




Entrada del usuario

Las acciones del usuario, como hacer clic en un vínculo, presionar un botón e introducir texto, generan eventos DOM. En esta página se explica cómo enlazar esos eventos a controladores de eventos de componentes mediante la sintaxis de enlace de eventos Angular.

Ejecutar el [ejemplo en vivo](#) / ejemplo de [descarga](#) .

Enlace a eventos de entrada del usuario

Puede usar [enlaces de eventos Angular](#) para responder a cualquier evento [DOM](#) . Muchos eventos DOM se desencadenan por la entrada del usuario. El enlace a estos eventos proporciona una manera de obtener la entrada del usuario.

Para enlazar a un evento DOM, rodee el nombre del evento DOM entre paréntesis y asígnelo una [instrucción de plantilla](#) entre comillas.

En el ejemplo siguiente se muestra un enlace de eventos que implementa un controlador de clics:

```
src/app/click-me.component.ts
```

```
<button (click)="onClickMe()">Click me!</button>
```

El a la izquierda del signo igual identifica el evento click del botón como **el destino del enlace**. El texto entre comillas a la derecha del signo igual es la **instrucción template**, que responde al evento click llamando al método del componente. `(click)` `onClickMe`

Al escribir un enlace, tenga en cuenta el contexto de **ejecución** de una instrucción de plantilla. Los identificadores de una instrucción de plantilla pertenecen a un objeto de contexto específico, normalmente el componente Angular que controla la plantilla. El ejemplo anterior muestra una sola línea de HTML, pero ese HTML pertenece a un componente más grande:

```
src/app/click-me.component.ts
```

```
@Component({
  selector: 'app-click-me',
  template: `
    <button (click)="onClickMe()">Click me!</button>
    {{clickMessage}}`
})
export class ClickMeComponent {
  clickMessage = '';

  onClickMe() {
    this.clickMessage = 'You are my hero!';
  }
}
```

Cuando el usuario hace clic en el botón, Angular llama al método desde

`.onClickMe` `ClickMeComponent`

Obtener la entrada del usuario del objeto \$event

Los eventos DOM llevan una carga de información que puede ser útil para el componente. En esta sección se muestra cómo enlazar al evento de un cuadro de entrada para obtener la entrada del usuario después de cada pulsación de tecla. `keyup`

El código siguiente escucha el evento y pasa toda la carga del evento () al controlador de eventos del componente. `keyup` `$event`

```
src/app/keyup.components.ts (plantilla v.1)
```

```
template: `
  <input (keyup)="onKey($event)">
  <p>{{values}}</p>
`
```

Cuando un usuario presiona y suelta una tecla, se produce el evento y Angular proporciona un objeto de evento DOM correspondiente en la variable que este código pasa como parámetro al método del componente. `keyup` `$event` `onKey()`

src/app/keyup.components.ts (clase v.1)

```
export class KeyUpComponent_v1 {  
  values = '';  
  
  onKey(event: any) { // without type info  
    this.values += event.target.value + ' | ';  
  }  
}
```

Las propiedades de un objeto varían en función del tipo de evento DOM. Por ejemplo, un evento de mouse incluye información diferente a un evento de edición de cuadro de entrada. `$event`

Todos los [objetos de evento DOM estándar](#) tienen una propiedad, una referencia al elemento que generó el evento. En este caso, hace referencia al elemento `<input>` y devuelve el contenido actual de ese elemento. `target.target.event.target.value`

Después de cada llamada, el método anexa el contenido del valor del cuadro de entrada a la lista de la propiedad del componente, seguido de un carácter separador (-). La [interpolación](#) muestra los cambios del cuadro de entrada de acumulación desde la propiedad. `onKey()valuesvalues`

Suppose the user enters the letters "abc", and then backspaces to remove them one by one. Here's what the UI displays:

a | ab | abc | ab | a | |

Give me some keys!

Alternatively, you could accumulate the individual keys themselves by substituting for in which case the same user input would produce: `event.keyevent.target.value`

a | b | c | backspace | backspace | backspace |

Type the \$event

El ejemplo anterior convierte el como un tipo. Esto simplifica el código a un costo. No hay información de tipo que pueda revelar las propiedades del objeto de evento y evitar errores tontos. `$event any`

En el ejemplo siguiente se reescribe el método con tipos:

src/app/keyup.components.ts (clase v.1 - mecanograbada)

```
export class KeyUpComponent_v1 {  
  values = '';  
  
  onKey(event: KeyboardEvent) { // with type info  
    this.values += (event.target as HTMLInputElement).value + ' | ';  
  }  
}
```

El ahora es un archivo . específico No todos los elementos tienen una propiedad por lo que se convierte en un elemento de entrada. El método expresa más claramente lo que espera de la plantilla y cómo interpreta el evento. `$event KeyboardEvent value target OnKey`

Pasar \$event es una práctica dudosa

Escribir el objeto de evento revela una objeción significativa a pasar todo el evento DOM al método: el componente tiene demasiada conciencia de los detalles de la plantilla. No puede extraer información sin saber más de lo que debería sobre la implementación HTML. Esto interrumpe la separación de preocupaciones entre la plantilla (*lo que ve el usuario*) y el componente (*cómo la aplicación procesa los datos de usuario*).

En la siguiente sección se muestra cómo utilizar variables de referencia de plantilla para solucionar este problema.

Obtener la entrada del usuario de una variable de referencia de plantilla

Hay otra forma de obtener los datos de usuario: utilice [variables de referencia de plantilla](#) angular. Estas variables proporcionan acceso directo a un elemento desde dentro de la plantilla. Para declarar una variable de referencia de plantilla, anteponga un identificador con un carácter hash (o libra).

En el ejemplo siguiente se utiliza una variable de referencia de plantilla para implementar un bucle invertido de pulsación de tecla en una plantilla simple.

```
src/app/loop-back.component.ts
```

```
@Component({
  selector: 'app-loop-back',
  template: `
    <input #box (keyup)="0">
    <p>{{box.value}}</p>
  `
})
export class LoopbackComponent { }
```

La variable de referencia de plantilla denominada `box`, declarada en el elemento, hace referencia al propio elemento. El código utiliza la variable para obtener el elemento de entrada y mostrarlo con interpolación entre etiquetas. `box<input><input>box value<p>`

La plantilla es completamente independiente. No se enlaza al componente y el componente no hace nada.

Escriba algo en el cuadro de entrada y observe cómo se actualiza la pantalla con cada pulsación de tecla.

keyup loop-back component

↵

Esto no funcionará en absoluto a menos que se enlace a un evento.

Angular actualiza los enlaces (y, por lo tanto, la pantalla) solo si la aplicación hace algo en respuesta a eventos asíncronos, como pulsaciones de teclas. Este código de ejemplo enlaza el evento al número 0, la instrucción de plantilla más corta posible. Aunque la instrucción no hace nada útil, satisface los requisitos de Angular para que Angular actualice la pantalla. `keyup`

Es más fácil llegar al cuadro de entrada con la variable de referencia de plantilla que pasar por el objeto. Esta es una reescritura del ejemplo anterior que utiliza una variable de referencia de plantilla para obtener la entrada del usuario. `$event keyup`

src/app/keyup.components.ts (v2)

```
@Component({
  selector: 'app-key-up2',
  template: `
    <input #box (keyup)="onKey(box.value)">
    <p>{{values}}</p>
  `,
})
export class KeyUpComponent_v2 {
  values = '';
  onKey(value: string) {
    this.values += value + ' | ';
  }
}
```

Un buen aspecto de este enfoque es que el componente obtiene valores de datos limpios de la vista. Ya no requiere conocimiento de la estructura y su estructura. `$event`

Filtrado de eventos clave (con `key.enter`)

El controlador de eventos escucha *cada pulsación de tecla*. A veces sólo importa la tecla *Intro*, porque indica que el usuario ha terminado de escribir. Una forma de reducir el ruido sería examinar cada uno y tomar medidas solo cuando la clave es *Intro*. `(keyup)$event.keyCode`

Hay una manera más fácil: enlazar al pseudo-evento de Angular. A continuación, Angular llama al controlador de eventos solo cuando el usuario presiona *Intro*. `keyup.enter`

src/app/keyup.components.ts (v3)

```
@Component({
  selector: 'app-key-up3',
  template: `
    <input #box (keyup.enter)="onEnter(box.value)">
    <p>{{value}}</p>
  `,
})
export class KeyUpComponent_v3 {
  value = '';
  onEnter(value: string) { this.value = value; }
}
```

Así es como funciona.

Type away! Press [enter] when done

En el desenfoque

En el ejemplo anterior, el estado actual del cuadro de entrada se pierde si el usuario se aleja y hace clic en otro lugar de la página sin presionar primero *Intro*. La propiedad del componente se actualiza solo cuando el usuario presiona *Intro*. `value`

Para solucionar este problema, escuche la tecla *Intro* y el evento *de desenfoque*.

src/app/keyup.components.ts (v4)

```
@Component({
  selector: 'app-key-up4',
  template: `
    <input #box
      (keyup.enter)="update(box.value)"
      (blur)="update(box.value)">

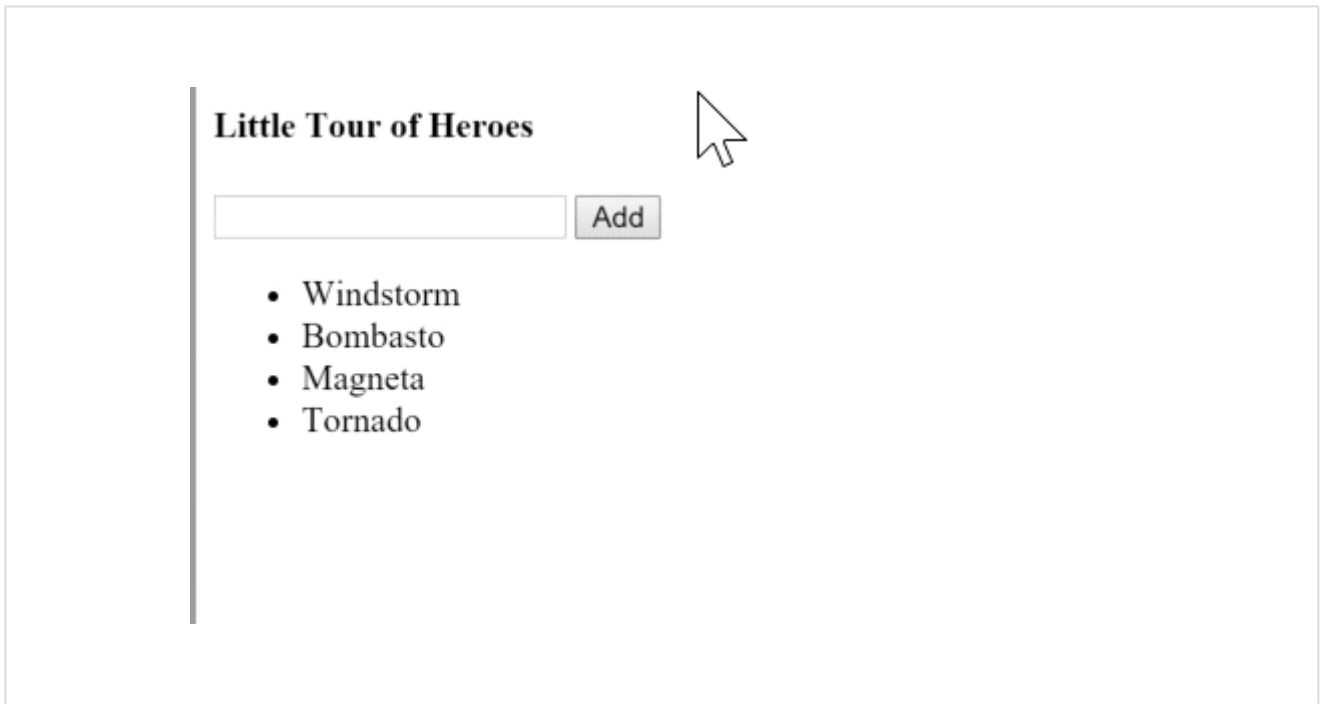
    <p>{{value}}</p>
  `
})
export class KeyUpComponent_v4 {
  value = '';
  update(value: string) { this.value = value; }
}
```

Ponlo todo junto

La página anterior mostraba cómo [mostrar datos](#). En esta página se muestran técnicas de enlace de eventos.

Ahora, ponlo todo en una micro-app que puede mostrar una lista de héroes y añadir nuevos héroes a la lista. El usuario puede agregar un héroe escribiendo el nombre del héroe en el cuadro de entrada y

haciendo clic en **Agregar**.



A continuación se muestra el componente "Little Tour of Heroes".

src/app/little-tour.component.ts

```
@Component({
  selector: 'app-little-tour',
  template: `
    <input #newHero
      (keyup.enter)="addHero(newHero.value)"
      (blur)="addHero(newHero.value); newHero.value='' ">

    <button (click)="addHero(newHero.value)">Add</button>

    <ul><li *ngFor="let hero of heroes">{{hero}}</li></ul>
  `
})
export class LittleTourComponent {
  heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
  addHero(newHero: string) {
    if (newHero) {
      this.heroes.push(newHero);
    }
  }
}
```


Observaciones

- **Usar variables de plantilla para hacer referencia a elementos** : la variable de plantilla hace referencia al elemento. Puede hacer referencia desde cualquier elemento secundario o secundario del elemento. `newHero<input>newHero<input>`
- **Pasar valores, no elementos** — En lugar de pasar el método del componente, obtenga el valor del cuadro de entrada y páselo: `newHeroaddHeroaddHero`
- **Mantener las instrucciones de plantilla simples**: el evento está enlazado a dos instrucciones JavaScript. La primera instrucción llama a `addHero`. La segunda instrucción `addHero`, borra el cuadro de entrada después de agregar un nuevo héroe a la lista. `(blur)addHeronewHero.value=''`

Código fuente

A continuación se muestra todo el código que se describe en esta página.

`click-me.component.ts`

`keyup.components.ts`

`loop-back.component.ts`

`little-tour`

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

@Component({
  selector: 'app-click-me',
  template: `
    <button (click)="onClickMe()">Click me!</button>
    {{clickMessage}}`
})
export class ClickMeComponent {
  clickMessage = '';

  onClickMe() {
    this.clickMessage = 'You are my hero!';
  }
}
```

Angular también admite detectores de eventos pasivos. Por ejemplo, puede usar los pasos siguientes para que el evento de desplazamiento sea pasivo.

1. Cree un archivo en el directorio. `zone-flags.ts` `src`
2. Agregue la siguiente línea a este archivo.

```
(window as any)['__zone_symbol__PASSIVE_EVENTS'] = ['scroll'];
```

3. En el archivo, antes de importar zone.js, importe el archivo `src/polyfills.ts` `zone-flags`

```
import './zone-flags';  
import 'zone.js/dist/zone'; // Included with Angular CLI.
```

Después de esos pasos, si agrega detectores de eventos para el evento, los agentes de escucha serán `scrollpassive`

Resumen

Ha dominado los primitivos básicos para responder a la entrada y los gestos del usuario.

Estas técnicas son útiles para demostraciones a pequeña escala, pero rápidamente se vuelven detalladas y torpes al manejar grandes cantidades de entradas de usuario. El enlace de datos bidireccional es una forma más elegante y compacta de mover valores entre los campos de entrada de datos y las propiedades del modelo. En la página siguiente, se explica cómo escribir enlaces bidireccionales con `Forms` `NgModel`