# Angular 依赖注入

# Dependency Injection（DI）基础

## 什么是依赖

随着程序规模的增长，我们常常遇到应用模块需要互相通信的情况。当模块A需要模块B才能运行时，我们就说B是A的依赖。

获取依赖的最常见方式之一就是直接导入（import）一个文件。

在Angular中的一个注入器的范围内，依赖都是单例的。在AngularJs 1.x中的单列只能是全局单列，而Angular中是组件级单列。

## 控制反转

Inversion of Control（IOC原则）正像此原则的非正式称谓“好莱坞原则”。它来自好莱坞的一句常用语“别打给我们，我们会打给你（don’t call us, we’ll call you）”。控制反转的概念最早在2004年由Martin Fowler提出，是针对面向对象设计不断复杂化而提出的设计原则，是一种利用面向对象编程法则来降低应用程序耦合的设计模式。IoC强调的是对代码引用的控制权由调用方转移到外部容器，在运行时通过某种方式注入进来，实现了控制的反转，这大大降低了服务类之间的耦合度。依赖注入是最常用的一种实现IoC的方式，另一种是依赖查找，感兴趣的同学可以自行了解。

## 什么是依赖注入

在软件工程中，依赖注入是种实现控制反转用于解决依赖性设计模式。一个依赖关系指的是可被利用的一种对象（即服务提供端）。依赖注入是将所依赖的传递给将使用的从属对象（即客户端）。该服务是将会变成客户端的状态的一部分。传递服务给客户端，而非允许客户端来建立或寻找服务，是本设计模式的基本要求。 --- 维基百科。

在依赖注入模式中，应用组件无需关注所依赖对象的创建或初始化过程，可以认为框架已经初始化好了，开发者只管调用即可。依赖注入有利于应用程序中各模块之间的解耦，使得代码更容易维护。这种优势可能一开始体现不出来，但随着项目复杂度的增加，各模块、组件、第三方服务等相互调用更频繁时，依赖注入的优点就体现得淋漓尽致。

注意，只有明白了依赖注入同学们才能创建可重用的组件，而只有知道创建可重用的组件，才能说我会Angular了。

## 依赖注入的优点

1. 使组件和服务松耦合（组件和服务的具体实现，以及服务的依赖都隔离开了，让组件有更高的重用性）。
2. 便于测试。LoginComponent -> MockLoginService

## Angular中依赖注入的构成

1. 依赖（Dependence）
2. 注入器（Injector）
3. 提供器（Provider）

依赖指定了被依赖对象的类型，注入器会根据此类型创建对应的对象。

注入器就像制造工厂，提供了一系列的接口用于创建依赖对象的实例。

提供器用于配置注入器，注入器通过它来创建被依赖对象的实例，Provider把标识（Token）映射到工厂方法，被依赖的对象就是通过该方法来创建的。

整体流程：在Angular中通过指定 Providers告诉Angular哪些对象需要依赖注入，组件或者服务在构造函数（constructor）中声明需要的依赖，这时注入器就会把实例好的对象拿给组件。

# Angular中依赖注入使用

注意：@Injectable() 标识一个类可以被注入器实例化。 通常，在试图实例化没有被标识为@Injectable()的类时，注入器会报错。我们建议为每个服务类都添加@Injectable()。因为你没法判断别人改写它的时候会不会在里面注入别的内容。此外例如@component装饰器都包含了@Injectable()，所以我们可以在组件中注入服务。

在Angular中只有一种注入，就是构造函数注入。

注入器默认是隐式创建，在Angular里通过TypeScript 的构造器语法来同时定义参数和属性。

## 在组件中注入服务

三种组件间共享数据的方式：@input/@Output、 路由参数、 共享服务。

在组件中使用依赖注入需要完成以下三个步骤：

1. 导入：通过import导入被依赖对象的服务。
2. 声明：在组件中配置注入器（默认隐式创建）。在启动组件时，Angular会读取@Component装饰器里的providers元数据，它是一个数组，配置了该组件需要使用到的所有依赖，Angular的依赖注入框架会根据这个列表去创建对应对象的实例。
3. 注入：在组件构造函数中声明需要注入的依赖。注入器会根据构造函数上的声明，在组件初始化时通过第二步中的providers元数据配置依赖，为构造函数提供对应的依赖服务，最终完成注入的过程。

步骤：

1. 创建服务： ng g s xxx
2. 在构造函数中来声明需要的服务： constructor(private contactService: ContactService)
3. 在模块或者组件中提供注入： providers: [ContactService]
4. 直接来使用服务。

## 在服务中注入服务

道理同理，参见loggerService 被注入到contactService中

## 在模块中注入服务

我们可以把Providers 的声明提升到模块中。

Angular里建议把Providers声明放置在根模块中。

## 层级注入关系

应用级的注入器. -> 主组件注入器 -> 子组件注入器

不存在模块级的注入器。我们注入在根模块中的注入器就是应用级的。

上面上个注入器，越靠后越优先执行。

## 限定方式的依赖注入

到目前为止，注入都是假定依赖对象是存在的，然而实际情况往往并非如此，比如上层提供的Provider被移除，导致了之前注入的依赖可能已经不存在了，此时再按照前面讲的依赖注入方式进行相关服务的调用，应用就会出错。这是就要用到限定方式。

Angular的限定注入方式使得开发者能够修改默认的依赖查找规则，@Optional可以兼容依赖不存在的情况，提高系统的健壮性。@Host 可以限定查找规则，明确实例初始化的位置，避免一些莫名的共享对象问题。（查找到当前元素的宿主元素为止）@Self 限定依赖只能自己提供。@ SkipSelf 限定依赖从父注入器开始。

# 提供器（Provider）

提供器由两部分组成第一个是令牌 (token)，它作为键值 (key) 使用，用于定位依赖值和注册提供商。第二个是一个提供商定义对象。 可以把它看做是指导如何创建依赖值的配方。 有很多方式创建依赖值…… 也有很多方式可以写配方。

## 类Provider

用法：{

provide: ContactService,

useClass: AnotherContactService

}

## 值Provider

在 Angular 中，provider 的 token 的类型可以是字符串或 Type 类型。我们可以根据实际应用场景，选用不同的类型。只要我们知道依赖对象的类型，我们就可以方便地注入对应类型的实例对象。但是有时候，我们需要注入的是普通的JavaScript对象，而不是Type 类型的对象。比如，我们需要注入一个config对象，或者有时候我们需要注入一个原始数据类型的数值，如字符串或布尔值。在这种情况下，我们是不能使用 String 或 Boolean 类型，因为如果使用这些类型，我们只能获得类型对应的默认值。想解决这个问题，但我们又不想引入一种新的类型来表示原始数据类型的值。这时我们可以考虑使用字符串作为 token，而不用引入新的类型。

定义：{ provide: 'NAME', useValue: "Derry"}

用法在构造函数中：@Inject('NAME') name: string

问题：当第三方也有一个叫’NAME’的提供器的时候，这样子就会起冲突。这个时候有我们用InjectionToken来解决这个问题。InjectionToken还是会使用相同的字符串去创建实例对象，但是每次返回都是一个新的实例。从而保证了全局的唯一性。

用法： const NAME\_URL = new InjectionToken<string>('NAME');

{ provide: NAME\_URL, useValue: "Derry"}

InjectionToken： 是Angular 4.x使用的方法。

OpaqueToken： 是Angular 2.x 用法。

## 别名Provider

假设某个旧组件依赖一个OldLogger类。 OldLogger和NewLogger具有相同的接口，但是由于某些原因， 我们不能升级这个旧组件并使用它。当旧组件想使用OldLogger记录消息时，我们希望改用NewLogger的单例对象来记录。不管组件请求的是新的还是旧的日志服务，依赖注入器注入的都应该是同一个单例对象。 也就是说，OldLogger应该是NewLogger的别名。

例如:

{ provide: NewLoggerService, useClass: NewLoggerService },

{ provide: OldLoggerService, useExisting: NewLoggerService },

## 工厂Provider

当服务对象需要参数条件的时候使用工厂提供器。

定义： 通过 useFactory 来定义工厂方法。

原理：useFactory根据我们不同条件和参数返回一个 useClass

Deps：里面定义了需要传给 factory方法的参数的实例。

## Provider扩展

Multi providers 让我们可以使用相同的 Token 去注册多个 Provider我们可以使用相同的 token 值，注册不同的 provide。当我们使用对应的 token 去获取依赖项时，我们获取的是已注册的依赖对象列表。

# ForwardRef

Angular 通过引入 forwardRef 让我们可以在使用构造注入时，使用尚未定义的依赖对象类型。

示例：

@Injectable()

class Socket {

constructor(@Inject(forwardRef(() => Buffer))

private buffer) { }

}

class Buffer {

constructor(private size: Number) { }

}

# 查找父组件

在组件的课程种，我们讲了父组件获取子组件引用的方法（通过@ViewChildren的方式）。反过来，在子组件获取父组件的实例就相对麻烦一些，考虑到每个组件的实例都会添加到注入器的容器里，因此可通过依赖注入来找到父组件的实例。

有两种情况：

1. 已知父组件类型
2. 未知父组件类型：用到一种方式叫作‘类-接口’的方式

# 拓展

## 在其他IT技术中提供器的存在

在.Net中有MembershipProvider

在Hibernate里有 ConnectionProvider

在Android中有contentProvider。

Provider提供了很好的可扩展性和灵活性。

## Angular中依赖注入的层级结构原理

Angular 中注入器是有层级结构的，即创建完注入器，我们可以基于它创建它的子注入器。

resolveAndCreate() - 根据设置的 provider 数组创建注入器

resolveAndCreateChild() - 调用已有注入器对象上的该方法，创建子注入器

当调用注入器 get() 方法，获取 token 对应的对象时，默认的查找方式是，优先从本级注入级获取，如果未找到，则往上一级查找，直到根级注入器。若未找到对应的依赖对象，默认会抛出异常。