如今**Android**最为流行的网络请求框架莫过于**Retrofit**+**Rxjava**，网上也是诞生了大量**Mvp**+**Retrofit**+**Rxjava**+**Dagger**+**GreenDao**的相关技术文章。

**Mvp**、**Mvc**、**Mvvm 的区别：**

在说**Mvp**之前，相信大家应该都熟悉了**Mvc**（model 、 view 、 control）。

**Mvc**相对于**Mvp**和**Mvvm**较为落后、耦合度太高、职责不明确。

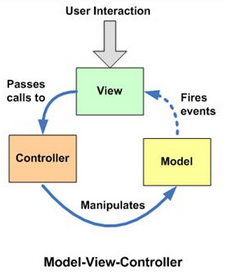
正因为**Mvc**有这样那样的缺点，所以才演化出了**Mvp**和**Mvvm**这两种框架。

**Mvvm**使用了**DataBind**的**Mvp**，普及性不如**Mvp**+**ButterKnife**，**ButterKnife**更简单的绑定控件。

**Mvvm**的架构我个人觉得是**Android**往后发展的趋势，毕竟谷歌都推出了**Datebinding**,而使用**Datebinding**也就可以不用去使用**Bufferknife**了，而且绑定的这种机制也确实带来了model,view与vm的分离，从逻辑上看也确实清晰了很多，支持双向绑定，这个谷歌也更新了，这个比较适合大型项目，暂时不适合小型的项目。**DataBind**还是比较影响项目进度，使得 Bug 很难被调试。

关于**Android**程序的构架, 最早期的**Mvc**模式，当前比较成熟且使用最多的应该就是**Mvp**架构了,当然还有微软提出来的**Mvvm**架构是以后的趋势。

**Mvc**



**Mvc**全名是：Model(模型) View(视图) Controller(控制器) 是软件[架构]中最常见的框架，简单来说，就是通过Controller的控制去操作Model层的数据，并且返回给View作展示。

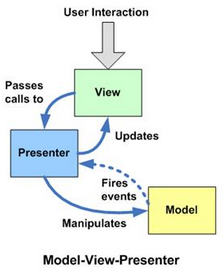
**Mvc优点：**把业务逻辑全部放到到Controller中，当业务逻辑变更的时候，不需要变更View和Model，只需要更换Controller就行了。类文件比较少，逻辑和view都放到一个类里面。

**Mvc缺点**：在Android的架构中Activity,fragment,布局的xml相当于View。

然而在实际的开发过程中，Android的View层任务太繁重，大家将V和C都糅杂在Activity、Fragment中，这就导致了在实际开发中View层太过累赘，几次代码迭代过后，一个Activity或者Fragment中的代码就有几千行，有时候修改个功能在笨重的代码中摸爬滚打半天才找到关键点，对于不熟悉代码的开发者来说（或者是前人编写的代码）那代码迭代接手后简直是妙不可言。所以将视图和业务逻辑代码分开势在必行。

大家看上面那幅图，view层和model层是相互可知的，这意味着两层之间存在耦合，耦合对于一个大型程序来说是非常致命的，因为这表示开发，测试，维护都需要花大量的精力。

**Mvc**使用场景：适用于较小，功能较少，业务逻辑较少的项目。

**Mvp**  


**Mvp**作为**Mvc**的演化，解决了**Mvc**不少的缺点，对于**Android**来说，**Mvp**的model层相对于**Mvc**是一样的，而activity和fragment不再是controller层，而是纯粹的view层，所有关于用户事件的转发全部交由presenter层处理。

最明显的差别就是view层和model层不再相互可知，完全的解耦，取而代之的presenter层充当了桥梁的作用，用于操作view层发出的事件传递到presenter层中，presenter层去操作model层，并且将数据返回给view层，整个过程中view层和model层完全没有联系。

目前实际开发中火热的**Mvp**模式（model 、 view 、presenter）。用presenter将model和view隔离开来，一切业务逻辑处理都是通过presenter来进行操作，也就是说presenter是视图的数据的桥梁，视图和数据相隔两端，“可远观而不可亵玩”。强制只能使用presenter作为“邮递员”来”通信。“。

**优点：1）**最大的优点剥离了视图层和业务逻辑层，让各个类的分工更为明确，逻辑更为清晰，代码扩展性更高，后期代码迭代更新更加容易，同时也方便了单元测试的编写，以前Mvc将视图与业务混合在一起的时候，要写单元测试很多时候真的是无从下手。相对于Mvc模式来说写单元测试就更加容易了。因为职责更加清晰，测试的目标就明确多了。

**2）**View高度复用

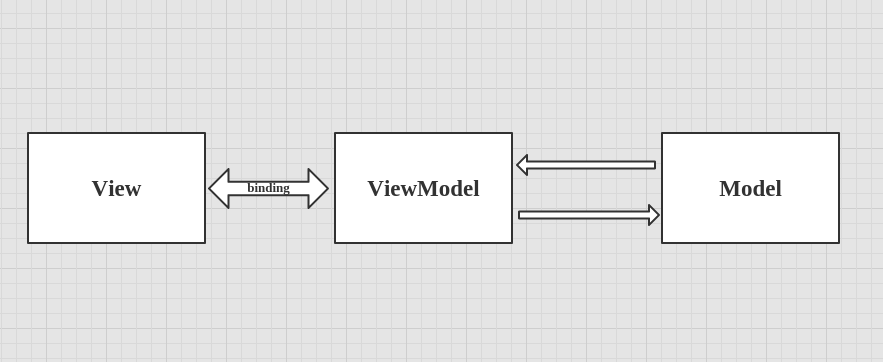
**3）**View可以进行组件化。在Mvp当中，View不依赖Model。这样就可以让View从特定的业务场景中脱离出来，可以说View可以做到对业务逻辑完全无知。它只需要提供一系列接口提供给上层操作。

**缺点：1）** Mvp模式的类文件也多了，简单的一个业务逻辑操作就要各方来配合协作（即是需要presenter 和 view的接口）。但是这个问题完全在可以接收的范围。完全符合Java的抽象封装设计原则（接口隔离，开闭原则）

**使用场景：**

**1）**视图界面不是很多的中小型项目中。

MVVM



从图中看出，它和MVP的区别貌似不大，只不过是presenter层换成了viewmodel层，还有一点就是view层和viewmodel层是相互绑定的关系，这意味着当你更新viewmodel层的数据的时候，view层会相应的变动ui。所以在我看来说它是mvp的升级版也不为过。

**优点：**

**1）**提高可维护性，解决了**Mvp**大量的手动View和Model同步的问题，提供双向绑定机制。提高了代码的可维护性。

**2）**简化测试，因为同步逻辑是交由Binder做的，View跟着Model同时变更，所以只需要保证Model的正确性，View就正确。大大减少了对View同步更新的测试。

**3）**ViewModle易于单元测试。

**缺点：**

**1）** 数据绑定使得 Bug 很难被调试。你看到界面异常了，有可能是你 View 的代码有 Bug，也可能是 Model 的代码有问题。数据绑定使得一个位置的 Bug 被快速传递到别的位置，要定位原始出问题的地方就变得不那么容易了。

**2）**一个大的模块中，model也会很大，虽然使用方便了也很容易保证了数据的一致性，当时长期持有，不释放内存，就造成了花费更多的内存。

**3）**数据双向绑定不利于代码重用。客户端开发最常用的重用是View，但是数据双向绑定技术，让你在一个View都绑定了一个model，不同模块的model都不同。那就不能简单重用View了。

**使用场景：**

**1）**适用于界面展示的数据较多的大型项目。

**网络请求为什么要用Retrofit？**

Retrofit与okhttp共同出自于Square公司，retrofit就是对okhttp做了一层封装。

OkHttp现在已经得到Google官方认可，大量的app都采用OkHttp做网络请求。

然后Retrofit是Square公司开发的一款针对Android网络请求的框架。

Retrofit其实我们可以理解为OkHttp的加强版，它也是一个网络加载框架。底层是使用OKHttp封装的。准确来说,网络请求的工作本质上是OkHttp完成，而 Retrofit 仅负责网络请求接口的封装。

## Retrofit的好处？

超级解耦

1）可以配置不同HttpClient来实现网络请求，如OkHttp、HttpClient...

2）支持同步、异步和RxJava

3）可以配置不同的反序列化工具来解析数据，如json、xml...

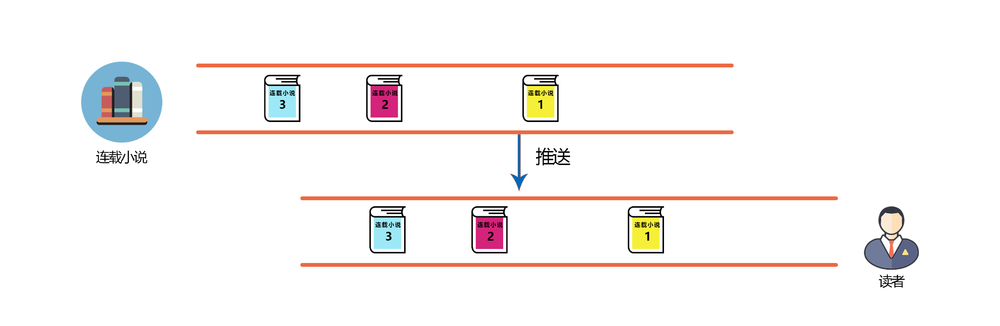
4）请求速度快，使用非常方便灵活

**什么是Rxjava？**

网上有很多关于Rxjava从放弃到入门到文章。

认识Rxjava之前必须要弄明白什么是观察者模式。

简单介绍一下，A和B两个，A是被观察者，B是观察者，B对A进行观察，B并不是需要时刻盯着A，而是A如果发生了变化，会主动通知B，B会对应做一些变化。举个例子，假设A是连载小说，B是读者，读者订阅了连载小说，当小说出现了新的连载的时候，会推送给读者。读者不用时刻盯着小说连载，而小说有了新的连载会主动推送给读者。这就是观察者模式。而RxJava正是基于观察者模式开发的。



## 应用场景

1）与Retrofit联用  
Retrofit+RxJava的上网模式已经非常火了

2）Rx全家桶的使用  
基于RxJava的开源类库Rxpermissions、RxBinding以及RxBus在很多项目中已经非常常见，并且被证明了是极其好用的。

举例子，Evenbus在Android 5.0 以后 已经取代了Android Handle 机制的地位。

而Rxbus 取代了Evenbus的地位。

Rxbus还能配合RxAndroid 使用。

Rxpermissions可以更轻易的获取Android 权限。

RxBinding 可以双向绑定控件。

3）所有用到异步的地方

因为RxJava就是一个支持异步的链式编程，所以所有的用到异步的地方，我们都可以用RxJava来完成

**什么是Dagger2？**

Dagger是为Android和Java平台提供的一个完全静态的，在编译时进行依赖注入的框架，原来是由Square公司维护，现在由Google维护。

我们知道Dagger是一个依赖注入的框架，那么什么是依赖注入呢？

我们在activity中有可能会用到很多很多的类，这些类要在activity中进行实例化，这样就导致我们的activity非常依赖这么多的类，这样的程序耦合非常

严重，不便于维护和扩展，有什么办法可以不去依赖这些类呢，这时候就需要有一个容器（IoC），将这些类放到这个容器里并实例化，我们activity在用

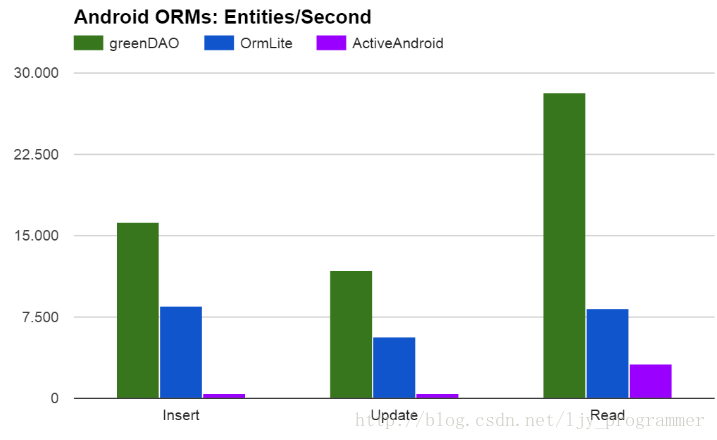
到的时候去容器里面取就可以了，我们从依赖类到依赖这个容器，实现了解耦，这就是我所理解的依赖注入，即所谓控制反转；

简单的说 Dagger就是用来创造这个容器，所有需要被依赖的对象在Dagger的容器中实例化，并通过Dagger注入到合适的地方，实现解耦，MVP框架就是为解耦而生，因此MVP和Dagger是绝配。

**为什么Android 数据库使用GreenDao？**

Android开发中我们或多或少都会接触到数据库。Android中提供了一个占用内存极小的关系型数据库-SQLite。虽然Android系统中提供了许多操作SQLite的API，但是我们还是需要手动去编写SQL语句，这经常会出现一些莫名其妙的问题。所以便出现了许多ORM（对象关系映射）框架。其中比较著名的有GreenDao、OrmLite、ActiveAndroid等。下面主要介绍一下GreenDao，我平时用的最多的也是GreenDao。

**附上一张官方提供的，GreenDAO、OrmLite、ActiveAndroid的对比图**



GreenDao是一个高效的数据库访问ORM框架，节省了自己编写SQL的时间，快速的增删查改等操作。

**1）性能最大化，是Android 平台上最快的ORM框架之一**

**2）易于使用APi**

**3）最少的内存开销**

**4）依赖体积小**

**5）支持数据库加密**

**6）代码自动生成**

**7）强大的社区支持**

效率很高，插入和更新的速度是sqlite的2倍，加载实体的速度是ormlite的4.5倍，目前git上一直在做更新维护，start数量超过1万。