Angular Notes

Sunday, April 12, 2020 12:37 PM

Data binding is a technique, where the data stays in sync between the component and the view. Whenever the user updates the data in the view, Angular updates the component. When the component gets new data, the Angular updates the view.

The Angular supports four types of Data binding

1. Interpolation
   Data is bind from component to View
2. Property Binding
   Data is bind from component to the property of an HTML control in the view like
3. Event Binding
   The DOM Events are bind from View to a method in the Component
4. Two-way Binding/Model Binding
   The data flow in both directions from view to component or from component to view

Building Blocks of Angular Application

the seven main building blocks of an Angular Application.

1) Component
2) Templates
3) Metadata
4) Data Binding
5) Directives
6) Services
7) Dependency Injection

Directive

The Directives help us to manipulate the view.
A directive is a class, which we create using the `@Directive` class decorator. It contains the metadata and logic to manipulate the DOM

Angular supports two types of directives. One is **structural directives** which change the structure of the View and the other one is **attribute directive**, which alters the style of our view.

Services

The Services provide service to the Components or to the other Services.
Angular does not have any specific definition for Services. You just create a class, export a method, that does some specific task and it becomes a service. You do not have to do anything else.

```
export class MyLogger {
    AddTolog(msg: any)
    {
        console.log(msg);
    }
}
```

Dependency Injection

Dependency injection is a method by which, a new instance of a service is injected into the component, which requires it.
The dependencies injection is mostly used to inject services into components or to other services.

If the service is already created, then the injector does not create the service but uses it. The Service needs to tell the Angular that it can be injected into any components, which requires it. This is done by using the @Injectable call decorator

Features of Typescript

Typescript also supports Modules, classes, Interfaces, and Generics. This makes Typescript an ideal choice for our Angular Application.

What is a Module Loader

Module loader takes a group of modules with their dependencies and merges them into a single file in the correct order. This process is called as **Module bundling**.

Why Module Loader Required

In Our applications, we create a lot of javascript files. We then include them in our main HTML file using the **<script>** tag. When user requests for your file, the browser loads these file. This is inefficient as it reduces the page speed as the browser requests each file separately.
The above problems can be solved by bundling several files together into one big file. The entire file can be downloaded in one single request reducing the number of requests. You can also minify the file (remove the extra spaces, comments, unnecessary characters, etc) and make files smaller
There are many module loaders are available. The two more popular Module loaders are Webpack and SystemJS
Webpack is the Module Loader than Angular installs when we use the Angular CLI. It also configures it, Hence we do not need to anything

What is an Angular Component

The Component is the main building block of an Angular Application.
The component contains the data & user interaction logic that defines how the View looks and behaves. A view in Angular refers to a template (HTML).

The Component is responsible to provide the data to the view. The Angular does this by using data binding to get the data from the

Component to the View.

The Angular applications will have lots of components. Each component handles a small part of UI. These components work together to produce the complete user interface of the application

The Components consists of three main building blocks

- Template
- Class
- MetaData

Interpolation

The interpolation is much more powerful than just getting the property of the component. You can use it to invoke any method on the component class or to do some mathematical operations etc.

If you want to bind the expression that is other than a string (for example – boolean), then Property Binding is the best option.

The Angular updates the view, when it runs the **change detection**. The change detection runs only in response to asynchronous events, such as the arrival of HTTP responses, raising of events, etc. In the example above whenever you type on the input box, it raises the `keyup` event. It forces the angular run the change detection, hence the view gets the latest values.

The safe navigation operator (?)

You can make use of a safe navigation operator (?) to guards against `null` and `undefined` values.

Property Binding

Property Binding is a one-way data-**binding** technique. In **property binding**, we **bind** a **property** of a DOM element to a field which is a defined **property** in our component TypeScript code.

The brackets, [], tell Angular to evaluate the template expression. If you omit the brackets, Angular treats the expression as a constant string and initializes the target property with that string:

Property Binding Vs Interpolation

```
<h1> {{ title }} </h1>
```

is same as the following Property binding

```
<h1 [innerText]="title"></h1>
```

In fact, Angular automatically translates interpolations into the corresponding property bindings before rendering the view.

Interpolation requires the expression to return a string. If you want to set an element property to a **non-string** data value, you must use property binding.

$event Payload

DOM Events carries the event payload. I.e the information about the event. We can access the event payload by using `$event` as an argument to the handler function. Otherwise, it will result in an error.

Two-Way data binding

It automatically sets up property binding to value property of the element. It also sets up the event binding to `valueChange` Property. But since we hardly have any HTML element, which follows those naming conventions unless we create our own component. This is where `ngModel` directive from `FormsModule` steps in and provides two way binding to all the known HTML form elements.

Directives

The Angular directive helps us to manipulate the DOM. You can change the appearance, behavior, or layout of a DOM element using the directives. They help you to extend HTML. The Angular directives are classified into three categories based on how they behave. They are Component, Structural and Attribute Directives

Structural Directives

Structural directives can change the DOM layout by adding and removing DOM elements. All structural Directives are preceded by Asterix symbol

ngFor, ngIf, ngSwitch

Attribute Directives

An Attribute or style directive can change the appearance or behavior of an element.

ngModel, ngClass, ngStyle

Trackby in ngFor and in Mat-Table

Angular Trackby option improves the Performance of the ngFor if the collection has a large no of items and keeps changing

```
trackByFn(index, item) {
    return item.title;
  }
```

```
<li *ngFor="let movie of movies; let i=index;trackBy: trackByFn;">
```

We should always specify the primary key or unique key as the `trackBy` clause.

Why use trackyBy with ngFor directive :

- ngFor directive may perform poorly with large lists.
- A small change to the list like, adding a new item or removing an existing item may trigger several DOM manipulations.

Any kind of data source that corresponds to repeated rows or items, especially ones that are fetched via Observables, should ideally allow you to use `trackBy`

For a table of tasks fetched as Observables, we can have:

```
<table mat-table [dataSource]="category.tasksObs" [trackBy]="trackTask"></table>
```

ngIf else

The `ngIf` allows us to define optional `else` block using the `ng-template`

```
<div *ngIf="condition; else elseBlock">
   content to render, when the condition is true
</div>

<ng-template #elseBlock>
   content to render, when the condition is false
</ng-template>
```

ngIf then else

You can also define `then` `else` block using the `ng-template`

```
<div *ngIf="condition; then thenBlock else elseBlock">
   This content is not shown
</div>

<ng-template #thenBlock>
   content to render when the condition is true.
</ng-template>

<ng-template #elseBlock>
   content to render when condition is false.
</ng-template>
```

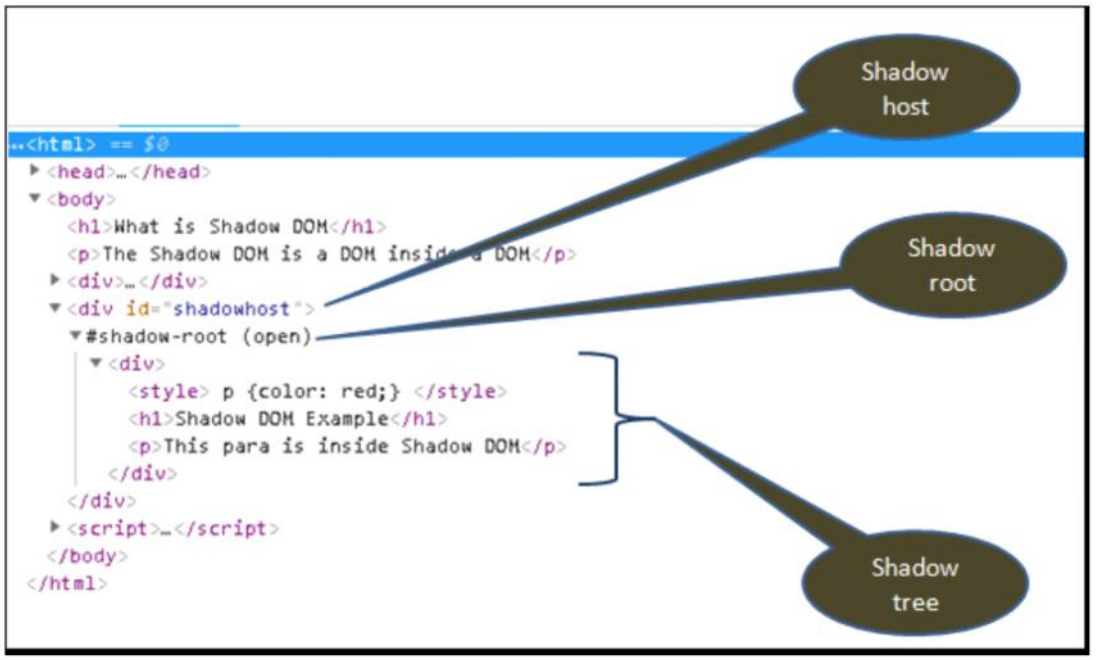
What is Shadow DOM

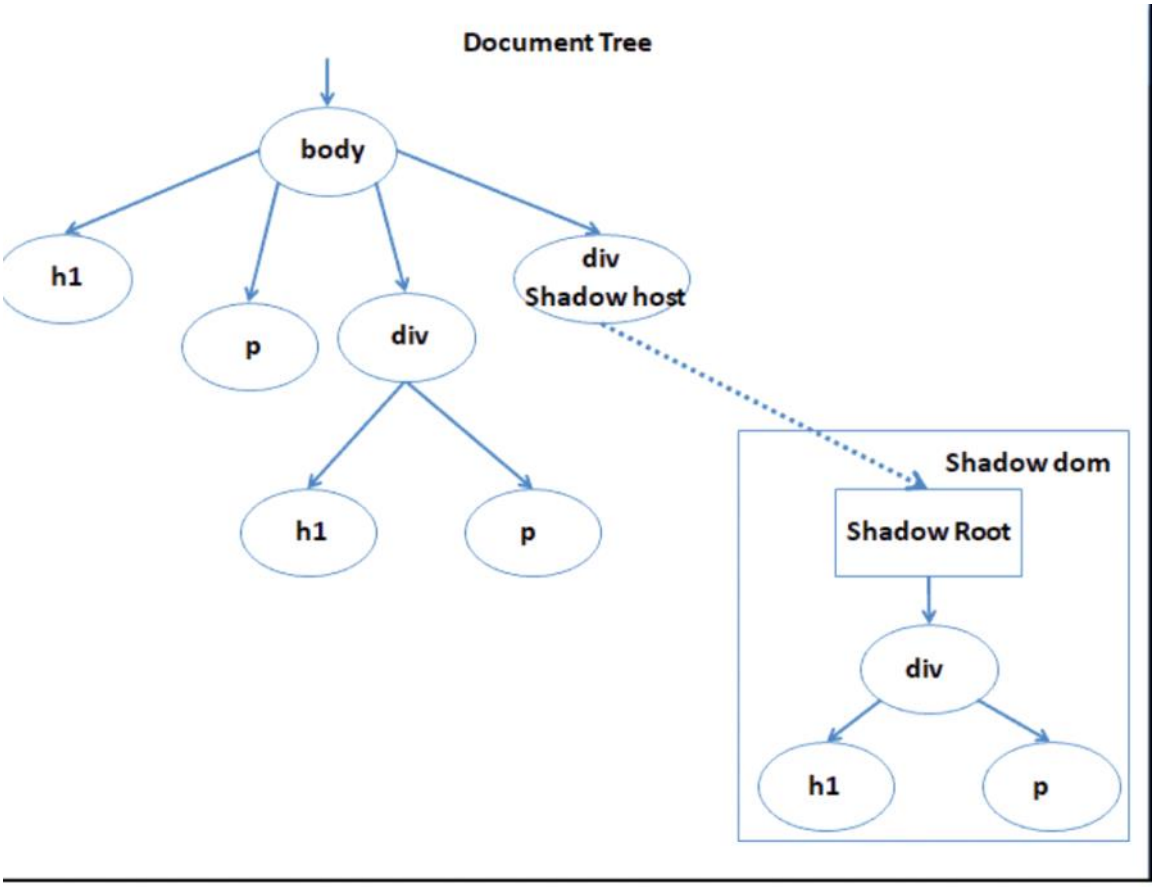
The Shadow DOM is a scoped sub-tree of DOM. It is attached to a element (shadow host) of the DOM tree, but do not appear as child nodes of that element.

https://www.tektutorialshub.com/html5/shadow-dom/

```
<html>
  <head>
    <style>
      p  {color: orange;}
    </style>
    <title>Hello Shadow DOM </title>
  </head>
  <body>
    <h1>What is Shadow DOM</h1>
    <p>The Shadow DOM is a DOM inside a DOM</p>
```

```
    <div>
      <style>p {color: blue;}</style>
      <h1>Shadow DOM Example</h1>
      <p>This is not Shadow DOM</p>
    </div>

    <div id="shadowhost"></div>

  </body>
</html>

<script>
    var host =document.getElementById('shadowhost');
    var shadowRoot=host.attachShadow({mode: 'open'});

    var div = document.createElement('div');

    div.innerHTML='<style> p {color: red;} </style>  <h1>Shadow DOM Example</h1>   <p>This para is
inside Shadow DOM</p>';

    shadowRoot.appendChild(div);
</script>
```





What is View Encapsulation in Angular ?

The View Encapsulation in Angular is a strategy which determines how angular hides (encapsulates) the styles defined in the component from bleeding over to the the other parts of the application.

The following three strategies provided by the Angular to determine how styles are applied.

- ViewEncapsulation.None
- ViewEncapsulation.Emulated // this is default
- ViewEncapsulation.ShadowDOM

ViewEncapsulation.None

The ViewEncapsulation.None is used, when we do not want any encapsulation. When you use this, the styles defined in one component affects the elements of the other components.

The important points are

- The styles defined in the component affect the other components
- The global styles affect the element styles in the component

```
import { Component,ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-none',
```

```
  template: `<p>I am not encapsulated and in blue
             (ViewEncapsulation.None) </p>`,
  styles: ['p { color:blue}'],
  encapsulation: ViewEncapsulation.None
})
export class ViewNoneComponent {
}
```

ViewEncapsulation.Emulated

The ViewEncapsulation.Emulated strategy in angular adds the unique HTML attributes to the component CSS styles and to the markup so as to achieve the encapsulation.

From <https://www.tektutorialshub.com/angular/angular-view-encapsulation/>

The important points are

- This strategy isolates the component styles. They do not bleed out to other components.
- The global styles may affect the element styles in the component
- The Angular adds the attributes to the styles and mark up

```
@Component({
  selector: 'app-emulated',
  template: `<p>Using Emulator</p>`,
  styles: ['p { color:red}'],
  encapsulation: ViewEncapsulation.Emulated
})
```

```
<app-emulated _ngcontent-c0 _nghost-c2>
    <p _ngcontent-c2>Using Emulator</p>
</app-emulated>
```

ViewEncapsulation.ShadowDOM

The Shadow DOM is part of the Web Components standard.

The browser keeps the shadow DOM separate from the main DOM. The rendering of the Shadow dom and the main DOM happens separately. The browser flattens them together before displaying it to the user. The feature, state & style of the Shadow DOM stays private and not affected by the main DOM. Hence it achieves the true encapsulation.

```
@Component({
  selector: 'app-shadowdom',
  template: `<p>I am encapsulated inside a Shadow DOM ViewEncapsulation.ShadowDom</p>`,
  styles: ['p { color:brown}'],
  encapsulation: ViewEncapsulation.ShadowDom
})
```

The important points are

- The shadow dom achieves the true encapsulation
- The parent and sibling styles still affect the component. but that is angular's implementation of shadow dom
- The shadow dom not yet supported in all the browsers. You can check it from the this link.