# 日记手帐项目说明

## 概述

本项目为一个基于数据库的日记管理项目，本项目基本架构采用MVP的设计模式，UI采用Material Design的设计语言，整个项目功能完整，可扩展性强，乃居家旅行必备项目。

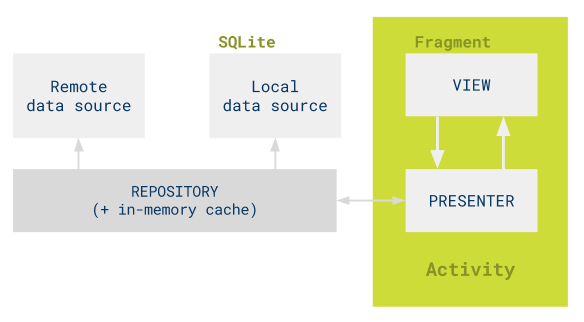
## 采用的框架和插件

* Android Material Design Icon Generator:一款可以生成Material Design风格图标的插件
* ButterKnife：一款注释型View注入框架
* Room Persistence Library：Google官方提供的基于SQLite的数据库框架
* Material Design组件包
* CardView组件包

## 项目架构

### 基本概述

本项目采用MVP设计模式的架构，具有高内聚，低耦合的特点，功能模块分明，方便项目的维护，是项目易于扩展，但是同时也有代码冗余的问题，如果编写不当可能会产生内存泄漏的风向。但是同时具有基本架构如图：



本项目主要利用以下接口来实现MVP架构

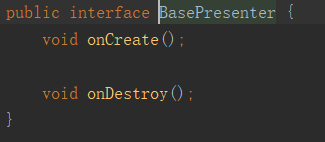
* 一个契约类规定了View层和Presenter层的行为
* Actvivity用于创建Presenter层，自身可以作为为View层，也可以用于创建Fragment(View层)
* Presenter实现了契约中的Presenter接口

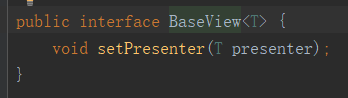
Presenter层主要负责其绑定的业务逻辑，其持有的View层实例用于处理android的UI工作，View层内不应该含有逻辑，View层通过Presenter层的命令来执行UI操作和监听用户操作，用户操作由View层传到presenter层进行处理。同时Presenter层持有Model层实例，用于处理数据逻辑和数据与View的交互，View层与Model层是无法直接交互的。

#### 具体实现

1. 基类

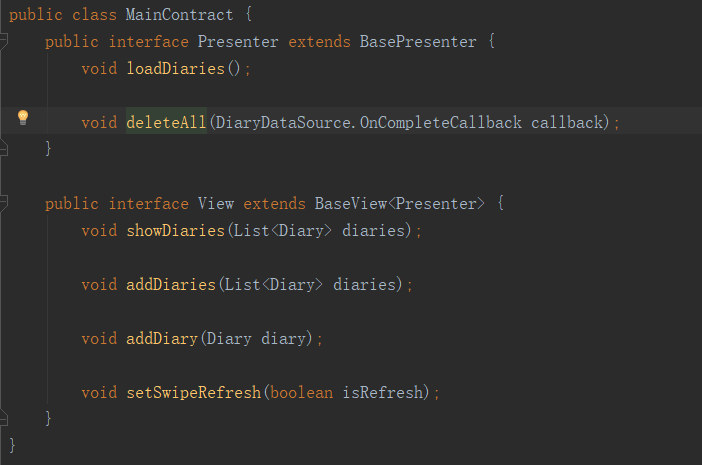
基类主要有BasePresenter和BaseView两个接口，定义了所有Presenter层和View层都具备的基本方法。





1. 契约类

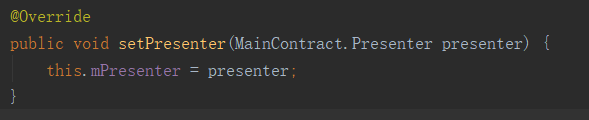
契约类规定了特定Presenter层和View层的功能，这样各个层分别具备什么样的功能一目了然，方便后期维护和Debug



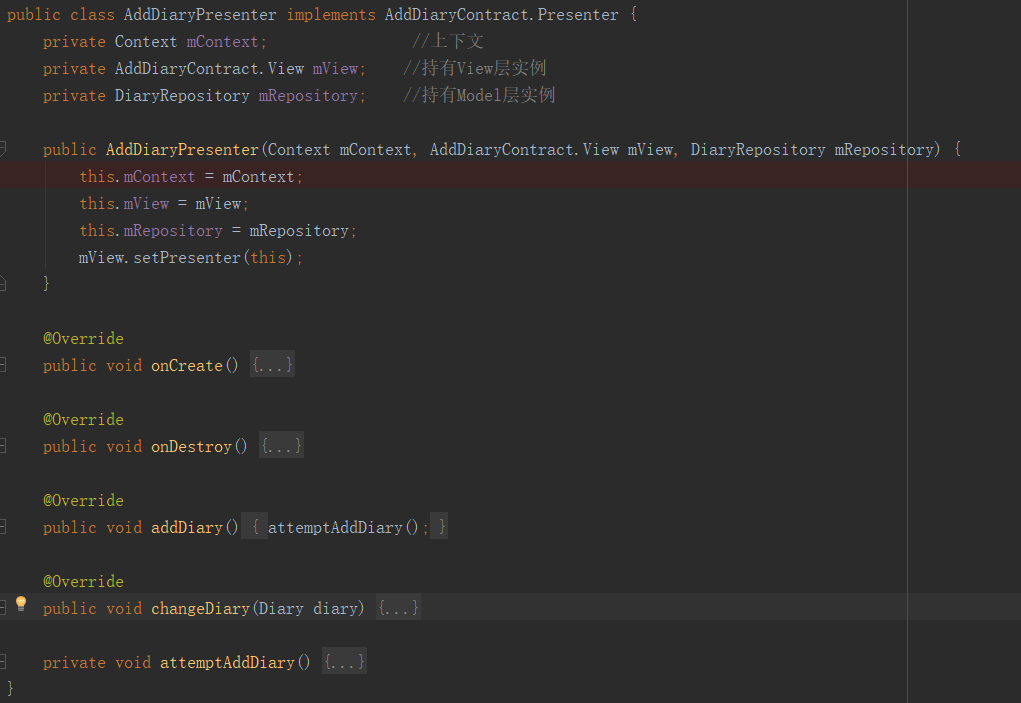
1. mvp的实现与组织

在官方架构示例中，activity作为全局的调度者负责创建View和Presenter，而这种方式在小型项目中过于复杂，故本项目采用activity可以作为View层的做法，缩短项目开发周期。

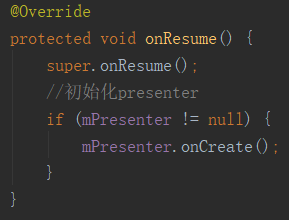
在View层实现相应View接口之后，通过setPresenter()方法为自己绑定相应的Presenter实例，从而可以调用Presenter中的方法来处理业务逻辑。



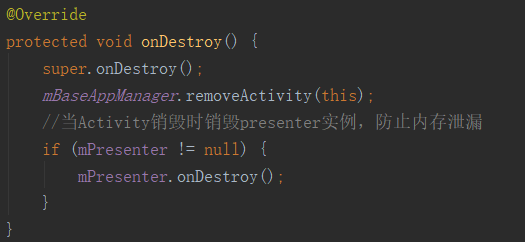
而在Presenter层中，子类实现Presenter父类接口之后代码如下：



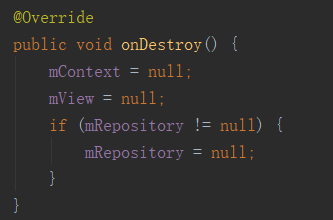
可以看到，在Presenter层的构造方法中，传入了view层实例和Model层实例，并且调用view层的setPresenter()方法将自己与View层绑定，但是这种互相持有对方实例的模式会有问题，在业务逻辑相当耗时的情况下，如果View层突然被用户销毁，由于其持有的Presenter层引用依然持有了自身的实例，导致view和presenter都无法被销毁，造成内存泄漏，这对于APP是致命的，在本项目中，我采用了将Presenter层与activity生命周期绑定的方法，在activity的onResume()方法调用presenter的onCreate()方法



在activity的onDestroy()中调用presenter的onDestroy()方法，在presenter的onDestroy方法中做引用的销毁工作，这样就不会出现内存泄漏的问题了。



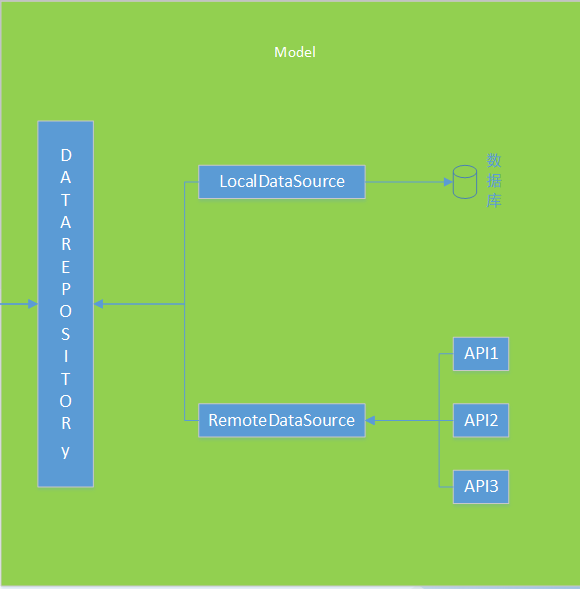
Activity的onDestroy方法



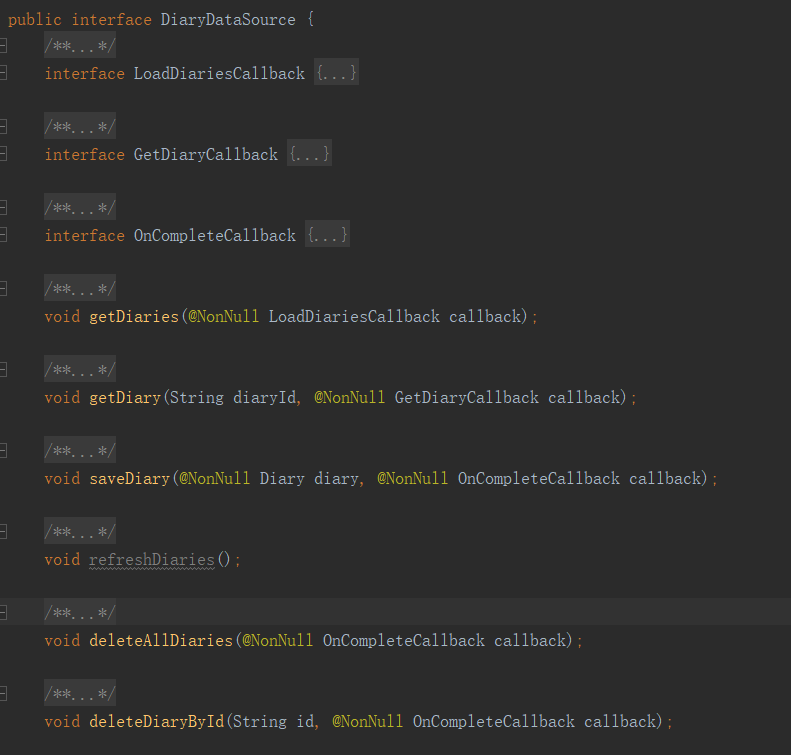
presenter的onDestroy方法

1. Model层的设计

Model层基本架构如图，可以看到，所有数据与外界的交互都是最终通过DataRepository这个层面来交互的，而DataRepository通过LocalDataSource来获取本地数据库的数据，通过RemoteDataSource来获取服务器上的数据（本项目只涉及获取本地数据），这样可以屏蔽上层对于数据的细节，一来上层无需关心数据的来源细节，二来底层可以根据需求修改，不会影响上层。

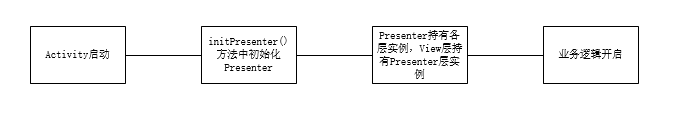


在具体的设计中，定义了一个DiaryDataSource的接口来统一两个层面对于数据的行为，接口定义如下：

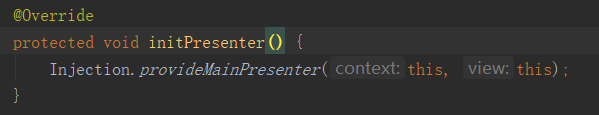


#### 架构小节

总结来说，具体项目的流程如下图：

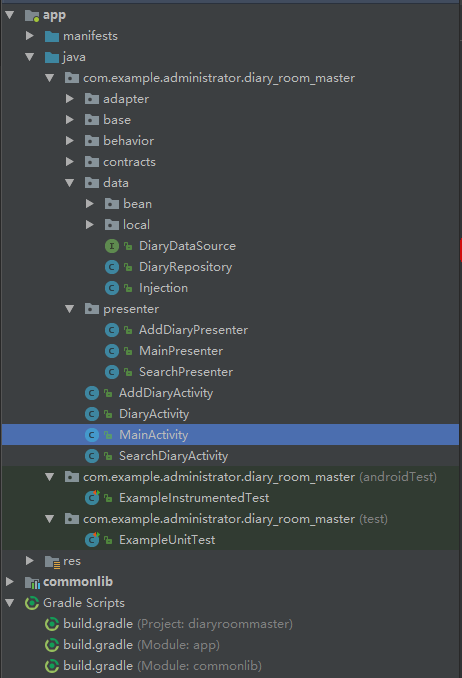


1. activity启动，在onCreate中调用initPresenter()方法
2. 在initPresenter中实例化Presenter

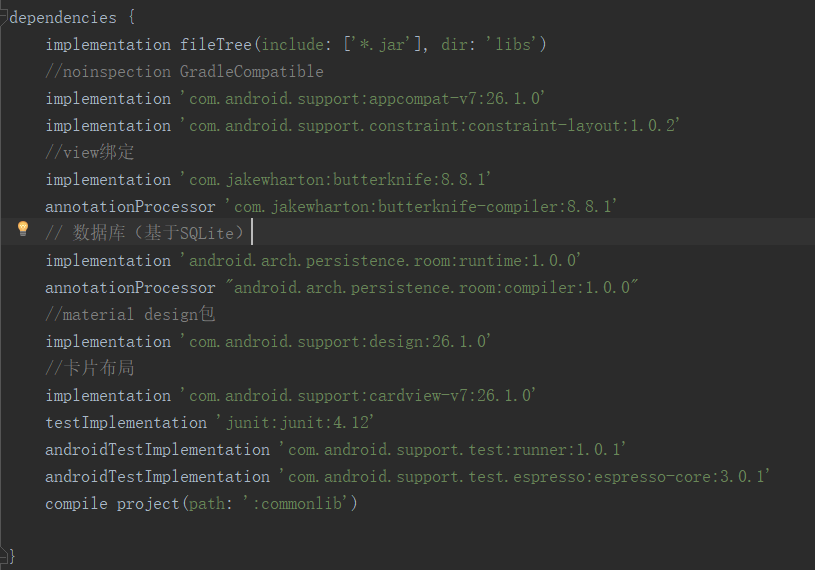


1. presenter的构造方法中会持有View层和Model层实例，调用setPresenter方法使得View层持有presenter实例，业务逻辑开启。

## 项目分包说明



整个项目包结构如图，其中commonlib为个人开发的快速开发库，主项目依赖这个模块（可以调用库中的方法），下面有三个build.gradle，分别是项目gradle，app gradle和模块gradle，在gradle的dependences中可以看到项目依赖的库。



* Adpter包中存放的是项目界面呈现列表（RecyclerView）所需要的适配器。
* Base包中存放的是各个基类
* Behavior中存放的是material design的行为类，这里实现的效果是向下滑动列表时添加日记的按钮（FloatingActionButton）会消失
* Contracts包中存放的是各个契约类
* Data包中存放Model层相关的类
* Presenter包中存放Presenter层的相关类
* 最外层存放Activity（View）