Android架构——MVP

一、MVP介绍

随着UI创建技术的功能日益增强，UI层也履行着越来越多的职责。为了更好地细分视图(View)与模型(Model)的功能，让View专注于处理数 据的可视化以及与用户的交互，同时让Model只关系数据的处理，基于MVC概念的MVP(Model-View-Presenter)模式应运而生。

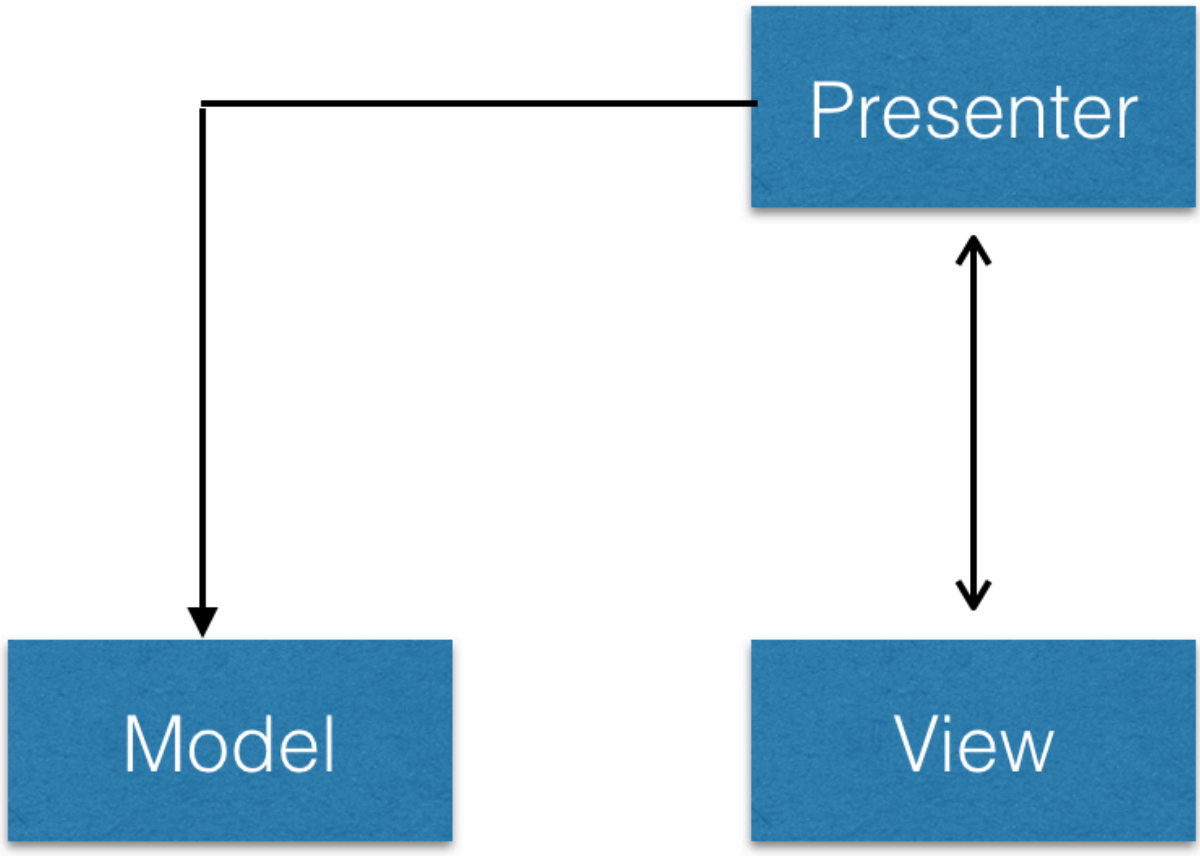
MVP模式里通常包含4个要素：

- **View**：负责绘制UI元素、与用户进行交互(在Android中体现为Activity);
- **View Interface**：需要View实现的接口，View通过View interface与Presenter进行交互，降低耦合;
- **Model**：负责存储、检索、操纵数据;
- **Presenter**：作为View与Model交互的中间纽带，处理与用户交互的负责逻辑。

二、为什么使用MVP模式

在Android开发中，Activity并不是一个标准的MVC模式中的Controller，它的首要职责是加载应用的布局和初始化用户界面，并接受并处理来自用户的操作请求，进而作出响应。随着界面及其逻辑的复杂度不断提升，Activity类的职责不断增加，以致变得庞大臃肿。当我们将其中复杂的逻辑处理移至另外的一个类（Presneter）中时，Activity其实就是MVP模式中 View，它负责UI元素的初始化，建立UI元素与Presenter的关联，自己处理一些简单的逻辑，复杂的逻辑交由 Presenter处理。

三、MVP的特点

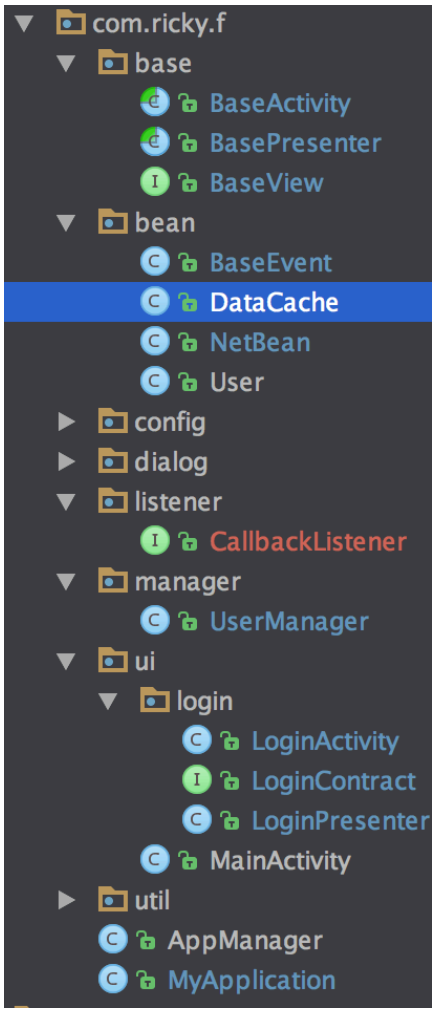


- View不直接与Model交互，而是通过与Presenter交互来与Model间接交互;
- Presenter与View的交互是通过接口来进行的，更有利于添加单元测试;
- 通常View与Presenter是一对一的，但复杂的View可能绑定多个Presenter来处理逻辑。

四、MVP实践

我们通过一个用户登录的例子来了解一下MVP模式

先来看看目录结构：



- base：基类包
- bean：实体包
- manager：api包
- ui：与View相关

1、当用户点击登录后首先会执行 **LoginPresenter** 中的login(...)方法，当收到结果后通过请求时的tag来处理不同的结果，并通过View接口通知UI层处理后面的逻辑：

```
public class LoginPresenter extends LoginContract.Presenter {

    UserManager userManager = new UserManager();

    public User user;

    @Override
    public void login(String name, String password) {
        mView.showLoadingDialog("请稍候...");
        userManager.login("ricky", "111", callbackListener);
    }

    @Override
    protected void success(NetBean bean) {
        if(UserManager.Login.equals(bean.getTag())){
            mView.dismissLoadingDialog();
            user = FastJsonUtil.getBean(bean.getData(), User.class);
            mView.loginSuccess();
            return;
        }
        if(UserManager.Register.equals(bean.getTag())){

            return;
        }
    }

    @Override
    protected void failure(String tag, int errCode, String message) {
        super.failure(tag, errCode, message);
        mView.showToast(tag + " " + message);
    }
}
```

2、在 **LoginPresenter** 中会调用 **UserManager** 中的login(...)方法发起HTTP请求，并将HTTP请求返回的结果进行封装返回到 **LoginPresenter** 中：

```
public class UserManager {

    public static final String Login = "Login";
    public static final String Register = "Register";

    public void login(String name, String password, CallbackListener callbackListe
ner) {
        HttpUtils.getInstance().send(HttpUtils.HttpMethod.GET, Login, AppConfig.BA
SE_URL, callbackListener);
    }
}
```

3、数据回到 **LoginPresenter** 后首先交由其父类 **BasePresenter** 处理，如果请求成功则执行 success(...)抽象方法，如果失败则执行 failure(...)方法，由父类优先处理：

```
public abstract class BasePresenter<V> {

    /**
     * 内存不足时释放内存
     */
    protected WeakReference<V> mViewRef;
    protected V mView;

    public void attachView(V view) {
        mViewRef = new WeakReference<>(view);
        mView = mViewRef.get();
    }

    protected BaseView getBaseView() {
        return (BaseView) mView;
    }

    //用于在activity销毁时释放资源
    public void detachView() {
        if (mViewRef != null) {
            mViewRef.clear();
            mViewRef = null;
        }
    }

    protected CallbackListener callbackListener = new CallbackListener() {
        @Override
        public void onResult(NetBean netBean) {
            if(netBean.isOk()){
                success(netBean);
            } else {
                unifyErrHandle(netBean.getTag(), netBean.getCode(), netBean.getMes
sage());
            }
        }
    };

    /**
     * 统一处理异常
     */
    private void unifyErrHandle(String tag, int errCode, String message) {
        /**
         * 未登录跳转到登录
         */
        switch (errCode) {
```

```
            case ErrCode.UNAUTHORIZED:
                getBaseView().openActivity(RouterSchema.LoginActivity, RequestCode
.REQUEST_LOGIN);
                break;
            default:
                failure(tag, errCode, message);
                break;
        }
    }

    protected abstract void success(NetBean bean);

    protected void failure(String tag, int errCode, String message) {
        getBaseView().dismissLoadingDialog();
    }
}
```

可以看到，View只负责处理与用户进行交互，并把数据相关的逻辑操作都扔给了Presenter去做。而Presenter调用Model处理完数据之后，再通过接口更新View显示的信息。