Angular1 – scope, watchers, digest cycle, factory, services, custom directives, isolated scope

Ng2 – providers, declarations, bootstap component, input, output, difference,

<https://medium.com/wizardnet972/48-answers-on-stack-overflow-to-the-most-popular-angular-questions-52f9eb430ab0>

Ngrx

Rxjs

<http://reactivex.io/rxjs/manual/overview.html#introduction>

<http://reactivex.io/rxjs/class/es6/MiscJSDoc.js~ObserverDoc.html>

<https://github.com/ReactiveX/rxjs/blob/master/doc/introduction.md>

<http://reactivex.io/documentation/operators.html>

<http://rxmarbles.com/>

<https://codecraft.tv/courses/angular/reactive-programming-with-rxjs/streams-and-reactive-programming/>

<https://blog.angularindepth.com/learn-to-combine-rxjs-sequences-with-super-intuitive-interactive-diagrams-20fce8e6511>

angular http interceptors

implmetation = >

constructor(public http: HttpInterceptor) {};

getData() {

return this

.http.get(`data?limit=20&offset=40`)

// will hit the api on http://api.localhost.com:4000/data?limit=20&offset=40

.map(res => res.json().data)

}

Suppose we are implementing a basic authentication on our backend server. And we want every http request to contain a secret set under Authorisation key. That request should look something like that in below image.

<https://medium.com/aviabird/http-interceptor-angular2-way-e57dc2842462>

PIPES

@Pipe({name: 'exponentialStrength'})

export class ExponentialStrengthPipe implements PipeTransform {

transform(value: number, exponent: string): number {

let exp = parseFloat(exponent);

return Math.pow(value, isNaN(exp) ? 1 : exp);

}

}

REact vs angular -

React does nt have ngmodule decorator since It does not have modules , it don’t have structured packaged components to import to other modules since it is a javascript modules(just export and import).

React does nt have index.js to export all files so that we can put {component1,2,3} from /comp/index;

Even though if we make it to work , when error comes cant identify which index.js

<https://angular.io/guide/ngmodule-vs-jsmodule>

Router

The order of route configuration matters. The router accepts the first route that matches a navigation request path.

When all routes were in one AppRoutingModule, you put the default and [wildcard](https://angular.io/guide/router) routes last, after the /heroes route, so that the router had a chance to match a URL to the /heroes route *before* hitting the wildcard route and navigating to "Page not found".

angular life hooks

**ngOnInit**. This is a life cycle hook where its ideal to fetch data. AFter dom is ready. Similar to componentDidMount in react.

**Ngchanges** - called before nginit for input bound changes and similar to getDerivedStateFromProps in react. Ngchanges gives prev and next value of all input data properties … so used when we want to change mostly all input properites. Otherwise we can use setter and getter in input decorator.

<https://ngdev.space/angular-2-input-property-changes-detection-3ccbf7e366d2>

<https://stackblitz.com/edit/input-properties-change-detection?embed=1&file=src/main.ts>

**Ngdocheck** – component which don’t update

For grabbing all the content children, we should use @ContentChildren (or @ContentChild if there’s only one child), and the children will be set on **ngAfterContentInit**

**ngaftercontentchecked**

**NgAfterViewInit:** after parent renders, it goes to compute n initialize child views .after that this hook called.

**Ngafterviewchecked**

It's used when viewchild or viewChildren used

**NgOnDestroy -** it gets called when user move to another component

Migration

DI => When service is injected at ngmodule root level, then a new shared single instance is available throughout the module . if injected at component level, then it new shared single instance is available for particular component and its child.

Components => Components Interaction

**Available:** e2e tests, clear timer, setter, ngonchanges, local variable #timer

**Usage:** pass via input binding, service (using subject observable), access via local variable, viewchild

<https://stackblitz.com/edit/angular-eqa9pq?file=src%2Fapp%2Fhello.component.ts>

<https://stackblitz.com/edit/angular-fobggu?file=src%2Fapp%2Fmission-service.ts>

ElementRef

Wrapper around the native element after runtime . Denotes and access the runtime dom

<https://stackblitz.com/edit/angular-eu3odm>

ng content template viewchild 'ren viewcontent viewproviders

<https://blog.mgechev.com/2016/01/23/angular2-viewchildren-contentchildren-difference-viewproviders/>

@Component({ selector: 'todo-app',

template: `<section>Add todo:

<**todo-input** (todo)="addTodo($event)"></todo-input>

</section>

<**todo-item** \*ngFor="let todo of todos.getAll()" [todo]="todo">

<ng-content **select="app-footer"**></ng-content>

`

})

class TodoAppComponent {

constructor(private todos: TodoList) {}

addTodo(todo) { this.todos.add(todo); }

}

In html template:

**<todo-app></todo-app>**

Well, this is basically an XML, so between the opening and closing tags of the todo-app element we can put some content:

<todo-app>

**<app-footer> Yet another todo app! </app-footer>**

</todo-app>

With ng-content we can grab the content between the opening and closing tag of the todo-app element and project it somewhere inside of the template!

Since **select=”app-footer”** is given, <app-footer> will be placed there if it is like footer area. If sidebar , sidebar area.

**todo-input** and **todo-item** could be considered view children of todo-app

**app-footer** (if it is defined as Angular component or directive) could be considered as a content child.

#### viewProviders vs providers

When we declare viewProviders: [someService] then we can inject in parent component constructor and can be used for all view child n children.

But in content child it throws error.

So there we need Providers: [someService] so it will be available for all child of parent component.

=======================

**Directives**

For resusable context, can use **hostbinding** to particular attribute directive like hover on element to toggle color or show / hide where this usecase mostly widely used.. so instead of writing as event handler create a shared directive and apply to all components needed.

@[Directive](https://angular.io/api/core/Directive)({ selector: '[appHighlight]'})

export class HighlightDirective {

constructor(el: [ElementRef](https://angular.io/api/core/ElementRef)) {

el.nativeElement.style.backgroundColor = 'yellow';

}

}

brackets ([]) that make it an attribute selector. Angular locates each element in the template that has an attribute named appHighlight and applies the logic of this directive to that element.

It's like normal class attribute in html, document look for class name in css file and apply the color or respective properties of class.

<https://angular.io/guide/attribute-directives>

**Structural directives** are responsible for HTML layout. They shape or reshape the DOM's *structure*, typically by adding, removing, or manipulating elements. \*ngIf \***ngFor (asterisk**\*)

You can apply many attribute directives to one host element. **You can only apply one structural directive to a host element.**

src/app/app.component.html (asterisk)

<div \*[ngIf](https://angular.io/api/common/NgIf)="hero" class="name">{{hero.name}}</div>

The asterisk is "syntactic sugar" for something a bit more complicated. Internally, Angular translates the \*[ngIf](https://angular.io/api/common/NgIf)attribute into a <ng-template> *element*, wrapped around the host element, like this.

src/app/app.component.html (ngif-template)

<ng-template [[ngIf](https://angular.io/api/common/NgIf)]="hero"> <div class="name">{{hero.name}}</div></ng-template>

Angular consumed the <ng-template> content during its actual rendering and replaced the <ng-template> with a diagnostic comment.

The [NgFor](https://angular.io/guide/structural-directives) and [NgSwitch...](https://angular.io/guide/structural-directives) directives follow the same pattern.

<div \*ngFor="let hero of heroes; let i=index; let odd=odd; trackBy: trackById" [class.odd]="odd"> ({{i}}) {{hero.name}}</div> <ng-template ngFor let-hero [ngForOf]="heroes" let-i="index" let-odd="odd" [ngForTrackBy]="trackById"> <div [class.odd]="odd">({{i}}) {{hero.name}}</div></ng-template>---------------------------------<div [[ngSwitch](https://angular.io/api/common/NgSwitch)]="hero?.emotion"> <app-happy-hero \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'happy'" [hero]="hero"></app-happy-hero> <app-sad-hero \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'sad'" [hero]="hero"></app-sad-hero> <app-confused-hero \*[ngSwitchCase](https://angular.io/api/common/NgSwitchCase)="'app-confused'" [hero]="hero"></app-confused-hero> <app-unknown-hero \*[ngSwitchDefault](https://angular.io/api/common/NgSwitchDefault) [hero]="hero"></app-unknown-hero></div>

The switch value assigned to [NgSwitch](https://angular.io/api/common/NgSwitch) (hero.emotion) determines which (if any) of the switch cases are displayed.

[NgSwitch](https://angular.io/api/common/NgSwitch) itself is not a structural directive. It's an *attribute* directive that controls the behavior of the other two switch directives. That's why you write [[ngSwitch](https://angular.io/api/common/NgSwitch)], never \*[ngSwitch](https://angular.io/api/common/NgSwitch).

<https://docs.angularjs.org/guide/directive> // ng1

Create custom directive

### **What is the Angular equivalent to an AngularJS $watch?**

In AngularJS you were able to specify watchers to observe changes in scope variables using the $watch function of the $scope. What is the equivalent of watching for variable changes (in, for example, component variables) in Angular?

In Angular 2, change detection is automatic… $scope.$watch() and $scope.$digest() R.I.P.

Unfortunately, the Change Detection section of the dev guide is not written yet (there is a placeholder near the bottom of the [Architecture Overview](https://angular.io/docs/ts/latest/guide/architecture.html) page, in section “The Other Stuff”).

Here’s my understanding of how change detection works:

* Zone.js “monkey patches the world” — it intercepts all of the asynchronous APIs in the browser (when Angular runs). This is why we can use setTimeout() inside our components rather than something like $timeout... because setTimeout() is monkey patched.
* Angular builds and maintains a tree of “change detectors”. There is one such change detector (class) per component/directive. (You can get access to this object by injecting [ChangeDetectorRef](https://angular.io/docs/ts/latest/api/core/index/ChangeDetectorRef-class.html).) These change detectors are created when Angular creates components. They keep track of the state of all of your bindings, for dirty checking. These are, in a sense, similar to the automatic $watches() that Angular 1 would set up for {{}} template bindings.  
  Unlike Angular 1, the change detection graph is a directed tree and cannot have cycles (this makes Angular 2 much more performant, as we'll see below).
* When an event fires (inside the Angular zone), the code we wrote (the event handler callback) runs. It can update whatever data it wants to  the shared application model/state and/or the component’s view state.
* After that, because of the hooks Zone.js added, it then runs Angular’s change detection algorithm. By default (i.e., if you are not using the onPush change detection strategy on any of your components), every component in the tree is examined once (TTL=1)... from the top, in depth-first order. (Well, if you're in dev mode, change detection runs twice (TTL=2). See [ApplicationRef.tick()](https://angular.io/docs/ts/latest/api/core/index/ApplicationRef-class.html#!#tick-anchor) for more about this.) It performs dirty checking on all of your bindings, using those change detector objects.
* Lifecycle hooks are called as part of change detection.   
  If the component data you want to watch is a primitive input property (String, boolean, number), you can implement ngOnChanges() to be notified of changes.   
  If the input property is a reference type (object, array, etc.), but the reference didn't change (e.g., you added an item to an existing array), you'll need to implement ngDoCheck() (see [this SO answer](https://stackoverflow.com/a/34298708/215945) for more on this).   
  You should only change the component's properties and/or properties of descendant components (because of the single tree walk implementation -- i.e., unidirectional data flow). Here's [a plunker](http://plnkr.co/edit/XWBSvE0NoQlRuOsXuOm0?p=preview) that violates that. Stateful pipes can also [trip you up](https://stackoverflow.com/questions/34456430/ngfor-doesnt-update-data-with-pipe-in-angular2/34497504#34497504) here.
* For any binding changes that are found, the Components are updated, and then the DOM is updated. Change detection is now finished.
* The browser notices the DOM changes and updates the screen.

Other references to learn more:

* [Angular’s $digest is reborn in the newer version of Angular](https://blog.angularindepth.com/angulars-digest-is-reborn-in-the-newer-version-of-angular-718a961ebd3e) — explains how the ideas from AngularJS are mapped to Angular
* <https://blog.angularindepth.com/exploring-angular-dom-abstractions-80b3ebcfc02>
* [Everything you need to know about change detection in Angular](https://blog.angularindepth.com/everything-you-need-to-know-about-change-detection-in-angular-8006c51d206f) — explains in great detail how change detection works under the hood
* [Change Detection Explained](http://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html) — Thoughtram blog Feb 22, 2016 — probably the best reference out there
* Savkin’s [Change Detection Reinvented](https://www.youtube.com/watch?v=jvKGQSFQf10) video — definitely watch this one
* [How does Angular 2 Change Detection Really Work?](http://blog.jhades.org/how-does-angular-2-change-detection-really-work/)- jhade’s blog Feb 24, 2016
* [Brian’s video](https://www.youtube.com/watch?v=3IqtmUscE_U) and [Miško’s video](https://www.youtube.com/watch?v=V9Bbp6Hh2YE) about Zone.js. Brian’s is about Zone.js. Miško’s is about how Angular 2 uses Zone.js to implement change detection. He also talks about change detection in general, and a little bit about onPush.
* Victor Savkins blog posts: [Change Detection in Angular 2](http://victorsavkin.com/post/110170125256/change-detection-in-angular-2), [Two phases of Angular 2 applications](http://victorsavkin.com/post/114168430846/two-phases-of-angular-2-applications), [Angular, Immutability and Encapsulation](http://victorsavkin.com/post/110170125256/change-detection-in-angular-2). He covers a lot of ground quickly, but he can be terse at times, and you’re left scratching your head, wondering about the missing pieces.
* [Ultra Fast Change Detection](https://docs.google.com/document/d/1QKTbyVNPyRW-otJJVauON4TFMHpl0zNBPkJcTcfPJWg/edit) (Google doc) — very technical, very terse, but it describes/sketches the ChangeDetection classes that get built as part of the tree

Template driven:

Using $event and type like keyboardEvent specically

another way to get the user data: use Angular [**template reference variables**](https://angular.io/guide/template-syntax#ref-vars). These variables provide direct access to an element from within the template with a hash (or pound) character (#) like refs in react .. uncontrolled component

<input #box (keyup)="0">

### **Observations**

* **Use template variables to refer to elements** — The newHero template variable refers to the <input> element. You can reference newHero from any sibling or child of the <input> element.
* **Pass values, not elements** — Instead of passing the newHero into the component's addHero method, get the input box value and pass that to addHero.
* **Keep template statements simple** — The (blur) event is bound to two JavaScript statements. The first statement calls addHero. The second statement, newHero.value='', clears the input box after a new hero is added to the list.

form builder -

Template-driven > like ng1 we can validating in template

Reactive-forms > we need code in ts to handle validation instead of writing in template

name = new [FormControl](https://angular.io/api/forms/FormControl)(''); for each input

In template <input type="text" [formControl]="name">

**Formgroup**

profileForm = new [FormGroup](https://angular.io/api/forms/FormGroup)({

firstName: new [FormControl](https://angular.io/api/forms/FormControl)(''),

lastName: new [FormControl](https://angular.io/api/forms/FormControl)(''),

});

**Used as**

<form [formGroup]="profileForm">

<label>First Name: <input type="text" [formControlName](https://angular.io/api/forms/FormControlName)="firstName"></label>

**Submit as**

<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">

**By formBuilders can make validate easily**

hostbinding

ViewContainerRef

diff btw angular versions

Angular 6 – ng update, add external party libs, service injection in providers reduced since we can write provide in root in injectable enables to provision to rootmodule so no need to import in appmodule

Basically some programmatic changes, syntax , like ng-content template [style] httpClient, ngfactory build, performance wise, cli update, tree shaking , SSR

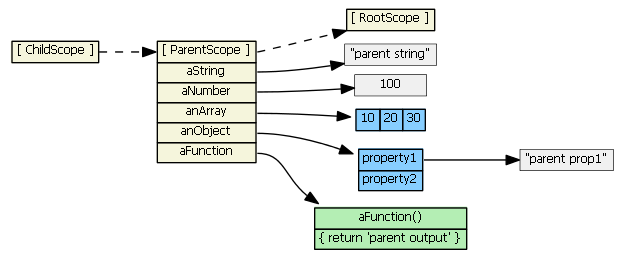
Angular 1-> 2 :

Primitives don’t change values changed from child to parent when parent send values to child initially. But objects will change since both point to same reference

<https://github.com/angular/angular.js/wiki/Understanding-Scopes>

<http://next.plnkr.co/edit/zZfUQN?p=preview&utm_source=legacy&utm_medium=worker&utm_campaign=next&preview>

Suppose parentScope has properties aString, aNumber, anArray, anObject, and aFunction. If childScope prototypically inherits from parentScope, we have:



If we try to access a property defined on the parentScope from the child scope, JavaScript will first look in the child scope, not find the property, then look in the inherited scope, and find the property. (If it didn't find the property in the parentScope, it would continue up the prototype chain... all the way up to the root scope).

<https://stackblitz.com/edit/angular-h1jy2t>

In angular 2 > …. changing value of object does not change value in object in parent component but in angular 1 it changes.

Object value passed from parent to child

<https://stackblitz.com/edit/angular-racctn?file=src%2Fapp%2Fhello.component.html>

**Rxjs operators -**

* Filtering Operators, Conditional Operators, Creation Observables
* Mathematical Operators, Transformation Operators, Utility Operator

Fork join - is equal to join multiple observable to emit after all the response are successful and completed like promise.all()

Create: create observable with callback funtion to create .next() , .error(), .complete()

Of: emits values simultaniealy

Map, filter

<https://coursetro.com/posts/code/150/RxJS-Operators-Tutorial---Learn-How-to-Transform-Observables>

Change detection strategy

Input() and output() => Why do we do this? This creates a pure uni-directional dataflow. The data comes from AppComponent, flows into <counter>, the counter makes a change, and emits that change back to the parent on our command - via the EventEmitter we setup. Once we’ve got that data back up, we merge those changes back into our parent (stateful) component.

Has two – default n onPush

Default behaves when one state/ model changes , all components will be rendered.

it will become performance bottleneck in complex Single Page Applications when lot of events are flowing through the system.

OnPush behaves changes only to particular component not triggers all component. It works when new reference is given to child comps instead of mutating

Angular will skip the change detection for a component as long as the references to the inputs do not change (immutable).

equivalent to utilizing the “[shouldComponentUpdate()](https://reactjs.org/docs/optimizing-performance.html" \l "shouldcomponentupdate-in-action)” life-cycle hook in React to cut down unnecessary render cycles

change detection in Angular framework is unidirectional and top-down (unlike the digest cycles in AngularJS)

Basically application state change can be caused by three things:

* **Events** - click, submit, …
* **XHR** - Fetching data from a remote server
* **Timers** - setTimeout(), setInterval()

<https://hackernoon.com/angular-2-4-visualizing-change-detection-default-vs-onpush-3d7ed1f69f8e>

proxy for spa projects :

"proxy": {

"/msapi/address/v1/serviceavailability/address": {

"target": "http://localhost:8088/",

"secure": false

},

How to manually trigger change detection strategy

### Triggering change detection manually in Angular

I’m writing an Angular component that has a property Mode(): string. I would like to be able to set this property programmatically not in response to any event. The problem is that in the absence of a browser event, a template binding {{Mode}} doesn't update. Is there a way to trigger this change detection manually?

Try one of these:

* [ApplicationRef.tick()](https://angular.io/docs/ts/latest/api/core/index/ApplicationRef-class.html#!#tick-anchor) - similar to AngularJS's $rootScope.$digest() -- i.e., check the **full component tree**
* [NgZone.run(callback)](https://angular.io/docs/ts/latest/api/core/index/NgZone-class.html#!#run-anchor) - similar to $rootScope.$apply(callback) -- i.e., evaluate the callback function inside the Angular zone. I think, but I'm not sure, that this ends up checking the full component tree after executing the callback function.
* [ChangeDetectorRef.detectChanges()](https://angular.io/docs/ts/latest/api/core/index/ChangeDetectorRef-class.html#!#detectChanges-anchor) - similar to $scope.$digest() -- i.e., **check only this component and its children**

You can inject ApplicationRef, NgZone, or ChangeDetectorRef into your component.

Zones

promise vs observable

const source = Rx.Observable.of({name: 'Brian'}, [1,2,3], function hello(){ return 'Hello'});

//output: {name: 'Brian}, [1,2,3], function hello() { return 'Hello' }

const subscribe = source.subscribe(val => console.log(val));

In observable return multple values but promise return one value with lot of objects or value inside.

Better not to compare promise with observable since observable is a producer of data or stream of data and can be consumed only when subscribed I.e like a calling a function.

Normal function returns single data;

Observable returns mutliple data, for Ex: can be async calls data, event handler data, timer funtions

Subject:

We are creating two observables that are independent of each other. But what if you need the second observer to get the same events has the first?

|  |
| --- |
| const interval$ = Rx.Observable.interval(1000); |
|  |  |
|  | interval$.subscribe(console.log); |
|  |  |
|  | setTimeout(() => { |
|  | interval$.subscribe(console.log); |
|  | }, 2000); |

Subject can act as a bridge/proxy between the source Observable and many observers, making it possible for multiple observers to **share** the same Observable execution.

|  |
| --- |
| const interval$ = Rx.Observable.interval(1000); |
|  | const subject = new Rx.Subject(); |
|  | interval$.subscribe(subject); |

Subject is observing the interval Observable, so every time the interval send values to the Subject, the Subject send this values to all his observers.

<https://netbasal.com/understanding-subjects-in-rxjs-55102a190f3>

Angular 1 – digest, watch, apply

Model when attached to scope , model available to scope so when changed in template or js , scope variable changes as two way binding. When model attached to scope, angular automatically attaches to watch function --> then watch listener function calls digest when model changes

**When Do You Call** $apply() **Manually?**

If AngularJS usually wraps our code in $apply() and starts a $digest cycle, then when do you need to do call $apply() manually? Actually, AngularJS makes one thing pretty clear. It will account for only those model changes which are done inside AngularJS’ context (i.e. the code that changes models is wrapped inside $apply()). Angular’s built-in directives already do this so that any model changes you make are reflected in the view. However, if you change any model outside of the Angular context, then you need to inform Angular of the changes by calling $apply()manually. It’s like telling Angular that you are changing some models and it should fire the watchers so that your changes propagate properly.

For example, if you use JavaScript’s setTimeout() function to update a scopemodel, Angular has no way of knowing what you might change. In this case it’s your responsibility to call $apply() manually, which triggers a $digest cycle. Similarly, if you have a directive that sets up a DOM event listener and changes some models inside the handler function, you need to call $apply() to ensure the changes take effect.

<https://codepen.io/Sandeep92/pen/fukyn> - without apply

<https://codepen.io/Sandeep92/pen/bEmBn> - with apply updates the UI where setTimeout is out of angular scope

**Note**: By the way, you should use $timeout service whenever possible which is setTimeout() with automatic $apply() so that you don’t have to call $apply()manually.

<https://www.sitepoint.com/understanding-angulars-apply-digest/>

=======================

aot compiler build and before aot compiler what used

**We need compilation for achieving higher level of efficiency of our Angular applications.** By efficiency I mostly mean performance improvements but also energy and sometimes bandwidth consumption.

AngularJS 1.x had quite a dynamic approach for both rendering and change detection. For instance, the AngularJS 1.x compiler is quite generic. It is supposed to work for any template by performing a set of dynamic computations. Although this works great in the general case, the JavaScript Virtual Machines (VM) [struggles with optimizing the calculations on lower level](http://mrale.ph/blog/2012/06/03/explaining-js-vms-in-js-inline-caches.html) because of their dynamic nature. **Since the VM doesn’t know the shapes of the objects which provide context for the dirty-checking logic (i.e. the so called scope), it’s inline caches get a lot of misses which slows the execution down.**

Angular, version 2 and above, took different approach. Instead of using the same logic for performing rendering and change detection for each individual component, the framework **generates** VM-friendly code at runtime or build time. **This allows the JavaScript virtual machine to perform property access caching and execute the change detection/rendering logic much faster.**

What needs to be compiled?” as well. We want to compile the templates of the components to JavaScript classes. These classes have methods that contain logic for detecting changes in the bindings and rendering the user interface.

HMR

Hot Module Reload is the ability of the application to update itself when the source code is changed — without full page reload !

<https://medium.com/@kubal5003/angular-4-hot-module-reload-explained-e832ea616304>

======================

Tips:

try to import this way:

import {Observable} from 'rxjs/Rx';

<https://stackblitz.com/> - online ide for angular, react,js,ionic,native

<https://codesandbox.io/>