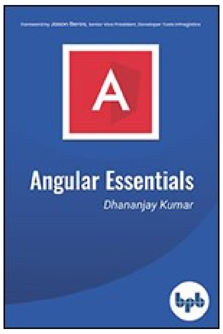

Angular Essentials: The Essential Guide to Learn Angular

by Dhananjay Kumar
BPB Publications. (c) 2019. Copying Prohibited.

Chapter 11: Services and Providers

Service is one of the most important concept of Angular. Usually Service is a singleton class, and it is injected in a component or other service by Angular DI container. In this chapter, you will learn about the following:

- Services
- Providers
- Injectors

Services

Service is a singleton class. Angular creates one instance of service class per injection. Services is a class in Angular with a specific purpose. You may write service for the following scenarios:

- Logging
- Authentication
- Working with API
- Authorization
- Sharing data between unrelated components

Angular service should be singleton and we should configure that while creating. You can use Angular CLI to create a service. Use the following command::

`ng` ***Generate Service App***

This command will generate **`AppService`** as shown in *code listing 11.1*.

Code Listing 11.1

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

@Injectable({
     providedIn: 'root'
})
export class AppService {
     constructor() { }
}
```

Let us talk through the code,

- Service is a class decorated with @Injectable() decorator.
- provideIn property of injector makes service singleton by injecting it at root module.
- In constructor, you can inject another services such as $http to make API call etc.

Let us modify **`AppService`** such that we can use it inside a component. We modified service and added a function **`getData.`** Right now function is returning simple string, however it can return observables and perform complex operations. Consider *Code Listing 11.2.*

Code Listing 11.2

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

@Injectable({
     providedIn: 'root'
})
export class AppService {
     constructor() { }

     getData() {
        return 'Hey I am from Service';
     }
}
```

To use service, do the following steps:

1. Import it in module in which you want to use.
2. Pass it in a provider array. You can learn more about providers in next section.
3. Inject it in constructor of Component to use it.

In component constructor pass service as shown in *code listing 11.3,* such as Angular perform Dependency Injection to pass object to component.

Code Listing 11.3

```
constructor(private service: AppService)   {

}
```

You can use service methods as shown in *code listing 11.4.*

Code Listing 11.4

```
export class AppComponent implements OnInit {
   data;
   constructor(private service: AppService)   {

   }
   ngOnInit() {
      this.data = this.service.getData();
      console.log(this.data);
   }
}
```

In above listing we are calling `getData` method of service inside `ngOnInit` life cycle hook and assigning return data to local variable.

Services are mainly used to work with API using Angular built-in service $http, authentications, authorizations, logging, and so on.

Providers

An Angular Service provider delivers a runtime version of a dependency value. Therefore, when you inject a service, the Angular injector looks at the providers to create the instance of the service. It is the provider that determines which instance or value should be injected at the runtime in component, pipes, or directives. There are many jargons involved here, so to understand purpose of types of providers, let us start with creating a service. Let's say we have a service called `ErrorService`, which is just logging the error message as shown in *code listing 11.5.*

Code Listing 11.5

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

@Injectable()
export class ErrorService {
        logError(message: string) {
           console.log(message);
      }
}
```

Now, we can use this service in a component, as shown in *code listing 11.6.*

Code Listing 11.6

```
import { Component } from '@angular/core';
import { ErrorService } from './errormessage.service';

@Component({
     selector: 'app-root',
     template: `
  <input [(ngModel)]='err' type='text'/>
  <button (click)='setError()'>set error</button>
  ',
     providers: [ErrorService]
})
export class AppComponent {
   constructor(private errorservice: ErrorService) { }
     err: string;
     setError() {
        this.errorservice.logError(this.err);
     }
}
```

We are importing the service, passing it in the providers array, and injecting it in the constructor of component. We are calling the service method on click of the button, and you can see error message passed in the console. Very simple right?

Here, Angular will rely on the values passed in the providers array of component (or module) to find which instance should be injected at the run time.

A Provider determines how object of certain token can be created.

useClass

When you pass a service name in providers array of either component or module, as shown in *code listing 11.7:*

Code Listing 11.7

```
providers:   [ErrorService]
```

Here, Angular is going to use token value **`ErrorService`** and, for token **`ErrorService`**, it will create object of **`ErrorService`** class. The above syntax is a shortcut of the syntax shown in *code listing 11.8.*

Code Listing 11.8

```
   providers:   [{
      provide:  ErrorService,  useClass:  ErrorService
}]
```

The provide property holds the token that serves as the key for the following:

- Locating the dependency value.
- Registering the dependency.

The second property (it is of four types) is used to create the dependency value. There are four possible values of second parameter, as follows:

1. useClass
2. useExisting
3. useValue
4. useFactory

We just saw example of **`useClass.`** Now, consider a scenario that you have a new class for better error logging called **`NewErrorService`** as shown in *code listing 11.9.*

Code Listing 11.9

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

@Injectable()
export class NewErrorService {

           logError(message: string) {
              console.log(message);
              console.log('logged by DJ');
           }
   }
```

useExisting

Now, we want that instead of the instance of `ErrorService`, the instance of `NewErrorService` should be injected. Also, ideally, both classes must be implementing the same Interface, which means they will have same method signatures with different implementation. So now, for the token `ErrorService`, we want the instance of `NewErrorService` to be injected. It can be done by using `useClass`, as shown in *code listing 11.10.*

Code Listing 11.10

```
providers:   [
      NewErrorService,
      { provide:  ErrorService,  useClass: NewErrorService
}
]
```

The problem with the above approach is that there will be two instances of `NewErrorService`. This can be resolved by the use of `useExisting` as shown in *code listing 11.11.*

Code Listing 11.11

```
providers:   [
      NewErrorService,
      { provide: ErrorService, useExisting: NewErrorService
}
]
```

Now there will be only one instance of `NewErrorService` and for token **ErrorService** instance of `NewErrorService` will be created. Let us modify component to use `NewErrorService` as shown in *code listing 11.12.*

Code Listing 11.12

```
import { Component } from '@angular/core';
import { ErrorService } from './errormessage.service';
import { NewErrorService } from './newerrormessage.
service';
@Component({
     selector: 'app-root',
     template: `
   <input [(ngModel)]='err' type='text'/>
   <button (click)='setError()'>set error</button>
     <button (click)='setnewError()'>Set New eroor</
button>
   ',
        providers: [
           NewErrorService,
              { provide: ErrorService, useExisting:
NewErrorService }
      ]
})
export class AppComponent {
        constructor(private errorservice: ErrorService,
private newerrorservice: NewErrorService) { }
     err: string;
     setError() {
        this.errorservice.logError(this.err);
      }
      setnewError() {
        this.newerrorservice.logError(this.err);
      }
}
```

Here, for demonstration purpose, I am injecting service in the component, however, to use service at the module level you can inject in the module itself. Now, on the click of set error button **`NewErrorService`** would be called.

useValue

Both **`useClass`** and **`useExisting`** create instance of a service class to inject for a particular token, but sometimes you want to pass value directly instead of creating instance. So, if you want to pass readymade object instead of instance of a class, you can use **`useValue`** as shown in *code listing 11.13.*

Code Listing 11.13

```
providers:   [
   {
      provide:  ErrorService,  useValue:   {
         logError:  function  (err)   {
            console.log('inhjected directly 1 + err);
         }
      }
}
```

So here, we are injecting a readymade object using **`useValue.`** So for token **`ErrorService,`** Angular will inject the object.

useFactory

There could be scenario where, until runtime, you do not have idea about what instance is needed. You need to create dependency on the basis of information you do not have until the last moment. Let us consider example that you may need to create instance of **`ServiceA`** if user is logged in or **`ServiceB`** if user is not logged in. Information about logged in user may not be available until last time or may change during the use of application.

We need to use **`useFactory`** when information about dependency is dynamic. Let us consider our two services used in previous examples:

1. ErrorMessageService
2. NewErrorMessageService

For token **`ErrorService`** on a particular condition we want either instance of **`ErrorMessageService`** or **`newErrorMessageService.`**

We can achieve that using **useFactory** as shown in *code listing 11.14.*

Code Listing 11.14

```
providers:   [ {
        provide: ErrorService, useFactory: () => {
           let m = 'old'; // this value can change
           if (m === 'old') {
              return new ErrorService();
           } else {
              return new NewErrorService();
           }
        }
     }
]
```

Here, we have taken a very hardcoded condition. You may have complex conditions to dynamically create instance for a particular token. Also, keep in mind that in **`useFactory,`** you can pass dependency. So assume that you have dependency on **`LoginService to`** create instance for **`ErrorService`** token. For that pass **`LoginService`** is deps array as shown in *code listing 11.15.*

Code Listing 11.15

```
providers: [
   {
        provide: ErrorService, useFactory: () => {
           let m = 'old'; // this value can change
           if (m === 'old') {
              return new ErrorService();
           } else {
              return new NewErrorService();
           }
        },
        deps: [LoginService]
  }
]
```

These are fours ways of working with providers in Angular.

Using an Injector

To understand injector, let us revisit Dependency Injection; it is a design pattern, in which a class request external program called DI container for its dependencies. Angular framework has DI container in its design.

For example, **`AppComponent`** has dependency on **`ErrorService.`** Now instead of creating instance of **`ErrorService`** itself, **`AppComponent`** will ask Angular DI container to create an object and pass it to that. Whenever you pass a service in constructor of an Angular class, DI container will create the object of service class and inject to the component. In *code listing 11.16,* Angular will inject object of **`ErrorService`** to **`AppComponent.`**

Code Listing 11.16

```
export class AppComponent {

      constructor(private d:  ErrorService){
      }
}
```

Keep in mind that DI container provides declared dependency when the class is initiated. DI container injects the class in a component, which is decorated with **`@Injectable.`** Any class with **`@Injectable`** decorator can be injected, however to inject it you have to configure the provider, which you learnt in previous section.

A provider instructs an injector how to create the service. You must configure an injector with a provider before that injector can create a service. Injector can be configured at various level of app:

- In the @Injectable() decorator for the service itself. Service will be provided at application root level and will be available in entire application.
- In the @NgModule() decorator for an NgModule. Service will be provided at module level and will be available in the configured module.
- In the @Component() decorator for a component. Service will be provided at component level and will be scoped to the component.

In Service Using providein

At time of creating service, you can use property of **`@Injectable`** decorator as shown in *code listing 11.17* to determine the injector.

Code Listing 11.17

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

@Injectable({
     providedIn: 'root'
})
export class AppService {
     constructor() { } }
```

@Injector() provideIn property provides or inject the service at the root level of application. When you provide service using `provideln,` Angular provides this service at `App Root level.` Since service is provided at root level, Angular can easily optimize the app by removing service reference, if it is not being used. One other important thing, you keep in mind that service provided using `@lnjectbale() provideln` is available in entire application. All components, sub modules etc. will use the same service object created at the application root level. If you have re injected the same service in some feature module that would be ignored an object created at application root level will be used.

In ngModule Providers Array

You can provide service at a particular module level also, for that; you have to pass service in providers array of `@ngModule` decorator as shown in *code listing 11.18.*

Code Listing 11.18

```
@NgModule({
   declarations:   [
      AppComponent
   ],
   imports:   [
      BrowserModule
   ],
   providers:   [AppService],
   bootstrap:   [AppComponent]
})
export class AppModule {   }
```

In this scenario, service will be scoped to `AppModule` level. Any component, sub modules, pipes etc. of `AppModule` can use `AppService.`

In Component

You can provide service at a particular component level also, for that; you have to pass service in providers array of `@component` decorator as shown in *code listing 11.19.*

Code Listing 11.19

```
@Component({
     selector: 'app-root',
     providers: [AppService],
     templateUrl: 'app.component.html'
   '
})
export class AppComponent {
```

In this scenario, service will be scoped to `AppComponent` level.

In this scenario, service will be scoped to `AppComponent` level.

Summary

Services are essential of Angular application. In this chapter, you learnt about following topics:

- Services
- Providers
- Injectors