AngularRendererStudyNotes:

Manipulating DOM in Angular the right way

he 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.

Why not ElementRef:

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](https://www.tektutorialshub.com/angular/server-side-rendering-using-angular-universal/)), 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:

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

constructor(private renderer:Renderer2) {

}

Example 1: (setting & removing styles setStyle & removeStyle)

*//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');

}

Example 2: add & remove css classes ( addClass & removeClass )

*//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');

}

Example 3: set or remove attributes ( setAttribute & removeAttribute )

*//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');

}

################### End of Example 3 ###############

Example 4:

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

setProperty() {

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

}

Modifying DOM elements:

Example 1: CreateText allow us to add text to the DOM.

*//Template*

 <h2>Create Text Example</h2>

<div #divCreateText> </div>

<button (click)="createText()">Create Text</button>

//component

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

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

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

Example 2: create a new element

//template

<h2>Renderer2 Create Element</h2>

<div #mydiv style="border: 1px solid black;">

**This** **is** a div

</div>

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

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

//component

@ViewChild('mydiv', { static: **false** }) mydiv?: ElementRef;

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);

  }

## Example 3: InsertBefore

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

//template

<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>

//component

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**.div3.nativeElement);

  }

}

## Example 4: ParentNode & NextSibling

*//Returns the parent Node of div3*

**this**.renderer.parentNode(**this**.div3.nativeElement);

*//Returns the next Sibling node of div2*

**this**.renderer.nextSibling(**this**.div2.nativeElement);

// select a node based on a selector

selectRootElement(selectorOrNode: any, preserveContent?: boolean)

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.

//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** **is** Div3</div>

</div>

//component

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);

   }

Example 5. Listen to DOM events

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**): () => **void**

//html template

<h1>Renderer2 Listen Events Example</h1>

 <button #hello>hello</button>

 Click Count {{Count}}

//component

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

@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() // remember to unsubscribe

  }

}