personalizado, una excelente manera para que los componentes primarios y secundarios se comuniquen:**

**src / app / app.component.html**

**content\_copy<app-item-detail [childItem]="parentItem"></app-item-detail>**

### **objetivos**

**Una propiedad de elemento entre corchetes que encierra identifica la propiedad de destino. La propiedad de destino en el siguiente código es la srcpropiedad del elemento de imagen .**

**src / app / app.component.html**

**content\_copy<img [src]="itemImageUrl">**

**También existe la bind-alternativa de prefijo:**

**src / app / app.component.html**

**content\_copy<img bind-src="itemImageUrl">**

**En la mayoría de los casos, el nombre de destino es el nombre de una propiedad, incluso cuando parece ser el nombre de un atributo. Entonces, en este caso, srces el nombre de la <img>propiedad del elemento.**

**Las propiedades de los elementos pueden ser los objetivos más comunes, pero Angular mira primero para ver si el nombre es una propiedad de una directiva conocida, como lo es en el siguiente ejemplo:**

**src / app / app.component.html**

**content\_copy<p [**[**ngClass**](https://angular.io/api/common/NgClass)**]="**[**classes**](https://angular.io/api/core/DebugElement#classes)**">[[ngClass](https://angular.io/api/common/NgClass)] binding to the** [**classes**](https://angular.io/api/core/DebugElement#classes) **property making this blue</p>**

**Técnicamente, Angular hace coincidir el nombre con una directiva , uno de los nombres de propiedad enumerados en la matriz de la directiva o una propiedad decorada con . Dichas entradas se asignan a las propias propiedades de la directiva.@[Input](https://angular.io/api/core/Input)()inputs@**[**Input**](https://angular.io/api/core/Input)**()**

**Si el nombre no coincide con una propiedad de una directiva o elemento conocido, Angular informa un error de "directiva desconocida".**

**Aunque el nombre de destino suele ser el nombre de una propiedad, hay una asignación automática de atributo a propiedad en Angular para varios atributos comunes. Estos incluyen class/ className, innerHtml/ innerHTMLy tabindex/ tabIndex.**

### **Evitar el  efectos secundarios**

**La evaluación de una expresión de plantilla no debería tener efectos secundarios visibles. El lenguaje de expresión en sí, o la forma en que escribe expresiones de plantilla, ayuda en cierta medida; no puede asignar un valor a nada en una expresión de enlace de propiedad ni usar los operadores de incremento y decremento.**

**Por ejemplo, podría tener una expresión que invocara una propiedad o método que tuviera efectos secundarios. La expresión podría llamar algo así como getFoo()solo tú sabes lo que getFoo()hace. Si getFoo()cambia algo y resulta que está vinculado a ese algo, Angular puede mostrar o no el valor cambiado. Angular puede detectar el cambio y lanzar un error de advertencia. Como práctica recomendada, respete las propiedades y los métodos que devuelven valores y evitan los efectos secundarios.**

### **Devuelve el  tipo adecuado**

**La expresión de plantilla debe evaluar el tipo de valor que la propiedad de destino espera. Devuelve una cadena si la propiedad de destino espera una cadena, un número si espera un número, un objeto si espera un objeto, y así sucesivamente.**

**En el siguiente ejemplo, la childItempropiedad del ItemDetailComponentespera una cadena, que es exactamente lo que está enviando en el enlace de propiedad:**

**src / app / app.component.html**

**content\_copy<app-item-detail [childItem]="parentItem"></app-item-detail>**

**Puede confirmar esto buscando en ItemDetailComponentdonde el tipo se establece en una cadena: @**[**Input**](https://angular.io/api/core/Input)**src / app / item-detail / item-detail.component.ts (configurando el tipo @Input ())**

**content\_copy@**[**Input**](https://angular.io/api/core/Input)**() childItem: string;**

**Como puede ver aquí, el parentItemin AppComponentes una cadena, que ItemDetailComponentespera:src / app / app.component.ts**

**content\_copyparentItem = 'lamp';**

#### **Pasando en un  objeto**

**El ejemplo simple anterior mostró pasar una cadena. Para pasar un objeto, la sintaxis y el pensamiento son los mismos.**

**En este escenario, ListItemComponentestá anidado dentro AppComponenty la**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**propiedad espera un objeto.**

**src / app / app.component.html**

**content\_copy<app-list-item [items]="currentItem"></app-list-item>**

**La**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**propiedad está declarada en el ListItemComponentcon un tipo de Itemy decorada con :@[Input](https://angular.io/api/core/Input)()**

**src / app / list-item.component.ts**

**content\_copy@**[**Input**](https://angular.io/api/core/Input)**() items: Item[];**

**En esta aplicación de muestra, un Itemes un objeto que tiene dos propiedades; an idy a name.**

**src / app / item.ts**

**content\_copyexport class Item {**

**id: number;**

**name: string;**

**}**

**Si bien existe una lista de elementos en otro archivo, mock-items.tspuede especificar un elemento diferente app.component.tspara que el nuevo elemento presente:**

**src / app.component.ts**

**content\_copycurrentItem = [{**

**id: 21,**

**name: 'phone'**

**}];**

**Solo tiene que asegurarse, en este caso, de que está suministrando un objeto porque ese es el tipo de**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**componente anidado y es lo que ListItemComponentespera.**

**En este ejemplo, AppComponentespecifica un**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**objeto diferente ( currentItem) y lo pasa al anidado ListItemComponent. ListItemComponentfue capaz de usar currentItemporque coincide con lo que corresponde a un Itemobjeto item.ts. El item.tsarchivo es donde ListItemComponentobtiene su definición de un**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**.**

### **Recuerda el  paréntesis**

**Los corchetes, le []dicen a Angular que evalúe la expresión de la plantilla. Si omite los corchetes, Angular trata la cadena como una constante e inicializa la propiedad de destino con esa cadena:**

**src / app.component.html**

**content\_copy<app-item-detail childItem="parentItem"></app-item-detail>**

**Omitir los corchetes representará la cadena parentItem, no el valor de parentItem.**

### **inicialización de cadena de una sola vez**

**Usted debe omitir los paréntesis cuando todas las condiciones siguientes son verdaderas:**

* **La propiedad de destino acepta un valor de cadena.**
* **La cadena es un valor fijo que puede colocar directamente en la plantilla.**
* **Este valor inicial nunca cambia.**

**Rutinariamente inicializa los atributos de esta manera en HTML estándar, y funciona igual de bien para la inicialización de propiedades de directivas y componentes. El siguiente ejemplo inicializa la prefixpropiedad de StringInitComponentuna cadena fija, no una expresión de plantilla. Angular lo establece y lo olvida.**

**src / app / app.component.html**

**content\_copy<app-string-init prefix="This is** [**a**](https://angular.io/api/router/RouterLinkWithHref) **one-time initialized string."></app-string-init>**

**El enlace, por otro lado, sigue siendo un enlace vivo a la propiedad del componente .[[item](https://angular.io/api/core/IterableChangeRecord" \l "item)]currentItem**

### **propiedad versus  interpolación**

**A menudo puede elegir entre interpolación y enlace de propiedad. Los siguientes pares de enlace hacen lo mismo:**

**src / app / app.component.html**

**content\_copy<p><img src="{{itemImageUrl}}"> is the <i>interpolated</i> image.</p>**

**<p><img [src]="itemImageUrl"> is the <i>property bound</i> image.</p>**

**<p><span>"{{interpolationTitle}}" is the <i>interpolated</i> title.</span></p>**

**<p>"<span [innerHTML]="propertyTitle"></span>" is the <i>property bound</i> title.</p>**

**La interpolación es una alternativa conveniente al enlace de propiedad en muchos casos. Al representar los valores de datos como cadenas, no hay ninguna razón técnica para preferir una forma a la otra, aunque la legibilidad tiende a favorecer la interpolación. Sin embargo, al establecer una propiedad de elemento en un valor de datos que no sea de cadena, debe usar el enlace de propiedad .**

### **seguridad de contenido**

**Imagine el siguiente contenido malicioso.**

**src / app / app.component.ts**

**content\_copyevilTitle = '**[**Template**](https://angular.io/) **<script>alert("evil never sleeps")</script> Syntax';**

**En la plantilla del componente, el contenido puede usarse con interpolación:**

**src / app / app.component.ts**

**content\_copy<p><span>"{{evilTitle}}" is the <i>interpolated</i> evil title.</span></p>**

**Afortunadamente, el enlace de datos angular está en alerta por HTML peligroso. En el caso anterior, el HTML se muestra como está y el Javascript no se ejecuta. Angular no permite que el HTML con etiquetas de script se filtre en el navegador, ni con interpolación ni enlace de propiedades.**

**Sin embargo, en el siguiente ejemplo, Angular**[**desinfecta**](https://angular.io/guide/security#sanitization-and-security-contexts)**los valores antes de mostrarlos.**

**src / app / app.component.html**

**content\_copy<!--**

**Angular generates** [**a**](https://angular.io/api/router/RouterLinkWithHref) **warning for the following line as it sanitizes them**

**WARNING: sanitizing** [**HTML**](https://angular.io/api/core/SecurityContext#HTML) **stripped some content (see http://g.co/ng/security#xss).**

**-->**

**<p>"<span [innerHTML]="evilTitle"></span>" is the <i>property bound</i> evil title.</p>**

**La interpolación maneja las <script>etiquetas de manera diferente al enlace de propiedad, pero ambos enfoques hacen que el contenido sea inofensivo. La siguiente es la salida del navegador de los evilTitleejemplos.**

**content\_copy"**[**Template**](https://angular.io/) **Syntax" is the interpolated evil title.**

**"Template alert("evil never sleeps")Syntax" is the property bound evil title.**

## **enlaces de atributos, clases y estilos**

**La sintaxis de la plantilla proporciona enlaces unidireccionales especializados para escenarios menos adecuados para el enlace de propiedades.**

**Para ver los enlaces de atributo, clase y estilo en una aplicación en funcionamiento, consulte la publicación**[**ejemplo en vivo**](https://angular.io/generated/live-examples/attribute-binding/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/attribute-binding/attribute-binding.zip)**Especialmente para esta sección.**

### **enlace de atributo**

**Establezca el valor de un atributo directamente con un enlace de atributo . Esta es la única excepción a la regla de que un enlace establece una propiedad de destino y el único enlace que crea y establece un atributo.**

**Por lo general, es preferible establecer una propiedad de elemento con un**[**enlace de propiedad**](https://angular.io/guide/template-syntax#property-binding)**que establecer el atributo con una cadena. Sin embargo, a veces no hay ninguna propiedad de elemento para vincular, por lo que la vinculación de atributos es la solución.**

**Considere el**[**ARIA**](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA)**y**[**SVG**](https://developer.mozilla.org/en-US/docs/Web/SVG)**. Son puramente atributos, no corresponden a las propiedades del elemento y no establecen las propiedades del elemento. En estos casos, no hay objetivos de propiedad a los que vincularse.**

**La sintaxis de enlace de atributo se asemeja al enlace de propiedad, pero en lugar de una propiedad de elemento entre paréntesis, comience con el prefijo attr, seguido de un punto ( .) y el nombre del atributo. Luego establece el valor del atributo, utilizando una expresión que se resuelve en una cadena, o elimina el atributo cuando la expresión se resuelve en null.**

**Uno de los casos de uso principales para el enlace de atributos es establecer atributos ARIA, como en este ejemplo:**

**src / app / app.component.html**

**content\_copy<!-- create and set an aria attribute for assistive technology -->**

**<button [attr.aria-label]="actionName">{{actionName}} with Aria</button>**

#### **colspany colSpan**

**Observe la diferencia entre el colspanatributo y la colSpanpropiedad.**

**Si escribiste algo como esto:**

**content\_copy<tr><td colspan="{{1 + 1}}">Three-Four</td></tr>**

**Obtendría este error:**

**content\_copy**[**Template**](https://angular.io/) **parse errors:**

**Can't bind to 'colspan' since it isn't** [**a**](https://angular.io/api/router/RouterLinkWithHref) **known** [**native**](https://angular.io/) **property**

**Como dice el mensaje, el <td>elemento no tiene una colspanpropiedad. Esto es cierto porque colspanes un atributo colSpan, con mayúscula S, es la propiedad correspondiente. La interpolación y el enlace de propiedades solo pueden establecer propiedades , no atributos.**

**En su lugar, usaría el enlace de propiedad y lo escribiría así:**

**src / app / app.component.html**

**content\_copy<!-- Notice the colSpan property is camel case -->**

**<tr><td [colSpan]="1 + 1">Three-Four</td></tr>**

### **enlace de clase**

**Agregue y elimine nombres de clase CSS del classatributo de un elemento con un enlace de clase .**

**Aquí se explica cómo configurar el atributo sin enlace en HTML simple:**

**content\_copy<!-- standard class attribute setting -->**

**<div class="**[**item**](https://angular.io/api/core/IterableChangeRecord#item) **clearance special">Item clearance special</div>**

**La sintaxis de unión de clase se asemeja propiedad de unión, pero en lugar de una propiedad de elemento entre paréntesis, se inicia con el prefijo class, opcionalmente seguido de un punto ( .) y el nombre de una clase CSS: [class.class-name].**

**Puede reemplazar eso con un enlace a una cadena de los nombres de clase deseados; Este es un enlace de reemplazo de todo o nada.**

**src / app / app.component.html**

**content\_copy<h3>Overwrite all existing** [**classes**](https://angular.io/api/core/DebugElement#classes) **with** [**a**](https://angular.io/api/router/RouterLinkWithHref) **new class:</h3>**

**<div class="**[**item**](https://angular.io/api/core/IterableChangeRecord#item) **clearance special" [attr.class]="resetClasses">Reset all** [**classes**](https://angular.io/api/core/DebugElement#classes) **at once</div>**

**También puede agregar agregar una clase a un elemento sin sobrescribir las clases que ya están en el elemento:**

**src / app / app.component.html**

**content\_copy<h3>Add** [**a**](https://angular.io/api/router/RouterLinkWithHref) **class:</h3>**

**<div class="**[**item**](https://angular.io/api/core/IterableChangeRecord#item) **clearance special" [class.item-clearance]="itemClearance">Add another class</div>**

**Finalmente, puede enlazar a un nombre de clase específico. Angular agrega la clase cuando la expresión de la plantilla se evalúa como verdadera. Elimina la clase cuando la expresión es falsa.**

**src / app / app.component.html**

**content\_copy<h3>toggle the "special" class on/off with** [**a**](https://angular.io/api/router/RouterLinkWithHref) **property:</h3>**

**<div [class.special]="isSpecial">The class binding is special.</div>**

**<h3>binding to class.special** [**overrides**](https://angular.io/api/platform-browser/HammerGestureConfig#overrides) **the class attribute:</h3>**

**<div class="special" [class.special]="!isSpecial">This one is not so special.</div>**

**<h3>Using the bind- syntax:</h3>**

**<div bind-class.special="isSpecial">This class binding is special too.</div>**

**Si bien esta técnica es adecuada para alternar un solo nombre de clase, tenga en cuenta la**[**NgClass**](https://angular.io/guide/template-syntax#ngClass)**directiva al administrar varios nombres de clase al mismo tiempo.**

### **estilo**

**Puede establecer estilos en línea con un enlace de estilo .**

**La sintaxis de enlace de estilo se asemeja al enlace de propiedad. En lugar de una propiedad de elemento entre paréntesis, comenzar con el prefijo**[**style**](https://angular.io/api/animations/style)**, seguido de un punto ( .) y el nombre de una propiedad de estilo CSS: [style.style-property].**

**src / app / app.component.html**

**content\_copy<button [style.color]="isSpecial ? 'red': 'green'">Red</button>**

**<button [style.background-color]="canSave ? 'cyan': 'grey'" >Save</button>**

**Algunos estilos de encuadernación de estilo tienen una extensión de unidad. El siguiente ejemplo establece condicionalmente el tamaño de fuente en unidades "em" y "%".**

**src / app / app.component.html**

**content\_copy<button [style.font-size.em]="isSpecial ? 3 : 1" >Big</button>**

**<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button>**

**Esta técnica es adecuada para establecer un estilo único, pero tenga en cuenta la**[**NgStyle**](https://angular.io/guide/template-syntax#ngStyle)**directiva cuando configure varios estilos en línea al mismo tiempo.**

**Tenga en cuenta que un nombre de propiedad de estilo se puede escribir en**[**guión**](https://angular.io/guide/glossary#dash-case)**, como se muestra arriba, o en**[**camelCase**](https://angular.io/guide/glossary#camelcase)**, como fontSize.**

## **enlace de evento(event)**

**El enlace de eventos le permite escuchar ciertos eventos, como pulsaciones de teclas, movimientos del mouse, clics y toques. Para ver un ejemplo que muestra todos los puntos de esta sección, consulte la publicación**[**ejemplo de enlace de eventos**](https://angular.io/generated/live-examples/event-binding/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/event-binding/event-binding.zip)**.**

**La sintaxis de enlace de evento angular consiste en un nombre de evento de destino entre paréntesis a la izquierda de un signo igual y una declaración de plantilla citada a la derecha. El siguiente enlace de evento escucha los eventos de clic del botón, llamando al onSave()método del componente cada vez que se produce un clic:**

### **Evento objetivo**

**Como arriba, el objetivo es el evento de clic del botón.**

**src / app / app.component.html**

**content\_copy<button (click)="onSave($event)">Save</button>**

**Alternativamente, use el on-prefijo, conocido como la forma canónica:**

**src / app / app.component.html**

**content\_copy<button on-click="onSave($event)">on-click Save</button>**

**Los eventos de elemento pueden ser los objetivos más comunes, pero Angular busca primero para ver si el nombre coincide con una propiedad de evento de una directiva conocida, como lo hace en el siguiente ejemplo:**

**src / app / app.component.html**

**content\_copy<h4>myClick is an event on the custom ClickDirective:</h4>**

**<button (myClick)="clickMessage=$event" clickable>click with myClick</button>**

**{{clickMessage}}**

**Si el nombre no coincide con un evento de elemento o una propiedad de salida de una directiva conocida, Angular informa un error de "directiva desconocida".**

### **$ evento y declaraciones de manejo de eventos**

**En un enlace de evento, Angular configura un controlador de eventos para el evento de destino.**

**Cuando se genera el evento, el controlador ejecuta la declaración de la plantilla. La declaración de plantilla generalmente involucra un receptor, que realiza una acción en respuesta al evento, como almacenar un valor del control HTML en un modelo.**

**El enlace transmite información sobre el evento. Esta información puede incluir valores de datos como un objeto de evento, cadena o número llamado$event .**

**El evento objetivo determina la forma del $eventobjeto. Si el evento de destino es un evento de elemento DOM nativo, entonces $eventes un**[**objeto de evento DOM**](https://developer.mozilla.org/en-US/docs/Web/Events)**, con propiedades como**[**target**](https://angular.io/api/router/RouterLinkWithHref#target)**y target.value.**

**Considere este ejemplo:**

**src / app / app.component.html**

**content\_copy<input [value]="currentItem.name"**

**(input)="currentItem.name=$event.target.value" >**

**without** [**NgModel**](https://angular.io/api/forms/NgModel)

**Este código establece la <input> valuepropiedad mediante el enlace a la namepropiedad. Para escuchar los cambios en el valor, el código se une al input evento del <input>elemento. Cuando el usuario realiza cambios, el inputevento se produce, y la unión ejecuta la sentencia dentro de un contexto que incluye el objeto de evento DOM, $event.**

**Para actualizar la namepropiedad, el texto modificado se recupera siguiendo la ruta $event.target.value.**

**Si el evento pertenece a una directiva, recuerde que los componentes son directivas.$event tiene cualquier forma que produzca la directiva.**

### **Eventos personalizados con**[**EventEmitter**](https://angular.io/api/core/EventEmitter)

**Las directivas suelen generar eventos personalizados con un Angular**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**. La directiva crea una**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**y la expone como una propiedad. La directiva llama EventEmitter.emit(payload)a disparar un evento, pasando una carga útil de mensaje, que puede ser cualquier cosa. Las directivas principales escuchan el evento vinculando esta propiedad y accediendo a la carga útil a través del $eventobjeto.**

**Considere una ItemDetailComponentque presenta información del elemento y responde a las acciones del usuario. Aunque ItemDetailComponenttiene un botón Eliminar, no sabe cómo eliminar al héroe. Solo puede generar un evento que informe la solicitud de eliminación del usuario.**

**Aquí están los extractos pertinentes de eso ItemDetailComponent:**

**src / app / item-detail / item-detail.component.html (plantilla)**

**content\_copy<img src="{{itemImageUrl}}" [style.display]="displayNone">**

**<span [style.text-decoration]="lineThrough">{{ item.name }}**

**</span>**

**<button (click)="delete()">Delete</button>**

**src / app / item-detail / item-detail.component.ts (deleteRequest)**

**content\_copy// This component makes** [**a**](https://angular.io/api/router/RouterLinkWithHref)[**request**](https://angular.io/api/common/http/testing/TestRequest#request) **but it can't actually delete** [**a**](https://angular.io/api/router/RouterLinkWithHref) **hero.**

**@**[**Output**](https://angular.io/api/core/Output)**() deleteRequest = new** [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<Item>();**

**delete() {**

**this.deleteRequest.emit(this.item);**

**this.displayNone = this.displayNone ? '' : 'none';**

**this.lineThrough = this.lineThrough ? '' : 'line-through';**

**}**

**El componente define una deleteRequestpropiedad que devuelve un**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**. Cuando el usuario hace clic en Eliminar , el componente invoca el delete()método, diciéndole**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**que emita un Itemobjeto.**

**Ahora imagine un componente principal de alojamiento que se une al deleteRequestevento de ItemDetailComponent.**

**src / app / app.component.html (enlace de evento a componente)**

**content\_copy<app-item-detail (deleteRequest)="deleteItem($event)" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-item-detail>**

**Cuando se deleteRequestdesencadena el evento, Angular llama al deleteItem()método del componente principal y pasa el elemento a eliminar (emitido por ItemDetail) en la $eventvariable.**

### **Las declaraciones de plantilla tienen un  efectos secundarios**

**Aunque**[**las expresiones de plantilla**](https://angular.io/guide/template-syntax#template-expressions)**no deberían tener**[**efectos secundarios**](https://angular.io/guide/template-syntax#avoid-side-effects)**, las declaraciones de plantilla generalmente sí. losdeleteItem() método tiene un efecto secundario: elimina un elemento.**

**La eliminación de un elemento actualiza el modelo y, según su código, desencadena otros cambios, incluidas consultas y guardar en un servidor remoto. Estos cambios se propagan a través del sistema y finalmente se muestran en esta y otras vistas.**

## **enlace bidireccional[(...)]**

**El enlace bidireccional le brinda a su aplicación una forma de compartir datos entre una clase de componente y su plantilla.**

**Para ver una demostración de la sintaxis y los fragmentos de código en esta sección, consulte la publicación**[**ejemplo de enlace bidireccional**](https://angular.io/generated/live-examples/two-way-binding/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/two-way-binding/two-way-binding.zip)**.**

### **Conceptos básicos del enlace de  bidireccional**

**La unión bidireccional hace dos cosas:**

1. **Establece una propiedad de elemento específico.**
2. **Escucha un evento de cambio de elemento.**

**Angular ofrece un especial de enlace de datos bidireccionales sintaxis para este propósito, [()]. La [()]sintaxis combina los soportes de propiedad de enlace, []con el paréntesis de la unión caso, ().**

**[()] = BANANA EN UNA CAJA**

**Visualice un plátano en una caja para recordar que los paréntesis van dentro de los corchetes.**

**La [()]sintaxis es fácil de demostrar cuando el elemento tiene una propiedad configurable llamada xy un evento correspondiente llamado xChange. Aquí hay un SizerComponentque se ajusta a este patrón. Tiene una sizepropiedad de valor y un sizeChangeevento complementario :**

**src / app / sizer.component.ts**

**content\_copyimport {** [**Component**](https://angular.io/api/core/Component)**,** [**Input**](https://angular.io/api/core/Input)**,** [**Output**](https://angular.io/api/core/Output)**,** [**EventEmitter**](https://angular.io/api/core/EventEmitter) **} from '@angular/core';**

**@**[**Component**](https://angular.io/api/core/Component)**({**

**selector: 'app-sizer',**

**templateUrl: './sizer.component.html',**

[**styleUrls**](https://angular.io/api/core/Component#styleUrls)**: ['./sizer.component.css']**

**})**

**export class SizerComponent {**

**@**[**Input**](https://angular.io/api/core/Input)**() size: number | string;**

**@**[**Output**](https://angular.io/api/core/Output)**() sizeChange = new** [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<number>();**

**dec() { this.resize(-1); }**

**inc() { this.resize(+1); }**

**resize(delta: number) {**

**this.size = Math.min(40, Math.max(8, +this.size + delta));**

**this.sizeChange.emit(this.size);**

**}**

**}**

**La inicial sizees un valor de entrada de un enlace de propiedad. Al hacer clic en los botones aumenta o disminuye size, dentro de las restricciones de valor mínimo / máximo, y luego aumenta o emite el sizeChangeevento con el tamaño ajustado.**

**Aquí hay un ejemplo en el que el AppComponent.fontSizePxenlace bidireccional es SizerComponent:**

**src / app / app.component.html (bidireccional-1)**

**content\_copy<app-sizer [(size)]="fontSizePx"></app-sizer>**

**<div [style.font-size.px]="fontSizePx">Resizable Text</div>**

**El AppComponent.fontSizePxestablece el SizerComponent.sizevalor inicial .**

**src / app / app.component.ts**

**content\_copyfontSizePx = 16;**

**Al hacer clic en los botones, se actualiza a AppComponent.fontSizePxtravés del enlace bidireccional. El AppComponent.fontSizePxvalor revisado fluye a través del enlace de estilo , haciendo que el texto mostrado sea más grande o más pequeño.**

**La sintaxis de enlace bidireccional es realmente solo azúcar sintáctico para un enlace de propiedad y un enlace de evento . Angular desugar la SizerComponentunión en esto:**

**src / app / app.component.html (bidireccional-2)**

**content\_copy<app-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></app-sizer>**

**La $eventvariable contiene la carga útil del SizerComponent.sizeChangeevento. Angular asigna el $eventvalor a AppComponent.fontSizePxcuando el usuario hace clic en los botones.**

### **enlace bidireccional en formularios**

**La sintaxis de enlace bidireccional es una gran conveniencia en comparación con los enlaces de propiedad y eventos separados. Sería conveniente utilizar el enlace bidireccional con elementos de formulario HTML como <input>y <select>. Sin embargo, ningún elemento HTML nativo sigue el x valor y el xChangepatrón de eventos.**

**Para obtener más información sobre cómo usar el enlace bidireccional en formularios, consulte Angular**[**NgModel**](https://angular.io/guide/template-syntax#ngModel)**.**

## **directivas incorporadas**

**Angular ofrece dos tipos de directivas integradas: directivas de atributos y directivas estructurales. Este segmento revisa algunas de las directivas incorporadas más comunes, clasificadas como**[**directivas de atributo**](https://angular.io/guide/template-syntax#attribute-directives)**o**[**directivas estructurales**](https://angular.io/guide/template-syntax#structural-directives)**y tiene su propia**[**ejemplo de directivas integradas**](https://angular.io/generated/live-examples/built-in-directives/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/built-in-directives/built-in-directives.zip)**.**

**Para obtener más detalles, incluido cómo crear sus propias directivas personalizadas, consulte**[**Directivas de atributos**](https://angular.io/guide/attribute-directives)**y**[**Directivas estructurales**](https://angular.io/guide/structural-directives)**.**

### **directivas de atributo incorporado**

**Las directivas de atributos escuchan y modifican el comportamiento de otros elementos, atributos, propiedades y componentes HTML. Por lo general, los aplica a los elementos como si fueran atributos HTML, de ahí el nombre.**

**Muchos NgModules como the**[**RouterModule**](https://angular.io/guide/router)**y the**[**FormsModule**](https://angular.io/guide/forms)**definen sus propias directivas de atributos. Las directivas de atributos más comunes son las siguientes:**

* [**NgClass**](https://angular.io/guide/template-syntax#ngClass)**: Agrega y elimina un conjunto de clases CSS.**
* [**NgStyle**](https://angular.io/guide/template-syntax#ngStyle)**: Agrega y elimina un conjunto de estilos HTML.**
* [**NgModel**](https://angular.io/guide/template-syntax#ngModel)**: Agrega enlace de datos bidireccional a un elemento de formulario HTML.**

### [**NgClass**](https://angular.io/api/common/NgClass)

**Agregue o elimine varias clases CSS simultáneamente con**[**ngClass**](https://angular.io/api/common/NgClass)**.**

**src / app / app.component.html**

**content\_copy<!-- toggle the "special" class on/off with** [**a**](https://angular.io/api/router/RouterLinkWithHref) **property -->**

**<div [**[**ngClass**](https://angular.io/api/common/NgClass)**]="isSpecial ? 'special' : ''">This div is special</div>**

**Para agregar o eliminar una sola clase, use el**[**enlace de clase en**](https://angular.io/guide/template-syntax#class-binding)**lugar de**[**NgClass**](https://angular.io/api/common/NgClass)**.**

**Considere un setCurrentClasses()método de componente que establece una propiedad de componente currentClasses, con un objeto que agrega o elimina tres clases en función del estado true/ falsede otras tres propiedades de componente. Cada clave del objeto es un nombre de clase CSS; su valor es truesi se debe agregar la clase, falsesi se debe eliminar.**

**src / app / app.component.ts**

**content\_copycurrentClasses: {};**

**setCurrentClasses() {**

**// CSS** [**classes**](https://angular.io/api/core/DebugElement#classes)**: added/removed per current** [**state**](https://angular.io/api/animations/state) **of component** [**properties**](https://angular.io/api/core/DebugElement#properties)

**this.currentClasses = {**

**'saveable': this.canSave,**

**'modified': !this.isUnchanged,**

**'special': this.isSpecial**

**};**

**}**

**Agregar un**[**ngClass**](https://angular.io/api/common/NgClass)**enlace de propiedad para currentClassesestablecer las clases del elemento en consecuencia:**

**src / app / app.component.html**

**content\_copy<div [**[**ngClass**](https://angular.io/api/common/NgClass)**]="currentClasses">This div is initially saveable, unchanged, and special.</div>**

**Recuerde que en esta situación llamaría setCurrentClasses(), tanto inicialmente como cuando cambian las propiedades dependientes.**

### [**NgStyle**](https://angular.io/api/common/NgStyle)

**Utilícelo**[**NgStyle**](https://angular.io/api/common/NgStyle)**para establecer muchos estilos en línea de forma simultánea y dinámica, según el estado del componente.**

#### **Sin**[**NgStyle**](https://angular.io/api/common/NgStyle)

**Para el contexto, considere establecer un valor de estilo único con**[**enlace de estilo**](https://angular.io/guide/template-syntax#style-binding)**, sin**[**NgStyle**](https://angular.io/api/common/NgStyle)**.**

**src / app / app.component.html**

**content\_copy<div [style.font-size]="isSpecial ? 'x-large' : 'smaller'">**

**This div is x-large or smaller.**

**</div>**

**Sin embargo, para establecer muchos estilos en línea al mismo tiempo, use la**[**NgStyle**](https://angular.io/api/common/NgStyle)**directiva.**

**El siguiente es un setCurrentStyles()método que establece una propiedad de componente currentStyles, con un objeto que define tres estilos, en función del estado de otras tres propiedades de componente:**

**src / app / app.component.ts**

**content\_copycurrentStyles: {};**

**setCurrentStyles() {**

**// CSS styles: set per current** [**state**](https://angular.io/api/animations/state) **of component** [**properties**](https://angular.io/api/core/DebugElement#properties)

**this.currentStyles = {**

**'font-style': this.canSave ? 'italic' : 'normal',**

**'font-weight': !this.isUnchanged ? 'bold' : 'normal',**

**'font-size': this.isSpecial ? '24px' : '12px'**

**};**

**}**

**Agregar un**[**ngStyle**](https://angular.io/api/common/NgStyle)**enlace de propiedad para currentStylesestablecer los estilos del elemento en consecuencia:**

**src / app / app.component.html**

**content\_copy<div [**[**ngStyle**](https://angular.io/api/common/NgStyle)**]="currentStyles">**

**This div is initially italic, normal weight, and extra large (24px).**

**</div>**

**Recuerde llamar setCurrentStyles(), tanto inicialmente como cuando cambian las propiedades dependientes.**

### **[(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]:  enlace bidireccional**

**La**[**NgModel**](https://angular.io/api/forms/NgModel)**directiva le permite mostrar una propiedad de datos y actualizar esa propiedad cuando el usuario realiza cambios. Aquí hay un ejemplo:**

**src / app / app.component.html (ejemplo de NgModel)**

**content\_copy<label for="example-ngModel">[(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]:</label>**

**<input [(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]="currentItem.name" id="example-ngModel">**

#### **Importar**[**FormsModule**](https://angular.io/api/forms/FormsModule)**para usar el**[**ngModel**](https://angular.io/api/forms/NgModel)

**Antes de usar la**[**ngModel**](https://angular.io/api/forms/NgModel)**directiva en un enlace de datos bidireccional, debe importarlo**[**FormsModule**](https://angular.io/api/forms/FormsModule)**y agregarlo a la importslista de NgModule . Obtenga más información sobre**[**FormsModule**](https://angular.io/api/forms/FormsModule)**y**[**ngModel**](https://angular.io/api/forms/NgModel)**en**[**formularios**](https://angular.io/guide/forms#ngModel)**.**

**Recuerde importar el**[**FormsModule**](https://angular.io/api/forms/FormsModule)**para que esté disponible de la siguiente manera:[(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]**

**src / app / app.module.ts (importación de FormsModule)**

**content\_copyimport {** [**FormsModule**](https://angular.io/api/forms/FormsModule) **} from '@angular/forms'; // <--- JavaScript import from Angular**

**/\* . . . \*/**

**@**[**NgModule**](https://angular.io/api/core/NgModule)**({**

**/\* . . . \*/**

**imports: [**

[**BrowserModule**](https://angular.io/api/platform-browser/BrowserModule)**,**

[**FormsModule**](https://angular.io/api/forms/FormsModule) **// <--- import into the** [**NgModule**](https://angular.io/api/core/NgModule)

**],**

**/\* . . . \*/**

**})**

**export class AppModule { }**

**Se podría conseguir el mismo resultado con fijaciones separadas a la <input>del elemento de valuepropiedad y inputeventos:**

**src / app / app.component.html**

**content\_copy<label for="without">without** [**NgModel**](https://angular.io/api/forms/NgModel)**:</label>**

**<input [value]="currentItem.name" (input)="currentItem.name=$event.target.value" id="without">**

**Para simplificar la sintaxis, la**[**ngModel**](https://angular.io/api/forms/NgModel)**directiva oculta los detalles detrás de sus propias propiedades de**[**ngModel**](https://angular.io/api/forms/NgModel)**entrada y ngModelChangesalida:**

**src / app / app.component.html**

**content\_copy<label for="example-change">(ngModelChange)="...name=$event":</label>**

**<input [**[**ngModel**](https://angular.io/api/forms/NgModel)**]="currentItem.name" (ngModelChange)="currentItem.name=$event" id="example-change">**

**La**[**ngModel**](https://angular.io/api/forms/NgModel)**propiedad de datos establece la propiedad de valor del elemento y la ngModelChangepropiedad de evento escucha los cambios en el valor del elemento.**

#### [**NgModel**](https://angular.io/api/forms/NgModel)**y  valor de acceso**

**Los detalles son específicos para cada tipo de elemento y, por lo tanto, la**[**NgModel**](https://angular.io/api/forms/NgModel)**directiva solo funciona para un elemento compatible con un**[**ControlValueAccessor**](https://angular.io/api/forms/ControlValueAccessor)**que adapta un elemento a este protocolo. Angular proporciona accesores de valor para todos los elementos básicos de formulario HTML y la guía de**[**formularios**](https://angular.io/guide/forms)**muestra cómo enlazarlos.**

**No puedes aplicar [(**[**ngModel**](https://angular.io/api/forms/NgModel)**)] a un elemento nativo que no sea de forma o a un componente personalizado de terceros hasta que escriba un descriptor de acceso de valor adecuado. Para obtener más información, consulte la documentación de la API en**[**DefaultValueAccessor**](https://angular.io/api/forms/DefaultValueAccessor)**.**

**No necesita un descriptor de acceso de valor para un componente Angular que escriba porque puede nombrar el valor y las propiedades del evento para adaptarse a la**[**sintaxis de enlace bidireccional**](https://angular.io/guide/template-syntax#two-way)**básica de Angular y omitir por**[**NgModel**](https://angular.io/api/forms/NgModel)**completo. El sizeren el**[**Binding de dos vías**](https://angular.io/guide/template-syntax#two-way)**sección es un ejemplo de esta técnica.**

**Los**[**ngModel**](https://angular.io/api/forms/NgModel)**enlaces separados son una mejora sobre el enlace a las propiedades nativas del elemento, pero puede simplificar el enlace con una sola declaración utilizando[(**[**ngModel**](https://angular.io/api/forms/NgModel)**)] sintaxis:**

**src / app / app.component.html**

**content\_copy<label for="example-ngModel">[(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]:</label>**

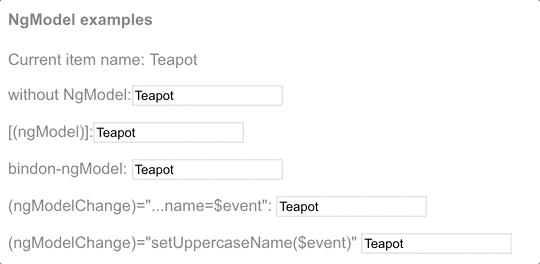
**<input [(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]="currentItem.name" id="example-ngModel">**

**Esta sintaxis solo puede establecer una propiedad vinculada a datos. Si necesita hacer algo más, puede escribir el formulario expandido; por ejemplo, lo siguiente cambia el valor a mayúsculas:[(**[**ngModel**](https://angular.io/api/forms/NgModel)**)]<input>**

**src / app / app.component.html**

**content\_copy<input [**[**ngModel**](https://angular.io/api/forms/NgModel)**]="currentItem.name" (ngModelChange)="setUppercaseName($event)" id="example-uppercase">**

**Aquí están todas las variaciones en acción, incluida la versión en mayúsculas:**

****

## **directivas estructurales incorporado**

**Las directivas estructurales son responsables del diseño HTML. Forman o remodelan la estructura del DOM, típicamente agregando, eliminando y manipulando los elementos host a los que están unidos.**

**Esta sección es una introducción a las directivas estructurales integradas comunes:**

* [**NgIf**](https://angular.io/guide/template-syntax#ngIf)**—Condicionalmente crea o destruye subvistas desde la plantilla.**
* [**NgFor**](https://angular.io/guide/template-syntax#ngFor)**: Repita un nodo para cada elemento de una lista.**
* [**NgSwitch**](https://angular.io/guide/template-syntax#ngSwitch)**—Un conjunto de directivas que cambian entre puntos de vista alternativos.**

**Los detalles profundos de las directivas estructurales se tratan en la guía de**[**Directivas estructurales**](https://angular.io/guide/structural-directives)**, que explica lo siguiente:**

* **Por qué**[**prefijas el nombre de la directiva con un asterisco (\*)**](https://angular.io/guide/structural-directives#the-asterisk--prefix)**.**
* **Se utiliza**[**<ng-container>**](https://angular.io/guide/structural-directives#ngcontainer)**para agrupar elementos cuando no hay un elemento host adecuado para la directiva.**
* **Cómo escribir su propia directiva estructural.**
* **Que solo puede aplicar**[**una directiva estructural**](https://angular.io/guide/structural-directives#one-per-element)**a un elemento.**

### **NgIf**

**Puede agregar o eliminar un elemento del DOM aplicando una**[**NgIf**](https://angular.io/api/common/NgIf)**directiva a un elemento host. Vincula la directiva a una expresión de condición como**[**isActive**](https://angular.io/api/router/RouterLinkActive#isActive)**en este ejemplo.**

**src / app / app.component.html**

**content\_copy<app-item-detail \***[**ngIf**](https://angular.io/api/common/NgIf)**="**[**isActive**](https://angular.io/api/router/RouterLinkActive#isActive)**" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**"></app-item-detail>**

**No olvides el asterisco ( \*) delante de**[**ngIf**](https://angular.io/api/common/NgIf)**. Para obtener más información sobre el asterisco, consulte la sección de**[**prefijo de asterisco (\*)**](https://angular.io/guide/structural-directives#the-asterisk--prefix)**de las**[**Directivas estructurales**](https://angular.io/guide/structural-directives)**.**

**Cuando la**[**isActive**](https://angular.io/api/router/RouterLinkActive#isActive)**expresión devuelve un valor verdadero,**[**NgIf**](https://angular.io/api/common/NgIf)**agrega el ItemDetailComponental DOM. Cuando la expresión es falsa,**[**NgIf**](https://angular.io/api/common/NgIf)**elimina el ItemDetailComponent del DOM, destruyendo ese componente y todos sus subcomponentes.**

#### **Mostrar / ocultar vs**[**NgIf**](https://angular.io/api/common/NgIf)

**Ocultar un elemento es diferente de eliminarlo con**[**NgIf**](https://angular.io/api/common/NgIf)**. A modo de comparación, el siguiente ejemplo muestra cómo controlar la visibilidad de un elemento con un enlace de**[**clase**](https://angular.io/guide/template-syntax#class-binding)**o**[**estilo**](https://angular.io/guide/template-syntax#style-binding)**.**

**src / app / app.component.html**

**content\_copy<!-- isSpecial is true -->**

**<div [class.hidden]="!isSpecial">Show with class</div>**

**<div [class.hidden]="isSpecial">Hide with class</div>**

**<p>ItemDetail is in the DOM but hidden</p>**

**<app-item-detail [class.hidden]="isSpecial"></app-item-detail>**

**<div [style.display]="isSpecial ? 'block' : 'none'">Show with** [**style**](https://angular.io/api/animations/style)**</div>**

**<div [style.display]="isSpecial ? 'none' : 'block'">Hide with** [**style**](https://angular.io/api/animations/style)**</div>**

**Cuando oculta un elemento, ese elemento y todos sus descendientes permanecen en el DOM. Todos los componentes para esos elementos permanecen en la memoria y Angular puede continuar buscando cambios. Podría estar reteniendo recursos informáticos considerables y degradando el rendimiento innecesariamente.**

[**NgIf**](https://angular.io/api/common/NgIf)**Funciona de manera diferente. Cuando**[**NgIf**](https://angular.io/api/common/NgIf)**es false, Angular elimina el elemento y sus descendientes del DOM. Destruye sus componentes, liberando recursos, lo que resulta en una mejor experiencia de usuario.**

**Si está ocultando árboles de componentes grandes, considérelo**[**NgIf**](https://angular.io/api/common/NgIf)**como una alternativa más eficiente que mostrar / ocultar.**

**Para obtener más información sobre**[**NgIf**](https://angular.io/api/common/NgIf)**y**[**ngIfElse**](https://angular.io/api/common/NgIf#ngIfElse)**, consulte la**[**documentación de**](https://angular.io/api/common/NgIf)**la**[**API sobre NgIf**](https://angular.io/api/common/NgIf)**.**

#### **Protéjase contra el  nulo**

**Otra ventaja de esto**[**ngIf**](https://angular.io/api/common/NgIf)**es que puede usarlo para protegerse contra nulos. Show / hide es el más adecuado para casos de uso muy simples, por lo que cuando necesite un protector, opte por él**[**ngIf**](https://angular.io/api/common/NgIf)**. Angular arrojará un error si una expresión anidada intenta acceder a una propiedad denull .**

**Lo siguiente muestra**[**NgIf**](https://angular.io/api/common/NgIf)**custodiando dos <div>s. El currentCustomernombre aparece solo cuando hay un currentCustomer. El nullCustomerno se mostrará el tiempo que es null.**

**src / app / app.component.html**

**content\_copy<div \***[**ngIf**](https://angular.io/api/common/NgIf)**="currentCustomer">Hello, {{currentCustomer.name}}</div>**

**src / app / app.component.html**

**content\_copy<div \***[**ngIf**](https://angular.io/api/common/NgIf)**="nullCustomer">Hello, <span>{{nullCustomer}}</span></div>**

**Consulte también el**[**operador de navegación segura a**](https://angular.io/guide/template-syntax#safe-navigation-operator)**continuación.**

### **NgFor**

**NgFores una directiva de repetidor, una forma de presentar una lista de elementos. Define un bloque de HTML que define cómo se debe mostrar un solo elemento y luego le dice a Angular que use ese bloque como plantilla para representar cada elemento de la lista. El texto asignado a\***[**ngFor**](https://angular.io/api/common/NgForOf)**es la instrucción que guía el proceso del repetidor.**

**El siguiente ejemplo muestra NgForaplicado a un simple <div>. (No olvide el asterisco ( \*) delante de**[**ngFor**](https://angular.io/api/common/NgForOf)**).**

**src / app / app.component.html**

**content\_copy<div \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items">{{item.name}}</div>**

**También puede aplicar una NgFora un elemento componente, como en el siguiente ejemplo.**

**src / app / app.component.html**

**content\_copy<app-item-detail \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**"></app-item-detail>**

**\* NG PARA MICROSINTAXIS**

**La cadena asignada a no es una**[**expresión de plantilla**](https://angular.io/guide/template-syntax#template-expressions)**. Más bien, es una microsintaxis, un pequeño lenguaje propio que Angular interpreta. La cadena significa:\***[**ngFor**](https://angular.io/api/common/NgForOf)**"let**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**of items"**

**Tome cada elemento de la *items*matriz, guárdelo en la**[***item***](https://angular.io/api/core/IterableChangeRecord#item)**variable de bucle local y póngalo a disposición del HTML con plantilla para cada iteración.**

**Angular traduce esta instrucción en un <ng-template>elemento alrededor del host, luego usa esta plantilla repetidamente para crear un nuevo conjunto de elementos y enlaces para cada uno**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**en la lista. Para obtener más información sobre microsintaxis, consulte la guía de**[**Directivas estructurales**](https://angular.io/guide/structural-directives#microsyntax)**.**

#### **variables de entrada de plantilla**

**La letpalabra clave anterior**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**crea una variable de entrada de plantilla llamada**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**. La**[**ngFor**](https://angular.io/api/common/NgForOf)**directiva itera sobre la itemsmatriz devuelta por la itemspropiedad del componente principal y se establece**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**en el elemento actual de la matriz durante cada iteración.**

**Referencia**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**dentro del**[**ngFor**](https://angular.io/api/common/NgForOf)**elemento host así como dentro de sus descendientes para acceder a las propiedades del elemento. El siguiente ejemplo hace referencia**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**primero en una interpolación y luego pasa un enlace a la**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**propiedad del <app-item-detail>componente.**

**src / app / app.component.html**

**content\_copy<div \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items">{{item.name}}</div>**

**<!-- . . . -->**

**<app-item-detail \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**"></app-item-detail>**

**Para obtener más información sobre las variables de entrada de plantilla, consulte**[**Directivas estructurales**](https://angular.io/guide/structural-directives#template-input-variable)**.**

#### **\***[**ngFor**](https://angular.io/api/common/NgForOf)**con index**

**La indexpropiedad del NgForcontexto de directiva devuelve el índice basado en cero del elemento en cada iteración. Puede capturar la indexvariable de entrada en una plantilla y usarla en la plantilla.**

**El siguiente ejemplo captura el indexen una variable llamada iy lo muestra con el nombre del elemento.**

**src / app / app.component.html**

**content\_copy<div \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items; let i=index">{{i + 1}} - {{item.name}}</div>**

**NgFores implementado por la**[**NgForOf**](https://angular.io/api/common/NgForOf)**directiva. Leer más sobre los otros**[**NgForOf**](https://angular.io/api/common/NgForOf)**valores de contexto tales como last,**[**even**](https://angular.io/api/common/NgForOfContext#even)**y**[**odd**](https://angular.io/api/common/NgForOfContext#odd)**en la**[**referencia de la API NgForOf**](https://angular.io/api/common/NgForOf)**.**

#### **\* ngFor con trackBy**

**Si utiliza NgFor con listas grandes, un pequeño cambio en un elemento, como eliminar o agregar un elemento, puede desencadenar una cascada de manipulaciones DOM. Por ejemplo, volver a consultar el servidor podría restablecer una lista con todos los objetos de elementos nuevos, incluso cuando esos elementos se mostraban previamente. En este caso, Angular solo ve una lista nueva de referencias de objetos nuevos y no tiene más remedio que reemplazar los elementos DOM antiguos con todos los elementos DOM nuevos.**

**Puedes hacer esto más eficiente con trackBy. Agregue un método al componente que devuelve el valor que NgFordebe rastrear. En este caso, ese valor es del héroe id. Si idya se ha procesado, Angular lo rastrea y no vuelve a consultarlo en el servidor id.**

**src / app / app.component.ts**

**content\_copytrackByItems(index: number,** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**: Item): number { return item.id; }**

**En la expresión de microsintaxis, establezca trackByel trackByItems()método.**

**src / app / app.component.html**

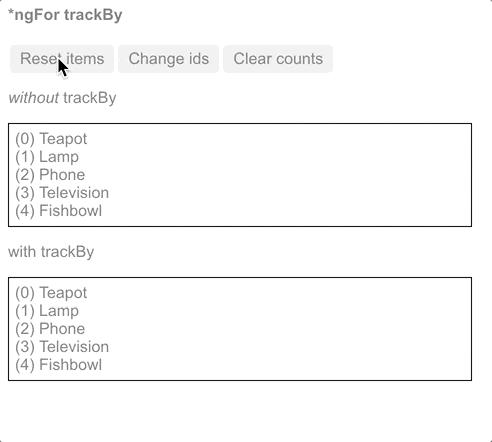
**content\_copy<div \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items; trackBy: trackByItems">**

**({{item.id}}) {{item.name}}**

**</div>**

**Aquí hay una ilustración del trackByefecto. "Restablecer elementos" crea nuevos elementos con el mismo item.ids. "Cambiar identificadores" crea nuevos elementos con nuevos item.ids.**

* **Sin trackBy, ambos botones activan el reemplazo completo del elemento DOM.**
* **Con trackBy, solo se cambia el idreemplazo del elemento desencadenante.**

****

**Las directivas incorporadas usan solo API públicas; es decir, no tienen acceso especial a ninguna API privada a la que otras directivas no puedan acceder.**

## **El**[**NgSwitch**](https://angular.io/api/common/NgSwitch)**directivas**

**NgSwitch es como la switchdeclaración de JavaScript . Muestra un elemento entre varios elementos posibles, en función de una condición de interruptor. Angular coloca solo el elemento seleccionado en el DOM.**

[**NgSwitch**](https://angular.io/api/common/NgSwitch)**En realidad, es un conjunto de tres, directivas cooperante:**[**NgSwitch**](https://angular.io/api/common/NgSwitch)**,**[**NgSwitchCase**](https://angular.io/api/common/NgSwitchCase)**y**[**NgSwitchDefault**](https://angular.io/api/common/NgSwitchDefault)**como en el siguiente ejemplo.**

**src / app / app.component.html**

**content\_copy<div [**[**ngSwitch**](https://angular.io/api/common/NgSwitch)**]="currentItem.feature">**

**<app-stout-item \***[**ngSwitchCase**](https://angular.io/api/common/NgSwitchCase)**="'stout'" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-stout-item>**

**<app-device-item \***[**ngSwitchCase**](https://angular.io/api/common/NgSwitchCase)**="'slim'" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-device-item>**

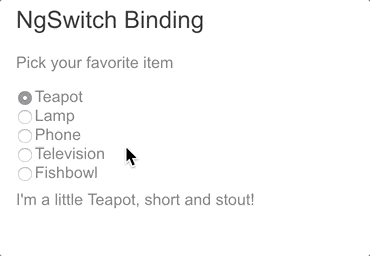
**<app-lost-item \***[**ngSwitchCase**](https://angular.io/api/common/NgSwitchCase)**="'vintage'" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-lost-item>**

**<app-best-item \***[**ngSwitchCase**](https://angular.io/api/common/NgSwitchCase)**="'bright'" [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-best-item>**

**<!-- . . . -->**

**<app-unknown-item \***[**ngSwitchDefault**](https://angular.io/api/common/NgSwitchDefault) **[**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-unknown-item>**

**</div>**

****

[**NgSwitch**](https://angular.io/api/common/NgSwitch)**es la directiva del controlador. Vincúlelo a una expresión que devuelva el valor de cambio , como feature. Aunque el featurevalor en este ejemplo es una cadena, el valor del interruptor puede ser de cualquier tipo.**

**Ate a[[ngSwitch](https://angular.io/api/common/NgSwitch)] . Recibirás un error si intentas configurarlo porque es un\***[**ngSwitchNgSwitch**](https://angular.io/api/common/NgSwitch)**directiva de atributo , no estructural directiva . En lugar de tocar directamente el DOM, cambia el comportamiento de sus directivas complementarias.**

**Enlace a y\***[**ngSwitchCase**](https://angular.io/api/common/NgSwitchCase)**\***[**ngSwitchDefault**](https://angular.io/api/common/NgSwitchDefault)**. Las directivas**[**NgSwitchCase**](https://angular.io/api/common/NgSwitchCase)**y**[**NgSwitchDefault**](https://angular.io/api/common/NgSwitchDefault)**son estructurales porque agregan o eliminan elementos del DOM.**

* [**NgSwitchCase**](https://angular.io/api/common/NgSwitchCase)**agrega su elemento al DOM cuando su valor límite es igual al valor de cambio y elimina su valor límite cuando no es igual al valor de cambio.**
* [**NgSwitchDefault**](https://angular.io/api/common/NgSwitchDefault)**agrega su elemento al DOM cuando no hay seleccionado**[**NgSwitchCase**](https://angular.io/api/common/NgSwitchCase)**.**

**Las directivas de conmutación son particularmente útiles para agregar y eliminar elementos componentes . Este ejemplo cambia entre cuatro**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**componentes definidos en el item-switch.components.tsarchivo. Cada componente tiene un**[**item**](https://angular.io/api/core/IterableChangeRecord#item)[**propiedad de entrada**](https://angular.io/guide/template-syntax#inputs-outputs)**que está vinculada acurrentItem del componente principal.**

**Las directivas de conmutación también funcionan con elementos nativos y componentes web. Por ejemplo, podría reemplazar el<app-best-item> caja interruptor con lo siguiente.**

**src / app / app.component.html**

**content\_copy<div \***[**ngSwitchCase**](https://angular.io/api/common/NgSwitchCase)**="'bright'"> Are you as bright as {{currentItem.name}}?</div>**

## **variables de referencia de plantilla ( #var)**

**Una variable de referencia de plantilla suele ser una referencia a un elemento DOM dentro de una plantilla. También puede hacer referencia a una directiva (que contiene un componente), un elemento,**[**TemplateRef**](https://angular.io/api/core/TemplateRef)**o un**[**componente web**](https://developer.mozilla.org/en-US/docs/Web/Web_Components)**.**

**Para ver una demostración de la sintaxis y los fragmentos de código en esta sección, consulte la publicación**[**ejemplo de variables de referencia de plantilla**](https://angular.io/generated/live-examples/template-reference-variables/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/template-reference-variables/template-reference-variables.zip)**.**

**Use el símbolo hash (#) para declarar una variable de referencia. La siguiente variable de referencia #phone, declara una phonevariable en un <input>elemento.**

**src / app / app.component.html**

**content\_copy<input #phone placeholder="phone number" />**

**Puede hacer referencia a una variable de referencia de plantilla en cualquier lugar de la plantilla del componente. Aquí, <button>más abajo, la plantilla se refiere a la phonevariable.**

**src / app / app.component.html**

**content\_copy<input #phone placeholder="phone number" />**

**<!-- lots of other elements -->**

**<!-- phone refers to the input element; pass its `value` to an event handler -->**

**<button (click)="callPhone(phone.value)">Call</button>**

### **Cómo una variable de referencia obtiene su valor**

**En la mayoría de los casos, Angular establece el valor de la variable de referencia en el elemento en el que se declara. En el ejemplo anterior, se phonerefiere al número de teléfono <input>. El controlador de clic del botón pasa el<input> valor al callPhone()método del componente .**

**La**[**NgForm**](https://angular.io/api/forms/NgForm)**directiva puede cambiar ese comportamiento y establecer el valor en otra cosa. En el siguiente ejemplo, la variable de referencia de plantilla itemForm, aparece tres veces separada por HTML.**

**src / app / hero-form.component.html**

**content\_copy<form #itemForm="**[**ngForm**](https://angular.io/api/forms/NgForm)**" (ngSubmit)="onSubmit(itemForm)">**

**<label for="name"**

**>Name <input class="form-control" name="name"** [**ngModel**](https://angular.io/api/forms/NgModel) **required />**

**</label>**

**<button type="submit">Submit</button>**

**</form>**

**<div [hidden]="!itemForm.form.valid">**

**<p>{{ submitMessage }}</p>**

**</div>**

**El valor de referencia de itemForm, sin el valor del atributo ngForm, sería**[**HTMLFormElement**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement)**. Sin embargo, hay una diferencia entre un Componente y una Directiva en que un**[**Component**](https://angular.io/api/core/Component)**se hará referencia sin especificar el valor del atributo, y a**[**Directive**](https://angular.io/api/core/Directive)**no cambiará la referencia implícita (es decir, el elemento).**

**Sin embargo, con**[**NgForm**](https://angular.io/api/forms/NgForm)**, itemFormes una referencia al**[**NgForm**](https://angular.io/api/forms/NgForm)**directiva con la capacidad de realizar un seguimiento del valor y la validez de cada control en el formulario.**

**El <form>elemento nativo no tiene una formpropiedad, pero la**[**NgForm**](https://angular.io/api/forms/NgForm)**directiva sí, lo que permite deshabilitar el botón de envío si itemForm.form.validno es válido y pasar todo el árbol de control del formulario al componente principalonSubmit() método .**

### **consideraciones de variables de referencia de plantilla**

**Una variable de referencia de plantilla ( #phone) no es lo mismo que una variable de entrada de plantilla ( let phone) como en un**[**\*ngFor**](https://angular.io/guide/template-syntax#template-input-variable)**. Consulte las**[**directivas estructurales**](https://angular.io/guide/structural-directives#template-input-variable)**para obtener más información.**

**El alcance de una variable de referencia es la plantilla completa. Por lo tanto, no defina el mismo nombre de variable más de una vez en la misma plantilla, ya que el valor de tiempo de ejecución será impredecible.**

#### **Sintaxis alternativa**

**Puede usar la ref-alternativa de prefijo a #. Este ejemplo declara la faxvariable como en ref-faxlugar de #fax.**

**src / app / app.component.html**

**content\_copy<input ref-fax placeholder="fax number" />**

**<button (click)="callFax(fax.value)">Fax</button>**

## **@**[**Input**](https://angular.io/api/core/Input)**()y  propiedades@**[**Output**](https://angular.io/api/core/Output)**()**

**@**[**Input**](https://angular.io/api/core/Input)**()y permitir que Angular comparta datos entre el contexto principal y las directivas o componentes secundarios. Un@**[**Output**](https://angular.io/api/core/Output)**()@[Input](https://angular.io/api/core/Input)() propiedad se puede escribir mientras que una propiedad es observable.@**[**Output**](https://angular.io/api/core/Output)**()**

**Considere este ejemplo de una relación niño / padre:**

**content\_copy<parent-component>**

**<child-component></child-component>**

**</parent-component>**

**Aquí, el <child-component>selector, o directiva del niño, está incrustado dentro de a <parent-component>, que sirve como contexto del niño.**

**@**[**Input**](https://angular.io/api/core/Input)**()y actúan como la API, o interfaz de programación de aplicaciones, del componente hijo, ya que permiten que el niño se comunique con el padre. Piense en y como puertos o puertas: es la puerta de entrada al componente que permite que los datos fluyan mientras@**[**Output**](https://angular.io/api/core/Output)**()@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**() que la puerta sale del componente, lo que permite que el componente secundario envíe datos.**

**Esta sección sobre y tiene su propia@**[**Input**](https://angular.io/api/core/Input)**()@[Output](https://angular.io/api/core/Output)()**[**ejemplo en vivo**](https://angular.io/generated/live-examples/inputs-outputs/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/inputs-outputs/inputs-outputs.zip)**. Las siguientes subsecciones resaltan puntos clave en la aplicación de muestra.**

#### **@**[**Input**](https://angular.io/api/core/Input)**()y son  independiente@**[**Output**](https://angular.io/api/core/Output)**()**

**Aunque ya menudo aparecen juntas en las aplicaciones, puede usarlas por separado. Si el componente anidado es tal que solo necesita enviar datos a su padre, no necesitaría un , solo un . Lo contrario también es cierto en que si el niño solo necesita recibir datos del padre, solo lo necesitaría .@[Input](https://angular.io/api/core/Input)()@**[**Output**](https://angular.io/api/core/Output)**()@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()@**[**Input**](https://angular.io/api/core/Input)**()**

## **Cómo usar el @**[**Input**](https://angular.io/api/core/Input)**()**

**Use el decorador en un componente secundario o directiva para que Angular sepa que una propiedad en ese componente puede recibir su valor de su componente principal. Es útil recordar que el flujo de datos es desde la perspectiva del componente secundario. Entonces, permite que los datos se ingresen en el componente secundario desde el componente primario.@[Input](https://angular.io/api/core/Input)()@**[**Input**](https://angular.io/api/core/Input)**()**

**Para ilustrar el uso de , edite estas partes de su aplicación:@**[**Input**](https://angular.io/api/core/Input)**()**

* **La clase y la plantilla del componente hijo**
* **La clase de componente principal y la plantilla**

### **En el  hijo**

**Para usar el decorador en una clase de componente hijo, primero importe y luego decore la propiedad con :@[Input](https://angular.io/api/core/Input)()**[**Input**](https://angular.io/api/core/Input)**@**[**Input**](https://angular.io/api/core/Input)**()**

**src / app / item-detail / item-detail.component.ts**

**content\_copyimport {** [**Component**](https://angular.io/api/core/Component)**,** [**Input**](https://angular.io/api/core/Input) **} from '@angular/core'; //** [**First**](https://angular.io/)**, import** [**Input**](https://angular.io/api/core/Input)

**export class ItemDetailComponent {**

**@**[**Input**](https://angular.io/api/core/Input)**()** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**: string; // decorate the property with @**[**Input**](https://angular.io/api/core/Input)**()**

**}**

**En este caso, decora la propiedad , que tiene un tipo de , sin embargo, las propiedades pueden tener cualquier tipo, tales como , , , o@**[**Input**](https://angular.io/api/core/Input)**()itemstring@**[**Input**](https://angular.io/api/core/Input)**()numberstringbooleanobject . El valor de**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**vendrá del componente principal, que cubre la siguiente sección.**

**A continuación, en la plantilla del componente secundario, agregue lo siguiente:**

**src / app / item-detail / item-detail.component.html**

**content\_copy<p>**

**Today's** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**: {{**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**}}**

**</p>**

### **En el  principal**

**El siguiente paso es vincular la propiedad en la plantilla del componente principal. En este ejemplo, la plantilla del componente principal esapp.component.html .**

**Primero, use el selector del niño, aquí <app-item-detail>, como directiva dentro de la plantilla del componente padre. Luego, utilice el**[**enlace de propiedad**](https://angular.io/guide/template-syntax#property-binding)**para vincular la propiedad del niño a la propiedad del padre.**

**src / app / app.component.html**

**content\_copy<app-item-detail [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem"></app-item-detail>**

**A continuación, en la clase de componente principal app.component.ts, designe un valor para currentItem:**

**src / app / app.component.ts**

**content\_copyexport class AppComponent {**

**currentItem = 'Television';**

**}**

**Con , Angular pasa el valor para al niño para que se procese como@**[**Input**](https://angular.io/api/core/Input)**()currentItem**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**Television .**

**El siguiente diagrama muestra esta estructura:**

**El objetivo entre corchetes [], es la propiedad con la que decoras@**[**Input**](https://angular.io/api/core/Input)**() en el componente secundario. La fuente de enlace, la parte a la derecha del signo igual, son los datos que el componente padre pasa al componente anidado.**

**La conclusión clave es que cuando se vincula a la propiedad de un componente secundario en un componente primario, es decir, lo que está entre corchetes, debe decorar la propiedad con el componente secundario.@[Input](https://angular.io/api/core/Input)()**

#### [**OnChanges**](https://angular.io/api/core/OnChanges)**y @**[**Input**](https://angular.io/api/core/Input)**()**

**Para ver los cambios en una propiedad, use uno de los**[**ganchos**](https://angular.io/guide/lifecycle-hooks#onchanges)**del**[**ciclo**](https://angular.io/guide/lifecycle-hooks#onchanges)**de**[**vida**](https://angular.io/guide/lifecycle-hooks#onchanges)**de Angular . está específicamente diseñado para trabajar con propiedades que tiene el decorador. Consulte la sección de la guía**[**Lifecycle Hooks**](https://angular.io/guide/lifecycle-hooks)**para obtener más detalles y ejemplos.@[Input](https://angular.io/api/core/Input)()**[**OnChangesOnChanges**](https://angular.io/api/core/OnChanges)**@**[**Input**](https://angular.io/api/core/Input)**()**[**OnChanges**](https://angular.io/guide/lifecycle-hooks#onchanges)

## **Cómo usar el @**[**Output**](https://angular.io/api/core/Output)**()**

**Use el decorador en el componente o directiva hijo para permitir que los datos fluyan desde el hijo hacia el padre.@[Output](https://angular.io/api/core/Output)()**

**Normalmente, una propiedad debe inicializarse en un Angular con valores que salen del componente como**[**eventos**](https://angular.io/guide/template-syntax#event-binding)**.@[Output](https://angular.io/api/core/Output)()**[**EventEmitter**](https://angular.io/api/core/EventEmitter)

**Al igual que con , puede usar una propiedad del componente hijo, pero su tipo debería serlo .@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()**[**EventEmitter**](https://angular.io/api/core/EventEmitter)

**@**[**Output**](https://angular.io/api/core/Output)**()marca una propiedad en un componente hijo como una puerta a través de la cual los datos pueden viajar del niño al padre. El componente hijo debe generar un evento para que el padre sepa que algo ha cambiado. Para generar un evento, trabaja de la mano con , que es una clase en@**[**Output**](https://angular.io/api/core/Output)**()**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**@angular/core que se utilizan para emitir eventos personalizados.**

**Cuando lo use , edite estas partes de su aplicación:@**[**Output**](https://angular.io/api/core/Output)**()**

* **La clase y la plantilla del componente hijo**
* **La clase de componente principal y la plantilla**

**El siguiente ejemplo muestra cómo configurar un componente secundario que empuja los datos que ingresa en un HTML a una matriz en el componente primario.@[Output](https://angular.io/api/core/Output)()<input>**

**El elemento HTML <input>y el decorador angular son diferentes. Esta documentación trata sobre la comunicación de componentes en Angular en lo que respecta a y . Para obtener más información sobre el elemento HTML , consulte la**[**Recomendación del W3C**](https://www.w3.org/TR/html5/sec-forms.html#the-input-element)**.@**[**Input**](https://angular.io/api/core/Input)**()@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()<input>**

### **En el  hijo**

**Este ejemplo presenta un lugar <input>donde un usuario puede ingresar un valor y hacer clic en uno <button>que genera un evento. El**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**entonces retransmite los datos al componente padre.**

**Primero, asegúrese de importar**[**Output**](https://angular.io/api/core/Output)**y**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**en la clase de componente hijo:**

**content\_copyimport {** [**Output**](https://angular.io/api/core/Output)**,** [**EventEmitter**](https://angular.io/api/core/EventEmitter) **} from '@angular/core';**

**A continuación, aún en el elemento secundario, decore una propiedad con en la clase de componente. Se llama al siguiente ejemplo y su tipo es , lo que significa que es un evento.@**[**Output**](https://angular.io/api/core/Output)**()@**[**Output**](https://angular.io/api/core/Output)**()newItemEvent**[**EventEmitter**](https://angular.io/api/core/EventEmitter)

**src / app / item-output / item-output.component.ts**

**content\_copy@**[**Output**](https://angular.io/api/core/Output)**() newItemEvent = new** [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<string>();**

**Las diferentes partes de la declaración anterior son las siguientes:**

* **@**[**Output**](https://angular.io/api/core/Output)**()—Una función de decorador que marca la propiedad como una forma de que los datos pasen del niño al padre**
* **newItemEvent-El nombre de @**[**Output**](https://angular.io/api/core/Output)**()**
* [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<string>—El tipo@**[**Output**](https://angular.io/api/core/Output)**()**
* **new**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**<string>()—Dice Angular para crear un nuevo emisor de eventos y que los datos que emite son de tipo cadena. El tipo puede ser cualquier tipo, como number, booleanetc. Para obtener más información sobre**[**EventEmitter**](https://angular.io/api/core/EventEmitter)**, consulte la**[**documentación de la API EventEmitter**](https://angular.io/api/core/EventEmitter)**.**

**A continuación, cree un addNewItem()método en la misma clase de componente:**

**src / app / item-output / item-output.component.ts**

**content\_copyexport class ItemOutputComponent {**

**@**[**Output**](https://angular.io/api/core/Output)**() newItemEvent = new** [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<string>();**

**addNewItem(value: string) {**

**this.newItemEvent.emit(value);**

**}**

**}**

**La addNewItem()función utiliza el , para generar un evento en el que se emite el valor que el usuario escribe en el@**[**Output**](https://angular.io/api/core/Output)**()newItemEvent<input> . En otras palabras, cuando el usuario hace clic en el botón Agregar en la interfaz de usuario, el niño le informa al padre sobre el evento y le da esa información al padre.**

#### **En el  la plantilla del niño**

**La plantilla del niño tiene dos controles. El primero es un archivo HTML <input>con una**[**variable de referencia plantilla**](https://angular.io/guide/template-syntax#ref-var)**, #newItemdonde el usuario escribe un nombre de elemento. Cualquier cosa que el usuario escriba en el <input>se almacena en la #newItemvariable.**

**src / app / item-output / item-output.component.html**

**content\_copy<label>Add an** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**: <input #newItem></label>**

**<button (click)="addNewItem(newItem.value)">Add to parent's list</button>**

**El segundo elemento es una <button> con una**[**unión evento**](https://angular.io/guide/template-syntax#event-binding)**. Usted sabe que es un evento de unión debido a que la parte a la izquierda del signo igual es entre paréntesis, (click).**

**El (click)evento está vinculado al addNewItem()método en la clase de componente hijo que toma como argumento cualquiera que sea el valor de #newItem.**

**Ahora el componente hijo tiene un para enviar datos al padre y un método para generar un evento. El siguiente paso es en el padre.@[Output](https://angular.io/api/core/Output)()**

### **En el  principal**

**En este ejemplo, el componente principal es AppComponent, pero podría usar cualquier componente en el que pudiera anidar al hijo.**

**En AppComponenteste ejemplo, se incluye una lista de items una matriz y un método para agregar más elementos a la matriz.**

**src / app / app.component.ts**

**content\_copyexport class AppComponent {**

**items = ['item1', 'item2', 'item3', 'item4'];**

**addItem(newItem: string) {**

**this.items.push(newItem);**

**}**

**}**

**El addItem()método toma un argumento en forma de una cadena y luego empuja, o agrega, esa cadena a laitems matriz.**

#### **En el  la plantilla del padre**

**Luego, en la plantilla del padre, vincule el método del padre al evento del niño. Ponga el selector de hijo, aquí <app-item-output>, dentro de la plantilla del componente padre, app.component.html.**

**src / app / app.component.html**

**content\_copy<app-item-output (newItemEvent)="addItem($event)"></app-item-output>**

**El enlace de evento (newItemEvent)='addItem($event)', le dice a Angular que conecte el evento en el niño, newItemEvental método en el padre addItem(), y que el evento sobre el que el niño está notificando al padre debe ser el argumento addItem(). En otras palabras, aquí es donde tiene lugar la transferencia real de datos. El $eventcontiene los datos que el usuario escribe en la <input> en la interfaz de usuario plantilla hija.**

**Ahora, para ver el funcionamiento, agregue lo siguiente a la plantilla del padre:@[Output](https://angular.io/api/core/Output)()**

**content\_copy<ul>**

**<li \***[**ngFor**](https://angular.io/api/common/NgForOf)**="let** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **of items">{{**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**}}</li>**

**</ul>**

**El itera sobre los elementos en la matriz. Cuando ingresa un valor en el niño y hace clic en el botón, el niño emite el evento y el método del padre empuja el valor a la matriz y lo muestra en la lista.\***[**ngFor**](https://angular.io/api/common/NgForOf)**items<input>addItem()items**

## **@**[**Input**](https://angular.io/api/core/Input)**()y juntos @**[**Output**](https://angular.io/api/core/Output)**()**

**Puede usar y en el mismo componente secundario como en el siguiente:@[Input](https://angular.io/api/core/Input)()@**[**Output**](https://angular.io/api/core/Output)**()**

**src / app / app.component.html**

**content\_copy<app-input-output [**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**]="currentItem" (deleteRequest)="crossOffItem($event)"></app-input-output>**

**El objetivo,**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**que es una propiedad de la clase componente secundario, recibe su valor de la propiedad de los padres, . Cuando hace clic en Eliminar, el componente hijo genera un evento , que es el argumento para el método del padre .@**[**Input**](https://angular.io/api/core/Input)**()currentItemdeleteRequestcrossOffItem()**

**El siguiente diagrama es de una y una en el mismo componente hijo y muestra las diferentes partes de cada una:@[Input](https://angular.io/api/core/Input)()@**[**Output**](https://angular.io/api/core/Output)**()**

**Como muestra el diagrama, use las entradas y salidas juntas de la misma manera que las usa por separado. Aquí, el selector secundario es <app-input-output>with**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**y deleteRequestbeing y propiedades en la clase de componente secundario. La propiedad y el método están en la clase de componente principal.@[Input](https://angular.io/api/core/Input)()@**[**Output**](https://angular.io/api/core/Output)**()currentItemcrossOffItem()**

**Para combinar enlaces de propiedades y eventos utilizando la sintaxis banana-in-a-box [()], consulte**[**Enlace bidireccional**](https://angular.io/guide/template-syntax#two-way)**.**

**Para obtener más detalles sobre cómo funcionan, consulte las secciones anteriores sobre**[**Entrada**](https://angular.io/guide/template-syntax#input)**y**[**Salida**](https://angular.io/guide/template-syntax#output)**. Para verlo en acción, vea el**[**Ejemplo de entradas y salidas**](https://angular.io/generated/live-examples/inputs-outputs/stackblitz.html)**/**[**ejemplo de descarga**](https://angular.io/generated/zips/inputs-outputs/inputs-outputs.zip)**.**

## **@**[**Input**](https://angular.io/api/core/Input)**()y  declaraciones@**[**Output**](https://angular.io/api/core/Output)**()**

**En lugar de usar los decoradores y para declarar entradas y salidas, puede identificar miembros en las matrices y de los metadatos de la directiva, como en este ejemplo:@[Input](https://angular.io/api/core/Input)()@**[**Output**](https://angular.io/api/core/Output)**()inputsoutputs**

**src / app / in-the-metadata / in-the-metadata.component.ts**

**content\_copy// tslint:disable: no-inputs-metadata-property no-outputs-metadata-property**

**inputs: ['clearanceItem'],**

**outputs: ['buyEvent']**

**// tslint:enable: no-inputs-metadata-property no-outputs-metadata-property**

**Si bien es posible declarar inputsy outputsen los metadatos y , es una mejor práctica utilizar los decoradores de clase y , en su lugar, de la siguiente manera:@**[**Directive**](https://angular.io/api/core/Directive)**@**[**Component**](https://angular.io/api/core/Component)**@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()**

**src / app / input-output / input-output.component.ts**

**content\_copy@**[**Input**](https://angular.io/api/core/Input)**()** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**: string;**

**@**[**Output**](https://angular.io/api/core/Output)**() deleteRequest = new** [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<string>();**

**Consulte la sección**[**Decorar propiedades de entrada y salida**](https://angular.io/guide/styleguide#decorate-input-and-output-properties)**de la**[**Guía de estilo**](https://angular.io/guide/styleguide)**para más detalles.**

**Si obtiene un error de análisis de plantilla cuando intenta usar entradas o salidas, pero sabe que las propiedades existen, verifique que sus propiedades estén anotadas con / o que las haya declarado en una matriz / :@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()inputsoutputs**

**content\_copyUncaught** [**Error**](https://angular.io/api/core/MissingTranslationStrategy#Error)**:** [**Template**](https://angular.io/) **parse errors:**

**Can't bind to '**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**' since it isn't** [**a**](https://angular.io/api/router/RouterLinkWithHref) **known property of 'app-item-detail'**

## **entradas y salidas de**

**A veces, el nombre público de una propiedad de entrada / salida debe ser diferente del nombre interno. Si bien es una buena práctica evitar esta situación, Angular ofrece una solución.**

### **Alias ​​en el  metadatos**

**Alias ​​entradas y salidas en los metadatos usando una :cadena delimitada por dos puntos ( ) con el nombre de propiedad de la directiva a la izquierda y el alias público a la derecha:**

**src / app / aliasing / aliasing.component.ts**

**content\_copy// tslint:disable: no-inputs-metadata-property no-outputs-metadata-property**

**inputs: ['input1: saveForLaterItem'], // propertyName:alias**

**outputs: ['outputEvent1: saveForLaterEvent']**

**// tslint:disable: no-inputs-metadata-property no-outputs-metadata-property**

### **Alias ​​con el  / decorador@**[**Input**](https://angular.io/api/core/Input)**()@**[**Output**](https://angular.io/api/core/Output)**()**

**Puede especificar el alias para el nombre de la propiedad pasando el nombre del alias al / decorador. El nombre interno permanece como siempre.@[Input](https://angular.io/api/core/Input)()@**[**Output**](https://angular.io/api/core/Output)**()**

**src / app / aliasing / aliasing.component.ts**

**content\_copy@**[**Input**](https://angular.io/api/core/Input)**('wishListItem') input2: string; // @**[**Input**](https://angular.io/api/core/Input)**(alias)**

**@**[**Output**](https://angular.io/api/core/Output)**('wishEvent') outputEvent2 = new** [**EventEmitter**](https://angular.io/api/core/EventEmitter)**<string>(); // @**[**Output**](https://angular.io/api/core/Output)**(alias) propertyName = ...**

## **operadores de expresión de plantilla**

**El lenguaje de expresión de plantilla angular emplea un subconjunto de sintaxis de JavaScript complementado con algunos operadores especiales para escenarios específicos. Las siguientes secciones cubren tres de estos operadores:**

* [**tubo**](https://angular.io/guide/template-syntax#pipe)
* [**operador de navegación segura**](https://angular.io/guide/template-syntax#safe-navigation-operator)
* [**operador de aserción no nulo**](https://angular.io/guide/template-syntax#non-null-assertion-operator)

### **El  operador de tubería ( |)**

**El resultado de una expresión puede requerir alguna transformación antes de que esté listo para usarlo en un enlace. Por ejemplo, puede mostrar un número como moneda, cambiar el texto a mayúsculas o filtrar una lista y ordenarla.**

**Las tuberías son funciones simples que aceptan un valor de entrada y devuelven un valor transformado. Son fáciles de aplicar dentro de expresiones de plantilla, utilizando el operador de tubería ( |):**

**src / app / app.component.html**

**content\_copy<p>**[**Title**](https://angular.io/api/platform-browser/Title) **through uppercase pipe: {{title |** [**uppercase**](https://angular.io/api/common/UpperCasePipe)**}}</p>**

**El operador de tubería pasa el resultado de una expresión a la izquierda a una función de tubería a la derecha.**

**Puede encadenar expresiones a través de múltiples tuberías:**

**src / app / app.component.html**

**content\_copy<!-- convert title to uppercase, then to lowercase -->**

**<p>**[**Title**](https://angular.io/api/platform-browser/Title) **through** [**a**](https://angular.io/api/router/RouterLinkWithHref) **pipe chain: {{title |** [**uppercase**](https://angular.io/api/common/UpperCasePipe) **|** [**lowercase**](https://angular.io/api/common/LowerCasePipe)**}}</p>**

**Y también puede**[**aplicar parámetros**](https://angular.io/guide/pipes#parameterizing-a-pipe)**a una tubería:**

**src / app / app.component.html**

**content\_copy<!-- pipe with configuration argument => "February 25, 1980" -->**

**<p>Manufacture date with date format pipe: {{item.manufactureDate | date:'longDate'}}</p>**

**La**[**json**](https://angular.io/api/common/JsonPipe)**canalización es particularmente útil para depurar enlaces:**

**src / app / app.component.html**

**content\_copy<p>Item json pipe: {{**[**item**](https://angular.io/api/core/IterableChangeRecord#item) **|** [**json**](https://angular.io/api/common/JsonPipe)**}}</p>**

**La salida generada se vería así:**

**content\_copy{ "name": "Telephone",**

**"manufactureDate": "1980-02-25T05:00:00.000Z",**

**"price": 98 }**

**El operador de tubería tiene una precedencia más alta que el operador ternario ( ?:), lo que significa que**[**a**](https://angular.io/api/router/RouterLinkWithHref)**? b : c | xse analiza como**[**a**](https://angular.io/api/router/RouterLinkWithHref)**? b : (c | x). Sin embargo, por varias razones, el operador de tubería no puede usarse sin paréntesis en el primer y segundo operandos de ?:. Una buena práctica es usar paréntesis en el tercer operando también.**

### **El operador de navegación segura ( ?) y las rutas de propiedades nulas**

**El operador de navegación segura angular ?, protege contra nully undefined valores en las rutas de propiedad. Aquí, protege contra una falla de renderizado de vista si**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**es así null.**

**src / app / app.component.html**

**content\_copy<p>The** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **name is: {{**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**?.name}}</p>**

**Si**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**es así null, la vista aún se muestra pero el valor mostrado está en blanco; solo verá "El nombre del elemento es:" sin nada después.**

**Considere el siguiente ejemplo, con a nullItem.**

**content\_copyThe null** [**item**](https://angular.io/api/core/IterableChangeRecord#item) **name is {{nullItem.name}}**

**Dado que no hay ningún operador navegación segura y nullItemes null, JavaScript y angular lanzaría un nullerror de referencia y romper el proceso de prestación de Angular:**

**content\_copyTypeError: Cannot** [**read**](https://angular.io/api/core/Query#read) **property 'name' of null.**

**Sin embargo, a veces, los nullvalores en la ruta de la propiedad pueden estar bien en ciertas circunstancias, especialmente cuando el valor comienza siendo nulo pero los datos llegan eventualmente.**

**Con el operador de navegación segura ?, Angular deja de evaluar la expresión cuando alcanza el primer nullvalor y muestra la vista sin errores.**

**Funciona perfectamente con rutas de propiedad largas como**[**a**](https://angular.io/api/router/RouterLinkWithHref)**?.b?.c?.d.**

### **El operador afirmación no nulo ( !)**

**A partir de Typecript 2.0, puede aplicar**[**una comprobación nula estricta**](http://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-0.html)**con--strictNullChecks indicador. Luego, TypeScript asegura que ninguna variable sea involuntariamente nula o indefinida.**

**En este modo, las variables escritas no están permitidas nully undefinedpor defecto. El verificador de tipo arroja un error si deja una variable sin asignar o intenta asignar nullo undefineda una variable cuyo tipo no permite nully undefined.**

**El verificador de tipo también arroja un error si no puede determinar si una variable será nullo no definida en tiempo de ejecución. Le dice al verificador de tipos que no arroje un error aplicando el**[**operador de afirmación no nulo**](http://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-0.html#non-null-assertion-operator)**postfix,! .**

**El operador de aserción angular no nulo !, sirve el mismo propósito en una plantilla angular. Por ejemplo, después de usar**[**\* ngIf**](https://angular.io/guide/template-syntax#ngIf)**para verificar que**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**está definido, puede afirmar que las**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**propiedades también están definidas.**

**src / app / app.component.html**

**content\_copy<!--No color, no error -->**

**<p \***[**ngIf**](https://angular.io/api/common/NgIf)**="**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**">The** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**'s color is: {{**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**!.color}}</p>**

**Cuando el compilador Angular convierte su plantilla en código TypeScript, evita que TypeScript informe que**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**podría ser nullo undefined.**

**A diferencia del**[**operador de navegación segura**](https://angular.io/guide/template-syntax#safe-navigation-operator)**, el operador de aserción no nulo no protege contra nullo undefined. Más bien, le dice al verificador de tipos TypeScript que suspenda las nullverificaciones estrictas para una expresión de propiedad específica.**

**El operador de aserción no nulo !es opcional, con la excepción de que debe usarlo cuando activa las comprobaciones nulas estrictas.**

[**volver arriba**](https://angular.io/guide/template-syntax#top-of-page)

## **funciones de plantilla incorporado**

### **El $any() función de**

**A veces, una expresión de enlace desencadena un error de tipo durante la**[**compilación AOT**](https://angular.io/guide/aot-compiler)**y no es posible o difícil especificar completamente el tipo. Para silenciar el error, puede usar la $any()función de conversión para convertir la expresión al**[**anytipo**](http://www.typescriptlang.org/docs/handbook/basic-types.html#any)**como en el siguiente ejemplo:**

**src / app / app.component.html**

**content\_copy<p>The** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**'s undeclared best by date is: {{$any([item](https://angular.io/api/core/IterableChangeRecord" \l "item)).bestByDate}}</p>**

**Cuando el compilador Angular convierte esta plantilla en código TypeScript, evita que TypeScript informe que bestByDateno es miembro del**[**item**](https://angular.io/api/core/IterableChangeRecord#item)**objeto cuando ejecuta la verificación de tipos en la plantilla.**

**La $any()función de conversión también funciona thispara permitir el acceso a miembros no declarados del componente.**

**src / app / app.component.html**

**content\_copy<p>The** [**item**](https://angular.io/api/core/IterableChangeRecord#item)**'s undeclared best by date is: {{$any(this).bestByDate}}</p>**

**los $any() función de conversión funciona en cualquier lugar de una expresión de enlace donde una llamada de método es válida.**

## **SVG en el  plantillas**

**Es posible usar SVG como plantillas válidas en Angular. Toda la sintaxis de la plantilla a continuación es aplicable tanto a SVG como a HTML. Obtenga más información en las especificaciones SVG**[**1.1**](https://www.w3.org/TR/SVG11/)**y**[**2.0**](https://www.w3.org/TR/SVG2/)**.**

**¿Por qué usaría SVG como plantilla, en lugar de simplemente agregarlo como imagen a su aplicación?**

**Cuando usa un SVG como plantilla, puede usar directivas y enlaces al igual que con las plantillas HTML. Esto significa que podrá generar dinámicamente gráficos interactivos.**

**Consulte el fragmento de código de muestra a continuación para ver un ejemplo de sintaxis:**

**src / app / svg.component.ts**

**content\_copyimport {** [**Component**](https://angular.io/api/core/Component) **} from '@angular/core';**

**@**[**Component**](https://angular.io/api/core/Component)**({**

**selector: 'app-svg',**

**templateUrl: './svg.component.svg',**

[**styleUrls**](https://angular.io/api/core/Component#styleUrls)**: ['./svg.component.css']**

**})**

**export class SvgComponent {**

**fillColor = 'rgb(255, 0, 0)';**

**changeColor() {**

**const r = Math.floor(Math.random() \* 256);**

**const g = Math.floor(Math.random() \* 256);**

**const b = Math.floor(Math.random() \* 256);**

**this.fillColor = `rgb(${r}, ${g}, ${b})`;**

**}**

**}**

**Agregue el siguiente código a su svg.component.svgarchivo:**

**src / app / svg.component.svg**

**content\_copy<svg>**

**<g>**

**<rect x="0" y="0" width="100" height="100" [attr.fill]="fillColor" (click)="changeColor()" />**

**<text x="120" y="50">click the rectangle to change the fill color</text>**

**</g>**

**</svg>**

**Aquí puede ver el uso de un click()enlace de evento y la sintaxis de enlace de propiedad ( [attr.fill]="fillColor").**