**安卓MVP浅析及Presenter的生命周期问题**

**m :  model 模型.**

很多人理解成bean对象,我觉得不全是, 用获取数据比较贴切.

在以前写mvc的时候,我习惯抽成initData()来表示modle.

public interface  Model {

      Bean load(String url,params)

}

Public interface Modle{

@GET(Constant.URLS.LOGIN)

Observable<BaseResponse<LoginBean>> login(@QueryMap HashMap<String, String> map, @Query(“version”) int version);

}

**v:view视图**

在mvp中,有人把activity/fragment理解成presenter,

我觉得还是理解成v好一点,

比较纠结是,activity有生命周期,怎么就是单纯的v呢.

但如果理解成presenter的话,那么和mvc区别就不大了吧.

具体使用:

1.写一个view接口

public interface View {

  //抽取相对的activity里面的各种行为

  //如:

  void show();

  void bindData(Bean b);

}

2.activity实现这个设计好的view接口

public class MainActivity extends Activity implements View{

    Presenter presenter;

    @Override

    protected voidonCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        presenter = new Presenter(this);

    }

     public void show(){

     }

     public void bindData(Bean b){

          ........

         //等待数据,设置view.

     }

}

3.实现每个行为的逻辑:对某个控件做什么事.

public void show(){

  //具体操作.比如弹个toast啥的,设置某个控件啥的

}

4.持有presenter

这里有点难理解,持有presenter是什么意思?

我们不管做什么操作,都是一次事件

而这个事件由谁接收,当然是activity接收了.

如果activity不持有presenter,怎么告诉presenter,我需要获得数据.

**对于view的功能设计**

想好这个界面有哪些操作了.这个也不是很难,

拿着效果图,分析一下就出来了,

比如textview设置个内容啊,点击事件啊,选中啊,button弹出对话框啊

就拿TextView来说吧,Textview这个控件是不是有settext(),setColor().setTextsize().....方法.

在mvp模式里面,我们把activity理解成View,假设Activity里面有settext(),setColor().setTextsize()方法

我们写这个view接口就相当于把这些方法从Activity抽取出来,

这个接口和Activity写好了一般是不需要改的,除非界面大改.那没办法,只能两边一起改了

因为xml都改了,控件啥的都改了,对应的Activity和view接口能不改么.

不过由于分离开了,改起来也简单.因为没有逻辑操作.

**p:控制 presenter**

一般写法:

public interface Presenter {

void loadData();

}

public interface PresenterImpl implements Presenter {

View v;

    public PresenterImpl(View v){

          this.v = v;

    }

    publc void loadData(){

         Modler  modle   =    new  Modle();//创建modle

         Bean  bean =  modle.load("url",params);//获得数据

         v.bindData(bean);//数据绑定view

    }

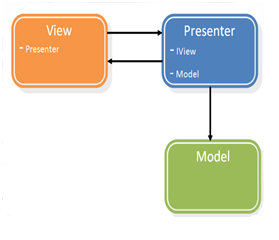
}

既然view关心本身有哪些行为,那么p就是控制view了

通过事件触发,presenter去连接model获得数据,然后将model与view绑定,

也可以只是调用view的单个或多个操作.

结构图:



**总结:**

presenter和view相互持有调用,presenter可以同时操作Modle和View,但是Modle和View之间不能调用.

mvc的纠结的地方就是,其实只有mv,没有c,因为c和v是几乎是一个东西,activity/fragment嘛.

很多时候除了bean,mvc其实整个都在Activity里面- -

**Presenter生命周期**

这个问题很揪心,我想很多人懵逼的地方就在这

举个例子:

Presenter控制逻辑,也就是控制网络操作,绑定数据,一系列逻辑都在这.

那么,当activity关闭以后,Presenter怎么处理网络请求,异步操作呢?

比如上面的loadData(),如果acitivity已经关闭了,而网络操作又没走完.

就会内存泄漏.会不会空指针不好说.

view虽然持有p,但是也不能在Activity的onDestroy里面直接的将p=null吧

对象=null也只是断开引用而已,还并不是真的释放.

这样没用的,而且p还持有view呢,异步也不会因此结束,

所以得有个接口告诉p,我要挂了,你也自杀吧.

所以:

interface BasePresenter {

void onStart();

void onDestroy();

}

让所有的Presenter都继承BasePresenter,然后在activity中相应的生命周期里面调用

在相应的方法里面,初始化,结束异步操作,释放资源,将view=null;

而在activity里面,由于Presenter并没有view的引用了,

所以p随着activity的销毁也就跟着销毁了.不会造成上下文的泄漏等.

**mvp的优点:**

1.真正用起来,会发现模块很清晰,

2.解耦,模块清晰,耦合度自然就低了,但是耦合度低不代表改起来就很方便,哈哈

需求改动大的话,其实改起来还是麻烦,不过比起在上千行的activity里面改代码,还是舒服很多的,

比较推荐的写法:

public class Contract {

    public interface View {

        void show();

        void bindData();

   }

    public interface Presenter extends BasePresenter{

         void loadData();

    }

   public interface  Model {

      void load(String url,params)

}

}

用一个Contract集中对应acitivity的所有mvp用到的接口,

所有功能一目了然,改起来也方便.

如果改需求,肯定先改接口,再去改实现类嘛.

android studio 已经有mvpHelper插件,可以很方便生成的mvp的接口类以及实现类.

**mvp的缺点:**

类会非常多,每一个acitivity或者fragment要对应1个或者多个Presenter,真的很多

小项目费力,有时不如mvc方便,界面如果非常简单,用mvp反而显得繁琐了.