Event Binding in Angular

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

← 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

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

```

(target-event)="TemplateStatement"

```

- 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

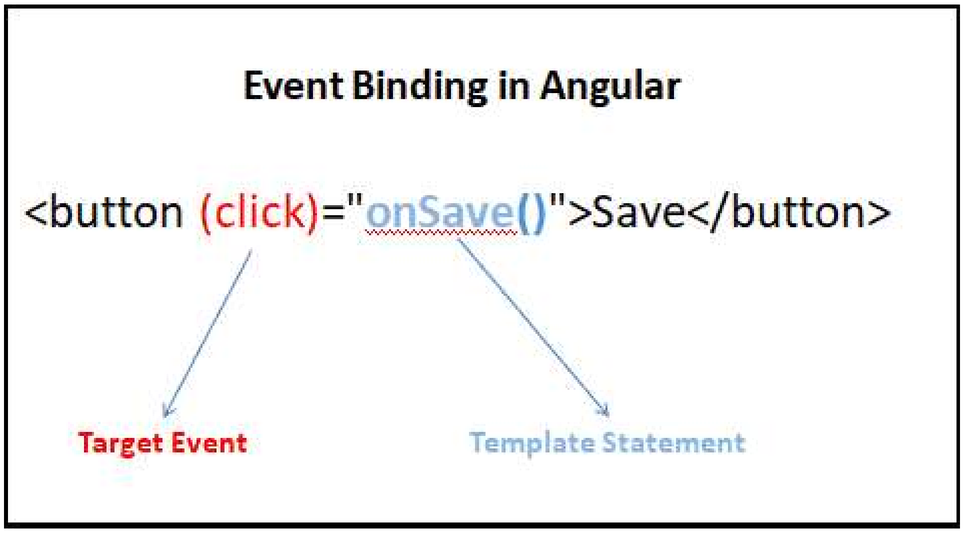
Angular event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

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

```

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

```



Event Binding Example

Create a new angular application

```
ng new event
```

Copy the following code to app.component.html

app.component.html

```
<h1 [innerText]="title"></h1>

<h2>Example 1</h2>
<button (click)="clickMe()">Click Me</button>
<p>You have clicked {{clickCount}}</p>
```

Source Code

Add the following code to the app.component.ts

app.component.ts

```
clickCount=0
  clickMe() {
    this.clickCount++;
  }

```

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.

```

<button on-click="clickMe()">Click Me</button>

```

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

```

```

```
clickCount1=0;

```

Source Code

And in the template, call `clickMe()` method and then an assignment `clickCount1=clickCount`

```

//Template

<h2>Example 2</h2>
<button (click)="clickMe() ; clickCount1=clickCount">Click Me</button>
<p>You have clicked {{clickCount}}</p>
<p>You have clicked {{clickCount1}}</p>

```

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.

```

<input (input)="handleInput($event)">
<p>You have entered {{value}}</p>

```

Source Code

And in the component

```

value=""
handleInput(event) {
  this.value = (event.target as HTMLInputElement).value;
}

```

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

```

<h2>Template Reference Variable</h2>
<input #el (input)="handleInput1(el)">
<p>You have entered {{val}}</p>

```

Source Code

```

val="";
handleInput1(element) {
  this.val=element.value;
}


```

Source Code

Key event filtering (with key.enter)

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

```

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

```

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

```

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

```

Source Code

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

```

<input (keyup.enter)="value3=$any($event.target).value" (keyup.escape)="$any($event.ta
<p>You entered {{value3}}</p>

```

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

```

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

```

Source Code

Custom events with EventEmitter

Directives & components can also raise events with EventEmitter. Using EventEmiiiter 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 →

Related Posts