Angular Component Lifecycle

Every Angular component has a lifecycle. Actually, every Angular component and Angular directive have a lifecycle and the following information can be applied to both. The lifecycle is managed internally by Angular. So what is the lifecycle? According to Angular's documentation

> Angular creates it, renders it, creates and renders it's children, checks it when it's data-bound properties change, and destroys it before removing it from the DOM.
> https://angular.io/guide/lifecycle-hooks

That is a very simple description of the sequence of events that an Angular component's life experiences. These events are called "Lifecycle Hooks". Developers can use these lifecycle hooks to do something (run some code) whenever one of these events occur. There are eight different lifecycle hooks that a developer can tap into in any component or directive. To do so, a developer just needs to add one of the eight function calls that correspond to the lifecycle event. Add the hooks you need and leave out the ones you don't.

8 Lifecycle Hooks

- ngOnChanges()
  - Used in pretty much any component that has an input.
  - Called whenever an input value changes
  - Is called the first time before ngOnInit

- ngOnInit()
  - Used to initialize data in a component.
  - Called after input values are set when a component is initialized.
  - Added to every component by default by the Angular CLI.
  - Called only once
- ngDoCheck()
  - Called during all change detection runs
    - A run through the view by Angular to update/detect changes
- ngAfterContentInit()

    - Called only once after first ngDoCheck()
    - Called after the first run through of initializing content
- ngAfterContentChecked()
    - Called after every ngDoCheck()
    - Waits till after ngAfterContentInit() on first run through
- ngAfterViewInit()
    - Called after Angular initializes component and child component content.
    - Called only once after view is initialized
- ngAfterViewChecked()
    - Called after all the content is initialized and checked. (Component and child components).
    - First call is after ngAfterViewInit()
    - Called after every ngAfterContentChecked() call is completed
- ngOnDestroy()
    - Used to clean up any necessary code when a component is removed from the DOM.
        - Fairly often used to unsubscribe from things like services.
    - Called only once just before component is removed from the DOM.

In my experience as an Angular developer, I primarily use only four of these hooks. Mostly because I don't want to do something to a component **after** the content has already been checked.

- ngOnChanges()
- ngOnInit()
- ngAfterViewInit()
- ngOnDestory()

The first two I use fairly frequently. They are very useful when dealing with input values or setting your component state based on outside data. The other two are very use case specific. If for some reason you need to do something after your component content has been set, use ngAfterViewInit. As I stated above, clean up your component with ngOnDestory(). Next I am going to set up an example module to demonstrate the different hooks.

Example

I am going to add a new module to my ng-example repo. It's going to be called `lifecycle-hooks` and it has a parent component (lifecycle-hooks) and a child component for a `ngOnChanges` example called `changes-example`. I added routing to the sidebar to route to the `lifecycle-hooks` component and place the `changes-example` component inside of that component. Basically it's a parent component with a child component. Next I am going to add all of the above lifecycle hooks to both components and console.log() the name of the hook that is called. I added "child" to the child's console log statements to differentiate the two. This is what the parent component looks like.

```
export class LifecycleHooksComponent implements OnInit {
  constructor() { }
  ngOnInit() {
    console.log('ngOnInit');
  }
  ngOnChanges(){
    console.log('ngOnChanges');
  }
  ngDoCheck(){
    console.log('ngDoCheck');
  }
  ngAfterContentInit(){
    console.log('ngAfterContentInit');
  }
  ngAfterContentChecked(){
    console.log('ngAfterContentChecked')
  }
  ngAfterViewInit(){
    console.log('ngAfterViewInit');
  }
  ngAfterViewChecked(){
    console.log('ngAfterViewChecked');
  }
  ngOnDestroy(){
    console.log('ngOnDestory');
  }
}
```

Again, the child is identical but with "child" as apart of the console.log statement. So now, if we run `ng serve` we will see the order that these are fired during initialization.

ngOnInit	lifecycle-hooks.component.ts:17
ngDoCheck	lifecycle-hooks.component.ts:25
ngAfterContentInit	lifecycle-hooks.component.ts:29
ngAfterContentChecked	lifecycle-hooks.component.ts:33
child ngOnInit	changes-example.component.ts:14
child ngDoCheck	changes-example.component.ts:25
child ngAfterContentInit	changes-example.component.ts:29
child ngAfterContentChecked	changes-example.component.ts:33
child ngAfterViewInit	changes-example.component.ts:37
child ngAfterViewChecked	changes-example.component.ts:41
ngAfterViewInit	lifecycle-hooks.component.ts:37
ngAfterViewChecked	lifecycle-hooks.component.ts:41

This should give you

a decent picture of the order of operations for the lifecycle hooks on initialization. The component initializes, the content is initialized and is checked, then the child runs through its initialization and checks. Finally, the component view (including children) declares it has been initialized and has been checked. Next I just want to add a simple event to the parent component that manipulates some input value to the child. All I am going to do is add a button in the parent that increases a number by one when it is clicked. That number is going to be passed as an input to the child and displayed there. The additions to the parent .ts look like this.

```
export class LifecycleHooksComponent implements OnInit {

  num:number = 0;

  constructor() { }

  add(){
    console.log('CLICKED')
    this.num++;
  }

  ...
}
```

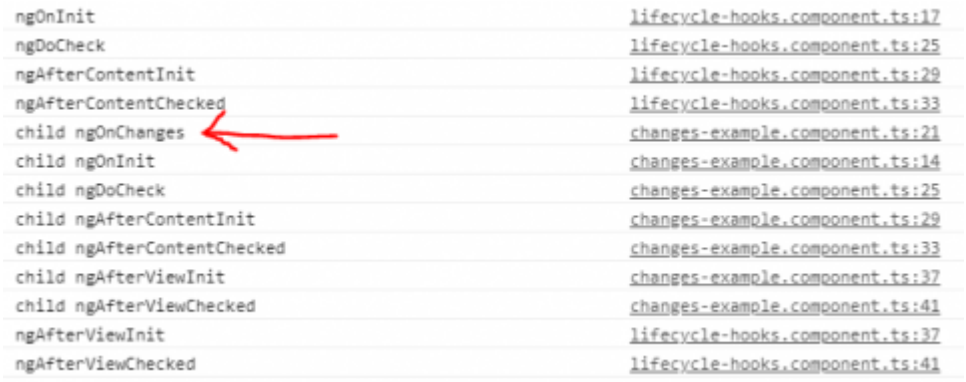
The parent html looks like this. (bootstrap classes added to make it look presentable).

```
<div class="w-100 pt-4 justify-content-center d-flex">
  <button class="btn" (click)="add()">+</button>
  <div class="pl-4">
    <app-changes-example [num]="num"></app-changes-example>
  </div>
</div>
```

And then I am just displaying the input “num” in the child html.

```
<h4>{{num}}</h4>
```

So now when we click the ‘+’ button, an event is fired and the components will experience a series of lifecycle events. Let’s see what the console looks like on initialization.



Notice now that the `ngOnChanges` event fires in the child. That's because the child now has an input value that `ngOnChanges` can detect. Now I am going to click the button and fire the event. The console will show the hooks that fire when the event occurs.

Here we can see the parent runs a check of itself, once that components direct content is check the child component does it's own check. However, the child's check runs after changes are detected and handled.

Bonus: SimpleChanges

`SimpleChanges` is a class that can be hooked into in `ngOnChanges` that will give you some context of the changes occurring to your input values. Every input value will have a few properties when they change:

- previousValue
  - The value before the change
- currentValue
  - The new value after the change
- firstChange
  - A boolean value set to `true` is it is the first change for the input value.

It's use looks something like this.

```
1   ngOnChanges(changes: SimpleChanges){
2     if(changes.num.currentValue >> changes.num.previousValue){
3       console.log('num went up from: ' + changes.num.previousValue + ' to ' + changes.num.c
4     }
```

```
5 }
```

Here “num” is the name of the input that is experiencing a change.

Conclusion

With that I have provided you an overview of what lifecycle hooks are, the order in which lifecycle events occur and a way to leverage those events. Remember, these events occur in both components and directives. Be sure to choose carefully which event is needed for what you are doing. Thanks for reading! Here is some additional Angular reading from some of my recent posts: