Angular : Main Building Blocks

Angular is a TypeScript-based open-source web application framework

Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps.

A single-page application (SPA) is a web application or website that interacts with the web browser by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages.

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for development of large applications and transcompiles to JavaScript.

NPM – Node Package Manager

The architecture of an Angular application relies on certain fundamental concepts.

* The basic building blocks are NgModules, which provide a compilation context for components.
* An NgModule can associate its components with related code, such as services, to form functional units.
* An Angular app is defined by a set of NgModules.
* Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.
* NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the Router NgModule.
* So, in conclusion Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability.
* In addition, this technique lets you take advantage of **lazy-loading**—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

Components

Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM).

Component controls a patch of screen real estate that we could call a view and declares reusable UI building blocks for an application.

The core concept or the basic building block of Angular Application is nothing but the components. That means an angular application can be viewed as a collection of components and one component is responsible for handling one view or part of the view.

Each component defines

A class - that contains application data and logic,

A Template - an HTML template that defines a view to be displayed in a target environment

Decorators - metadata that tells angular how to process a class.

Decorators

Decorators are a design pattern that is used to separate modification or decoration of a class without modifying the original source code. Decorators are functions that allow a service, directive or filter to be modified prior to its usage

There are many built-in decorators are available in angular. Some of them are as follows:

@NgModule to define a module.

@Component to define components.

@Injectable to define services.

@Input and @Output to define properties, etc.

Usage of decorators

Class Decorators: @Component and @NgModule

Property Decorators: @Input and @Output (These two decorators are used inside a class)

Method Decorators: @HostListener (This decorator is used for methods inside a class like a click, mouse hover, etc.)

Parameter Decorators: @Inject (This decorator is used inside class constructor).

All the above built-in decorators are imported from **@angular/core** library

Directives

Directives are instructions in the DOM. They specify how to place your components and business logic in the Angular.

Directives are js class and declared as @directive. There are 3 directives in Angular.

Component Directives - Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime. @Component

Structural Directives - Structural directives start with a \* sign. These directives are used to manipulate and change the structure of the DOM elements. For example, \*ngIf and \*ngFor.

Attribute Directives - Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass, ngStyle etc.

Services and DI

* For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class.
* To define a class as a service in Angular, use the @ Injectable () decorator to provide the metadata that allows Angular to inject it into a component as a dependency.
* Similarly, use the @ Injectable () decorator to indicate that a component or other class (such as another service, a pipe, or an NgModule) has a dependency.
* A service class definition is immediately preceded by the @Injectable() decorator.

Dependency injection (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

Is a software design pattern in which components are given their dependencies instead of hard coding them within the component. This relieves a component from locating the dependency and makes dependencies configurable.

Routing

The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app

To define navigation rules, you associate navigation paths with your components

when you create your project using the CLI. You will be prompted if you Would you like to add Angular routing? (y/N). If you answer with y for Yes, the Angular router will be automatically setup in your project without having to add it manually.

If you choose a N, then you can add the routes later as below :

For Eg, let us assume, we have created the below components using ng g c

AccountListComponent

AccountCreateComponent

then to add these to routes array manually, Open the src/app/app-routing.module.ts file and start by adding the following imports to the components:

import { AccountListComponent } from './account-list/account-list.component';

import { AccountCreateComponent } from './account-create/account-create.component';

Data Binding

Why do we need Data Binding?

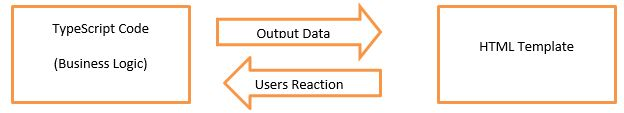
Whenever we want to develop any data-driven web application, then as a developer we need to keep the focus on two important things i.e. Data and the UI (User Interface) and it is more important to find an efficient way to bind them (Data and UI) together. Also, the data can arrive in several chunks and you need to update the user interface with the latest or updated data.

Data Binding means to bind the data (Component’s field) with the View (HTML Content).

Angular Databinding is used for communication. It is used to communicate between your TypeScript code (your business logic) and the other component which is shown to the users i.e. HTML layout.

There are 2 types of Data Binding

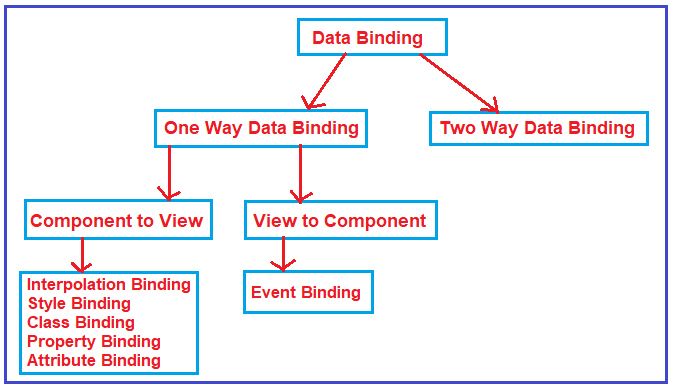
1. One-Way – where HTML template is changed when we make changes in TypeScript code.
2. Two-Way - automatic synchronization of data happens between the Model and the View.



One-Way binding (It flows from Component to Template) is achieved using

1. Interpolation – Works only with strings and not other data types{{}}
2. Property Binding – is used when you have to set an element property to a non-string data value.[]

The main advantage of property binding is that it facilitates you to control elements property.



Event Binding

Angular facilitates us to bind the events along with the methods. This process is known as event binding. Event binding is used with parenthesis ().

Eg: <button (click)="onClickMe()">Click me!</button>()

Attribute Binding :

Note : Both (Interpolation and Property Binding) are dealing with the DOM Properties but not with the HTML attributes. But there are some HTML elements (such as colspan, area, etc) that do not have the DOM Properties. Then we cannot use interpolation or two way binding. In such cases we use attribute binding

To tell the angular framework that we are setting an attribute value, we have to prefix the attribute name with the attr and a DOT as shown below.

If we declare columnspan:number=2 in our class, then we can specify the attribute binding as below

[attr.colspan]=”columnspan” or {{columnspan}}

HTML Attributes vs DOM Properties

Example :see the below binding

Interpolation example: <button disabled='{{IsDisabled}}’>Click Me</button>

Property binding example: <button [disabled]=’IsDisabled’>Click Me</button>

If you look at the above two examples, you may feel that you are binding to the Button’s disabled attribute, but that is not true. You are actually binding to the disabled property of the button object. So, the Angular data-binding is all about binding to the DOM object properties and not the HTML element attributes.

What is the difference between the HTML element attribute and DOM property?

The Attributes are defined by HTML whereas the properties are defined by the DOM.

The attribute’s main role is to initializes the DOM properties. So, once the DOM initialization complete, the attributes job is done.

Property values can change, whereas the attribute values can never be changed.

Angular Directives

The Angular Directives are the elements which are basically used to change the behavior or appearance or layout of the DOM (Document Object Model) element. In other words, we can say that the directives are basically used to extend the power of HTML attributes and to change the appearance or behavior of a DOM element.

Built-in attribute directives

NgClass—adds and removes a set of CSS classes.

NgStyle—adds and removes a set of HTML styles. (can effect many inline styles)

NgModel—adds two-way data binding to an HTML form element

Built-in structural directives

NgIf—conditionally creates or destroys subviews from the template.

NgFor—repeat a node for each item in a list.

NgSwitch—a set of directives that switch among alternative views.

