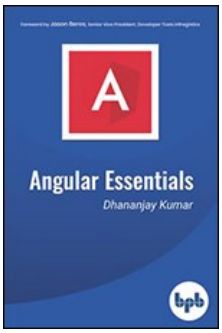

Angular Essentials: The Essential Guide to Learn Angular

by Dhananjay Kumar

Chapter 4: Angular Directives

In this chapter, you *learn* will about Angular Directives. Following topics will be covered in this chapter:

- What i**s Di**rectiv**e**s
- Structural D**irectives**
- Custom Attribute Directives
- @HostListener in Attribute Directives
- @HostBinding in Attribute Directives

What is Directives

Directives creates DOM elements , change their structure, or behavior in an Angular applicatian. There are three types of directives in Angular:

- Components : Directives w**ith tem**plate
- Attribute Directives : Change appearance and behavior of an element, component, or another directives
- Structural Directives : Change DOM layout by adding or removing elements

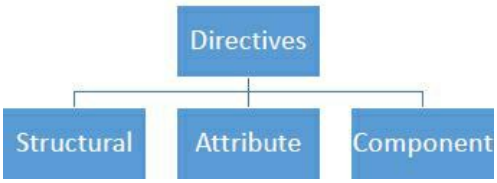

Figure 4.1

The basic difference between a component and a directive is that a component has a template, whereas an attribute or structural directive does not have a template. Angular has provided us many inbuilt structural and attribute directives. Inbuilt structural directives are **`*ngFor, *ngIf`** and attribute directives as **`NgStyle`** and **`NgModel`**.

Structural Directives

Structural directive changes the structure of DOM elements. For example, **`*ngIf`** is a structure directive which is used to provide an 'if' condition to the statements to be executed. If the expression evaluates to a False value, the elements are removed from the DOM whereas if it evaluates to True, then element is added to the DOM.

Consider *code listing 4.1,* in this **`*nglf`** directive will add div in DOM is value of **`showMessage`** property is true.

Code Listing 4.1

```
@Component({
   selector:   'app-message',
   template:'
   <div *ngIf =  'showMessage'>
      Show Message
   </div>
    `
})
export class AppMessageComponent {
         showMessage    = true;
   }
```

Keep in mind that **`*nglf`** will does not hide or show DOM element. Rather, it adds or removes depending on the condition.

`*ngFor` structure directive creates DOM elements in a loop. Consider *code listing 4.2,* in this **`*ngFor`** directive will add rows in a table depending on number of items in data array.

Code Listing 4.2

```
@Component({
   selector:   'app-message',
   template: `
   <table>
   <tr *ngFor='let f of data'>
   <td>{{f.name}}</td>
   </tr>
  </table>
 `
})
export class AppMessageComponent {
         data = [
         {name  :   'foo'},
         {name:   'koo'}
      ];
}
```

In most of the cases, you will not have to create custom structural directive and built-in directives should be enough.

Custom Attribute Directive

Attribute Directives change appearance and behavior of an element, component, or another directives. Angular provides many attribute directives such as **`NgStyle`** and **`NgModel`**. We have seen use of **`NgModel`** in previous sections. In this section, let us create a custom Attribute Directive. To do this, we need to create a class and decorate it with **`@ directive`** decorators. A simple attribute directive to change the color of an element can be created as shown in the *code listing 4.3:*

Code Listing 4.3

```
import { Directive, ElementRef, Renderer } from '@
angular/core';

@Directive({
      selector:   '[appChbgcolor]'
})
export class ChangeBgColorDirective {

constructor(private el: ElementRef, private renderer:
Renderer)   {
         this.ChangeBgColor('red');
   }
   ChangeBgColor(color:  string)   {
      this.renderer.setElementStyle(this.l.nativeEleme
nt,'color',  color);
      }
}
```

To create a custom attribute directive, you need to create a class and decorate it with **`@Directive`**. In the constructor of the directive class, inject the services **`ElementRef`** and **`Renderer`**. Instances of these two classes are needed to get the reference of the host element and of the renderer.

While we are creating an attribute directive to change the color of the element, keep in mind that we do not need to create a new attribute directive to just change the color; simple colors can be changed by using property binding. However, the attribute directive we created will change color of an element in this example. There are few important points to remember:

- Import required modules like Directive, ElementRef, and Renderer from Angular core library.
- Create a TypeScript class.
- Decorate the class with @directive.
- Set the value of the selector property in @directive decorator function. The directive would be used, using the selector value on the elements.
- In the constructor of the class, inject ElementRef and Renderer object.

We are injecting **`ElementRef`** in the directive's constructor to access the DOM element. We are also injecting Renderer in the

directive's constructor to work with DOM's element style. We are calling the renderer's **`setElementStyle`** function. In the function, we pass the current DOM element by using the object of **`ElementRef`** and setting the color style property of the current element. We can use this attribute directive by its selector in **`AppComponent`** as shown in the *code listing 4.4.*

Code Listing 4.4

```
@Component(
   {
      selector:   'app-container',
      template:  '<p appChbgcolor >{{message}}</p>'
   })
export class AppComponent {
```

We used an attribute directive on a paragraph element. It will change the color of the paragraph text to red. In addition to use a directive, we need to import and declare the attribute directive at the **`app.module.ts`**. Right now, the **`appChbgcolor`** directive will change the color of the host element.

`@HostListener()` in Attribute Directives

In Angular, the **`@HostListener()`** function decorator allows you to handle events of the host element in the directive class.

Let us take the following requirement: when you hover mouse on the host element, only the color of the host element should change. In addition, when the mouse is gone, the color of the host element should change to its default color. To do this, you need to handle events raised on the host element in the directive class. In Angular, you do this using **`@ HostListener()`**.

To understand **`@HostListener()`** in a better way, consider another simple scenario: on the click of the host element, you want to show an alert window. To do this in the directive class, add **`@HostListener()`** and pass the event 'click' to it. Also, associate a function to raise an alert as shown in the *code listing 4.5:*

Code Listing 4.5

```
@HostListener('click')  onClick()   {
   window.alert('Host Element Clicked');
}
```

In Angular, the **`@HostListener()`** function decorator makes it super easy to handle events raised in the host element inside the directive class. Let us go back to our requirement that says you must change the color to red only when the mouse is hovering, and when it is gone, the color of the host element should change to black. To do this, you need to handle the **`mouseenter`** and **`mouseexit`** events of the host element in the directive class. To achieve this, modify the **`appChbgcolor`** directive class as shown in *code listing 4.6:*

Code Listing 4.6

```
import { Directive, ElementRef, Renderer, HostListener
}  from '@angular/core';

@Directive({
   selector:   '[appChbgcolor]'
})
export class ChangeBgColorDirective {
   constructor(private      el:       ElementRef,       private
renderer:  Renderer)   {
         // this.ChangeBgColor('red');
   }

   @HostListener('mouseover')  onMouseOver()   {
         this.ChangeBgColor('red');
   }
     @HostListener('click')  onClick()   {
         window.alert('Host Element Clicked');
   }
@HostListener('mouseleave')  onMouseLeave()   {
         this.ChangeBgColor('black');
   }
   ChangeBgColor(color:  string)   {
         this.renderer.setElementStyle(this.
el.nativeElement,   'color',  color);
```

```
    }
}
```

In the directive class, we are handling the **mouseenter** and **mouseexit** events. As you see, we are using **@HostListener()** to handle these host element events and assigning a function to it. So, let's use **@ HostListener()** function decorator to handle events of the host element in the directive class.

@HostBinding() in Attribute Directives

In Angular, the **@HostBinding()** function decorator allows you to set the properties of the host element from the directive class.

Let's say you want to change the style properties such as height, width, color, margin, border etc. or any other internal properties of the host element in the directive class. Here, you'd need to use the **@HostBinding()** decorator function to access these properties on the host element and assign a value to it in directive class.

The **@HostBinding()** decorator takes one parameter, the name of the host element property which value we want to assign in the directive.

In our example, our host element is a HTML **div** element. If you want to set border properties of the host element, you can do that using **@ HostBinding()** decorator as shown in *code listing 4.7:*

Code Listing 4.7

```
@HostBinding('style.border')  border:  string;

@HostListener('mouseover')  onMouseOver()   {
   this.border =  '5px solid green';
}
```

Using this code, on a mouse hover, the host element border will be set to a green, solid 5-pixel width. Therefore, using **@HostBinding** decorator, you can set the properties of the host element in the directive class.

Summary

Directives are essential of an Angular application. You use structural directives to manipulate structure of DOM and use attribute directives to change attribute of DOM Elements. In this chapter, you learnt following topics:

- What is Directives
- Structural Directives
- Custom Attribute Directives
- `@HostListener` in Attribute Directives
- `@HostBinding` in Attribute Directives