

Event Binding In Angular

4 Comments / March 9, 2023 / 4 minutes of reading

← [Property Binding](#)

[Angular Tutorial](#)

[Two Way data Binding](#) →

In this guide, we will explore the Event Binding in Angular. Event binding is one way from view to component. We use it to perform an action in the component when the user performs an action like clicking on a button, changing the input, etc in the view.

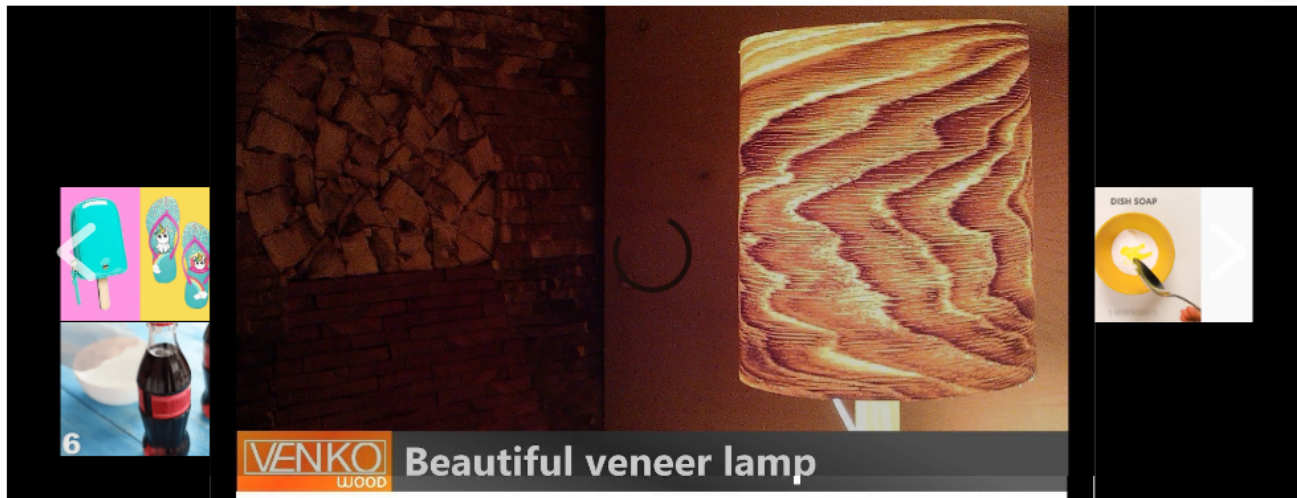


Table of Contents

[Event Binding](#)

[Syntax](#)

[Event Binding Example](#)

[Template statements have side effects](#)

[on-](#)

[Multiple event handlers](#)

[\\$event Payload](#)

Template reference variable

Key event filtering (with `key.enter`)

Custom events with `EventEmitter`

Event Binding

Event binding allows us to bind events such as keystroke, clicks, hover, touche, etc to a method in component. It is one way from view to component. By tracking the user events in the view and responding to it, we can keep our component in sync with the view. For Example, when the user changes to an input in a text box, we can update the model in the component, run some validations, etc. When the user submits the button, we can then save the model to the backend server.

Syntax

The Angular event binding consists of two parts

```
1  
2 (target-event)="TemplateStatement"  
3
```

- We enclose the target event name in parentheses on the left side
- Assign it to a template statement within a quote on the right side

Angular event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

The following event binding listens for the button's `click` events, calling the component's `onSave()` method whenever a click occurs

```
1  
2 <button (click)="onSave()">Save</button>  
3
```

Event Binding in Angular

`<button (click)="onSave()">Save</button>`

Target Event

Template Statement

Event Binding Example

Create a new angular application

```
1  
2 ng new event  
3
```

Copy the following code to `app.component.html`

app.component.html

```
1  
2 <h1 [innerText]="title"></h1>  
3  
4 <h2>Example 1</h2>  
5 <button (click)="clickMe()">Click Me</button>  
6 <p>You have clicked {{clickCount}}</p>  
7
```

[Source Code](#)

Add the following code to the `app.component.ts`

app.component.ts

```
1  
2 clickCount=0  
3 clickMe() {
```

```
4   this.clickCount++;  
5   }  
6
```

[Source Code](#)

In the above example, the component listens to the click event on the button. It then executes the `clickMe()` method and increases the `clickCount` by one.

Template statements have side effects

Unlike the [Property Binding](#) & [Interpolation](#), where we use the template expression is used, in the case of event binding we use template statement.

The Template statement can change the state of the component. Angular runs the change detection and updates the view so as to keep it in sync with the component.

on-

Instead of parentheses, you can also use the `on-` syntax as shown below.

```
1  
2 <button on-click="clickMe()">Click Me</button>  
3
```

Multiple event handlers

You can also bind an unlimited number of event handlers on the same event by separating them with a semicolon ;

Add a new component property

```
1  
2 clickCount1=0;  
3
```

[Source Code](#)

And in the template, call `clickMe()` method and then an assignment

```
clickCount1=clickCount
```

```
1
2 //Template
3
4 <h2>Example 2</h2>
5 <button (click)="clickMe() ; clickCount1=clickCount">Click Me</button>
6 <p>You have clicked {{clickCount}}</p>
7 <p>You have clicked {{clickCount1}}</p>
8
```

[Source Code](#)

\$event Payload

DOM Events carries the event payload. I.e the information about the event. We can access the event payload by using `$event` as an argument to the handler function.

```
1
2 <input (input)="handleInput($event)">
3 <p>You have entered {{value}}</p>
4
```

[Source Code](#)

And in the component

```
1
2 value=""
3 handleInput(event) {
4   this.value = (event.target as HTMLInputElement).value;
5 }
6
```

[Source Code](#)

The properties of a `$event` object vary depending on the type of DOM event. For

example, a mouse event includes different information than an input box editing event.

Remember you need to use the variable as `$event` in the Template statement. Example `handleInput($event)`. Otherwise, it will result in an error

Template reference variable

We can also make use of the template reference variable to pass the value instead of `$event`.

In the template

```
1
2 <h2>Template Reference Variable</h2>
3 <input #el (input)="handleInput1(el)">
4 <p>You have entered {{val}}</p>
5
```

[Source Code](#)

```
1
2 val="";
3 handleInput1(element) {
4   this.val=element.value;
5 }
6
7
```

[Source Code](#)

Key event filtering (with key.enter)

We use `keyup/keydown` events to listen for keystrokes. The following example does that

```
1
2 <input (keyup)="value1= $any($event.target).value" />
3 <p>You entered {{value1}}</p>
4
```

[Source Code](#)

But Angular also offers a feature, where it helps to filter out certain keys. For Example, if you want to listen only to the `enter` keys you can do it easily

```
1
2 <input (keyup.enter)="value2=$any($event.target).value">
3 <p>You entered {{value2}}</p>
4
```

[Source Code](#)

Here is an interesting example. On pressing `enter` key it updates the `value3` variable and on `escape` clears the variable.

```
1
2 <input (keyup.enter)="value3=$any($event.target).value" (keyup.escape)="$any($event.target).value=null">
3 <p>You entered {{value3}}</p>
4
```

[Source Code](#)

Note that we are using `$any` to cast `$event.target` to [any type](#). Otherwise, the typescript will raise the error [Property 'value' does not exist on type 'EventTarget'](#)
[Error in Angular](#)

Angular calls these pseudo-events.

You can also listen for the key combination

```
1
2 <input (keyup.control.shift.enter)="value4=$any($event.target).value">
3 <p>You entered {{value4}}</p>
4
```

[Source Code](#)

Custom events with EventEmitter

CUSTOM EVENTS WITH EVENTEMITTER

Directives & components can also raise events with [EventEmitter](#). Using [EventEmiiter](#) you can create a property and raise it using the `EventEmitter.emit(payload)`. The Parent component can listen to these events using the event binding and also read the payload using the `$event` argument.

← [Property Binding](#)

[Angular Tutorial](#)

[Two Way data Binding](#) →

4 thoughts on “Event Binding in Angular”



ANONYMOUS

OCTOBER 14, 2021 AT 6:52 AM

In the multiple event handlers section, the semicolon should be “;” and not “:”, maybe thinking of correcting it so it wont be confused 😊

[Reply](#)



ANONYMOUS

AUGUST 24, 2021 AT 5:29 AM

Great tutorial for beginners!!

[Reply](#)



R

MAY 15, 2021 AT 6:05 AM

hi, awesome tutorial btw

```
value4=$any($event.target).value
```

why is the value outside parenthesis? like below

```
value4=$any($event.target.value)
```

Reply



ANONYMOUS

MAY 15, 2021 AT 8:24 AM

`$event.target.value` will result in an Typescript compiler error as type of `$event.target` is not clear

Hence we convert `$event.target` to any using `$any($event.target)` and then access the value property.

Read

<https://www.tektutorialshub.com/angular/property-value-does-not-exist-on-type-eventtarget-error-in-angular/>

Reply

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Website

Post Comment »

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)



Copyright © 2023 TekTutorialsHub |