What is the difference between component and directive?

Asked 3 years, 8 months ago Active 3 months ago Viewed 51k times

71

20

I have just started working with Angular 2.

I was wondering what are the differences between components and directives in Angular 2?

angular components directive

edited Jul 18 '18 at 11:56
Sangwin Gawande
4,276 7 29 55

asked Jan 5 '16 at 13:36
uksz
8,831 18 58 117

This is explained in the docs on the Attributes Directives page, the first section, "Directives overview". – Mark Rajcok Jan 5 '16 at 15:19

4 Possible duplicate of @Directive v/s @Component in angular2 – John May 17 '16 at 10:50

7 Answers

88

Basically there are three types of directives in Angular2 according to documentation.

- Component
- Structural directives
- Attribute directives

Component

It is also a type of directive with template,styles and logic part which is most famous type of directive among all in Angular2. In this type of directive you can use other directives whether it is custom or builtin in the `@Component` annotation like following:

```
@Component({
    selector: "my-app"
    directives: [custom_directive_here]
})
```

Use this directive in your view as:

```
<my-app></my-app>
```

For the component directive i have found best tutorial here.

Structural directives

Like `*ngFor` and `*ngIf` , used to change the DOM layout by adding and removing DOM elements. explained here

Attribute directives

They are used to give custom behavior or style to the existing elements by applying some functions/logic. Like `ngStyle` is an attribute directive to give style dynamically to the elements. We can create our own directive and use this as attribute of some predefined or custom elements, here is the example of a simple directive:

Firstly we have to import directive from `@angular/core`

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

@Directive({
  selector: '[Icheck]',
})
export class RadioCheckbox {
   // custom logic here...
}
```

We can use this in the view as shown below:

```
<span Icheck>HEllo Directive</span>
```

For more info you can read the official tutorial here and here edited Jun 10 at 11:19

answered Jan 5 '16 at 16:11
Pardeep Jain
47.5k 21 112 160

Which kind is the „router-outlet“ directive of? It matches IMO none of the three types above. – user2516186 Apr 13 '18 at 3:58

1 @lealceldeiro thank you for the update – Pardeep Jain Jun 10 at 11:19

58

Components have their own view (HTML and styles). Directives are just "behavior" added to existing elements and components. `Component` extends `Directive` .

Because of that there can only be one component on a host element, but multiple directives.

Structural directives are directives applied to `<template>` elements and used to add/remove content (stamp the template). The `*` in directive applications like `*ngIf` causes a `<template>` tag to be created implicitly.

edited Feb 2 '18 at 10:59

answered Jan 5 '16 at 13:44
Günter Zöchbauer
366k 83 1168 1059

7

To complete what Günter said, we can distinguish two kinds of directives:

- The [structural ones](#) that updates the DOM layout by adding or removing elements. The two common ones are `NgFor` and `NgIf` . These ones are linked to the template concept and must be prefixed by an `*` . See the section "Templates and *" in this link for more details: http://victorsavkin.com/post/119943127151/angular-2-template-syntax
- The [attribute ones](#) that updates the behavior of the appearance of the element they are attached one.

Hope it helps you, Thierry

answered Jan 5 '16 at 13:53
Thierry Templier
157k 29 341 311

1 Do not see the purpose of attribute directives. What do they offer beyond CSS? – Tim McNamara Jan 5 '16 at 14:33

3 @TimMcNamara, Angular directives can have logic/methods, hence you can do more with an attribute directive than you can with just CSS. You could

pass in some parent property value into an attribute directive and have the element appear or behave differently based on that property value. – Mark Rajcok Jan 5 '16 at 15:17

You can find a good example here: angular.io/docs/ts/latest/guide/attribute-directives.html – Joris Brauns Sep 22 '16 at 13:22

2

Angular 2 follows component/Service model of architecture.

An angular 2 Application is made of components. A component is the combination of an HTML template and a component class (A typescript class) that controls a portion of the screen.

For the good practice, component class is used for data binding to the respective view. Two way data binding is a great feature provided by angular framework.

Components are reusable across your application using selector name provided.

Component is also a kind of directive with a template.

Other two directives are

1. Structural directives—change the DOM layout by adding and removing DOM elements. Ex: `NgFor` and `NgIf` .
2. Attribute directives—change the appearance or behavior of an element, component, or another directive. Ex: `NgStyle`

edited Jun 2 '17 at 10:13
Derick
2,386 4 24 34

answered Jun 2 '17 at 9:06
Malatesh Patil
2,243 1 6 11

2

Here is the actual definition.

- If it has a *template*, it is a **Component**
- else if it has a *selector in brackets* "[likethis]", it is an **Attribute Directive**
- else it is a **Structural Directive**.

Any other definition is wrong.

answered Jul 17 '18 at 15:03
John Henckel

4,109 1 35 50

1

Summary:

A component is a directive with an associated view (i.e. HTML to be rendered). All components are directives, but not all directives are components. There are three types of directives:

- **Component**: A view with associated behaviour. This type of directive actually adds DOM elements
- **Attribute directives**: Can be attached to DOM elements (and components since they are DOM elements) to modify the appearance or behaviour of an element.
- **Structural directives**: Can be attached to DOM elements (and components since they are DOM elements) to modify the DOM layout. Structural directives start with a * and actually add or remove DOM element. For example `*ngIf` which can insert or remove an DOM element (or angular component which is a custom DOM element, but still a DOM element).

Example:

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

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
 }
}

@Component({
  selector: 'app-root',
  template: `
    <div *ngIf='myBool' appHighlight>Hi there</div>
  `,
  styleUrls: ['./app.component.scss'],
})
export class AppComponent  {

  myBool:boolean = true;

}
```

In the above example we can observe the following:

- The component `AppComponent` has a template with a `<div>` element which displays, hi there.
- The attribute directive HighlightDirective is located on the `<div>` element. This means it will manipulate the behaviour of the `<div>` element. In this case it will highlight the text and will turn it yellow.
- The structural directive `*ngIf` is also located on the `<div>` element and will determine if the element is to be inserted. The `<div>` will be conditionally shown depending on whether the expression `myBool` can be coerced to `true` .

answered Nov 8 '18 at 21:27
Willem van der Veen
7,729 4 54 55

0

Actually components are also directives, but have differences between them.

> **Attribute Directives** :

Attribute directives are classes that are able to modify the behavior or appearance of a single element. For creating an attribute directive apply the `@Directive` to a class.

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

@Directive({
     selector: "[custom-attr]", })

export class CustomAttrDirective {

   constructor(element: ElementRef) {
      element.nativeElement.classList.add("bg-success", "text-white");
   }
}
```

Adding a directive attribute template.html File

```
<tr *ngFor="let item of getProducts(); let i = index" custom-attr>
   <td>{{i + 1}}</td>
   <td>{{item.name}}</td>
</tr>
```

> **Structural Directives** :

Structural directives change the layout of the HTML document by adding and removing elements, as a micro-templates. Structural directives allow content to be added conditionally based on the result of an expression such as `*ngIf` or for the same content to be repeated for each object in a data source such as `*ngFor` .

You can use the built-in directives for common tasks, but writing custom structural directives provides ability to tailor behavior to your application.

```
<p *ngIf="true">
  Expression is true and ngIf is true.
  This paragraph is in the DOM.
</p>
<p *ngIf="false">
  Expression is false and ngIf is false.
  This paragraph is not in the DOM.
</p>
```

> **Components** :

Components are directives that their own templates, rather than relying on content provided from elsewhere. Components have access to all directive features, still have a host element, can still define input and output properties, and so on.But they also define their own content.

It can be easy to underestimate the importance of the template, but attribute and structural directives have limitations. Directives can do useful and powerful work, but they don't have much insight into the elements they are applied to. Directives are most useful when they are general-purpose tools, such the `ngModel` directive, which can be applied to any data model property and any form element, without regard to what the data or the element is being used for.

Components, by contrast, are closely tied to the contents of their templates. Components provide the data and logic that will be used by the data bindings that are applied to the HTML elements in the template, which provide the context used to evaluate data binding expressions and act as the glue between the directives and the rest of the application. Components are also a useful tool in allowing large Angular projects to be broken up into manageable chunks.

```
import { Component, Input } from '@angular/core';

import { Hero } from './hero';

@Component({
  selector: 'app-hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @Input('master') masterName: string;
}
```

from official angular

from Pro-Angular book

answered May 26 at 9:11

fgul
440 6 9