



【菜鸟的成长史】

在向老鸟蜕变的过程中请不要对我嘲笑...

[目录视图](#)[摘要视图](#)[RSS 订阅](#)

个人资料



Jabin.

[+ 加关注](#)[发私信](#)

访问： 248966次

积分： 2597

等级：BLOG > 5

排名： 第8635名

原创： 40篇 转载： 8篇

译文： 0篇 评论： 61条

说明：

注意

最新内容更新：
blog.fidroid.com

欢迎加入AndroidStudio
Tech交流群：115457704

欢迎关注微信订阅号:developers
关注博主最新动态

博客专栏

Android开发一
点一滴

文章：12篇

阅读：68093

阅读排行

- Android开源框架(一) : A (41729)
- Android Studio Win7安装 (20467)
- android 判断字符串是否 (19249)
- Android 外部存储权限分 (17019)
- Android 依赖注入 : Dagger (15316)
- Android 依赖注入 : Dagger (11855)
- Android 数据库存取图片 (10309)
- 侧滑、listView中折叠效果 (7254)
- Android Toast 显示时间 (6631)
- Spinner--动态增加数据 (5570)

文章搜索



人工智能实战——人工神经网络(C库iOS交叉编译) 前端精品课程免费看，写课评赢心动大礼！[JavaScript知识库发布](#)

Android 依赖注入：Dagger 实例讲解 (Demo下载)

标签：[android](#) [Dagger](#) [Dependency](#)2014-01-24 23:50 15316人阅读 [评论\(6\)](#) [收藏](#) [举报](#)分类：[android](#) (35)

版权声明：本文为博主原创文章，未经博主允许不得转载。

(Dagger--A fast dependency injector for Android and Java 实例讲解)

IDE: AndroidStudio 4.2

Dagger 是一种android平台的依赖注入框架，是有一家专注于移动支付的公司，Square公司推出的库，这家公司也推出了

其他在Android开发中常用库：otto，okhttp，retrofit等等，这些在接下的博文中会一一介绍。

对Dagger的介绍，除了官方文档的介绍外，接下来的这些分析，本人觉得是比较不错的，也许在不熟悉Dagger的情况下看这

写内容，你会觉得无厘头，不知道讲什么。根据本人经验，建议先了解Dagger的使用再来看这个会对了解Dagger有比较好的效果。

如果你不喜欢看英文官方文档介绍，请参考：<http://fanxu.me/post/2013-07-18>

Dagger的简介，请阅读：<http://www.infoq.com/cn/news/2012/10/dagger>

接下来就进入正题,进行实例讲解，与官方例子一样，我这里也用煮咖啡的例子来说明，只不过官方的例子是用在Java上，这里

的例子使用Android开发来进行，并且对其也进行了改进。

首先我先说明一下这个过程,既然是煮咖啡，就要由一个加热器(Heater),加热之后要有一个倒(Pump)的过程,倒好之后才能供给人

们喝(Drink)。就这么一个简单的逻辑，你也许会和我开始一样，觉得这有什么难的，很少的代码就能搞定。是的，我们就要用这么一

个简单的事件，来看看用Dagger是怎么实现的。

首先，我们来设计Heater、Pump、Drink这三个接口，如下：

```
[java] view plain copy print ? C ⌂
package com.example.app.dagger;
public interface Heater {
    void on(); // 加热器打开
    void off(); // 加热器关闭
    boolean isHot(); // 加热是否完毕
}
```

```
[java] view plain copy print ? C ⌂
package com.example.app.dagger;
public interface Pump {
    void pump(); // 倒咖啡
    boolean isPumped(); // 是否倒好
}
```

推荐文章

- *Android View体系(七) 从源码解析View的measure流程
- *你真的了解iOS代理设计模式吗?
- *浅谈Storm流式处理框架
- *有关深度学习领域的几点想法
- *浅谈百度地图的简单开发之结合方向传感器实现定位功能(三)
- *大数据常用十种开发语言

评论排行

- Android开源框架(一) : A (10)
- Android 依赖注入 : Dag (8)
- Android 依赖注入 : Dagg (6)
- Android 自定义进度条 (5)
- Android Studio Win7安装 (4)
- android 判断字符串是否 (4)
- Android Toast 显示时间 (4)
- android CircularSeekBar (3)
- Android 外部存储权限分 (3)
- android 定时发送短信实 (3)

```
[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02.
03. /**
04. * Created by zjb on 14-1-22.
05. */
06. interface Drink {
07.     void drink(); //喝咖啡
08. }
```

Ok,接口已经设计完毕，是否合理暂且不细究，接下来我们分别实现这三个接口(有用到AndroidAnnotations框架(请看上篇博文))：

```
[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02. import android.content.Context;
03. import android.widget.Toast;
04. import org.androidannotations.annotations.EBean;
05. import org.androidannotations.annotations.RootContext;
06. import org.androidannotations.annotations.UiThread;
07.
08. /**
09. * Created by zjb on 14-1-22.
10. */
11. @EBean
12. class ElectricHeater implements Heater {
13.     boolean heating = false;
14.
15.     @RootContext
16.     Context context;
17.
18.     @Override
19.     public void on() {
20.         heating = true;
21.         System.out.println("----Heating----");
22.         reportHeating();
23.     }
24.
25.     @UiThread
26.     void reportHeating() {
27.         Toast.makeText(context, "Electric heater heating....", Toast.LENGTH_LONG).show();
28.     }
29.
30.     @Override
31.     public void off() {
32.         heating = false;
33.     }
34.
35.     @Override
36.     public boolean isHot() {
37.         return heating;
38.     }
39. }
```

ElectricHeater是对Heater接口的实现类，@EBean,@RootContext,@UiThread是AndroidAnnotations框架中的注解。

使用@EBean注解会在编译过程中产生一个ElectricHeater子类ElectricHeater_.class, 接下来会用到。

Pump接口实现：

```
[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02. import javax.inject.Inject;
03.
04. /**
05. * Created by zjb on 14-1-22.
06. */
07. class Thermosiphon implements Pump {
08.     private final Heater heater;
09.     boolean pumped = false;
10.
11.     @Inject
12.     Thermosiphon(Heater heater) {
13.         this.heater = heater;
14.     }
15.
16.     @Override
17.     public void pump() {
18.         if (heater.isHot()) {
```

```

19.         System.out.println("-----Pumping-----");
20.         pumped = true;
21.         try {
22.             Thread.sleep(1000);
23.         } catch (InterruptedException e) {
24.             e.printStackTrace();
25.         }
26.     }
27. }
28. @Override
29. public boolean isPumped() {
30.     return pumped;
31. }
32.

```

```

[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02. import javax.inject.Inject;
03.
04.
05. /**
06. * Created by zjb on 14-1-22.
07. */
08. class PeopleDrink implements Drink {
09.     private Pump pump;
10.     @Inject
11.     PeopleDrink(Pump pump) {
12.         this.pump = pump;
13.     }
14.     @Override
15.     public void drink(){
16.         if(pump.isPumped()){
17.             System.out.println("-----Drinking-----");
18.         }
19.         try {
20.             Thread.sleep(1000);
21.         } catch (InterruptedException e) {
22.             e.printStackTrace();
23.         }
24.     }

```

三个接口已经实现完毕，对比三个实现类，有没有发现什么共同点。是的，你会发现，三个类中都有使用@Inject来注解

他们的构造函数，这是因为Dagger要用@Inject来注解一个类的实例构造函数，当请求一个新实例的时候，Dagger就会获取这个

参数值并调用这个构造函数。也许你不明白，没关系，继续往下看，会给出详细解释。

Dagger不仅能向上述代码那样注解构造函数，也能直接注解fields(Dagger can inject fields directly)，看这个类：

```

[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02. import javax.inject.Inject;
03. import dagger.Lazy;
04.
05. /**
06. * Created by zjb on 14-1-22.
07. */
08.
09. class CoffeeMaker {
10.     @Inject
11.     Lazy<Heater> heater;
12.     @Inject
13.     Pump pump;
14.     @Inject
15.     Drink drink;
16.
17.     public void brew() {
18.         heater.get().on();
19.         pump.pump();
20.         System.out.println("-----Pumped-----");
21.         heater.get().off();
22.         drink.drink();
23.     }

```

将Heater、Pump及Drink注入到类CoffeeMaker中，就可以直接使用并调用其方法。值得注意的是，在注解的时候Dagger

就会通过@Module中的@Provides方法调用构造函数来获得实例对象（下面马上介绍）。如果你@Inject fields

却没有@Inject

构造函数，Dagger就会使用一个存在的无参构造函数，若没有@Inject构造函数，就会出错。继续看@Module

```
[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02.
03. import android.content.Context;
04. import javax.inject.Singleton;
05. import dagger.Module;
06. import dagger.Provides;
07.
08. /**
09. * Created by zjb on 14-1-22.
10. */
11. @Module(injects={CoffeeActivity_.class},library = true,complete = false)
12. /*@Module(injects = {CoffeeActivity_.class},includes = {PumpModule.class,DrinkModule.class})*/
13. class DripCoffeeModule {
14.     private final Context context;
15.
16.     public DripCoffeeModule(Context context) {
17.         this.context = context.getApplicationContext();
18.     }
19.
20.     @Provides
21.     @Singleton
22.     Context appliactionContext() {
23.         return context;
24.     }
25.
26.     @Provides
27.     @Singleton
28.     Heater provideHeater(){
29.         return ElectricHeater_.getInstance_(appliactionContext());
30.     }
31.
32.     @Provides
33.     @Singleton
34.     Drink provideDrink(PeopleDrink drink){
35.         return drink;
36.     }
37.
38.     @Provides
39.     @Singleton
40.     Pump providePump(Thermosiphon pump){
41.         return pump;
42.     }
43. }
```

上面就是使用@Module注解的类，Dagger要求所有的@Provides必须属于一个Module.他们仅仅是一个使用@Module注解的类。

解释一下前面一句话：如果说@Inject实现了注入，那么@Provides就是实现依赖关系。@Provides方法方法的返回类型就定义了它

所满足的依赖。你也许注意到了我注释掉的@Module，这是什么意思呢？是这样的，假如我这里将providesDrink方法删除，我可以

另建一个DrinkModule.java文件，由于所有的@Provides必须属于一个Module，所以必须将DrinkModule类includes进来：

```
[java] view plain copy print ? C ⌂
01. package com.example.app.dagger;
02. import dagger.Module;
03.
04. /**
05. * Created by zjb on 14-1-22.
06. */
07. @Module(library = true,complete = false)
08. public class DrinkModule {
09.     @Provides
10.     @Singleton
11.     Drink provideDrink(PeopleDrink drink){
12.         return drink;
13.     }
14. }
```

(@Module后的library和complete是什么意思这里先不说)

至此，Dagger中的三个重要annotation已经全部涉及到了，那么它是如何管理这些依赖关系的呢？继续往下看：

```
[java] view plain copy print ? ⌂ ⌂
01. package com.example.app.dagger;
02. import android.app.Application;
03.
04. import org.androidannotations.annotations.EApplication;
05.
06. import dagger.ObjectGraph;
07.
08. /**
09. * Created by zjb on 14-1-22.
10. */
11.
12. @EApplication
13. public class CoffeeApplication extends Application {
14.     private ObjectGraph objectGraph;
15.
16.     @Override
17.     public void onCreate() {
18.         super.onCreate();
19.         objectGraph =  快速回复 create(new DripCoffeeModule(this));
20.     }
21.     
22.     public ObjectGraph getObjectGraph() {
23.         return objectGraph;
24.     }
25.
26. }
```

上边提到，Dagger是通过什么管理或者组织依赖关系的呢，就是通过ObjectGraph(对象图表)。

最后的主程序：在界面中就一个按钮来触发整个过程

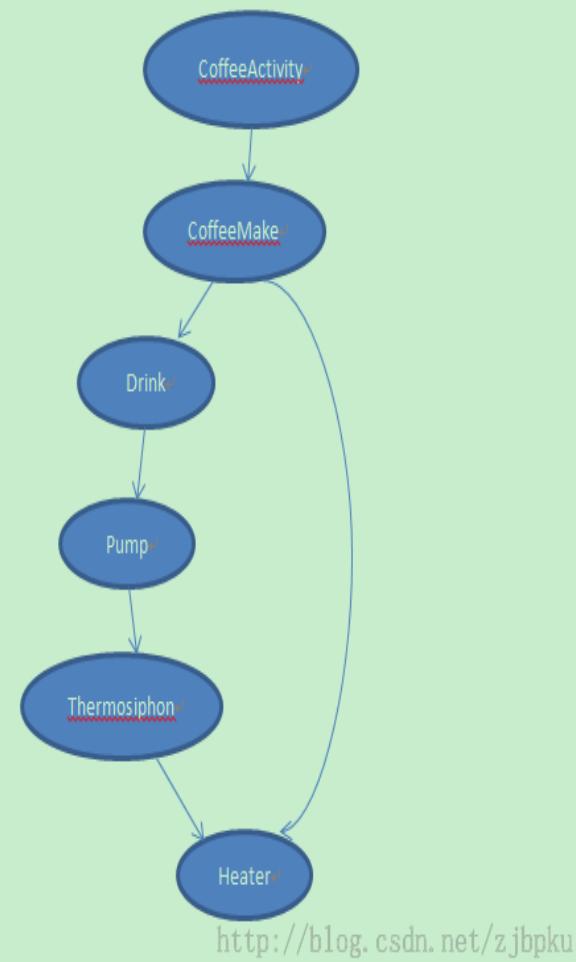
```
[java] view plain copy print ? ⌂ ⌂
01. package com.example.app.dagger;
02.
03. import android.app.Activity;
04. import android.widget.Toast;
05. import com.example.app.R;
06. import org.androidannotations.annotations.AfterInject;
07. import org.androidannotations.annotations.App;
08. import org.androidannotations.annotations.Background;
09. import org.androidannotations.annotations.Click;
10. import org.androidannotations.annotations.EActivity;
11. import org.androidannotations.annotations.UiThread;
12. import javax.inject.Inject;
13. import dagger.ObjectGraph;
14.
15. /**
16. * Created by zjb on 14-1-22.
17. */
18.
19. @EActivity(R.layout.coffee)
20. public class CoffeeActivity extends Activity {
21.     @App
22.     CoffeeApplication coffeeApplication;
23.     @Inject
24.     CoffeeMaker maker;
25.
26.     @AfterInject
27.     void daggerInject(){
28.         ObjectGraph objectGraph = coffeeApplication.getObjectGraph();
29.         objectGraph.inject(this);
30.     }
31.     @Click(R.id.coffeeClick)
32.     @Background
33.     void coffeeClicked(){
34.         maker.brew();
35.         coffeeBrew();
36.     }
37.     @UiThread
38.     void coffeeBrew(){
39.         Toast.makeText(this,"Coffee has been pumped...",Toast.LENGTH_LONG).show();
40.     }
41. }
```

程序运行的结果：

```
[java] view plain copy print ? ⌂ ⌂
01. com.example.app I/System.out: -----Heating-----
02. com.example.app I/System.out: -----Pumping-----
03. com.example.app I/System.out: -----Pumped-----
04. com.example.app I/System.out: -----Drinking-----
```

使用Dagger，我们能够很好的实现依赖关系，也能更清楚的看到我们的代码在做些事情，能够清晰地显示出各部分的

逻辑关系，通过下面这张图，我们能够很清楚的看到它的每一步操作：



好了，Dagger呢就介绍到这里，源码这次就先不上传了，等到介绍完Otto之后，我会抽空上传到资源，供大家下载学习。

内容细节说的不全请见谅、指教。如果你对使用dagger有兴趣也欢迎一起讨论、学习。这是春节前最后一篇文章了，祝大家

抢到一张好票，马上有钱、有对象！

注：文章原创 转载请注明出处：CSDN **菜鸟的成长史**

附：

Demo [下载](#)

Jake Warthon的Dagger ppt [下载](#)资源



顶
9 踩
0

上一篇 Android开源框架(一)：AndroidAnnotations

下一篇 Android数据库存取对象--CupBoard

我的同类文章

android (35)

- RecyclerView 实例 2015-06-01 阅读 780
- GreenDao 执行sql语句 2014-09-14 阅读 3234
- Android 外部存储权限分析 2014-05-24
- 如何优化app，看Facebook的秘诀 2014-06-21 阅读 2081
- 阅读 16965
- Android数据库存取对象--... 2014-03-18 阅读 3437
- Android开源框架(一)：Android Intent 2014-01-23
- Google Play Services 4.1 ... 2014-01-11 阅读 1491
- 阅读 41599

更多文章

猜你在找

- Android开发精品课程【Java核心知识】
- Android必备的Java基础知识(二)
- Android底层技术：Java层系统服务(Android Service)
- 如何快速开发一款具有类微信即时通讯功能App
- 解析移动应用的身份认证、数据分析及信息推送

查看评论

5楼 大树 2016-03-08 12:12发表



不懂他的构造方法的参数是在哪传的 先mark

4楼 大风中的自己 2015-09-06 19:19发表



讲的很不错

3楼 ze帆_fan 2015-04-24 11:19发表



看这篇文章就想了解 library 和 complete 两个属性是干什么的, 没想到看到最后, 博主竟然说先不说. 囧.....

2楼 Jabin. 2014-07-17 17:26发表



引用“A328240784”的评论：

这些东西没用的，一群搞javaee的人企图把自己那套复杂的东西搬到android上，我们以前的傻逼架...

Dagger的好处你不懂，当然学习是需要时间等成本的

1楼 追风筝的孩子 2014-07-17 10:35发表



这些东西没用的，一群搞javaee的人企图把自己那套复杂的东西搬到android上，我们以前的傻逼架构师也试过。你自己从学到掌握到灵活运用是需要一段时间的，而你的队友也需要时间，最后的结果就是效率反而降低了，代码可读性还不高，必须学过才看得懂。

发表评论

用户名：

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap

