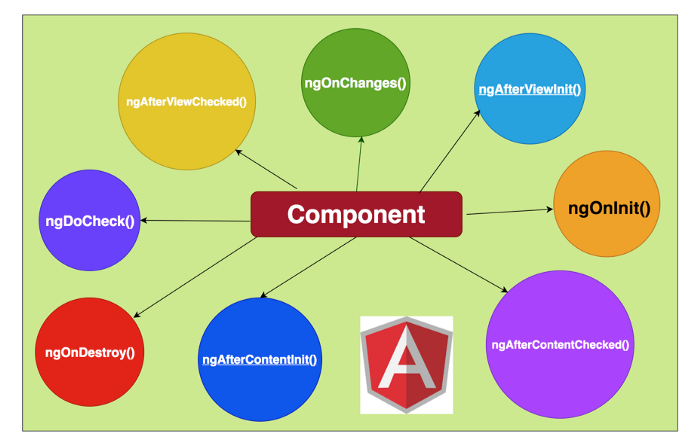
**Angular — Understanding Angular lifecycle hooks with a Sample Project**

Understanding Component lifecycle hooks are very important for any successful Angular project.

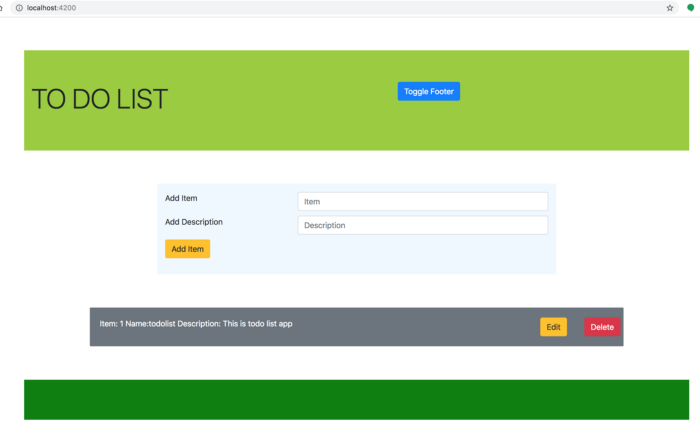
Every component/directive undergo a lifecycle as Angular create, update and destroys them over time. Let’s discuss what are those, why and when we should use those with a sample project.



**Component lifecycle hooks**

Angular provides eight interfaces to implement those as component goes from creation to destruction phases. Let’s understand these with the sample project.

This is a TODO list App where you can add, edit and delete items. This is a very good example to understand all our angular lifecycle hooks.



**TODO List App**

**Understand lifecycle methods order**

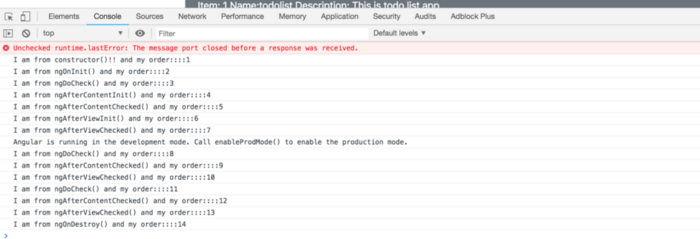
We will see the order of these methods and when and where all these methods are called with the simple component.

If we look at the above home screen, we have the footer with green color and it is a very simple component and is included in the App components' view. Let’s put all the lifecycle methods in this component and test it and we have ***Toggle Footer*** button in the header to test ngOnDestory() method.

**Without any input to component**

Let’s test the footer without any input to the component and click on Toggle footer button to test ngOnDestroy() method.

**footer.component.ts and app.component.html**

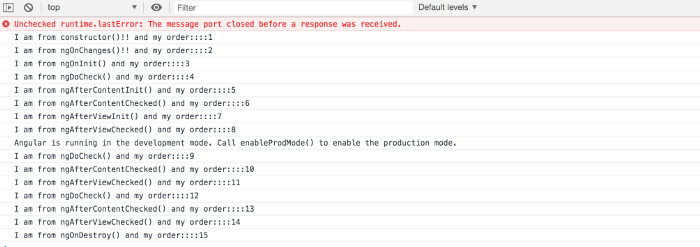


**Lifecycle methods order in Action without input to the component**

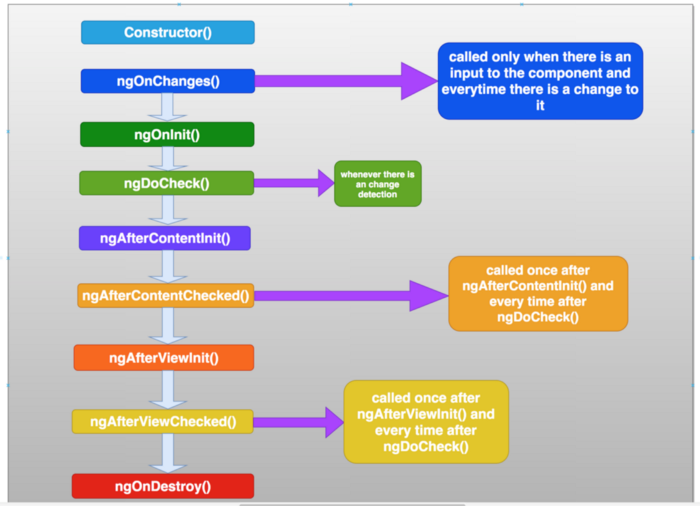
**With the input to component**

Let’s test with input ***footerText***to see the order of lifecycle methods, We can notice that ngOnChanges() method is called before ngOnInit().

**footer.component.ts with input**



**Lifecycle methods order in Action with input to the component**



**Angular Lifecycle hooks order**

**Lifecycle methods With Some Examples**

If you understand all lifecycle methods already you can skip to the summary section otherwise Let’s discuss these in detail.

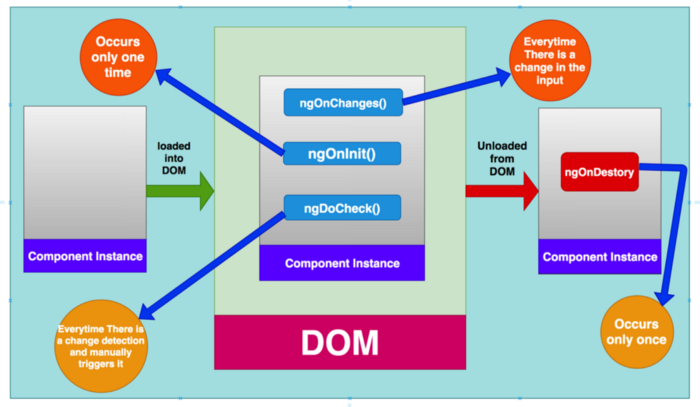
Let me categorize lifecycle methods into three. All these methods are called even without the view and content projection But, I just categorize these to show real use cases.

* ***Lifecycle methods with the basic component***
* ***Lifecycle methods with component’s view***
* ***Lifecycle methods with content projection into Component’s view***

***Lifecycle methods with the basic component***

These are the methods which occur with a component in Angular. As components load and unload from the DOM, these lifecycle methods triggers which give us the chance to do something in these key moments. Please look at the below illustration for better understanding.

* ***ngOnInit()***
* ***ngOnChanges()***
* ***ngOnDestroy()***
* ***ngDoCheck()***

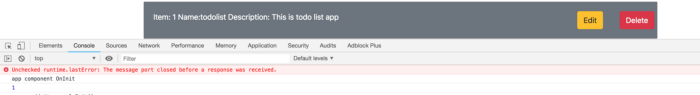


**Component lifecycle as component loads and unloads**

**ngOnInit()**

This lifecycle hook is called after the component is loaded and all the data-bound properties are set and **it is called only once when the component is loaded into the DOM.**We don’t have to implement this interface **OnInit** Angular implements it whether you implement or not.

Look at the below file from the project, we want to display default todo items whenever the app is loaded the first time so we are subscribing it to the service to load default items.



**we can see console.log statement from the ngOnInit() method**

**use cases**

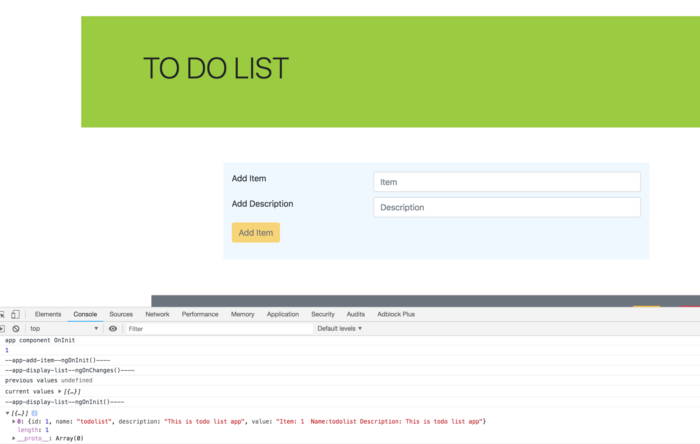
* if we want to add something once the component is loaded, this is the place like subscribing to data from the API, initialize some third party lib etc.

**ngOnChanges()**

This lifecycle hook is called before ngOnInit() and whenever there is a change in the input properties. I**t is called every time the data changes.**We have to implement this interface **OnChanges** and it takes SimpleChanges Object as an input parameter. This simplechanges object has currentValue and previousValue so that we can compare and do all sorts of things.

Look at the below file from this project, we want to add index number for each item added so that we can display number before each entry in the list. Since we need to change these before we display on the screen, **ngOnChanges()** is the best place.

**we are changing the item before it displays it on the screen in ngOnChanges**



**ngOnChanges in action**

**use cases**

* whenever there is a change in the @Input data property, we can do some more changes in this method by comparing previous and current values.

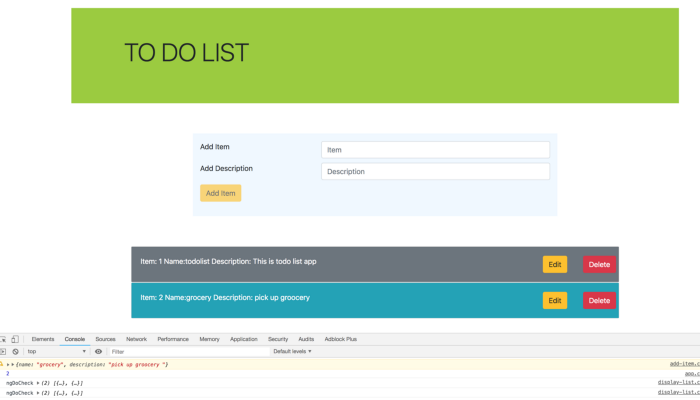
**ngDoCheck()**

There is a problem with the above code if you add an item, it won’t be added to the list. it’s because we didn’t change the reference of the array, we just changed the content of it by pushing one element each time.

In the above file, all the logic is in the ngOnChanges method and it doesn’t detect the changes and item is not displayed in the list.

**In these scenario’s we need to use ngDoCheck() because we need to trigger change detection manually.** Look at the logic in ngDoCheck, we are comparing the array and assigning it to the items array.

**added ngDoCheck() because ngOnChanges doesn’t see the input changes**



**ngDoCheck in action**

**ngOnDestroy()**

Angular cleans up all the event handlers or subscriptions in this method. This is called just before the component is destroyed. This is the best place to unsubscribe all the subscriptions and detach any event handlers to avoid memory leaks.

Here is an example where we are unsubscribing appService items observable in the AppComponent **ngOnDestroy** Method. We have added **takeUntil** operator from the rxjs library which allows subscribing until the certain condition met true.

**notice ngOnDestroy() and ngOnInit() methods**

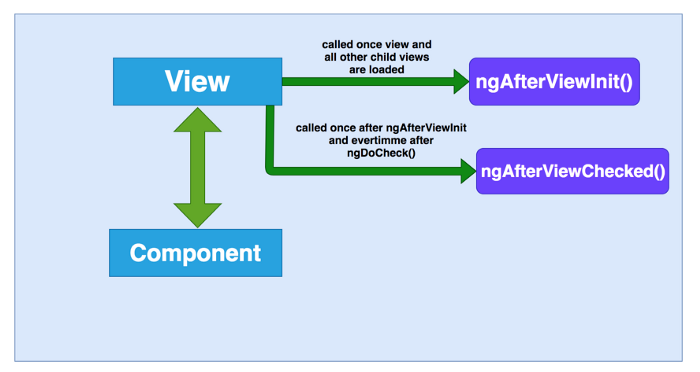
**use cases**

* The best place to clean up event handlers or any subscriptions.

**Lifecycle methods with component’s view**

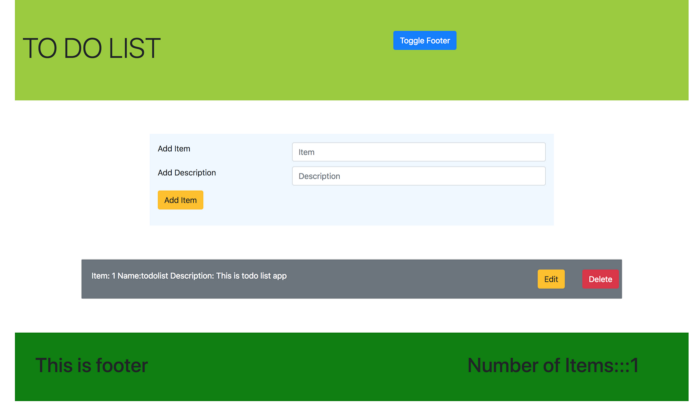
In Angular, we follow the component design pattern and every component has a view associated with it. These lifecycle methods are called once the view is loaded.

* ***ngAfterViewinit()***
* ***ngAfterViewChecked()***

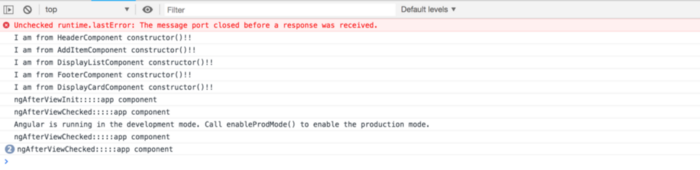


**These lifecycle methods are called once view and child views are loaded**

Let’s load the Home screen of the app and display Number of items in the footer like in the diagram and I put the console.log statements in all the component’s constructors. We can see how ngAfterViewInit() and ngAfterViewChecked() is loaded in the second figure.



**Number of items displayed in the footer**



**notice the order in which these lifecycle methods are called**

***ngAfterViewinit()***

we can initialize the number of items in the footer component with the help of ngAfterViewInit() method in app.component.ts. Look at the following code

**with the help of viewChild and setting the number of items on load**

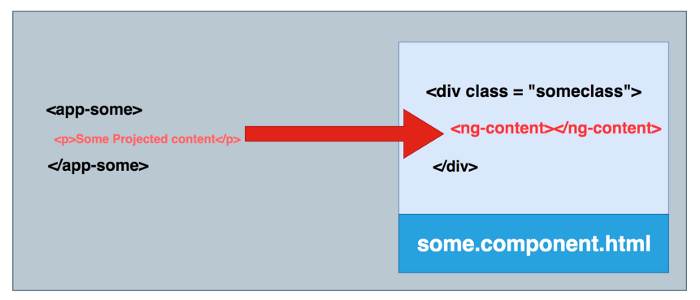
**ngAfterViewChecked()**

This is called once after ngAfterViewInit() and every time after ngDoCheck(). if possible avoid this for performance reasons.

**Lifecycle methods with content projection into Component’s view**

In Angular, There is a way we can project content into the component’s view. Look at the following illustration, whatever you put between the component tag will be projected into the component’s view in place of ***<ng-content>.***These lifecycle methods will be called once content projection is completed

* ***ngAfterContentInit()***
* ***ngAfterContentChecked()***



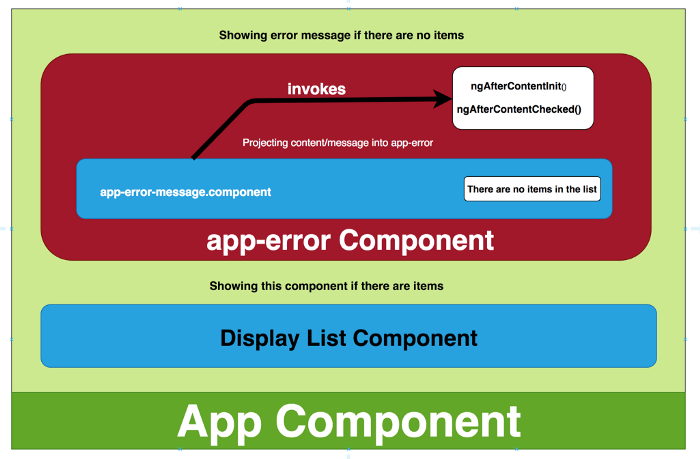
**app-some component projecting content**

**ngAfterContentInit()**

This lifecycle hook is called whenever Angular projects content into the view that this directive is in. This is called only once after ngDoCheck.

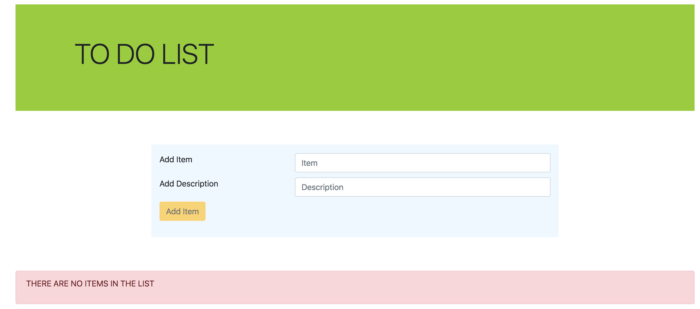
Look at the following illustration, we have implemented **AfterContentInit** and **AfterContentChecked**in app-error Component. We have defined another component app-error-message component with input message, In this way, we will have the flexibility with the messages and we are projecting and this **app-error-message** component into the **app-error** component which invokes lifecycle methods **ngAfterContentInit()** and **ngAfterContentChecked()** in the**app-error component.**

// displaying app-error when there are no items<app-error \*ngIf="itemList.length ==0">  
 <app-error-message   
 message="There are no items in the list">   
 </app-error-message>  
</app-error>



**ngAfterContentInit() and ngAfterContentChecked() illustration with error component**

So, in ngAfterContentInit we are making error message to uppercase, No matter what case the message is sent, it will appear in uppercase.



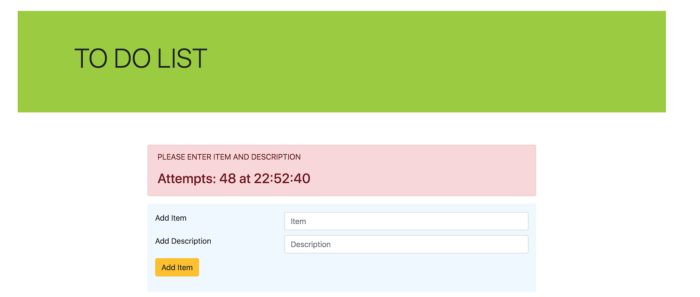
**we can see the error message in uppercase**

**ngAfterContentChecked()**

This lifecycle hook is called whenever Angular projects content into the view that this directive is in. This is called every time after ngDoCheck.

Look at the following example, we know we are using content projection for error component. what if we want to show a number of attempts and the time of the last attempt.

we can't use **ngAfterContentInit()** for it because it occurs only once.



**Summary**

* **ngOnInit():**This is called when the component is loaded into the DOM and it occurs only once.
* **ngOnChanges():**This is called when there is Input property to the component and it is called before ngOnInit() and occurs every time there is a change in the input property. It is not called if there is no input property to the component
* **ngDoCheck():**occurs every time there is a change detection and immediately after ngOnChanges() and ngOnInit().
* **ngOnDestroy():** This is called right before the component gets destroyed and occurs only once.
* **ngAfterViewInit():**This is called once after the component’s view/ child views are loaded and right after ngAfterContentChecked() and occurs only once.
* **ngAfterViewChecked():** This is called after component’s view/ child views are loaded and called after ngAfterViewInit() and every time after ngDoCheck().
* **ngAfterContentInit():**This is called once external content is projected into component’s view and right after the first ngDoCheck() and occurs only once.
* **ngAfterContentChecked():**This is called after external content projected into component’s view and called after ngAfterContentInit() and every subsequent ngDoCheck().