Renderer2 Example: Manipulating DOM in Angular

3 Comments / 8 minutes of reading / March 9, 2023

← ElementRef
Angular Tutorial
ContentChild & ContentChildren →

The Renderer2 allows us to manipulate the DOM elements, without accessing the DOM directly. It provides a layer of abstraction between the DOM element and the component code. Using Renderer2 we can create an element, add a text node to it, append child element using the appendchild method., etc. We can also add or remove styles, HTML attributes, CSS Classes & properties, etc. We can also attach and listen to events etc.

Table of Contents

Why not ElementRef?

We can use the nativeElement property of the ElelemtRef to manipulate the DOM. We learned this in our last tutorial on ElementRef. The nativeElement Property contains the reference to the underlying DOM object. This gives us direct access to the DOM, bypassing the Angular. There is nothing wrong with using it, but it is not advisable for the following reasons.

1. Angular keeps the Component & the view in Sync using Templates, data binding & change detection, etc. All of them are bypassed when we update the DOM Directly.
2. DOM Manipulation works only in Browser. You will not able to use the App in other platforms like in a web worker, in Server (Server-side rendering), or in a Desktop, or in the mobile app, etc where there is no browser.
3. The DOM APIs do not sanitize the data. Hence it is possible to inject a script, thereby, opening our app an easy target for the XSS injection attack.

Using Renderer2

First import it from the @angular/core

```

import {Component, Renderer2, ElementRef, ViewChild, AfterViewInit } from '@angular/core

```

inject it into the component

```

constructor(private renderer:Renderer2) {
}

```

Use ElementRef & ViewChild to get the reference to the DOM element, which you want to manipulate.

```

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

```

Use the methods like setProperty , setStyle etc to change the property, styles of the element as shown below.

```

this.renderer.setProperty(this.divHello.nativeElement,'innerHTML',"Hello Angular")

this.renderer.setStyle(this.divHello.nativeElement, 'color', 'red');

```

Code Example

Setting & Removing Styles (setStyle & removeStyle)

Use the setStyle & RemoveStyle to add or remove the styles. It accepts four argument.

The first argument is the element to which we want to apply the style. name of the styles is the second argument. The value of the style is the third argument. The last argument is Flags for style variations

```
abstract setStyle(el: any, style: string, value: any, flags?: RendererStyleFlags2): void

abstract removeStyle(el: any, style: string, flags?: RendererStyleFlags2): void

```

Example

```

//Template

<div #hello>Hello !</div>


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


setStyle() {
  this.renderer.setStyle(this.divHello.nativeElement, 'color', 'blue');
}


removeStyle() {
  this.renderer.removeStyle(this.divHello.nativeElement, 'color');
}

```

Use the last option RendererStyleFlags2 to add the `!important` or to make use of `DashCase`

Adding / Removing CSS Classes (addClass & removeClass)

Use the methods `addClass` / `removeClass` to add or remove classes.

Syntax

```
abstract addClass(el: any, name: string): void


abstract removeClass(el: any, name: string): void
```

Example

```
//Template

<div #hello>Hello !</div>



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

addClass() {
  this.renderer.addClass(this.divHello.nativeElement, 'blackborder' );
}

removeClass() {
  this.renderer.removeClass(this.divHello.nativeElement, 'blackborder');
}
```

Setting or Remove Attributes (setAttribute & removeAttribute)

We can add remove attributes using the `setAttribute` & `removeAttribute`

```

setAttribute(el: any, name: string, value: string, namespace?: string): void

removeAttribute(el: any, name: string, namespace?: string): void

```

Example

```

//Template

<h2>Add/ Remove Attributes </h2>
<input #inputElement type='text'>
<button (click)="addAttribute()">Set Attribute</button>
<button (click)="removeAttribute()">Remove Attribute</button>



//Component

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


addAttribute() {
  this.renderer.setAttribute(this.inputElement.nativeElement, 'value', 'name' );
}


removeAttribute() {
  this.renderer.removeAttribute(this.inputElement.nativeElement, 'value');
}


```

Setting Property (setProperty)

Set any property of a DOM element using the `setProperty` method.

```

setProperty(el: any, name: string, value: any): void

```

```

setProperty() {
  this.renderer.setProperty(this.divHello.nativeElement,'innerHTML',"Hello Angular")
}

```

AppendChild

Use the `appendChild` to append a new element (child element) to any existing element (parent element).

```

appendChild(parent: any, newChild: any): void

```

It accepts two arguments. The first argument is the parent node, where we want to append a new child node. The second argument is the child node, which you want to add.

The next two examples, shows how to use `appendChild`.

Insert Text Element (CreateText & appendChild)

`CreateText` allow us to add text to the DOM.

Example

The following template has a empty `div` (`#divCreateText`)

```
//Template

<h2>Create Text Example</h2>
<div #divCreateText> </div>
<button (click)="createText()">Create Text</button>

```

Use the ViewChild to inject the reference to the `divCreateText` in the component.

```
@ViewChild('divCreateText', { static: false }) divCreateText: ElementRef;

```

Use the `createText` method the create text node. At this point it is not added to the DOM.

```
const text = this.renderer.createText('Example of Create Text');

```

Use the `appendChild` method to add it to an existing element (`divCreateText`).

```

this.renderer.appendChild(this.divCreateText.nativeElement, text);

```

Creating new Element (createElement & appendChild)

We can easily create a new element using the `createElement` & `appendChild`.

`createElement` creates a new element, but does not insert it into the DOM. To insert into the DOM, we need to add it to an element, which already exists in the DOM using `appendChild` method.

The following example shows how to create a new element and append to the DOM.

First, we inject ElementRef in the constructor. This will inject the DOM element hosting the component/directive.

```

  constructor(private el: ElementRef,
            private renderer:Renderer2) {
  }

```

Create a new `div` element using the method `createElement('div')`. It is not yet added to the DOM.

```

const div = this.renderer.createElement('div');

```

Make use of the `createText('Inserted at bottom')` to create a new text node.

```

const text = this.renderer.createText('Inserted at bottom');

```

Use `appendChild` to append the newly created text node to the `div` element. Note that `div` is not yet added to the DOM.

```

this.renderer.appendChild(div, text);

```

Finally, we add the `div` element to an existing DOM element i.e. host element.

```

this.renderer.appendChild(this.div.nativeElement, div);

```

The complete code as under. The `createElement2` appends the new child node a another `div`.

```

import { Component, Renderer2, OnInit, ElementRef, ViewChild, AfterViewInit, VERSION }

@Component({
  selector: 'app-create-element',
  templateUrl: './create-element.component.html',
  styleUrls: ['./create-element.component.css']
})
export class CreateElementComponent  {


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


  constructor(private el: ElementRef,
              private renderer:Renderer2) {
  }


  createElement() {
    const div = this.renderer.createElement('div');
    const text = this.renderer.createText('Inserted at bottom');

    this.renderer.appendChild(div, text);
    this.renderer.appendChild(this.el.nativeElement, div);
  }


  createElement2() {
    const div = this.renderer.createElement('div');
    const text = this.renderer.createText('Inserted inside div');

    this.renderer.appendChild(div, text);
    this.renderer.appendChild(this.div.nativeElement, div);
  }

}

```

```

<h2>Renderer2 Create Element</h2>

<div #div style="border: 1px solid black;">
  This is a div
</div>

<button (click)="createElement()">Create Element</button>
<button (click)="createElement2()">Create Element</button>

```

InsertBefore

We can also insert the new element, before an element in the DOM element using the `insertBefore` method. The syntax of `insertBefore` as shown below.

```

insertBefore(parent: any, newChild: any, refChild: any): void

```

`parent` is the parent node. `newChild` is the new node, which you want to insert. `refChild` is the existing child node before which `newChild` is inserted.

The following Example inserts a new element before `div1`.

```

<h1>Angular Renderer2 InsertBefore Example</h1>


<div #div1>
  This is div 1
</div>

<div #div2>
  This is div 2

  <div #div3>
    This is div 3
  </div>

</div>



<button (click)="insertBeforeDiv1()" >Insert Before Div1</button>

<button (click)="insertBeforeDiv2()" >Insert Before Div2</button>

<button (click)="insertBeforeDiv3()" >Insert Before Div3</button>

```

First, use the ViewChild to get the reference to the div1 . Inject ElementRef , which gives us the reference to the Host DOM element.

Create a new div element using createElement . Add a text node to it using createText and append it to the div

Use the insertBefore method to add the div element

```

import { Component, OnInit, ViewChild, ElementRef, Renderer2 } from '@angular/core';

@Component({
  selector: 'app-insert-before',
  templateUrl: './insert-before.component.html',
  styleUrls: ['./insert-before.component.css']
})
export class InsertBeforeComponent {

  @ViewChild('div1', { static: false }) div1: ElementRef;
  @ViewChild('div2', { static: false }) div2: ElementRef;
  @ViewChild('div3', { static: false }) div3: ElementRef;


  constructor(private renderer:Renderer2, private el:ElementRef) { }



  insertBeforeDiv1() {
    const div = this.renderer.createElement('div');
    const text = this.renderer.createText('This Text is Inserted before the div1');
    this.renderer.appendChild(div, text);

    this.renderer.insertBefore(this.el.nativeElement,div,this.div1.nativeElement);
  }



  insertBeforeDiv2() {
    const div = this.renderer.createElement('div');
    const text = this.renderer.createText('This Text is Inserted before the div2');
    this.renderer.appendChild(div, text);

    this.renderer.insertBefore(this.el.nativeElement,div,this.div2.nativeElement);
  }




  insertBeforeDiv3() {
    const div = this.renderer.createElement('div');
    const text = this.renderer.createText('This Text is Inserted before the div3');
    this.renderer.appendChild(div, text);

    //Using parentNode to retrieve the Parent Node
    this.renderer.insertBefore( this.renderer.parentNode(this.div3.nativeElement),div,this.c
  }


}

```

Insert Comment

createComment creates comment node. It accepts comment as the argument. You can then use the appendChild or insertBefore to insert it anywhere in the DOM.

```

createComment(value: string): any

```

ParentNode & NextSibling

ParentNode method returns the parent of a given node in the host element's DOM.

```

//Returns the parent Node of div3
this.renderer.parentNode(this.div3.nativeElement);

```

nextSibling method returns the next sibling node of a given node in the host element's DOM.

```

//Returns the next Sibling node of div2
this.renderer.nextSibling(this.div2.nativeElement);

```

SelectRootElement

We can also use the `selectRoomElement` to select a node element based on a selector.

Syntax

```

selectRootElement(selectorOrNode: any, preserveContent?: boolean)

```

The first argument is the selector or node. The Renderer2 uses the selector to search for the DOM element and returns it.

The second argument is `preserveContent`. If `no` or `undefined`, the renderer2 will remove all the child nodes. If yes the child nodes are not removed.

Example, consider the following template

```

```

```
<h1>Renderer2 selectRootElement Example</h1>

<div class="outerDiv" style="border: 1px solid black; padding :5px;">

  <div class="div1" style="border: 1px solid black; margin :5px;">This is Div1</div>
  <div class="div2" style="border: 1px solid black; margin :5px;">This is Div2</div>
  <div class="div3" class="div3class" style="border: 1px solid black; margin :5px;">This

</div>


```

The selectRootElement in the following example returns the element div1 , but it removes all the content it holds. Because the second argument is false. Change false to true , then the renderer2 will not remove the content.

```

exampleDiv1() {
   const e = this.renderer.selectRootElement('.div1',false);
}

```

Examples.

```

  exampleDiv2() {
    //Conent is always replaced. becuase preserveContent is false
    const e = this.renderer.selectRootElement('.div2',false);
    const t = this.renderer.createText('Content added to div2');
    this.renderer.appendChild(e, t);

  }

  exampleDiv3() {
    //Conent is always appended. becuase preserveContent is true
    const e = this.renderer.selectRootElement('.div3',true);
    const t = this.renderer.createText('Content added to div3');
    this.renderer.appendChild(e, t);
  }

```

Listen to DOM events

You can also, listen to DOM events using the `listen` method.

The listen method accepts three arguments. the first argument is the DOM element (`target`). The second argument is the name of the event (`eventName`) and the third argument is the `callback`

```

abstract listen(target: any, eventName: string, callback: (event: any) => boolean | void):

```

In the following example, we listen to the `click` event of a button.

```

//Component

import { Component, OnInit, ViewChild, ElementRef, Renderer2, AfterViewInit } from '@an

@Component({
  selector: 'app-listen-events',
  templateUrl: './listen-events.component.html',
  styleUrls: ['./listen-events.component.css']
})
export class ListenEventsComponent implements AfterViewInit {

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

  Count=0
  clicklistener;

  constructor(private renderer:Renderer2) { }

  ngAfterViewInit() {

    this.clicklistener = this.renderer.listen(this.divHello.nativeElement, 'click', (evt) => {
     this.Count++;
    });

  }

  ngOnDestroy() {
    this.clicklistener.unsubscribe()
  }

}

```

Do not forget to unsubscribe to the `this.clicklistener.unsubscribe()`

```

//Template

<h1>Renderer2 Listen Events Example</h1>


<button #hello>hello</button>

Click Count {{Count}}

```

Reference

1. Renderer2

Read More

1. ViewChild
2. ElementRef
3. Server Side Rendering

← ElementRef Angular Tutorial ContentChild & ContentChildren →