**changeDetection**

Angular.js uses digest cycle for two-way binding. But angular2 components are tree structured, each component has own Change Detection(CD). If anywhere CD happens, angular entire tree structured CD will be trigger. But we can control the CD calling by using Immutable& Observable input and changeDetectionstatergy.onPush.

**changeDetectionstatergy.onPush**

it will enable to prevent CD, if the value types are changed. it will call only on entire object reference changed.

|  |
| --- |
| Trigger CD  @inputVariable = 1232;  @inputVariable ={ name : 'newName'}  CD will not trigger  @inputVariable.name ='newName' |

LIVE DEMO : <https://stackblitz.com/edit/angular-6b4rwm>

**ChangeDetectorRef**

We can get this reference from constructor. this will provide following methods to handle manually.

|  |
| --- |
| markForCheck() - explicitly marks the view as changed.  detach() - Entirely view has been detaching from model. so any value/reference will not trigger CD.  detectChanges() - it will trigger CD for entire component.  checkNoChanges() – it will make sure no changes are there.  reattach() - opposite to detach. |

**Template syntax**

Angular2 offers template syntax to bind from view to source, source to view and two way binding as usual. But some notable points are mentioned here.

innerHTML - will not allow to use <script> tag...

|  |  |
| --- | --- |
| data Source to view | {{val}  [target]="val"  bind-target="val" |
| view to data Source - | (target)="statement"  on-target="val" |
| two-way binding - | [(target)]="val"  bindon-target="val" |

**Structural Directive**

**\*ngFor**

we can use track by for better performance. By default, angular update only changed DOM object. but if you reassign entire object, angular recreate entire dom.

Live demo: <https://stackblitz.com/edit/angular-6b4rwm-y1wdr1>

|  |
| --- |
| collection[0].name = "new"; it will update only 0th object by default  collection.push( {id:345,name:'test'} ); it will update only 0 th object by default  collection = [{id:234,name:'BAT'}..]; by default entire DOM will recreate. use trackBy only update changed dom.  Syntax:  <div\*ngFor="let item of collection;trackBy:trackByFnPtr"> |

**Template Reference**

we can use to get reference from DOM / directive / component using #refvar/ref-var.

|  |
| --- |
| Two ways to define:  Syntax:  <div ref-btn (click)="my(btn)">  <div #btn (click)="my(btn)"> |

**Input & output**

Following methods to define input & output

|  |  |
| --- | --- |
| @Input () name: any;  @Output () event: new EventEmiter<any> (); | @Component ({  inputs: ['name'],  outputs: ['event'],  }) |
| @Input("name") alias: any;  @Output("event") alias: new EventEmiter<any> (); | @Component ({  inputs: ['alias: name'],  outputs: ['alias: event'],  }) |

**Points to remember**

|  |
| --- |
| The safe navigation operator ?. - {{ obj ?. prop }} - obj is not there it will run throw any error  The non-null assertion operator ! - {{ obj !. prop}} - obj is not there it will throw error compile time itself.  The $any type cast function ($any( <expression> )) |

**Life cycle hooks**

**NgOnChanges**

1. only call if component have @input,
2. ngOnchanges only having parameter to check whether input value changed or not.
3. ngOnChanges should not call if input property not changed (object property are changed)

**Calling order**

|  |
| --- |
| constructor  child-constructor  ngOnChanges -  ngOninit  ngDoCheck  ngAfterContentInit  ngAfterContentChecked  child-onChanges  child-onInit  child-doCheck  child-afterContentInit  Child-AfterContentChecked  if any child ...  child-afterViewInit  child-AfterContentChecked  child-ondestroy  ngAfterViewInit  ngAfterViewChecked  ngOnDestroy |

Live demo : <https://stackblitz.com/edit/angular-6b4rwm-dvgoae>

**Styles**

Ways to apply CSS in component

1. Styles - meta data to component
2. StylesUrl - meta data to component
3. <style> tag inside the component.
4. <link> tag inside the component.
5. @import './stye.css' - using typescript.

Note: but all are encapsulated.

**pseudo-class selector**

Component styles have a few special selectors from the world of shadow DOM style scoping.

1. : host - it used to select the host element. (<app></app>).\
2. : host-context(cont.) - it used to apply style based on condition. if ancestor have class, then only the style will apply.
3. /deep/ is used set style to all (not encapsulate). But its deprecated.

|  |
| --- |
| : host-context(.isClass) h1{  //styles  }  : host /deep/ h3 {  //styles  } |

**View Encapsulation**

Angular offers view encapsultion.Dom logic and styles are encapsulating under single element. We can apply component style using styles, styleUrls property in component decorator.it will applicable only for that component. (Not for children component). like if you give type="date", html automatically render date picker.

Actually angular not use shadow DOM, but we can enable manually. (It will not work for all browsers)

Three types

1. ViewEnapsulation.none - it will disable encapsulation, styles will common for all component.
2. ViewEncapsulation.Native - It will use native shadow dom. but most of the browsers will not support.
3. ViewEncapsulation.Emulated - default. styles will encapsulate without native shadow dom. By CSS alone.

**Custom Elements**

Angular uses custom element to create component. Currently crome support other opera,safri supports throw polyfills. Angular createCustomElement() Api provide the bridge to create custom element dynamically.

Use the createCustomElement() function to convert a component into a class that can be registered with the browser as a custom element. After you register your configured class with the browser's custom-element registry, you can use the new element just like a built-in HTML element in content that you add directly into the DOM:

|  |
| --- |
| const PopupElement = createCustomElement(PopupComponent, {injector});  customElements.define('popup-element', PopupElement); |

**Dynamic Component Loader**

Angular provides componentFactoryResolver API to load dynamic components. We can load component based on condition.

|  |
| --- |
| @Component({  selector: 'app-ad-banner',  template: `  <div class="ad-banner">  <h3>Advertisements</h3>  <div ref-dynamic></div>  </div> `  })  export class AdBannerComponent implements OnInit, OnDestroy {  @ViewChild("dynamic",{ read: ViewContainerRef }) dynamic:ViewContainerRef;  constructor(private componentFactoryResolver: ComponentFactoryResolver) { }    ngOnInit() {  this.loadComponent();  }  loadComponent() {  let componentFactory = this.componentFactoryResolver.resolveComponentFactory(ComponentName);  this.dynamic.clear();  let componentRef =this.dynamic.createComponent(componentFactory);  (<ComponentName>componentRef.instance).data = { name:'test'}; //@input("data")  }  } |

**Directives**

Angular provides 3 types directives. Selector may be attribute,event,element.

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

Link: <https://stackblitz.com/edit/angular-6b4rwm-yybqjj>

**Component Directive**

Component also known as directive, Selector also may be attribute,event,element. But only different is @Component only have template. Otherwise we can do whatever we doing using @Directive.

**Attribute Directive**

Attribute Directive used change the appearance of the element. We can change the element style’s. it will not remove / add the element. You can apply one or more directive to the same element.

|  |
| --- |
| //it will applicable for both attribute and component directive  export class commonCls {  constructor(private el:ElementRef,private ren:Renderer){  this.el.nativeElement.style.color='black'; // type 1  this.ren.setElementStyle(this.el.nativeElement,'border','1px solid') // type 2  }  // type 3  @HostBinding('style.padding') pad : any;  @HostBinding('style.background') bgColor :any;  @HostBinding('attr.id') id:any;  @HostListener('mouseover') myfun(){  this.pad = '10px 10px';  this.bgColor = 'yellow';  this.id='compID';  }  @HostListener('mouseleave') reset(){  this.pad ='0px 0px';  this.bgColor = '';  this.id='';  }  } |

**Structural Directive**

Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements. Normally \* prefix denotes the structural directive.

For Structural directive we should inject TemplateRef and ViewContainerRef.More than one structural directives are not allowed. For this we can use ng-container.Ref example.

Angular all structural direcitives (\*ngIf,\*ngFor,\*ngSwitchCase,\*ngSwitchDefault) are internally use the ng-template. Ng-template by default will not display anything. Here structural directive will convert into attribute directive. Ref example.

|  |
| --- |
| @Component({  selector: 'app-root',  template: `  <h5>Structual Directive</h5>  <div \*heroLoop='let hero comefrom heros;let index=rank'>  {{hero}} is a {{index}} st hero  </div>    <h5>By Default ng-template will not be show</h5>  <ng-template ngIf='true'>  This is from ng-template  <h6>But here am using ngIf without \*</h6>  </ng-template>    <!-- ng-container also we can use for multiple structural directe single elemtn -->  <div>ng-container</div>  <ng-container \*ngFor="let item of heros">  <ng-container \*ngIf="isTrue">  <div>  {{item}}  </div>  </ng-container>  </ng-container>  `  })  //Structural Directive  @Directive({  selector:'[heroLoop]' //[attr] | (attr) | attr  })  export class structCls{  constructor(private tRef:TemplateRef<any>,private viewRef:ViewContainerRef ){  }  @Input() set heroLoopComefrom(heros: any) {  //NgOnChnages also we can put it  this.viewRef.clear();  for(let i=0;i<heros.length;i++){  let context = {  $implicit:heros[i], //first let variable is $implicit  rank : i + 1  }  this.viewRef.createEmbeddedView(this.tRef,context)  }  } } |

**Pipes**

Pipes used to transform actual data to user readable data. Angular provides some built in pipes(lowercase,uppercase,date,json,async).we can use multiple pipes in same expression using | sympol also arguments. Two types of pipes are there.

**Pure pipe**

it will trigger only on primitive data types are changed or object refrence changed.(based on pipe arguments, if args are changed)

**Impure pipe**

It will trigger on every angular change detection. For this we need to pipe:true in @Pipe decorator.

[Live demo](https://stackblitz.com/edit/angular-6b4rwm-eb9vs1)

|  |
| --- |
| @Component({  selector: 'app-root',  template: `  <h3>Pure Pipes</h3>  <ul>  <li \*ngFor="let item of (heros | filter:'name':searchTxt)">  {{item.name}}  </li>  </ul>  <h3>Impure Pipes</h3>  <ul>  <li \*ngFor="let item of (heros | filterImpure:'name':searchTxt)">  {{item.name}}  </li>  </ul>  `  })  //Pure pipe  @Pipe({  name:'filter'  })  export class filterPipe implements PipeTransform{  transform(arr:any,field:any,value:any){  if(value){  console.log(arr);  return arr.filter(o=>o[field].indexOf(value.toUpperCase()) != -1);  }else{  return arr;  }  }  }  //Impure Pipe  @Pipe({  name:'filterImpure',  pure:false  })  export class filterImpurePipe extends filterPipe{  } |

**Animation**

Angular provides some animation features. Angular animation will run top of the web animation api.

Implement animation steps

1. Import BrowserAnimationsModule from ‘@angular/platform-browser/animations’ and import to ngModule.
2. Import trigger,state,style,animation,transition to component.
3. Write animations inside the animation array from @component decoration.
4. Every element animation need to write. Inside the trigger we should define the state.
5. Every state we can assign style, also changing the state we can apply styles using transition.

|  |
| --- |
| import { trigger,state, style, animate, transition} from '@angular/animations';  @Component({  selector: 'app-root',  template: `  <h1>Animation</h1>  <div>  <button [@btnAnimate]="state" (click)="toggleState()"  (@btnAnimate.start)="animation = 'Animating'"  (@btnAnimate.done)="animation = 'Animation Done'" >  Btn1  </button>  {{animation}}  </div>  `,  animations: [  trigger('btnAnimate', [  state('inactive', style({  backgroundColor: 'blue',  transform: 'scale(1)'  })),  state('active', style({  backgroundColor:'red',  transform: 'scale(1)'  })),  // transition('\* => \*',animate('1000ms')),//Any to any state  //Enter the animation (first time)  transition('void => \*',animate(1000,style({  height:100,  width:100,  }))),  transition('inactive => active', animate('1000ms ease-in',style({  height:100,  width:100,  }))),  transition('active => inactive', animate('100ms ease-out'))  ])  ]  })  export class AppComponent {  animation = '';  state = 'inactive';  toggleState(){  this.state = this.state == 'active'? 'inactive' : 'active';  }  } |

**Angular Forms**

Forms used to get input input from users. it will intract with users.Angular provides two forms.

**Template Driven forms**

By default template driven directives(ngModel,ngSubmit,etx..) will not inculed in angualr/core. For template driven forms we should Include FormsModule in NgModule.

|  |
| --- |
| <form (ngSubmit)="onsubmit()" #studForm="ngForm">  <div>  <label>Name</label>    <input type="text"  ref-studName="ngModel"  [(ngModel)]='stud.name'  required name="studName"  [style.border]="!studName.valid && !studName.pristine ? '1px solid red' : '' " />  <label [hidden]="studName.valid || studName.pristine">\*name req</label>  </div>    <div>  <input type="submit" [disabled]="!studForm.form.valid" value="submit" />  </div>  </form> |

**Form Validation**

We can validate ngmodel, whether is touched or unteched, dirty or pristine , valid or invalid. We can all html attribute to validate user input , like required,minlenght,maxlength..

Custom validator also we can add.