Bo Vandersteene
Posted on Jan 14, 2020

Create a reusable angular service

#angular #architecture

Create a reusable angular service

The goal is to create a service that can be reused throughout our Angular application.
The different kind of use cases this service would be usefull for would be like:

- Call different APIs that return the same output format
- Use different settings for that service
- Have a local state inside the service, and that state should be different amongst different use cases
- ...

The most common way of using a service

By default we use services through dependency injection. Registering a dependency can be done in multiple ways.

- Globally using `provideIn: 'root'` metadata on the service `@Injectable` decorator
- On module level in the providers array of the module
- On component level in the providers array of the component

Now let's create a service that we're not going to inject on the root level.
In this case we will inject it into the module.

For the purpose of the example a simple service is created.

```
@Injectable()
export class ReuseService {
  name: string = 'reuse me!';
}
```

Provide the service on module level.

```
@NgModule({
  // imports, declarations, ....
  providers: [ReuseService]
})
export class AppModule { }
```

And use it in you're component

```
@Component({
  selector: 'my-app',
  template: `
  <ul>
    <li>reuseService: {{reuseService.name}}</li>
  </ul>
  `
})
export class AppComponent  {
  constructor(public reuseService: ReuseService ){}
}
```

But now it turns out that the name should be set outside the service...

Use a factory

> Factory's are used to create a service dynamically. Sometime a value that we use inside the service don't have exist until run time.

To use a factory inside angular, a factory provider should be created. This can be done through a function.

The reuseFactory, is responsible to get the name from somewhere else so the factory function looks as following:

```
// create the factory
export function reuseFactory(name: string){
  return () => new ReuseService(name)
}
```

Now we want to give the responsibility of setting the name to our module. This can be done with a factory

done with a factory.

```
@NgModule({
  // imports, declarations, ....
  providers: [
    // Provide you're service through a factory
    {
      provide: ReuseService,
      useFactory: reuseFactory("reuse service")
    },
 ]
})
export class AppModule { }
```

In this example we only inject a string, but it can also be an other provider.

> More information on how to use factory providers: Angular Docs - factory providers

Create Injection Tokens

An injection token can be used in a DI provider. It can be used also to create unique names for service. Create a `token.ts` file and give an unique name for the token

```
export const REUSE  = new InjectionToken<ReuseService>('ReuseService1');
```

The token can be used in the provider section of the factory

```
@NgModule({
  // imports, declarations, ....
  providers: [
    {
      // use token instead of service class
      provide: REUSE,
      useFactory: reuseFactory("reuse service")
    }
    ]
})
export class AppModule {}
```

Now the injection token can be used inside the component

Now the injection token can be used inside the component

```
@Component({
  selector: 'my-app',
  template: `
  <ul>
    <li>reuseService: {{reuseService.name}}</li>
  </ul>
  `
})
export class AppComponent  {
  constructor(@Inject(REUSE) public reuseService: ReuseService ){}
}
```

Use the service with different settings

Now we get to the point were we want to be albe to use the service multiple times with a different kind of configuration. This can be useful if you want to trigger different endpoints that return the same object format for example.

First some unique tokens will be created.

```
export const REUSE_1  = new InjectionToken<ReuseService>('ReuseService1');
export const REUSE_2  = new InjectionToken<ReuseService>('ReuseService2');
export const REUSE_3  = new InjectionToken<ReuseService>('ReuseService3');
```

These tokens will be used to define our services. In the example bellow we instantiate the service 4 times, each service will have its own instance of the service. If we add state to the services, they will not be shared amongst each other. They live on their own.

```
@NgModule({
  // imports, declarations, ....
  providers: [
    {
      provide: ReuseService,
      useFactory: reuseFactory("reuse service")
    },
    {
      provide: REUSE_1,
      useFactory: reuseFactory("reuse 1")
    },
```

```
    {
      provide: REUSE_2,

      useFactory: reuseFactory("reuse 2")
    },
    {
      provide: REUSE_3,
      useFactory: reuseFactory("reuse 3")
    }
  ]
})
export class AppModule {}
```

Now the different tokens can be used inside the component

```
@Component({
  selector: 'my-app',
  template: `
  <ul>
    <li>reuseService: {{reuseService.name}}</li>
    <li>reuseService 1: {{reuseService1.name}}</li>
    <li>reuseService 2: {{reuseService2.name}}</li>
    <li>reuseService 3: {{reuseService3.name}}</li>
  </ul>
  `
})
export class AppComponent  {
  constructor(public reuseService: ReuseService,
              @Inject(REUSE_1) public reuseService1: ReuseService,
              @Inject(REUSE_2) public reuseService2: ReuseService,
              @Inject(REUSE_3) public reuseService3: ReuseService, ){}
}
```

app.component.ts

```
import { Component, Inject
from '@angular/core';
import { ReuseService } fr
'./reuse.service';
import { REUSE_1, REUSE_2,
REUSE_3 } from "./token";

@Component({
  selector: 'my-app',
  template: `
  <ul>
    <li>reuseService: {
    {reuseService.name}}</
    <li>reuseService 1: {
    {reuseService1.name}}<
    <li>reuseService 2: {
    {reuseService2.name}}<
    <li>reuseService 3: {
    {reuseService3.name}}<
  </ul>
  ` ,
  styleUrls: [
  './app.component.css' ]
})
```

- reuseService: reuse service
- reuseService 1: reuse 1
- reuseService 2: reuse 2
- reuseService 3: reuse 3

Console 1

Discussion (2)

Deniss M • Jan 19 '20

Hey! Thanks for the article! What's the most popular use case for such setup?

Bo Vandersteene • Jan 22 '20

I use it in my current application, where I need to have the different states, for the same data-structure.

I use it also if I need to have different api calls, for lists for example, if they have the same output.

Bo Vandersteene

Frontend Software Engineer & Mentor, works with Angular, Typescript, Javascript. If I have some time over,I like to blog

LOCATION
Belgium

WORK
Senior Software Engineer at Reibo

JOINED
Aug 9, 2019

More from Bo Vandersteene

When a service got destroyed in angular
#typescript #angular #javascript