Angular How to use APP_INITIALIZER

14 Comments / 5 minutes of reading / March 9, 2023

← Location Service | Angular Tutorial | Runtime configuration →

In This tutorial, we look what is APP_INITIALIZER is and how to use in Angular applications. We will show you how to use it by creating an example app.

Table of Contents

What is APP_INITIALIZER

The APP_INITIALIZER is an instance of InjectionToken . It is a built in Injection token provided by Angular.

The Angular will execute the function provided by this token when the application loads. If the function returns the promise, then the angular will wait until the promise is resolved. This will make it ideal place to perform some initialization logic before the application is initialized.

Dependency Injection Recap

The Angular injector uses the DI token to locate the dependencies in the Angular Provider. We register the dependency in the provider using the token

```
providers :[{ provide: token, useClass: SomeService }]
```

The `token` can be either `type`, a `string` or an instance of `InjectionToken`.

The type token

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

The string token

```
providers :[ {provide:'MESSAGE', useValue: 'Hello Angular'}]
```

The InjectionToken

The `InjectionToken` is used whenever the type that is being used does not have runtime representation such as when injecting an interface, callable type,array etc

```
export const MESSAGE = new InjectionToken<string>('Hello Angular');

providers :[ { provide: HELLO_MESSAGE, useValue: 'Hello World!' }];
```

Where to use APP_INITIALIZER

As mentioned earlier, the APP_INITIALIZER is run when the application is initialized. The Angular suspends the app initialization until all the functions provided by the APP_INITIALIZER are run. If any of those intializers return a promise , then the angular waits for it to resolve , before continuing with the App initialization

This gives us an opportunity to hook into the initialization process and run some our application custom logic. You can load runtime configuration information. load important data from the backend etc.

APP_INITIALIZER Example

Create a new Angular Project

Create app-init.service.ts under the folder src/app .

```
1
2  import { Injectable }  from '@angular/core';
3
4  @Injectable()
5  export class AppInitService {
6
7     constructor() {
8     }
9
10    Init() {
11
12       return new Promise<void>((resolve, reject) => {
13          console.log("AppInitService.init() called");
14          ////do your initialisation stuff here
15          setTimeout(() => {
16             console.log('AppInitService Finished');
17             resolve();
18          }, 6000);
19
20       });
21    }
22 }
23
```

This is a simple service, which has one method `Init`. The method returns a `Promise`.

```
1
2 return new Promise<void>((resolve, reject) => {
3
```

Inside the method we have setup a timer which waits for *6000 milliseconds* and then calls the `resolve`

Open the `app.module.ts`

```
1
2  import { BrowserModule } from '@angular/platform-browser';
3  import { NgModule, APP_INITIALIZER } from '@angular/core';
4  import { HttpClientModule } from '@angular/common/http';
5
6  import { AppRoutingModule } from './app-routing.module';
7
8  import { AppComponent } from './app.component';
9  import { AboutUsComponent } from './about-us.component';
10 import { HomeComponent } from './home.component';
11 import { ContactUsComponent } from './contact-us.component';
12
13 import { AppInitService } from './app-init.service';
14
15 export function initializeApp1(appInitService: AppInitService) {
16   return (): Promise<any> => {
17     return appInitService.Init();
18   }
19 }
20
21 @NgModule({
22   declarations: [
23     AppComponent, AboutUsComponent,HomeComponent,ContactUsComponent
24   ],
25   imports: [
26     HttpClientModule,
27     BrowserModule,
28     AppRoutingModule,
29   ],
30   providers: [
31     AppInitService,
```

```
    { provide: APP_INITIALIZER,useFactory: initializeApp1, deps: [AppInitService], multi: tr
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

First, we need to import `APP_INITIALIZER` from the `@angular/core`

```

import { NgModule, APP_INITIALIZER } from '@angular/core';

```

We need to execute the `appInitService.Init()`. We cannot do it directly from the provider. We need to create a function which invokes the `appInitService.Init()` and returns a `Promise`. We do that in `initializeApp1` function.

```

import { AppInitService } from './app-init.service';

export function initializeApp1(appInitService: AppInitService) {
  return (): Promise<any> => {
    return appInitService.Init();
  }
}

```

Finally,. use the `APP_INITIALIZER` token to provide the `initializeApp1`

```

  providers: [
    AppInitService,
    { provide: APP_INITIALIZER,useFactory: initializeApp1, deps: [AppInitService], multi: tru
  ],

```

The `useFactory` is used because `initializeApp1` is a function and not a class. The Angular Injector executes this function , which in turn calls the `appInitService.Init()` .

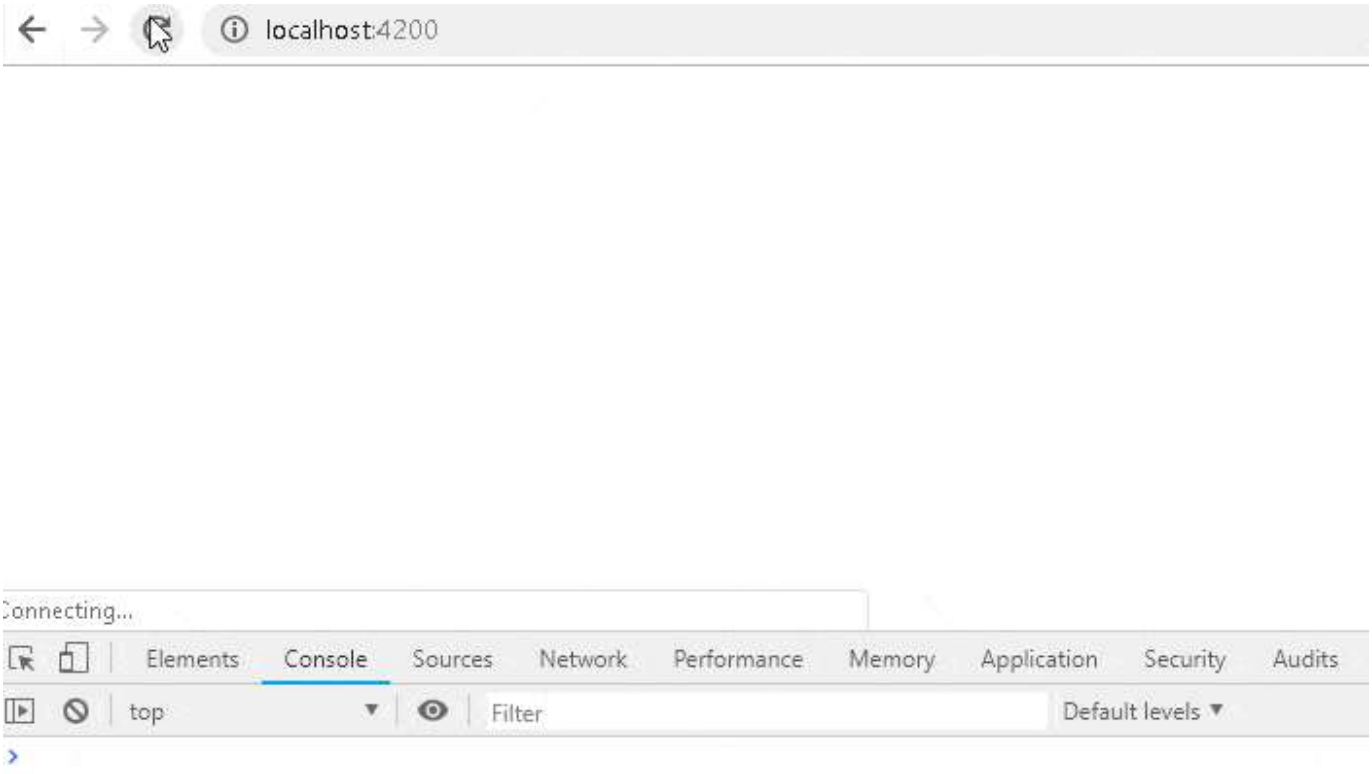
The Angular Dependency injection injects dependencies to classes & components, but not to functions. But our `initializeApp1` is a function and needs `AppInitService` to be injected as the argument. We do that by using the `deps:` flag and let angular know that it needs to create a instance of `AppInitService` and inject it to the `initializeApp1` function.

The `multi : true` creates the multi provider DI token. Which means that you can provide array of providers for a DI token.

If `multi: false` (which is default) is set and use a token more than once, the last to register will override all the previous tokens. i.e you can have only one provider for token.

If `multi: true` is set, then the new providers are added to the previously registered providers making it more than one provider for a token. The angular will execute all of them when the token in invoked.

The run the app & Open the Chrome developer console. You will see that the messages from the service appears first, before the "*Angular running in development mode*" message.



If you return reject from the service, the angular app will not start.

> The Observables are not yet supported in APP_INITIALIZER

Multi Providers in APP_INITIALIZER

You can use the `multi: true` to create Multi Provider token. This means we can create more than one function/service and invoke it during initialization.

Create another factory function `initializeApp2`, which just writes to console after a timeout of 2000 milliseconds.

```
1
2 export function initializeApp2() {
3   return (): Promise<any> => {
4     return new Promise((resolve, reject) => {
```

```
    console.log(`initializeApp2 called`);
    setTimeout(() => {
      console.log(`initializeApp2 Finished`);
      resolve();
    }, 2000);
  });
 };
}

```

Next, register it with the `APP_INITIALIZER` token as shown below.

```

providers: [
  AppInitService,
  { provide: APP_INITIALIZER,useFactory: initializeApp1, deps: [AppInitService], multi: tru
  { provide: APP_INITIALIZER,useFactory: initializeApp2, multi: true}
],

```

Run the app. You will observe the following

1. Both `initializeApp1` & `initializeApp2` runs in succession without waiting for each other.
2. `initializeApp2` finishes first, although it is invoked after `initializeApp1`
3. The Angular waits for both the functions to finish