Directives:

It gives instructions to the dom

e.g:

component-> selector gives information to dom

1)structural directives:

* Look like a normal HTML attribute but have a leading \*(for desugaring)
* Affect a whole area in the DOM(Elements get added/remove)
* We don’t have more one than one structural directive on same element

e.g: if we use ngif on the paragraph if that condition is false the paragraph remove from the dom so overall view container is affected.

2)attribute directives:

* It looks like a normal HTML attribute(possibly with databind or event vbinding)
* Only effect/change the element they are added to

Don’t add or remove elments.They only change the element they were placed on

e.g:in this we never destroy an element from the DOM YOU CONLY CHANGE THE PROPERTY OF THE ELEMENT FOR E.G:BACKGROUND Color:

[ngClass]=”{odd:odd%2!==0)”

[ngStyle]=”{backgroundColor:odd%2!==0?’yellow’:’transparent’}”

\*ngFor=”let odd of oddNumber”

LifeCycle of Component:

1)ngOnChanges:

it’s executed right at the start when a new component is created but thereafter,it’s also alwasys called whenever one of our bound input properties changes. And with that ,I mean properties decorated with @input, so whenever these properties received new values.

2)ngOnInit:

It is executed once the component is initialized.ngOninit will run after the constructor.

3)ngDoCheck:

It also runs multiple times because this will run whenever change detection runs.now change detection simply is the system by which angular determines whether something changed on the template of a component or inside of a component.whether some property value changed from 1 to 2 and that property is output in the template well of course angular needs to re render that part of the template and ngdocheck is a hook executed on every check angular makes.Now important,on every check so not just if something changed,a lot of times ngDoCheck will run because you clicked some button which doesn’t change anything.

4)ngAfterContentInit:

Called after content(ng-content)has been projected into view

5)ngAfterContentChecked:

Called everytime the projected content has been checked

6)ngAfterViewInit:

Called after the component’s view(and child views) has been initialized once our view has been you could say.

7)ngAfterViewChecked:

Called everytime the view(And child views) have been checked

8)ngOnDestroy:

Called once the component is about to be destroyed

@ViewChild:

It is use for gaining value from same component html template with # local reference.

@ContentChild:

It is use for gaining value from other component html template with # local reference.

ngContent:

it is use for projection of template of outside of the component.

@output:

Event listen to the outside of the component

Means example:

@output variable in game component and it listens on appcomponent using []=fired($event)

@input variable is use when you want to pass value from outside

Custom Directive:

Import{Directive,OnInit,ElementRef,Renderer2} from ‘angular/core’;

@Directive({

Selector:’[appBetterHighlight]’

}}

Export class BetterHighlightDirective implements Oninit{

Constructor(private elRef:ElementRef,private renderer:Renderer2){}

ngOnInit{

this.renderer.setStyle(this.elRef.nativeElemen,’background-color’,’blue’);

}

}

Why renderer is better approach to access dom?

Angular is not limited to running in the browser here it for example:also work with service workers and these are environments where you might not have access to the DOM so if you try to change the DOM as you did here in basic highlight by directly accessing the native and the style of this element,you might get an error in circumstances

In the last lecture,we wused the angular renderer class to change style of a Html element.As explained in that lecture you should use the Renderer for any DOM manipulations.

Ofcourse,you can do more than simply change the styling of an element via setStyle().

Another Approach rather than renderer is @HostBinding:

@HostBiniding(‘style.backgroundColor’) backgroundColor:string=’transparent’;

@HostListener is use for making dynamic behavior.

@HostListener(‘mousenter’)mouseover(eventData:Event)

{

//this.renderer.setStyle(this.elRef.nativeElement,’background-color’,’blue’,false,false);

This.backgroundColor=’blue’

}

@HostListener(‘mouseleave’)mouseleave(eventData:Event)

{

This.backgroundColor=transparent

}

Custom property Binding:

For setting value from outside.

@Input() defaultColor:string=’transparent’;

@Input() highlightColor:string=blue;

@HostBiniding(‘style.backgroundColor’) backgroundColor:string;

ngOnInit()

{

This. backgroundColor= this.defaultColor;

}

@HostListener(‘mousenter’)mouseover(eventData:Event)

{

//this.renderer.setStyle(this.elRef.nativeElement,’background-color’,’blue’,false,false);

This.backgroundColor=this.highlightColor;

}

@HostListener(‘mouseleave’)mouseleave(eventData:Event)

{

This.backgroundColor=this.defaultColor;

}

Html:

<p appBetterHighlight [defaultColor]=”’yellow’” [highlightColor]=”’red’”>Style me with a better directive </p>

Custom Structural Directive:

e.g:opposite of ngif directive:

@Input() set appUnless(condition:boolean)

{

If(!condition)

{

//createEmbbedView creates a view in this view container

//tempalteRef -> template we need which we reference

This.vcRef.createEmbeddedView(this.templateRef);

}

Else

{

This.vcRef.clear();

}

}

//viewcontainer ref: marks the place where we placed this directive in the document

constructor(private tempalteRef:TempalteRef<any>,private vcRef:ViewContainerRef){

}

Use in Html:

<div \*appUnless=”onlyOdd”>

</div>

ngSwitch:

app.component.ts:

value=10;

<div [ngSwitch]=”value”>

<p \*ngSwitchCase=”5”>Value is 5 </p>

<p \*ngSwitchCase=”10”>Value is 5 </p>

<p \*ngSwitchCase=”100”>Value is 5 </p>

<p \*ngSwitchDefault>Value is Default </p>

What are services?

Service acts a a centeral repository,as a central business unit you could say,something where you can store,where you can centralize your code in.

What are dependency injector?

Well a dependency is something a class of ours will depend on

For e.g:

The new account component depends on the loggingService because we want to call a method in that service and the dependency injector simply injects this dependency,injects an instance of this class inoto our component automatically.

//providers show how to create this loggingservice

@Component({

Providers:[LogginService]

)}

constructor(private loggingService:LogginService){}

Herirecal Injector are on three ways:

1)app.module.ts:shared one instance whole app

2)app.component.ts:shared one instance parent to child.

3)account.component.ts:shared only in one single component.

Inject services into services:

We should use @injectable in where we receive some other service into our service

e.g:

logging service injects into account service

note:

we don’t need to use @injectable where we don’t need to receive other service.

Using Services for Cross-Component Communication:

If you want to listen event from outside of the compoenent then use it

Account.service.ts:

statusUpdated=new EventEmitter<string>();

account.component.ts:

onSetTo(status:string){

this.accountsService.statusupdated.emit(status);

}

//I want to listen now

New-acount.component.ts:

Constructor(private accountService:AccountsService){

//subscribe because event emiiter returns observable

This.accountsService.statusUpdated.subscribe(

(status:string)=>alert(‘New Status:’+status)

);

}

}