ANGULAR 12+

COMPLETE COURSE 2022

procademy

Angular Lifecycle Hooks

Angular Lifecycle Hook

The Angular lifecycle hooks are the methods that angular invokes on the directives and components as it creates, changes, and destroys them.

```
<app-product></app-product>
<app-product></app-product>
<app-product></app-product>
```

Angular Lifecycle Hook

1. When the angular application starts, it first creates and renders the **root** component. Then, it creates and renders its Children's & their children. It forms a tree of components

2. Once Angular loads the components, it starts the process of rendering the view. To do that, it needs to check the **input** properties, evaluate the data bindings & expressions, render the projected content etc.

3. Angular lets us know when these events happen using lifecycle hooks. For Example:
   - `ngOnInit` when Angular initializes the component for the first time.
   - When a component's input property change, Angular invokes `ngOnChanges`
   - If the component is destroyed, Angular invokes `ngOnDestroy`

What is Change detection cycle?

Change detection is the mechanism by which angular keeps the template in sync with the component.

What is Change detection cycle?

```
<div>Hello {{name}}</div>
```

1. Projected content: Projected content is that HTML content which replaces the `<ng-content>` directive in child component.

Child Component

```
<h2>Child Component</h2>
<ng-content></ng-content>
```

Parent Component

```
<h1>Parent Component</h1>
<app-child>
  <p>This content is injected from parent</p>
</app-child>
```

2. Input bound properties: These are those properties of a component class which is decorated with `@Input()` decorator.

Child Component

```
@Input() message: string
```

Constructor of a Component

1. Life Cycle of a component begins, when Angular creates the component class. First method that gets invoked is class **Constructor**.
2. **Constructor** is neither a life cycle hook nor it is specific to Angular. It is a JavaScript feature. It is a method which gets invoked, when a class is created.
3. When a constructor is called, at that point, none of the components **input** properties are updated and available to use. Neither its child components are constructed. Projected contents are also not available.
4. Once Angular instantiates the class, it kick-start the first change detection cycle of the component.

Angular Lifecycle Hook

ngOnChanges

Child Component

```
@Input() message: string
```

Parent Component

```
<app-child [message]="message">
</app-child>
```

1. It is executed right at the start, when a new component is created, and it also gets executed whenever one of the bound input property changes.
2. Angular invokes **ngOnChanges** life cycle hook whenever any data-bound input property of the component or directive changes.
3. **Input** properties are those properties, which we define using the `@Input` decorator. It is one of the ways by which a parent component communicates with the child component.
4. Initializing the Input properties is the first task that angular carries during the change detection cycle. And if it detects any change in input property, then it raises the **ngOnChanges** hook. It does so during every change detection cycle.
5. This hook is not raised if change detection does not detect any changes.

Angular Lifecycle Hook

ngOnChanges

ngOnInit

1. Angular raises the **ngOnInit** hook, after it creates the component and updates its input properties.
2. This hook is fired only once and immediately after its creation (during the first change detection).
3. This is a perfect place where you want to add any initialization logic for your component.
4. Here you have access to every input property of the component. You can use them in **http get requests** to get the data from the back-end server or run some initialization logic etc.
5. But remember that, by the time ngOnInit get's called, none of the child components or projected content are available at this point. Hence any properties we decorate with `@ViewChild`, `@ViewChildren`, `@ContentChild` & `@ContentChildren` will not be available to use.

Angular Lifecycle Hook

- ngOnChanges
- ngOnInit
- ngDoCheck

1. The Angular invokes the **ngDoCheck** hook event during every change detection cycle. This hook is invoked even if there is no change in any of the properties.
2. For example, **ngDoCheck** will run if you clicked a button on the webpage which does not change anything. But still, it's an event.
3. Angular invokes **ngDoCheck** it after the **ngOnChanges** & **ngOnInit** hooks.
4. You can use this hook to Implement a custom change detection, whenever Angular fails to detect the changes made to Input properties.
5. **ngDoCheck** is also a great method to use, when you want to execute some code on every change detection cycle.

Angular Lifecycle Hook

- ngOnChanges
- ngOnInit
- ngDoCheck
- ngAfterContentInit

1. **ngAfterContentInit** Life cycle hook is called after the Component's projected content has been fully initialized.
2. Angular also updates the properties decorated with the **@ContentChild** and **@ContentChildren** before raising this hook. This hook is also raised, even if there is no content to project.
3. The content here refers to the external content injected from the parent component via **Content Projection**.
4. The Angular Components can include the **ng-content** element, which acts as a placeholder for the content from the parent.
5. Parent injects the content between the opening & closing selector element. Angular passes this content to the child component

Child Component

```
<h2>Child Component</h2>
<ng-content></ng-content>
```

Parent Component

```
<h1>Parent Component</h1>
<app-child>
  <p>This content is injected from parent</p>
</app-child>
```

Angular Lifecycle Hook

- `ngOnChanges`
- `ngOnInit`
- `ngDoCheck`
- `ngAfterContentInit`
- `ngAfterContentChecked`

1. ngAfterContentChecked Life cycle hook is called during every change detection cycle after Angular finishes checking of component's projected content.
2. 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.
3. This hook is very similar to the `ngAfterContentInit` hook. Both are called after the external content is initialized, checked & updated.
4. Only difference is that `ngAfterContentChecked` is raised after every change detection cycle. While `ngAfterContentInit` during the first change detection cycle.
5. This is a component only hook.

Angular Lifecycle Hook

- `ngOnChanges`
- `ngOnInit`
- `ngDoCheck`
- `ngAfterContentInit`
- `ngAfterContentChecked`
- `ngAfterViewInit`

1. `ngAfterViewInit` hook is called after the Component's View & all its child views are fully initialized. Angular also updates the properties decorated with the `@ViewChild` & `@ViewChildren` properties before raising this hook.
2. The View here refers to the view template of the current component and all its child components & directives.
3. This hook is called during the first change detection cycle, where angular initializes the view for the first time
4. At this point all the lifecycle hook methods & change detection of all child components & directives are processed & Component is completely ready.
5. This is a component only hook.

Angular Lifecycle Hook

- `ngOnChanges`
- `ngOnInit`
- `ngDoCheck`
- `ngAfterContentInit`
- `ngAfterContentChecked`
- `ngAfterViewInit`
- `ngAfterViewChecked`

1. The Angular fires this hook after it checks & updates the component's views and child views. This event is fired after the `ngAfterViewInit` and after that during every change detection cycle.
2. This hook is very similar to the `ngAfterViewInit` hook. Both are called after all the child components & directives are initialized and updated.
3. Only difference is that `ngAfterViewChecked` is raised during every change detection cycle. While `ngAfterViewInit` is raised during the first change detection cycle.
4. This is a component only hook.

Angular Lifecycle Hook

- `ngOnChanges`
- `ngOnInit`
- `ngDoCheck`
- `ngAfterContentInit`
- `ngAfterContentChecked`
- `ngAfterViewInit`
- `ngAfterViewChecked`
- `ngOnDestroy`

1. If you destroy a component, for example, when you placed `ngIf` on a component, and this `ngIf` then set to `false`, at that time, `ngIf` will remove that component from the DOM, at that time, `ngOnDestroy` is called.
2. This method is the great place to do some cleanup work, because this is called right before the objects will be destroyed itself by angular.
3. This is the correct place where you would like to Unsubscribe Observables and detach event handlers to avoid memory leaks.