# Dagger 2 完全解析（四），Android 中使用 Dagger 2

最帅的 JohnnyShieh 2017-08-09 20:05


阅读本文大约需要 8 分钟。

在理解了 Dagger 2 完全解析系列的前三篇文章后，可能还是会对在 Android 实际项目中如何使用 Dagger 2 有些疑问，本文以 GoogleSamples 的 android-architecture（mvp 分支）为例，逐步说明如何在 ToDo 项目中使用 Dagger 2。

本文中代码示例的地址：https://github.com/JohnnyShieh/ToDo/tree/mvp

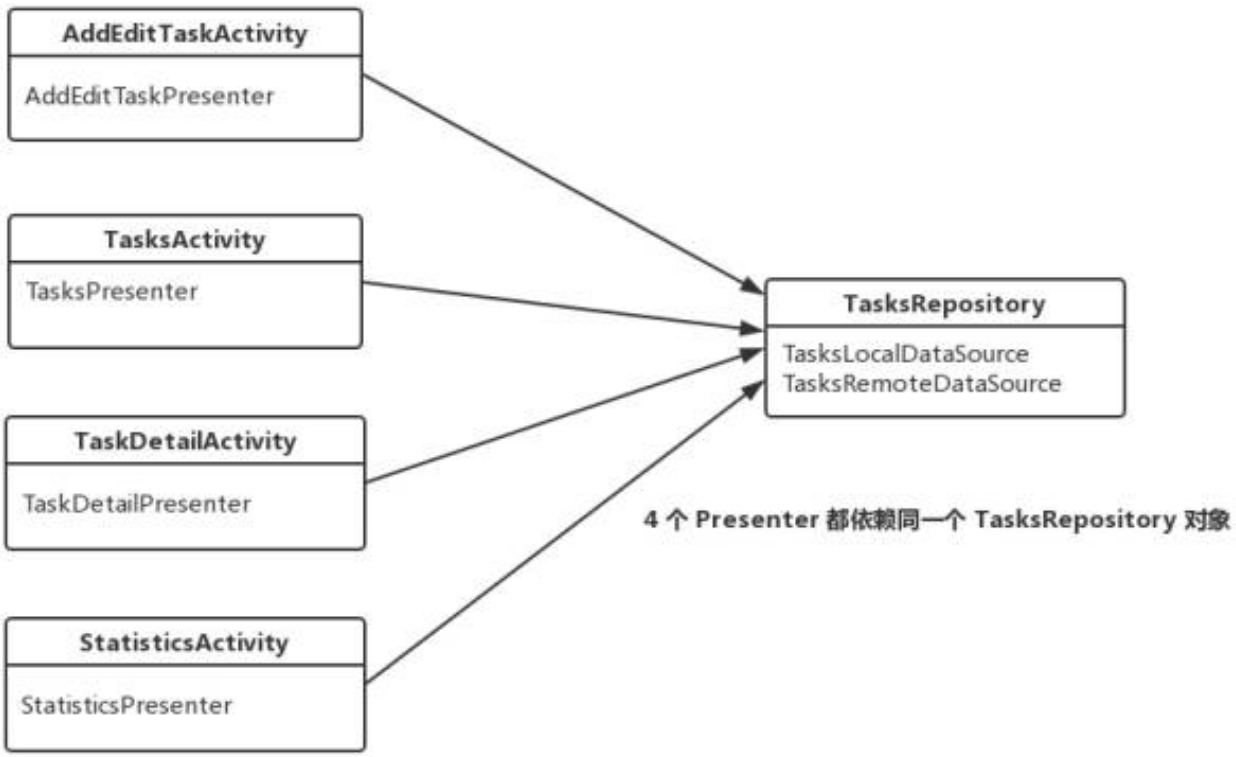
使用 Dagger 2 后的代码地址：https://github.com/JohnnyShieh/ToDo/tree/mvp-dagger2

## 1. Android 中使用 Dagger 2

为了具有代表性，我选择 android-architecture 中的 todo-mvp 作为例子，MVP 架构大家都比较熟悉。现在假设我们的 Android 项目为 ToDo，那么如何引入 Dagger 2 实现依赖注入？

### 1.1 绘制依赖关系图

在使用 Dagger 2 之前，需要理清项目的中依赖关系，这样方便设计 Component，建议大家引入 Dagger 2 之前也绘制下大致的依赖关系图。ToDo 项目中的依赖关系如下图：




ToDo 项目中依赖关系非常简单，4 个 Activity 各自依赖自己的 Presenter，而 Presenter 依赖同一个 TaskRepository 对象。在 Android 项目，依赖关系一般以 Activity、Fragment 这些组件开始划分，因为 app 运行中是以这些组件存在的，所以在后面的 Component 的划分中一般以 Activity、Fragment 来划分。

### 1.2 绘制 Component 依赖关系图

在确定了项目依赖关系，就可以以此划分 Component，从而进一步绘制 Dagger 2 中的依赖关系图。4 个 Activity 分别对应一个 Component，而这 4 个 Component 都依赖一个 TasksRepository。所以还需要一个 Component 提供单例的 TasksRepository 依赖，一般是 AppComponent，用来管理 app 中单例的依赖，而其他 Activity 的 Component 是 AppComponent 的 SubComponnet。

AddEditTaskComponent 、 StatisticsComponenet 、 TaskDetailComponent 、 TasksComponent 都继承 AppComponent，共享同一个 TasksRepository 依赖。

### 1.3 引入 Dagger 2

添加 Dagger 2 依赖，在第一篇文章也有讲述：

```
dependencies {
    ...
```

```
    compile 'com.google.dagger:dagger:2.11-rc2'
    annotationProcessor 'com.google.dagger:dagger-compiler:2.11-rc2'
}
```

上面已经设计好 Component 依赖关系图，下面看看具体的 Component、Module 的编写。

先看看 TasksComponent，其他三个 SubComponent 也是类似的：

```
@ActivityScope
@Subcomponent(modules = TasksPresenterModule.class)
public interface TasksComponent {

    void inject(TasksActivity activity);

    @Subcomponent.Builder
    interface Builder {      // SubComponent 必须实现 @Subcomponent.Builder
        @BindsInstance
        Builder view(TasksContract.View view);    // 在创建 Component 前绑定

        TasksComponent build();
    }
}

@Module
public class TasksPresenterModule {

    @Provides
    @ActivityScope
    TasksPresenter provideTasksPresenter(TasksRepository tasksRepository
        return new TasksPresenter(tasksRepository, view);
    }
}
```

首先来看 `@ActivityScope` 注解，这是一个自定义作用域，和网上一些示例中的 `@PerActivity` 类似。`@ActivityScope` 可以让 TasksComponent 间接持有 TasksPresenter 的引用，而且增加可读性，表示 TasksComponent 的生命周期应该在对应的 Activity 中。

TasksComponent 作为 SubComponent 必须实现 `@Subcomponent.Builder` 接口，因为 TasksComponent 的创建必须由 AppComponent 调用 TasksComponent.Builder 完成。TaskPresenter 还需要 TasksContract.View 依赖，但是它只能在创建 TasksComponent 时提供，有两种方法：（1）TasksPresenterModule 把 TasksContract.View 作为构造函数参数，这样 TasksComponent.Builder 还需要添加

`Builder tasksPresenterModule(TasksPresenterModule module)`；（2）使用`@BindsInstance`方法，如同上面的代码一样，更多关于`@BindsInstance`请看 Dagger 2 完全解析（二），进阶使用 Lazy、Qualifier、Scope 等的末尾部分。这时推荐使用第二种方法，简单明了。

上面可以看到 TasksPresenterModule 的 provideTasksPresenter 方法中还有参数，provide 方法中的参数必须由绑定的 Component 提供依赖，而这里 tasksRepository 实例由 AppComponent 提供依赖，view 实例由`@BindsInstance`方法绑定到 TasksComponent 提供。

看完 SubComponent，再看 AppComponent：

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

    TasksRepository tasksRespository();

    AddEditTaskComponent.Builder addEditTaskComponent();

    StatisticsComponenet.Builder statisticsComponenet();

    TaskDetailComponent.Builder taskDetailComponent();

    TasksComponent.Builder tasksComponent();
}

@Module(subcomponents = {AddEditTaskComponent.class, StatisticsComponene
    TaskDetailComponent.class, TasksComponent.class})
public class AppModule {

    private final Context mContext;

    public AppModule(Context context) {
        mContext = context;
    }

    @Provides
    @Singleton
    Context provideContext() {
        return mContext;
    }
}

@Module
public class TasksRepositoryModule {
```

```
    @Provides
    @Singleton
    @Local
    TasksDataSource provideTasksLocalDataSource(Context context) {
        return new TasksLocalDataSource(context);
    }

    @Provides
    @Singleton
    @Remote
    TasksDataSource provideTasksRemoteDataSource() {
        return new TasksRemoteDataSource();
    }
}
```

上面可以看到 AppModule 提供了 ApplicationContext 依赖，而且确定了 4 个 SubComponent 的继承关系。TasksRepositoryModule 提供了两个 TasksDataSource 依赖（TasksRepositoryModule 在 prod 和 mock 中各有一份），用 `@Local` 和 `@Remote` 两个自定义 Qualifier 区分，但是没有提供 TasksRepository 依赖，那么 AppComponent 管理的 TasksRepository 依赖从哪里来呢？大家不要忘记了提供依赖的两种方法 Module 和 Inject 构造函数。

```
@Singleton
public class TasksRepository implements TasksDataSource {
    ...
    @Inject
    TasksRepository(@Remote TasksDataSource tasksRemoteDataSource,
       @Local TasksDataSource tasksLocalDataSource) {
        mTasksRemoteDataSource = checkNotNull(tasksRemoteDataSource);
        mTasksLocalDataSource = checkNotNull(tasksLocalDataSource);
    }
    ...
}
```

AppComponent 调用 Inject 构造函数创建 TasksRepository 时，会使用 TasksRepositoryModule 提供的两个 TasksDataSource 依赖。推荐大家 clone ToDo 到本地，编译后到 `app/build/generated/source/apt/` 目录下看 DaggerAppComponent 的源码，这里就不为大家分析，之前介绍 Component 时有分析过它的原理。

### 1.4 调用 Component 完成依赖注入

最后一步就是使用 Component 完成依赖注入了，先看 AppComponent：

```
public class ToDoApplication extends Application {

    // Application 只生成一个 AppComponent，AppComponent 只生成一个 TasksRep
    private AppComponent mAppComponent;

    @Override
    public void onCreate() {
        super.onCreate();

        // AppComponent 这里为任何 target 注入依赖，是为它的 SubComponent 提供
        mAppComponent = DaggerAppComponent.builder()
            .appModule(new AppModule(this))
            .tasksRepositoryModule(new TasksRepositoryModule())
            .build();
    }

    public AppComponent getAppComponent() { // 提供该接口后是为了 Activity
        return mAppComponent;
    }
}
```

再看 TasksComponent 的使用：

```
public class TasksActivity extends AppCompatActivity {
    ...
    @Inject
    TasksPresenter mTasksPresenter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        // 调用 AppComponent..tasksComponent() 返回 TasksComponent.Builder
        ((ToDoApplication) getApplication()).getAppComponent()
            .tasksComponent()
            .view(tasksFragment)    // @BindsInstance 方法必须在 build 前调
            .build()
            .inject(this);
         ...
    }
}
```

其他几个 SubComponent 的调用过程也是类似的。

## 2. 其他 Dagger 2 使用示例

除了上面 ToDo App 的示例，我还写了一个 Gank Android 客户端，基于 Dagger 2、RxJava、Retrofit、Glide等开源库使用 RxFlux 架构，里面还有 FragmentComponent 部分，Component 的继承关系为 FragmentComponent -> ActivityComponent -> AppComponent，有兴趣的朋友可以去看下。

## 3. 总结

- 一般会有个 AppComponent，管理 app 中的单例依赖，同时提供 ApplicationContext 依赖。
- 一般一个页面对应一个 Component，例如 Activity、Fragment 对应各自的 Component，但是两个页面的依赖相同时，可以用同一个 Component。
- Android 中推荐使用 Component 的继承关系。
- 尽量多使用 Scope 作用域，增加可读性还能方便控制依赖实例的生命周期。

如何在 Android 项目中使用 Dagger 2 就讲解到这里，有什么问题欢迎各位在下面留言。

END

一个白日做梦的工程师！

不只有技术，还有咖啡和彩蛋！

个人博客：http://johnnyshieh.me

阅读原文