How to initialize a service on startup

July 27, 2020 by Benoit Paul

Sometimes, you need to initialize a service before your angular application starts. Maybe you want to:

- Fetch data from an external API
- Load configuration data

In order to initialize the service when the application bootstraps, you need the use APP_INITIALIZER injection token.

How to use APP_INITIALIZER

1. Create a factory function. This is the function that will be executed during the application bootstrap. We'll initialize the service in it.
2. Provide the factory function in the AppModule with the APP_INITIALIZER injection token.

Here is a simple example:

init-synchronous.factory.ts

```
export function initSynchronousFactory() {
  return () => {
    console.log('initSynchronousFactory');
    // run initialization code here
  };
}
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { initSynchronousFactory } from './init-synchronous.factory';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, HttpClientModule],
  providers: [
    {
      provide: APP_INITIALIZER,
      useFactory: initSynchronousFactory
    }
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

How to use APP_INITIALIZER with a Promise

As defined in the documentation, if the factory function returns a promise, the application initialization will be blocked until the promise returns.

Note that Observables are not supported yet.

In the following example, the application startup will be blocked for 5 seconds, until the promise is resolved. This means that the AppComponent will be rendered after 5 seconds.

init-long-running.factory.ts

```
export function initLongRunningFactory() {
  return () => {
    return new Promise((resolve) => {
      console.log('initLongRunningFactory - started');
      setTimeout(() => {
        console.log('initLongRunningFactory - completed');
        resolve();
      }, 5000);
    });
  };
}
```

app.component.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { initLongRunningFactory } from './init-long-running.factory';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, HttpClientModule],
  providers: [
    {
      provide: APP_INITIALIZER,
      useFactory: initLongRunningFactory
    }
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

How to use APP_INITIALIZER with dependencies

If you want to initialize a service before the application starts, you need to pass an instance of the service to the factory function, via the `deps` property of the APP_INITIALIZER provider.

First, let's declare a service that will fetch a quote, asynchronously:

quote-of-the-day.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class QuoteOfTheDayService {
  constructor(private httpClient: HttpClient) {}

  fetchQuote() {
    return this.httpClient.get('https://quotes.rest/qod?language=en');
  }
}
```

Next, create a factory function that will use this service. Note that a QuoteOfTheDayService instance is passed as a parameter to the function.

init-with-dependecy.factory.ts

```
import { QuoteOfTheDayService } from './quote-of-the-day.service';
export function initWithDependencyFactory(
  quoteOFTheDayService: QuoteOfTheDayService
) {
  return () => {
    console.log('initWithDependencyFactory - started');
    return quoteOFTheDayService
      .fetchQuote()
      .toPromise()
      .then((result) => {
        console.log('result', result);
        console.log('initWithDependencyFactory - completed');
      });
  };
}
```

Finally, inject the factory function:

app.module.ts

```
import { QuoteOfTheDayService } from './quote-of-the-day.service';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { initWithDependencyFactory } from './init-with-dependency.factory';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, HttpClientModule],
  providers: [
    {
      provide: APP_INITIALIZER,
      useFactory: initWithDependencyFactory,
      deps: [QuoteOfTheDayService]
    }
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

How to initialize configuration from external file and initialize a service

Here is a more useful, realistic scenario. You want to use a LoggingService to log debugging information to the console. You want to turn the logging on in the development environment and off in production. One way to do this is to store this information in a configuration file.

We're going to use the APP_INITIALIZER injection token to load the configuration file, retrieve whether the logging should be on or off and initialize the LoggingService

1- Create the configuration file

app.json

```
{
  "logging": true
}
```

2- Create a configuration service

The `ConfigurationService` will read the configuration file and store its value.

configuration.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class ConfigurationService {
  logging: boolean;

  constructor(private httpClient: HttpClient) {}

  async loadConfiguration(): Promise<any> {
    const config = await this.httpClient
      .get('./assets/config/app.json')
      .toPromise();
    Object.assign(this, config);
    return config;
  }
}
```

3- Create a logging service

The `LoggingService` will log information to the console, if the option is turned on. The options is set via the `initialize()` function.

logging.service.ts

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

@Injectable({
  providedIn: 'root',
})
export class LoggingService {
  enabled: boolean;

  constructor() {}

  initialize(enabled: boolean) {
    this.enabled = enabled;
  }

  log(message: string) {
    if (this.enabled) {
      console.log(message);
    }
  }
}
```

4- Create a factory function that will use the ConfigurationService and LoggingService

init-services.factory.ts

```
import { ConfigurationService } from './configuration.service';
import { LoggingService } from './logging.service';
export function initServicesFactory(
  configurationService: ConfigurationService,
  loggingService: LoggingService
) {
  return async () => {
    console.log('initServicesFactory - started');
    const config = await configurationService.loadConfiguration();
    loggingService.initialize(config.logging);
    console.log('initServicesFactory - completed');
  };
}
```

5- Call the factory function when the application bootstraps

This is the glue that will make sure the ConfigurationService and LoggingService are properly initialized when the application starts.

Notice that since `ConfigurationService` and `LoggingService` are expected parameters of the `initServicesFactory()` function, they are passed into the `deps` array.

app.module.ts

```
import { LoggingService } from './logging.service';
import { ConfigurationService } from './configuration.service';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { initServicesFactory } from './init-services.factory';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, HttpClientModule],
  providers: [
    {
      provide: APP_INITIALIZER,
      useFactory: initServicesFactory,
      deps: [ConfigurationService, LoggingService]
    },
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

How to use multiple APP_INITIALIZER

You may want to use multiple initialization factory functions in your application. For example, you could be developing a large application, with many modules (a configuration module, a security module, several feature modules, etc.). It would make sense for each module to run its own initialization factory function, independently from each other.

To run multiple initialization factory functions, you can register many instances of the APP_INITIALIZER provider and set the property `multi: true` for each one of them.

All functions will run simultaneously, and the application will wait for all of them to complete(and promises to resolve) before bootstrapping.

Building on the examples above, you can run all initialize function simultaneously like so:

app.component.ts

```
import { LoggingService } from './logging.service';
import { ConfigurationService } from './configuration.service';
import { QuoteOfTheDayService } from './quote-of-the-day.service';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { initLongRunningFactory } from './init-long-running.factory';
import { initWithDependencyFactory } from './init-with-dependency.factory';
import { initServicesFactory } from './init-services.factory';
import { initSynchronousFactory } from './init-synchronous.factory';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, HttpClientModule],
  providers: [
    {
      provide: APP_INITIALIZER,
      useFactory: initSynchronousFactory,
      multi: true,
    },
    {
      provide: APP_INITIALIZER,
      useFactory: initLongRunningFactory,
      multi: true,
    },
    {
```

```
      provide: APP_INITIALIZER,
      useFactory: initWithDependencyFactory,
      deps: [QuoteOfTheDayService],
      multi: true,
    },
    {
      provide: APP_INITIALIZER,
      useFactory: initServicesFactory,
      deps: [ConfigurationService, LoggingService],
      multi: true,
    },
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

StackBlitz demo

Angular

< How to use parameters in ngx-translate

> How to use enum in HTML?