## Injectors

**Learning Objectives**

* Know how an injector resolves a *token* into a *dependency*.
* Know how child injectors work.

**Creating Injectors**

At the core of the DI framework is an *injector*.

An injector is passed a *token* and returns a *dependency* (or list of).

We say that an injector *resolves* a token into a dependency.

Normally we never need to implement an *injector*. Angular handles low level injectable implementation details for us and typically we just configure it to give us to behaviour we want.

However to explain how injectors work we will implement some injectable code, like so:

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

①

class MandrillService {};

②

class SendGridService {};

let injector = ReflectiveInjector.resolveAndCreate([

③

MandrillService,

SendGridService

])

;

let emailService = injector.get(MandrillService);

④

console.log(emailService);

1. We import our injector class.
2. We create two service classes, a MandrillService which sends email via the Mandrill platform and the SendGridService which sends email via the SendGrid platform.
3. We configure our injector by providing an array of classes.
4. We pass in a token, the class name, into our injector and ask it to *resolve* to a dependency. In this case it simply returns an instance of MandrillService.



The injector doesn’t return the class, but an

*instance*

of the class instantiated with

new

, like so:

emailService = new MandrillService()

**Dependency caching**

The dependencies returned from injectors are cached. So multiple calls to the *same* injector for the *same* token will return the *same* instance, like so:

let emailService1 = injector.get(MandrillService); let emailService2 = injector.get(MandrillService); console.log(emailService1 === emailService2); // true

 A *different* injector for the *same* token might return a *different* instance of a dependency but the *same* injector will always return the *same* instance.

emailService1 and emailService2 point to exactly the same thing, therefore we can share *state* between two different parts of our application by injecting the same dependency, like so:

let emailService1 = injector.get(MandrillService);

emailService1.foo = "moo";

let emailService2 = injector.get(MandrillService);

console.log(emailService2.foo); // moo

①

① Since emailService1 and emailService2 are the same instance, setting a property on one will mean it can be read from the other and vice versa.

**Child Injectors**

Injectors can have one or more *child* injectors. These behave just like the parent injector with a few additions.

A. Each injector creates it’s own *instance* of a dependency import { ReflectiveInjector } from '@angular/core'; class EmailService {}

let injector = ReflectiveInjector.resolveAndCreate([EmailService]); ① let childInjector = injector.resolveAndCreateChild([EmailService]);

console.log(injector.get(EmailService) === childInjector.get(EmailService)); // false

②

1. The childInjector and parent injector are both configured with the same providers.
2. The childInjector resolves to a different *instance* of the dependency compared to the parent injector.

Both injectors are configured with the *same* EmailService. They each resolved the dependency and returned different instances.

I’ve mentioned previously that different injectors return different instances of dependencies, this is also true even if the injector is a *child* injector, it will still resolve to a different instance to the parent.

B. Child injectors forward requests to their parent injector if they can’t resolve the token locally.

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

let injector = ReflectiveInjector.resolveAndCreate([EmailService]); ① let childInjector = injector.resolveAndCreateChild([]); ②

console.log(injector.get(EmailService) === childInjector.get(EmailService)); // true

③

1. We configure a parent injector with EmailService.
2. We create a child injector from the parent injector, this child injector is not configured with any providers.
3. The parent and child injectors resolve the same token and both return the *same* instance of the dependency.

We request the token EmailService from the childInjector, it can’t find that token locally so it asks it’s parent injector which returns the instance it had cached from a previous direct reqeust.

Therefore the dependency returned by the child and the parent is exactly the same instance .

Don’t worry if the significance of this isn’t clear yet, it’ll become clear in the lecture on configuring DI in Angular.

**Summary**

We configure injectors with providers.

We pass to injectors a token and then resolve this into a dependancy.

Injectors cache dependancies, so multiple calls result in the same instance being returned.

Different injectors hold different caches, so resolving the same token from a different injector will return a different instance.

We create child injectors from parent injectors.

A child injector will forward a request to their parent if it can’t resolve the token itself.

So far we’ve only covered providers that provide *classes* providers can provide other types of dependencies which is the topic of the next lecture.

**Listing**

<http://plnkr.co/edit/0Sp1eLq1bZysbyF3N86p?p=preview>

*script.ts*

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

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

// Simple Injector Example

{

console.log("Simple Injector Example");

class MandrillService {

}

class SendGridService {

}

let injector = ReflectiveInjector.resolveAndCreate([

MandrillService,

SendGridService

]);

let emailService = injector.get(MandrillService);

console.log(emailService);

// Injector Caching Example

{

console.log("Injector Caching Example");

let emailService1 = injector.get(MandrillService);

let emailService2 = injector.get(MandrillService);

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

}

// Injector Caching Caching State Sharing Example

{

console.log("Injector Caching Caching State Sharing Example");

let emailService1 = injector.get(MandrillService);

emailService1.foo = "moo";

let emailService2 = injector.get(MandrillService);

console.log(emailService2.foo); // moo

}

}

// Child Injector Forwards Request to Parent

{

console.log("Child Injector Forwards Request to Parent");

class EmailService {

}

let injector = ReflectiveInjector.resolveAndCreate([EmailService]);

let childInjector = injector.resolveAndCreateChild([]);

console.log(injector.get(EmailService) === childInjector.get(EmailService)); // true

}

// Child Injector Returns Different Instance

{

console.log("Child Injector Returns Different Instance");

class EmailService {

}

class PhoneService {

}

let injector = ReflectiveInjector.resolveAndCreate([EmailService]);

let childInjector = injector.resolveAndCreateChild([EmailService]);

console.log(injector.get(EmailService) === childInjector.get(EmailService));

}