从零开始的Android新项目（3）：谁告诉你MVP和MVVM是互斥的

2016-04-22 伯乐在线/翟一帆 安卓应用频道

> 来源：伯乐在线 - markzhai
> 链接：http://android.jobbole.com/82809/
> 点击 → 了解如何加入专栏作者

本系列：
《从零开始的Android新项目（1）：架构搭建篇》
《从零开始的Android新项目（2）：Gradle 篇》

前言

去年5月左右的时候，笔者在逛GitHub的时候，看到了一个MVP的项目，叫做mosby，仔细看了源码和作者介绍的文章后，发现确实有点意思，虽然会需要多写几个类和方法，但是解决了activity/fragment过重的问题，通过V/P分离能够帮助提高可维护性。时至去年年底，今年年初，MVP才逐渐被大家所知，也不时看到些文章介绍其概念和实践。

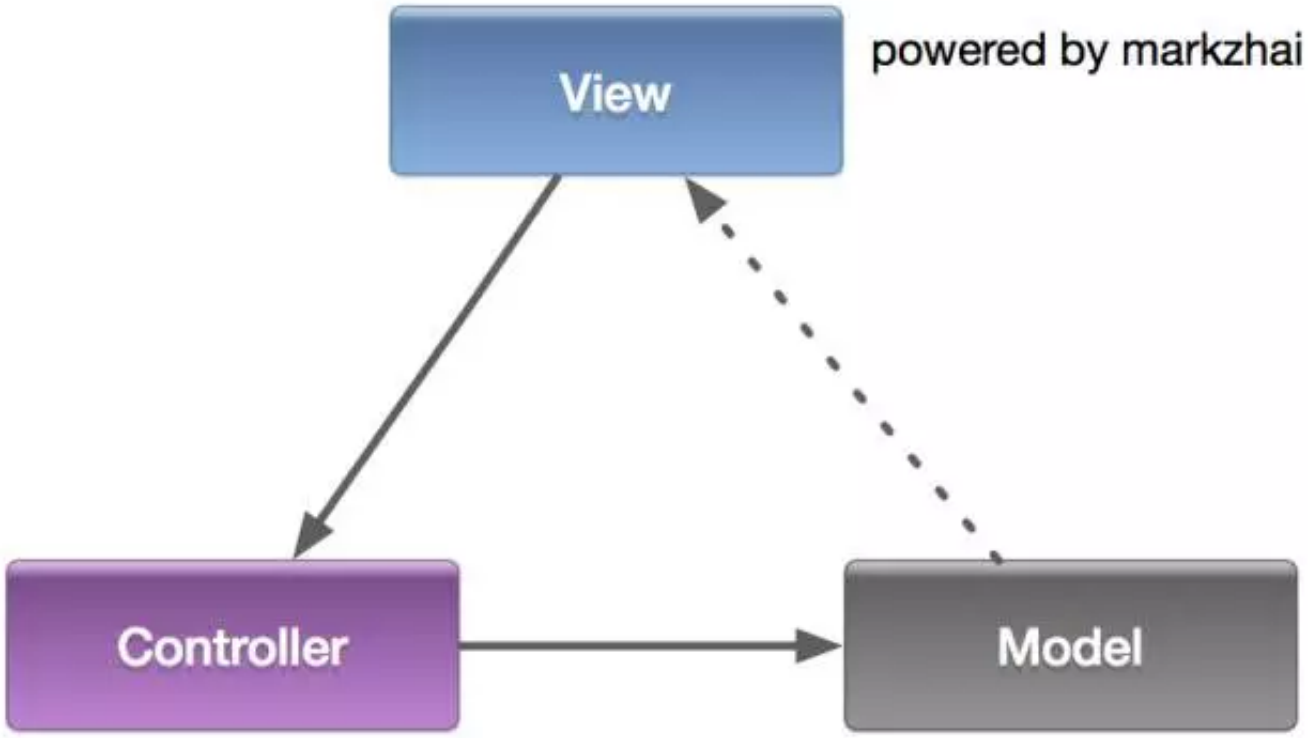
再说说MVVM (Model-View-ViewModel)，在Android上对应data binding。即ViewModel到View的映射，不需要再去自己找到view，然后更新字段，而是在映射建立后直接更新ViewModel然后反映到View上。

值得一提的是，MVP和MVVM都是微软提出的理念，最早都是在WPF里面被应用的，只是时至今日才在Android上被真正用起来。本文不是来介绍这两个的，所以不再赘述，讲讲正题。

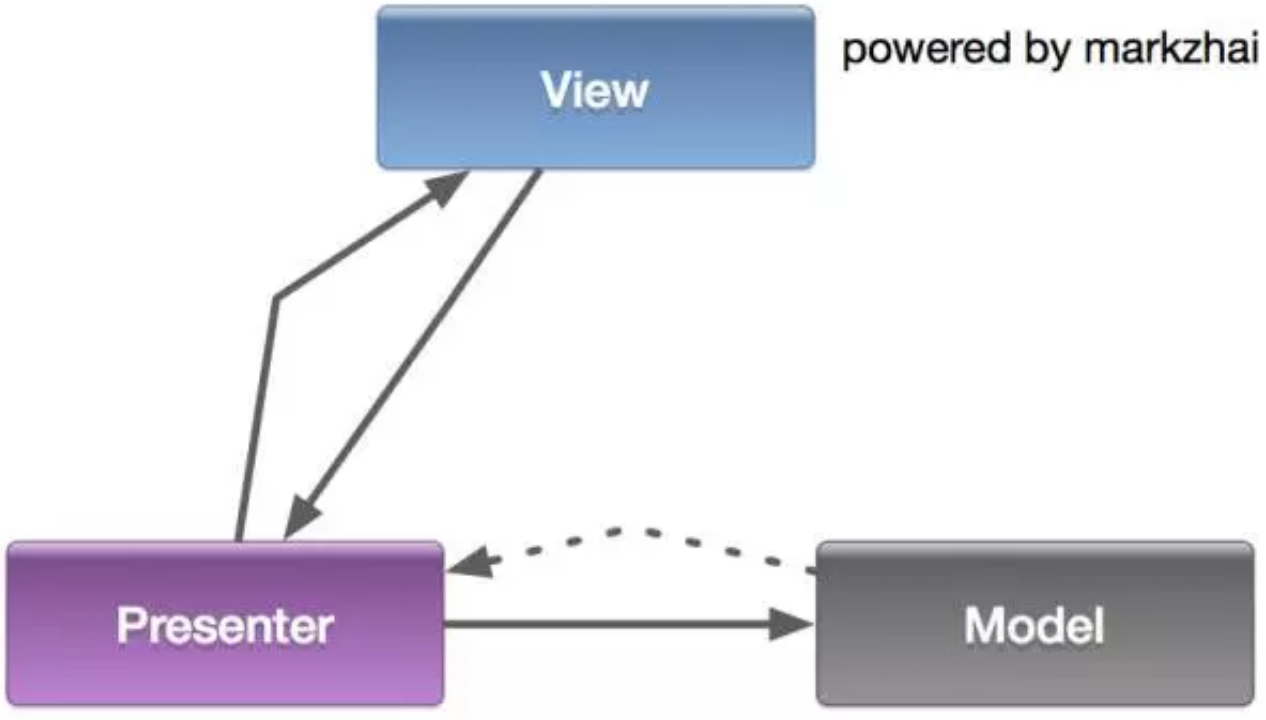
在本系列首篇中我曾经提到过在我设计的新应用中，采用了MVP+MVVM的混合（当初也考虑过Flux，但感觉并不合适Android），后来有一次CJJ同学和我探讨这个架构的时候，问到了我有没有什么正式的名字，我就有楞，因为这个组合和应用是我自己设计的，所以还真没想过这个问题，Google一搜，还真有这么个东西（见参考资料，文章写得很棒，建议英文不错的同学读一读）！

这就是本文我要介绍的东西，MVPVM (Model-View-Presenter-ViewModel)。
Quick glance

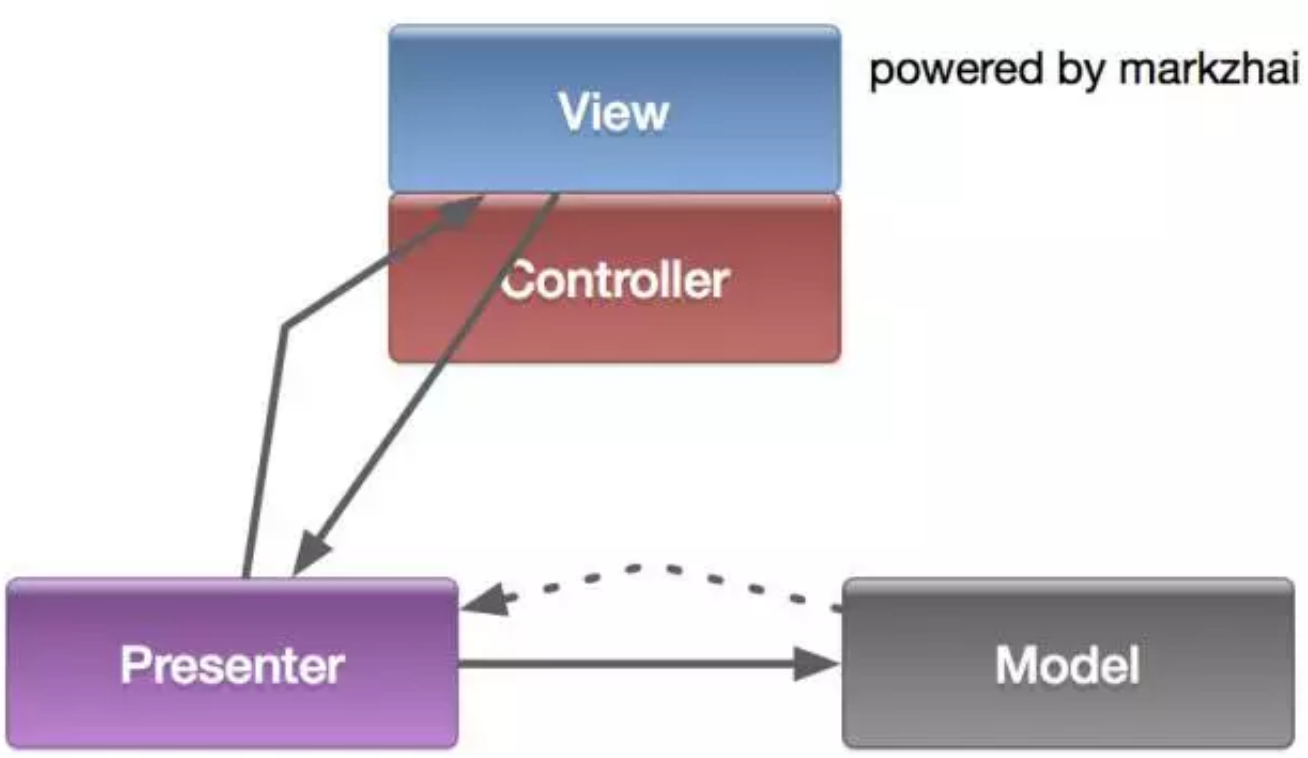
以下所有Model，并不单单指的是Bean，而是Model层，更像是repository或者business logic。



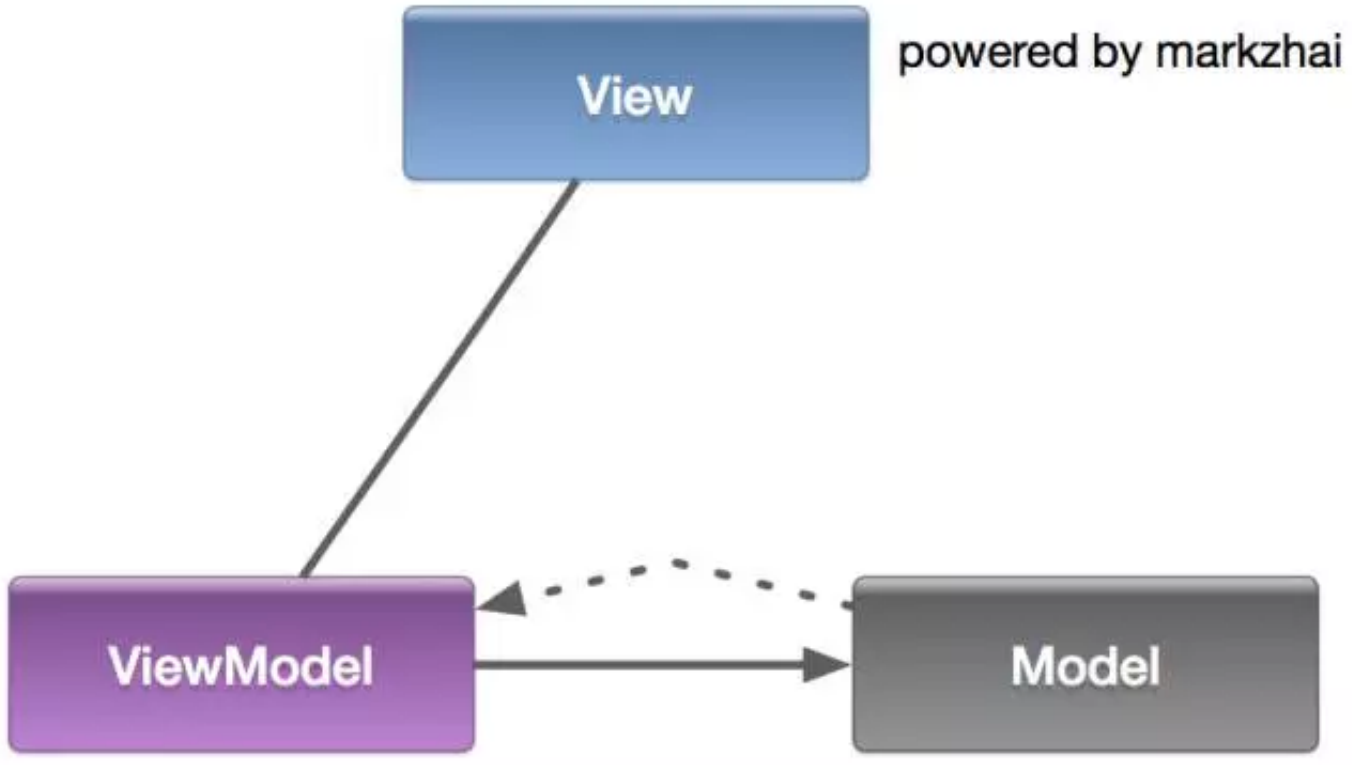
MVC: View持有Controller，传递事件给Controller，Controller由此去触发Model层事件，Model更新完数据（如从网络或者数据库获得数据后）触发View的更新事件。



乍一看，MVP似乎是MVC的变种，即C的位置被P取代了，但如果我们再看一看下图：



其实MVP是MVC的一个wrap，C层仍然可以在那里，代替View处理点击事件、数据绑定、扮演ListView的观察者，从而View可以专注于处理纯视觉的一些东西。而Presenter则避免了Model直接去触发View的更新，View彻底成为了一个被动的东西，只有Presenter告知其更新视觉，它才会去更新，比如showLoading(), showEmpty()。



MVVM通过View和ViewModel的双向绑定，让我们可以

- 直接更新ViewModel，View会进行对应刷新
- View的事件直接传递到ViewModel，ViewModel去对Model进行操作并接受更新。

Why MVPVM

如果你仔细读过Clean architecture的源码，会发现其中已经有了ViewModel这一层。如果你熟悉DO（Domain Object），PO（Persistent Object），VO（View Object），或许了解visibility这个概念，各层只需要知道其应该知道的。这些Object代表了完全独立不同的概念。

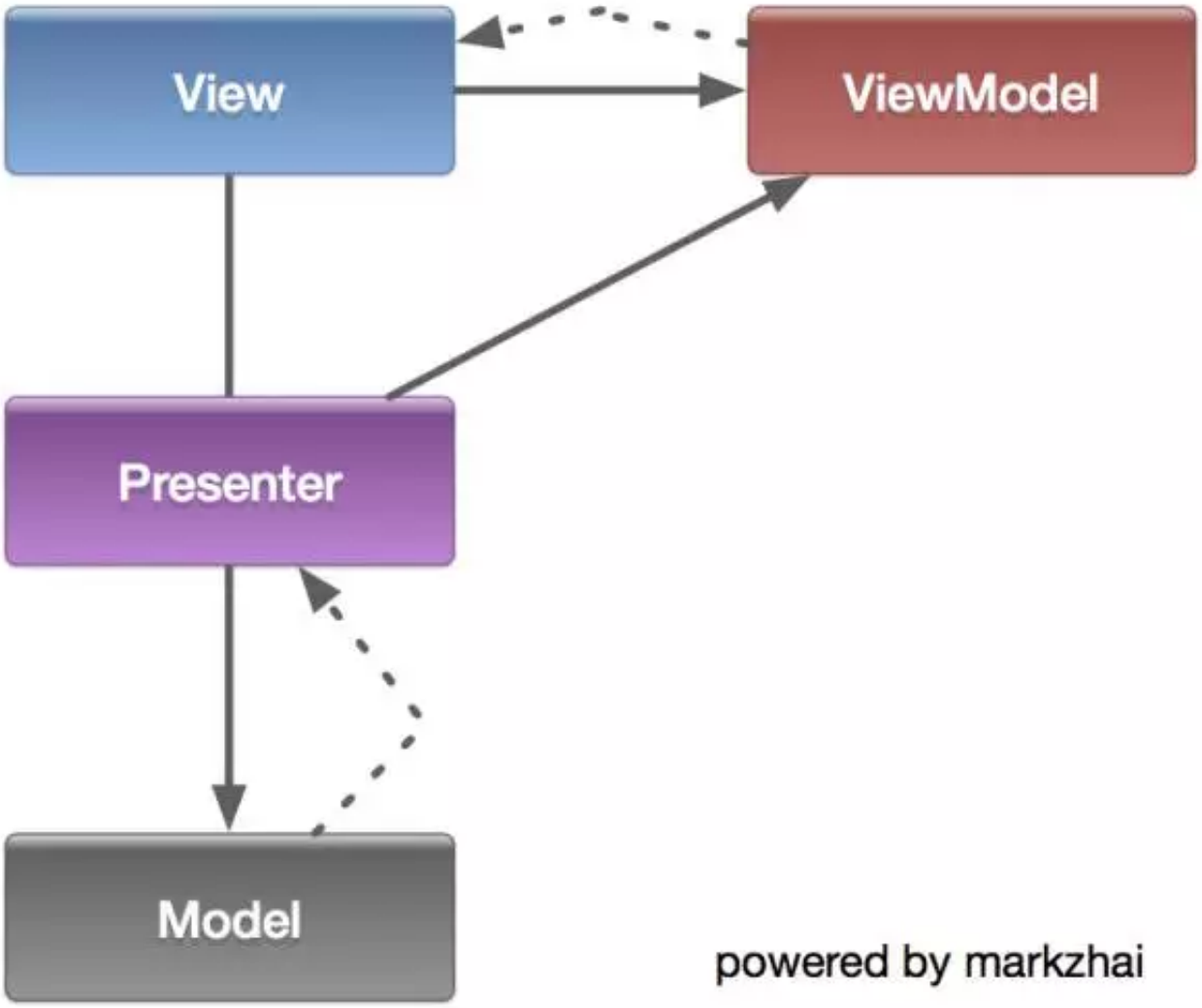
ViewModel层的必要性，简单举个例子，服务器传来一个Date String，但我们需要显示的是该Date到现在的相对时间描述，比如1分钟前，2天前，为了避免在view中绑定数据时去做这个逻辑，ViewModel会代替来进行这个的转换。

MVVM尽管确实省去了绑定数据到View的boilerplate，但

- ViewModel引用了View，从而导致ViewModel无法重用于其他View。
- 并没有解决View层过重的问题，仅仅去掉了数据绑定，尤其对一些复杂业务逻辑的页面。

模式的引入都是为了通过可拔插化以提高可复用性，松耦合和尽量小的接口可以给予最大的可复用性，使得组件能重组使用。

所以有了MVPVM：



在我的个人实践中:

- Model: data和domain模块组成，包含了Interactor（UseCase）、Repository、Datastore、Retrofit、Realm、DO、部分PO等。
- View: Activity/Fragment。
- Presenter：Presenter，包含了Subscriber，并通过Dagger2注入UseCase从而减轻耦合。
- ViewModel：由Model转换而成，继承BaseObservable或SortedList，大部分直接wrap了model，从而去掉了mapper的boilerplate。通过Data Binding绑定到xml。

从Presenter的Subscriber往下都是RxJava的流世界，stream in stream out。如果你原来就应用了MVP或者Clean Architecture，那会发现再加上ViewModel简直太简单了，同时让代码库更小，逻辑更清晰。

接着看看各个组件在MVPVM中的standing。

MVPVM: Model

实际对应的是Repository层，即第一篇文章中提到的data/domain module。具体的Model理论上应该是PO，但我们大部分场景并不需要PO，所以也可以是domain层的DO。

MVPVM: View

View对ViewModel不需要了解太多，这样才能保持两者的解耦，两者之间的协议只需要：

- ViewModel支持View需要展示的properties。
- View实现了ViewModel的观察者模式接口（如Listener）。

所以这里ViewModel到View是一条虚线，而不是MVVM中的双向实线。

MVPVM: Presenter

和在MVP一样，Presenter站在View和Model层之间。这里值得一提的是Presenter到ViewModel是有耦合的，因为Presenter需要把model更新到ViewModel中，也就是map行为，然后调用View的对应接口进行binding。

Presenter是MVPVM中唯一不需要解耦的，它紧紧地与View、ViewModel、Model层耦合。如果你的Presenter被多个View重用了，那你可能需要考虑它是不是更应该作为一个module，比如（第三方）登陆。

MVPVM: ViewModel

MVPVM让ViewModel可以重用，因为它再也不是直接和特定View绑定，而仅仅作为数据到View的一个绑定用展示。ViewModel因为用户操作而触发的事件不再直接对Model进行操作，而由View去负责任务流。ViewModel本身基本没有field，而是通过暴露get方法来让data binding找到对应要显示的property，get方法中直接调用持有的model的对应属性get方法。

理想化的架构是通过一个mapper类进行转换，但我想大部分的程序员面对这个工作都会抓狂，毕竟很多字段其实就是一个复制，而且对性能也有一些影响（遍历list，new对象，一个个字段转换，添加到新的list）。所以折中地，让ViewModel持有Model，在get方法中直接返回对应model的具体字段，在一些特殊的field如相对时间、添加一些描述性字符的地

方再去进行拼接和特殊处理。

啊，对了，说到ViewModel，Data Binding现在支持双向绑定了哦，
见 https://halfthought.wordpress.com/2016/03/23/2-way-data-binding-on-android/，语法如：

```
<EditText android:text="@={user.firstName}" .../>
```

不同于单向绑定的@{}，使用了@={}，毕竟双向绑定这个东西还是慎用，一方面早成数据流混乱不好理解，另一方面容易出现死循环。

NO Presenter

在MVP中，我们有时候碰到的问题是，Presenter真的有必要存在吗，尤其是一些较为静态，没什么业务逻辑，只需要纯展示的页面，结果就是为了MVP而特意去创建一个Presenter。

所以Presenter不应该被强求，正如MVP中，V和C其实被并在了一起，在某些情况下（确实就是个纯展示，或者很少的业务逻辑），应该允许去Presenter，并让View承担其任务。比如注册页面，我真的就只是想把用户的输入发到服务器验证一下，何必非得去搞一个presenter套着呢？

我们不能永远理想化地去选择所谓最好的设计，在现实的必要情况下，我们要敢于舍弃，最合适的设计才是最好的设计。为此，Presenter不是强制的；为此，ViewModel并不一定通过mapper生成，而可以返回持有的DO对象对应字段。

总结

本篇讲了讲MVPVM及其在Android的实践，因为时间原因来不及写个demo来说说具体实现，欢迎大家提出意见和建议。有空的话我最近会在GitHub上写一下demo，你如果有兴趣可以follow一下等等更新: markzhai。

下集预告

Dagger匕首，比ButterKnife黄油刀锋利得多。Square为什么这么有自信地给它取了这个名字，Google又为什么会拿去做了Dagger2呢？笔者看了很多国内Dagger2的文章，但发现它们都保留在介绍API和官网翻译的层面，无法让读者能明白究竟为什么用Dagger2，又如何用好Dagger2。希望能在下一次为大家讲清楚。

参考资料

- MVPVM Design Pattern – The Model-View-Presenter-ViewModel Design Pattern for WPF

专栏作者简介（ 点击 → 加入专栏作者 ）

markzhai：Software Engineer, Android, JavaScript, Data Mining, Security。曾就职Google HK，腾讯，阿里。

打赏支持作者写出更多好文章，谢谢！

阅读原文