ContentChild and ContentChildren in Angular

Leave a Comment / 6 minutes of reading / March 9, 2023

← Renderer2
Angular Tutorial
afterviewinit, afterviewchecked, aftercontentinit, aftercontentchecked →

The ContentChild & ContentChildren are decorators, which we use to Query and get the reference to the Projected Content in the DOM. Projected content is the content that this component receives from a parent component.

The ContentChild & ContentChildren is very similar to the ViewChild & ViewChildren. We use the `ViewChild` or `ViewChildren` to Query and get the reference of any DOM element in the Component. The DOM element can be an HTML element, Child Component or directive, etc. But, We cannot use the `ViewChild` or `ViewChildren` to get the reference to the template inserted using the Content projection.

Table of Contents

Content Projection recap

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 designate a spot for the template in the template of the child component. The ng-content also allows us to create multiple slots using the selector attribute. The parent can send different content to each of those slots.

ContentChild and ContentChildren Example

To Understand how ContentChild & ContentChildren works, let us create a simple card application. The application has a CardComponent, which displays a single Card

```

import { Component} from '@angular/core';


@Component({
  selector: 'card',
  template: `

    <div class="card">
      <ng-content select="header"></ng-content>
      <ng-content select="content"></ng-content>
      <ng-content select="footer"></ng-content>
  </div>

 `,
 styles: [
   ` .card { min- width: 280px;  margin: 5px;  float:left  }
   `
 ]
})
export class CardComponent {

}

```

The component defines three ng-content slots. The slots are given names `header`, `content` & `footer`. The users of the card component can send content to any or all of these three slots.

The following code is from the Card List Component.

The Card List Component instantiates three Card Components. Sends content for header, content & footer.

Also, note that we have `#header` template reference variable on `h1` tag on `header` content. Now let us access the `h1` element using the `ContentChild` query in the Card Component.

```

import { Component } from '@angular/core';

@Component({
  selector: 'card-list',
  template: `

  <h1> Card List</h1>

      <card>
        <header><h1 #header>Angular</h1></header>
        <content>One framework. Mobile & desktop.</content>
        <footer><b>Super-powered by Google </b></footer>
      </card>

      <card>
        <header><h1 #header style="color:red;">React</h1></header>
        <content>A JavaScript library for building user interfaces</content>
        <footer><b>Facebook Open Source </b></footer>
      </card>

      <card>
        <header> <h1 #header>Typescript</h1> </header>
        <content><a href="https://www.tektutorialshub.com/typescript-tutorial/"> Typescrip
        <footer><i>Microsoft </i></footer>
      </card>

  `,
})
export class CardListComponent {

}

```

Using ContentChild and ContentChildren

Now, let us go back to the Card Component.

To use ContentChild, we need to import it first from the @angular/core.

```

import { Component, ContentChild, ContentChildren, ElementRef, Renderer2,  ViewChild } fr

```

Then use it to query the `header` from the projected content.

```

@ContentChild("header") cardContentHeader: ElementRef;

```

Here, `cardContentHeader` is the variable. We apply the `@ContentChild` decorator on that variable. The `header` is the template variable, that we want to read. Since it is applied on `h1` tag, it is going to return the `ElementRef`.

The `cardContentHeader` will not be available to use immediately. i.e because of Component lifecycle hooks. The angular initializes the component first. It then raises the ngOnChanges, ngOnInit & ngDoCheck hooks. The projected components are initialized next. And then Angular raises the `AfterContentInit` & `AfterContentChecked` hooks. Hence the `cardContentHeader` is available to use only after the `AfterContentInit` hook.

Once, we have reference to the DOM Element, we can use the `renderor2` to manipulate its styles etc.

```

ngAfterContentInit() {

    this.renderor.setStyle(this.cardContentHeader.nativeElement,"font-size","20px")

}

```

ViewChild Vs ContentChild

ViewChild or ViewChildren can access any DOM element, component, or directive. But it cannot be used to access the projected content. While ContentChild or ContentChildren can access only the projected content, but cannot be used to access any other content.

For Example, in the card component, use the ViewChild query to read the header element. You will find out that the `cardViewHeader` will be undefined

```

@ViewChild("header") cardViewHeader: ElementRef;

```

ContentChild Syntax

The `ContentChild` query returns the first matching element from the DOM and updates the component variable on which we apply it.

Syntax

The Syntax of the `ContentChild` is as shown below.

```

ContentChild(selector: string | Function | Type<any>, opts: { read?: any; static: boolean; 

```

We apply the `contentChild` decorator on a **Component Property**. It takes two arguments. A `selector` and `opts`.

selector (Query Selector): The directive type or the name used for querying

opts: has two options.

static True to resolve query results before change detection runs, false to resolve after change detection. Defaults to false.

read: Use it to read the different token from the queried elements.

Selector or Query Selector

The change detector looks for the first element that matches the selector and updates the component property with the reference to the element. If the DOM changes and a new element matches the selector, the change detector updates the component property

The query selector can be a string, a type, or a function that returns a string or type. The following selectors are supported.

- Any Component or directive class type
- A template reference variable as a string

```

//Using a Template Reference Variable
@ContentChild("header") cardContentHeader: ElementRef;

```

```

//Using component/directive as type
@ContentChild(childComponent) cardChildComponent: childComponent;

```

Static

Determines when the query is resolved. `True` is when the view is initialized **(before the first change detection**) for the first time. `False` if you want it to be resolved **after every change detection**

Read Token

Use it to read the different token from the queried elements.

For Example, consider the following projected content. The `nameInput` can be either a input element or a ngModel directive.

```

<input #nameInput [(ngModel)]="name">

```

The ContentChild in the following code, returns the `input` element as `elementRef`.

```

@ContentChild('nameInput',{static:false}) nameVar;
```

```
 
```

You can make use of the read token, to ask ContentChild to return the correct type.

For Example `read: NgModel` returns the `nameInput` as of type `NgModel`

```

@ContentChild('nameInput',{static:false, read: NgModel}) nameVarAsNgModel;
@ContentChild('nameInput',{static:false, read: ElementRef}) nameVarAsElementRef;
@ContentChild('nameInput', {static:false, read: ViewContainerRef }) nameVarAsViewContai


```

ContentChildren

Use the `ContentChildren` decorator to get the **list of element references** from the projected content.

`ContentChildren` is different from the `ContentChild`. `ContentChild` always returns the reference to a single element. If there are multiple elements the `ContentChild` returns the first matching element,

`ContentChildren` always returns all the matching elements as a QueryList. You can iterate through the list and access each element.

Syntax

The Syntax of the contentChildren is as shown below. It is very much similar to the syntax of contentChild . It does not have the static option but has the descendants option

```

ContentChildren(selector: string | Function | Type<any>, opts: {descendants?:boolean, rea

```

Make descendants True to include all descendants, otherwise include only direct children.

The ContentChildren is always resolved after the change detection is run. i.e why it does not have static option. And also you cannot refer to it in the ngOnInit hook as it is yet to initialize.

Source Code

1. Source Code of Content Child
2. Example of Read Token

Reference

1. ContentChild
2. ContentChildren API

Read More

1. **Angular Tutorial**
2. **Typescript Tutorial**