

硕 士 研 究 生 读 书 报 告



题目 Dagger2研究与分析

作者姓名 杨洁

作者学号 21651146

指导教师 李启雷

学科专业 软件工程

所在学院 软件学院

提交日期 二○一七年 一 月

The Research and Analysis Of Dagger2

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Li Qilei

By

Yang Jie

Zhejiang University, P.R. China

2017

摘要

本文重点探讨了Android 平台的依赖注入框架Dagger2。依赖注入是一种控制反转的实现方式，可以实现代码的低耦合，且能够容易地进行测试。虽然依赖注入早已作为一种设计模式为人所知，但到了近段时间它才被广泛应用到 Android 应用的开发中，主要是因为最近才出现了各种各样优秀的依赖注入框架。Dagger2诞生于开发者对Dagger1半静态化半运行时的不满，改进后的Dagger框架可以静态生成完整的依赖图，实现完全的代码依赖生成。Dagger2的目标是将MVP模式中的VP进一步解耦，达到模块化最大解耦，使得代码更容易维护。

文章首先介绍了IoC模式，进而对依赖注入做了一个简要概述并简述其重要性。接下来对Dagger2框架的实现、工作流程、性能优化做了详细介绍。最后重点介绍了Dagger2的使用规则及实现流程。

**关键词**：Android，IoC ，依赖注入 ，Dagger2 ，Java注解

Abstract

The paper discusses the Dependency Injection frame in Android platform——Dagger2. Dependency Injection is a way to realize Inversion of Control which help reducing code coupling and to test easily. Although dependency injection has been around as a kind of design pattern for a long time, it is used widely in Android just a short time because recently appeared various dependency injection frame. Dagger2 originated in the developers’ discontentment for the Semi static semi operation of Dagger1.After improvement, Dagger2 can static generate dependency graph

and realize complete code dependency. It aims to decouple V and P in MVP pattern further more, in order to achieve maximum decoupling in module and make code easy to maintain.

The paper introduce IoC firstly and then make a brief introduction of dependency injection and sketch the importance of it. Next, the paper introduces the realization, workflow, performance optimization in detail. Finally, the paper focuses on the use regulation and implementation process.

**Keywords：**Android，IoC ，Dependency Injection ，Dagger2 ，Java annotation

1引言

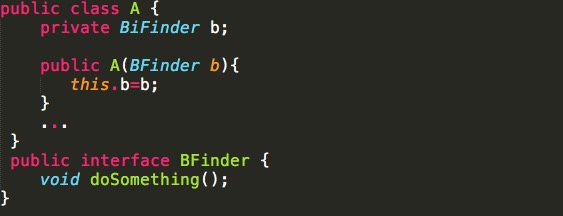
面向对象的思想已经深入人心，但是要利用面向对象的思想开发出优秀的应用程序却不是一件容易的事情。在企业级 Java的世界里 ,开发者常遇到的一个 问题就是如何组装不同的程序元素:如果Web控制器体系结构和数据库接口是由 不同的团队所开发的,彼此几乎一无所知,你应该如何让它们配合工作?很多框架尝试过解决这个问题,有几个框架索性朝这个方向发展,提供了更通用的“组装各 层组件”的方案这样的框架通常被称为“轻量级容器”。在这些容器背后，一些设计模式发挥着作用，其中IoC模式更是广泛应用于各种应用程序框架中，将组件的配置进行了倒置，不再由组件自己来建立这种依赖关系而交给容器去管理。

**1.1 IoC模式简介**

RobertC. Martin在其敏捷软件开发中描述的依赖倒置原则DIP(Dependency Inversion Principle) ，其含义为“不用向容器要资源，容器会自动给你所需要的”。依赖注入（Dependency Injection）是Martin Flower 对IoC模式的一种扩展解释。IoC 是一种用来解决组件（实际上也可以是简单的Java类）之间依赖关系、配置及生命周期的设计模式，其中对组件依赖关系的处理是 IoC 的精华部分。早在2004年，[Martin Fowler](https://zh.wikipedia.org/wiki/Martin_Fowler" \o "Martin Fowler)就提出了“哪些方面的控制被反转了？”这个问题。他总结出是依赖对象的获得被反转了，因为大多数应用程序都是由两个或是更多的类通过彼此的合作来实现[业务逻辑](https://zh.wikipedia.org/w/index.php?title=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&action=edit&redlink=1" \o "业务逻辑（页面不存在）)，这使得每个对象都需要获取与其合作的对象（也就是它所依赖的对象）的引用。如果这个获取过程要靠自身实现，那么这将导致代码高度[耦合](https://zh.wikipedia.org/wiki/%E8%80%A6%E5%90%88%E5%8A%9B_(%E8%A8%88%E7%AE%97%E6%A9%9F%E7%A7%91%E5%AD%B8)" \o "耦合力 (计算机科学))并且难以维护和调试。

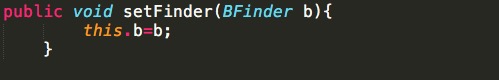
**1.2依赖注入**

依赖注入作为控制反转的一种实现方式，实际上就是将实例变量传入到一个对象中。在实际应用场景中，如果Class A中用到了Class B的对象b，一般情况下，需要在A的代码中显式的new一个B的对象。然而这种设计是有很大问题的，A 类直接依赖于 B 类，这种依赖关系意味着当B 类发生修改时，A类也会受到影响。这显然违反了“高层模块不应该依赖于低层模块，两者都应该依赖于抽象；抽象不应该依赖于具体实现，细节应该依赖于抽象”这一原则（DIP原则）。这种在一个类中直接创建另一个类的对象的代码，和硬编码（hard-coded strings）以及硬编码的数字（magic numbers）一样，是一种导致耦合的坏味道，我们可以把这种坏味道称为硬初始化（hard init）。而依赖注入技术，不是直接new来获得这个对象，而是通过相关的容器控制程序来将对象在外部new出来并注入到A类里的引用中。一般情况下，我们有如下三种实现依赖注入的方式：

1. 构造函数注入 

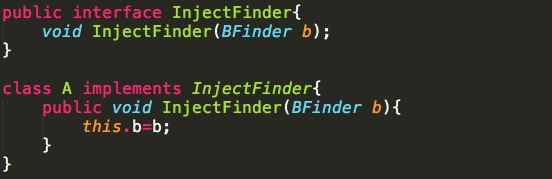
A只依赖于我们定义的BFinder接口，而不依赖于BFinder的实现。

1. setter注入



类似的，增加一个setter函数来创建BFinder对象，这样同样可以避免在A中硬初始化这个对象。

1. 接口注入



创建一个注入要使用的接口，在A中实现这个接口。

也就是说我们不再去显示的创建某个组件，让组件的创建、配置及声明周期总是由外部容器来管理，让容器将A类、B类及B类实现类组装起来。

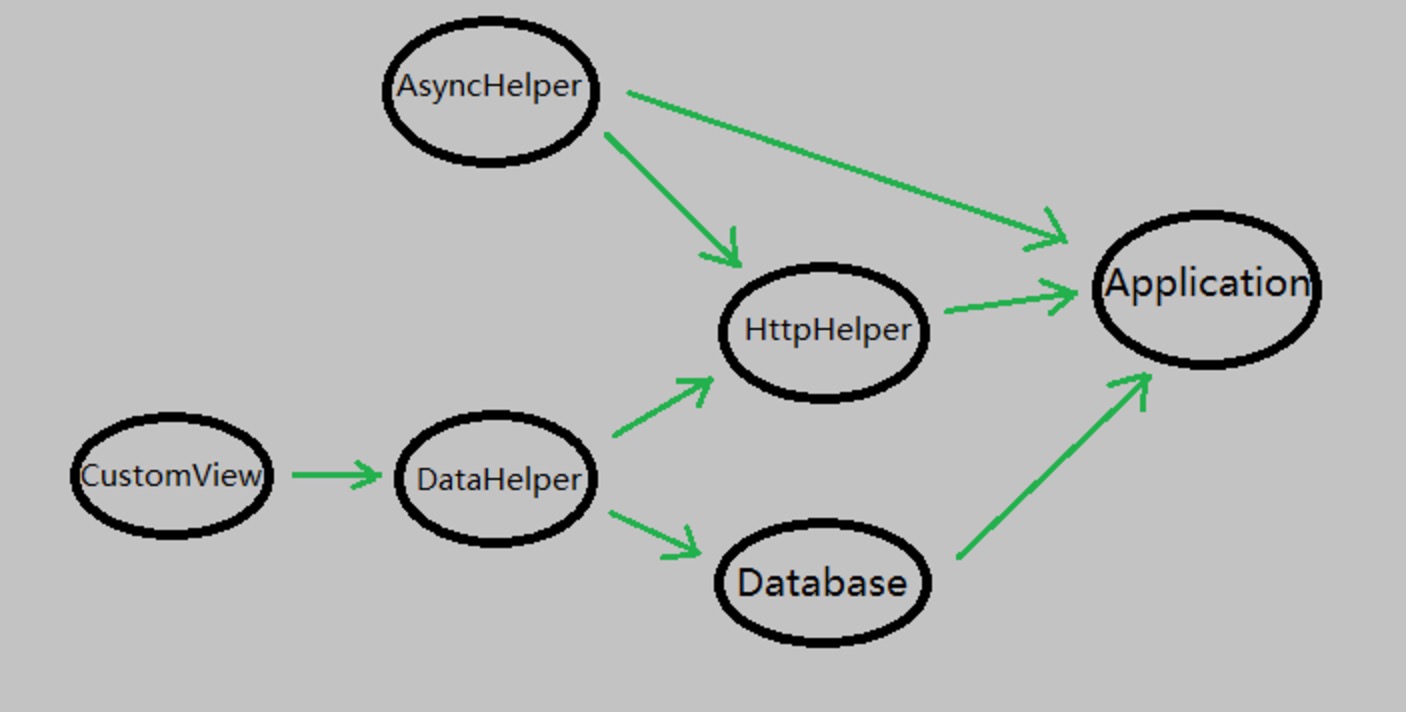
依赖注入的实现有多种途径，而在Java 中，使用注解是最常用的。通过在字段中添加 @Inject 注解进行标记，来实现依赖对象的自动注入。这听起来很神奇，只是增加了一个注解，依赖对象就能自动注入了吗？显然不是，我们还需要一个依赖注入框架，并进行简单的配置。现在 Java 语言中较流行的依赖注入框架有 [Google Guice](https://github.com/google/guice)、[Spring](http://projects.spring.io/spring-framework/) 等，而在Android 平台中，开发者们对Dagger1的半静态半运行时的不满诞生了Dagger2，它能够静态生成依赖图，实现完全的依赖代码生成式，这些特性都让Dagger2获得开发者特性，广泛应用于各种应用程序的实现中。

**2 Android 依赖框架Dagger2**

**2.1 Dagger简介**

Dagger2是一个Android依赖注入框架，由谷歌开发，最早的版本Dagger1 由Square公司开发。依赖注入框架主要用于模块间解耦，提高代码的健壮性和可维护性。Dagger 这个库的取名不仅仅来自它的本意“匕首”，同时也暗示了它的原理。Jake Wharton 在对 Dagger 的介绍中指出，Dagger 即 DAG-er，这里的 DAG 即数据结构中的 DAG——有向无环图(Directed Acyclic Graph)。也就是说，Dagger 是一个基于有向无环图结构的依赖注入库，因此Dagger的使用过程中不能出现循环依赖。

Dagger 的运作机制，是运用 [APT(Annotation Process Tool)](http://docs.oracle.com/javase/7/docs/technotes/guides/apt/) 在编译时生成一些用于设定规则的代码，然后在运行时将这些规则进行动态组合。一个程序中的整套依赖关系其实就是一个 DAG。而实际上，Dagger 也是这么做的：预先建立一个 DAG，然后在需要获取对象的时候通过这个依赖关系图来获取到对象并返回，若获取失败则进行查找，查找到后再补充到 DAG 中，过程如下：



Dagger 是支持传递依赖的。例如在上图中，当需要获取一个 CustomView，会首先获取一个 DataHelper 作为获取 CustomView 的必要参数；此时如果 DataHelper 还未初始化，则还要分别拿到 HttpHelper 和 Database 用来初始化 DataHelper；以此类推。Dagger 不支持循环依赖，即依赖关系图中不能出现环。

**2.2 注解介绍**

Dagger2 通过注解来生成代码，定义不同的角色，主要的注解有：@Inject、@Module 、@Component 、@Provides 、@Scope 、@SubComponent 等

1. Inject

通常在需要依赖的地方使用这个注解。可以用注解(Annotation)来标注目标类中所依赖的其他类，同样用注解来标注所依赖的其他类的构造函数,换句话说，你用它告诉Dagger这个类或者字段需要依赖注入。这样，Dagger就会构造一个这个类的实例并满足他们的依赖。这样我们就可以让目标类中所依赖的其他类与其他类的构造函数之间有了一种无形的联系。但是要想使它们之间产生直接的关系，还得需要一个桥梁来把它们之间连接起来。那这个桥梁就是Component了。

1. Module

当项目中使用到第三方类库时，并不能使用@inject修改，封装好的第三方类库可以使用可以使用Module进行管理，Module其实是一个简单工厂模式，里面的方法基本都是创建类实例的方法。同样也需要component将实例与Module进行连接。

1. Component

Component是注入器，它一端连接目标类，另一端连接目标类依赖实

例，它把目标类依赖实例注入到目标类中。上文中的Module是一个提供类实例的类，所以Module应该是属于Component的实例端的（连接各种目标类依赖实例的端），Component的新职责就是管理好Module，Component中的modules属性可以把Module加入Component，modules可以加入多个Module。

1. Provides

Module中的创建类实例方法用Provides进行标注，Component在搜索到目标类中用Inject注解标注的属性后，Component就会去Module中去查找用Provides标注的对应的创建类实例方法，这样就可以解决第三方类库用dagger2实现依赖注入了。

1. Scope

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。

1. Qualifier

当类的类型不足以鉴别一个依赖的时候，我们就可以使用这个注解解

示。例如：在Android中，我们会需要不同类型的context，所以我们就可以定义 qualifier注解“@ForApplication”和“@ForActivity”，这样当注入一个context的时候，我们就可以告诉 Dagger我们想要哪种类型的context。

**2.3 Component 组织方式**

（1）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

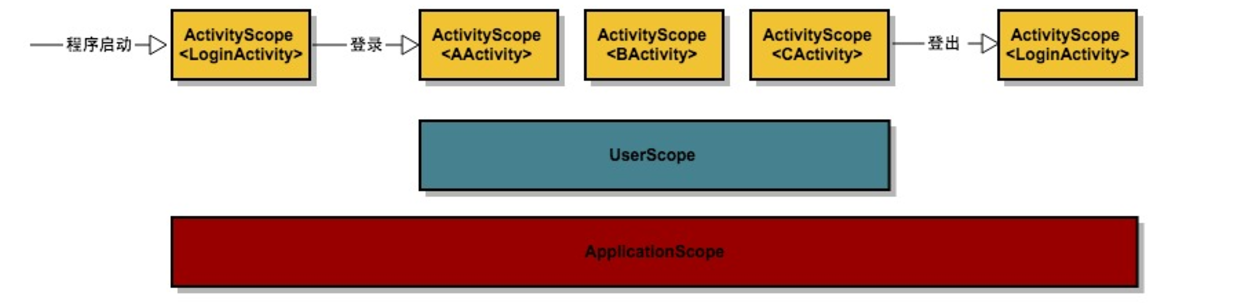
（2）包含方式  
 一个Component是包含一个或多个Component的，被包含的Component还可以继续包含其他的Component。这种方式特别像Activity与Fragment的关系。SubComponent就是包含方式的具体实现。

（3）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。

* 1. **自定义Scope**

在Dagger 2中，Scope机制可以使得在scope存在时保持类的单例。在实践中，这意味着被限定范围为@ApplicationScope的实例与Applicaiton对象的生命周期一致。@ActivityScope保证引用与Activity的生命周期一致。简单来说scope给我们带来了“局部单例”，生命周期取决于scope自己。

Scope的实现归结于对Components的一个正确的设置。一般情况下我们有两种方式:使用Subcomponent注解或者使用Components依赖。它们两者最大的区别就是对象图表的共享。Subcomponents可以访问它们parent的所有对象图表，而Component依赖只能访问通过Component接口暴露的对象。假设如下应用场景：有个项目包含用户体系，用户登录成功后，A界面、B界面和C界面要依赖用户来获取一些数据，LoginActivity界面不依赖于用户体系。  
我们想要User对象实例可以在A、B、C界面共用。那么整体项目的scope划分为：



实现方式如下：

1. 自定义UserScope注解



1. 新建UserModule来提供User的实例,提供实例方法使用自定义的UserScope注解，表示提供实例仅限于UserScope范围内使用。



1. 新建Component桥梁。他是一个子Component,依赖于一个全局的父Component。



自定义Scope的作用很明显：

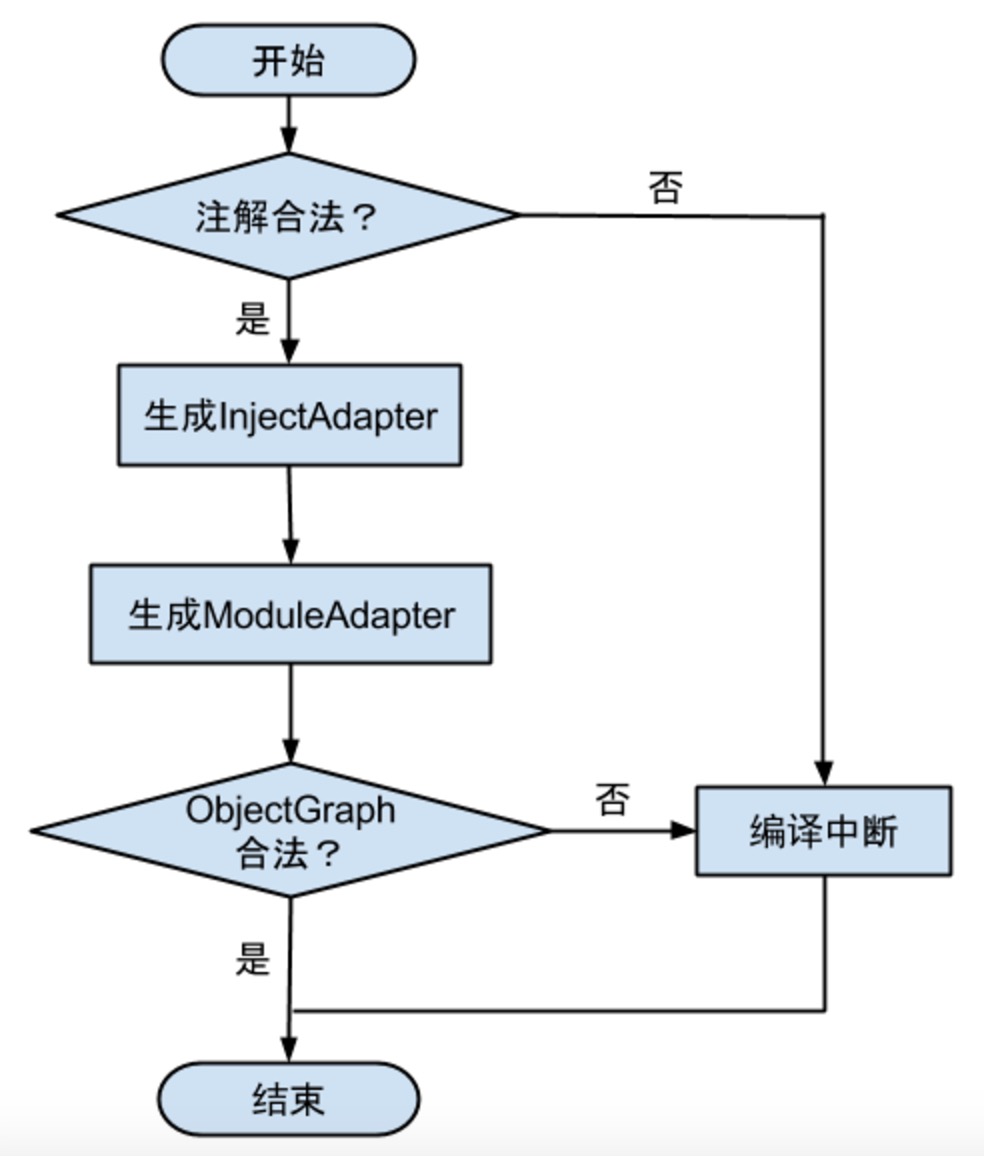
（1）更好的管理Component之间的组织方式，不管是依赖方式还是包含方式，都有必要用自定义的Scope注解标注这些Component，这些注解最好不要一样了，不一样是为了能更好的体现出Component之间的组织方式。还有编译器检查有依赖关系或包含关系的Component，若发现有Component没有用自定义Scope注解标注，则会报错。

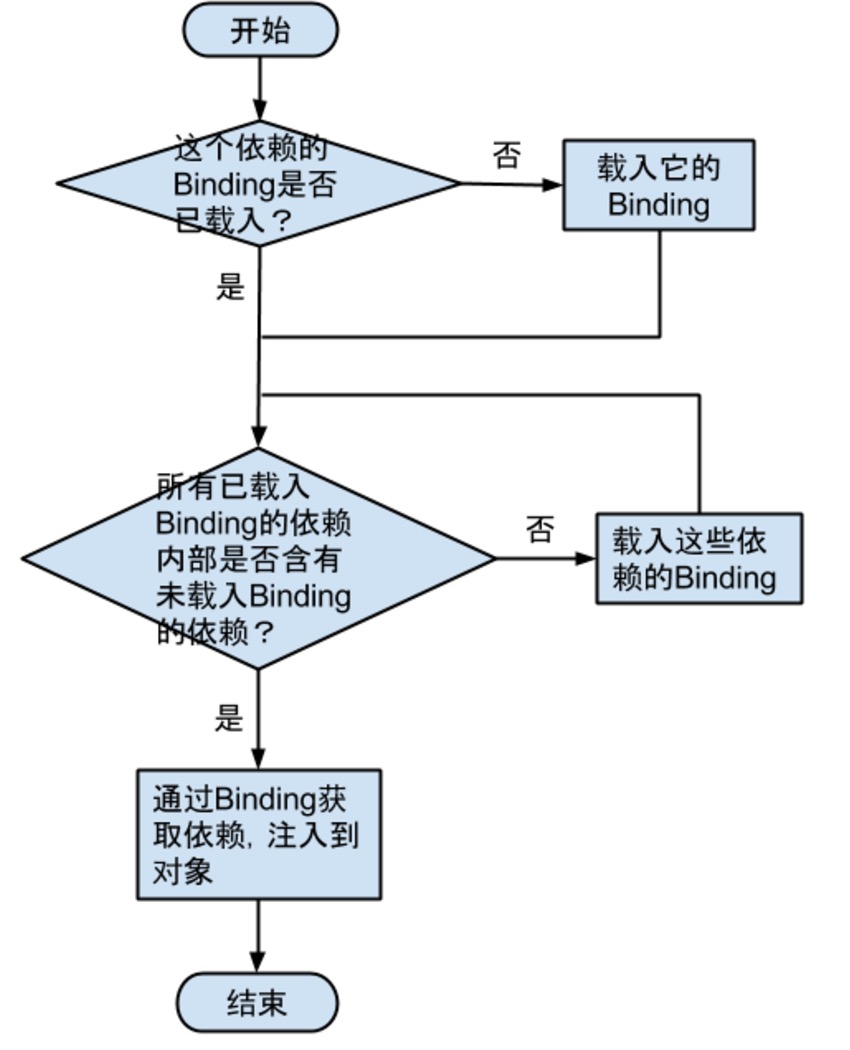
（2）更好的管理Component与Module之间的匹配关系，编译器会检查 Component管理的Modules，若发现标注Component的自定义Scope注解与Modules中的标注创建类实例方法的注解不一样，就会报错。

（3）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。

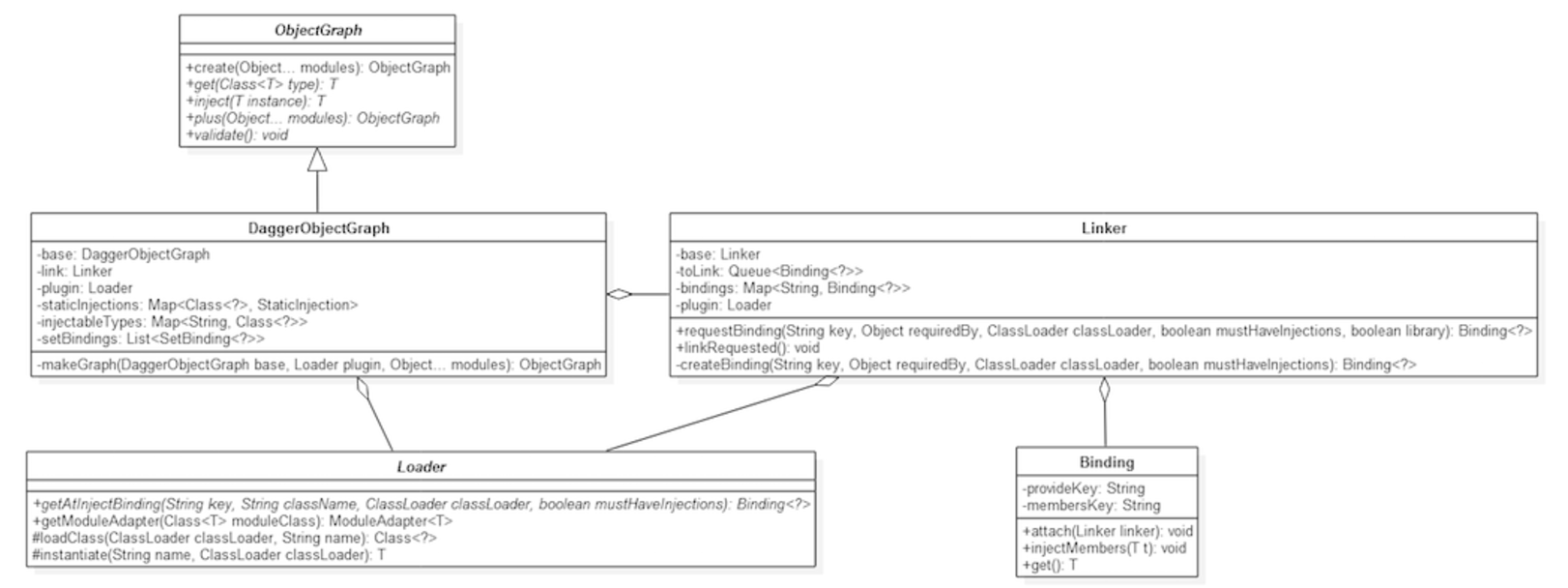
1. **Dagger 性能分析**

**3.1 Dagger 工作流程解析**



编译时，通过 APT 查看所有 java 文件，并根据注解生成一些新的 java 文件，即InjectAdapter、ProvidesAdapter、ModuleAdapter，这些文件用于运行时辅助 DAG 的创建和完善。然后，将这些新生成的 java 文件和项目原有的 java 文件一并编译成 class 文件。 运行时，在 Application 或某个具体模块的初始化处，使用ObjectGraph类来加载部分依赖(实质上是利用编译时生成的ModuleAdapters加载了所有的ProvidesBinding)，形成一个不完整的依赖关系图。  
 这个不完整的依赖关系图生成之后，就可以调用ObjectGraph的相应函数来获取实例和注入依赖了。实现依赖注入的函数有两个：(1)ObjectGraph.get(Class<T> type)函数，用于直接获取对象；(2)ObjectGraph.inject(T instance)函数，用于对指定对象进行属性的注入。

在这些获取实例和注入依赖的过程中，如果用到了还未加载的依赖，程序会自动对它们进行加载(实质上是加载的编译时生成的InjectAdapter)。在此过程中，内存中的 DAG 也被补充地越来越完整。类关系图如下：



大致原理可以描述为：Linker通过Loader加载需要的Binding并把它们拼装成合理的依赖关系图 ObjectGraph，由ObjectGraph(其子DaggerObjectGraph)最终实现依赖注入的管理。ObjectGraph 是个抽象类，DaggerObjectGraph 是它目前唯一的子类，对 Dagger 的调用实际都是对 DaggerObjectGraph 的调用。

* 1. **Dagger2 优化介绍**

Dagger是由[Square](https://corner.squareup.com/)公司的几个十分优秀的开发人员在2012年创建的，他们觉得将依赖注入应用到Java中是一个非常棒的点子，但是却觉得当时使用的[Guice](https://github.com/google/guice)库（当时的标准）的速度稍微有点慢。所以，他们开发了一个依赖于基于注解的代码生成（JSR-330），拥有着和Guice相似的API，但是性能更强、拥有更加灵活的库。Dagger的工作原理就是通过声明一些module，这些module里包括了所有你想要注入的依赖的提供者方法（provider method）， 把这些module加载到一个对象图谱（object graph）里，最终将图谱中的内容根据需要注入到对象（target）中。 足够简单的结构（当然实现起来还是不那么简单的）有利于开发者解耦他们的代码,并且通过把初始化代码移动到由库自动生成的injector中,从而删掉每个类开始的地方那些丑陋的工厂类初始化代码。

Dagger2在Dagger1的基础上有了更大的改进，实现在编译时静态生成依赖图，其工作流程与上文中提到的过程类似，但是放弃了反射机制，在编译阶段将完成依赖图的工作全部完成。Dagger2与Dagger1的对比如下。

Dagger1存在的主要问题：(1)图谱（Graph）是在运行时构建的 -影响性能，尤其是在频繁请求的用例中（后台服务器场景）；(2) 使用了反射，比如说 用Class.forName()来获取已经生成的类型，使得生成的代码难以跟踪，同时ProGuard配置文件的编写成了噩梦。(3) 生成的代码不美观 ，特别是与那些人手写的工厂类初始化方法相比较时，这一点尤其明显。

时间快进到现在，Google的core libraries团队（[Guava](https://github.com/google/guava)的创造者）与原来的Square的创建者一起推出了新一代的Dagger，Dagger2解决了老版本的许多问题：(1) 不再使用反射 所有东西都是通过明确的方法调用来完成的（不需要配置ProGuard文件就可以正常混淆）。(2) 不再在运行时构建图表（graph） - 提高了性能，包括在单请求（per-request）用例中（据[Gregory Kick](https://plus.google.com/+GregoryKick)说，在谷歌的搜索产品中，它的表现比以前快了13%）(3) 可追溯的 - 生成的代码更优雅，同时没有使用到反射，使得代码可读性提高，并且容易跟踪

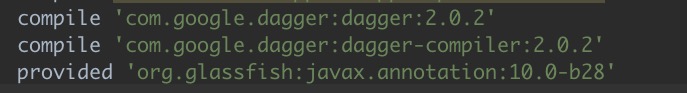
**4 Dagger2 代码实现**

**4.1在Android Studio中实现流程**

**(1)**在工程根目录的build.gradle引入apt插件

/Users/yangjie/Downloads/IMG_2111.JPG

(2) 在app目录下引入



(3) 在Java 代码中可以通过两个维度来创建类实例：

1. 通过用Inject注解标注的构造函数来创建，使用Inject注解标注目标类中其他类，并用Inject注解标注其他类的构造函数，调用Component（注入器）的injectXXX（Object）方法开始注入： 
2. 通过工厂模式的Module来创建，使用module注解，项目中使用到了第三方的类库，第三方类库又不能修改，所以根本不可能把Inject注解加入这些类中，这时我们的Inject就失效了，简单实例如下：



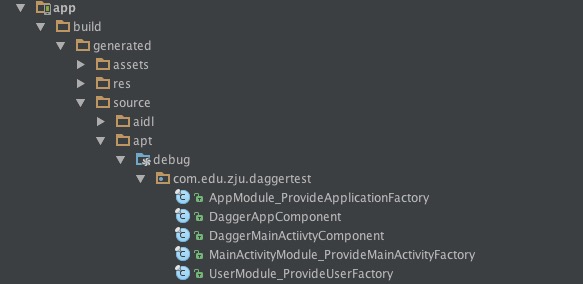
这2个维度是有优先级之分的，Component会首先从Module维度中查找类实例，若找到就用Module维度创建类实例，并停止查找Inject维度。否则才是从Inject维度查找类实例。所以创建类实例级别Module维度要高于Inject维度。

(4) 完成依赖注入：

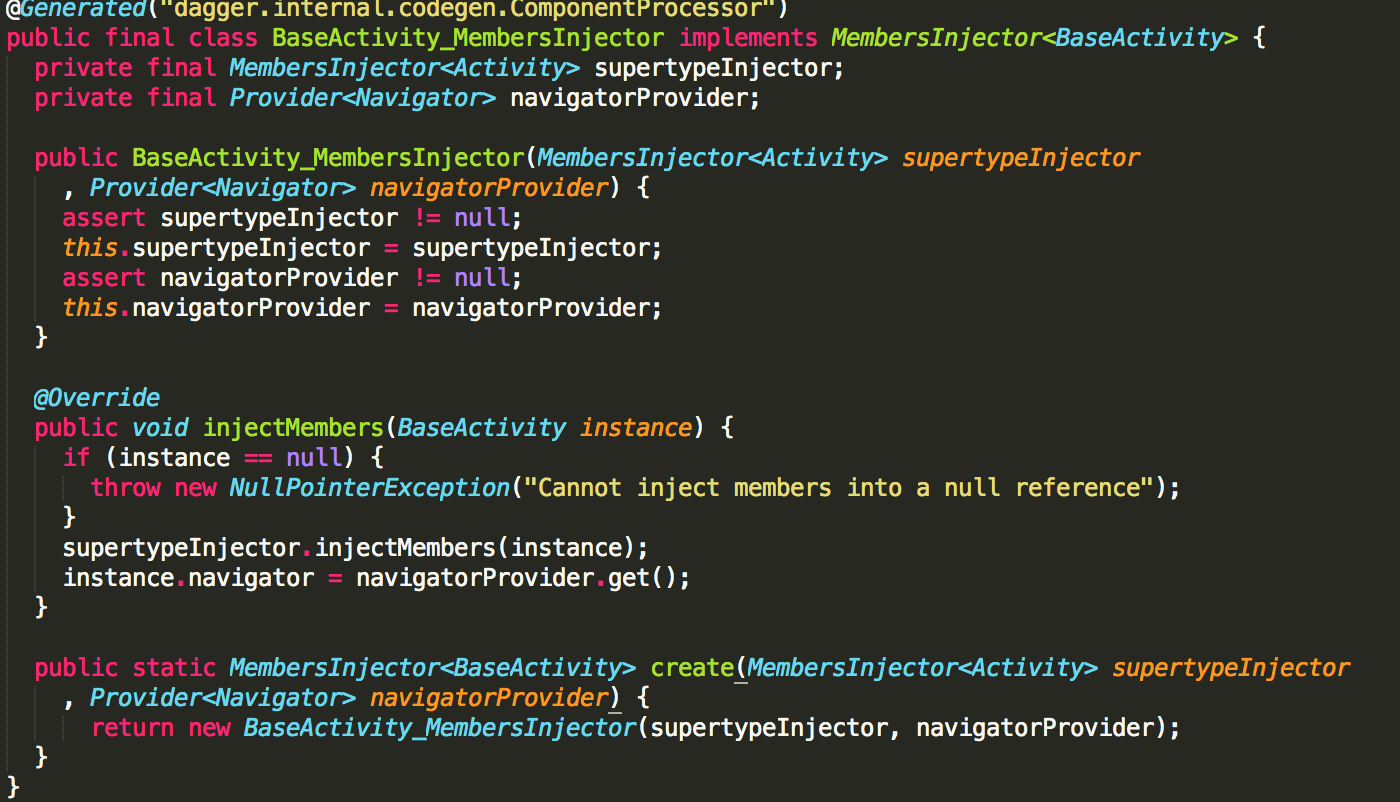


**4.2 Dagger2生成代码实现注入**

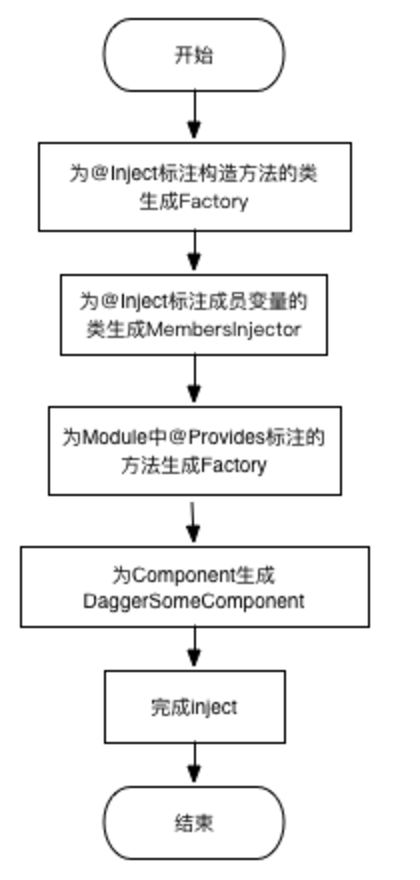
编译时 Dagger 会处理我们的注解，为 @Components 生成实现并命名为 Dagger$${YouComponentClassName}，如 UserComponent -> DaggerUserComponent。在 app/build/generated/source/apt 下可以找到相关的类：



由于我们要将依赖注入到activity中，所以会得到一个注入这个比成员的注入器（由Dagger生成的BaseActivity\_MembersInjector）：



DaggerAppComponent类有一个Provider，它不仅仅是一个提供实例的接口，它还是被ScopedProvider构造出来的，可以记录创建实例的范围。我们可以通过代码调试的方式了解到Dagger2的注入过程。分析上述生成的代码，可以发现Dagger2主要完成的工作流程如下：



(1) 处理@Inject标注的构造方法，为每个类生成Factory；为@Inject标注成员变量的类（即注入类）生成对应的MembersInjector类,实际上是在MembersInjector类中完成对成员变量的初始化;

(2)为@Module标注的Module中被@Provides标注的方法，生成对应的Factory；

(3)为Component生成DaggerSomeComponent,在DaggerSomeComponent中,使用Inject 或是 Module生成的Factory作为对象提供者,使用MembersInjector完成对象初始化。Component实现类在对象提供者中查找到依赖对象需要的方法后，将其设置到依赖需求方中。

（4）通过调用Initializer方法完成注入。

**5总结**

依赖注入是一种优秀的编程思想，它可以通过解耦项目来提升项目的可阅读性、可扩展性和可维护性，并使得单元测试更为方便。Dagger 由于其自身的复杂性，其实是一个上手难度颇高的库，难学会、难用好。但从功能上来讲，它又是一个实用价值非常高的库。Dagger2相较于最初的版本，性能上有了很大的提高，虽然减少了反射机制的使用，节省了时间，但也丧失了一部分的灵活性。

参考文献

[1]Effective Java中文版（第2版）.Joshua Bloch .机械工业出版社

[2] MartinFlower.IoC容器和 DependencyInjection模式[J].Jan.2004: 1— 5.

[3] <https://google.github.io/dagger/android.html> Dagger2官方文档

[4] RobertCMartin.敏捷软件开发 :原则、模式 与实践 [M].清华大学出版社

[5] Martin Flower. Patterns of Enterprise Application Architecture [M]. Nov.2o02.