Angular Change Detection

Our Goal

In this lesson we will understand the following:

- What is change detection (CD)?
- When does Angular perform CD?
- How does Angular perform CD?
- Understand change detection strategies
- Learn how to optimize the CD in our app

What is CD?

- Angular saves the state of our component
- The state of a component can change
  - What can cause a state change?
- Angular needs to know when the state of our component might change
- When the state might be changed angular needs to re render the changes of the component

When and How?

To understand change detection we need to ask 2 questions:

- When does Angular suspects change in state?
- How does Angular rerender the view?

When in Angular 5

To understand when angular suspects a change, we have to dig into a new library called **zone.js**

What is zone.js?

- Zone is an execution context
- Async tasks run in the same zone they are registered
- Currently at stage 0
- Maintained by google
- Benefits of using zones
  - Debugging
  - Transfer data in zone
  - Connect hook when async action happens in zone
- When zone is included it wraps all the execution context in a special root zone
- All zones are children of the root zone
- You can only be on one zone

Create a New Zone

- Let’s play with zones using node
- Install typescript and create a default config
- Import zone.js
- A zone has to have a parent
- To create a zone we call fork on the parent zone
- We pass the fork **ZoneSpec** with the following properties:
    - name - used for debugging
    - properties - used to transfer data
    - hooks - you can attach a hook to a zone will be covered later
- Let’s create a child zone from the parent zone and name that child zone angular

Run Code in a Zone

- To run a code in a zone you call **zone.run**
- Run the function sync in the zone you choose
- After execution is finished the previous zone is restored
- You can only be in one zone
- You can get the current zone you are in by calling the static method **Zone.current**
- By default if you include zone you are running on the root zone
- Zone needs to monkey patch async api and enter the zone when running async code
- Let's try to run some async code in our Angular zone

Zone Hooks

- Hooks allow parent zone to intercept events on child zone
- Available common hooks
  - **onHasTask -** get notified when the queue status of the zone changes
  - **onScheduledTask -** get notified when a task is added to the queue
  - **onInvokeTask -** when a task is removed from the queue
  - **onInvoke** - when a zone is entered. How can we enter a zone?
- Let’s try to use **onHasTask** to detect when the queue state is changed to empty
  - Hint hint! can you think of a possible implementation for this hook?

NgZone

- Angular runs all context of the framework under a zone called **Angular**
- There is a service which can be dependency injected called **NgZone** which wraps the Angular zone
- In the private **_inner** property you can access the wrapped zone
- The **_outer** property is the parent zone which is root
- You can use this service to run code outside the angular zone which will not update the UI **runOutsideAngular**
- Or you can use **run** to run code in Angular zone
- Angular attached hooks to the angular zone and can tell when async tasks or events finished and can then run change detection

How in Angular 5

- When component is instantiated Angular creates a change detector
- The CD service is called: **ChangeDetectorRef** and can be injected
- The CD is responsible for propagating the component’s binding
- You can inject that service and manually run change detection on the entire tree or just the component and its children
- Angular will traverse the tree from top to bottom a when **NgZone** detects a change

Performance Problem

- Angular traverse the tree from top to bottom
- If a child in the tree had an input event the entire tree will be checked for changes and updated
- How can we improve the performance?

CD Strategy

- By default Angular will run change detection on the entire CD tree and for every component
- In the component decorator you can specify **changeDetection** strategy
- Let’s create a simple app with an app component and a child component
- We will place a button on the parent component and change the child component strategy

OnPush

- Let’s try and add a string input to the child
- Clicking the button in the parent will change the string
- Is the child component rendering?

OnPush

- Let’s try and add a time async on the child
- Is the child component re rendering?

OnPush

- On push will run change detection when the following happen
    - The @input of the component is changed
    - Events happen on the component
- Also when OnPush set if the change detection on the child is not running so are the entire tree of children from the component
- On push will compare input change by reference not by value
- If we pass the data from a service it won't work

OnPush by reference problem

- so OnPush will consider change if the reference to the object is changed
- this means that if we are passing an object to an OnPush component and change a field in that object the component won’t rerender
- how can we set change detection to OnPush and still maintain rendering of the component when changing a field in the object?

immutable.js

- immutable data cannot be changed after created
- immutable.js offer us api to change the data but instead of changing the data it will yield a new immutable object
- immutable.js offer us immutable data structures we will cover the popular ones:
  - Map
  - List
- We can install immutable.js with npm
  - **npm install immutable --save**
- let’s try and create a map and a list and see the common api for those data structures

Improving CD with immutable

- to solve our Change Detection problem the input data that we pass to our component can be an immutable.js map
- this means that when changing the value of the map the entire reference will change and our component will re render

Manually Render Changes

- We can as Angular to inject the change detection service: **ChangeDetectorRef**
- **detectChanges -** triggers change detection from component down, this means components with on push (which is not the component that called the **detectChanges**) will not render
- **markForCheck -** will start from the parent and activate change detection on the tree until the component reached, components with on push will render as well

Observable and OnPush

- as we seen before, subscribing to an observable in the a component that has OnPush strategy, won’t cause the component to re render
- What happens when the subscribe is done not in the class but in the template of the component?
- How can we subscribe to an observable in the template?

async pipe

- once you are familiar with this pipe, this will be the most common pipe you will use
- this pipe subscribes to observable or promise and allows you get the data within the observable/promise
- example

```
@Component({...})
export class AsyncExampleComponent {
        public greeting: Observable<string>
}
```

```
<h1>
{{ greeting | async}}
</h1>
```

Improving CD with Observables

- Let’s check if OnPush component will re render is subscribing to an observable with the async pipe in the template
- we will create an observable that creates a counter that increments every second
- we will subscribe to that observable with the async pipe and check if the component re renders
- Does the component re render?
- Can we use observables to leverage OnPush strategy?

Optimizing with OnPush

- It’s possible to set all component in change detection strategy on push
- This will significantly improve the performance
- It’s easier to manage components with on push if using redux to manage the state

Summary

- Now that we understand how CD works, we can start thinking how to optimize our app, while keeping all component in an on push strategy
- We will learn how to do it easily by using redux