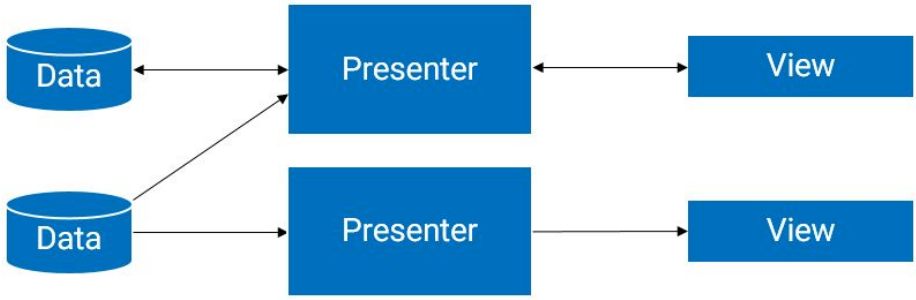
界面开发框架palette(MVPs)

zhulantian - 2015.09.14

MVP模式

在客户端的开发中，为了逻辑上的解耦，我们通常希望将数据层(M)、界面层(V)、控制逻辑(C)进行分离，也即我们通常所说的MVC模式，MVC模式的优点是三层逻辑之间是解耦的，每一层都可以单独的开发和测试。

Android中Activity/Fragment充当了Controller的角色，但由于Activity是一个比较大的组件，他兼具着处理界面(例如ActionBar)、处理各类系统事件(生命周期、消息等)，实际使用中如果还要把整个界面的控制逻辑写在Activity中，就会使得Activity显得非常的臃肿。此时，我们单独把Controller中与界面展现相关的代码剥离出来，于是产生了Presenter。Presenter单纯只用来控制界面显示，它是Controller的一个部分。



MVP组件关系

- **场景1：用户详情界面**

  Presenter会被用来将User对象展示到一个View上，它会交由Activity管理，同时Activity还负责处理横竖屏变化、打Log、处理ActivityResult等内容。

- **场景2：用户列表界面**

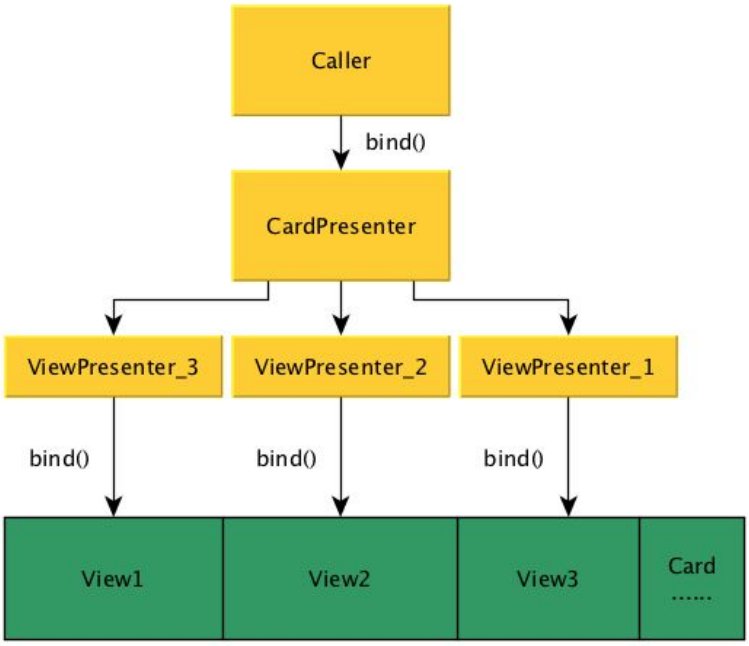
  Presenter会被用来将一个User展示到ListView的一个ItemView(ViewHolder)

上，Fragment通过Adapter控制Presenter。

以上是传统的MVP模式，其基本实现了分层的目的。但在客户端逐渐发展成为一个更为复杂的系统时，我们发现由于卡片开始包含更多的功能，例如一张卡片上可能展示视频、多张图片、附件、各种Span等等，此时这张卡片对应的Presenter变的非常的臃肿。随之带来的是这个Presenter的可复用性大幅下降，某些可能只有少量不用的卡片(例如没有视频展示)，只能通过继承原先的Presetner来进行改造，由于Java不能支持多继承，导致代码被迫越写约耦合，如此反复，使得整个工程开发和维护成本大幅提高。

MVPs

MVPs的目标是提高Presenter的复用程度，通过拆分一个大的CardPresenter为多个部分的ViewPresenter，再使用组合的模式进行拼装，达到每个小的ViewPresenter复用程度提高的目的。这些小的ViewPresenter就像不同颜色的颜料一样，不同的搭配就可以得到不同的画面(界面CardPresenter)，这也是框架Palette(调色板)的由来。



Presenter组合模式

Palette对CardPresenter的拆分是灵活的，它基于ID来分配ViewPresenter，最小粒度为单个View，并且一个ID允许设置多个ViewPresenter。

- 例如，你可以将一个播放视频的Surfaceview及其相关的播放按钮进度条等View一并作为一个ViewPresenter，这个ViewPresenter可以被多个需要播放视频的界面使用。

- 例如，一个View既要响应点击之后下载一个App的事件，同时还要显示App下载时的进度，你可以将Click写成一个ViewPresneter，将显示进度的逻辑写成另一个ViewPresenter，并把这两个ViewPresenter都设置给该View。这样的好处是，如果有其它的View只需要响应下载事件而不需要进度，此时可以直接复用。

- 代码示例

```
return new CardPresenter(view)
        .add(R.id.user_avatar, new PostBasicPresenter())
        .add(R.id.user_name, new PostBasicPresenter())
        .add(R.id.reason, new PostBasicPresenter())
        .add(R.id.publish_date, new PostBasicPresenter())
        .add(R.id.post_content, new PostContentPresenter())
        .add(R.id.user_role, new PostBasicPresenter())
        .add(R.id.location, new PositionPresenter())
        .add(R.id.attach_container, new PostAttachPresenter())
        .add(R.id.comment_container, new PostCommentsPresenter())
        .add(R.id.repost_btn, new
RepostPresenter(mPostShareDlg))
        .add(R.id.digg_btn, new DiggPostPresenter())
        .add(R.id.comment_btn, new FeedCommentPresenter())
        .add(R.id.operations, new
OperationPresenter(mPermission))
        .add(R.id.resend_btn, new ResendPresenter())
```

综上，Presenter的拆分是灵活的，原则是可预见的需要复用的先拆分，其它的尽可能合并，直到发现需要复用其中部分逻辑的时候，把这部分逻辑再拆分出去。

Data-Binding支持

Palette是为了提高Presenter的可复用性，因此在应对通用(简单)的Bind需求时需要更简单通用一些。

- 例如，不同User的卡片上元素可能不一样，A卡片可能是username，avatar，create_time，summary这些，B卡片可能只有username，avatar这两个。此时如果为了复用把username和avatar的Bind逻辑拆出去则显得有点繁琐，因为实际这两个字段的绑定只有两行代码(setText，setUrl)。

Android基于MVVM的Data-Binding框架提供了新的思路，即在XML中定义好需要使用的字段，运行时从绑定对象中取出对应的属性设置到View中，这个绑定的逻辑就是BindingPresenter，它支持对属性的赋值和运算。

原生的Data-Binding需要使用apt插件，同时由于是类型安全的，因此使用起来相对复杂一些。Palette采用弱类型语法，不check属性的类型，简化代码的复杂程度，但同时也要求了更高的代码编写规范性。

Palette BindingPresenter使用Demo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:topic="http://schemas.android.com/apk/res-auto"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:background="@drawable/bg_post_item"
              android:orientation="vertical">

   <TextView
       android:id="@+id/title"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:gravity="center_vertical"
       android:textColor="@color/post_text_color"
       android:textSize="13sp"
       topic:bindValue="${.mEntity.content}"/>

   <com.ss.android.image.AsyncImageView
       android:id="@+id/content_image"
       android:layout_width="match_parent"
       android:layout_height="300dp"
       android:layout_marginTop="@dimen/feed_margin"
       android:scaleType="centerCrop"
       topic:bindValue="${.mEntity.large_image_list[0]}"/>
```

```
</LinearLayout>
```

```
ViewGroup contentView =
    ViewInflater.inflate(parent, R.layout.post_item);
return new CardPresenter(contentView)
                .add(R.id.title, new BindingPresenter())
                .add(R.id.content_image, new BindingPresenter());
```

AQuery操作

为了配合Presenter的组合绑定模式(基于ID分配ViewPresenter)，因此引入了ViewHelper来辅助进行View操作。ViewHelper是对AQuery这个项目做的精简，基于此工具类可以对View的操作进行链式调用，同时省去ViewHolder来存储界面对象，其操作的单元也是ID，因此与Palette可以无缝的衔接。

➢ 代码示例：

```
helper().text(comment.mUser.mRole.mRoleName).textColor(R.color.
role_text_red).background(R.drawable.bg_user_role_red);
```

列表支持

Palette对列表的支持是基于Adapter的，对于ListView的Adapter需要集成CardAdapter并实现响应的方法即可。