Angular 8 : Online Class

08-September-2019

By: Sahosoft Solutions

Presented by : Chandan Kumar

Angular 8 : Online Class

<ng-template> in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

<ng-template>

<ng-template> is an angular element for rendering HTML. It is never displayed directly. It can be displayed using structural directive that we use all the time: ngIf, ngFor and ngSwitch.

Before rendering HTML, angular replaces <ng-template> with a comment. <ng-template> can be used with structural directive, ViewContainerRef etc.

If you put some HTML inside of an <ng-template>tag, it not only won't be on the screen, but it won't be in the DOM either. Angular will replace the <ng-template> tag and its contents with a comment. The key is that <ng-template> will only be displayed if used in partnership with a structural directive.

A rendered <ng-template> doesn't itself get turned into a DOM element, only the contents are rendered to the DOM.

<ng-template>

- ✓ ***<ng-template>*** is a virtual element and its contents are displayed only when needed (based on conditions).
- ✓ ***<ng-template>*** should be used along with structural directives like **[nglf],[ngFor],[NgSwitch]** or custom structural directives.
- ✓ ***<ng-template>*** never meant to be used like other HTML elements. It's an internal implementation of Angular's structural directives.
- ✓ When you use a structural directive in Angular we will add a prefix asterisk(*) before the directive name. This asterisk is short hand notation for ***<ng-template>***.
- ✓ Whenever Angular encounter with the asterisk(*) symbol, we are informing Angular saying that it is a structural directive and Angular will convert directive attribute to ***<ng-template>*** element.
- ✓ ***<ng-template>*** is not exactly a true web element. When we compile our code, we will not see a *<ng-template>* tag in HTML DOM.
- ✓ Angular will evaluate the ***<ng-template>*** element to convert it into a comment section in HTML DOM.

<ng-template>

We will see below example one by one

- ✓ Use <ng-template> in Angular
- ✓ Using <ng-template> with *ngIf example
- ✓ Using <ng-template> with *ngFor example
- ✓ Using <ng-template> with NgSwitch example

Angular 8 : Online Class

<ng-container> in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

<ng-container>

ng-container is an element that’s available in Angular 2+

<ng-container> is a logical container that can be used to group nodes but is not rendered in the DOM tree as a node. <ng-container> is rendered as an HTML comment.

In order to avoid having to create that extra div, we can instead use ng-container directive.

<ng-container> is an Angular grouping element that is similar to <ng- template> in that it doesn’t represent a DOM element. The difference is that it will always be rendered, whereas an <ng-template> will only be rendered if it is explicitly requested. <ng-container> are useful anywhere you need an extra container for some template elements, but don’t want to (or can’t) create a container such as a div to hold them with due to syntax or style constraints.

<ng-container>

For example,

It is not allowed in Angular to put two structural directives on the same element.
If you needed to loop through an array and display a <tr> for each element, but only if a different condition was met, you may want to put both an *ngFor and *ngIf on the <tr> element. Angular does not allow this, however, and wrapping the <tr> in a to hold one of the structural directives is not valid HTML. The utility of <ng-container> shines here, where we can use the <ng-container>to hold a structural directive and contain the <tr> without breaking the HTML layout.

Angular 8 : Online Class

ng-content & Content Projection in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

ng-content

When we want to add dynamic content in a fixed view template, we should think Transclusion.

Transclusion

In Angular 1.0, there is a concept of Transclusion. Actually, transclusion in an Angular 1.x represents the content replacement such as a text node or HTML and injecting it into a template at a specific entry time. The same thing in Angular 4.0 is totally forbidden. This is now done in Angular 4.0 through modern web APIs such as shadow DOM which is known as content projection.

Content projection

Content projection allows you to insert a shadow DOM in your component. To put it simply, if you want to insert HTML elements or other components in a component, then you do that using the concept of content projection. In Angular, you achieve content projection using **< ng-content >< /ng-content >.** You can make reusable components and scalable applications by properly using content projection.

ng-content

Transclusion

Using the @Input() decorator, you can pass a simple string to the Componnet, but what if you need to pass different types of data to the Component such as:

1. Inner HTML
2. HTML Elements
3. Styled HTML
4. Another Component, etc.

To pass or project styled HTML or another component, content projection is used. Let us modify the GreetComponent to use content projection in this code:

ng-content

Transclusion

You can select a particular slot for projection using the <**ng-content**> selector. There are four types of selectors:

1. Project using tag selector.
2. Project using class selector.
3. Project using id selector.
4. Project using attribute selector.

Angular 8 : Online Class

@ViewChild() in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

@ViewChild()

When building an application with Angular you may come across the following annotations:

1. @ViewChild
2. @ViewChildren
3. @ContentChild
4. @ContentChildren

If you want to access following inside the Parent Component, use @ViewChild decorator of Angular.

1. Child Component
2. Directive
3. DOM Element

@ViewChild() can be used for component communication. A component will get instance of another component inside it using @ViewChild(). In this way parent component will be able to access the properties and methods of child component. The child component selector will be used in parent component HTML template.

@ViewChild()

@ViewChild() using Component

ViewChild are used for component communication in Angular. Therefore, if a parent component wants access of child component then it uses ViewChild. Any component, directive, or element which is part of a template is ViewChild.

@ViewChild() decorator can be used to get the first element or the directive matching the selector from the view DOM.

@ViewChild() provides the instance of another component or directive in a parent component and then parent component can access the methods and properties of that component or directive.

In a parent component we can use @ViewChild() for components, directives and template reference variable with ElementRef or TemplateRef.

To use @ViewChild() we need to pass child component name or directive name or template variable as an argument.

@ViewChild()

@ViewChild() using Directive

@ViewChild() can instantiate a directive within a component and in this way the component will be able to access the directive methods.

@ViewChild() provides the instance of another component or directive in a parent component and then parent component can access the methods and properties of that component or directive.

ViewChild are used for component communication in Angular. Therefore, if a parent component wants access of child component then it uses ViewChild.

@ViewChild()

@ViewChild() using Template Variable

@ViewChild() can instantiate ElementRef corresponding to a given template reference variable. The template variable name will be passed in @ViewChild() as an argument.

In this way component will be able to change the appearance and behavior of element of given template variable.

@ViewChild() provides the instance of another component or directive in a parent component and then parent component can access the methods and properties of that component or directive.

ViewChild are used for component communication in Angular.

Therefore, if a parent component wants access of child component then it uses ViewChild.

Angular 8 : Online Class

@ViewChildren in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

@ViewChildren

Angular @ViewChildren Decorator is used to get the QueryList of elements or directives from the view DOM. When a new child element is added or removed, the QueryList will be updated and its changes function will emit new value. @ViewChildren sets the data before ngAfterViewInit callback.

@ViewChildren has following metadata properties.

- ✓ selector: Selector for querying.
- ✓ read: It is used to read different item from the queried elements.

@ViewChildren supports following selector.

1. A Component class

```
@ViewChildren(WriterComponent)
writers: QueryList<WriterComponent>;
```

2. A Directive Class

```
@ViewChildren(MessageDirective)
msgList: QueryList<MessageDirective>;
```

@ViewChildren

3. Template reference variable
@ViewChildren('pname')
persons: QueryList<ElementRef>;

4. Using read metadata.
@ViewChildren(WriterComponent, { read: ElementRef })
writers: QueryList<ElementRef>;

Angular 8 : Online Class

QueryList in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

QueryList

ViewChildren and ContentChildren uses QueryList to store elements or directives from view DOM and content DOM respectively.

We can subscribe to the changes of QueryList to get the current state of QueryList.

It provides methods such as map, filter, find, forEach etc. QueryList can contain the elements of the type directive, component, ElementRef, any etc.

QueryList is used with @ViewChildren as given below.

```
@ViewChildren('Writer')
allWriters: QueryList<testComponent>
```

QueryList is used with @ContentChildren as given below.

```
@ContentChildren(BookDirective)
topBooks: QueryList<BookDirective>
```

QueryList

- ✓ QueryList: length, first, last
- ✓ QueryList.forEach
- ✓ QueryList.changes
- ✓ QueryList.find
- ✓ QueryList.map
- ✓ QueryList.filter

QueryList

QueryList: length, first, last

We can get the length of QueryList of current state, first element and last element as following.

```
@ViewChildren('qlWriter')
allWriters: QueryList<WriterComponent>

let len = this.allWriters.length;
let firstEl = this.allWriters.first;
let lastEl = this.allWriters.last;
```


QueryList.forEach

To iterate QueryList, it provides forEach() method.

```
@ViewChildren('qlWriter')
allWriters: QueryList<WriterComponent>

this.allWriters.forEach(writer => console.log(writer.writerName + ' - ' + writer.bookName));
```

QueryList

QueryList.changes

To listen the changes in QueryList, we can subscribe to its changes. QueryList provides changes getter property that returns the instance of Observable.

```
@ViewChildren('bkWriter')
allWriters: QueryList<WriterComponent>

this.allWriters.changes.subscribe(list => {
    list.forEach(writer => console.log(writer.writerName + ' - ' + writer.bookName));
});
```

In the list we will get current state of QueryList at any time.

QueryList

QueryList.find

To find an element with a specific value, QueryList provides find method that returns the matching object.

```
@ViewChildren('qlWriter')
allWriters: QueryList<WriterComponent>

let javaWriter = this.allWriters.find(writer => writer.bookName === 'Angular 8 Tutorials');
```

QueryList.map

To map the elements of a QueryList, it provides map method that returns an array of mapped elements.

```
@ViewChildren('qlWriter')
allWriters: QueryList<WriterComponent>

let wnames = this.allWriters.map(writer => writer.writerName);
```

QueryList.filter

To filter a QueryList, it provides filter method that returns an array of elements matching the filter condition.

```
@ViewChildren('qlWriter')
allWriters: QueryList<WriterComponent>

let writers = this.allWriters.filter(writer => writer.writerName === 'Ajeet');
```