How event binding works

In an event binding, Angular configures an event handler for the target event. You can use event binding with your own custom events.

When the component or directive raises the event, the handler executes the template statement. The template statement performs an action in response to the event.

Handling events

A common way to handle events is to pass the event object, $event, to the method handling the event. The $event object often contains information the method needs, such as a user's name or an image URL.

The target event determines the shape of the $event object. If the target event is a native DOM element event, then $event is a DOM event object, with properties such as target and target.value.

In the following example the code sets the <input> value property by binding to the name property.

src/app/app.component.html

content_copy<input [value]="currentItem.name"

(input)="currentItem.name=getValue($event)">

With this example, the following actions occur:

1. The code binds to the input event of the <input> element, which allows the code to listen for changes.
2. When the user makes changes, the component raises the input event.
3. The binding executes the statement within a context that includes the DOM event object, $event.
4. Angular retrieves the changed text by calling getValue($event.target) and updates the name property.

If the event belongs to a directive or component, $event has the shape that the directive or component produces.

The type of $event.target is only EventTarget in the template. In the getValue() method, the target is cast to an HTMLInputElement to allow type-safe access to its value property.

content_copygetValue(event: Event): string {

return (event.target as HTMLInputElement).value;

}

(This article is sourced from: https://angular.io/guide/event-binding-concepts and the content of the article is here in case of link breakage.)