ElementRef in Angular

2 Comments / 4 minutes of reading / March 9, 2023

← Viewchild, viewchildren & querylist

Angular Tutorial

Renderer2 →

Angular ElementRef is a wrapper around a native DOM element (HTML element) object. It contains the property `nativeElement`, which holds the reference to the underlying DOM object. We can use it to manipulate the DOM. We use the ViewChild to get the ElementRef of an HTML element in the component class. Angular also inject `ElementRef` of the Host element of the component or directive when you request for it in the constructor. In this tutorial, let us explore how to use `ElementRef` to get the reference of an HtmlElement & manipulate the DOM in Angular Applications.

Table of Contents

ElementRef

The DOM objects are created and maintained by the Browser. They represent the structure and content of the Document. In a Vanilla JavaScript code, we access these DOM objects to manipulate the View. We can create and build documents, navigate their structure, and add, modify, or delete elements and content.

Angular provides a lot of tools & techniques to manipulate the DOM. We can add/remove components. It provides a lot of directives like Class Directive or Style directive. to Manipulate their styles etc.

We may still need to access the DOM element on some occasions. This is where the ElementRef comes into the picture.

Getting ElementRef in Component Class

To manipulate the DOM using the `ElementRef`, we need to get the reference to the DOM element in the component/directive.

To get the reference to DOM elements in the component

1. Create a template reference variable for the element in the component/directive.
2. Use the template variable to inject the element into component class using the ViewChild or ViewChildren.

To get the DOM element hosting the component/directive

1. Ask for `ElementRef` in the constructor (Angular Dependency injection), the Angular will inject the reference element hosting the component/directive.

For Example, in the following code, the variable hello refers to the HTML element `div`.

```

<div #hello>Hello Angular</div>

```

hello is the Template Reference variable, which we can use in the Template.

In the Component class, we use the ViewChild to inject the hello element. The Angular injects the hello as a type of ElementRef.

```

@ViewChild('hello', { static: false }) divHello: ElementRef;

```

Read token

Consider the following example

```

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

```

The nameInput Template Reference Variable now binds to the input element. But at the same time, we have ngModel directive applied to it.

Under such circumstance, we can use the read token to let angular know that we need ElementRef reference as shown below

```

//ViewChild returns ElementRef i.e. input HTML Element

@ViewChild('nameInput',{static:false, read: ElementRef}) elRef;


//ViewChild returns NgModel associated with the nameInput
@ViewChild('nameInput',{static:false, read: NgModel}) inRef;
```

```

```

ElementRef Example

Once we have the ElementRef , we can use the nativeElement property to manipulate the DOM as shown below.

We need to wait for Angular to Initializes the View, before accessing the ViewChild variables. Hence we wait until the AfterViewInit life cycle hook, to start making use of the variable.

```

import { Component,ElementRef, ViewChild, AfterViewInit } from '@angular/core';

@Component({
  selector: 'app-root',
  template:  '<div #hello>Hello</div>'
  styleUrls: ['./app.component.css']
})
export class AppComponent implements AfterViewInit {

 @ViewChild('hello', { static: false }) divHello: ElementRef;

 ngAfterViewInit() {
   this.divHello.nativeElement.innerHTML = "Hello Angular";
 }

}

```

You can manipulate the DOM very easily.

```

 ngAfterViewInit() {
   this.divHello.nativeElement.innerHTML = "Hello Angular";
   this.divHello.nativeElement.className="someClass";
   this.divHello.nativeElement.style.backgroundColor="red";
```

```
}


```

ElementRef in Custom Directive

One of the use case for `ElementRef` is the Angular directive. We learned how to create a custom directive in Angular. The following is the code for the `ttClass` custom attribute directive.

```

import { Directive, ElementRef, Input, OnInit } from '@angular/core'

@Directive({
  selector: '[ttClass]',
})
export class ttClassDirective implements OnInit {

  @Input() ttClass: string;

  constructor(private el: ElementRef) {
  }

  ngOnInit() {
    this.el.nativeElement.classList.add(this.ttClass);
  }

}

```

Note that we are injecting the `ElementRef` in the constructor. Whenever we ask for the `ElementRef` in the constructor, the Angular will inject the reference to the host DOM element of the directive.

Use with caution

[From the Angular Documents]

Use this API as the last resort when direct access to DOM is needed. Use templating and data-binding provided by Angular instead. Alternatively, you can take a look at Renderer2 which provides API that can safely be used even when direct access to native elements is not supported.

Relying on direct DOM access creates tight coupling between your application and rendering layers which will make it impossible to separate the two and deploy your application into a web worker.

ElementRef & XSS Injection Attack

Improper use of ElementRef can result in an XSS Injection attack. For Example in the following code, we are injecting a script using the elementRef. When the component containing such code runs, the script is executed

```

constructor(private elementRef: ElementRef) {
   const s = document.createElement('script');
   s.type = 'text/javascript';
   s.textContent = 'alert("Hello World")';
   this.elementRef.nativeElement.appendChild(s);
 }

```

Reference

1. ElementRef

←
Viewchild, viewchildren & querylist

Angular Tutorial

Renderer2 →