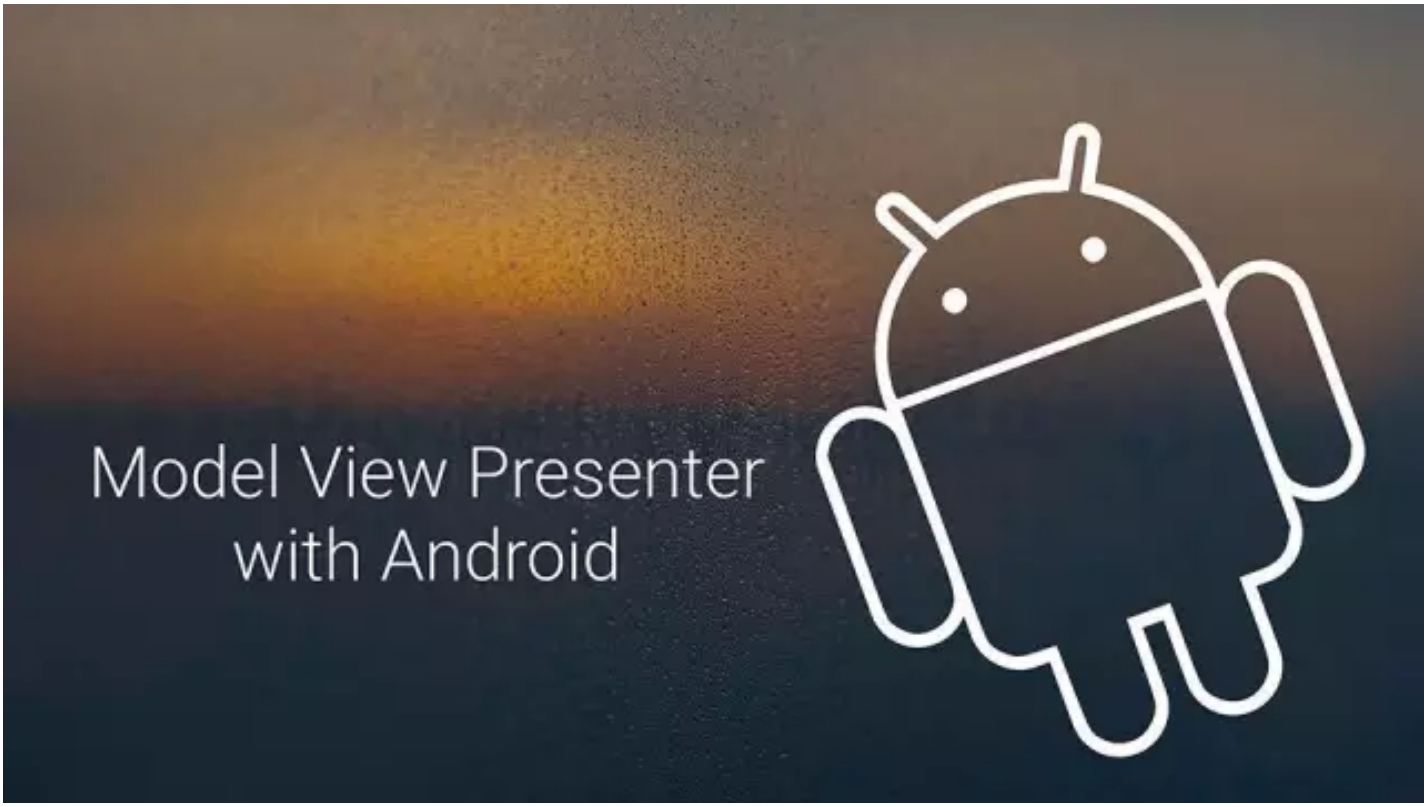
Android官方MVP架构示例项目解析

Original 2016-04-12 吕英斌 移动开发前线



> 前段时间Google在Github推出了一个项目，专门展示Android引用各种各样的MVP架构，算是官方教程了。趁着还新鲜，让我们来抛砖引玉一探究竟，看看在Google眼里什么样算是好的MVP架构。

App架构在Android开发者中一直是讨论比较多的一个话题，目前讨论较多的有MVP、MVVM、Clean这三种。google官方对于架构的态度一直是非常开放的，让开发者自主选择组织和架构app的方式，期望能留给开发者更多的灵活性。

由于没有一套权威的架构实现，现在很多App项目中在架构方面都有或多或少的问题。第一种常见问题是没有架构，需求中的一个页面对应项目中的一个activity或一个fragment，所有的界面响应代码、业务逻辑代码、数据请求代码等等都集中在其中。第二种常见的问题是架构实现的不断变化，不断在各种架构间摇摆，一直找不到一个适合自己的架构。

就在近日，google在官方示例中给出了一系列不同架构的app实现，这对于一直困惑于到底该用何种架构的android开发者来说确实是福音，开发者只要根据自己项目的情况，在不同的实现中选择一种即可，当然google也明确表示了这些示例只是用来做参考，而并不是要为了当做标准，下面先为大家简单介绍下此项目。

项目介绍

Google把这个项目命名为：Android架构蓝图。

项目地址为：https://github.com/googlesamples/android-architecture

下面的内容引用自项目说明：

> 项目目的是通过展示各种架构app的不同方式来帮助开发者解决架构问题。项目中通过不同的架构概念及方式实现了功能相同的app。你可以用示例来当做参考，或是干脆拿来当做创建app项目的基础。项目中，希望大家能把关注点集中到代码结构、整体架构、可测试性、可维护性这四个方面。当然实现app有很多种方式，千万不要把它当做定式。
>
> **项目中有哪些示例**
>
> 目前已经完成的示例有
>
> - todo-mvp（mvp基础架构示例）
> - todo-mvp-loaders（基于mvp基础架构项目，获取数据部分使用了Loaders架构）
> - todo-mvp-databinding（基于mvp基础架构项目，使用了数据绑定组件）
>
> 仍在进展中的示例有
>
> - todo-mvp-contentproviders（基于mvp基础架构项目，使用了Content Providers）
> - todo-mvp-clean（基于mvp基础架构项目，使用了clean架构的概念）
> - todo-mvp-dagger（基于mvp基础架构项目，使用了dagger2进行依赖注入）
>
> **如何进行选择**
>
> 这个还是需要开发者自己来做决定，每个项目的说明文件中都说明了该实现的特性。app规模、团队状况、维护工作量的大小、平板是否支持、代码简洁程度偏好，这些都会影响你的选择。

到了这里，想必大家都很想一探究竟了，到底官方示例是如何实现的呢？还是那句话，源码面前，了无秘密。为了能够更好的理解官方mvp架构实现，下面我们从源码的角度来分析todo-mvp（mvp基础架构示例）的实现。我们先从项目的整体组织方式开始，再看项目究竟使用了哪些组件，最后当然是最重要的具体mvp的实现方式。

源码分析

项目代码组织方式

项目含一个app src目录，4个测试目录，分别是androidTest（UI层测试）、androidTestMock（UI层测试mock数据支持）、test（业务层单元测试）、mock（业务层单元测试mock数据支持）。src目录的代码组织方式完全是按照功能来组织的，功能内部分为xactivity、xcontract、xfragment、xpresenter四个类文件(x代表业务名称)。

平时用到较多的另一种组织方式是按照类型，比如按照activity、adapter、fragment、contract、presenter进行划分，不同的类文件分别放到不同的目录中，笔者觉得两种方式没有什么太大的区别，完全看个人喜好了。

组件使用

由于项目是基于gradle进行编译的，所以我们可以从build.gradle文件看到项目依赖的全貌。

Guava

项目中使用到了Guava库（https://github.com/google/guava），该库是Google在基于java的项目中都会引用到得一个库，库中包含大约14k的方法数，是个很大的库，其中包含了集合、缓存、并发、基本注解、字符串处理、io处理等等。项目中使用Guava库主要是处理null这种不安全的情况，因为一般我们在使用有可能为null的对象时，一般会增加一次判断，代码如下：

```
String possible = value;
if(TextUtils.isEmpty(value)) {
  Log.e(“tag”,”value is empty”);
} else {
  Log.e(“tag”,”value is “+value);
}
```

而如果有Guava的时候，可以通过如下方式

```
Optional possible = Optional.fromNullable(emptyToNull(value));
Log.e(“tag”,”value is “+possible.or(“empty”));
```

这样面对空的时候，就不用再多写很多代码了，确实是方便了很多。但是不建议为了null安全直接引入如此大的一个库，因为我们都知道android apk的65k方法数限制，如果要用的话可以把源码中涉及到得部分直接拿出来用。当然Guava中还有很多重要的功能，其他功能读者可以自行研究，关于Guava就先到这里了。

测试相关组件

示例项目在可测试方面做的非常好，由于对视图逻辑(view层)和业务逻辑(presenter层)进行了拆分，所以我们就可以对UI、业务代码分别进行测试。为了进行UI测试引入了Espresso，为了对业务层进行单元测试引入了junit，为了生成测试mock对象引入了mockito，为了支撑mockito又引入了dexmaker，hamcrest的引入使得测试代码的匹配更接近自然语言，可读性更高，更加灵活。

项目MVP实现方式

这节我们就具体来看官方示例到底是如何实现mvp的。这里我们先看下总体的轮廓，关于项目中业务代码我们仅列出了任务详情页（taskDetail）的相关类，其他业务代码类似。

基类

我们首先来看两个Base接口类，BasePresenter与BaseView，两类分别是所有Presenter与View的基类。

BasePresenter中含有方法start(),该方法的作用是presenter开始获取数据并调用view中方法改变界面显示，其调用时机是在Fragment类的onResume方法中。

BaseView中含方法setPresenter，该方法作用是在将presenter实例传入view中，其调用时机是presenter实现类的构造函数中。

契约类

与笔者之前见到的所有mvp实现都不同，官方的实现中加入了契约类来统一管理view与presenter的所有的接口，这种方式使得view与presenter中有哪些功能，一目了然，维护起来也方便，实例如下

activity在mvp中的作用

activity在项目中是一个全局的控制者，负责创建view以及presenter实例，并将二者联系起来，下面是activity中创建view及presenter的代码

我们可以从上面看到整个创建过程，而且要注意的是创建后的fragment实例作为presenter的构造函数参数被传入，这样就可以在presenter中调用view中的方法了。

mvp的实现与组织

实例中将fragment作为view层的实现类，为什么是fragment呢？有两个原因，第一个原因是我们把activity作为一个全局控制类来创建对象，把fragment作为view，这样两者就能各司其职。第二个原因是因为fragment比较灵活，能够方便的处理界面适配的问题。我们先看view的实现，我们只挑一部分重要的方法来看

上面可以看到setPresenter方法，该方法继承于父类，通过该方法，view获得了presenter得实例，从而可以调用presenter代码来处理业务逻辑。我们看到在onResume中还调用了presenter得start方法，

下面我们再看presenter的实现

presenter构造函数中调用了view得setPresenter方法将自身实例传入，start方法中处理了数据加载与展示。如果需要界面做对应的变化，直接调用view层的方法即可，这样view层与presenter层就能够很好的被划分。

最后还剩下model层实现，项目中model层最大的特点是被赋予了数据获取的职责，与我们平常model层只定义实体对象截然不同，实例中，数据的获取、存储、数据状态变化都是model层的任务，presenter会根据需要调用该层的数据处理逻辑并在需要时将回调传入。这样model、presenter、view都只处理各自的任务，此种实现确实是单一职责最好的诠释。

总结

到这里我们就基本分析完了，我们再来整体看下官方的实现方式有哪些特性。

首先是复杂度，我们可以从上面的分析看出整体的复杂度还是较低的，易学的；然后是可测试性，由于将UI代码与业务代码进行了拆分，整体的可测试性非常的好，UI层和业务层可以分别进行单元测试；最后是可维护性和可扩展性，由于架构的引入，虽然代码量有了一定的上升，但是由于界限非常清晰，各个类职责都非常明确且单一，后期的扩展，维护都会更加容易。有了这个架构之后，我们再回头看下之前的实现是不是有很多不足，没有关系，那么接下来就是在项目中进行实践的时间了。

由InfoQ主办的面向中高级移动开发者的GMTC全球移动技术大会（点击了解详情）将要召开了，去哪儿技术总监、《App研发录》作者包建强将为我们带来Android插件化：从入门到放弃。大会目前正值6折售票期间，预购从速！~