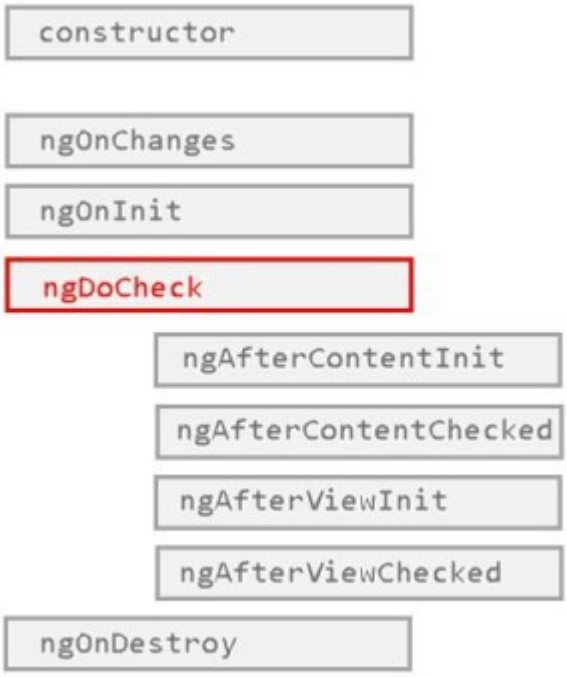
If you think `ngDoCheck` means your component is being checked—read this article

Max Koretskyi aka Wizard Follow
Aug 14, 2017 · 6 min read



> **We're organizing the first Angular conference in Kiev, Ukraine as part of Angular In Depth community. Read more about it here.** I'll be happy to see you at the conference. We'll have experts from Angular team and the community and we made tickets affordable for most developers (around $150), even if you pay out of your own pocket.

I use OnPush, why is ngDoCheck triggered?

There's one question that comes up again and again on stackoverflow. The question is about `ngDoCheck` lifecycle hook that is triggered for a

component that implements `OnPush` change detection strategy. It's usually formulated something like:

> *I have used* `OnPush` *strategy for my component and no bindings have changed, but the* `ngDoCheck` *lifecycle hook is still triggered. Is the strategy not working?*

It's an interesting question which stems from misunderstanding of when the the `ngDoCheck` lifecycle hook is triggered and why it's provided by the framework. This article gives an answer to this common question by showing when the hook in question is triggered and what's its purpose.

> I work as a developer advocate at **ag-Grid**. If you're curious to learn about data grids or looking for the ultimate Angular data grid solution, give it a try with the guide "**Get started with Angular grid in 5 minutes**". I'm happy to answer any questions you may have. **And follow me to stay tuned!**

• • •

When is ngDoCheck triggered

The official docs don't tell much about this lifecycle hook:

> *Detect and act upon changes that Angular can't or won't detect on its own.*
> *Called during every change detection run, immediately after* `ngOnChanges()` *and* `ngOnInit()`.

So we know that it's triggered after `ngOnChanges` and `ngOnInit` but is the component being checked or not when it's triggered? To answer that question, we need to first define "component check". The article Everything you need to know about change detection in Angular states that there are three core operations related to component change detection:

- update child component input bindings
- update DOM interpolations
- update query list

Besides these core operations Angular triggers lifecycle hooks as part of change detection. What is interesting is that the hooks for the child component are triggered when the parent component is being checked. Here is the small example to demonstrate that. Suppose we have the following components tree:

```
ComponentA
    ComponentB
        ComponentC
```

So when Angular runs change detection the order of operations is the following:

```
Checking A component:
  – update B input bindings
  – call NgDoCheck on the B component
  – update DOM interpolations for component A

 Checking B component:
    – update C input bindings
    – call NgDoCheck on the C component
    – update DOM interpolations for component B

   Checking C component:
      – update DOM interpolations for component C
```

If you read the article on change detection I referenced above you will notice that it's a bit abridged list of operations, but it will do for the purpose of demonstrating when `ngDoCheck` is triggered.

You can see that `ngDoCheck` is called on the child component **when the parent component is being checked**. Now suppose we implement `onPush` strategy for the `B` component. How does the flow change? Let's see:

```
Checking A component:
  – update B input bindings
  – call NgDoCheck on the B component
  – update DOM interpolations for component A

 if (bindings changed) –> checking B component:
    – update C input bindings
    – call NgDoCheck on the C component
    – update DOM interpolations for component B
```

```
    Checking C component:
       - update DOM interpolations for component C
```

So with the introduction of the `OnPush` strategy we see the small condition `if (bindings changed) -> checking B component` is added before `B` component is checked. If this condition doesn't hold, you can see that Angular won't execute the operations under `checking B component`. However, the `NgDoCheck` on the `B` component is still triggered even though the `B` component will not be checked. It's important to understand that the hook is triggered only for the top level `B` component with `OnPush` strategy, and not triggered for its children— `C` component in our case.

So, the answer to the question:

> *I have used `OnPush` strategy for my component, but the `ngDoCheck` lifecycle hook is still triggered. Is the strategy not working?*

is—the strategy is working. The hook is triggered by design and the next chapter shows why.

• • •

Why do we need ngDoCheck?

You probably know that Angular tracks binding inputs by object reference. It means that if an object reference hasn't changed the binding change is not detected and change detection is not executed for a component that uses `OnPush` strategy. In AngularJS this is the standard approach as well and in the following example the changes to `o` object won't be detected:

```
const o = {some: 3};

$scope.$watch(
  () => {  return o;},
  () => {  console.log('changed'); } // nothing is logged
);

$timeout(() => {  o.some = 4; }, 2000);
```

But AngularJS has a few variations of standard `$watch` function to allow tracking object and array mutations—deep watch and collection

watch. To enabled the deep watch you have to pass the third parameter `true` to the `$watch` function:

```
$scope.$watch(
  () => {  return o;},
  () => {  console.log('changed'); }, // logs `changed`
  true
);
```

And to watch collections you have to use `$watchCollection` method:

```
const o = [3];

$scope.$watchCollection(
  () => {  return o;},
  () => {  console.log('changed'); } // logs `changed`
);

$timeout(() => {  o.push(4) }, 2000);
```

But there's no equivalent in Angular. If we want to track an object or an array mutations we need to manually do that. And if discover the change we need to let Angular know so that it will run change detection for a component even though the object reference hasn't changed.

Let's use the example from AngularJS in the Angular application. We have `A` component that uses `OnPush` change detection strategy and takes `o` object through input binding. Inside the component template it references the `name` property:

```
@Component({
  selector: 'a-comp',
  template: `<h2>The name is: {{o.name}}</h2>`,
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class AComponent {
  @Input() o;
}
```

And we also have the parent `App` component that passes the `o` object down to the child `a-comp`. In 2 seconds it mutates the object

by updating the `name` and `id` properties:

```
@Component({
  selector: 'my-app',
  template: `
    <h1>Hello {{name}}</h1>
    <a-comp [o]="o"></a-comp>
  `,
})
export class App {
  name = `Angular! v${VERSION.full}`;
  o = {id: 1, name: 'John'};

  ngOnInit() {
    setTimeout(() => {
      this.o.id = 2;
      this.o.name = 'Jane';
    }, 2000);
  }
}
```

Since Angular tracks object reference and we mutate the object without changing the reference Angular won't pick up the changes and it will not run change detection for the `A` component. Thus the new `name` property value will not be re-rendered in DOM.

Luckily, we can use the `ngDoCheck` lifecycle hook to check for object mutation and notify Angular using markForCheck method. In the implementation above we will track only `id` property mutation but if required you can implement full-blown change tracking similar to deep watch in AngularJS.

Let's do just it:

```
export class AComponent {
  @Input() o;

  // store previous value of `id`
  id;

  constructor(private cd: ChangeDetectorRef) {}

  ngOnChanges() {
    // every time the object changes
    // store the new `id`
    this.id = this.o.id;
  }

  ngDoCheck() {
    // check for object mutation
    if (this.id !== this.o.id) {
```

```
        this.cd.markForCheck();
      }
    }
  }
```

Here is the plunker that shows that approach. To practice on your own you can try to implement the full-blown deep watch and watch collection approaches similar to AngularJS.

One thing to bear in mind is that Angular team recommends using immutable objects instead of object mutations so that you can rely on default Angular bindings change tracking mechanism. But since it's not always possible as you've just learnt Angular provides a fallback in the form of `ngDoCheck` lifecycle hook.

. . .

Thanks for reading! If you liked this article, hit that clap button below 👏. It means a lot to me and it helps other people see the story.

For more insights follow me on Twitter and on Medium.