Angular DOM Manipulations

Bad Practices

❌ JS methods

❌JQuery

DOM Queries

Angular has provided two ways to query/access various reference types within a Component/Directive. These are

- ViewChild/ViewChildren
- ContentChild/ContentChildren

A selector can be a string representing a template reference variable, or a Component/Directive class, or a TemplateRef or a provider defined in the child component tree.

```
@ViewChild("myElem") template: ElementRef;
```

The second parameter is optional and is only required to query some reference types which can't be inferred easily by Angular like `ViewContainerRef`.

```
@ViewChild("myContainer", {read: ViewContainerRef}) container:
ViewContainerRef;
```

DOM access via ElementRef

`ElementRef` is a very basic abstraction layer on a DOM element in Angular. It's an angular wrapper around the native element.

You can get hold of ElementRef in a Component or Directive in following ways:

Dependency Injection

Host element of a Component or Directive can be accessed via direct DI in the constructor.

```
@Component({
  selector: 'app-test',
  template: '<div>I am a test component</div>'
})
export class TestComponent implements OnInit {

  constructor(private element: ElementRef) { }

  ngOnInit() {
    console.log(this.element.nativeElement);
  }

}
```

```
/*
* Output:
*   <app-test>
*     <div>I am a test component</div>
*   </app-test>
*
*/
```

Using ViewChild and Template Reference Variables

```
@Component({
  selector: 'app-test',
  template: `
    <div #child1>First Child</div>
    <div>Second Child</div>
  `
})
export class TestComponent implements OnInit {

  @ViewChild("child1") firstChild: ElementRef;

  constructor() { }

  ngOnInit() {
    console.log(this.firstChild.nativeElement);
  }

}

/*
* Output: <div>First Child</div>
```

```
* */
```

Host Views

Host Views are quite similar to Embedded View. The only difference is that the Host Views are created from components instead of templates.

Creating a host view

In order to create a host view, first you need to create a `ComponentFactory` of the component you want to render using `ComponentFactoryResolver`.

```
constructor(
  private componentFactoryResolver: ComponentFactoryResolver
) {
  this.someComponentFactory =
this.componentFactoryResolver.resolveComponentFactory(SomeComponent
);
}
```

Then, a dynamic instance of the component is created by passing an `Injector` instance to the factory. Every component should be bound to an instance of `Injector`. You can use the injector of the parent component for the dynamically created components.

```
const componentRef =
this.someComponentFactory.create(this.injector);
const viewRef = componentRef.hostView;
```

Rendering a host view

Rendering a host view is almost similar to rendering an embedded view. You can directly insert it into a view container.

```
@Component({
  selector: 'app-test-component',
  template: `
    <div class="header">I am a header</div>
    <div class="body">
      <ng-container #container></ng-container>
    </div>
    <div class="footer">I am a footer</div>
  `,
})
export class TestComponentComponent implements AfterContentInit {

  @ViewChild("container", {read: ViewContainerRef}) container:
ViewContainerRef;

  private someComponentFactory: ComponentFactory<SomeComponent>;

  constructor(
    private componentFactoryResolver: ComponentFactoryResolver,
    private injector: Injector
  ) {
    this.someComponentFactory =
this.componentFactoryResolver.resolveComponentFactory(SomeComponent
);
  }

  ngAfterContentInit(): void {
    const componentRef =
this.someComponentFactory.create(this.injector);
    const viewRef = componentRef.hostView;
```

```
      this.container.insert(viewRef);
    }
  }
```

Or by directly calling the `createComponent` method of `ViewContainerRef` and passing the component factory instance.

```
this.container.createComponent(this.someComponentFactory);
```

```
▼<app-test-component _ngcontent-c0 _nghost-c1>
   <div _ngcontent-c1 class="header">I am a header</div>
  ▼<div _ngcontent-c1 class="body">
     <!---->
     <app-some-component _nghost-c2>Hi, I am a view created from a component.</app-some-component>
   </div>
   <div _ngcontent-c1 class="footer">I am a footer</div>
 </app-test-component>
```

Now, similar to embedded view, we can also shift the whole logic of host view creation in template itself using `ngComponentOutlet`.

```
@Component({
  selector: 'app-test-component',
  template: `
    <div class="header">I am a header</div>
    <div class="body">
      <ng-container [ngComponentOutlet]="comp"></ng-container>
    </div>
    <div class="footer">I am a footer</div>
  `
})
export class TestComponent {
  comp = SomeComponent;
}
```

Don’t forget to store the reference of the component class in parent component’s field. The template has access only to the fields of the components.

Summary

Here we come to an end. Let’s conclude what we have understood till now.

- We can access the DOM in Angular using different reference types like`ElementRef`, `TemplateRef`, `ViewRef`, `ComponentRef` and `ViewContainerRef`.
- These reference types can be queried from templates using `@ViewChild` and `@ContentChild`.
- Browser’s native DOM element can be accessed via `ElementRef`. However, manipulating this element directly is discouraged because of security reasons.
- Concept of Views.
- How to create and render an Embedded View.

- How to create and render a Component View.