Angular Directives

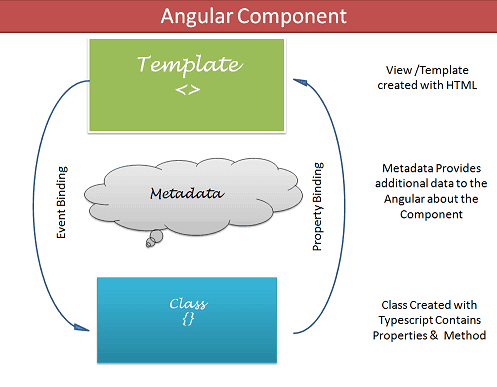
Angular Directives are the most important features of Angular. In this, tutorial We will look at three types of directives that Angular supports like Component, Structural, and Attribute Directives. We also look at the few of the most commonly used Angular directives.

## What is Angular Directive

The Angular directive helps us to manipulate the DOM. You can change the appearance, behavior, or layout of a DOM element using the Directives. They help you to extend HTML

There are three kinds of directives in Angular:

1. Component Directive
2. Structural directives
3. Attribute directives



## Building blocks of the Angular Components

### Template (View)

The template defines the layout and content of the View. Without the template,  there is nothing for Angular to render to the DOM.

The Templates are nothing but HTML codes along with the Angular specific special HTML markups ( knows as the Angular Template Syntax).

You can add [Angular directives](https://www.tektutorialshub.com/angular/angular-directives/) , [Angular Pipes](https://www.tektutorialshub.com/angular/angular-pipes/) & Other Angular Components on the template.

The data to Template comes from the Component, which in turn gets it from a [Angular Service](https://www.tektutorialshub.com/angular/angular-services/). Using the [data binding](https://www.tektutorialshub.com/angular/angular-data-binding/) techniques, we can keep the Template in sync with the Component. The templates can use the [Event Binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) or [two way data binding](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/) to notify the component, when user changes something on the View.

There are two ways you can specify the Template in Angular.

1. Defining the Template Inline
2. Provide an external Template

### Class

The Class provides the data & logic to the View. It contains the JavaScript code associated with Template (View). We use [TypeScript](https://www.tektutorialshub.com/typescript-tutorial/) to create the class, but you can also use JavaScript directly in the class.

Class Contains the Properties & Methods. The Properties of a class can be bind to the view using [Data Binding](https://www.tektutorialshub.com/angular/angular-data-binding/).

The simple Angular Class

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | export class AppComponent  {      title : string ="app"  } |

By convention we prefix the Component class with Component so as to easily identify them.

### Metadata

Metadata Provides additional information about the component to the Angular. Angular uses this information to process the class. We use the @Component decorator to provide the Metadata to the Component.

#### @Component decorator

A decorator is a function that adds metadata to class, its methods & to its properties. The Components are defined with a @Component class decorator.

When Angular sees a class with @Component decorator, it treats the class as Component.

A Decorator is always prefixed with @. We must place the Decorator immediately before the class definition. We can also build our own decorators. The decorators are Similar to attributes in C#

#### Important Component metadata properties

##### Selector

Selector specifies the simple CSS selector. The Angular looks for the CSS selector in the template and renders the component there.

##### Providers

The Providers are the [Angular Services](https://www.tektutorialshub.com/angular/angular-services/), that our component going to use. The Services provide service to the Components or to the other Services.

##### Directives

The[directives](https://www.tektutorialshub.com/angular/angular-directives/) that this component going to use are listed here.

##### Styles/styleUrls

The CSS styles or style sheets, that this component needs. Here we can use either external stylesheet (using styleUrls) or inline styles (using Styles). The styles used here are specific to the component

##### template/templateUrl

The HTML template that defines our View. It tells Angular how to render the Component’s view. The templates can be inline (using a template) or we can use an external template (using a templateUrl). The Component can have only one template. You can either use inline template or external template and not both

## Creating the Component

We have already shown how to create the Angular Application using Angular CLI in [how to create first Angular application](https://www.tektutorialshub.com/angular/angular-create-first-application/) tutorial.  The Angular CLI has automatically created the root component **app.component.ts**.

In this tutorial, we will not create the Angular Component, but let us see the Component creation process in detail. The creation of the Angular component requires you to follow these steps

1. Create the Component file
2. Import the required external Classes/Functions
3. Create the Component class and export it
4. Add @Component decorator
5. Add metadata to @Component decorator
6. Create the Template
7. Create the CSS Styles
8. Register the Component in Angular Module

### 1. Creating the Component File

The Component app.component.ts. is already been created for us by Angular CLI under the folder src.

By Convention, the file name starts with the **feature name** (app) and then followed by the **type of class**(component). These are separated by a dot. The extension used is **ts** indicating that this is a typescript module file.

You can read more about naming conventions from the [Angular Style Guide](https://angular.io/guide/styleguide)

### 2. Import the Angular Component Library

Before we use any Angular (or external) functions or classes, we need to tell Angular how and where to find it. This is done using the Import statement. The Import statement is similar to the using statement in c#, which allows us to use the external modules in our class

To define the Component class, we need to use the @Component decorator. This function is part of the Angular Core library. So we import it in our class as shown below

|  |  |
| --- | --- |
| 1  2  3 | import { Component } from '@angular/core'; |

### 3. Create the Component Class and export it

The third step is to create the Component class using the export keyword. The Export keyword allows other components to use this component importing it. The AppComponent class is shown below

|  |  |
| --- | --- |
| 1  2  3  4  5 | export class AppComponent {    title = 'app';  } |

Note we are using Pascal case naming conventions for the class name. In the above class defines a Property named title and assign a default value “app”

### 4. Add @Component decorator

The next step is to inform the Angular that this is a Component class. We do that by adding the @Component decorator. We must add the decorator immediately above the class definition.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Component({  })  export class AppComponent {    title = 'app';  } |

### 5. Add meta data to @Component decorator

The next step is to add the metadata to the component using the @component decorator. Add the following to the component metadata

# Angular ngFor Directive

Open the app.component.ts and add the following code. The code contains a list of Top 10 movies. Let us build a template to display the movies using ngFor.

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title: string ="Top 5 Movies" ;

movies: Movie[] =[

{title:'Zootopia',director:'Byron Howard, Rich Moore',cast:'Idris Elba, Ginnifer Goodwin, Jason Bateman',releaseDate:'March 4, 2016'},

{title:'Batman v Superman: Dawn of Justice',director:'Zack Snyder',cast:'Ben Affleck, Henry Cavill, Amy Adams',releaseDate:'March 25, 2016'},

{title:'Captain American: Civil War',director:'Anthony Russo, Joe Russo',cast:'Scarlett Johansson, Elizabeth Olsen, Chris Evans',releaseDate:'May 6, 2016'},

{title:'X-Men: Apocalypse',director:'Bryan Singer',cast:'Jennifer Lawrence, Olivia Munn, Oscar Isaac',releaseDate:'May 27, 2016'},

{title:'Warcraft',director:'Duncan Jones',cast:'Travis Fimmel, Robert Kazinsky, Ben Foster',releaseDate:'June 10, 2016'},

]

CompositeKey (index,item){

return item.title + item.director ;

}

}

class Movie {

title : string;

director : string;

cast : string;

releaseDate : string;

}

Using ngFor

To use ngFor,

Create a block of HTML elements, which can display a single movie.

Use the ngFor to repeat the block for each movie in the movies.

Open the app.component.html and add the following code.

<h1> {{title}} </h1>

<ul>

<li \*ngFor="let movie of movies">

{{ movie.title }} - {{movie.director}}

</li>

</ul>

We use the ul to display the movies. The li element displays a single movie. We need to repeat the li for each movie. Hence we apply the ngFor on the li element.

let movie of movies will iterate over the movies collection, which is a property on the component class. movie is the Template input variable, which represents the currently iterated movie from the collection. We use the Angular Interpolation to display the movie title & name of the director

Here is the output

The Angular generates the following code. You can see li element for every movie.

<ul \_ngcontent-gop-c0="">

<li \_ngcontent-gop-c0=""> Zootopia - Byron Howard, Rich Moore </li>

<li \_ngcontent-gop-c0=""> Batman v Superman: Dawn of Justice - Zack Snyder </li>

<li \_ngcontent-gop-c0=""> Captain American: Civil War - Anthony Russo, Joe Russo </li>

<li \_ngcontent-gop-c0=""> X-Men: Apocalypse - Bryan Singer </li>

<li \_ngcontent-gop-c0=""> Warcraft - Duncan Jones </li>

</ul>

Similarly, you can use the table element to display the movies as shown below. Here we need to repeat the tr element for each movie. Hence apply the directive on tr

<div class='panel panel-primary'>

<div class='panel-heading'>

{{title}}

</div>

<div class='panel-body'>

<div class='table-responsive'>

<table class='table'>

<thead>

<tr>

<th>Title</th>

<th>Director</th>

<th>Cast</th>

<th>Release Date</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let movie of movies;">

<td>{{movie.title}}</td>

<td>{{movie.director}}</td>

<td>{{movie.cast}}</td>

<td>{{movie.releaseDate}}</td>

</tr>

</tbody>

</table>

</div>

</div>

</div>

Here is the output

ngFor Example Top 5 Movies in Table format

Nested Array

The following example shows how to use the ngFor in a nested array. The employees array has nested skills array.

employees = [

{

name: "Rahul", email: "rahul@gmail.com",

skills: [{ skill: 'Angular', exp: '2' },{ skill: 'Javascript', exp: '7' },{ skill: 'TypeScript', exp: '3' }

]

},

{

name: "Sachin", email: "sachin@gmail.com",

skills: [{ skill: 'Angular', exp: '1' },{ skill: 'Android', exp: '3' },{ skill: 'React', exp: '2' }

]

},

{

name: "Laxmna", email: "laxman@gmail.com",

skills: [{ skill: 'HTML', exp: '2' },{ skill: 'CSS', exp: '2' },{ skill: 'Javascript', exp: '1' }

]

}

]

Inside the main loop, use the local variable employee to get the list of skills and loop through it using \*ngFor="let skill of employee.skills;"

<div class='card'>

<div class='card-header'>

<p>Nested Array</p>

</div>

<div class='table-responsive'>

<table class='table table-bordered table-sm '>

<thead class="thead-dark">

<tr>

<th>Name</th>

<th>Mail ID</th>

<th>Skills</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let employee of employees;">

<td>{{employee.name}}</td>

<td>{{employee.email}}</td>

<td>

<table class='table table-sm '>

<tbody>

<tr \*ngFor="let skill of employee.skills;">

<td>{{skill.skill}}</td>

<td>{{skill.exp}}</td>

</tr>

</tbody>

</table>

</td>

</tr>

</tbody>

</table>

</div>

</div>

**Local Variables**

ngFor exposes several values, which help us to fine-tune display. We assign these values to the local variable and use it in our template

The list of exported values provided by ngFor directive

* index: number: The zero-based index of the current element in the collection.
* count: number: The total no of items in the collection
* first: boolean: True when the item is the first item in the collection.
* last: boolean: Is set to True, when the item is the last item in the collection.
* even: boolean: True when the item has an even index in the collection.
* odd: boolean: is set to True when the item has an odd index in the collection.

Formatting odd & even rows

We can use the odd & even values to format the odd & even rows alternatively. To do that create two local variables o & e. Assign the values of odd & even values to these variables using the let statement. Then use the ngClass to change the class name to either odd or even. The example code is shown below.

<tr \*ngFor="let movie of movies; let i=index; let o= odd; let e=even;"

[ngClass]="{ odd: o, even: e }">

<td> {{i}} </td>

<td>{{movie.title}}</td>

<td>{{movie.director}}</td>

<td>{{movie.cast}}</td>

<td>{{movie.releaseDate}}</td>

</tr>

Add the appropriate background color to the odd and even classes as shown below in app.component.css

.even { background-color: goldenrod; }

.odd { background-color: floralwhite; }

## First and the Last element of a list

Similarly, you can use the first & last values to style the first & last element. The code below will add a CSS classes first & last to the first and last movie using the [ngClass](https://www.tektutorialshub.com/angular/angular-ngclass-directive/).

 <tr \*ngFor="let movie of movies; let i=index; let first= first; let last=last;" [ngClass]="{ first: first, last: last }">

        <td> {{i}} </td>

        <td>{{movie.title}}</td>

        <td>{{movie.director}}</td>

        <td>{{movie.cast}}</td>

        <td>{{movie.releaseDate}}</td>

      </tr>

    </tbody>

.first { background-color: greenyellow; }

.last { background-color: orange }