

AfterViewInit, AfterViewChecked, AfterContentInit & AfterContentChecked in Angular

3 Comments / 7 minutes of reading / March 9, 2023



[Angular Tutorial](#)

[Angular Decorators](#)



[ContentChild &
ContentChildren](#)

AfterViewInit, AfterContentInit, AfterViewChecked & AfterContentChecked are the [lifecycle hooks](#). Angular raise them during the [lifecycle of a Component](#). In this tutorial, we will learn what are they and when Angular invokes them. We also learn the difference between the AfterViewInit Vs AfterContentInit Vs AfterViewChecked & AfterContentChecked.

Table of Contents

[Lifecycle hooks recap](#)

[Content Vs View](#)

[Content](#)

[View](#)

[AfterContentInit](#)

[AfterContentChecked](#)

[AfterViewInit](#)

[AfterViewChecked](#)

[Init Vs Checked](#)

[Init Hooks](#)[Checked Hooks](#)[Example](#)[On Component Creation](#)[On Component Running](#)[Content is first](#)[AfterViewInit & AfterViewChecked fires after child components are ready](#)[Init Hook fires only once](#)[Avoid using Checked Hooks](#)[Do not modify bindings in Checked Hooks](#)[Reference & Source Code](#)

Lifecycle hooks recap

The life of a [component](#) (or directive) starts, when angular instantiates the component.

Instantiation starts with invoking the component's constructor and injecting the services via [dependency injection](#).

Once the Angular instantiates the component, it starts the change detection cycle for the component. It checks & updates any data-bound [input property](#) of the component & Initializes the component. It then raises the following life cycle hooks.

[Onchanges](#), if Angular detects any changes to the [Input property](#). It runs every time angular detects an input change.

[OnInit](#), which tells us that the component is ready. This hook gives us a chance to run any initialization logic, updates a few properties, etc. This hook runs only once.

[DoCheck](#) which allows us to run custom change detection because change detection may overlook some of the changes. This hook runs during every change detection cycle.

After this angular invokes four more hooks. They are `AfterContentInit`, `AfterContentChecked`, `AfterViewInit` & `AfterViewChecked`. We will look at them in detail.

Finally, when we remove the component, Angular invokes the [ngOnDestroy](#) hook and then destroys the component.

Content Vs View

Before diving into these hooks, we need to know the difference between **Content** & **View**. The hooks `AfterContentInit` & `AfterContentChecked` deals with the **Content**, While `AfterViewInit`, `AfterViewChecked` deals with the **View**.

Content

Content refers to the external content injected into this component using the [Content Projection](#).

[Content projection](#) is a way to pass the HTML content from the parent component to the child component. The child component will display the template in a designated spot. We use the [ng-content](#) element to create a spot in the template of the child component as shown below.

```
1  
2 <h2>Child Component</h2>  
3 <ng-content></ng-content> <!-- place holder for content from parent -->  
4
```

Parent injects the content between the opening & closing element. Angular passes this content to the child component.

```
1  
2  
3 <h1>Parent Component</h1>  
4 <app-child> This <b>content</b> is injected from parent</app-child>  
5
```

View

View refer to the the template of the component.

AfterContentInit

The `AfterContentInit` is the Life cycle hook that angular calls after the Component's [content](#) has been fully initialized and injected into Components View.

Angular also updates the properties decorated with the [ContentChild](#) and [ContentChildren](#) before raising this hook.

Angular calls this hook even if there is no projected content in the component

This hook fires after the [ngDoCheck](#) hook.

Fires only once, during the first change detection cycle, immediately after the creation of the component.

AfterContentChecked

AfterContentChecked is the life cycle hook, that angular calls during every change detection cycle after Angular completes the checking of the content for changes.

Angular also updates the properties decorated with the [ContentChild](#) and [ContentChildren](#) before raising this hook.

This hook fires after the [ngDoCheck](#) & AfterContentInit .

AfterViewInit

A lifecycle hook that Angular calls during the change detection after it completes initialization of component's view and its child views.

Angular also updates the properties decorated with the [ViewChild](#) & [ViewChildren](#) properties before raising this hook.

Use this hook to handle any additional initialization tasks.

Fires only once, during the first change detection cycle, immediately after the creation of the component.

AfterViewChecked

A lifecycle hook that Angular calls after the change detector completes the checking of a component's view and child views for changes.

Angular also updates the properties decorated with the [ViewChild](#) & [ViewChildren](#) properties before raising this hook.

Init Vs Checked

Init Hooks

Angular fires the AfterContentInit & AfterViewInit hooks, when the content or view is initialized for the first time. That happens during the first change detection cycle, which angular invokes immediately after the instantiation of the component.

Checked Hooks

Angular fires the AfterContentChecked & AfterViewChecked hooks, where Angular checks if the the content or view has changed. i.e previously rendered content or view is same as the current content or view.

Example

Now, now let use see above hooks using an example

child-component.ts

```
1
2 import { Component } from "@angular/core";
3
4 @Component({
5   selector: "app-child",
6   template: `
7     <div style="border:solid; border-width:1px;">
8
9       <h2>Child Component</h2>
10
11       message : <input [(ngModel)]="message">
12
13       <p> Injected Content Below</p>
14       <ng-content></ng-content>
15
16     </div>
17   `,
18 })
19 export class ChildComponent {
20   message = ""
21
22   ngOnChanges() {
23     console.log(' ChildComponent==>ngOnChanges');
24   }
25
26   ngOnInit() {
27     console.log(' ChildComponent==>ngOnInit');
28   }
29
30   ngDoCheck() {
31     console.log(' ChildComponent==>ngDoCheck');
32   }
33 }
```

```

34  ngAfterContentInit() {
35      console.log(' ChildComponent==>ngAfterContentInit');
36  }
37
38  ngAfterContentChecked() {
39      console.log(' ChildComponent==>ngAfterContentChecked');
40  }
41
42  ngAfterViewInit() {
43      console.log(' ChildComponent==>AfterViewInit');
44  }
45
46  ngAfterViewChecked() {
47      console.log(' ChildComponent==>AfterViewChecked');
48  }
49
50 }
51
52

```

1. We have a input FORM element, which is bound to message property of the component using [ngModel](#)
2. <ng-content> </ng-content> is a place holder for the injected content from the parent.
3. The code ex: console.log(' ChildComponent==>ngOnChanges'); in the component logs to console, whenever change detection invokes the hook;

app.component.ts

```

1
2  import { Component, ViewChild } from "@angular/core";
3  import { ChildComponent } from "../child-component";
4
5  @Component({
6      selector: "my-app",
7      template: `
8
9      AfterContentInit, AfterContentChecked, AfterViewInit, AfterViewChecked
10
11      <app-child>
12          <b>Injected</b> content from the <i>Parent</i>
13      </app-child>

```



```
14
15
16
17 ` ,
18 })
19 export class AppComponent {
20
21   message="";
22
23   @ViewChild(ChildComponent) viewChild: ChildComponent;
24
25   ngOnChanges() {
26     console.log('AppComponent==>ngOnChanges');
27   }
28
29   ngOnInit() {
30     console.log('AppComponent==>ngOnInit');
31   }
32
33   ngDoCheck() {
34     console.log('AppComponent==>ngDoCheck');
35   }
36
37   ngAfterContentInit() {
38     console.log('AppComponent==>ngAfterContentInit');
39   }
40
41   ngAfterContentChecked() {
42     console.log('AppComponent==>ngAfterContentChecked');
43   }
44
45   ngAfterViewInit() {
46     console.log('AppComponent==>AfterViewInit');
47   }
48
49   ngAfterViewChecked() {
50     console.log('AppComponent==>AfterViewChecked');
51     this.message=this.viewChild.message;
52   }
53
54 }
55
56
```

1. We are injecting the content to `<app-child>` by placing the content `Injected` content from the `<i>Parent</i>` within the element tag.

2. Using ViewChild query to update the reference to the child component in the property ChildComponent
3. this.message=this.viewChild.message; updates the message property of this component with that of ChildComponent

Now let us run this app and see what happens.

On Component Creation

On the component creation, the hooks are fired in the following order.

1. [OnChanges](#)
2. [OnInit](#)
3. [DoCheck](#)
4. AfterContentInit
5. AfterContentChecked
6. AfterViewInit
7. AfterViewChecked

Hooks on application startup

AfterContentInit, AfterContentChecked,
AfterViewInit, AfterViewChecked

Child Component

message

Injected Content Below

Injected content from the *Parent*

The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. It displays the sequence of Angular lifecycle hooks for both the parent and child components. The hooks are grouped into two main categories:

- Child Component Hooks:** These are the first four hooks in the sequence: `ngOnInit`, `ngDoCheck`, `ngAfterContentInit`, and `ngAfterContentChecked`. They are followed by `ngAfterViewInit` and `ngAfterViewChecked`.
- Parent Component Hooks:** These are the last four hooks in the sequence: `ngDoCheck`, `ngAfterContentChecked`, `ngAfterViewInit`, and `ngAfterViewChecked`.

Annotations on the image highlight the following points:

- child component hooks:** A callout points to the first four hooks.
- Parent Component Hooks:** A callout points to the last four hooks.
- On Application startup, change detection fires twice:** A callout points to the `ngDoCheck` hooks for both the parent and child components, indicating that change detection is triggered twice during the initial startup.

Angular runs the change detection twice on application startup. Hence, the

On Component Running

Once the component is initialized, Angular do not fire the init hooks. Only the the checked hooks are invoked.

1. [DoCheck](#)
2. AfterContentChecked
3. AfterViewChecked

Content is first

Angular initializes and checks the content first, before the components view & child views.

AfterViewInit & AfterViewChecked fires after child components are ready

After content, angular initializes the components view. It also initializes the child views & runs their change detection. Hence, by the time we receive the hooks AfterViewInit & AfterViewChecked, the current component & all its children are ready to render.

Init Hook fires only once

Init hooks fire only once, during the **first** change detection cycle, which angular fires immediately after the creation of the component. This makes it the best place to run some custom initialization logic. Use the AfterContentInit for content related initialization & AfterViewInit for view related initializations.

Avoid using Checked Hooks

Checked hooks runs on every change detection cycle. For example when you just click on the input element and move away.

Hence it is better to avoid using these hooks. If you choose to implement these hooks then ensure that your code is extremely lightweight otherwise it may slow down the application.

Do not modify bindings in Checked Hooks

Open the `ngAfterViewChecked` method of the `app.component.ts`. here, we assign value of the `viewChild.message` to the `message` variable of parent component. Code does not raise any errors.

```
1  
2 ngAfterViewChecked() {  
3   console.log('AppComponent==>AfterViewChecked');  
4   this.message=this.viewChild.message;  
5 }  
6
```

Now add the following to the template of the `app.component.ts`

```
1  
2 message from child {{message}}  
3
```

and run the app.

There are two important points to note here.

1. The `{{message}}` in the `app.component.ts` waits a tick before updating
2. `ExpressionChangedAfterItHasBeenCheckedError`

Although the code looks fine, but this is what happens

1. Initially the value of `message` is empty
2. We enter **h** in input element. This starts a change detection cycle.
3. It checks the value of `message` variable. Its value is empty. **Hence updates the DOM with empty string.**
4. Angular fires the `AfterViewChecked` hook
5. We update the `message` variable to **h**.
6. Angular runs another check to see if all the bindings values are correct. If detects `message` value is now **h** is different from when it checked it in the step 3. It raises the `ExpressionChangedAfterItHasBeenCheckedError`
7. Change detection cycle ends.

Now, as you can see at the end of change detection **h** is not updated in DOM.

1. We, type **e** in input element.
2. A change detection cycle starts
3. It checks the value of message variable. Its value is **h** . Hence Angular updates the DOM with **h** .
4. Angular fires the AfterViewChecked hook
5. We update the message variable to **he** .
6. Angular runs another check to see if all the bindings values are correct. If detects message value is now h is different from when it checked it in the step 3. It raises the ExpressionChangedAfterItHasBeenCheckedError .
7. Change detection cycle ends.

The ***step 6 only happens only in development mode***. Angular only checks the bindings, but does not update the DOM if it detects any changes. It only raises the error.

Reference & Source Code

- [Source Code](#)
- [AfterViewInit](#)
- [AfterViewChecked](#)
- [AfterContentInit](#)
- [AfterContentChecked](#)

Read More

1. [Angular Tutorial](#)
2. [Typescript Tutorial](#)
3. [Data binding in Angular](#)
4. [Life Cycle hooks](#)
5. [OnInit](#)