当复仇者联盟遇上Dagger2、RxJava和Retrofit的巧妙结合

泡在网上的日子 发表于 2015-06-01 10:27 第 8823 次阅读 Dagger , RxJava , Retrofit

编辑推荐：稀土掘金，这是一个针对技术开发者的一个应用，你可以在掘金上获取最新最优质的技术干货，不仅仅是Android知识、前端、后端以至于产品和设计都有涉猎，想成为全栈工程师的朋友不要错过！

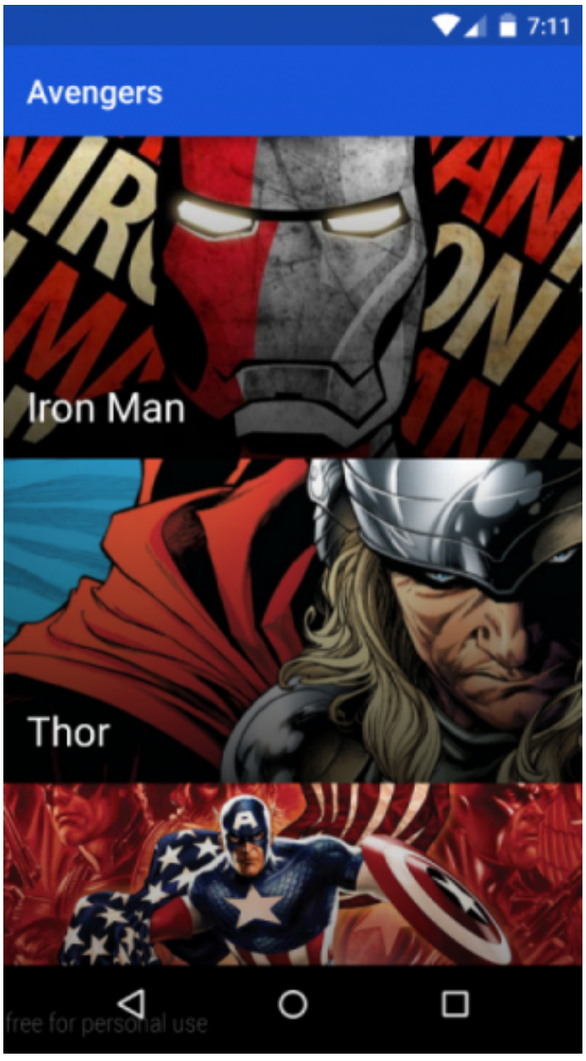
- 原文链接 : When the Avengers meet Dagger2, RxJava and Retrofit in a clean way
- 原文作者 : Saúl M
- 译文出自： 开发技术前线 www.devtf.cn
- 译者 : zhengxiaopeng

最近，许多文章、框架和 android 社区中的讨论都出现关于测试和软件架构方面的内容，就像上次 Droidcon Spain 上所说的，我们专注于做出健壮的程序而不是去开发特性功能。这些现象也意味着 Android 框架和当前 Android 社区的日渐成熟。

如果你是一名 Android 开发者，而到现在你还没听过 Dagger 2、RxJava、Retrofit 这些名词的话你就错过了一些东西了，这个（文章）系列将会把一些关注点放在怎么用一种 清晰架构 去综合使用这些框架。

我刚开始的想法是仅仅写一篇文章的，但是看到这些框架中有大量的内容所以我最终决定写一个最少 3 篇的系列文章。

一如既往，所有的代码都放在了 Github，所有的建议、错误提交和评论都欢迎，我可能没那么多时间去回答所有问题，先说声抱歉：)



依赖注入与 Dagger 2

弄懂这个框架的工作机制花费了一些时间，所以我将会根据我所学习到的内容用更加清晰的方式写出来。

Dagger 2是基于 依赖注入 模式的。

看下下面的代码片段：

```
// Thor is awesome. He has a hammer!
public class Thor extends Avenger {
    private final AvengerWeapon myAmazingHammer;

    public Thor (AvengerWeapon anAmazingHammer) {
        myAmazingHammer = anAmazingHammer;
    }

    public void doAmazingThorWork () {
        myAmazingHammer.hitSomeone();
    }
}
```

雷神（Thor）需要一个 `复仇者武器（AvengerWeapon）` 才能正确工作，依赖注入的基本思想是，如果雷神不是通过构造器创建他自己的 `复仇者武器` 而是在内部自己创建了出来那么他就不能得到很多的优势。如果雷神自己创建出雷锤将会增加耦合度。

`复仇者武器（AvengerWeapon）` 可以是一个接口，根据我们的逻辑可以有不同的实现和注入方式。

在 Android 中，因为框架已经设计好了，我们并不总是能访问构造器，`Activity` 和 `Fragment` 就是这样的例子。

这些依赖注入器框架像 http://google.github.io/dagger/、Dagger 、Guice 可以给我们带来便利。

使用 Dagger 2 我们可以把之前的代码改写成这样：

```
// Thor is awesome. He has a hammer!
    public class Thor extends Avenger {
        @Inject AvengerWeapon myAmazingHammer;

        public void doAmazingThorWork () {
            myAmazingHammer.hitSomeone();
        }
    }
```

我们没有直接访问雷神的构造方法，注入器使用了几个指令去创建了雷神的雷锤

```
    public class ThorHammer extends AvengerWeapon () {

        @Inject public AvengerWeapon() {

            initGodHammer();
        }
    }
```

`@Inject` 注解用于告诉 Dagger 2 构造器有用于创建雷神的雷锤。

Dagger 2

Dagger 2 由 Google 开发和维护，是 Square 的 Dagger 项目的分支。

首先必须配置注解处理器，`android-apt` 插件就是负责这个角色，允许使用注解处理器而不将其插入到最后的 .apk 中。处理器还配置由该处理器所产生的源代码。

`build.gradle`（项目的根目录中）

```
dependencies {
        ...
        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.4'
    }
```

`build.gradle`（你的 android module 中）

```
apply plugin: 'com.neenbedankt.android-apt'

    dependencies {
        ...
        apt 'com.google.dagger:dagger-compiler:2.0'
    }
```

组件（Components）、模块（modules）和复仇者

模块负责提供依赖，组件负责注入它们（依赖）。

这是一个例子：

```
@Module
public class AppModule {
    private final AvengersApplication mAvengersApplication;

    public AppModule(AvengersApplication avengersApplication) {
        this.mAvengersApplication = avengersApplication;
    }

    @Provides @Singleton
    AvengersApplication provideAvengersAppContext () {
        return mAvengersApplication;
```

```
    }

    @Provides @Singleton
    Repository provideDataRepository (RestRepository restRepository) {
        return restRepository;
    }
}
```

这个就是主模块，我们感兴趣的是它的依赖存在于程序的生命周期中，一个通用的上下文和一个取回信息的仓库。

很简单，对吧？

我们在 Dagger 2 中所说的 `@Provides` 注解，如果有需要则必须会创建其依赖。因此如果我们没有给一个特别的依赖指定一个提供者（provider），Dagger 2 将会去寻找有 `@Inject` 注解的构造方法。

组件使用模块去完成依赖注入，看看这个模块的组件：

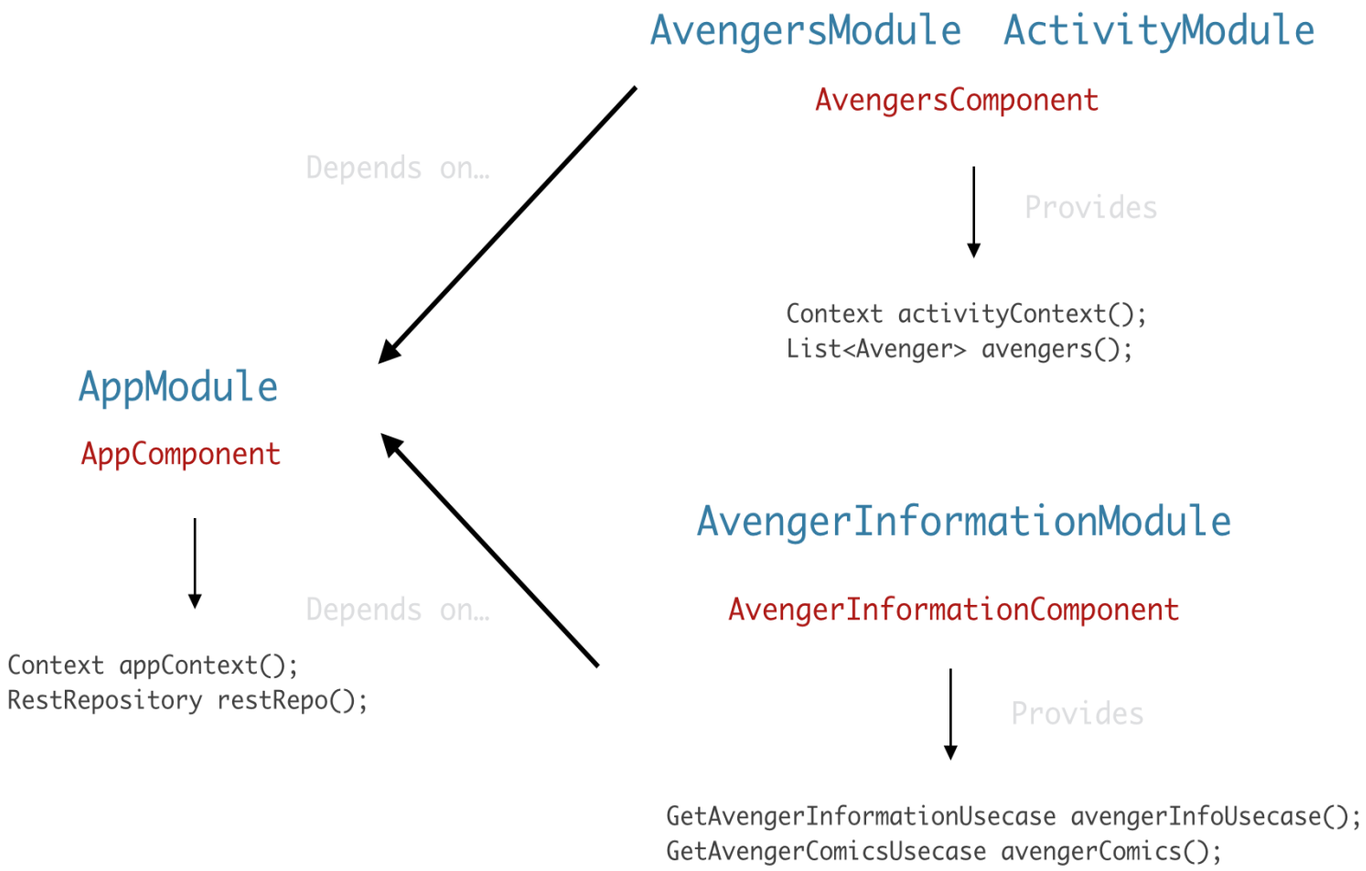
```
@Singleton @Component(modules = AppModule.class)
public interface AppComponent {

    AvengersApplication app();
    Repository dataRepository();
}
```

这个模块并不由任何的 activity 或者 fragment 去调用，而是通过更复杂的模块，以提供这些需要得到的依赖

```
AvengersApplication app();
Repository dataRepository();
```

组件必须暴露它们的依赖给图（该模块提供的依赖关系），也即是这个模块提供的依赖关系必须对其它组件是可见的，其它的组件有把当前这个组件作为依赖，如果这些依赖关系是不可见的，Dagger 2 将不会注入这些要求的依赖。

下面是我们的依赖关系树：




```
@Module
public class AvengersModule {

    @Provides @Activity
    List<Character> provideAvengers() {

        List<Character> avengers = new ArrayList<>(6);

        avengers.add(new Character(
            "Iron Man", R.drawable.thumb_iron_man, 1009368));

        avengers.add(new Character(
            "Thor", R.drawable.thumb_thor, 1009664));

        avengers.add(new Character(
            "Captain America", R.drawable.thumb_cap,1009220));

        avengers.add(new Character(
            "Black Widow", R.drawable.thumb_nat, 1009189));

        avengers.add(new Character(
            "Hawkeye", R.drawable.thumb_hawkeye, 1009338));

        avengers.add(new Character(
            "Hulk", R.drawable.thumb_hulk, 1009351));

        return avengers;
    }
}
```

这个模块将会用于一个特别的 activity 的依赖注入，实际上就是负责绘画的复仇者名单：

```
@Activity
```

```
@Component(
    dependencies = AppComponent.class,
    modules = {
        AvengersModule.class,
        ActivityModule.class
    }
)
public interface AvengersComponent extends ActivityComponent {

    void inject (AvengersListActivity activity);
    List<Character> avengers();
}
```

再次，我们暴露出我们的依赖：`List<Character>` 给其它的组件，在这种情况下出现了一个新方法：`void inject (AvengersListActivity activity)`。**在此方法被调用时**，这些依赖关系将可被消耗，将会被注入给 `AvengerListActivity`。

结合所有

我们的类 `AvengersApplication`，将负责提供应用到其他组件的组件，注意，仅仅提供组件而不会用于注入依赖。

再次提醒的是 Dagger 2 是在编译时生成必要的元素，如果你没有构建项目你是找不到 `DaggerAppComponent` 类的。

Dagger 2 从你的组件中生成的类的格式：`Dagger$${YourComponent}.`。

`AvengersApplication.java`

```
public class AvengersApplication extends Application {

    private AppComponent mAppComponent;

    @Override
    public void onCreate() {

        super.onCreate();
        initializeInjector();
    }

    private void initializeInjector() {

        mAppComponent = DaggerAppComponent.builder()
            .appModule(new AppModule(this))
            .build();
    }

    public AppComponent getAppComponent() {

        return mAppComponent;
    }
}
```

`AvengersListActivity.java`

```
public class AvengersListActivity extends Activity
    implements AvengersView {

    @InjectView(R.id.activity_avengers_recycler)
    RecyclerView mAvengersRecycler;

    @InjectView(R.id.activity_avengers_toolbar)
    Toolbar mAvengersToolbar;

    @Inject
    AvengersListPresenter mAvengersListPresenter;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_avengers_list);
        ButterKnife.inject(this);

        initializeToolbar();
        initializeRecyclerView();
        initializeDependencyInjector();
        initializePresenter();
    }

    private void initializeDependencyInjector() {

        AvengersApplication avengersApplication =
            (AvengersApplication) getApplication();

        DaggerAvengersComponent.builder()
            .avengersModule(new AvengersModule())
            .activityModule(new ActivityModule(this))
        .appComponent(avengersApplication.getAppComponent())
            .build().inject(this);
    }
```

当 `initializeDependencyInjector()` 中执行到 `.inject(this)` 中时 Dagger 2 就会开始工作并提供必要的依赖关系，请记住 Dagger 2 在注入时是严格执行的，我要说的意思是组件必须是 **完全相同** 的组件类，此组件负责调用 `inject()` 方法。

`AvengersComponent.java`

```
...
public interface AvengersComponent extends ActivityComponent {

    void inject (AvengersListActivity activity);
    List<Character> avengers();
}
```

否则，依赖关系将不会被解决。在如下情况，presen[illegible]化:


```
public class AvengersListPresenter imple[illegible]tener {

    private final List<Character> mAvengersList;
    private final Context mContext;
    private AvengersView mAvengersView;
    private Intent mIntent;

    @Inject public AvengersListPresenter (List<Character> avengers, Context context) {

        mAvengersList = avengers;
        mContext = context;
    }
```

Dagger 2 将会解决这个 presenter，因为其有 `@Inject` 注解。该构造方法的参数由 Dagger 2 解决，因为它知道怎么去构建它，这得益于这个模块中的 `@Provides` 方法。

总结

像 Dagger 2 这样，使用好了依赖注入器，其力量是无可争辩的，想象下根据该框架提供的 API 级别你可以有不同的策略，其可能性是无止境的。

资源

- **Chiu-Ki Chan** – Dagger 2 + Espresso + Mockito
- **Fernando Cejas** – Tasting Dagger 2 on Android
- **Google Developers** – Dagger 2, A new type of dependency injection
- **Mike Gouline** – Dagger 2, Even sharper, less square

相关文章

Retrofit源码解析	2015-06-02
使用Retrofit请求API数据 - codepath教程	2015-10-16
Retrofit 源码解读之离线缓存策略的实现	2016-01-15
Retrofit使用教程(二)	2016-03-23
使用Mockito、Robolectric和RxJava及Retrofit进行单元测试	2015-09-15
用 Retrofit 2 简化 HTTP 请求-来自 Droidcon NYC 2015 一个演讲	2015-11-09
浅谈 RxAndroid + Retrofit + Databinding	2016-01-31
Retrofit使用教程(三) : Retrofit与RxJava初相逢	2016-03-25
Android REST客户端库Retrofit的小教程	2015-04-18
Retrofit 2.0：有史以来最大的改进	2015-09-15

上一篇：关于Android的Data Binding技术

此次Google IO大会，给Andorid开发者带来了很多福利。我对其中的Data Binding技术最感兴趣，所以花时间研究了一下。 Data Binding与MVVM Data Binding即数据绑定，在很多前端框架中都有应用，比如Google维护的AngularJS就支持View和Model的双向绑定。当数据

下一篇：关于goole IO大会发布的android M和android studio1.3的更新

一听说google IO大会新发布了，就默默关注google的新的方向，这不，之前盛传已久的android M 棉花糖发布了（目前名字还没定，我比较希望google选择marshmallow这个名字），我就着急的去想去体验一把了，码农就这点爱好，喜欢尝鲜，再者体验一下新版android s

发表评论

网友218.75.69.100 . 2016-03-23

感谢分享，但是有些语句怎么读都读不通顺。例如：
请记住 Dagger 2 在注入时是严格执行的，我要说的意思是组件必须是 完全相同 的组件类，此组件类负责调用 inject() 方法。

0 0 回复

万事屋 . 2016-01-13

很好的学习demo，mvp，Dagger2，RxJava，Retrofi。建议熟悉每条知识后再看代码。

0 0 回复

网友58.250.197.82 . 2016-01-06

感觉这样反而把一个简单的事情搞复杂了。

0 4 回复

万事屋 . 2015-12-30

> XIAOMEIXW 的原帖：
> 这哥们特牛比~~~

看不懂

0 0 回复

XIAOMEIXW . 2015-06-30

Part 2 - RxJava RxAndroid Reactive Extensions & operators 请求翻译~~

0 0 回复

XIAOMEIXW . 2015-06-06

这哥们特牛比~~~

0 0 回复

推荐文章

android的图标资源及其巧用

RadioGroup实现类似ios的分段选择(UISegmentedControl)控件

可以下拉缩放HeaderView的ListView:PullToZoomInListView

android中利用Camera编写拍照应用

通过Hardware Layer提升Android动画性能

android中正确保存view的状态