Event Binding in Angular

Introduction

In this guide, we will explore on the topic of event binding in Angular. Event binding will help to build interactive web applications with the flow of data from component to the element and from element to component - both ends.

In many cases, users will not only just view the information or data on web applications or mobile applications, but will also interact with these applications using different user actions like clicks, keystrokes, change events, etc.

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

Syntax: (event)

Let's consider an example where we are binding an onClick() event to the button element. When the user clicks on the button, event binding listens to the button's click event and calls the component's onClick() method.

File Name: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

import { Component } from "@angular/core";

@Component({

selector: 'app-example',

template: `

<div>

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

</div>

`

})

export class ExampleComponent {

onClick(){

alert("You Clicked Me!");

}

}

typeScript

Below are some of the different ways and scenarios of event binding.

Target Event Binding

The target event is identified by the name within the parenthesis, ex: (click), which represents the click event. In the example, above we saw the target click event bound to the 'onClick()' method, which will listen to the button's click event.

1

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

html

We can also use the prefix on-, in event binding this is known as canonical form.

1

<button on-click = "onClick()">Click me!</button>

html

If the name of the target event does not match with the element's event, then Angular will throw an error "unknown directive".

$event Handling and Event Handling Statements

In event binding, we are binding an event handler for the target event. Whenever we perform some operations, an event will be raised. The event handler then executes the template statement. The template handler will have a receiver, which will perform the operation based on the event received and then respond. One such response would be storing a value from view to an array in the component.

If the event is a native DOM element event then the $event is a DOM element object with different properties like target and target.value.

File Name: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

import { Component } from "@angular/core";

import { Person } from '../person';

@Component({

selector: 'app-example',

template: `

<div>

<input [value]="person.name"

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

</div>

`

})

export class ExampleComponent {

person: Person;

}

typeScript

In the above example we can see 'person.name' bound to $event.target.value.

Binding Custom Events with EventEmitter

Syntax: EventEmitter.emit(payload)

We can create custom events using EventEmitter and expose their properties. We can fire an event by calling the EventEmitter.emit(payload) passing payload which can contain any value. The parent, or the component to which we are passing the value, will listen for the event by binding to this property and accessing the payload through the $event object.

Let's consider an example where we have an ExampleComponent that presents ‘person’ information and responds to user actions. Although the ExampleComponent has a click button, it doesn't know how to click the person itself. The best it can do is raise an event reporting the user's click request.

File Name: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

import { Component } from "@angular/core";

import { Person } from '../person';

@Component({

selector: 'app-example',

template: `

<img src="{{personImageUrl}}">

<span [style.text-decoration]="lineThrough">

{{prefix}} {{person?.name}}

</span>

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

</div>

`

})

export class ExampleComponent {

clickRequest = new EventEmitter<Person>();

onClick() {

this.clickRequest.emit(this.person);

}

}

typeScript

In the example above, component defines a clickRequest property that returns an EventEmitter. When the user clicks the 'Click me!' button, the component invokes the onClick() method, telling the EventEmitter to emit a Person object.

Now, let's consider a parent component that binds to the ExampleComponent's clickRequest event.

1

<app-person-details (clickRequest)="clickPerson($event)" [person]="personName"></app-person-details>

html

When the clickRequest event fires, Angular calls the parent component's clickPerson method, passing the person-to-click (emitted by PesonDetail) in the $event variable.

Conclusion

In this guide, we have explored the Event Binding technique in Angular. We have also seen different methods or ways through which we can bind an event to a DOM element.

As we know two-way data binding involves both the property binding and the event binding. You can learn about two-way data binding in my guide [One-way and Two-way Data Binding in Angular](https://app.pluralsight.com/guides/one-and-two-way-data-binding-angular).