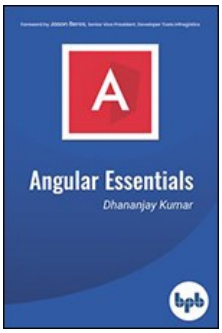

Angular Essentials: The Essential Guide to Learn Angular

by Dhananjay Kumar

BPB Publications. (c) 2019. Copying Prohibited.

Reprinted for Upendra Kumar, ACM

upendrakumar1@acm.org

Reprinted with permission as a subscription benefit of **Skillport**,

All rights reserved. Reproduction and/or distribution in whole or in part in electronic,paper or other forms without written permission is prohibited.

Chapter 10: Change Detection

In this chapter, you will learn about Change Detection in Angular. Following topics will be covered in this chapter:

- Change Detection
- Default Strategy
- onPush Strategy

Change Detection

Change Detection means updating the DOM whenever data is changed. There could be various reasons of Angular change detector to come into action and start traversing the component tree. They are:

- Events fired such as button click, etc.
- AJAX call or XHR requests.
- Use of JavaScript timer functions such as setTimeOut , SetInterval.

Angular provides two strategies for Change Detection.

1. Default strategy
2. onPush strategy

Default Strategy

In default strategy, whenever any data is mutated or changed, Angular will run change detector to update the DOM. In onPush strategy, Angular will run change detector only when new reference is passed to **@Input()** data. To update the DOM with updated data, Angular provides its own change detector to each component, which is responsible to detect change and update the DOM.

Let us say we have a **`MessageComponent`** as shown in *code listing 10.1.*

Code Listing 10.1

```
import { Component, Input } from '@angular/core';
@Component({
     selector: 'app-message',
     template: `
        <h2>
        Hey {{person.firstname}} {{person.lastname}} !
        </h2>
})
export class MessageComponent {
     @Input() person; }
```

In addition, we are using **MessageComponent** inside **AppComponent** as shown in *code listing 10.2.*

Code Listing 10.2

```
import { Component, OnInit } from '@angular/core';

@Component({
        selector: 'app-root',
        template: `
     <app-message [person]='p'></app-message>
     <button (click)='changeName()'>Change Name</button>
})
export class AppComponent implements OnInit {
   p: any;
     ngOnInit(): void {
        this.p = {
           firstname: 'Brad',
```


Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

```
            lastname: 'Cooper'
        };
    }
}
```

Let us talk through the code: all we are doing is using `MessageComponent` as child inside `AppComponent` and setting value of person using the property binding. At this point on running the application, you will get name printed as output.

Next, let us go ahead and update firstname property on the button click in `AppComponent` class as shown in *code listing 10.3.*

Code Listing 10.3

```
changeName()   {
        this.p.firstname =  'Foo';
    }
```

As soon as we changed the property of mutable object P, Angular fires the change detector to make sure that the DOM (or view) is in sync with the model (in this case object p). For each property changes, Angular change detector will traverse the component tree and update the DOM.

Let us start with understanding about component tree. An Angular application can be seen as a component tree. It starts with a root component and then go through to child components. In Angular, data flows from top to bottom in the component tree as shown in *figure 10.1*.

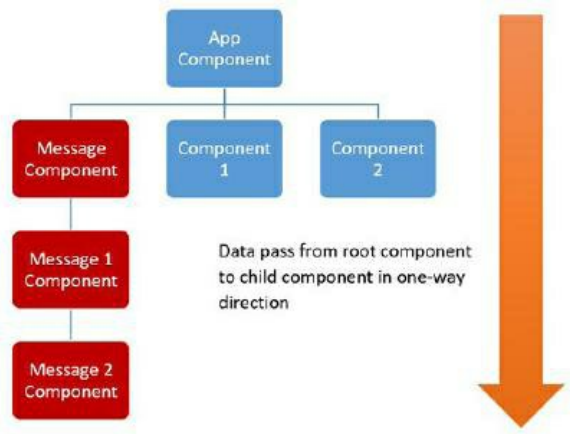

Figure 10.1

Whenever **@Input** type property will be changed, Angular change detector will start form the root component and traverse all child components to update the DOM. Any changes in primitive type's property will cause Angular change detection to detect the change and update the DOM.

In previous examples, you will find that on click of the button, the first name in the model will be changed. Then, change detection will be fired to traverse from root to bottom to update the view in **MessageComponent.**

Now, as you see, a single property change can cause change detector to traverse through the whole component tree. Traversing and change detection is a heavy process, which may cause performance degradation of application. Imagine that there are thousands of components in the tree and mutation of any data property can cause change detector to traverse all thousand components to update the DOM. To avoid this, there could be scenario when you may want to instruct Angular that when change detector should run for a component and its subtree, you can instruct a component's change detector to run only when object references changes instead of mutation of any property by choosing **onPush** Change Detection strategy.

onPush Strategy

You may wish to instruct Angular to run change detection on components and their sub-tree only when new references are passed to them versus when data is simply mutated by setting change detection strategy to `onPush`.

Let us go back to our example where we are passing object to `MessageComponent`. In the last example, we just changed firstname property and that causes change detector to run and to update the view of `MessageComponent`. However, now we want change detector to only run when reference of passed object is changed instead of just a property value. To do that let us

Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

modify **MessageComponent** to use **OnPush ChangeDetection** strategy. To do this set **changeDetection** property of **@Component** decorator to **ChangeDetectionStrategy.OnPush** as shown in *code listing 10.4.*

Code Listing 10.4

```
import { Component, Input, ChangeDetectionStrategy } from '@angular/core';

@Component({
  selector: 'app-message',
  template: `
     <h2>
     Hey {{person.firstname}} {{person.lastname}} !
     </h2>
     ',
     changeDetection: ChangeDetectionStrategy.OnPush,
})
export class MessageComponent {
  @Input() person;
}
```

At this point when you run the application, on click event of the button in **AppComponent** change detector will not run for **MessageComponent**, as only a property is being changed and reference is not changing. Since change detection strategy is set to **onPush**, now change detector will only run when reference of **@Input** property is changed as shown in *code listing 10.5.*

Code Listing 10.5

```
changeName()   {
         this.p =  {
            firstname:   'Foo',
            lastname:   'Kooper'
         };
   }
```

We are changing reference of object instead of just mutating just one property. Now when you run application, you will find on the click of the button that the DOM is being updated with new value.

By using **onPush** Change Detection, Angular will only check the tree if the reference passed to the component is changed instead of some property changed in the object. We can summarize that, and Use Immutable Object with **onPush** Change Detection to improve performance and run the change detector for component tree when object reference is changed.

We can further improve performance by using RxJS Observables because they emit new values without changing the reference of the object. We can subscribe to the observable for new value and then manually run **ChangeDetector**. Let us modify **AppComponent** to pass an observable to **MessageComponent** as shown in *code listing 10.6.*

Code Listing 10.6

```
import { Component, OnInit } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Component({
     selector: 'app-root',
     template: `
   <app-message [person]='data'></app-message>
   <button (click)='changeName()'>Change Name</button>
   '
})
export class AppComponent implements OnInit {
      p: any;
      data: any;
      ngOnInit(): void {
         this.p = {
            firstname: 'Brad',
            lastname: 'Cooper'
         };
         this.data = new BehaviorSubject(this.p);
        }

        changeName() {
           this.p = {
```


Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

```
            firstname: 'Foo',
            lastname: 'Kooper'
         };
         this.data.next(this.p);
      }
 }
```

In the code, we are using **`BehaviorSubject`** to emit next value as an observable to the **`MessageComponent`**. We have imported **`BehaviorSubject`** from RxJS and wrapped object p inside it to create an observable. On the click event of the button, it's fetching the next value in observable stream and passing to **`MessageComponent`**.

In the **`MessageComponent`**, we have to subscribe to the person to read the data as shown in *code listing 10.7.*

Code Listing 10.7

```
import { Component, Input, ChangeDetectionStrategy,
OnInit } from '@angular/core';
import { Observable } from 'rxjs';

@Component({
     selector: 'app-message',
     template: `
        <h2>
Hey {{_data.firstname}} {{_data.lastname}} !
        </h2>
        ',
    changeDetection: ChangeDetectionStrategy.OnPush,
})
export class MessageComponent implements OnInit {
     @Input() person: Observable<any>;
     _data;
     ngOnInit() {
        this.person.subscribe(data => {
           this._data = data;
        });
     }
}
```

Now, on click of the button, a new value is being created, however, a new reference is not being created as object is an observable object. Since a new reference is not created, due to **`onPush ChangeStrategy`**, Angular is not doing change detection. In this scenario, to update the DOM with new value of observable, we have to manually call the change detector as shown in *code listing 10.8.*

Code Listing 10.8

```
import { Component, Input, ChangeDetectionStrategy,
OnInit, ChangeDetectorRef } from '@angular/core';
import { Observable } from 'rxjs';

@Component({
     selector: 'app-message',
     template: '
        <h2>
        Hey {{_data.firstname}} {{_data.lastname}} !
        </h2>
        ',
     changeDetection: ChangeDetectionStrategy.OnPush,
})
export class MessageComponent implements OnInit {

     @Input() person: Observable<any>;
     _data;

     constructor(private cd: ChangeDetectorRef) { }
     ngOnInit() {
        this.person.subscribe(data => {
           this._data = data;
           this.cd.markForCheck();
        });
     }
}
```


Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

We have imported `ChangeDetectorRef` service and injected it, and then calling `markForCheck()` manually to cause change detector to run each time observable emits a new value. Now when you run application, and click on button, observable will emit new value and change detector will update the DOM also, even though a new reference is not getting created.

In a nutshell,

- If Angular ChangeDetector is set to default then for any change in any model property, Angular will run change detection traversing component tree to update the DOM.
- If Angular ChangeDetetor is set to onPush then Angular will run change detector only when new reference is being passed to the component.
- If observable is passed to the onPush change detector strategy enabled component then Angular ChangeDetctor has to be called manually to update the DOM.

Summary

Angular change detection strategy is very essential for performance of an application. You need to know when to use default strategy and when to use `onPush` strategy. In this chapter, you learnt about:

- Change Detection
- Default Strategy
- onPush Strategy

Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.