# 参考资料

1】 主：<https://juejin.cn/post/7022624191723601928> Mvc，mvp，mvvm，mvi

2】 <https://zhuanlan.zhihu.com/p/359674915>

3】 <https://juejin.cn/post/6901200799242649607#heading-0>

4】

<https://www.bilibili.com/video/BV1ZG4y1s7t2/?spm_id_from=333.337.search-card.all.click>

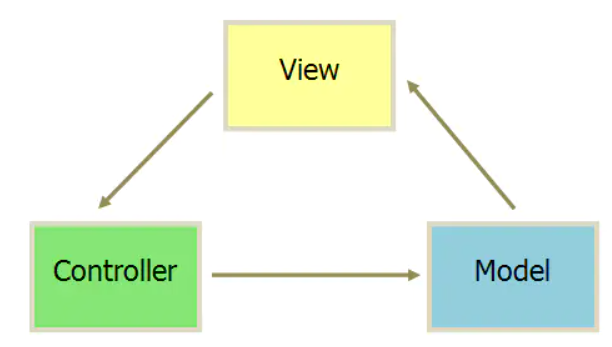
# MVC

补充参考：

1. <https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/9241230>
2. <https://blog.csdn.net/qq_48508278/article/details/122648284>
3. <https://www.runoob.com/design-pattern/mvc-pattern.html>
4. <https://www.jianshu.com/p/4b754ea48a40>

## （1） 概述

### 【1】 传统MVC



#### 3个组成：

MVC架构主要分为以下几部分：

 视图层（View）：展示模型层包含的数据。是用户界面，与用户直接进行交

互。

 控制器层（Controller）：主要负责业务逻辑

 模型层（Model）：主要负责网络请求、数据库处理、I/O的操作，即页面

的数据来源

#### 三者间的关系：

<https://www.jianshu.com/p/7398ec36d591>

View 持有了Controller，把事件传递给Controller，Controller由此去触发Model层的事件，Model更新完数据（网络或者本地数据）之后触发View的更新事件。

### 【2】 Android中的MVC

#### 前言

Google官方没有给出过MVC的架构使用建议，但AS的工程目录

体现的就是MVC。

#### 《2》 3层对应的android组成

 视图层（View）：对应于xml布局文件和java代码动态view部分

 控制层（Controller）：在android中由Activity承担，同时因为XML视图功能太弱，所以Activity既要负责视图的显示又要加入控制逻辑，承担的功能过多。

 模型层（Model）：同传统MVC

由于android中xml布局的功能性太弱，Activity实际上负责了View层与Controller层两者的工作，所以在android中mvc更像是这种形式：



#### 《3》 缺陷

因此MVC架构在android平台上主要存在以下问题：

1. Activity同时负责View与Controller层的工作，违背了单一职责原则
2. Model层与View层存在耦合，存在互相依赖，违背了最小知识原则/迪米特法则

### 【3】 适用场景



## （x） 实例

### 例1：读研时做的课程项目

完整项目：<https://yummyorder.github.io/dashboard/>

自己负责的android代码（基于MVC）：

<https://github.com/ericliyanghui/YummyOrder-FE>

### 例2：读研时上的课给的资料

见图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

# MVP

补充参考：

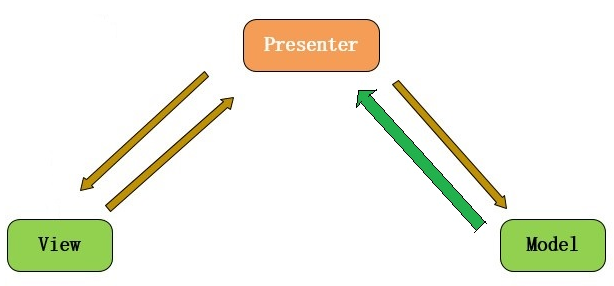
1. <https://www.jianshu.com/p/4b754ea48a40>
2. <https://www.jianshu.com/p/9a6845b26856>
3. <https://zhuanlan.zhihu.com/p/336870585>
4. <https://juejin.cn/post/6844903720036073480#heading-5>

## 概述

### 为什么要MVP【MVC的缺陷】

MVP通过P层解决了MVC中Activity角色过多、过于臃肿的问题。

### MVP的3个组成（结合Android）



#### 3个组成：

**View层（视图层）**：对应于Activity与XML,只负责显示UI,只与Presenter层交互，与Model层没有耦合

**Presenter层（主持人层）**：主要负责处理业务逻辑，通过接口回调View层

**Model层（模型层）**：主要负责网络请求、数据库处理等操作，这个没有什么变化

#### 三者间的关系：

View让Presenter请求Model中的数据，Model得到数据后返回给Presenter，然后Presenter回调View的接口刷新UI。

### Android中的MVP

<https://www.jianshu.com/p/4b754ea48a40>

Google官方是给出过MVP在Android中的用法示例的。

### 缺陷

1. V层给P层回调的接口过多，即接口地狱问题

### 适用场景

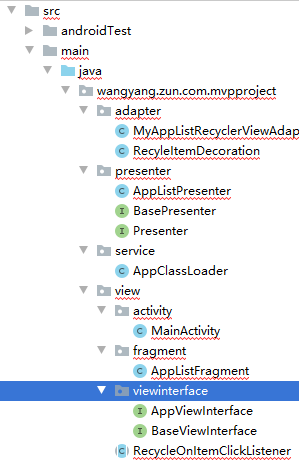


## 示例

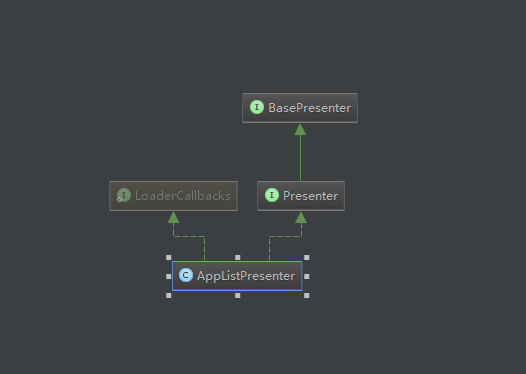
<https://www.jianshu.com/p/4b754ea48a40>

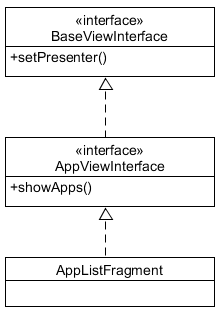
### 代码框架

下来我们看一个使用的用例吧，这个demo相对来说非常简单，下面是项目的架构，一个Activity，一个Fragment，Data层主要负责获取App已安装的应用列表，AppListPresenter负责业务逻辑处理



我们先看下presenter，viewinterface的结构





### 详解

看下AppListFragment的代码：

|  |
| --- |
| public class AppListFragment extends Fragment implements AppViewInterface {  private Presenter presenter;  private List<PackageInfo> packageInfoList = new ArrayList<>();  private RecyclerView recyclerView;  private MyAppListRecyclerViewAdapter myAppListRecyclerViewAdapter;  @Override  public void showAppList(List<PackageInfo> packageInfos) {  if (packageInfos.isEmpty())  return;  packageInfoList.clear();  packageInfoList.addAll(packageInfos);  myAppListRecyclerViewAdapter.notifyDataSetChanged();  }  @Override  public void setPresenter(Presenter presenter) {  this.presenter = presenter;  }  } |

代码比较容易理解，AppListFragment实现了AppViewInterface接口，我们需要在Activity中把AppListPresenter和AppViewInterface双向绑定。

接下来看下AppListPresenter层的代码，这里只列出了几个关键方法：

|  |
| --- |
| public class AppListPresenter implements Presenter, LoaderManager.LoaderCallbacks<List<PackageInfo>>{  private AppViewInterface viewInterface;  private AppClassLoader appClassLoader;  private LoaderManager loaderManager;  private final int id = 0;  public AppListPresenter(AppViewInterface viewInterface, AppClassLoader appClassLoader,  LoaderManager loaderManager) {  this.viewInterface = viewInterface;  this.appClassLoader = appClassLoader;  this.loaderManager = loaderManager;  viewInterface.setPresenter(this);  }  @Override  public void loadInstallApps() {  //通过loadmanager提供的方法获取安装的应用列表  loaderManager.initLoader(id, null, this);  }  @Override  public void destory() {  loaderManager.destroyLoader(id);  }  @Override  public void onLoadFinished(Loader<List<PackageInfo>> loader, List<PackageInfo> data) {  //获取到已安装的应用列表，调用AppViewInterface的showAppList方法  viewInterface.showAppList(data);  }  @Override  public void launchApp(PackageInfo packageInfo) {  Intent intent = appClassLoader.queryLaunchIntent(packageInfo);  if (intent != null)  appClassLoader.getContext().startActivity(intent);  else  Toast.makeText(appClassLoader.getContext(), "Can not start the app", Toast.LENGTH\_SHORT).show();  }  } |

关键方法是loadInstallApps，这个方法在MainActivity的onCreate中调用

|  |
| --- |
| private Presenter appListPresenter;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  setSupportActionBar(toolbar);  FragmentTransaction fragmentTransaction = getSupportFragmentManager().beginTransaction();  AppListFragment appListFragment = AppListFragment.newInstance();  fragmentTransaction.add(R.id.fm, appListFragment);  fragmentTransaction.commit();  appListPresenter = new AppListPresenter(appListFragment, new AppClassLoader(getApplicationContext()),  getSupportLoaderManager());  //调用loadInstallApps  appListPresenter.loadInstallApps();  } |

首先，我们获取一个AppListFragment的实例，在AppListPresenter构造函数里面我们传入AppViewInterface，同时在AppListPresenter的构造函数中又将presenter注入到了AppViewInerface里面，这样就实现了Presenter和ViewInerface双向绑定，之后调用AppListPresenter的loadInstallApps方法，在onLoadFinished回调里面又调用了AppViewInterface的showApps方法，这样数据就显示在界面。整个Activity和Fragment的代码精简了很多。

## 实例

### 例1：oppo智能家居客户端

该app就是采用MVP搭建的。

# MVVM

补充参考：

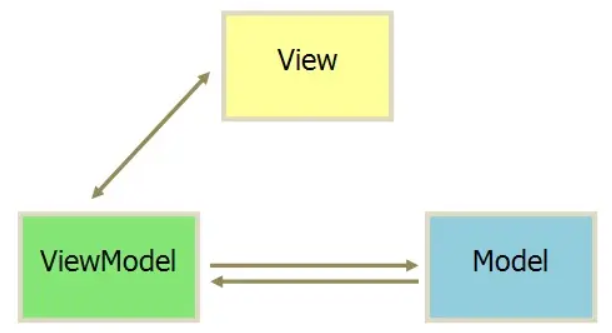
1. <https://www.cnblogs.com/loaderman/p/10076529.html>
2. <https://juejin.cn/post/7058542176375930887#heading-1>

## 概述

### 为什么要MVVM【MVP的缺陷】

1. 解决MVP接口地狱问题

### MVVM的3个组成



#### 3个组成：

**View层（视图层）**：定义了界面中的布局和外观

**ViewModel层（视图的模型层）**：逻辑控制层，负责处理数据和处理View层中的业务逻辑

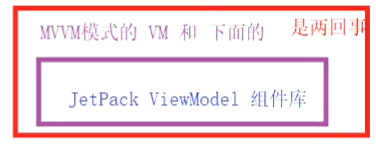
**Model层（模型层）**：用于获取业务数据模型

#### 三者间的关系：

MVVM 模式将 Presenter 改名为 ViewModel，基本上与 MVP 模式完全一致。  
唯一的区别是，它采用数据绑定（data-binding）：ViewModel的变动，自动反映在View；若View的变动也能反映在ViewModel就是双向绑定，否则就是单向绑定。

可以看出MVVM与MVP的主要区别在于,你不用去主动刷新UI了，只要Model数据变了，会自动反映到UI上。换句话说，MVVM更像是自动化的MVP。

#### 注意点：



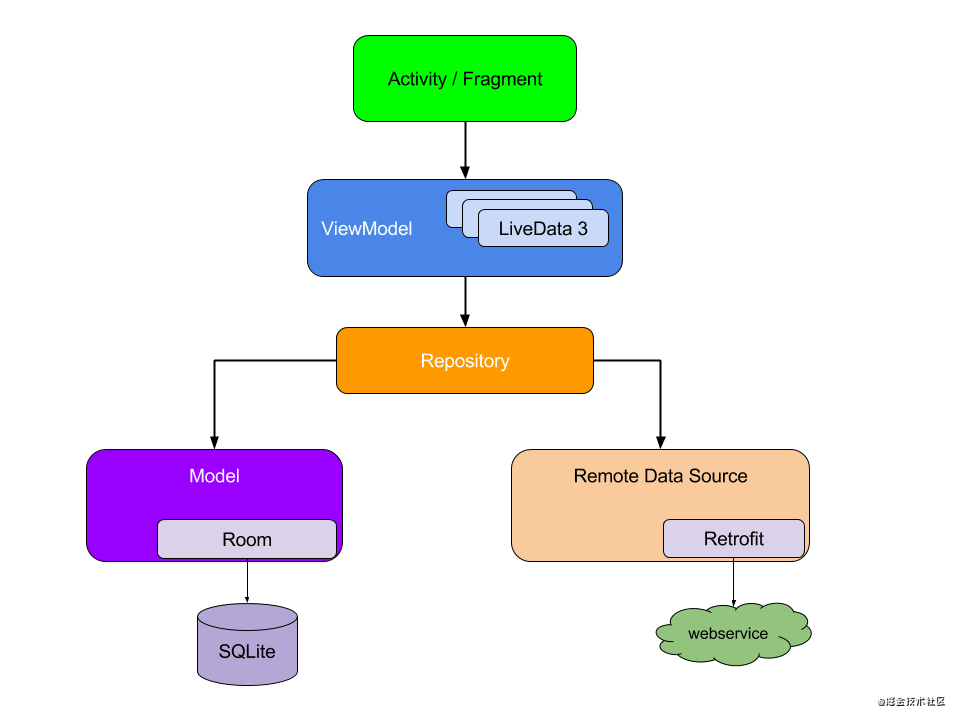
### 常见用法

#### Android官方推荐的MVVM用法【使用Jetpack】

<https://www.cnblogs.com/loaderman/p/10076529.html>

<https://juejin.cn/post/7058542176375930887>

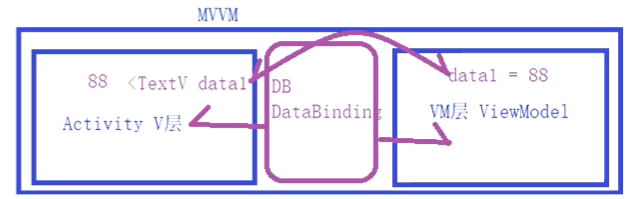
<https://zhuanlan.zhihu.com/p/359674915>



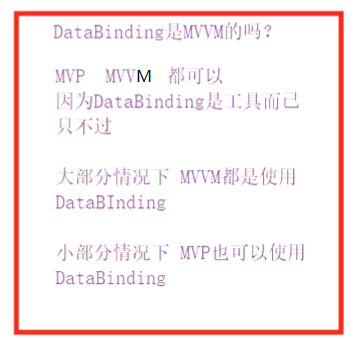
注：该图是官网以前的，现已找不到了

* Model层：对应图中的 Repository（网络请求数据、本地数据库数据等）。
* View层：对应图中的 Activity/Fragment。
* ViewModel层：对应图中的 ViewModel 和 LiveData。

#### 用DataBinding双向绑定



##### 注意点：



#### 不用DataBinding双向绑定

<https://juejin.cn/post/7022624191723601928#heading-4>

### 【4】 适用场景

图片包含 图示

描述已自动生成

## 示例

# MVI