MVP的了解

最近开始启动Video项目，考虑到video这个项目，大部分情景可能就是请求或者查询数据，然后更新UI，以及视频播放。结合之前gotube的项目经验，我们决定在新项目使用MVP架构。所以，对MVP做了一些了解。

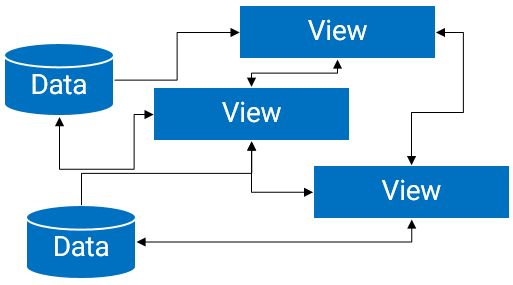
对于MVP（Model View Presenter），简单的说就是“MVC的演化版本”，“让Model和View完全解耦”。

MVC（Model View Controllor），其中View对应于布局文件，Model负责业务逻辑和实体模型，Controllor对应于Activity。这个View对应于布局文件，其实能做的事情特别少，实际上关于该布局文件中的数据绑定的操作，事件处理的代码都在Activity中，造成了Activity既像View又像Controller。

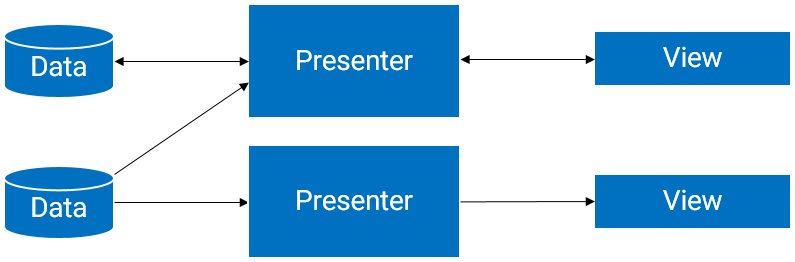
而当将架构改为MVP以后，Presenter的出现，将Actvity视为View层，Presenter负责完成View层与Model层的交互。MVP中View 对应于Activity，负责View的绘制以及与用户交互，Model 依然是业务逻辑和实体模型，Presenter 负责完成View于Model间的交互。其实我们就可以认为，从MVC到MVP的转变，其实可以认为就是减少了Activity的职责，简化了Activity中的代码，将复杂的逻辑代码提取到了Presenter中进行处理。与之对应的好处就是，耦合度更低，代码逻辑清楚。

如下图，可以清楚的体现MVP和MVC的区别：

MVC:



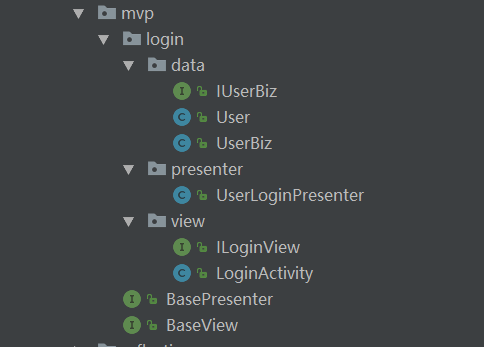
MVP:



其实最明显的区别就是，MVC中是允许Model和View进行交互的，而MVP中很明显，Model与View之间的交互由Presenter完成。还有一点就是Presenter与View之间的交互是通过接口的（代码中会体现)。

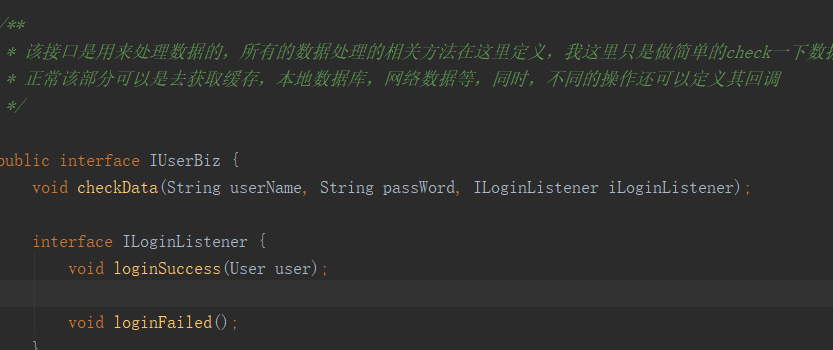
通过自己写了一个小demo，同时对比google官方的例子，来进行分析。

自己写了一个比较简单的登陆界面的小demo。先看一下代码结构：

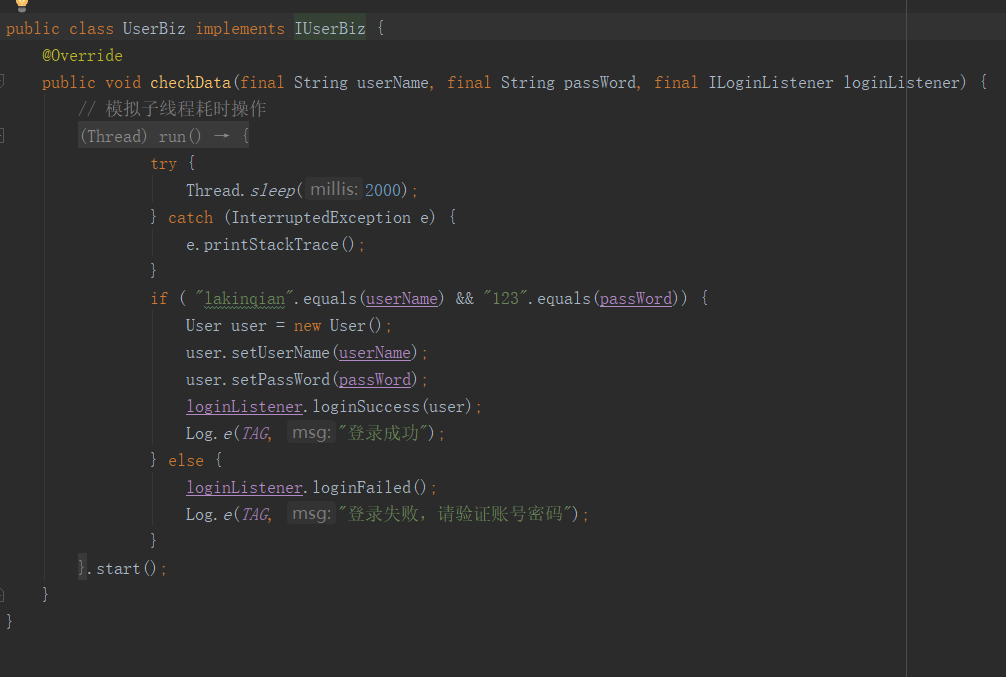


我比较喜欢每一个功能为一个package，里面包含MVP，如上图，login功能包含的一个mvp，不过有的情况下Model是整合应用共同使用一个，这个看情况。也可以login下面直接放MVP对应的类和接口，不要这么多层的包目录。

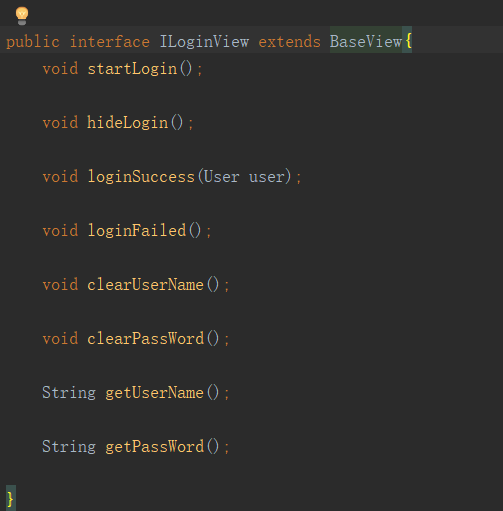
首先，demo中model层比较简单，只是伪装了一个数据验证，正常该部分可以是去获取缓存，本地数据库，网络数据等，同时，不同的操作还可以定义其回调。

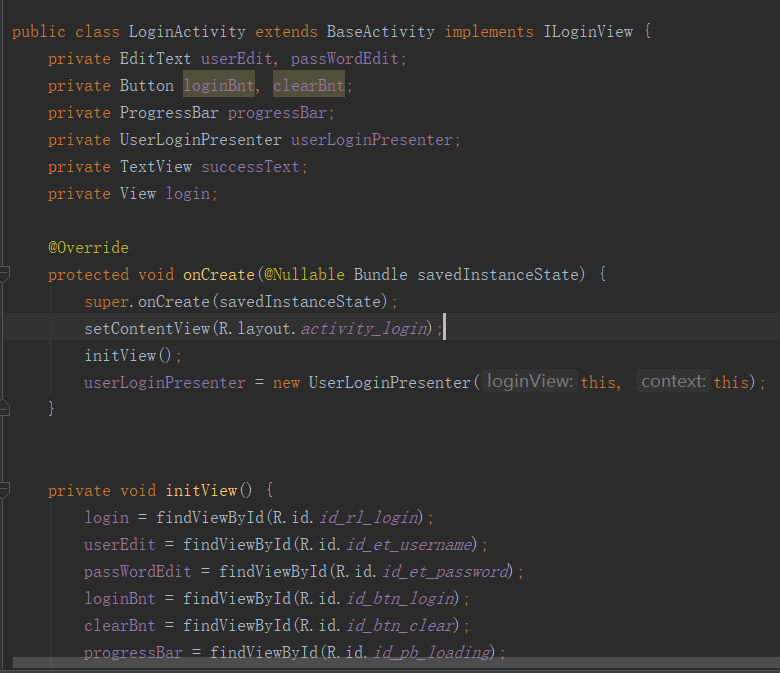


其实现类



再看view层，view层我这里其实就是一个activity，其他情况比如fragment等也是比较常见的。Activity实现用来作为view层的接口ILoginView，





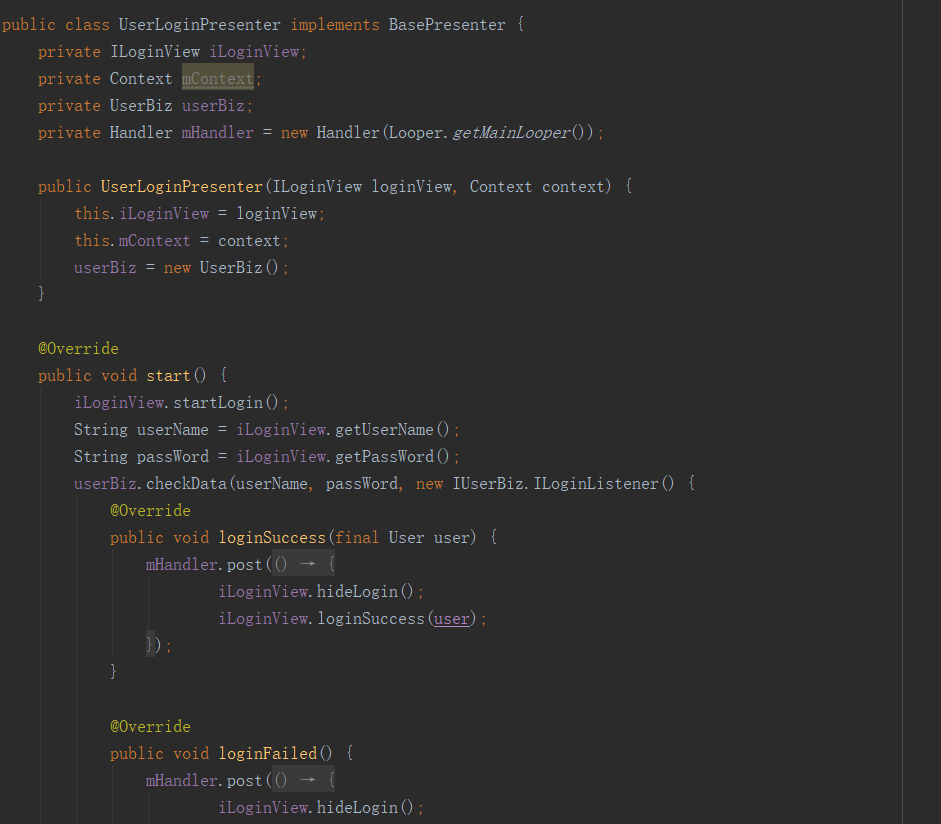
所有的UI更新都将在这一层完成，而且UI更新都是通过ILoginView这个接口里面实现的。

也就是说，ILoginView这个接口里面定义什么方法，一般是以对应的View层需要做什么UI更新为依据的。其他层对象不会持有里面的view去更改view的状态。但是该层不会有任何的逻辑判断，这样就是view层显得简洁。

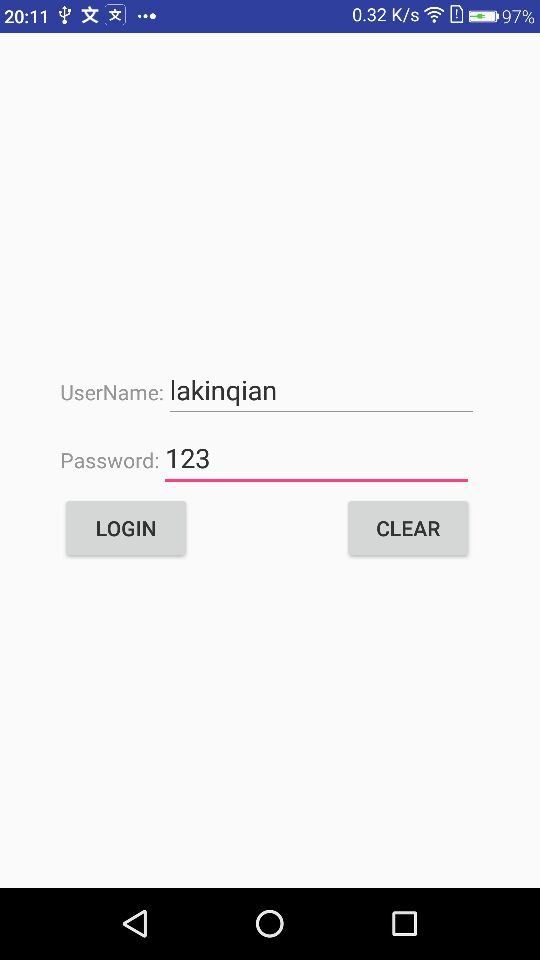
View层会持有一个Presenter对象，所有的更新view的逻辑是在Presenter里面实现的，Presenter会持有一个ILoginView对象，这样，当逻辑处理完后，通过持有的ILoginView对象，去调用在View实现的方法来更新UI，这样做到View层和Model层没有任何交互。

最后，是Presenter层，其实MVP中，Presenter应该是相对比较复杂的，因为他要处理模块的逻辑，还要负责沟通View层和Model层。Presenter层会持有一个ILoginView对象和一个加载数据的UserBiz对象，所以我们要尽量多的去把接口抽象出来，做到所有的操作，都有MVP这三个对象来操作，MVP各司其职，代码逻辑也比较好理解。

比如demo中，当最开始需要绑定数据时，View层持有的UserLoginPresenter对象，通过start方法，间接的通过UserBiz来加载数据，绑定数据过程中产生回调，再通过持有的ILoginView对象来更新UI,处理逻辑后产生的UI更新，也是通过ILoginView来更新的。总之，view层所有的数据获取，通过UserLoginPresenter对象的 UserBiz完成，View层所有的UI更新，通过UserLoginPresenter持有的ILoginView完成。



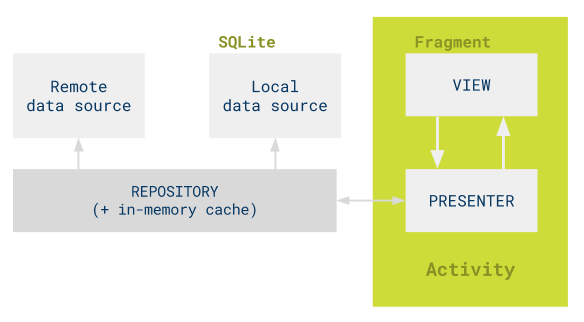
上面是我自己实现的一个简单demo。



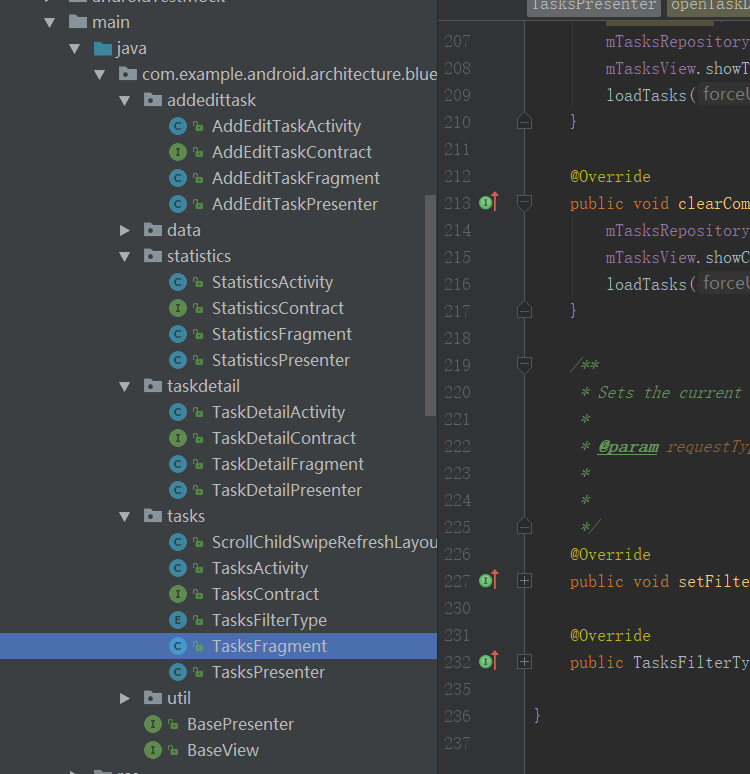
另外，也学习了一下google的demo，

github地址为https://github.com/googlesamples/android-architecture

官方给出的示例MVP流程图

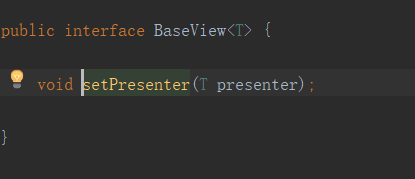
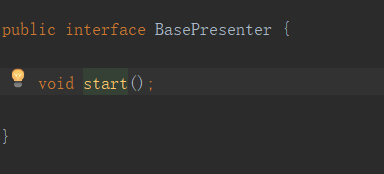


再看一下代码框架图，

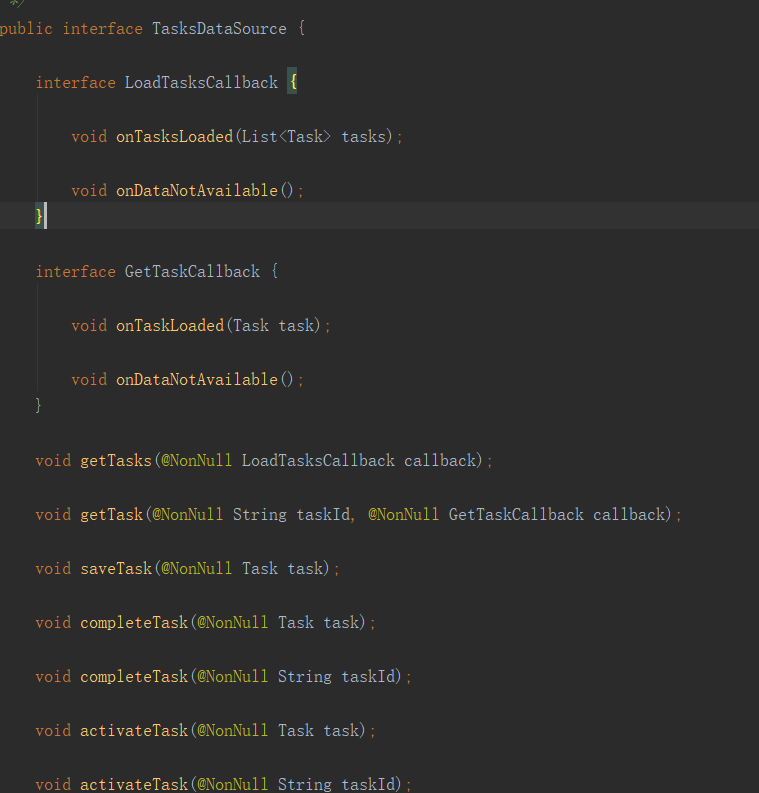


其中View层是fragment，而不是activity,Model层是数据的获取，其中四个模块tasks,taskdetail,statistics,addedittask分别对应列表页面，详情页面，统计页面，编辑页面四个模块，每个模块中fragment是V层，Presenter为P层，整个框架公用一个M层即data包。Contract中是View层和Presenter抽象出来的接口，我学习了tasks模块，其他模块也是一样的。

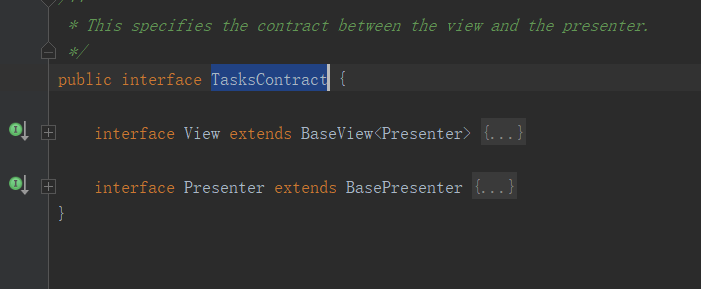
首先是View 和 Presenter 的基类 定义start setPresenter 方法。



Model 的基类可能跟View 的基类太一样， 因为只有所的模块公用一个Model模块，所以所有的Presenter都用同时用到这个Model 的基类。里面定义了加载数据使用的一些接口，同时定义了一下加载数据时候的Callback。

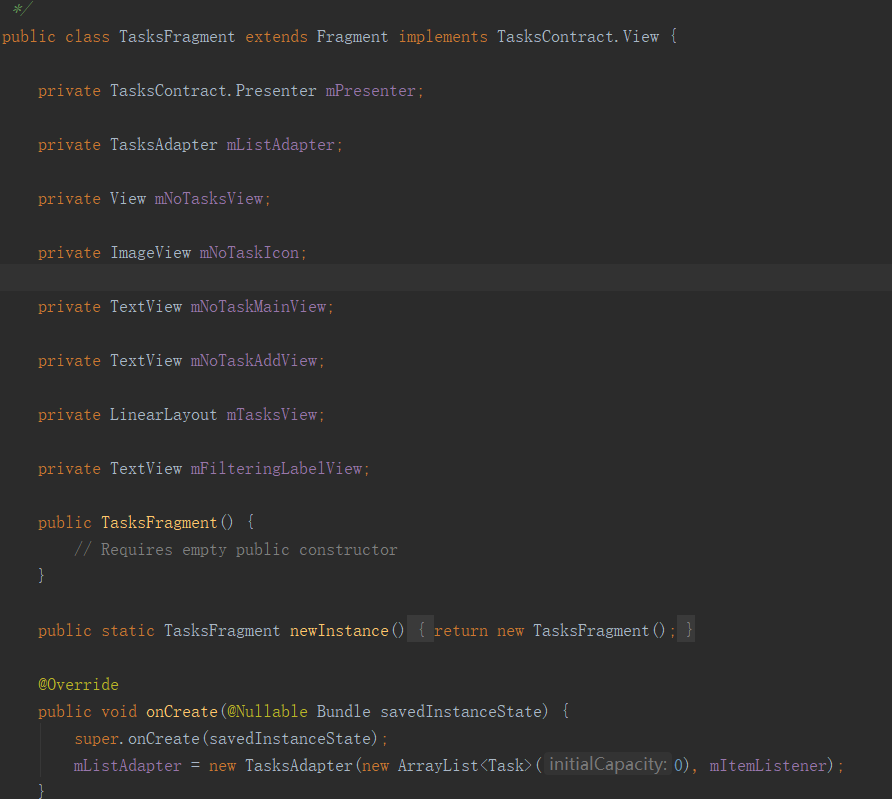


TasksContract

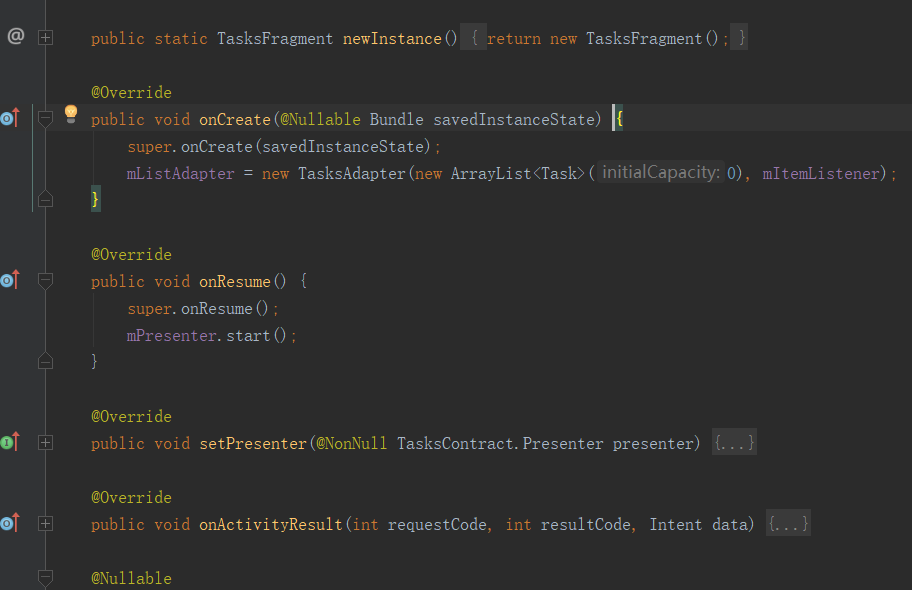


谷歌弄了一个契约类，是一个接口，定义内部接口继承View 和Persenter基类，里面抽象出来了大量方法，到时候V 和 P 直接实现这里面的接口就可以了，并且有什么方法都会在这里 确实是看方法的时候简洁明了。

TasksFragment



可以看到，fragment实现了契约类TasksContract.View，同时，还持有了一个TasksContract.Presenter。



整个fragment中，所有按钮点击后，处理逻辑都是通过持有的Presenter实现的，如下图，（代码太多，没法截全）



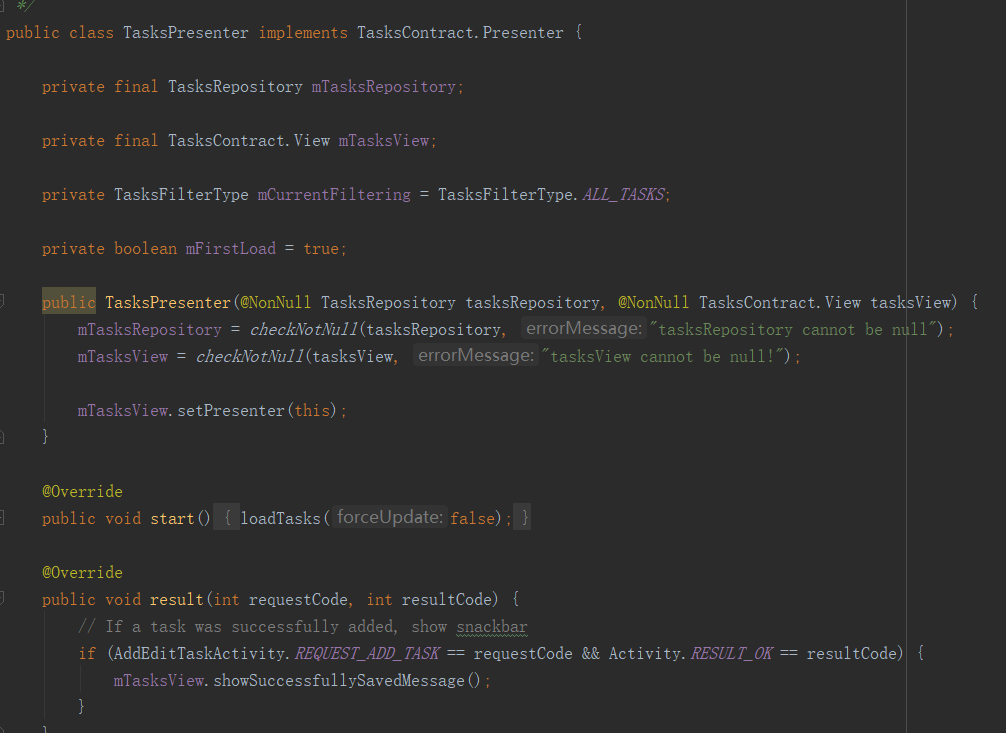
而fragment实现的TasksContract.View的抽象方法中，都是在做UI的更新，如下图（代码太多，没法截全）



所以，也就符合之前的逻辑，View层需要处理逻辑，通过持有的Presenter对象去处理逻辑，

同时Presenter会获得一个TasksContract.View对象，当逻辑处理完后，通过这个对象在View层实现UI更新。

TasksPresenter



很明显，TasksPresenter持有一个TasksContract.View和一个TasksDataSource对象，View层需要处理逻辑时，通过TasksPresenter来进行处理逻辑，TasksPresenter通过TasksDataSource处理数据相关，然后处理完逻辑后，通过持有的TasksContract.View去做UI更新。

TasksRepository

这是数据加载模块，这个模块主要是抽象出各种数据加载处理的情况，只能Presenter层进行交互。

