Android开发中的MVP架构

2016-01-05 安卓应用频道 安卓应用频道

(点击上方公众号，可快速关注)

来源：小鄧子（@Rx小鄧子）

链接：http://www.jianshu.com/p/7567ed0d1853

最近越来越多的人开始谈论架构。我周围的同事和工程师也是如此。尽管我还不是特别深入理解MVP和DDD，但是我们的新项目还是决定通过MVP来构建。

这篇文章是我通过研究和学习各种文章以及专题讨论所总结出来的，它包括以下几点：

- 为什么越来越多的人开始关注架构?
- 首先，MVP是什么?
- 哪种架构才是最好的，MVC，MVVM还是MVP?
- MVP的利与弊
- Show me the code!!!代码展示

不幸的，这篇文章将不包括：

- 详细生动的代码示例
- 如何编写测试代码

最后，我将告诉你如何更进一步学习这些专题。

顺便提一下，我于上周在当地的一个研讨会上对MVP架构进行了相关演讲。这篇文章与当时的演讲内容相差无几。

（译者注：阅读更多请点击原作者PPT）

介绍~Activity是上帝类~

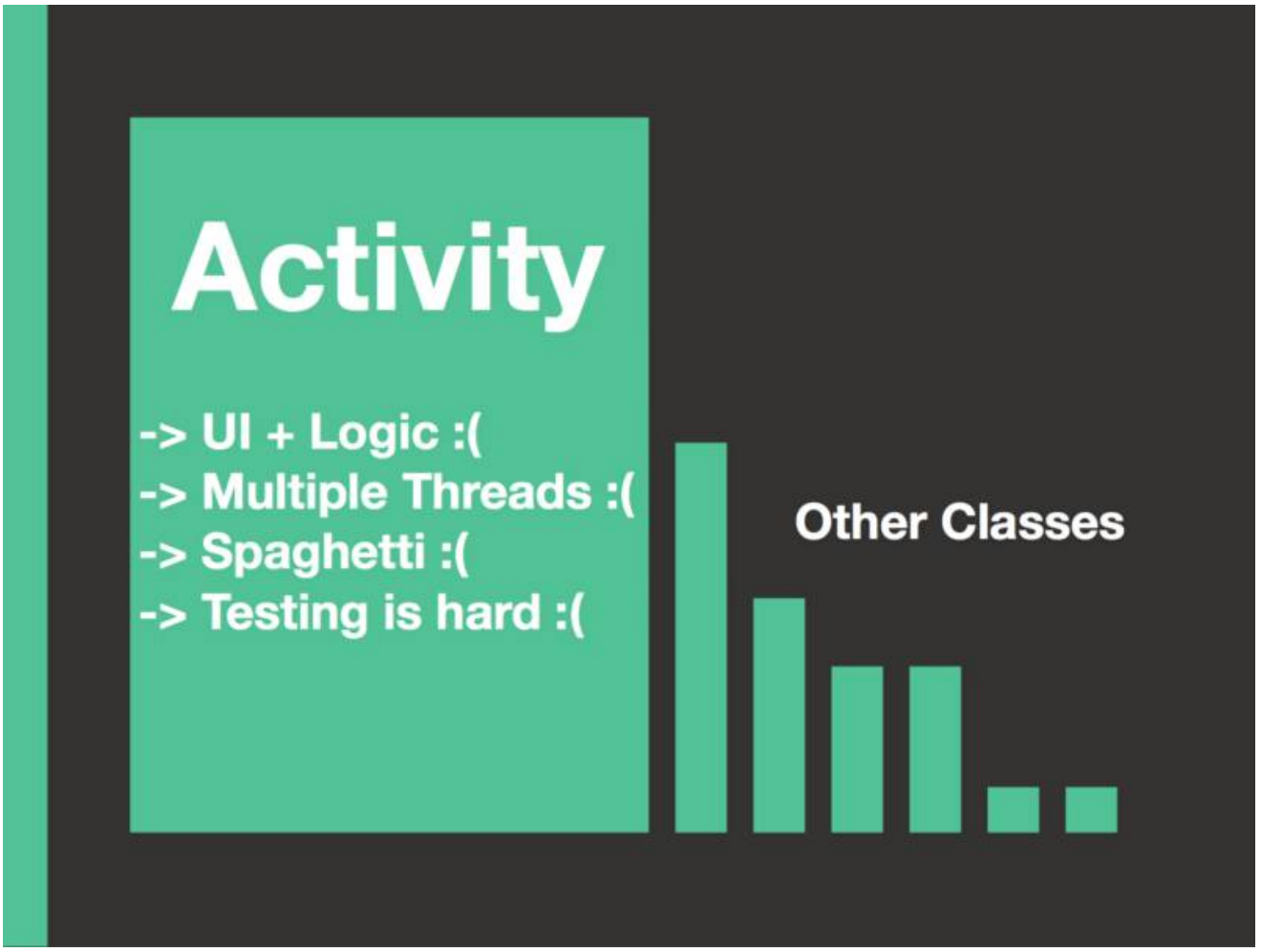
首先，让我们思考一下为什么在Android开发中如此迫切地需要一个清晰的软件架构。

该段摘自“代码大全第二版”：

> **避免创建神类。**避免创建无所不知，无所不能的上帝类。如果一个类需要花费时间从其他类中通过Get()和Set()检索数据（也就是说，需要深入业务并且告诉它们如何去做），所以是否应该把这些功能函数更好的组织到其它类而不是上帝类中。（Riel 1996）

上帝类的维护成本很高，你很难理解正在进行的操作，并且难以测试和扩展，这就是为什么要避免创建上帝类的黄金法则。

然而，在Android开发中，如果你不考虑架构的话，Activity类往往会越来越大。这是因为，在Android中，允许View和其它线程共存于Activity内。其实最大的问题莫过于在Activity中同时存在业务逻辑和UI逻辑。这会增加测试和维护的成本。



Activity是上帝

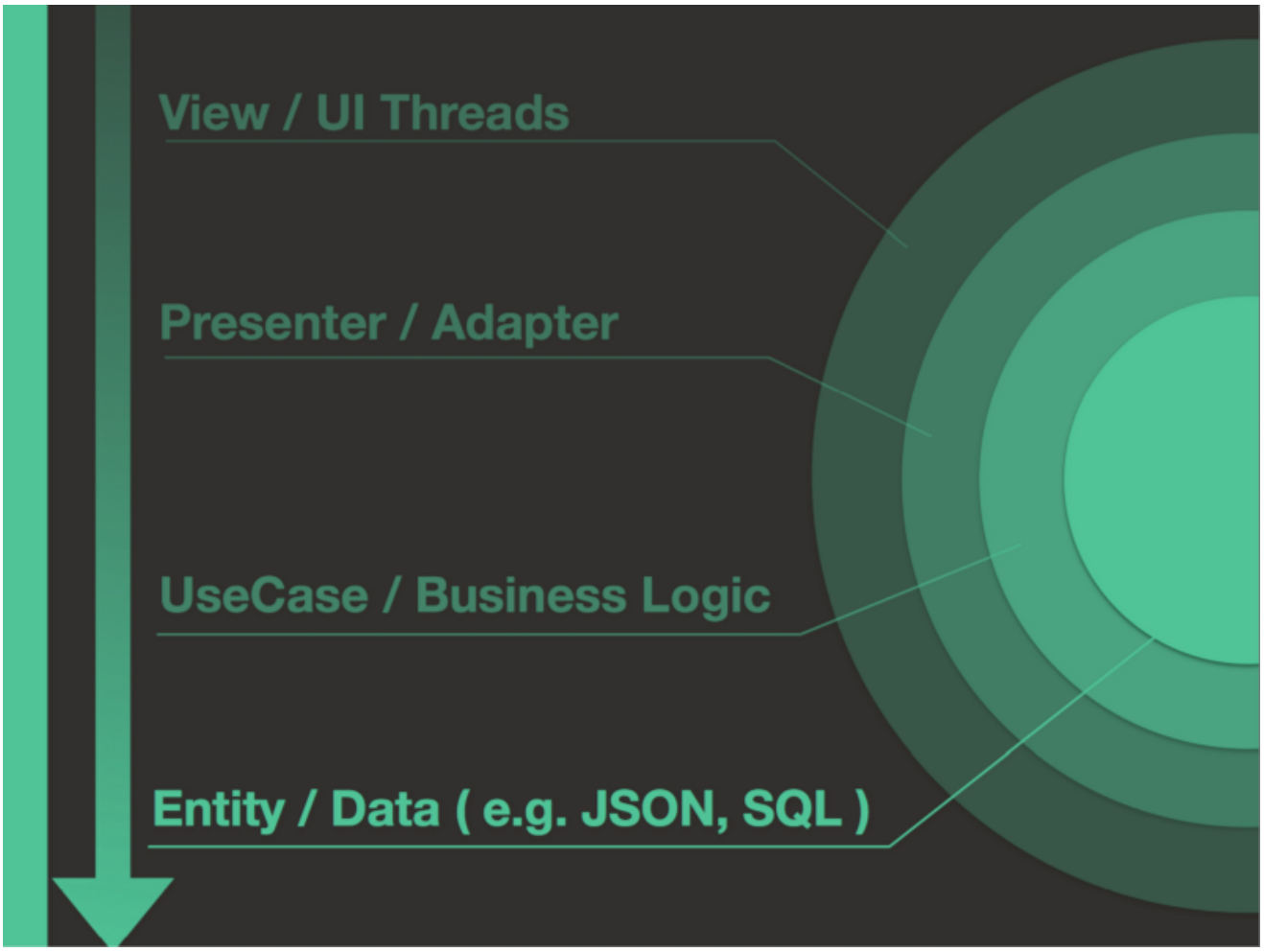
这是为什么需要清晰架构的原因之一。不仅会造成Activity的臃肿，还会引起其他问题，如使Activity和Fragment的生命周期变复杂，以及数据绑定等。

什么是MVP?

MVP代表Model，View和Presenter。

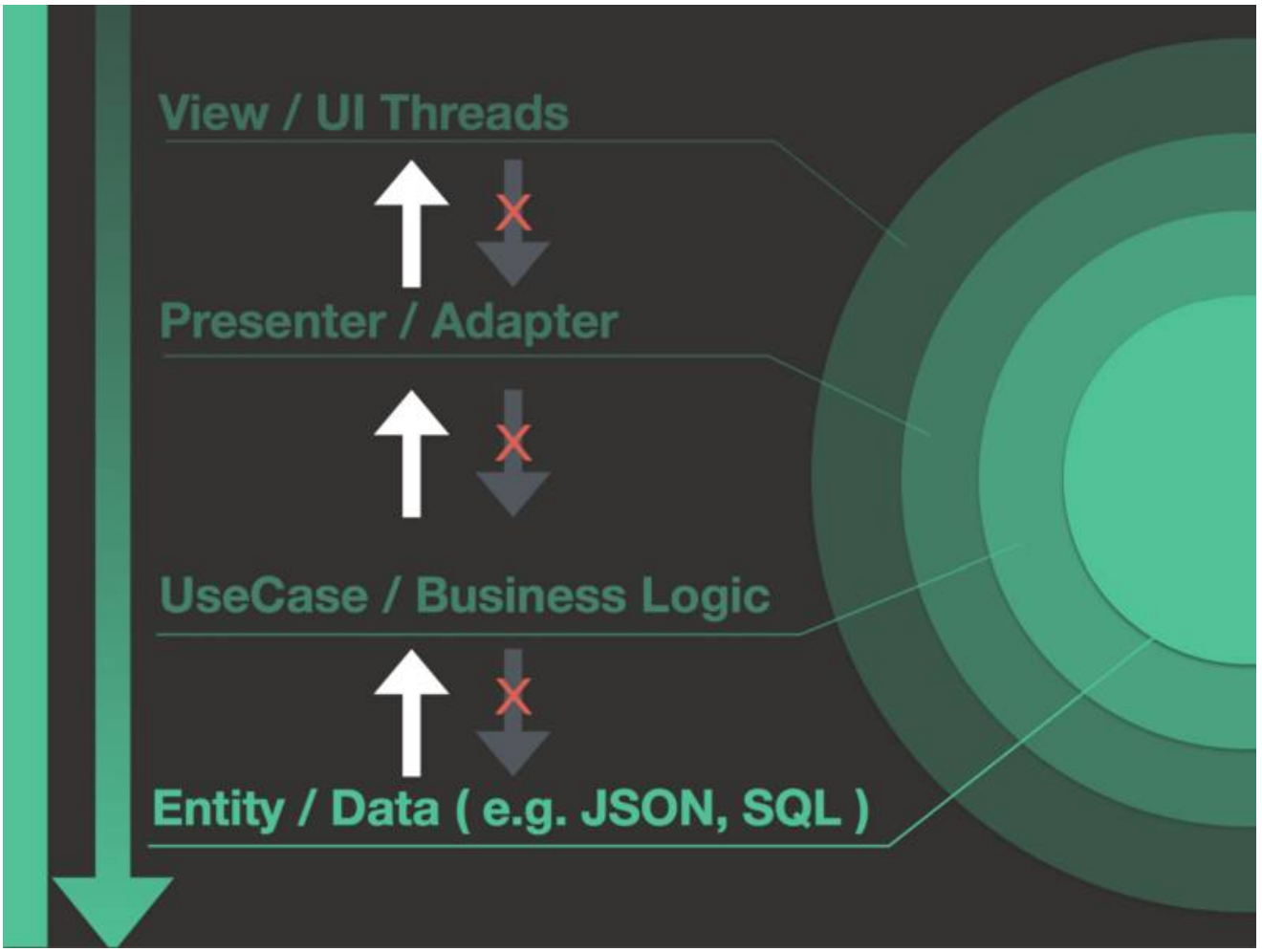
- View层负责处理用户事件和视图部分的展示。在Android中，它可能是Activity或者Fragment类。
- Model层负责访问数据。数据可以是远端的Server API，本地数据库或者SharedPreference等。
- Presenter层是连接（或适配）View和Model的桥梁。

下图是基于MVP架构的模式之一。View是UI线程。Presenter是View与Model之间的适配器。UseCase或者Domain在Model层中，负责从实体获取或载入数据。依赖规则如下：



The Dependency Injection

关键是，高层接口不知道底层接口的细节，或者更准确地说，高层接口不能，不应该，并且必须不了解底层接口的细节，是（面向）抽象的，并且是细节隐藏的。



The higher interfaces do not know about the details of the lower ones

依赖规则?

Uncle Bob的“The Clean Architecture”描述了依赖的规则是什么。

> 同心圆将软件划分为不同的区域，一般的，随着层级的深入，软件的等级也就越高。外圆是实现机制，内圆是核心策略。

这是上面片文章的摘要：

Entities：

- 可以是一个持有方法函数的对象
- 可以是一组数据结构或方法函数
- 它并不重要，能在项目中被不同应用程序使用即可

Use Cases

- 包含特定于应用程序的业务规则
- 精心编排流入Entity或从Entity流出的数据
- 指挥Entity直接使用项目范围内的业务规则，从而实现Use Case的目标

Presenters,，Controllers

- 将Use Case和Entity中的数据转换成格式最方便的数据
- 外部系统，如数据库或网页能够方便的使用这些数据
- 完全包含GUI的MVC架构

External Interfaces, UI, DB

- 所有的细节所在
- 如数据库细节，Web框架细节，等等

MVC，MVP还是MVVM?

那么，哪一个才是最好的呢？哪一个比其他的更优秀呢？我能只选择一个吗？

答案是，NO。

这些模式的动机都是一样的。那就是如何避免复杂混乱的代码，让执行单元测试变得容易，创造高质量应用程序。就这样。

当然，远不止这三种架构模式。而且任何一种模式都不可能是银弹，他们只是架构模式之一，不是解决问题的唯一途径。这些只是方法、手段而不是目的、目标。

利与弊

OK，让我们回到MVP架构上。刚刚我们了解了什么是MVP，讨论了MVP以及其它热门架构，并且介绍了MVC，MVP和MVVM三者间的不同。这是关于MVP架构利与弊的总结：

****利**

- 可测试（TDD）
- 可维护（代码复用）
- 容易Reviewe
- 信息隐蔽

****弊**

- 冗余的，尤其是小型App开发
- （有可能）额外的学习曲线
- 开始编写代码之前需要时间成本（但是我敢打赌，设计架构是所有项目开发所必需的）

Show me the code!!!

这里仅展示了MVP模式的一小段结构。如果你想了解更多项目或生动的代码示例，请参考文章末尾的“链接和资源”。那里有非常丰富和设计巧妙的示例，基本都托管在Github上，以便你能clone，在设备上运行，并了解工作原理。

首先，为每一个View定义接口。

```
/**
* Interface classes for the Top view
*/
public interface TopView {
/**
* Initialize the view.
*
* e.g. the facade-pattern method for handling all Actionbar settings
*/
void initViews();
/**
* Open {<a href="http://www.jobbole.com/members/57845349">@link</a> DatePickerDialog}
*/
void openDatePickerDialog();
/**
* Start ListActivity
*/
void startListActivity();
}
```

让我们重写TopView类，要点如下：

- TopActivity只是负责处理事件监听或者展示每个视图组件
- 所有的业务逻辑必须委托给Presenter类
- 在MVP中，View和Presenter是一 一对应的（在MVVM中是一对多的）

```
public class TopActivity extends Activity implements TopView {
// here we use ButterKnife to inject views
/**
* Calendar Title
*/
@Bind(R.id.calendar_title)
TextView mCalendarTitle;
private TopPresenter mTopPresenter;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_top);
ButterKnife.bind(this);
// Save TopPresenter instance in a meber variable field
mTopPresenter = new TopPresenter();
mTopPresenter.onCreate(this);
}
/*
* Overrides method from the {<a href="http://www.jobbole.com/members/57845349">@link</a> TopView} interfaces
*/
@Override
public void initViews() {
// Actionbar settins
// set event listeners
}
@Override
```

```
public void openDatePickerDialog() {
DatePickerFragment.newInstance().show(getSupportFragmentManager(),
DatePickerFragment.TAG);
// do not write logic here... all logic must be passed to the Presenter
mTopPresenter.updateCalendarDate();
}
@Override
public void startListActivity() {
startActivity(new Intent(this, ListActivity.class));
}
}
```

这是Presenter类，最重要的一点是Presenter仅仅是连接View与Model的适配桥梁。比如，TopUseCase#saveCalendarDate()是对TopPresenter细节隐藏的，同样对TopView也是如此。你不需要关心数据结构，也不需要关心业务逻辑是如何工作的。因此你可以对TopUseCase执行单元测试，因为业务逻辑与视图层是分离的。

```
public class TopPresenter {
@Nullable
private TopView mView;
private TopUseCase mUseCase;
public TopPresenter() {
mUseCase = new TopUseCase();
}
public void onCreate(@NonNull TopView topView) {
mView = topView;
// here you call View's implemented methods
mView.initViews();
}
public void updateCalendarDate() {
// do not forget to return if view instances is null
if (mView == null) {
return;
}
// here logic comes
String dateToDisplay = mUseCase.getDateToDisplay(mContext.getResources());
mView.updateCalendarDate(dateToDisplay);
// here you save date, and this logic is hidden in UseCase class
mUseCase.saveCalendarDate();
}
}
```

当然，尽管业务逻辑被实现在Activity类中，你依然可以执行单元测试，只不过这会耗费很多时间，而且有些复杂。可能需要更多的时间来运行App，相反，你本应该充分利用测试类库的性能，如Robolectric。

总结

这里没有万能药，而且MVP也仅仅是解决方案之一，它可以与其他方法协同使用，同样，也可以有选择的用于不同项目。

链接和资源

The Clean Architecture（译者注：清晰架构。译文） – Uncle Bob
http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

这篇文章由Uncle Bob撰写，描述了依赖规则的样子和它们之间的组件是如何工作的。我从一开始谈论的那张图表的灵感就来源于他的文章，虽然这篇文章不是针对Android开发的，但是同往常一样，字里行间蕴藏着很多精辟的道理，所以，必读。

Architecting Android...The clean way?（译者注：Android中的清晰架构。译文）- Fernando Cejas

http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/

我认为这是在探索如何将MVP架构到Android开发专题中最著名，也是最受欢迎的博客。我也是从他那篇简单易读，书写良好的博客中偶然发现“MVP”这个名词的。他的示例代码托管在Github上，以便那些想要将MVP架构运用到正式App上的Android开发者clone到。

Android Architecture（译者注：Android架构） – Thanos Karpouzis

https://medium.com/android-news/android-architecture-2f12e1c7d4db#.7ch5tkkx0

一个在Android项目中运用MVC，MVP，MVVM的简单指导。我从他的那篇普通却不平凡的文章中学到了很多，尤其是MVC，MVP和MVVM之间的不同。

Software Design patterns on Android English（译者注：Android开发中的软件设计模式） – Pedro Vicente Gómez Sánchez

http://www.slideshare.net/PedroVicenteGmezSnch/software-design-patterns-on-android

这是一个在Karumi工作的高级Android开发工程师所讲的，他解释了一些MVP架构中的设计模式（如，渲染模式，仓库模式和命令模式）。如果你想深入理解MVC或者MVP，那这就使你要找的。

M — Model in MVC, MVP, MVVC in Android（译者注：MVC，MVP，MVVC架构中Model层在Android中的定义） – Artem Zinnatullin

https://medium.com/@artem_zin/m-model-from-mvc-mvp-in-android-flow-and-mortar-bd1e50c45395#.82qjulegz

如果你不还了解Model层中的JSON与SQL，或者不能透彻理解Model层的图像模型，这篇文章将带你进一步理解什么是Model层以及为什么Model层独立于其他层。其中“Model layer is solution”部分很好的解释了如何通过面向接口的方式编写测试。