## 详细设计文档

## 目标

* 能够完成单条数据的加密解密
* 能够从普通文本文件中加载数据进行加密解密，并且将加密后的数据写入到文件
* 能够从Excel文件中加载数据进行加密解密，并且将加密后的数据写入到文件
* 使用Material Design设计语言优化界面

## 技术前提

### Data Binding

此项技术于2015年Google I/O大会上发布，而后在接下来几年内逐步完善。

The Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.

以上是官方对Data Binding技术的简介，一言以蔽之，Data Binding是一种将布局文件和数据绑定的库。

#### 为什么要用DataBinding？

* 开发的更快，不必再写findViewId()这样的代码
* 原生支持，已经集成到了SDK中，不必引入第三方库

#### 使用介绍

<?xml version="1.0" encoding="utf-8"?>  
<layout xmlns:android="http://schemas.android.com/apk/res/android">  
 <data>  
 <variable name="user" type="com.example.User"/>  
 </data>  
 <TextView android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="@{user.lastName}"/>  
</layout>

上述是单向绑定的一个简单实例，首先上述实例使用新的前缀<layout>，这是DataBinding的新格式；

<data><variable name="user" type="com.example.User"/></data>，

这句话引入了User类，并且生成了一个变量user；

android:text="@{user.lastName}"指定TextView显示的字符；

然后在代码中将User实例注入：

mBinding.user = new User();

其中mBinding是自动生成代码的实例，每个使用了DataBinding格式的布局文件都会生成这样的类，让开发者访问布局的变量和视图；

接下来的问题就是：

1. 后台数据更新，反馈到视图
2. 用户输入，反馈到Model数据

public class LoginViewModel extends BaseObservable {  
 // private Model data = ...  
  
 @Bindable  
 public Boolean getRememberMe() {  
 return data.rememberMe;  
 }  
  
 public void setRememberMe(Boolean value) {  
 // Avoids infinite loops.  
 if (data.rememberMe != value) {  
 data.rememberMe = value;  
  
 // React to the change.  
 saveData();  
  
 // Notify observers of a new value.  
 notifyPropertyChanged(BR.remember\_me);  
 }  
 }  
}

上述是实现了**双向绑定**的Model代码，首先第一个问题："后台数据更新，反馈到视图"，更新数据，将会调用set()方法，在检查输入值之后，通知数据变化，然后DataBinding通过注解调用get()获取新的数据；其次，第二个问题：”用户输入，反馈到Model数据“，每当用户输入时，DataBinding会不断的调用set()方法，更新数据到Model；

### MVP设计模式

MVP是由经典的模式MVC演变而来，它们的基本思想有相通的地方：Controller/Presenter负责逻辑的处理，Model提供数据，View负责显示。

同MVC模式不同的是，MVP设计模式中View并不直接使用Model，它们之间的通信是通过Presenter (MVC中的Controller)来进行的，所有的交互都发生在Presenter内部，而在MVC中View会直接从Model中读取数据而不是通过 Controller。

具体区别请看图例：

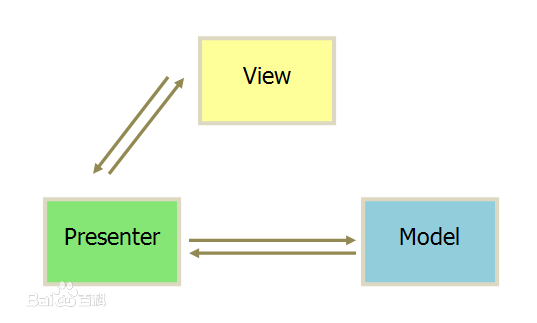


图 1.1 MVP 设计模式简例

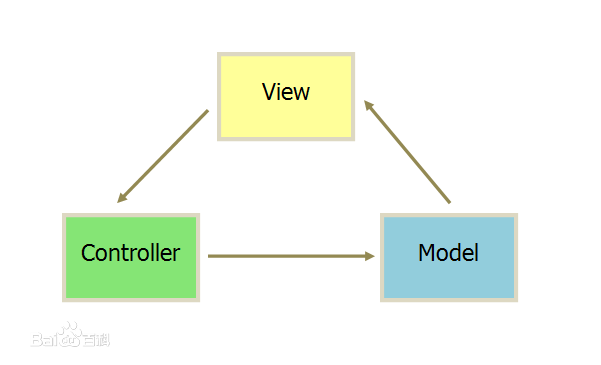


图 1.2 MVC设计模式简例

优点：

1. 模型与视图完全分离，我们可以修改视图而不影响模型
2. 可以更高效地使用模型，因为所有的交互都发生在一个地方——Presenter内部
3. 我们可以将一个Presenter用于多个视图，而不需要改变Presenter的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁。
4. 如果我们把逻辑放在Presenter中，那么我们就可以脱离用户接口来测试这些逻辑（单元测试）

缺点：

1. 对视图的渲染全放在了Presenter中，所以View和Presenter之间的交互会过于频繁。
2. 同样一旦视图变更，Presenter也有可能变更。

#### 改良

诚然，MVP设计模式有许多优点，但是它的缺点也同样明显；

这也就是说，View和Presenter之间的耦合度较高，两者之间需要解耦合，说到解耦合，我们通常想到抽象，再次思考这个问题，“视图变化引起逻辑变化”，视图变化通常由xml布局文件负责，逻辑变化也就是行为的变化，由此我们可以想到使用接口来实现View和Presenter之间的解耦合。

首先我们需要为View和Presenter分别生成一个接口，为了避免杂乱无章，我们将这两个接口声明在同一个文件中Contract；

在View接口中，声明与视图更新相关的方法，例如更新列表等操作。

在Presenter接口中，声明与业务逻辑相关的方法，例如发送请求获取列表等操作；

在这两个接口具体的实现类中，都分别持有对方的的接口实现类实例；

具体结构如下图：

图 1.3 MVP解耦合实例

从上往下看，可以发现我们在Contract接口中，定义了View和Presenter接口；Presenter类实现了Contract.Presenter 接口，同时持有Contract.View接口；DecryptionFragment实现了Contract.View接口，同时持有Contract.Presenter接口。

由此整个操作过程，检测到用户输入，View调用Presenter中的方法，Presenter执行方法完成业务后，将调用View中的方法，完成视图的更新；

通过上述操作，我们成功将View和Presenter之间解耦合；

### RxJava

RxJava is a Java VM implementation of [Reactive Extensions](http://reactivex.io): a library for composing asynchronous and event-based programs by using observable sequences.

即，使用可观察序列编写异步程序和基于事件的程序的库。这是大名鼎鼎的ReactiveX的JVM实现；这里面有三个关键词：**可观察序列**、**异步**、**基于事件**。

1. **可观察序列**，被观测的一组数据或者事件。
2. **异步**，包括两方面，一个时间，一个空间；可以是延后操作，也可以是切换线程操作。
3. **事件**，包括但不仅限于点击事件、触摸事件。

import io.reactivex.functions.Consumer;  
  
Flowable.just("Hello world")  
 .subscribe(new Consumer<String>() {  
 @Override public void accept(String s) {  
 System.out.println(s);  
 }  
 });

上述是一个简单地实例，详细情况请访问https://github.com/ReactiveX/RxJava/wiki

### Kotlin

Kotlin由JetBrains公司开发，并且在2017 Google I/O大会上成为官方推荐的开发语言。

Kotlin语法简洁，功能强大；现在越来越多的开发者使用了这种语言。Kotlin语法上有一部分类似脚本语言，所以它同样可以被编译为JS。

官网上所列举的优点如下：

1. Concise, Drastically reduce the amount of boilerplate code
2. Safe, Avoid entire classes of errors such as null pointer exceptions.
3. Interoperable, Leverage existing libraries for the JVM, Android, and the browser.
4. Tool-friendly, Choose any Java IDE or build from the command line.

详细的语法说明将会在一份新的文件中说明；

## 实现说明

在编写功能代码之前，考虑到许多也页面的行为都是相似的，比如显示消息和显示加载对话框等等；因此可以写一个公共类，将相关代码提取到这个公共类中；

下面是这两个类的结构：

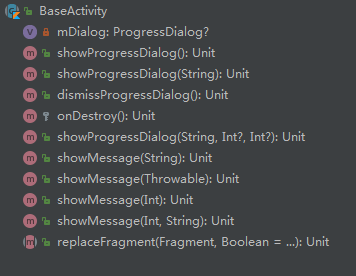
**BaseActivity** 

图 1.4 BaseActivity

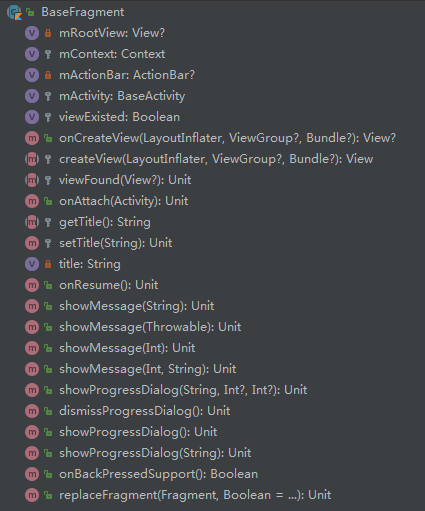
**BaseFragment** 

图 1.5 BaseFragment

它们的函数名称是显而易见的，就不在赘述。

值得一提的是，应用中的所有页面大多由Fragment实现，只有一个Activity作为Fragment的容器；

FunctionActivity的布局文件十分简单，一个ToolBar，一个DrawerLayout，DrawerLayout内部是一个FrameLayout作为Fragment的容器。

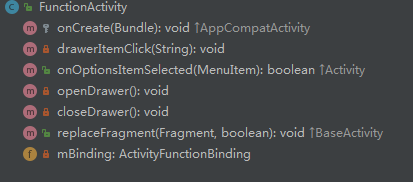


图 1.6 FunctionActivity

有几个函数值得说一下：

* drawerItemClick：抽屉侧边栏菜单项的点击事件，点击不同的菜单项替换不同的Fragment。
* openDrawer： 延迟打开侧边抽屉。
* closeDrawer： 延迟关闭侧边抽屉。
* replaceFragment： 实现父类BaseActivity的抽象方法，用于Fragment的切换，布尔量表示是否将Fragment入栈。

#### 单条数据的加密解密

本页面的实现使用了上面提到的MVP设计模式；

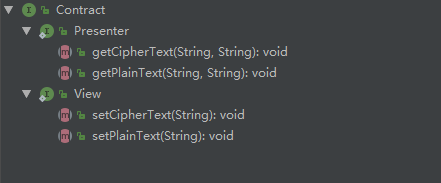


图 1.7 Contract

* Presenter
* getCipherText：通过明文和key生成密文；
* getPlainText： 通过密文和key解密生成明文；
* View

setCipherText：将生成的密文显示在视图；

setPlainText： 将生成的明文显示在视图；

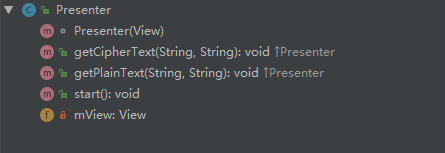


图 1.8 Presenter

实现了上述Presenter接口的逻辑。

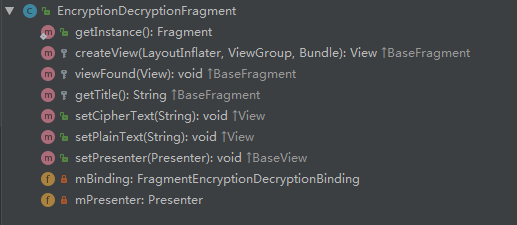


图 1.9 View

本类实现了View接口，继承了BaseFragment，实现了抽象类中的抽象方法。

三者之间的关系用类图表示如下：

从上往下看，在Contract接口中定义了两个接口View和Presenter，这两个接口分别声明了自己的方法，View声明了关于UI的方法，Presenter声明关于逻辑的方法。

然后Presenter类实现了Contract.Presenter接口，并且在构造函数中接收Contract.View作为字段。EncryptionDecryptionFragment继承了BaseFragment，实现了Contract.View 接口，并且拥有Contract.Presenter作为字段。这样作为Contract.Presenter和Contract.View的实现类，双方都持有对方的引用，便于交互。

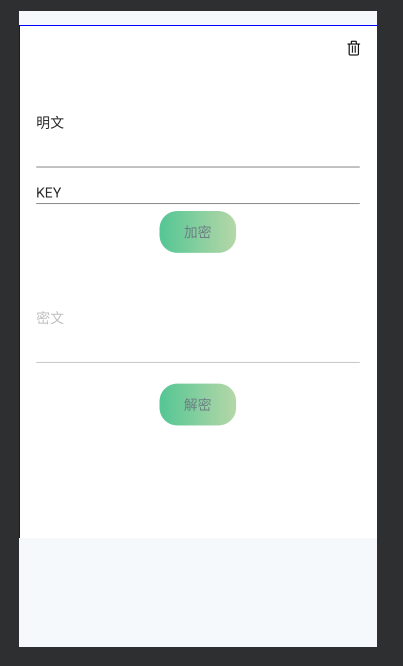
#### 从Excel文件中批量读取进行加密解密

涉及到Excel文件的写入和读取，这就需要寻求开源社区的帮助；经过一番搜索，Apache POI - the Java API for Microsoft Documents进入我的视野，在Maven Repository上的最后一次更新是在2018年的12月，并且在官网有着开始支持JDK 11 的消息，所以我选择这个API。

#### 列表显示

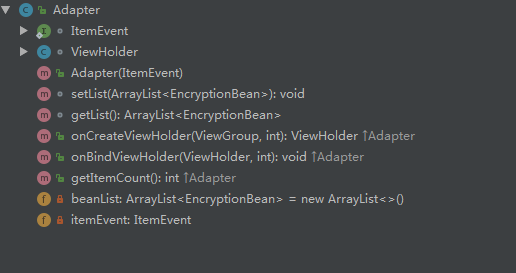
因为是从Excel文件中批量读取，所以需要把所有的数据用列表的形式显示出来。这样就需要先将子项的视图编写好。

子项视图总体采用垂直的线性布局，实现草图如下：



右上角的删除按钮实现删除该子项的功能，从文件中读取的明文和key会显示在编辑框中以供编辑，点击加密后，会根据明文和key生成密文显示在下方；点击解密后，会根据明文和key解密出明文；值得一提的是子项的数据使用的**双向绑定**；

有了子项，还需要为列表编写适配器，这里列表视图我选用了RecyclerView，适配器的代码结构如下：



首先是定义的ItemEvent接口，这是子项的点击事件，将会被Fragment类实现；

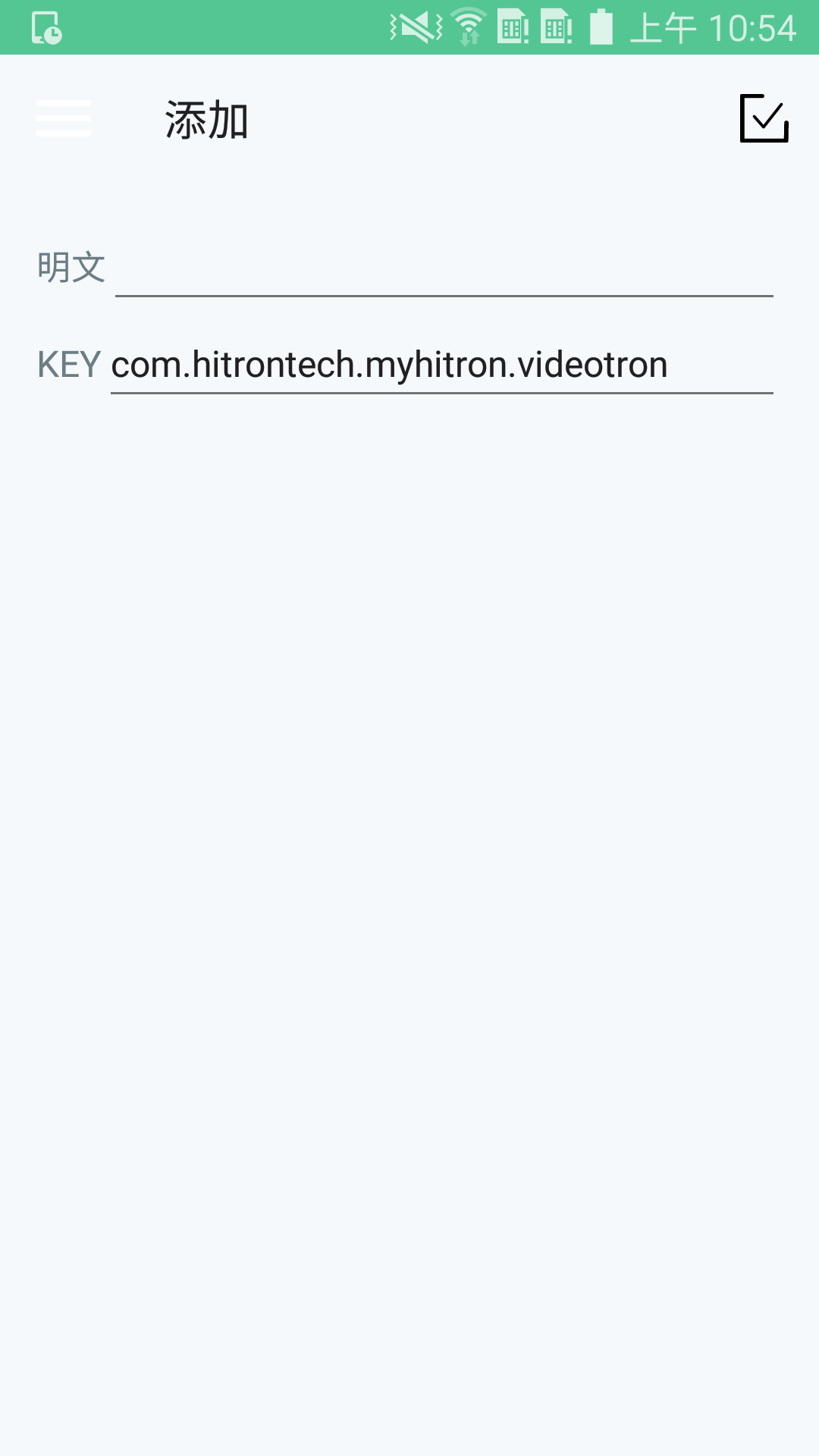
ViewHolder继承RecyclerView.ViewHolder，编写了自己的视图绑定和事件绑定函数。

和Fragment之间关系如下：

可以看到，ItemEvent接口定义在Adapter，FileEncryptionDecryptionFragment实现了该接口，最后FileEncryptionDecryptionFragment持有Adapter引用；实现了交互。

#### 添加子项

这个页面的布局文件比较简单，不在赘述，效果图如下：



默认明文为空，默认key为com.hitrontech.myhitron.videotron，点击提交后，会在列表中查重，若是有重复的则弹出提示，若是没有重复的则添加到列表，并且返回上一界面。因此我们需要重写EncryptionBean的equals方法，当明文和key相等时，就认为这两个对象相等；