Android面试题，架构MVC/MVP/MVVM，包括MVC、MVP、MVVM的介绍和优缺点，以及组件化、模块化的概念问题。

> 本文是我一点点归纳总结的干货，但是难免有疏忽和遗漏，希望不吝赐教。
> 转载请注明链接：https://blog.csdn.net/feather_wch/article/details/81282033

有帮助的话请点个赞！万分感谢！

# Android面试题-架构MVC/MVP/MVVM

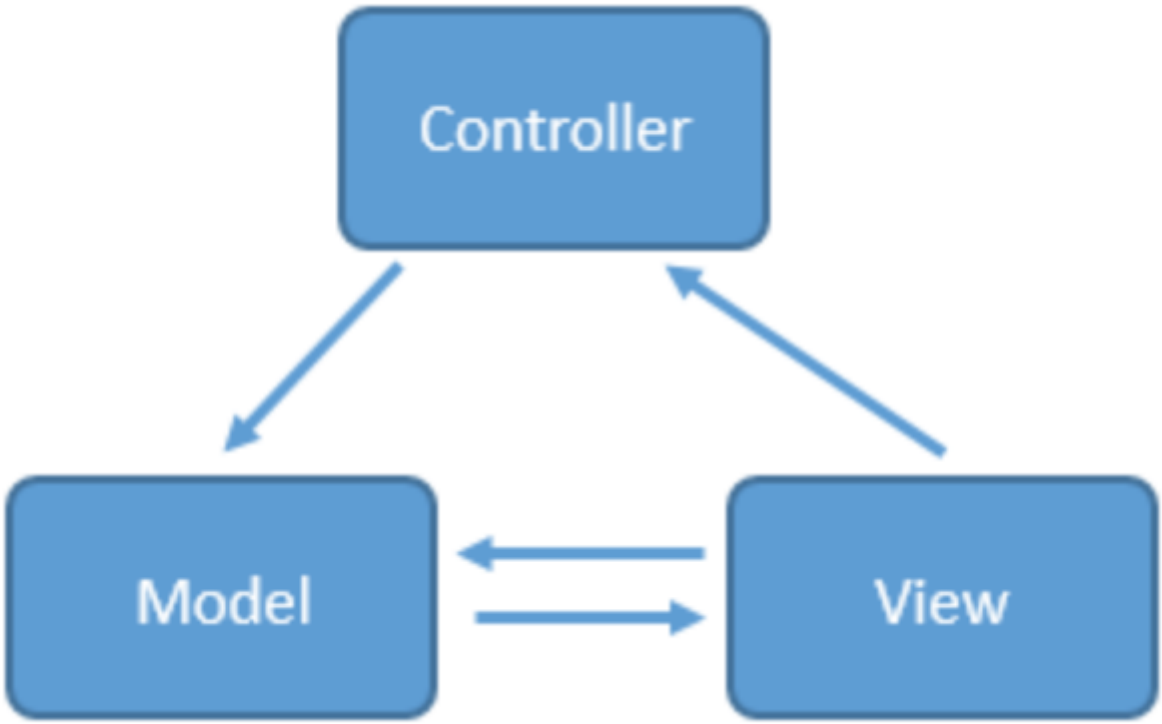
版本：2018/8/16-1（10:31）



## 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。 `Activity/Frgament` 也承担了View的功能。
> 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。



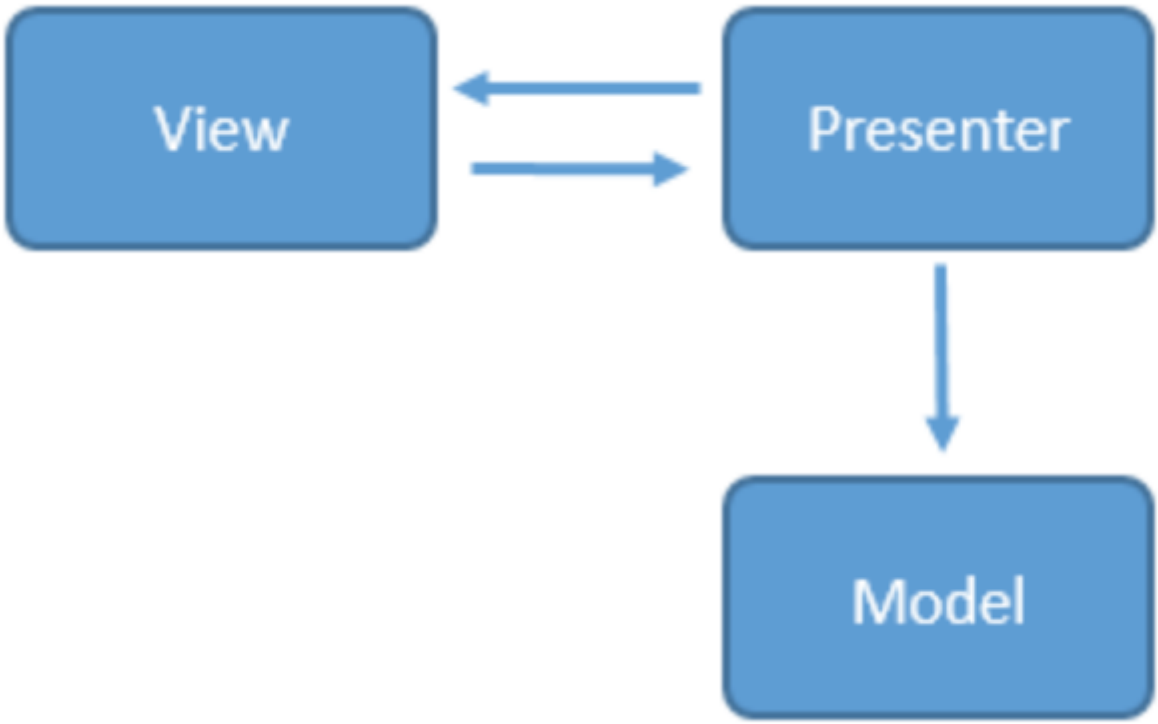

2、Android的MVC的缺点：

> 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， 耦合过重，代码量过大 ， 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?(2)

> 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 解耦，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。
> 5. 低耦合 ： Model和View的解耦，决定了该特性。

6、MVP的优点?

> 1. 低耦合：Model、View层的变换不会影响到对方。
> 2. 可重用性：Model层可以用于多个View。比如请求影视数据，可能有多个页面都需要这个功能，但是Model层代码只要有一份就可以了。
> 3. 方便测试 ： 可以单独对 Model 层和 View 层进行测试。

7、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。

## 8、MVP的实现?

> 请参考：https://blog.csdn.net/feather_wch/article/details/79729132
> 1、管理文件-LoginMVPContract.java

```
public interface LoginMVPContract{
    //View接口
    public interface ILoginView<T>{
        public void showLoginSuccess(T data);
        public void showLoginFailed(String errorMsg);
    }
    //任务接口
    public interface ILoginTask{
        public void startLogin(String phoneNumber, ILoginCallBack callback);
    }
    //Presenter
    public interface ILoginPresenter{
        public void startLogin(String phoneNumber);
    }
    //Presenter和Task间交互的接口
    public interface ILoginCallBack<T>{
        public void onLoginSuccess(T data);
        public void onLoginFailed(String errorMsg);
    }
}
```

> 2、Model的LoginResultBean和LoginTask.java

```
public class LoginResultBean {
}

public class LoginTask implements LoginMVPContract.ILoginTask{
    @Override
    public void startLogin(String phoneNumber, LoginMVPContract.ILoginCallBack callback) {
        if(true){
            callback.onLoginSuccess(new LoginResultBean());
        }else{
            callback.onLoginFailed("登录失败");
        }
    }
}
```

> 3、Presenter

```
public class LoginPresenter implements LoginMVPContract.ILoginPresenter, LoginMVPContract.ILogi

    LoginMVPContract.ILoginView mLoginView;
    LoginMVPContract.ILoginTask mTask;

    public LoginPresenter(LoginMVPContract.ILoginView loginView, LoginMVPContract.ILoginTask ta
        mLoginView = loginView;
        mTask = task;
    }

    /**
     * 接口回调至
     */
    @Override
    public void onLoginSuccess(Object data) {
        mLoginView.showLoginSuccess(data);
    }

    @Override
    public void onLoginFailed(String errorMsg) {
        mLoginView.showLoginFailed(errorMsg);
    }

    @Override
    public void startLogin(String phoneNumber) {
        mTask.startLogin(phoneNumber, this);
    }
}
```

> 4、View

```
public class LoginFragment extends SupportFragment implements LoginMVPContract.ILoginView<Logir

    LoginMVPContract.ILoginPresenter mLoginPresenter;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mLoginPresenter = new LoginPresenter(this, new LoginTask());
        mLoginPresenter.startLogin("17777777777");
    }

    @Override
    public void showLoginSuccess(LoginResultBean data) {
        //登陆成功
    }

    @Override
    public void showLoginFailed(String errorMsg) {
        //登录失败
    }
}
```

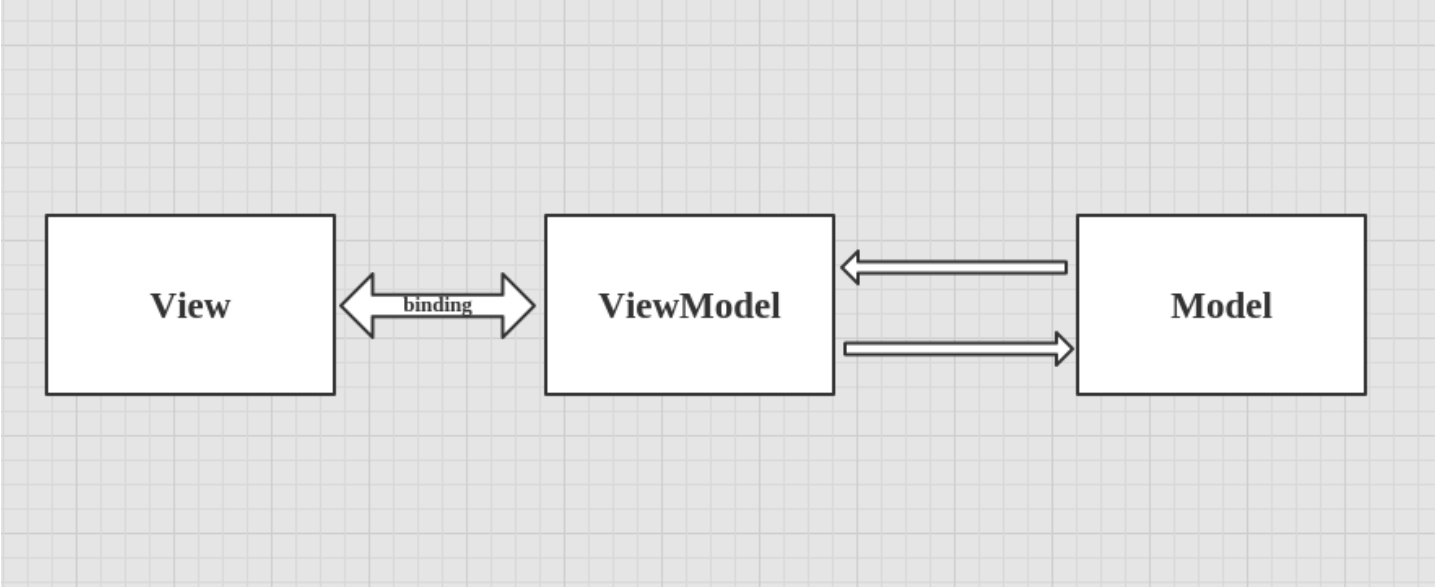
9、怎么优化MVP的类文件量

> 1. 采用 泛型 定义契约类，将 model、view、presenter 定义在一个 契约类中
> 2. 结构清晰，一个 契约类 对应一个 业务模块 。

# 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 更新显示
> 5. 谷歌发布了 MVVM 支持库 Data Binding ： 能将数据绑定到 xml 中
> 6. 现在谷歌又推出了 ViewModel和LiveData 组件用于更方便的实现 MVVM




# 模块化和组件化

11、什么是模块化

> 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
3. 组件化开源方案-详细总结
4. MVP\MVVM以及官方MVP样例工程