Define global constants

Asked 3 years, 6 months ago Active 1 month ago Viewed 230k times

225

89

In Angular 1.x you can define constants like this:

```
angular.module('mainApp.config', [])
.constant('API_ENDPOINT', 'http://127.0.0.1:6666/api/')
```

What would be the equivalent in Angular2 (with TypeScript)?

I just don't want to repeat the API base url over and over again in all my services.

typescript angular

edited May 30 at 22:36
Alexander Abakumov
5,529 5 50 79

asked Jan 25 '16 at 7:10
AndreFeijo
4,705 3 16 39

16 Answers

232

Below changes works for me on Angular 2 final version:

```
export class AppSettings {
   public static API_ENDPOINT='http://127.0.0.1:6666/api/';
}
```

And then in the service:

```
import {Http} from 'angular2/http';
import {Message} from '../models/message';
import {Injectable} from 'angular2/core';
import {Observable} from 'rxjs/Observable';
import {AppSettings} from '../appSettings';
import 'rxjs/add/operator/map';

@Injectable()
export class MessageService {

    constructor(private http: Http) { }

    getMessages(): Observable<Message[]> {
        return this.http.get(AppSettings.API_ENDPOINT+'/messages')
            .map(response => response.json())
            .map((messages: Object[]) => {
                return messages.map(message => this.parseData(message));
            });
    }

    private parseData(data): Message {
        return new Message(data);
    }
}
```

edited Jan 10 '17 at 17:24
Community ♦
1 1

answered Jan 25 '16 at 8:28
AndreFeijo
4,705 3 16 39

153

The solution for the configuration provided by the angular team itself can be found here.

Here is all the relevant code:

1) app.config.ts

```
import { OpaqueToken } from "@angular/core";

export let APP_CONFIG = new OpaqueToken("app.config");

export interface IAppConfig {
    apiEndpoint: string;
}

export const AppConfig: IAppConfig = {
    apiEndpoint: "http://localhost:15422/api/"
};
```

2) app.module.ts

```
import { APP_CONFIG, AppConfig } from './app.config';

@NgModule({
    providers: [
        { provide: APP_CONFIG, useValue: AppConfig }
    ]
})
```

3) your.service.ts

```
import { APP_CONFIG, IAppConfig } from './app.config';

@Injectable()
export class YourService {

    constructor(@Inject(APP_CONFIG) private config: IAppConfig) {
             // You can use config.apiEndpoint now
    }
}
```

Now you can inject the config everywhere without using the string names and with the use of your interface for static checks.

You can of course separate the Interface and the constant further to be able to supply different values in production and development e.g.

edited Aug 3 '18 at 16:27
Alexander Abakumov
5,529 5 50 79

answered Oct 27 '16 at 14:19
Ilya Chernomordik
10.7k 7 50 97

3 It only works when I don't specify the type in the constructor of the service. So it works when I do constructor(@Inject(APP_CONFIG) private config){} There's a mention of this here: blog.thoughtram.io/angular/2016/05/23/… but not why. – Mukus Nov 17 '16 at 0:40

I suppose you missed some import or export keyword or something like that, since I use it with the interface and it's as you say very important to have it explicitly statically typed. Maybe you need to provide exact exception here. – Ilya Chernomordik Nov 17 '16 at 7:41

35 None of these solutions, even the angular team recommended approach looks elegant. Why is trying to create constants a cumbersome process in Angular 2? Cant you see how seamless Angular1 made it? Why all the mess? – Rexford Jan 10 '17 at 17:17

27 For anyone else that hits this answer the OpaqueToken in Angular v4 is "deprecated" for InjectionToken - blog.thoughtram.io/angular/2016/05/23/… – mtpultz Mar 26 '17 at 0:20

2 Would it make sense to put the code from Step 1 into `environment.ts` and `environment.prod.ts` so that you can have different constants per environment? @IlyaChernomordik started to mention this in the last paragraph of his answer. – Robert Bernstein Apr 26 '17 at 23:21

62

+50

In Angular2, you have the following provide definition, which allows you to setup different kinds of dependencies:

```
provide(token: any, {useClass, useValue, useExisting, useFactory, deps, multi}
```

Comparing to Angular 1

`app.service` in Angular1 is equivalent to `useClass` in Angular2.

`app.factory` in Angular1 is equivalent to `useFactory` in Angular2.

`app.constant` and `app.value` has been simplified to `useValue` with less constraints. i.e. there is no `config` block anymore.

`app.provider` - There is no equivalent in Angular 2.

Examples

To setup with the root injector:

```
bootstrap(AppComponent,[provide(API_ENDPOINT, { useValue='http://127.0.0.1:6666/api/'
})]);
```

Or setup with your component's injector:

```
providers: [provide(API_ENDPOINT, { useValue: 'http://127.0.0.1:6666/api/'})]
```

`provide` is short hand for:

```
var injectorValue = Injector.resolveAndCreate([
  new Provider(API_ENDPOINT, { useValue: 'http://127.0.0.1:6666/api/'})
]);
```

With the injector, getting the value is easy:

```
var endpoint = injectorValue.get(API_ENDPOINT);
```

edited Dec 12 '16 at 17:05
calcinai
2,340 9 24

answered Jan 25 '16 at 7:17
pixelbits
41.7k 10 69 105

2 I actually would like to have my settings in an external file e.g.: settings.ts How would this file look like? – AndreFeijo Jan 25 '16 at 7:26

Have you considered server-side javascript such as NodeJS? – pixelbits Jan 25 '16 at 7:48

5 Sorry, I didn't understand how I'm going to inject it into my service? As I'm using an external file, do I need to export it? – AndreFeijo Jan 25 '16 at 7:49

I would make it part of your build configuration process. i.e. based on your environment, compile/package different files together, then deploy. All of this you can do with NodeJS with the proper modules. – pixelbits Jan 25 '16 at 7:57

1 NodeJS isn't an option, unfortunately. – AndreFeijo Jan 25 '16 at 8:00

52

In Angular 4, you can use environment class to keep all your globals.

You have environment.ts and environment.prod.ts by default.

For example

```
export const environment = {
  production: false,
  apiUrl: 'http://localhost:8000/api/'
};
```

And then on your service:

```
import { environment } from '../../environments/environment';
...
environment.apiUrl;
```

answered Jul 28 '17 at 14:17
Nacho
1,567 2 15 30

If you are trying to access a `const` inside of a service, you may have to "provide" it in your app module's providers array: `{ provide: 'ConstName',` `useValue: ConstName }` . I was getting a runtime error without this. – daleyjem Jan 2 '18 at 18:17

@daleyjem that's because you were trying to inject it. This approach doesn't use the injector – Aluan Haddad Apr 8 '18 at 0:37

Creating a constant like this is simplest one. I guess counter argument of loosing DI and thereby loosing testability/mockValue is some time over-hyped. In typical app we use so many non-DI component like (RxJS) without bothering testability. – Amitesh Aug 26 '18 at 4:53

41

Updated for Angular 4+

Now we can simply use environments file which angular provide default if your project is generated via angular-cli.

for example

In your environments folder create following files

- `environment.prod.ts`
- `environment.qa.ts`
- `environment.dev.ts`

and each file can hold related code changes such as:

- `environment.prod.ts`

```
export const environment = {
     production: true,
     apiHost: 'https://api.somedomain.com/prod/v1/',
     CONSUMER_KEY: 'someReallyStupidTextWhichWeHumansCantRead',
     codes: [ 'AB', 'AC', 'XYZ' ],
};
```

- `environment.qa.ts`

```
export const environment = {
     production: false,
     apiHost: 'https://api.somedomain.com/qa/v1/',
     CONSUMER_KEY : 'someReallyStupidTextWhichWeHumansCantRead',
     codes: [ 'AB', 'AC', 'XYZ' ],
};
```

- `environment.dev.ts`

```
export const environment = {
     production: false,
     apiHost: 'https://api.somedomain.com/dev/v1/',
     CONSUMER_KEY : 'someReallyStupidTextWhichWeHumansCantRead',
     codes: [ 'AB', 'AC', 'XYZ' ],
};
```

Use-case in application

You can import environments into any file such as services `clientUtilServices.ts`

```
import {environment} from '../../environments/environment';


getHostURL(): string {
    return environment.apiHost;
  }
```

Use-case in build

Open your angular cli file `.angular-cli.json` and inside `"apps": [{...}]` add following code

```
 "apps":[{
        "environments": {
            "dev": "environments/environment.ts",
            "prod": "environments/environment.prod.ts",
            "qa": "environments/environment.qa.ts",
           }
         }
       ]
```

If you want to build for production, run `ng build --env=prod` it will read configuration from `environment.prod.ts` , same way you can do it for `qa` or `dev`

## Older answer

I have been doing something like below, in my provider:

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

@Injectable()
```

```
export class ConstantService {

API_ENDPOINT :String;
CONSUMER_KEY : String;

constructor() {
    this.API_ENDPOINT = 'https://api.somedomain.com/v1/';
    this.CONSUMER_KEY = 'someReallyStupidTextWhichWeHumansCantRead'
  }
}
```

Then i have access to all Constant data at anywhere

```
import {Injectable} from '@angular/core';
import {Http} from '@angular/http';
import 'rxjs/add/operator/map';

import {ConstantService} from  './constant-service'; //This is my Constant Service


@Injectable()
export class ImagesService {
    constructor(public http: Http, public ConstantService: ConstantService) {
    console.log('Hello ImagesService Provider');

    }

callSomeService() {

    console.log("API_ENDPOINT: ",this.ConstantService.API_ENDPOINT);
    console.log("CONSUMER_KEY: ",this.ConstantService.CONSUMER_KEY);
    var url = this.ConstantService.API_ENDPOINT;
    return this.http.get(url)
  }
 }
```

edited May 9 '18 at 7:35

answered Oct 24 '16 at 11:57

Anjum....
2,150 24 30

4 This does not work like a Constant. A constant's value is always the same. In your case, your `API_ENDPOINT` value can be over-written at any point of time. If `this.ConstantService.API_ENDPOINT = 'blah blah'` is declared in the class anytime after your so called "constant" is imported from the `constant-service`, the new value of API_ENDPOINT would be `'blah blah'`. Your solution just shows how to access a variable using a service and not by using a constant. – Devner Jan 19 '17 at 13:51

1 @Devner just make them readonly `readonly API_ENDPOINT :String;` – Flavien Volken Jun 22 '17 at 19:17

@Anjum How angular selects the env files. Should I need to pass env name while starting the app ? – notionquest Apr 19 '18 at 15:33

@notionquest Yes you can pass it, like `ng build --env=prod` – Anjum.... Oct 28 '18 at 5:39

30

While the approach with having a AppSettings class with a string constant as ApiEndpoint works, it is not ideal since we wouldn't be able to swap this real ApiEndpoint for some other values at the time of unit testing.

We need to be able to inject this api endpoints into our services (think of injecting a service into another service). We also do not need to create a whole class for this, all we want to do is to inject a string into our services being our ApiEndpoint. To complete the excellent answer by *pixelbits*, here is the complete code as to how it can be done in Angular 2:

First we need to tell Angular how to **provide** an instance of our ApiEndpoint when we ask for it in our app (think of it as registering a dependency):

```
bootstrap(AppComponent, [
        HTTP_PROVIDERS,
        provide('ApiEndpoint', {useValue: 'http://127.0.0.1:6666/api/'})
]);
```

And then in the service we **inject** this ApiEndpoint into the service constructor and Angular will provide it for us based on our registration above:

```
import {Http} from 'angular2/http';
import {Message} from '../models/message';
import {Injectable, Inject} from 'angular2/core';  // * We import Inject here
import {Observable} from 'rxjs/Observable';
import {AppSettings} from '../appSettings';
import 'rxjs/add/operator/map';

@Injectable()
export class MessageService {

    constructor(private http: Http,
                @Inject('ApiEndpoint') private apiEndpoint: string) { }

    getMessages(): Observable<Message[]> {
        return this.http.get(`${this.apiEndpoint}/messages`)
            .map(response => response.json())
            .map((messages: Object[]) => {
                return messages.map(message => this.parseData(message));
            });
    }
    // the rest of the code...
}
```

edited May 23 '17 at 12:34
Community ♦
1 1

answered Apr 28 '16 at 15:19
Morteza Manavi
30.3k 5 90 80

1 There is now an "official" way of doing recommend by angular team in their tutorial. I have added an answer below: (stackoverflow.com/a/40287063/1671558) – Ilya Chernomordik Nov 14 '16 at 11:19

1 this code is not accurate anymore, implementing this will cause a ApiEndpoint not found on AppComponent. – WilliamX Jul 17 '17 at 16:47

Ok so I'm not alone. Do you know what version this broke? Is there an alternative way that doesn't require define values on a global object then providing them? – Jens Bodal Nov 17 '18 at 7:19

28

This is my recent experience with this scenario:

- **@angular/cli: 1.0.0**
- **node: 6.10.2**
- **@angular/core: 4.0.0**

I've followed the official and updated docs here:

https://angular.io/docs/ts/latest/guide/dependency-injection.html#!#dependency-injection-tokens

Seems **OpaqueToken** is now deprecated and we must use **InjectionToken**, so these are my files running like a charm:

```
app-config.interface.ts
```

```
export interface IAppConfig {

  STORE_KEY: string;

}
```

app-config.constants.ts

```
import { InjectionToken } from "@angular/core";
import { IAppConfig } from "./app-config.interface";

export const APP_DI_CONFIG: IAppConfig = {

  STORE_KEY: 'l@_list@'

};

export let APP_CONFIG = new InjectionToken< IAppConfig >( 'app.config' );
```

app.module.ts

```
import { APP_CONFIG, APP_DI_CONFIG } from "./app-config/app-config.constants";

@NgModule( {
  declarations: [ ... ],
  imports: [ ... ],
  providers: [
    ...,
    {
      provide: APP_CONFIG,
      useValue: APP_DI_CONFIG
    }
  ],
  bootstrap: [ ... ]
} )
export class AppModule {}
```

my-service.service.ts

```
  constructor( ...,
               @Inject( APP_CONFIG ) private config: IAppConfig) {

    console.log("This is the App's Key: ", this.config.STORE_KEY);
    //> This is the App's Key:  l@_list@

  }
```

The result is clean and there's no warnings on console thank's to recent comment of John Papa in this issue:

https://github.com/angular/angular-cli/issues/2034

The key was implement in a different file the interface.

edited May 27 '17 at 22:44

answered Apr 16 '17 at 16:49

JavierFuentes
1,018 9 9

see also stackoverflow.com/a/43193574/3092596 - which is basically the same, but creates injectable modules rather than providers – goredwards Jun 17 '17 at 0:38

16

All solutions seems to be complicated. I'm looking for the simplest solution for this case and I just want to use constants. Constants are simple. Is there anything which speaks against the following solution?

app.const.ts

```
'use strict';

export const dist = '../path/to/dist/';
```

app.service.ts

```
import * as AppConst from '../app.const';

@Injectable()
export class AppService {

    constructor (
    ) {
        console.log('dist path', AppConst.dist );
    }

}
```

edited Aug 3 '18 at 16:37

Alexander Abakumov
5,529 5 50 79

answered May 3 '17 at 7:29

Alexander Schmidt
690 9 18

2 Well, you are using variables outside the scope of the service so you could as well just use window globals then. What we are trying to do is get

constants into the Angular4 dependency injection system so we can keep the scope clean, stubable or mockable. – Joel Hernandez Sep 26 '17 at 8:19

8

Just use a Typescript constant

```
export var API_ENDPOINT = 'http://127.0.0.1:6666/api/';
```

You can use it in the dependency injector using

```
bootstrap(AppComponent, [provide(API_ENDPOINT, {useValue:
'http://127.0.0.1:6666/api/'}), ...]);
```

answered Jan 25 '16 at 7:12
SnareChops
9,706 6 57 84

1 Why inject it? No need for that I think... you can use it as soon as you import it. @SnareChops – Sasxa Jan 25 '16 at 7:17

@Sasxa I agree, though it could be good for unit testing and such. Just trying to provide a complete answer. – SnareChops Jan 25 '16 at 7:18

4 why not const ? – Andreas Jul 3 '16 at 13:49

1 @Andreas You could use `const` yest – SnareChops Jul 4 '16 at 0:16

Please provide a stackblitz of this working. I've seen so many examples of providing a service in the bootstrap method but have yet to find one with a sufficiently working example. Possibly something has changed in a more recent version of angular. – Jens Bodal Nov 17 '18 at 7:15

4

If you're using Webpack, which I recommend, you can set-up constants for different environments. This is especially valuable when you have different constant values on a per environment basis.

You'll likely have multiple webpack files under your `/config` directory (e.g., webpack.dev.js, webpack.prod.js, etc.). Then you'll have a `custom-typings.d.ts` you'll add them there. Here's the general pattern to follow in each file and a sample usage in a Component.

webpack.{env}.js

```
const API_URL = process.env.API_URL = 'http://localhost:3000/';
const JWT_TOKEN_NAME = "id_token";
...
    plugins: [
      // NOTE: when adding more properties, make sure you include them in custom-
typings.d.ts
      new DefinePlugin({
        'API_URL': JSON.stringify(API_URL),
        'JWT_TOKEN_NAME': JSON.stringify(JWT_TOKEN_NAME)
      }),
```

custom-typings.d.ts

```
declare var API_URL: string;
declare var JWT_TOKEN_NAME: string;
interface GlobalEnvironment {
  API_URL: string;
  JWT_TOKEN_NAME: string;
}
```

Component

```
export class HomeComponent implements OnInit {
  api_url:string = API_URL;
  authToken: string = "Bearer " + localStorage.getItem(JWT_TOKEN_NAME)});
}
```

answered Jan 20 '17 at 21:33
occasl
2,662 1 40 65

3

One approach for Angular4 would be defining a constant at module level:

```
const api_endpoint = 'http://127.0.0.1:6666/api/';

@NgModule({
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  providers: [
    MessageService,
    {provide: 'API_ENDPOINT', useValue: api_endpoint}
  ]
})
export class AppModule {
}
```

Then, in your service:

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

@Injectable()
export class MessageService {

    constructor(private http: Http,
      @Inject('API_ENDPOINT') private api_endpoint: string) { }

    getMessages(): Observable<Message[]> {
        return this.http.get(this.api_endpoint+'/messages')
            .map(response => response.json())
            .map((messages: Object[]) => {
                return messages.map(message => this.parseData(message));
            });
    }

    private parseData(data): Message {
        return new Message(data);
    }
}
```

answered Oct 26 '17 at 12:43
Juangui Jordán
1,553 19 21

2

Using a property file that is generated during a build is simple and easy. This is the approach that the Angular CLI uses. Define a property file for each environment and use a command during build to determine which file gets copied to your app. Then simply import the property file to use.

https://github.com/angular/angular-cli#build-targets-and-environment-files

answered Aug 10 '16 at 4:00
R.Creager

62 3

2

I have another way to define global constants. Because if we defined in ts file, if build in production mode it is not easy to find constants to change value.

```
export class SettingService  {

  constructor(private http: HttpClient) {

  }

  public getJSON(file): Observable<any> {
      return this.http.get("./assets/configs/" + file + ".json");
  }
  public getSetting(){
      // use setting here
  }
}
```

In app folder, i add folder configs/setting.json

Content in setting.json

```
{
    "baseUrl": "http://localhost:52555"
}
```

In app module add APP_INITIALIZER

```
   {
      provide: APP_INITIALIZER,
      useFactory: (setting: SettingService) => function() {return setting.getSetting()},
      deps: [SettingService],
      multi: true
    }
```

with this way, I can change value in json file easier. I also use this way for constant error/warning messages.

answered Feb 15 at 9:19
Hien Nguyen

11.7k 5 13 33

0

AngularJS's `module.constant` does not define a constant in the standard sense.

While it stands on its own as a provider registration mechanism, it is best understood in the context of the related `module.value` (`$provide.value`) function. The official documentation states the use case clearly:

> Register a value service with the $injector, such as a string, a number, an array, an object or a function. This is short for registering a service where its provider's $get property is a factory function that takes no arguments and returns the value service. That also means it is not possible to inject other services into a value service.

Compare this to the documentation for `module.constant` (`$provide.constant`) which also clearly states the use case (emphasis mine):

> Register a constant service with the $injector, such as a string, a number, an array, an object or a function. Like the value, it is not possible to inject other services into a constant. But unlike value, *a constant can be injected into a module configuration function (see angular.Module) and it cannot be overridden by an AngularJS decorator*.

Therefore, the AngularJS `constant` function does not provide a constant in the commonly understood meaning of the term in the field.

That said the restrictions placed on the provided object, together with its earlier availability via the $injector, clearly suggests that the name is used by analogy.

If you wanted an actual constant in an AngularJS application, you would "provide" one the same way you would in any JavaScript program which is

```
export const π = 3.14159265;
```

In Angular 2, the same technique is applicable.

Angular 2 applications do not have a configuration phase in the same sense as AngularJS applications. Furthermore, there is no service decorator mechanism (AngularJS Decorator) but this is not particularly surprising given how different they are from each other.

The example of

```
angular
  .module('mainApp.config', [])
  .constant('API_ENDPOINT', 'http://127.0.0.1:6666/api/');
```

is vaguely arbitrary and slightly off-putting because `$provide.constant` is being used to specify an object that is *incidentally* also a constant. You might as well have written

```
export const apiEndpoint = 'http://127.0.0.1:6666/api/';
```

for all either can change.

Now the argument for testability, mocking the constant, is diminished because it literally does not change.

One does not mock π.

Of course your application specific semantics might be that your endpoint could change, or your API might have a non-transparent failover mechanism, so it would be reasonable for the API endpoint to change under certain circumstances.

But in that case, providing it as a string literal representation of a single URL to the `constant` function would not have worked.

A better argument, and likely one more aligned with the reason for the existence of the AngularJS `$provide.constant` function is that, when AngularJS was introduced, JavaScript had no *standard* module concept. In that case, globals would be used to share values, mutable or immutable, and using globals is problematic.

That said, providing something like this through a framework increases coupling to that framework. It also mixes Angular specific logic with logic that would work in any other system.

This is not to say it is a wrong or harmful approach, but personally, if I want a *constant* in an Angular 2 application, I will write

```
export const π = 3.14159265;
```

just as I would have were I using AngularJS.

The more things change...

edited Apr 27 '17 at 0:50

answered Apr 27 '17 at 0:37
Aluan Haddad
14k 2 32 55

The best way to create application wide constants in Angular 2 is by using environment.ts files. The advantage of declaring such constants is that you can vary them according to the environment as there can be a different environment file for each environment.

0

answered Apr 18 '18 at 16:08
Hassan Arafat
11 2

This doesn’t work if you intend to build your application once then deploy it to multiple environments. – Jens Bodal Nov 17 '18 at 7:28

-1

You can make a class for your global variable and then export this class like this:

```
export class CONSTANT {
    public static message2 = [
        { "NAME_REQUIRED": "Name is required" }
    ]

    public static message = {
        "NAME_REQUIRED": "Name is required",
    }
}
```

After creating and exporting your `CONSTANT` class, you should import this class in that class where you want to use, like this:

```
import { Component, OnInit                       } from '@angular/core';
import { CONSTANT                                } from '../../constants/dash-constant';


@Component({
  selector    : 'team-component',
  templateUrl:
`../app/modules/dashboard/dashComponents/teamComponents/team.component.html`,
})

export class TeamComponent implements OnInit {
  constructor() {
    console.log(CONSTANT.message2[0].NAME_REQUIRED);
    console.log(CONSTANT.message.NAME_REQUIRED);
  }

  ngOnInit() {
    console.log("oninit");
    console.log(CONSTANT.message2[0].NAME_REQUIRED);
    console.log(CONSTANT.message.NAME_REQUIRED);
    }
  }
```

You can use this either in `constructor` or `ngOnInit(){}`, or in any predefine methods.

answered Feb 22 '17 at 10:11
Shubham Verma
3,365 3 32 61