

Angular Services

BY: Arpit Pandey

ASET,TAAZAA

What is services and why we need service?

As we know only one relation is exist between two components which is

**Parent-Child** relation in which data can be transfer. If there is no relation exist

between components .For sharing data between components we use service .

Dependency Injection?

1. In dependency Injection the DI container have all info about when class need an object or not.
2. The object creation is depend on DI container.

For Example – The actor needs a job (means a class need an object) but the recruiter have information of actor when he or she need job.

Treeshakable services?

1. The process in which the removal of unused files from the final output is known is **Treeshaking.**
2. Treeshakable services are those services that do not get contained in the final output bundle which is created by web pack library if it is not in use.

**Service Cycle?**

Service cycle is consist of four parts.

1. Create Service

ng Generate Service App (Command to create a service)

1. Provide Service

import { Injectable } from ‘@angular/core’;

@Injectable({ providedIn: ‘root’ })

export class AppService { constructor() { } }

1. Inject Service

import { Injectable } from ‘@angular/core’;

@Injectable({ providedIn: ‘root’ })

export class AppService {

constructor() { }

getData() {

return ‘Hey I am from Service’;

}

}

1. Use Service

constructor(private service: AppService) { }

export class AppComponent implements OnInit {

data;

constructor(private service: AppService) { }

ngOnInit() {

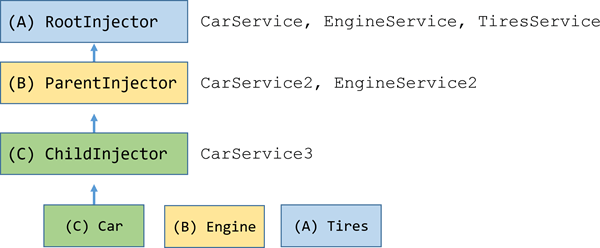
this. Data = this.service.getData();

console.log(this. Data); } }

How angular read cycle of service?

1. Angular check weather service is used of not.
2. Then service is injected or not.
3. Angular service provided or not.
4. Service is created or not.

Hierarchy to read token:



Singleton Service?

1. A singleton is a class that allows only a single instance of itself to be created and gives access to that created instance.
2. When we create a service by default it is singleton because of providedIn ‘root’ in service class.

@Injectable (

{ providedIn: ‘root’

})

Is service is singleton?

1. By default the service is singleton because of providedIn : ‘root’.
2. We can make multiple object by passing the service in providers array.

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

**How much time the service is called it will generate the object on the basis of service providers hierarchy.**

ProvidedIn and Providers array?

**Service using provideIn -** At time of creating service, you can use property of @Injectable decorator as shown in code to determine the injector.

import { Injectable } from ‘@angular/core’;

@Injectable(

{ providedIn: ‘root’ }

)

export class AppService {

constructor() { }

}

Providers Array - 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.

@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.

Properties of Providers Array?

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**

Now, consider a scenario that you have a new class for better error logging called NewErrorService

Code:

import { Injectable } from ‘@angular/core’;

@Injectable() export class NewErrorService {

logError(message: string)

{ console.log(message);

console.log(‘logged by XYZ’); } }

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.

Code:

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.

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.

providers: [ { provide: ErrorService, useValue: { logError: function (err)

{ console.log(‘inhjected directly ‘ + 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

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.

Code:

providers: [ { provide: ErrorService, useFactory: () => { let m = ‘old’; // this value can change

if (m === ‘old’) { return new ErrorService();

} else {

return new NewErrorService(); }}  
}}

Root and Any?

**Root**— This tells **Angular** to provide the service in the application **root** level and the service will be created once (singleton service) and provide the same instance in **every** module that injects the token.

​

**Any**— provides a **unique instance** in every module (including lazy modules) that injects the token and single instance for all remaining modules.

​

**Platform**— specifying providedIn: 'platform' makes the service available in a special singleton platform injector that is shared by all applications on the page.

Eagerly loaded module and Lazy loaded module?

Eagerly loaded module Feature modules under Eager Loading would be loaded before the application starts. This is the default module-loading strategy.

Lazy loaded module Feature modules under Lazy Loading would be loaded on demand after the application starts. It helps to start application faster.