Jonathan Roland

Web 425 Angular with TypeScript

Prof. Richard Krasso

2/3/2021

Discussion 6.1 – Input and Output Properties

Angular provides the means to address a common paradigm in managing inputs and outputs in a component hierarchy. When data needs to be shared between parent and child components, Angular provides the @Input() and @Output() directives.

When components are in a direct hierarchy, one component wraps another. The wrapping component is considered the ‘parent’, while the wrapped component is considered the ‘child’. For the parent component to send data to the child, the @Input() directive is used. This is configured in the child component by first importing Input from Angular core, then decorating a property with @Input():

import { Component, Input } from ‘@angular/core’;

export class ChildComponent {

@Input() example: string;

}

Then update the child component’s html as shown in the below snippet:

<p>

An example: {{ example }}

</p>

Next, the parent component must be configured by binding the child component’s exportable property (named ‘example’ above). We want to do this in the parent’s html template by identifying the child component’s selector element and adding a property binding reference to the child component element’s exported property:

<app-child-element [example]=”testExample”></app-child-element>

The reference to “testExample” above must then be matched in the parent element’s class:

export class ParentComponent {

testExample = ‘This is an example’;

}

So here if we had this structure in the parent component’s HTML:

<app-parent-element>

<app-child-element [example]=”testExample”>

<p>

An example: {{ example }}

</p>

</app-child-element>

</app-parent-element>

The parent class would be able to change the value of the ‘example’ property because it has been identified and bound to the parent. Since it is defaulted to the value listed in the class above, the front-end output would be:

This is an example

If we wanted instead to send data from the child to the parent component, we would start similarly by importing Output from the Angular core. The difference is that here we use Angular’s EventEmitter type for the example property. We also include the addNewExample() function to allow the value of eventExample to be emitted.

import { Output, EventEmitter } from ‘@angular/core’;

export class ChildOutputComponent {

@Output() eventExample = new EventEmitter<string>();

addNewExample(value: string){

this.eventExample.emit(value);

}

}

In the child template, we can demonstrate with a label, input and button. The input uses a template reference variable called #newExample to store a value entered by the user. We register an on click event to the addNewExample() function.

<label>Type an example: <input #newExample></label>

<button (click)=”addNewExample(newExample.value)”>Add an example string</button>

To display this value in the parent template, let’s imagine the parent has a simple header element that we want to update the text of when the child component emits a new example value. We need to configure the parent component to listen for the child component’s event:

<app-parent-element (eventExample)=”updateHeader($event)”>

<h3>{{ exampleText }}</h3>

</app-parent-element>

The updateHeader() function in the parent component takes in the value of exampleText and makes it available in the parent component’s scope, which is then referenced within the h3 above.

export class ParentComponent {

updateHeader(newExample: string){

this.exampleText = newExample;

}

}

The result would be that whatever is typed in the child component input updates the text of the parent component’s h3 element.

It should be noted that input and output can be used simultaneously for two-way data flow between parent and child components.

**Reference:**

Angular. (n.d.). *Angular*. Retrieved February 4, 2021, from https://angular.io/guide/inputs-outputs