Android架构之MVC、MVP、MVVM解析

一只修仙的猿 2020-12-13 12:34 4030

关注

前言

很高兴遇见你~

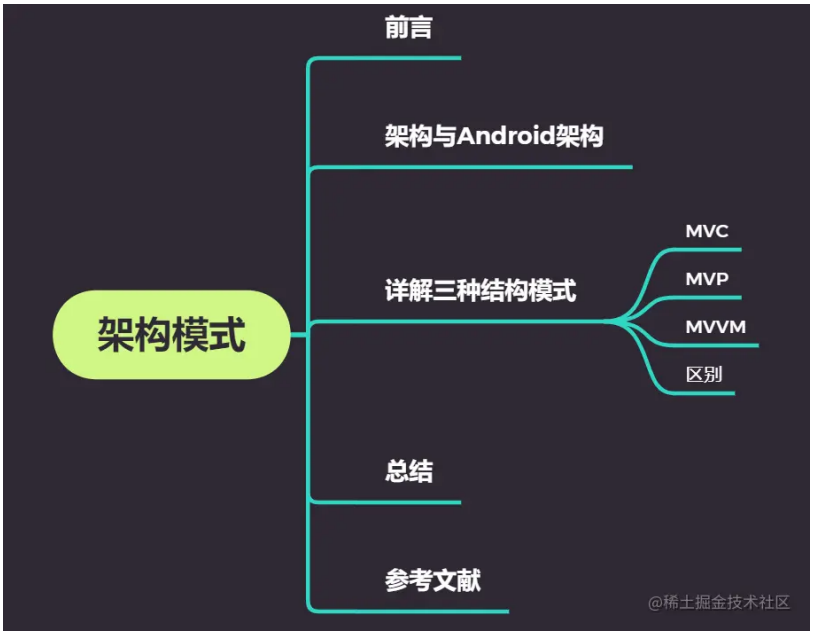
MVC、MVP、MVVM均为架构模式，应用在Android上，称为Android架构模式。可能你会觉得我在讲废话，清楚他的定义非常重要。这样会有几个问题：

> - 什么是架构模式？什么是android架构模式？
> - MVC、MVP、MVVM的本质区别是什么？
> - 他们在android上的应用是怎么样的？
> - 我们该如何选择？

弄清楚这几个问题，可以帮助我们更好地理解这三种架构模式，继而更好地运用它们。

这篇文章主要的内容是带你了解什么是架构以及android架构，以及详解三种架构模式的本质。

本文目录



架构以及Android架构

架构，即为骨架，是指导开发的关键。例如一个人，身体的骨骼即为身体的架构，有了基本骨架之后，才可以决定在头颅里开发大脑，在肋骨中开发肺部等。软件开发也是如此，也需要一个“骨架”，即架构。他可以指引我们什么地方该做什么事情，让整个软件的开发思路非常清晰。

Android架构，即为开发android时使用的架构。Android的开发一般分为三部分：UI逻辑，业务逻辑和数据操作逻辑。这里可以举个例子

> 获取天气详情并展示。首先要写布局xml，接着在Activity中获取到view实例。此为UI逻辑。然后需要通过网络请求获得数据，此为数据操作逻辑。接着获取到数据后，需要将数据进行缓存，解析，再set到view上进行展示，此为业务逻辑。

Android架构，就是为了更好地协调这三者的关系。达到：

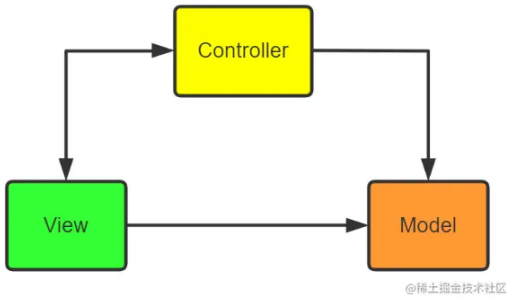
> 1. 各模块高内聚低耦合的状态，方便进行团队分工合作开发。
> 2. 代码思路清晰，提高代码的可维护性与可测试性。
> 3. 减少样板代码，提高开发效率，减少开发错误。

为了达到这些目的，所有才有各种架构不断涌现，却没有一个统一的开发框架。不同于移动端，web端开发有成熟的SpringMVC框架，可以快速规范地开发一个项目。而移动端缺乏框架的支撑，各路大神各显神通，不断涌现了不同的架构模式来适应不同的开发情景，如MVC，MVP等等。但由于没有历史的沉淀，各种架构模式的弊端也渐渐浮出水面。在这种情境下，谷歌退出了架构组件，用成熟的框架来减少样板代码，提高开发效率，有如SpringMVC的风范，这就是MVVM的框架实现。下面我们就详细展开讲这些架构模式。

详解三种架构模式

MVC

Android上的MVC架构我认为是来源于web开发的SpringMVC，MVC全名为Model-View-Controller，图解如下



- View：负责与用户交汇，显示界面。
- Controller：负责接收来自view的请求，处理业务逻辑。
- Model：负责数据逻辑，网络请求数据以及本地数据库操作数据等。

在MVC架构中，Controller是业务的主要承载者，几乎所有的业务逻辑都在Controller中进行编写。而View主要负责UI逻辑，而Model是数据逻辑，彼此分工。

> MVC的本质就是按照UI逻辑、业务逻辑、数据逻辑不同的职责分三大模块，彼此分工。

在Android中，view一般使用xml进行编写，但xml的能力不全面，需要Activity进行一些UI逻辑的编写，因而MVC中的 V 即为xml+Activity。Model数据层，在Android中负责网络请求和数据库操作，并向外暴露接口。Controller是争议比较多的写法：一种是直接把Activity当成Controller；一种是独立出Controller类，进行逻辑分离。比较符合MVC思想的笔者认为是后者。因为前者直接在Activity中进行书写业务逻辑就会和UI逻辑混合在一起了，达不到模块分工的效果。MVC架构的处理流程一般是：

- view接收用户的点击
- view请求controller进行处理或直接去model获取数据
- controller请求model获取数据，进行其他的业务操作
- 这一步可以有多种做法：
  - 利用callBack从controller进行回调
  - 把view实例给controller，让controller进行处理
  - 通知view去model获取数据

> 举个栗子。
>
> 现在有一个获取天气详情的功能。我需要在xml中写ui，在Activity中给控件设置监听事件和写方法给控件set数据，如textView.setText()。当点击界面按钮时，会调用首先会调用onClickListener，然后再调用Controller的方法让model获取数据，获取完成后通知view，view去Model获取数据更新自己。或者Activity直接给Controller一个callBack，

> controller就可以在业务处理完成后进行回调；或者直接把view给controller让controller直接去更新ui。

MVC可能是我们第一次开发安卓的时候就会使用的架构模式。直接在Activity中书写业务逻辑，只抽离出Model层。（笔者第一次开发时连Model层都没抽离，全部在Activity中写，一千多行的MainActivity）这样看起来很美好，但是有严重的问题。MVC的核心就是按照职责分离代码。但几乎所有的业务逻辑代码都在controller中，当项目越来越大时会导致controller极度臃肿，难以维护。view与model直接依赖，模块之间依赖不单一，view因直接通过model获取数据，不可避免的会耦合一些业务代码。

> - 几乎所有的业务逻辑代码都在controller中进行，会导致非常臃肿，降低项目的可测试性与可维护性。
> - view直接持有controller和model实例，不同职责的代码进行耦合，导致代码耦合性高，模块分工不清晰。

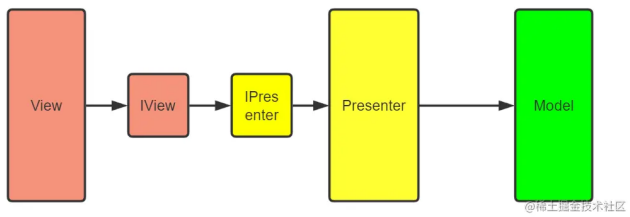
但MVC也有他的好处：简单。他不需要写很多的代码来让代码解耦，这在小型项目非常有用。小型项目总体的代码就不多，所以这样可以提高开发效率。但是最好不要尝试维护他，不是怕你崩溃，而是怕你砸了电脑对电脑不好。

因为可以发现MVC的改进方向就是：

> - 对模块进行更加彻底的分离，不要让view和model直接联系。
> - 对controller进行减压。

MVP

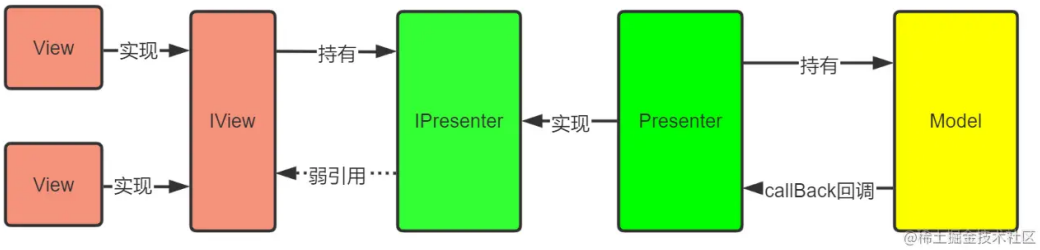
相比MVC，MVP的更加的完善。MVP全名是Model-View-Presenter。图解如下：



- View：UI模块，负责界面显示和与用户交汇。
- Presenter：负责业务逻辑，起着连接View和Model桥梁的作用。
- Model：专注于数据逻辑。

MVP和MVC的区别很明显就在这个Presenter中。为了解决MVC中代码的耦合严重性，把业务逻辑都抽离到了Presenter中。这样**View和Model完全被隔离，实现了单向依赖，大大减少了耦合度**。view和prensenter之间通过接口来通信，只要定义好接口，那么**团队可以合作同时开发不同的模块，同时不同的模块也可以进行独立测试**。也因各模块独立了，所以要只要符合接口规范，即可做到**动态更换模块而不需要修改其他的模块**。

在Android中，需要让Activity提供控件的更新接口，prensenter提供业务逻辑接口，Activity持有presenter的实例，prensenter持有Activity的弱引用（不用直接引用是为了避免内存泄露），Activity直接调用prensenter的方法更新界面，prensenter去model获取数据之后，通过view的接口更新view。如下图：



不同的view可以通过实现相同的接口来共享prensenter。presenter也可以通过实现接口来实现动态更换逻辑。Model是完全独立开发的，向外暴露的方法参数中含有callBack参数，可以直接调用callBack进行回调。

总结一下：

> - MVP通过模块职责分工，抽离业务逻辑，降低代码的耦合性
> - 实现模块间的单向依赖，代码思路清晰，提高可维护性
> - 模块间通过接口进行通信，降低了模块间的耦合度，可以实现不同模块独立开发或动态更换

但是，MVP真的就如此美好吗？当然不是。

MVP的最大特点就是接口通信，接口的作用是为了实现模块间的独立开发，模块代码复用以及模块的动态更换。但是我们会发现后两个特性，在Android开发中使用的机会非常少。presenter的作用就是接受view的请求，然后再model中获取数据后调用view的方法进行展示，但是每个界面都是不同的，很少可以共用模块的情景出现。这就导致了每个Activity/Fragment都必须写一个IView接口，然后还需要再写个IPresenter接口，从而产生了非常多的接口，需要编写大量的代码来进行解耦。如果在小型的项目，这样反而会大大降低了开发效率。 其次，prensenter并没有真正解耦，他还需要调用view的接口进行UI操作，解耦没有彻底。MVP也没有解决MVC中Controller代码臃肿的问题，甚至还把部分的UI操作带到了Presenter中。

因此，由于MVP有：

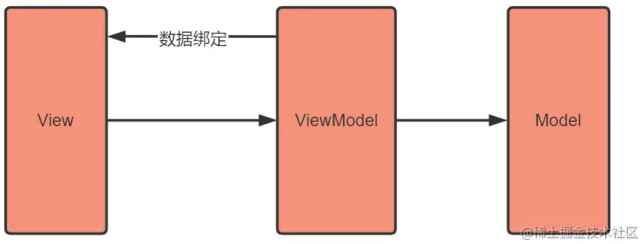
> - 过度设计导致接口过多，编写大量的代码来实现模块解耦，降低了开发效率
> - 并没有彻底进行解耦，prensenter需要同时处理UI逻辑和业务逻辑，presenter臃肿

这样的缺点，android开发者都在寻找一个更加完善的架构模式。当然读者知道下面我要讲MVVM了，但我想要说的是其实还有如AAC等结构模式的存在，但因他们的局限性以及上手难度，并没有被广泛使用。而到了MVVM，谷歌通过一系列的架构组件来让开发者可以简单地实现MVVM架构。

MVVM

终于到了MVVM，可能很多人都感觉“卧槽这么牛逼的架构我肯定学不会”然后被劝退了继续使用MVC或者MVP。在我看来，MVVM和上面两种架构模式一样都是一种架构思想，只是谷歌推出了jetpack架构组件来让我们更好的使用这种架构模式。

MVVM，全名为Model-View-ViewModel。图解：



- View：和前面的MVP、MVC中的View一样，负责UI界面的显示以及与用户的交汇。
- Model：同样是负责网络数据获取或者本地数据库数据获取。
- ViewModel：负责存储view的数据映像以及业务逻辑。

MVVM的view和model和前面的两种架构模式是差不多的，重点在ViewModel。viewModel通过将数据和view进行绑定，修改数据会直接反映到view上，通过**数据驱动型思想**，彻底把MVP中的Presenter的UI操作逻辑给去掉了。而viewModel是绑定于单独的view的，也就不需要进行编写接口了。但viewModel中依旧有很多的业务逻辑，但是因为把view和数据进行绑定，这样可以让view和业务彻底的解耦了。view可以专注于UI操作，而viewModel可以专注于业务操作。因而：

> - MVVM通过数据驱动型思想，彻底把业务和UI逻辑进行解耦，各模块分工职责明确。

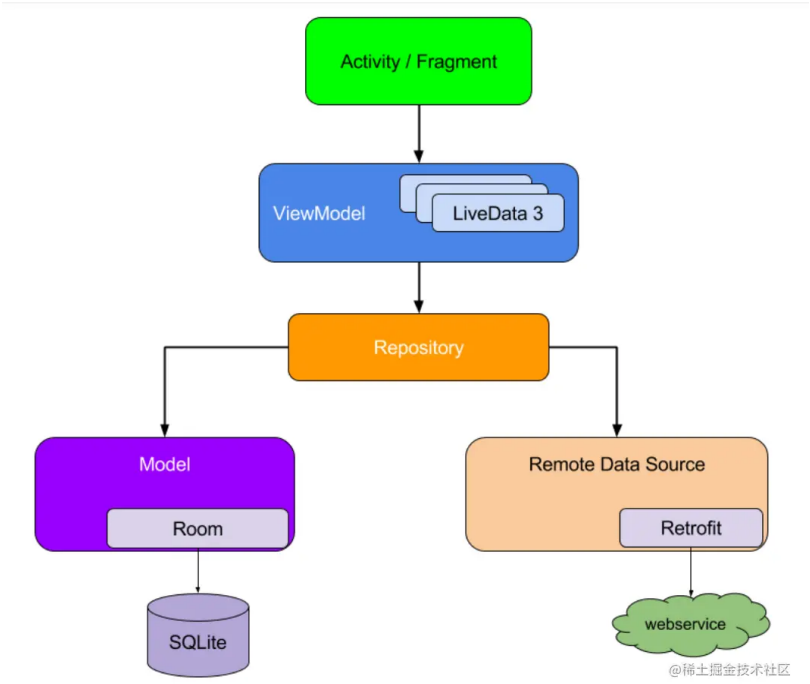
View只需要关注Viewmodel的数据部分，而无需知道数据是怎么来的；而ViewModel只需要关注数据逻辑，而不需要知道UI是如何实现的。View可以随意更换UI实现，但ViewModel却完全

不需要改变。

但依旧存在的问题是：viewModel会依旧很臃肿；需要一个绑定框架来对view和数据对象进行绑定。这是MVVM的两大弊端。上面的两种架构模式都是不需要框架的，但MVVM必须要有一个view-data绑定框架，来实现对data的更改可以实时反映到view上，这就造成了需要有一定的上手难度：学习框架。

> - MVVM的viewModel依旧很臃肿。
> - MVVM需要学习数据绑定框架，具有一定的上手难度。

为了解决上面两个问题，需要：1.简单易用的框架；2.为viewModel减少压力。所以谷歌推出了适合android开发的MVVM架构模式：



做个简单的解析：

- View对应的就是Activity和Fragment，在这里进行UI操作。
- ViewModel中包含了LiveData，这是一种可观察数据类型框架。View通过向LIveData注册观察者，当LiveData发生改变时，就会直接调用观察者的逻辑把数据更新到view上。
- ViewModel完全不需要关心UI操作，只需要专注于数据与业务操作。
- Repository代表了Model层，Repository对ViewModel进行了减压，把业务操作般到了Repository中，避免了viewModel臃肿。
- Repository对请求进行判断是要到本地数据库获取还是网络请求获取分别调用不同的模块。

这样，谷歌通过推出简单易用的架构框架，解决了我们上面讲的MVVM的两大问题，让MVVM架构达到了一种非常“完美”的境界。也是谷歌推荐的架构模式。

但为什么好像，MVVM使用的人还是那么少呢？因为jetpack的架构组件库，可不止是一个LiveData这么简单，他是一整套完整的架构组件库，包括了：DataBinding，LiveData，ViewModel，Navigation，Lifecycle。下面我们简单了解一下每个组件的功能：

> DataBinding：
>
> - 解基于数据驱动思想，决视图调用一致性问题，实现双向绑定
> - 避免编写样板式代码，提高效率
>
> LiveData：
>
> - 通过唯一可信源获取数据，正确分发数据
> - 与Lifecycle结合，拥有生命周期感知能力，配合viewModel实现作用域可控
> - 实现模块的单向依赖，抛弃接口回调
>
> ViewModel：
>
> - 托管界面状态，解决状态管理问题
> - 实现跨页面数据分享，并为数据设置作用域，做到作用域可控
> - 实现单向依赖，避免内存泄露
>
> Lifecycle：
>
> - 以简便地方式解决生命周期管理的一致性问题
> - 避免内存泄露的情况下让第三方组件随时获取生命周期状态，追踪事故所在的生命周期源，对错过时机的异步操作及时停止
>
> Navigation
>
> - 通过遵循导航定则实现对Fragment的管理
>
> 感觉对上面的作用没看懂，没关系，这不是本文的重点。通过列举这些组件的作用，我想表达的是：此时的MVVM已经不只是架构意义上的MVVM，而是框架层次上的。谷歌针对android开发的各种问题，定制各种各样的架构组件，结合MVVM思想，让我们的开发更加的简便和代码更加的健壮。

我们只需要遵循他的开发规范，使用他的架构框架，就可以开发出非常健壮的项目，有如Spring全家桶的风范。而众多的库，则足以劝退很多的开发者，而在公司的项目，对于框架的不熟悉与不可预期，不知道框架中会不会有什么坑，所以导致了公司项目使用的MVVM的人非常少，至少在我目前了解到的公司，均没有使用此架构。

> - 框架众多，学习成本高。
> - 框架使用少，对框架不熟悉，容易产生不可预期的异常且无法处理。

到这里我们会发现我们平常讲的MVVM并不只是简单的MVVM架构思想，更多的是指谷歌推出的一系列架构组件。MVVM的本质是什么?

> 一种基于数据驱动型，将UI逻辑和业务逻辑彻底分离的架构模式。

而我们平常说的“MVVM”是什么?

> 基于软件工程背景下，谷歌官方推出的一整套结合MVVM使用的架构组件。让开发者可以更简便，更高效率开发出非常健壮的代码。

关于如何实现谷歌意义上的MVVM架构模式，这涉及到框架组件的使用，这里不深入讲解。本文的目的是讲解架构思想，框架的使用，读者可前往官网自行学习。

区别

> 从应用范围来说，MVC和MVP属于广义上的架构模式，MVVM属于专用于页面开发的架构模式。

MVC是不同职责代码的解耦，MVP在MVC的基础上实现了模块间的进一步解耦。在生活中很多地方可以看到他们的影子：

> MVC举个洗衣机的例子。我们看到的操作面板就是View；里面的智能预定流程漂洗等属于Controller；洗衣机里的洗衣功能是model。这样我们通过view来请求controller设置洗衣流程，然后点击启动直接让model开始运作。这就可以看成一个MVC的例子。
>
> MVP举个充电线的例子。我们的手机可以看成是View；充电线是Presenter；插座是Model。手机向充电线暴露Type-c接口，而充电线向手机暴露对应的type-c插口。只要两者都符合这个接口规范即可，而不管具体是如何实现的。我们可以随意更换充电线，只要是type-c类型的就可以。我们也可以让一根充电线给不同的手机充电， 只要都是type-c接口就行。而插座和充电线之间的关系也是如此，但是插座会更加独立，因为全世界的插座都是这样，就像我们使用callBack一样。这些就体现了MVP的特点：不同职责模块分离，接口通信。

是吧，可以看到这两套思想是广义上的架构思想，但MVVM则不是。MVVM是专注于页面开发的。为什么这么说？MVVM的本质是把View和view要显示的数据对象进行分离绑定，通过数据驱动型思想彻底解耦UI和业务逻辑。前提要有view也就是页面，还有数据。所以他的应用场景

就被限制到页面开发中来了。我们不可能在洗衣机充电线上运用MVVM。因为没有页面和数据对象可言。所以：

> MVC是不同职责代码分离，MVP是在MVC的基础上通过接口通信降低模块间的耦合性，MVC，MVP都是广义上的架构模式，Android只是他们的一个应用场景。MVVM是专注于页面开发的架构模式，更加契合页面开发模式。

相信读者们都有一个问题：哪个架构模式是最好的？我应该选择哪个架构模式？是不是MVVM更加契合页面开发所以MVVM一定是最优解？

> 没有一定是最好的架构模式，只有特定情景下最适合的架构模式。

架构模式都是对于特定情景应运而生的，他是为了解决某个情景下的开发架构问题。例如android的问题是UI与业务逻辑的解耦，但充电线却是需要模块间的解耦。

那对于Android这个情景来说MVVM一定是最优解吗？当然不是。Android开发也是有很多的情景的，项目的大小，是否要进行维护，对时间有没有要求，项目紧急程度等等。首先要了解得是各大架构模式的优缺点。MVVM由于架构的加持，导致入手成本高，前期投入架构设计代码量多。如果只是一个微型项目，需要在一天内完成（别问我一天怎么完成）那么此时采用mvvm就有点不太适合了，可能搭完架构就已经过去半天了。这个时候因为项目代码很少，且要快速开发，使用MVC就是最合适的了。如果是中型项目，且涉及到团队开发，但是团队对于MVVM的框架并不熟悉，这个时候去学习MVVM再进行开发也是不太实际的，因为框架越多，可能出错的地方就越多，更何况是刚学的。这个时候模块间分离的MVP架构模式就似乎更加的适合了。所以，要根据不同的情景选择最适合的架构模式。

总结

这篇文章我们讲了是什么是架构和三种架构模式MVC、MVP、MVVM的架构思想，最后再讲了三种架构的区别与选择。相信通过这篇文章读者可能对这三种架构有了一定的认识。一个完美的架构模式是一个架构师一生的追求，在使用的时候我们可以多多思考这些架构模式的本质以及背景，想想架构师为什么这么设计，可以帮我们更好地理解架构。

> 全文到此为止。 有不同观点欢迎评论区或私信讨论。如需转载请私信或评论区交流。 另外欢迎阅读笔者的个人博客一只修仙的猿的个人博客，更精美的UI，拥有更好的阅读体验。

参考文献

- 《第一行代码》第三版
- 完全解析Android项目架构(1) - MVC
- 完全解析Android项目架构(2) - MVP
- 完全解析Android项目架构(3) - MVVM
- 是让人耳目一新的 Jetpack MVVM 精讲啊！
- 是让人提神醒脑的 MVP、MVVM 关系精讲！

标签： Android

文章被收录于专栏：

android

android系列文章

订阅专栏