Android应用中MVP最佳实践

2016-01-08 安卓应用频道 安卓应用频道

> 来源：Jude95
>
> 链接：http://www.jianshu.com/p/ed2aa9546c2c

所谓MVP(Model-View-Presenter)模式。是将APP的结构分为三层：

view – UI显示层

view 层主要负责：

5. 提供UI交互
6. 在presenter的控制下修改UI。
7. 将业务事件交由presenter处理。

注意. View层不存储数据，不与Model层交互。

presenter – 逻辑处理层

presenter 层主要负责：

5. 对UI的各种业务事件进行相应处理。也许是与Model层交互，也许自己进行一些计算，也许控制后台Task，Servic
6. 对各种订阅事件进行响应，修改UI。
7. 临时存储页面相关数据。

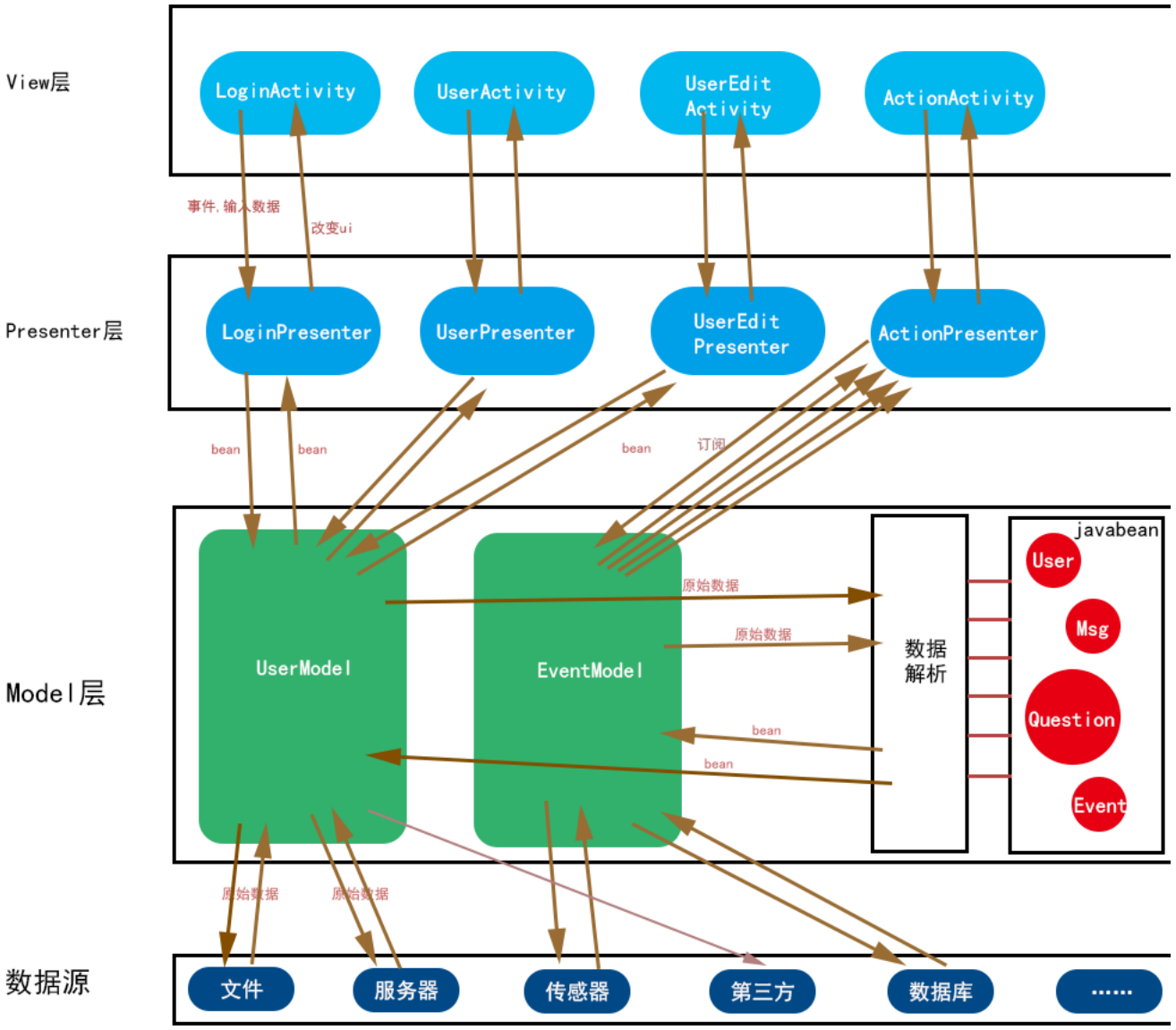
注意. Presenter内不出现View引用。

model – 数据层

model层主要负责：

5. 从网络，数据库，文件，传感器，第三方等数据源读写数据。
6. 对外部的数据类型进行解析转换为APP内部数据交由上层处理。
7. 对数据的临时存储,管理，协调上层数据请求。

如图示，里面的activity，presenter，model均为例子：



mvp
将复杂的功能分割为各层内的小问题。各层内功能单一。这样易于功能修改拓展与Debug。
解耦的设计，独立的模块，更有利于分工开发与测试。

Activity的异常重启

Activity会在很多情况下被系统重启：

6. 当用户旋转屏幕
7. 在后台时内存不足
8. 改变语言设置
9. attache 一个外部显示器等。

正确的方式应该是：
Presenter与Activity的绑定关系应由静态类管理。而不是由Activity管理。当Activity意外重启时Presenter不应重启。Activity重启时，Presenter与Activity重新绑定，根据数据恢复Activity状态。

而当Activity真正销毁时。对应Presenter才应该跟随销毁。

这样处理可以解决以下2个很实际的问题:

4. 不会每次翻转屏幕都去显示进度条，重新加载数据。
5. 某些低端机，调用系统拍照时内存不足，于是销毁后台Activity。于是接受拍照返回的Activity也销毁了。数据丢失。

生命周期

Activity是一个上帝类，其实不适合作为View。所以有些MVP方案将Activity作为Presenter。最主要在于他的生命周期牵扯太多逻辑处理业务。这些由Presenter负责的话情况可以改善很多。我建议将在顶级父类中将activity的生命周期在Presenter中实现一遍，然后生命周期有关的业务逻辑直接由Presenter来实现。

Model的初始化

Model不仅仅是javabean。Model是负责提供各类数据模型。在此基础上我将Model拓展为数据层提供数据交互。将javabean单独为数据层的一部分。Model层的各个Model一般使用单例。这样的好处在于这个唯一对象可以管理一些数据供所有上层使用。

Model的单例对象

```
public class UserModel extends AbsModel{
public static UserModel getInstance() {
return getInstance(UserModel.class);
}
@Override
protected void onAppCreate(Context ctx) {
super.onAppCreate(ctx);
//初始化
}
public void login(String number,String password,DataCallback<UserDetail> callback){
//进行登录请求与回调，并保存返回账号
}
public void register(String tel,String password,String code,int gender,String nickname,StatusCallback callback){
//进行注册请求与回调
}
public void findPassword(String number,String code,String password,DataCallback<User> callback){
//进行找回密码请求与回调
}
public void certification(String number,String school,String realName,String stuCard,DataCallback<User> callback){
//进行认证请求与回调
}
public void LoginOut(){
//登出操作
}
}
```

既然Model层管理数据，并且是单例。他就有初始化的需求，比如在APP启动时就请求数据，记录信息，开始一个后台线程与服务器同步信息等。这些操作与Presenter无关。是数据层自发的的功能。所以需要在Application启动时进行Model的初始化。
但又要注意不能在Application的onCreate()进行过多操作，否则会启动时间过长。所以可考虑在启动时创建一个后台线程，将即时性不强的初始化操作放到后台线程。

Adapter的处理

对于Adapter是放在View好还是Presenter好，这个问题确实难以解决。但在使用解耦的ViewHolder后这个问题便很明了。视图的创建与改变全由ViewHolder管理。然后Adapter仅仅处理面向ViewHolder的逻辑。
然后ViewHolder属于View，Adapter属于Presenter。参考EasyRecyclerView
Adapter:

```
public class PersonAdapter extends RecyclerArrayAdapter<Person> {
public PersonAdapter(Context context) {
super(context);
}
@Override
public BaseViewHolder OnCreateViewHolder(ViewGroup parent, int viewType) {
```

```
return new PersonViewHolder(parent);
}
}
```

ViewHolder:

```
public class PersonViewHolder extends BaseViewHolder<Person> {
private TextView mTv_name;
private SimpleDraweeView mImg_face;
private TextView mTv_sign;
public PersonViewHolder(ViewGroup parent) {
super(parent,R.layout.item_person);
mTv_name = $(R.id.person_name);
mTv_sign = $(R.id.person_sign);
mImg_face = $(R.id.person_face);
}
@Override
public void setData(final Person person){
mTv_name.setText(person.getName());
mTv_sign.setText(person.getSign());
mImg_face.setImageURI(Uri.parse(person.getFace()));
}
}
```

Rx的参与

Rx订阅发布模式在MVP中作用很大。可以极大简化层间通讯的处理。View向Presenter订阅数据。Presenter可以向Model层订阅数据。形成一个数据链。数据可以直接链式到达View层。优雅易拓展。
Presenter

```
public class QuestionShowPresenter extends BeamDataActivityPresenter<QuestionShowActivity,Question> {
@Override
protected void onCreate(QuestionShowActivity view, Bundle savedState) {
super.onCreate(view, savedState);
QuestionModel.getInstance().getQuestion(1).subscribe(this);
}
}
```

View

```
public class QuestionShowActivity extends BeamDataActivity<QuestionShowPresenter,Question> {
@Override
public void setData(Question data) {
//显示数据
}
@Override
public void setError(Throwable e) {
//显示错误
}
}
```

Beam

Beam是我做的一套基于MVP模式的快速开发框架。参考了nucleus。上面的示例代码都是使用了这个(为方便复制的这个框架demo代码.= =)。定义了一套开发规范。并提供了基于这套规范的Activity，Fragment，Presenter，Model等父类及控件和API等，完成APP开发过程中大量繁琐工作。并进行了一系列优化。详情看这里https://github.com/Jude95/Beam

> nucleus: https://github.com/konmik/nucleus

示例

> 豆逼: https://github.com/Jude95/Joy