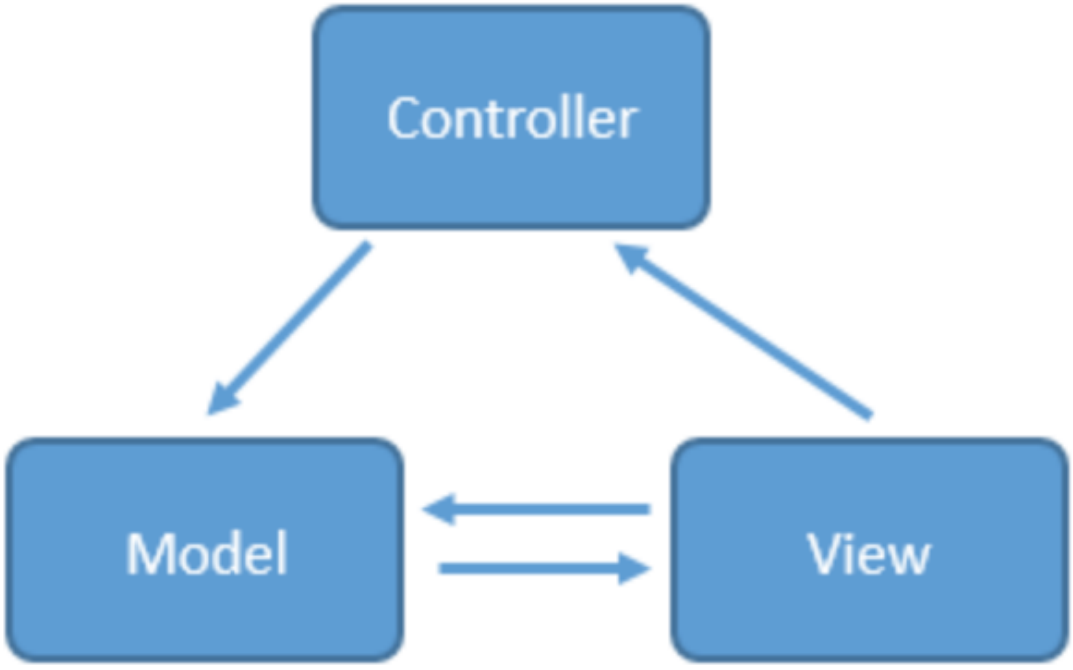
转载请注明链接：https://blog.csdn.net/feather_wch/article/details/79729132

Android架构(MVC/MVP/MVVM与模块化、组件化)

版本：2019/3/2-1(10:00)

MVC

1、Android中MVC是什么？特点?

> 1. `Model` :针对业务模型建立的数据结构和类（与View无关，只与业务相关）
> 2. `View`：`XML/JAVA` 或者 `JS+HTML` 进行页面的显示。
> 3. `Controller`：`Android` 的控制层通常在 `Activity、Fragment` 之中。
>
>   本质就是 `Controller` 操作 `Model` 层的数据，并且将 `数据` 返回给 `View` 层展示。
>
> 
>

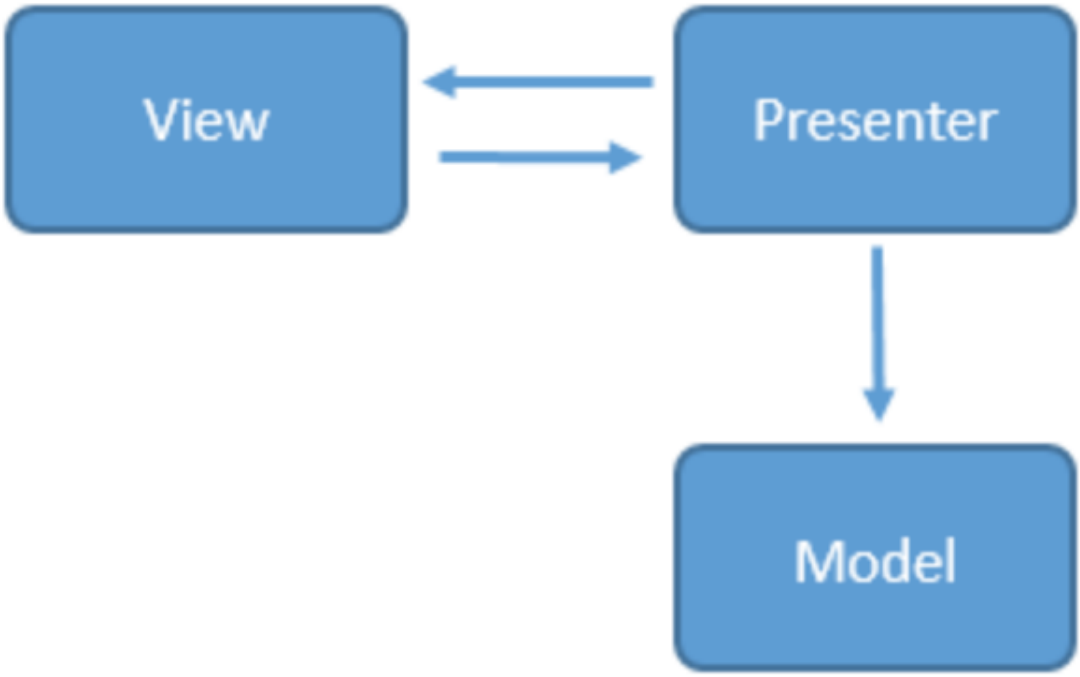
2、Android的MVC的缺点：

1. Activity 并不是 MVC 中标准的 Controller ， 既有 Controller 的职责也有 View 的职责，导致 Activity 的代码过于臃肿。
2. View层 和 Model层 互相耦合，不易于开发和维护。

MVP

3、Android中的MVP模式

1. MVP(Model-View-Presenter)
2. Model ： 主要提供数据的存储功能。 Presenter 需要通过 Model 存取数据。
3. View：负责处理 点击事件和视图展示（Activity、Fragment或者某个View控件 ）
4. Presenter ： View和Model 之间的桥梁，从 Model 检索数据后返回给 View 层。使得 M/V 之间不再有耦合关系。



4、MVP和MVC的区别

1. MVP 中绝对不允许 View 直接访问 Model
2. 本质是 增加了一个接口 降低一层 耦合度

5、MVP的特点

1. Presenter 完全将 Model 和 View 分离，主要逻辑处于 Presenter 中。
2. Presenter 和 具体View 没有直接关联，通过定义好的 接口 进行交互。
3. View 变更时，可以保持 Presenter 不变(符合面向对象编程的特点)
4. View 只应该有简单的 Set/Get 方法、用户输入、界面展示的内容，此外没有更多内容。

6、MVP的缺点

> 1. MVP 的中使用了接口的方式去连接 view层 和 presenter层 ， 如果有一个逻辑很复杂的页面，接口会有很多，导致维护接口的成本非常大。
> 2. 解决办法 ： 尽可能将一些通用的接口作为基类，其他的接口去继承。

7、MVP的实现

> Model层

```
//Model层-数据的实体类：NetInfo.java
public class NetInfo {
    private int code;
    private String msg;
    public NetInfo(int code, String msg){
        this.code = code;
        this.msg = msg;
    }
    public int getCode() {
        return code;
    }
    public void setCode(int code) {
        this.code = code;
    }
    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }
}
//Model层-请求数据时View和Model的交互接口(中间层Presenter去实现)：LoadTasksCallBack.java
public interface LoadTasksCallBack<T> {
    void onSuccess(T data);
    void onFailed();
}
//Model层-任务抽象基类【业务接口】：  NetTask.java
public interface NetTask<T> {
    void execute(T data, LoadTasksCallBack callBack);
}
/**==========================================
 * Model层核心-具体任务【业务的实际操作】:
 *   1. 实现Model层获取数据的操作.
 * //NetInfoTask.java
 *==========================================*/
public class NetInfoTask implements NetTask<String>{
    @Override
    public void execute(String ip, LoadTasksCallBack callBack) {
        if("192.168.1.1".equals(ip)){
            callBack.onSuccess(new NetInfo(1, "This is a Msg from " + ip));
        }else{
            callBack.onFailed();
        }
    }
}
```

> Presenter层

```
/**==========================================
 * 契约接口：
 *    存放相同业务的Preenter和View-便于查找和维护
 * //NetInfoContract.java
 *==========================================*/
public interface NetInfoContract{
    //1、Presenter定义了获取数据的方法
    interface Presenter{
        void getNetInfo(String ip);
    }
    //2、View中定义了与界面交互的方法
    interface View extends BaseView<Presenter> {
        void setNetInfo(NetInfo netInfo);
        void showError(String msg);
    }
}
/**==========================================
 * Presenter具体实现：NetInfoPresenter.info
 *   1. 分别与View层和Model层Task关联起来(持有了两者的对象)
 *   2. 实现接口getNetInfo()用于View层从Model层获取数据
 *   3. * 次要实现了Task执行中需要的回调接口-代理完成了View与Model的交互(避免了M/V的直接交互)
 *==========================================*/
public class NetInfoPresenter implements NetInfoContract.Presenter, LoadTasksCallBack<NetInfo>{
    //1. View层
    private NetInfoContract.View mView;
    //2. Model层任务
    private NetTask mNetTask;
    //3. 分别与View和Model建立关联
    public NetInfoPresenter(NetInfoContract.View view, NetTask netTask){
        mNetTask = netTask;
        mView = view;
    }
    //4. 能让View层获取到Model层数据
    @Override
    public void getNetInfo(String ip) {
        mNetTask.execute(ip, this);
    }
    //5. 实现Model层需要的回调接口-作用是将Model层数据交给View层
    @Override
    public void onSuccess(NetInfo netInfo) {
        mView.setNetInfo(netInfo);
    }
    @Override
    public void onFailed() {
        mView.showError("error");
    }
}
```

> View层

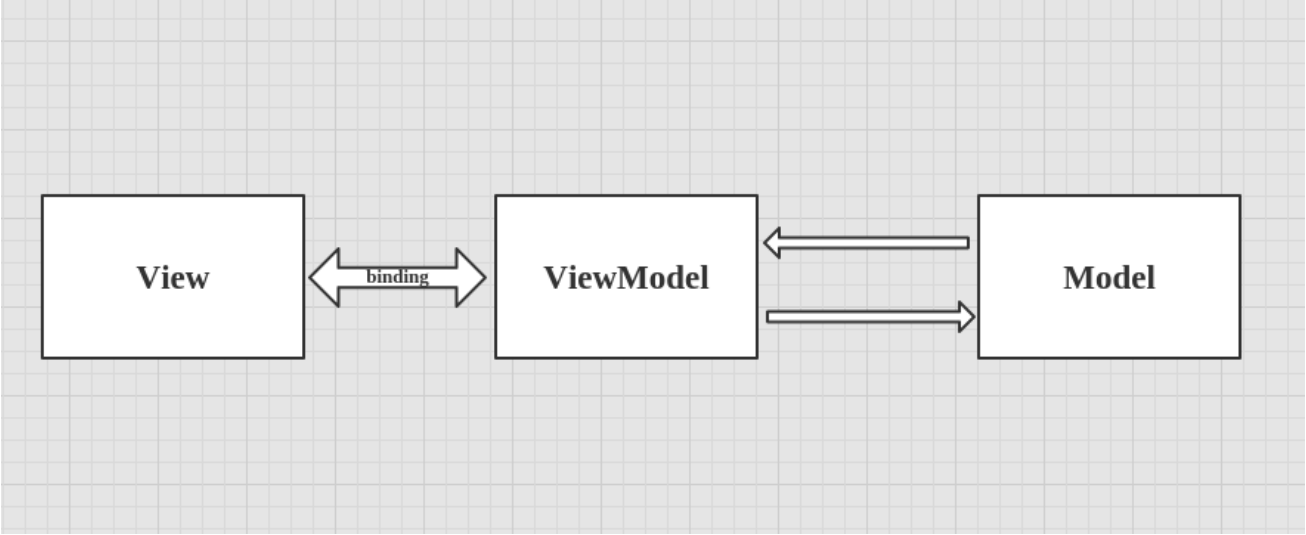
```
//BaseView.java
//View层的基类：定义了设置Presenter的接口
public interface BaseView<T> {
    void setPresenter(T presenter);
}
//HttpActivity.java
//  View层的具体实现：可以是Activity也可以是Fragment
public class HttpActivity extends Activity implements NetInfoContract.View{
    //1. 中间代理人
    private NetInfoContract.Presenter mPresenter;
    Button mGetButton, mSetButton;
    TextView mTitleTxtView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_activity_http);
        mGetButton = findViewById(R.id.get_fromnet_button);
        mSetButton = findViewById(R.id.set_button);
        mTitleTxtView = findViewById(R.id.title_textview);

        /**=============================================
         *  1、给View层设置Presenter和Model层的Task(获取数据)
         *=============================================*/
        setPresenter(new NetInfoPresenter(this, new NetInfoTask()));

        /**=============================================
         *  2、View层通过Presenter去获取数据
         *=============================================*/
        mGetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //TODO 从网络请求到数据
                mPresenter.getNetInfo("192.168.1.1");
            }
        });
        mSetButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mTitleTxtView.setText("Local Msg = Hello");
            }
        });
    }

    /**====================================
     * 3、实现View层的三个接口：设置Presenter和View界面相关功能
     *====================================*/
    @Override
    public void setPresenter(NetInfoContract.Presenter presenter) {
        mPresenter = presenter;
    }
    @Override
    public void setNetInfo(NetInfo netInfo) {
        mTitleTxtView.setText(netInfo.getMsg());
```

```
    }
    @Override
    public void showError(String msg) {
        mTitleTxtView.setText(msg);
    }
}
```

MVVM

10、MVVM模式的作用和特点?

> 1. Model-View-ViewModel ， 将 Presenter 替换为 ViewModel 。
> 2. ViewModel 和 Model/View 进行了双向绑定。
> 3. View 发生改变时， ViewModel 会通知 Model 进行更新数据
> 4. Model 数据更新后， ViewModel 会通知 View 更新
> 5. Google 发布了 MVVM 支持库 Data Binding



模块化和组件化

11、什么是模块化

> Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.
>
> 1. 一种 软件设计技术
> 2. 将 项目 的功能拆分为 独立 、 可交换 的模块
> 3. 每个 模块 都包含执行 单独功能 的 必要内容 。

12、什么是组件化

> 1. 组件化软件工程也被成为组件化开发，是一种软件工程的分支。
> 2. 强调将一个软件系统拆分为独立的组件(组件可以使模块也可以是web资源等等)

13、模块化和组件化的区别

> 1. 两者目的都是 重用和解耦
> 2. 主要是 叫法不同
> 3. 模块化 侧重于重用， 组件化 更侧重于 业务解耦

14、组件化优点

> 1. 组件间可以灵活组建
> 2. 一个 组件 的更改，只要 对外提供的接口 没有变化，则 其他组件 不需要再测试。
> 3. 缺点：对技术、业务理解度有更高要求。

15、模块化的层次拆分

> 1. 基础库
> 2. 通用业务层
> 3. 应用层

16、模块间通信

> 1. 可以自己实现但比较麻烦
> 2. 建议用 阿里巴巴 的开源库。

参考和学习资料

1. 认清Android框架 MVC，MVP和MVVM
2. Data Binding