

# dagger2 让你爱不释手：重点概念讲解、融合篇

2016-04-09 安卓应用频道

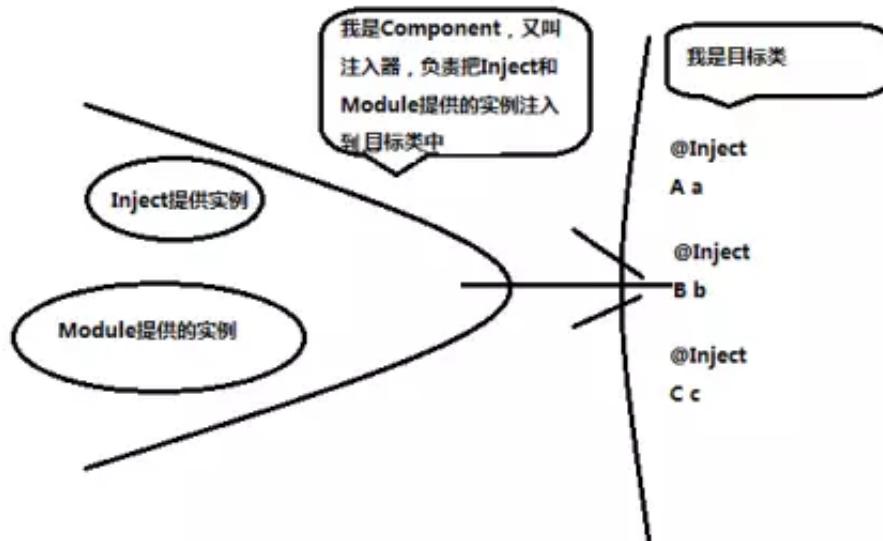
(点击上方公众号，可快速关注)

来源：伯乐在线 - 牛犇

链接：<http://android.jobbole.com/82704/>

## 前言

《dagger2 让你爱不释手：基础依赖注入框架篇》这篇讲解了Inject , Component , Module , Provides是如何构成dagger2整个依赖注入框架的



*component\_module\_inject.png*

因为dagger2的整个依赖注入框架已经构建完成，所以dagger2中剩下的Qualifier ( 限定符 ) 、 Singleton ( 单例 ) 、 Scope ( 作用域 ) ,SubComponent概念基本都是在对整个依赖注入框架进行细节上的完善。

我还是依然从抽象概念的角度讲解，讲解每个概念在整个依赖注入框架中到底起了什么作用，因为dagger2本身不容易上手，只有真正的了解了每个概念的作用，在使用时才会得心应手，大家别急，后面的章节会有dagger2的sample。

## 本节内容

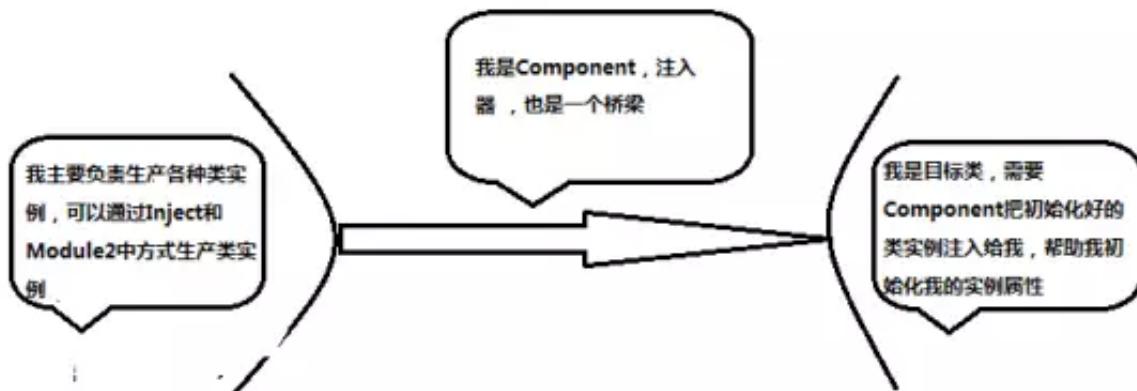
- Qualifier ( 限定符 ) 、 Singleton ( 单例 ) 、 Scope ( 作用域 ) 、 Component的组织方式概念讲解

- dagger2能带来哪些实惠 ?

在讲解时，我还依然沿用上一节的讲解方式，由简入难不断深入的进行。

## Qualifier ( 限定符 ) 是什么鬼？

上一节已经提到，Component是一个注入器（ Injector ），同时也起着桥梁的作用，一端是创建类实例端（ 创建类实例即负责生产类实例，下面会用该词来指代 ），另一端是目标类端（ 目标类需要进行依赖初始化的类，下面都会用目标类一词来指代 ），请看下图：



新的关系.png

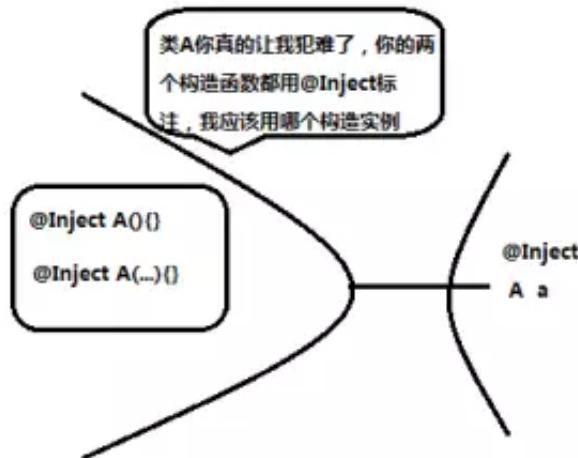
创建类实例有2个维度可以创建：

- 通过用Inject注解标注的构造函数来创建（以下简称Inject维度）
- 通过工厂模式的Module来创建（以下简称Module维度）

这2个维度是有优先级之分的，Component会首先从Module维度中查找类实例，若找到就用Module维度创建类实例，并停止查找Inject维度。否则才是从Inject维度查找类实例。所以创

建类实例级别Module维度要高于Inject维度。

现在有个问题，基于同一个维度条件下，若一个类的实例有多种方法可以创建出来，那注入器（Component）应该选择哪种方法来创建该类的实例呢？如下图，基于Inject维度：



Qualifier\_迷失.png

我把上面遇到的问题起个名字叫依赖注入迷失。

那么可以给不同的创建类实例的方法用标识进行标注，用标识就可以对不同的创建类实例的方法进行区分（标识就如给不同的创建类实例方法起了一个id值）。同时用要使用的创建类实例方法的标识对目标类相应的实例属性进行标注。那这样我们的问题就解决了，提到的标识就是Qualifier注解，当然这种注解得需要我们自定义。

Qualifier（限定符）就是解决依赖注入迷失问题的。

## 注意

dagger2在发现依赖注入迷失时在编译代码时会报错。

Scope（作用域）你真是挺坑的一个东东

我们暂且不介绍Singleton，因为它是Scope的一个默认实现，理解了Scope自然就理解Singleton了。

为什么要说Scope比较坑呢，在刚开始接触Scope的时候，看了网上各种关于Scope的介绍，总结Scope的作用是：

Dagger2可以通过自定义Scope注解，来限定通过Module和Inject方式创建的类的实例的生命周期能够与目标类的生命周期相同。或者可以这样理解：通过自定义Scope注解可以更好的管理创建的类实例的生命周期。

网上也有各种例子比如：自定义一个PerActivity注解，那创建的类实例就与Activity“共生死”。

或者用Singleton注解标注一个创建类实例的方法，该创建类实例的方法就可以创建一个唯一的类实例。

我对PerActivity和Singleton这些魔法性的注解产生了好奇，同时也产生了迷惑？迷惑是：

- 自定义Scope注解到底是怎么工作的
- 自定义的注解应该怎么定义名字，是不是定义一个名字就可以达到相应名字的效果。比如Singleton就可以实现单例，PerActivity就可以创建的类实例与Activity“共生死”，是不是我定义一个PerFragment的注解，同样可以达到创建的类实例就与Fragment“共生死”。大家别对我这幼稚的想法千万别见笑，当时我就把dagger2的Scope注解想的如此神通广大了

于是乎我在网上进行各种搜索，并且分析源码，最后得到的结果也是让我大吃一惊。自定义的Singleton、PerActivity注解根本就没有这些功能。所以也可以说我被Scope坑了，或者是由于自己没有对Scope有一个深入的理解，被自己坑了。这先卖个关子，后面会具体介绍Scope。

## Component组织方式重点中的重点

为什么说Component组织方式是重点中的重点呢？因为前面的各种概念都是在做铺垫工作，现在我们会从一个app的角度来把这些概念融合在一起。

一个app中应该根据什么来划分Component？

假如一个app（app指的是Android app）中只有一个Component，那这个Component是很难维护、并且变化率是很高，很庞大的，就是因为Component的职责太多了导致的。所以就有必要把这个庞大的Component进行划分，划分为粒度小的Component。那划分的规则这样

的：

- 要有一个全局的Component(可以叫ApplicationComponent),负责管理整个app的全局类实例（全局类实例整个app都要用到的类的实例，这些类基本都是单例的，后面会用此词代替）
- 每个页面对应一个Component，比如一个Activity页面定义一个Component，一个Fragment定义一个Component。当然这不是必须的，有些页面之间的依赖的类是一样的，可以公用一个Component。

第一个规则应该很好理解，具体说下第二个规则，为什么以页面为粒度来划分Component？

- 一个app是由很多个页面组成的，从组成app的角度来看一个页面就是一个完整的最小粒度了。
- 一个页面的实现其实是要依赖各种类的，可以理解成一个页面把各种依赖的类组织起来共同实现一个大的功能，每个页面都组织着自己的需要依赖的类，一个页面就是一堆类的组织者。
- 划分粒度不能太小了。假如使用mvp架构搭建app，划分粒度是基于每个页面的m、v、p各自定义Component的，那Component的粒度就太小了，定义这么多的Component，管理、维护就很非常困难。

所以以页面划分Component在管理、维护上面相对来说更合理。

### Singleton没有创建单例的能力

为什么要谈到创建单例呢？因为上面谈到一个app要有一个全局的Component（我们暂且叫ApplicationComponent），ApplicationComponent负责管理整个app用到的全局类实例，那不可否认的是这些全局类实例应该都是单例的，那我们怎么才能创建单例？

上一节提到过Module的作用，Module和Provides是为解决第三方类库而生的，Module是一个简单工厂模式，Module可以包含创建类实例的方法

现在Module可以创建所以类的实例。同时

Component会首先从Module维度中查找类实例，若找到就用Module维度创建类实例，并停止查找Inject维度。否则才是从Inject维度查找类实例。所以创建类实例级别Module维度要高于Inject维度。

所以利用以上2点，我们就可以创建单例。

- 在Module中定义创建全局类实例的方法
- ApplicationComponent管理Module
- 保证ApplicationComponent只有一个实例（在app的Application中实例化ApplicationComponent）

dagger2中真正创建单例的方法就是上面的步骤，全局类实例的生命周期也和Application一样了，很关键的一点就是保证ApplicationComponent是只初始化一次。那估计有朋友就会问Singleton那岂不是多余的？

答案当然是 no no no。 Singleton有以下作用：

- 更好的管理ApplicationComponent和Module之间的关系，保证ApplicationComponent和Module是匹配的。若ApplicationComponent和Module的Scope是不一样的，则在编译时报错。
- 代码可读性，让程序猿更好的了解Module中创建的类实例是单例。

## 组织Component

我们已经把一个app按照上面的规则划分为不同的Component了，全局类实例也创建了单例模式。问题来了其他的Component想要把全局的类实例注入到目标类中该怎么办呢？这就涉及到类实例共享的问题了，因为Component有管理创建类实例的能力。因此只要能很好的组织Component之间的关系，问题就好办了。具体的组织方式分为以下3种：

## 依赖方式

一个Component是依赖于一个或多个Component，Component中的dependencies属性就是依赖方式的具体实现

## 包含方式

一个Component是包含一个或多个Component的，被包含的Component还可以继续包含其他的Component。这种方式特别像Activity与Fragment的关系。SubComponent就是包含方式的具体实现。

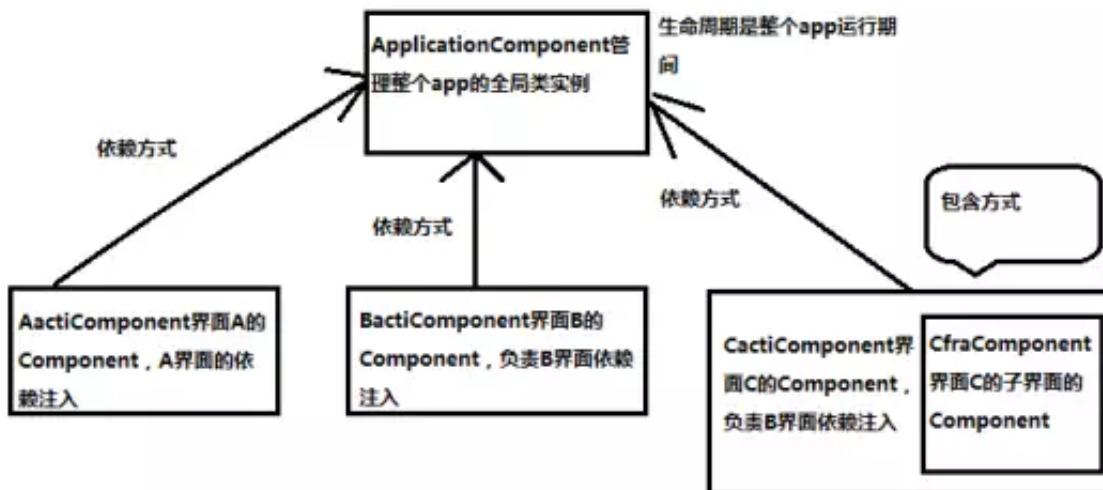
## 继承方式

官网没有提到该方式，具体没有提到的原因我觉得应该是，该方式不是解决类实例共享的问题，而是从更好的管理、维护Component的角度，把一些Component共有的方法抽象到一个父类中，然后子Component继承。

## Scope真正用武的时候了

前面也提到Scope的一些基本概念，那Scope的真正用处就在于Component的组织。

- 更好的管理Component之间的组织方式，不管是依赖方式还是包含方式，都有必要用自定义的Scope注解标注这些Component，这些注解最好不要一样了，不一样是为了能更好的体现出Component之间的组织方式。还有编译器检查有依赖关系或包含关系的Component，若发现有Component没有用自定义Scope注解标注，则会报错。
- 更好的管理Component与Module之间的匹配关系，编译器会检查 Component管理的 Modules，若发现标注Component的自定义Scope注解与Modules中的标注创建类实例方法的注解不一样，就会报错。
- 可读性提高，如用Singleton标注全局类，这样让程序猿立马就能明白这类是全局单例类。



app的结构.png

## 总结

关于dagger2概念性的东西基本都已经介绍完毕，剩下的比如Lazy、Provide等注解就不做介绍了，它们太简单了。同时也着重介绍了Scope，Qualifier等概念。还从整个app的角度来分析Component的组织方式。希望对大家能有帮助，因为dagger2上手还是比较复杂的，其实关键一点就是对于各种概念性的东东不了解，不知道它们到底有啥用途。所以我希望能帮到初学者对dagger2有一个整体性概念性的了解，然后在看网上的例子时能神清气爽。

---

## 安卓应用频道

专注分享安卓应用相关内容



微信号：AndroidPD



长按识别二维码关注

---

伯乐在线旗下微信公众号

商务合作QQ：2302462408

---