Component Lifecycle

A component instance has a lifecycle that starts when Angular instantiates the component class and renders the component view along with its child views. The lifecycle continues with change detection, as Angular checks to see when data-bound properties change, and updates both the view and the component instance as needed. The lifecycle ends when Angular destroys the component instance and removes its rendered template from the DOM. Directives have a similar lifecycle, as Angular creates, updates, and destroys instances in the course of execution.

Your application can use lifecycle hook methods to tap into key events in the lifecycle of a component or directive to initialize new instances, initiate change detection when needed, respond to updates during change detection, and clean up before deletion of instances.

Prerequisites

Before working with lifecycle hooks, you should have a basic understanding of the following:

- TypeScript programming
- Angular app-design fundamentals, as described in Angular Concepts

Responding to lifecycle events

Respond to events in the lifecycle of a component or directive by implementing one or more of the lifecycle hook interfaces in the Angular `core` library. The hooks give you the opportunity to act on a component or directive instance at the appropriate moment, as Angular creates, updates, or destroys that instance.

Each interface defines the prototype for a single hook method, whose name is the interface name prefixed with `ng`. For example, the OnInit interface has a hook method

named `ngOnInit()`. If you implement this method in your component or directive class, Angular calls it shortly after checking the input properties for that component or directive for the first time.

```
@Directive({selector: '[appPeekABoo]'})
export class PeekABooDirective implements OnInit {
constructor(private logger: LoggerService) { } // implement OnInit's `ngOnInit`
method ngOnInit() { this.logIt('OnInit'); } logIt(msg: string) {
this.logger.log(`#${nextId++} ${msg}`); } }
```

You don't have to implement all (or any) of the lifecycle hooks, just the ones you need.

Lifecycle event sequence

After your application instantiates a component or directive by calling its constructor, Angular calls the hook methods you have implemented at the appropriate point in the lifecycle of that instance.

Angular executes hook methods in the following sequence. Use them to perform the following kinds of operations.

HOOK METHOD	PURPOSE	TIMING
`ngOnChanges()`	Respond when Angular sets or resets data-bound input properties. The method receives a SimpleChanges object of current and previous property values. **NOTE**:	Called before `ngOnInit()` (if the component has bound inputs) and whenever one or more data-bound input properties change. **NOTE**: If your component has no inputs or you use it without providing any inputs, the

HOOK METHOD	PURPOSE	TIMING
	> This happens frequently, so any operation you perform here impacts performance significantly. See details in Using change detection hooks in this document.	framework will not call `ngOnChanges()`.
`ngOnInit()`	Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties. See details in Initializing a component or directive in this document.	Called once, after the first `ngOnChanges()`. `ngOnInit()` is still called even when `ngOnChanges()` is not (which is the case when there are no template-bound inputs).
`ngDoCheck()`	Detect and act upon changes that Angular can't or won't detect on its own. See details and example	Called immediately after `ngOnChanges()` on every change detection run, and immediately after `ngOnInit()` on the first run.

HOOK METHOD	PURPOSE	TIMING
	in Defining custom change detection in this document.	
`ngAfterContentInit()`	Respond after Angular projects external content into the component's view, or into the view that a directive is in. See details and example in Responding to changes in content in this document.	Called *once* after the first `ngDoCheck()`.
`ngAfterContentChecked()`	Respond after Angular checks the content projected into the directive or component. See details and example in Responding to projected content changes in this document.	Called after `ngAfterContentInit()` and every subsequent `ngDoCheck()`.
`ngAfterViewInit()`	Respond after Angular initializes the component's views and child views, or the view that contains the	Called *once* after the first `ngAfterContentChecked()`.

HOOK METHOD	PURPOSE	TIMING
	directive. See details and example in Responding to view changes in this document.	
`ngAfterViewChecked()`	Respond after Angular checks the component's views and child views, or the view that contains the directive.	Called after the `ngAfterViewInit()` and every subsequent `ngAfterContentChecked()`.
`ngOnDestroy()`	Cleanup just before Angular destroys the directive or component. Unsubscribe Observables and detach event handlers to avoid memory leaks. See details in Cleaning up on instance destruction in this document.	Called immediately before Angular destroys the directive or component.

Lifecycle example set

The live example / download example demonstrates the use of lifecycle hooks through a series of exercises presented as components under the control of the root `AppComponent`. In each case a parent component serves as a test rig for a child component that illustrates one or more of the lifecycle hook methods.