Angular Injector, @Injectable & @Inject

2 Comments / 4 minutes of reading / March 9, 2023

← Angular Dependency Injection

Angular Tutorial

Angular Providers →

The Angular Injector lives at the core of Angular Dependency injection system. In this article, we will look at it in more detail. We will also take a look at the `@Injectable` & `@Inject` decorators.

Table of Contents

What is Angular Injector

The Angular Injector is responsible for instantiating the dependency and injecting it into the component or service.

The Injector looks for the dependency in the Angular Providers using the Injection token. The Angular Providers array returns the Provider, which contains the

information about how to create the instance of the dependency. The Injector creates the instance and injects it into the Component or service.

When is Angular Injector is created

The Angular creates two Injector trees when the Application bootstraps. One is the `ModuleInjector` tree for the Modules and the other one is the `ElementInjector` tree which is for the Elements (Components & Directives etc).

The Angular loads the Root Module (named as `AppModule`) when the application bootstraps. It creates `RootModule` Injector for the Root Module. This Injector has an application-wide scope. The `RootModule` Injector becomes part of the `ModuleInjector` Tree.

Angular Root Module loads the `AppComponent`, which is the root component of our app. The `AppComponent` gets its own Injector. We call this root Injector. This Injector becomes the root of the `ElementInjector` tree.

The Root Component contains all other components. Angular App will create child components under the Root Component. All these child component can have their own child components creating a tree of components. The Angular also creates an Injector for all those components creating an Injector tree closely mimicking the component tree. These Injectors become part of the `ElementInjector` tree.

To know more about the Injector Tree refer to the article How Dependency Injection & Resolution Works in Angular.

The Every Injector gets its own copy of Providers.

Registering the service with injector

We register all dependencies of the application with the Providers. Every injector has a Provider associated with it. The Providers metadata of `@NgModule`, `@Component` or `@Directive` is where we register our dependency

```
1
2 providers: [ProductService, LoggerService]
3
```

Where you register your dependency defines the scope of the dependency. The dependency registered with the Module using `@NgModule` decorator is attached the Root Provider (Provider attached to the Root Injector). This Dependency is available to entire application.

The dependency registered with the component is available to that component and any child component of that component.

Another way to register the dependencies is to use the ProvidedIn property of the `Injectable` decorator

@Injectable

The `Injectable` is a decorator, which you need to add to the **consumer of the dependency**. This decorator tells angular that it must Inject the constructor arguments via the Angular DI system

Example of Injectable

We created an example application in the Angular Dependency injection tutorial. It had two services `LoggerService` & `ProductService` as shown below.

LoggerService

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

@Injectable()
export class LoggerService {
  log(message:any) {
    console.log(message);
  }
}

```

ProductService

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

import {Product} from './Product'
import {LoggerService} from './logger.service'

@Injectable()
export class ProductService{

   constructor(private loggerService: LoggerService) {
      this.loggerService.log("Product Service Constructed");
   }

   public  getProducts() {

      this.loggerService.log("getProducts called");
      let products:Product[];

      products=[
         new Product(1,'Memory Card',500),
         new Product(1,'Pen Drive',750),
         new Product(1,'Power Bank',100)
      ]

      this.loggerService.log(products);
      return products;
   }
}

```

The ProductService has a dependency on the LoggerService . Hence it is decorated with the @Injectable decorator. Remove @Injectable() from ProductService and you will get the following error.

> Uncaught Error: Can't resolve all parameters for ProductService: (?)

That is because without DI Angular will not know how to inject LoggerService into ProductService .

Remove @Injectable() from LoggerService will not result in any error as the LoggerService do not have any dependency.

The Components & Directives are already decorated with @Component & @Directive decorators. These decorators also tell Angular to use DI, hence you do not need to add the @Injectable() .

The injectable decorator also has the ProvidedIn property using which you can specify how Angular should provide the dependency.

```
1
2 @Injectable({
3   providedIn: 'root'
4 })
5 export class SomeService{
6 }
7
```

@Inject

The @Inject() is a constructor parameter decorator, which tells angular to Inject the parameter with the dependency provided in the given token. It is a manual way of

injecting the dependency

In the previous example, when we removed the `@Injectable` decorator from the `ProductService` we got an error.

We can manually inject the `LoggerService` by using the `@Inject` decorator applied to the parameter `loggerService` as shown below.

The `@Inject` takes the Injector token as the parameter. The token is used to locate the dependency in the Providers.

```
1
2 export class ProductService{
3    constructor(@Inject(LoggerService) private loggerService) {
4       this.loggerService.log("Product Service Constructed");
5    }
6 }
7
```

Read More

1. Angular Tutorial
2. Angular Services
3. Dependency injection
4. Injector, @Injectable & @Inject
5. Providers
6. Injection Token
7. Hierarchical Dependency Injection
8. Angular Singleton Service
9. ProvidedIn root, any & platform
10. @Self, @SkipSelf & @Optional Decorators