## Tokens

There are a number of different types of tokens we can use when configuring providers.

**Learning Objectives**

• Know the 3 different ways of defining tokens.

### String tokens

We can use strings as tokens, like so:

import { ReflectiveInjector } from '@angular/core';

class MandrillService {};

class SendGridService {};

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: "EmailService", useClass: MandrillService }

①

])

;

let emailService = injector.get("EmailService");

console.log(emailService); // new MandrillService()

① We use the token "EmailService".

In the above example we use the string "EmailService" as the token in our class provider configuration.

Although it’s possible to use strings as tokens their use isn’t recommended and instead it’s preferable to use either type tokens or an instance of an InjectionToken.

### Type tokens

We can implement the above example but instead of using strings as the token we can use a type, specifically we can use a class name as the *type*.

Let’s implement the above with EmailService as a base class which MandrillService and SendGridService extend, like so:

class EmailService {};

class MandrillService extends EmailService {};

class SendGridService extends EmailService {};

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: EmailService, useClass: SendGridService }

])

;

let emailService = injector.get(EmailService);

console.log(emailService);

We now use the base class EmailService as the token and not the string "EmailService"

### InjectionToken

The third way of defining a token is via an instance of an InjectionToken, like so:

import { ReflectiveInjector } from '@angular/core'; import { InjectionToken } from '@angular/core';

class MandrillService {}; class SendGridService {};

let EmailService = new InjectionToken<string>("EmailService"); ①

let injector = ReflectiveInjector.resolveAndCreate([ { provide: EmailService, useClass: SendGridService } ②

]);

let emailService = injector.get(EmailService); console.log(emailService);

1. We create an instance of InjectionToken and store it in a variable.
2. We use the instance of InjectionToken as the token in our provider.

The *string* "EmailService" that we pass to InjectionToken is only used to print a

meaningful message to the developer when there is an error. It doesn’t need to be unique.

If using a *base class* as the token is not an option then using an InjectionToken is the *preferred* method over using strings because it prevents name clashes that can occur, like so:

import { ReflectiveInjector } from '@angular/core';

class MandrillService {}; class SendGridService {};

let MandrillServiceToken = "EmailService"; let SendGridServiceToken = "EmailService";

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: SendGridServiceToken, useClass: SendGridService },

{ provide: MandrillServiceToken, useClass: MandrillService },

]);

let emailService1 = injector.get(SendGridServiceToken); let emailService2 = injector.get(MandrillServiceToken); console.log(emailService1 === emailService2);

In the above code we tried to use the *string* variables MandrillServiceToken and SendGridServiceToken as tokens, however they happened to use the same string "EmailService".

So when configuring the injector the above is the same as doing:

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: "EmailService", useClass: SendGridService },

{ provide: "EmailService", useClass: MandrillService },

]);

When configuring an injector with the *same* token multiple times, the last provider just *overwrites* the previous providers.

So in the above code the injector returns an instance of MandrilService as the dependency for both the SendGridServiceToken and the MandrillServiceToken.

The example above might feel quite forced but the issue is quite real. It’s common

for different third party libraries built by different people or teams to use the *same* configuration strings and therefore cause confusing clashes in you application.

InjectionToken solves this shortcoming of using string as tokens since each *instance* of an InjectionToken is unique even if the same descriptive string was passed in, like so:

import { InjectionToken } from '@angular/core';

export const EmailService1 = new InjectionToken<string>("EmailService"); export const EmailService2 = new InjectionToken<string>("EmailService"); console.log(EmailService1 === EmailService2); // false

Even though the two InjectionToken above have the same description, they are *not* equal.

 The string we pass into the token is just used to give better error messages.

So now we can re-write the code like so:

import { ReflectiveInjector } from '@angular/core'; import { InjectionToken } from '@angular/core';

class MandrillService {}; class SendGridService {};

const MandrillServiceToken = new InjectionToken<string>("EmailService"); const SendGridServiceToken = new InjectionToken<string>("EmailService");

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: SendGridServiceToken, useClass: SendGridService },

{ provide: MandrillServiceToken, useClass: MandrillService },

]);

let emailService1 = injector.get(SendGridServiceToken); let emailService2 = injector.get(MandrillServiceToken); console.log(emailService1 === emailService2); // false

The injector returns the correct dependency for the requested token.

**Summary**

A *token* can be either a string, a class or an instance of InjectionToken.

String tokens can cause name clashes so we prefer to use *InjectionTokens* instead.

In the next lecture we will look at how we actually configure DI in Angular.

**Listing**

<http://plnkr.co/edit/Rxs8GQOWQ4OZ5qZQWcO2?p=preview>

*script.ts*

import {ReflectiveInjector} from '@angular/core'; import {InjectionToken} from '@angular/core';

// String Token (Fail Case) Example

{

console.log("String Token (Fail Case) Example");

class MandrillService {

}

class SendGridService {

}

let MandrillServiceToken = "EmailService";

let SendGridServiceToken = "EmailService";

let injector = ReflectiveInjector.resolveAndCreate([

{provide: SendGridServiceToken, useClass: SendGridService},

{provide: MandrillServiceToken, useClass: MandrillService},

]);

let emailService1 = injector.get(SendGridServiceToken);

let emailService2 = injector.get(MandrillServiceToken);

console.log(emailService1 === emailService2);

}

// OpaqueToken

{

console.log("InjectionToken");

class MandrillService {

}

class SendGridService {

}

const MandrillServiceToken = new InjectionToken<string>("EmailService");

const SendGridServiceToken = new InjectionToken<string>("EmailService");

let injector = ReflectiveInjector.resolveAndCreate([

{provide: SendGridServiceToken, useClass: SendGridService},

{provide: MandrillServiceToken, useClass: MandrillService},

]);

let emailService1 = injector.get(SendGridServiceToken);

let emailService2 = injector.get(MandrillServiceToken);

console.log(emailService1 === emailService2); // false

}