# Angular - Services and Dependency Injection

In a given Angular application, there may be one or more services can be used. Similarly, an Angular component may depend on one or more services.

Also, Angular services may depend on another services to work properly. Dependency resolution is one of the complex and time consuming activity in developing any application. To reduce the complexity, Angular provides **Dependency Injection** pattern as one of the core concept.

Let us learn, how to use Dependency Injection in Angular application in this chapter.

## **Create Angular service**

An Angular service is plain Typescript class having one or more methods (functionality) along with **@Injectable** decorator. It enables the normal Typescript class to be used as service in Angular application.

import { Injectable } from '@angular/core'; @Injectable()

export class DebugService {

constructor() { }

}

Here, **@Injectable** decorator converts a plain Typescript class into Angular service.

## **Register Angular service**

To use **Dependency Injection**, every service needs to be registered into the system. Angular provides multiple option to register a service. They are as follows −

* ModuleInjector @ root level
* ModuleInjector @ platform level
* ElementInjector using providers meta data
* ElementInjector using viewProviders meta data
* NullInjector

### **ModuleInjector @ root**

**ModuleInjector** enforces the service to used only inside a specific module. **ProvidedIn**meta data available in **@Injectable** has to be used to specify the module in which the service can be used.

The value should refer to the one of the registered Angular Module (decorated with **@NgModule). root** is a special option which refers the root module of the application. The sample code is as follows −

import { Injectable } from '@angular/core'; @Injectable({

providedIn: 'root',

})

export class DebugService {

constructor() { }

}

### **ModuleInjector @ platform**

**Platform Injector** is one level higher than **ModuleInject** and it is only in advanced and rare situation. Every Angular application starts by executing **PreformBrowserDynamic().bootstrap** method (see **main.js**), which is responsible for bootstrapping root module of Angular application.

**PreformBrowserDynamic()** method creates an injector configured by **PlatformModule**. We can configure platform level services using **platformBrowser()** method provided by **PlatformModule**.

### **NullInjector**

**NullInjector** is one level higher than platform level **ModuleInjector** and is in the top level of the hierarchy. We could not able to register any service in the **NullInjector**. It resolves when the required service is not found anywhere in the hierarchy and simply throws an error.

### **ElementInjector using providers**

**ElementInjector** enforces the service to be used only inside some particular components. providers and **ViewProviders** meta data available in **@Component** decorator is used to specify the list of services to be visible for the particular component. The sample code to use providers is as follows −

**ExpenseEntryListComponent**

// import statement

import { DebugService } from '../debug.service';

// component decorator

@Component({

selector: 'app-expense-entry-list',

templateUrl: './expense-entry-list.component.html',

styleUrls: ['./expense-entry-list.component.css'],

providers: [DebugService] })

Here, **DebugService** will be available only inside the **ExpenseEntryListComponent** and its view. To make DebugService in other component, simply use **providers** decorator in necessary component.

### **ElementInjector using viewProviders**

**viewProviders** is similar to **provider** except it does not allow the service to be used inside the componentâ€™s content created using **ng-content** directive.

**ExpenseEntryListComponent**

// import statement

import { DebugService } from '../debug.service';

// component decorator

@Component({

selector: 'app-expense-entry-list',

templateUrl: './expense-entry-list.component.html',

styleUrls: ['./expense-entry-list.component.css'], viewProviders: [DebugService]

})

Parent component can use a child component either through its view or content. Example of a parent component with child and content view is mentioned below −

**Parent component view / template**

<div>

child template in view

<child></child>

</div>

<ng-content></ng-content>

**child component view / template**

<div>

child template in view

</div>

Parent component usage in a template (another component)

<parent>

<!-- child template in content -->

<child></child>

</parent>

Here,

* **child** component is used in two place. One inside the parent’s view. Another inside parent content.
* Services will be available in child component, which is placed inside parent’s view.
* Services will not be available in child component, which is placed inside parent’s content.