Templates, Styles & Directives

# **Template, Styles, View Encapsulation**

# **Built-in Directives**

# **Creating Attribute Directive**

# **Using Renderer to build attribute directive**

# **Host Listener to listen to Host Events**

# **Using Host Binding to bind to Host Properties**

# Template, Styles, View Encapsulation

A template is a HTML view where you can display data by binding controls to properties of an Angular component. You can store your component's template in one of two places.

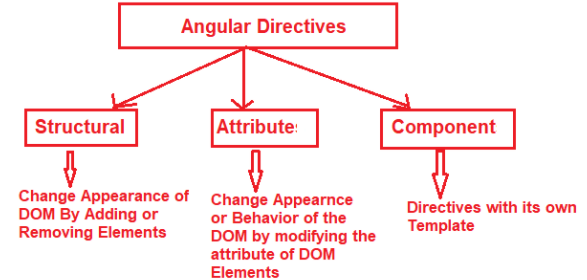
|  |  |
| --- | --- |
| Inline Template | External Template |
| @Component ({  selector: 'my-app',  template: '  <div>  <h1>{{title}}</h1>  <div>Learn Angular</div>  </div>  '  }) | @Component ({  selector: 'my-app',  templateUrl: 'app/app.component.html'  }) |

Use Case : Display all employee id and name as hyperlink in one panel and details in another column.

# Built-in Directives

* Directives are instructions in the DOM.
* The Angular Directives are the elements which are basically used to change the behavior or appearance or layout of the DOM (Document Object Model) element

### Types of Directives?



Angular offers two kinds of built directives:

* Attribute directives
* Structural directives.

|  |  |
| --- | --- |
| Attribute | Structural |
| Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components. | Structural directives are responsible for HTML layout |
| * [NgClass](https://angular.io/guide/built-in-directives#ngClass) - adds and removes a set of CSS classes. * [NgStyle](https://angular.io/guide/built-in-directives#ngStyle)- adds and removes a set of HTML styles. * [NgModel](https://angular.io/guide/built-in-directives#ngModel)- adds two-way data binding to an HTML form element. | * NgFor (\*ngFor) - Iteration * NgIf (\*ngIf)- Condition * NgSwitch (\*ngSwitch) - Condition |
| **When do you use?** |  |
| Attribute directive anytime you want logic or events to change the appearance or behavior of the view. Dropdowns, accordions, and tabs are just a few common use cases for custom attribute directives. When you have a UI element that will be common throughout your app, you can implement an attribute directive and share it across components and modules to avoid repeating the code for the same functionality |  |

ngClass

UseCase: Mark salary as RED if <=10000 else mark it as GREEN

The NgClass directive allows you to set the CSS class dynamically for a DOM element. It is also possible in Angular to add CSS Classes conditionally to an element, which will create the dynamically styled elements and this is possible because of Angular Class Binding.

<https://codecraft.tv/courses/angular/built-in-directives/ngstyle-and-ngclass/>

<p [ngClass]="{

'text-success':employee.salary >10000,

'text-danger':employee.salary <= 10000

}" > {{employee.salary}}

</p>

<https://dotnettutorials.net/lesson/angular-ngclass-directive/>

**Ways to apply ngClass:**

1. ngClass with string
2. ngClass with array
3. The ngClass with object
4. ngClass with component method

# Creating Attribute Directive

Use Case: Apply style using String, Array, Object, Method based on given below css.

**Prerequisite:**

**app.component.css**

**.one{** color: red; **}**

**.two{** font-size: 30px; **}**

**.three{** font-weight: bold; **}**

**.four{** font-style: italic; **}**

**.five{** color: green; **}**

|  |  |
| --- | --- |
| ngClass as String | |
| <tr [ngClass]="'one five'"> |  |
|  |
| ngClass with Array | |
| <td [ngClass]="['three', 'four', 'five']">{{ dept.did }}</td> |  |
| ngClass with Object | |
| <td [ngClass]="{'one':true,'two':false, 'four':true}">{{dept.dname }}</td> |  |
| ngClass with component method | |
| <td [ngClass]="AddCSSClasses('type1')">{{ dept.location }}</td> |  |
| Dept.component.ts  AddCSSClasses(flag:string) {  let Cssclasses;  if(flag == "type1")  { Cssclasses = {  'one' : true,  'two' : true  } } else {  Cssclasses = {  'four' : true,  'five' : true  } } return Cssclasses;  } |

ngStyle

The ngStyle directive is used to **set the DOM element** style properties.

Use Case :create a button with font size 20, the color red and font-weight bold.

|  |  |
| --- | --- |
| ngStyle as String | |
| <a href="logout">  <button type="button"  [ngStyle]="{'color':'red',  'font-weight': 'bold',  'font-size.px':10}">  Log Out</button></a> |  |
| ngStyle as dynamic values | |
| <p [ngStyle] ="{'background-color':employee.isActive === false ? 'red' : 'green'}" class="card-text">{{employee.isActive}}</p> |  |

# Create Custom Attribute directive

|  |  |  |
| --- | --- | --- |
|  | ng g d BasicBackgroungDirective | import { Directive } from '@angular/core';  @Directive({  selector: '[appBasicBackgroungDirective]'  })  export class BasicBackgroungDirectiveDirective {  constructor() { }} |
| 2. | Update Directive | import { Directive } from '@angular/core';  @Directive({  selector: '[appBasicBackgroungDirective]'  })  export class BasicBackgroungDirectiveDirective implements OnInit{  constructor(private elementRef: ElementRef) { }  ngOnInt()  { this.elementRef.nativeElement.style.backgroundColor='blue';}} |
| 3 | Test attribute in html<h1 style="text-align:center" appBasicBackgroungDirective> {{title}} </h1> |  |

# Using Renderer to build attribute directive

Sometime template renders without DOM, in that case the above custom attribute directive eg: appBasicBackgroungDirective will not be available.

The Renderer2 class is an abstraction provided by Angular in the form of a service that allows to manipulate elements of your app without having to touch the DOM directly. This is the recommended approach because it then makes it easier to develop apps that can be rendered in environments that don’t have DOM access, like on the server, in a web worker or on native mobile.

|  |  |  |
| --- | --- | --- |
| 1 | ng g d backgroundUsingRender | mport { Directive,ElementRef,Renderer2, OnInit } from '@angular/core';  @Directive({  selector: '[appBackgroundUsingRender]'  })  export class BackgroundUsingRenderDirective implements OnInit{  constructor() { }  ngOnInit()  { }  } |
| 2 | Update backgroundUsingRender.ts | import { Directive,ElementRef,Renderer2, OnInit } from '@angular/core';  @Directive({  selector: '[appBackgroundUsingRender]'  })  export class BackgroundUsingRenderDirective implements OnInit{  constructor(private elementRef: ElementRef, private renderer :Renderer2) {}  ngOnInit()  { this.renderer.setStyle(this.elementRef.nativeElement,'background-color','yellow'); }} |
| 4 | App.component.html  <h4 style="text-align:center" appBackgroundUsingRender > It works </h4> |  |

<https://www.digitalocean.com/community/tutorials/angular-using-renderer2>

# Host Listener to listen to Host Events

@HostBinding and @HostListener are two decorators in Angular that can be really useful in custom directives.

# Using Host Binding to bind to Host Properties

|  |  |
| --- | --- |
| **@HostListener** | @HostBinding |
| will listen to the event emitted by the host element that's declared with @HostListener. | Declares a host property binding. Angular automatically checks host property bindings during change detection. If a binding changes, it will update the host element of the directive.  If a binding changes, HostBinding will update the host element. |
|  |  |

|  |  |  |
| --- | --- | --- |
| 1 | Create Directive- | ng g d rainbowDirective |
| 2 | Update the rainbow directive | import {  Directive,  HostBinding,  HostListener } from '@angular/core';  @Directive({  selector: '[appRainbowDirective]'  })  export class RainbowDirectiveDirective {  possibleColors = [  'darksalmon', 'hotpink', 'lightskyblue', 'goldenrod', 'peachpuff', 'mediumspringgreen', 'cornflowerblue', 'blanchedalmond', 'lightslategrey'  ];  @HostBinding('style.color') color: string;  @HostBinding('style.border-color') borderColor: string;  @HostListener('mouseenter') newColor() {  const colorPick = Math.floor(Math.random() \*  this.possibleColors.length);  this.color = this.borderColor = this.possibleColors[colorPick  ];  }  } |
| 3 | Map the directive with tag | <h1 appRainbowDirective >Departments</h1> |
| 4 | Test by Entering mouse on Departments |  |

# Building Structural Directives

\*ngIf

The ngIf is a structural directive and it is used to add or removes the HTML element and its descendant elements from the DOM layout at runtime conditionally. That means it conditionally shows the inline template.

|  |  |
| --- | --- |
| ngIf with Then and Else block | |
|  | |
|  | |
|  |  |
| NgIf directive with else block | |
|  | **<div>**  **<input** type="radio" name="rb" (click)= "ChangeData(true)" checked **>** Valid  **<input** type="radio" name="rb" (click)= "ChangeData(false)"**>** Invalid  **</div>**  **<div** \*ngIf="isValid else elseblock"**>**  **<b>**The Data is valid.**</b>**  **</div>**  **<ng-template** #elseblock**>**  **<div** **>**  **<b>**The Data is invalid.**</b>**  **</div>**  **</ng-template>** |
| **<div>**  **<input** type="radio" name="rb" (click)= "ChangeData(true)" checked **>** Valid  **<input** type="radio" name="rb" (click)= "ChangeData(false)"**>** Invalid  **</div>**  **<div** \*ngIf="isValid then thenblock else elseblock"**>** **</div>**  **<ng-template** #thenblock**>**  **<div> <b>**The is Then Block**</b> </div>**  **</ng-template>**  **<ng-template** #elseblock**>**  **<div** **><b>**The is Else Block**</b></div>**  **</ng-template>** | |
|  | |

\*ngSwitch

The Angular ngSwitch directive is actually a combination of two directives i.e. an attribute directive and a structural directive. It is very similar to the switch statement of other programming languages like Java and C# but within a template.

The ngSwitch directive lets you hide or show the HTML elements based on an expression. Here, you can also define a default section using the ng-switch-default directive to show a section if no other sections get a match. So, while working with ngSwitch directive, you need three things to keep in mind, they are ngSwitch, ngSwitchCase and ngSwithDefault.

<https://dotnettutorials.net/lesson/angular-ngswitch-directive/>

Use Case: Apply Switch on Country

**Step 1: App.component.html**

**<h2>**Select Country**</h2>**

**<select** (change)="SetDropDownValue($event)"**>**

**<option** value=""**>**Select**</option>**

**<option** value="In"**>**In**</option>**

**<option** value="US"**>**US**</option>**

**<option** value="UK"**>**UK**</option>**

**</select>**

**<h2>**You Have Selected**</h2>**

**<div** [ngSwitch] = 'dropDownValue'**>**

**<h3** \*ngSwitchCase="'In'"**>**India**</h3>**

**<h3** \*ngSwitchCase="'US'"**>**United State**</h3>**

**<h3** \*ngSwitchCase="'UK'"**>**United Kingdom**</h3>**

**<h3** \*ngSwitchDefault=""**>**You have not selected any country**</h3>**

**</div>**

Step 2: set selected value in a variable

**export** **class** AppComponent **{**

**public** dropDownValue = "";

SetDropDownValue**(**drpValue : **any)** **{**

this.dropDownValue = drpValue.target.value;

**}}**