ctrip tech 携程技术中心

ctrip tech 携程技术中心 | IT大咖说 知识分享平台

携程技术沙龙

MVP模式在酒店业务的应用和扩展

赵伟麟

赵伟麟

- 09年开始从事Android APP和OS研发和架构，擅长业务架构，建模，性能优化

目录
CONTENTS

模型

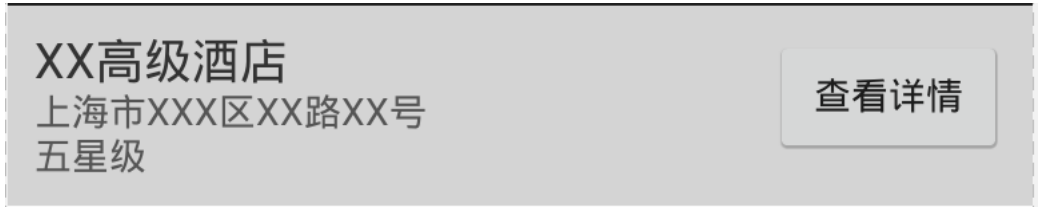
- public class HotelModel {

```
    //酒店名称
    public String hotelName;

    //酒店地址
    public String hotelAdress;

    //酒店星级
    public String hotelStar;

    //是否展示详情按钮
    public boolean showDetail;
}
```



Android MVC

```
public class HotelActivity extends Activity {

    private TextView mNameView;
    private TextView mAddressView;
    private TextView mStarView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        mNameView = (TextView) findViewById(R.id.hotel_view);
        mAddressView = (TextView) findViewById(R.id.address_view);
        mStarView = (TextView) findViewById(R.id.star_view);

        HotelModel hotel = HotelLoader.loadHotelById(1000);

        mHotelNameView.setText(hotel.hotelName);
        mHotelAddressView.setText(hotel.hotelAdress);
        mHotelStarView.setText(hotel.hotelStar);
    }
}
```

控制器C：HotelActivity

模型M：HotelModel，HotelLoader

视图V：mHotelNameView
mHotelAddressView
mHotelStarView

问题：

1、控制器管理得太多了，变得非常臃肿

2、封装性不高，维护成本高

真正意义上的 MVC

```
public class HotelActivity extends Activity {

    private HotelView mHotelView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        mHotelView = (HotelView) findViewById(R.id.hotel_view);
        HotelModel hotel = HotelLoader.loadHotelById(1000);
        mHotelView.setHotel(hotel);
    }
}
```

控制器C：HotelActivity

模型M：HotelModel，HotelLoader

视图V：HotelView

三个特点：

1、数据都Model进行封装

2、View绑定业务实体，view.setXXX

3、Controller不管理业务无关的View

一个相对复杂的界面

```
public class HotelInquireActivity extends Activity {

    private CTTitleBar mTitleBar;
    private CTAdView mAdView;
    private CTFormView mFomView;
    private CTEntranceView mEntranceView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        mTitleBar =(CTTitleBar) findViewById(R.id.title_bar);
        mAdView =(CTAdView) findViewById(R.id. ad_view);
        mFomView = (CTFormView)findViewById(R.id.form_view);
        mEntranceView = (CTEntranceView)findViewById(R.id.entrance_view);
        ……//调用xxxVIew.setXXXModel方法
    }
}
```

MVC的烦恼

方案一：继承ViewGroup，根据业务初始化子控件并添加到UI层级上去

方案二：继承ViewGroup，根据业务加载设计好的布局文件到UI层级上去

方案三：继承ViewGroup，定义好必须存在的子控件，布局文件直接使用自定义的控件

缺点：需要写大量的结构类似的自定义控件和布局文件，除了能够独立展示给定模型数据外，意义不大，大部分场景使用通用的layout就可以了。

典型列子：各种ListItemView

Android MVP

```
public class HotelActivity extends Activity  implements IHotelView {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        HotelModel hotel = HotelLoader.loadHotelById(1000);
        IPresenter presenter = new Presenter();
        presenter.setView(this);
        presenter.setData(hotel);
    }
    @Override
     public TextView getNameView() {

        return (TextView)findViewById(R.id.hotel_name_view);
     }

   @Override
    public TextView getAddressView() {

        return (TextView)findViewById(R.id.hotel_address_view);
    }

   @Override
    public TextView getStarView() {

        return (TextView)findViewById(R.id.hotel_address_view);
   }
}
```

```
public interface IHotelView {
    public TextView getNameView();
    public TextView getAddressView();
    public TextView getStarView();
}
```

```
public interface IHotelPresenter {
    public void setView(IHotelView hotelView);
    public void setData(HotelMotel hotel);
}
```

```
public class HotelPresenter implements IHotelPresenter {
    private IHotelView hotelView;
    public void setView(IHotelView hotelView) {
          this.hotelView  = hotelView;
    }
    public void setData(HotelModel hotel) {
      hotelView.getNameView().setText(hotel.hotelName);
      hotelView.getAddressView().setText(hotel.hotelAddress);
      hotelView.getStarView().setText(hotel.hotelStart);
    }
}
```

Android MVP

1、面向接口

2、View 不再依赖 Model，即View不再需要setData方法

3、Presenter 实际上是将原来View的setData方法抽离出来，专门负责绑定数据

4、Activity充当View，弱化Controller角色，业务分散到各自的Presenter中去完成

这很好

甚至是危险

有时也很别扭

Android MVP

1、业务复杂时，可能使得Activity变成更加复杂，比如要实现N个IView，然后写更多个模版方法

2、业务复杂时，各个角色之间通信会变得很冗长和复杂，回调链过长

3、Presenter处理业务，让业务变得很分散，不能全局掌握业务，很难去回答某个业务究竟是在哪里处理的

4、用Presenter替代Controller是一个危险的做法，可能出现内存泄漏，生命周期不同步，上下文丢失等问题

场景一

详情按钮的展示需要服务端下发标记位控制，展示时点击需要请求一个服务，服务返回时toast提示用户

```
public class HotelPresenter {
    private IHotelView mHotelView;
    private Handler handler = new  Handler(getMainLooper());
    public void setData(HotelModel hotelModel) {
        View button = mHotelView.getButtonView();
        int visibility = hotelModel.showButton ? .VISIBLE :GONE;
        button.setVisibility(visibility);
        if (hotelModel.showButton) {
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    sendRequest();
                }
            });
        }
    }
```

```
private void sendRequest() {
        new Thread() {
            public void run() {

                Thread.sleep(15*1000);

                handler.post(new Runnable() {
                    public void run() {
                        Toast.makeText(???)
                    }
                });
            }
        }.start();
    }
}
```

场景二

详情按钮的展示需要服务端下发标记位控制，展示是点击需要请求一个服务，服务返回时toast提示用户

```
public class HotelPresenter {
    private IHotelView mHotelView;
    private Fragment mFragment;
    private HotelPresenter(Fragment fragment) {
        this.mFragment = fragment;
    }
    private Handler handler = new Handler(Looper.getMainLooper());
    public void setData(HotelModel hotelModel) {
        View button = mHotelView.getButtonView();
        button.setVisibility(hotelModel.showButton ? VISIBLE :GONE);
        if (hotelModel.showButton) {
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    sendRequest();
                }
            }); }
    }
}
```

```
private void sendRequest() {
        new Thread() {
            public void run() {

                Thread.sleep(15*1000);

                handler.post(new Runnable() {
                    public void run() {
                        Context context = mFragment.getActivity();
                        int duration = LENGTH_SHORT;
                        Toast.makeText(context,"成功",duration).show();
                    }
                });
            }
        }.start();
    }
```

场景三

详情按钮的展示需要服务端下发标记位控制，展示是点击需要请求一个服务，服务返回时toast提示用户

```
public class HotelPresenter {
      private IHotelView mHotelView;
      private Handler handler = new Handler(Looper.getMainLooper());

      public void setData(HotelModel hotelModel) {
       View button = mHotelView.getButtonView();
       button.setVisibility(hotelModel.showButton ? VISIBLE :GONE);

       if (hotelModel.showButton) {
          button.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View v) {
                if (mCallback != null) {
                   mCallback.onSendButtonClicked();
                }
          });  }
   }
}
```

```
 public interface Callback {
         public void onSendButtonClicked();
}

private Callback mCallback;
public  void setCallback(Callback  callback) {
       mCallack = callback;
}
```

携程酒店MVP初探

ListAdapter过于庞大

```
public View getView(...) {
    ViewHolder holder;
    if (convertView !=null) {
        holder = (ViewHolder)convertView.getTag();
    } else {
        convertView = newView();
        holder = new ViewHolder();
        holder.xxxView = findViewByID(...);
        convertView.setTag(holder);
    }
    Model model = getItem(position);
    holder.xxxView.setText(model.xxx);
     .......
    return convertView;
}
```


```
public View getView(...) {
    ViewHolder holder;
    if (convertView !=null) {
        holder = (ViewHolder)convertView.getTag();
    } else {
        convertView = newView();
        holder = new ViewHolder();
        holder.setView(convertView);
    }
    Model model = getItem(position);
    holder.bindData(model);
    return convertView;
}
```

携程酒店MVP初探

1、ListAdapter变得非常简洁

2、Holder变成可重用的组件，等价于Presenter

3、发现这种思想不仅适合ListAdapter，整个页面都适用，Activity不但可以大幅瘦身，还提升了重用性

4、很欣慰后面google推出的RecyclerView的Adapter直接暴露Holder，采用相同数据绑定方案

携程酒店MVP详解

1、如何定义View接口? 弱类型View接口

2、如何定位Presenter？ 仅负责绑定数据到View，业务就不要去管了

3、如何对待Controller？ 仍然是核心

4、如何解决长长的回调链? 引入交互模型Interactor

MVCPI

携程酒店MVP详解-View

```
public interface IView {

    //用于展示酒店名称的控件
    int NAME_VIEW = R.id.name_view;
    //用于展示酒店地址的控件
    int ADDRESS_VIEW = R.id.address_view;
    //用于展示酒店星级的控件
    int STAR_VIEW = R.id.star_view;
    //用于展示酒店详情入口的的控件
    int DETAIL_BUTTON = R.id.detail_button;

}
```

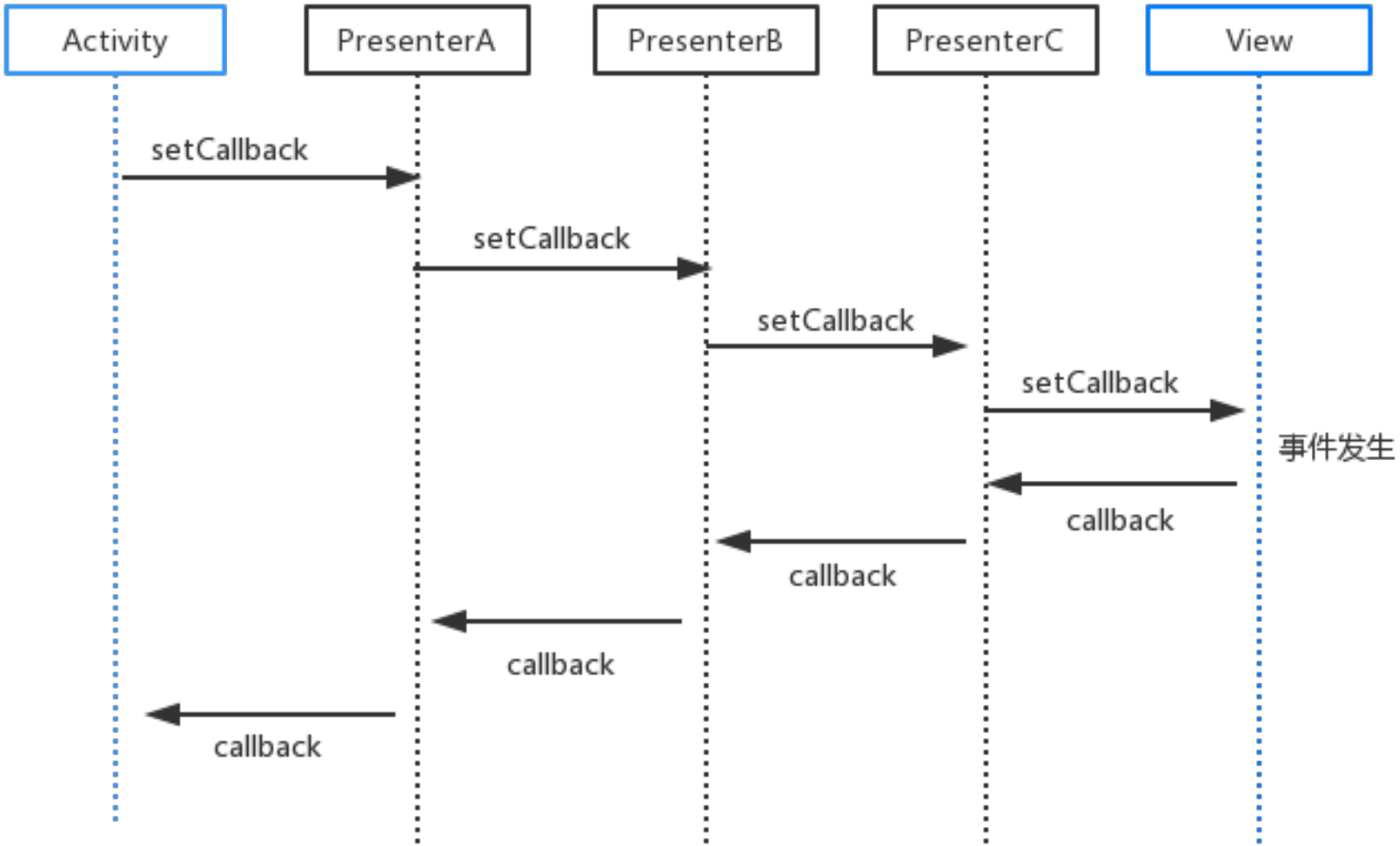
每个Presenter自己定义这样的接口，仅仅约定，View在layout中需要声明这些ID的控件，不需要任何实现

携程酒店MVP详解-Presenter

```
public class CtripHotelPresenter {
    TextView mNameView;
    TextView mAddressView;
    TextView mStarView;
    Button mDetailButton;
    public void setView(View view) {
        mNameView = (TextView) mView.findViewById(IView.NAME_VIEW);
        mAddressView = (TextView) mView.findViewById(IView.ADDRESS_VIEW);
        mStarView = (TextView) mView.findViewById(IView.STAR_VIEW);
        mDetailButton = (Button) mView.findViewById(IView.DETAIL_BUTTON);
    }
    public void setData(HotelModel hotel) {
        mNameView.setText(hotel.hotelName);
        mAddressView.setText(hotel.hotelAdress);
        mStarView.setText(hotel.hotelStar);
        int v = hotel.showButton ? View.VISIBLE : View.GONE;
        mDetailButton.setVisibility(v);
    }
```

弱类型，只要是View都可以

携程酒店MVP详解-Interactor



携程酒店MVP详解-Interactor

1、几十个交互需求

2、分布于不同的模块

3、分布于不同深度的嵌套层次中

某天，产品经理找你聊一下该界面有哪些交互，是怎么处理的？

携程酒店MVP详解-Interactor

1、专门用于描述交互的模型，可以通过Interactor知晓所有的交互

2、作为页面的重要组件而存在，在多个组件中共享，从而消除各种自定义接口和回调链

3、事件处理和事件分离，降低耦合，强化控制器的作用

```
public class HotelOrderDetailListeners {

    public View.OnClickListener mBackListener; // 返回按钮点击事件监听者

    public View.OnClickListener mShareClickListener;//分享按钮事件监听者

    public View.OnClickListener mConsultClickListener;//咨询按钮事件监听者
    ……
}
```

携程酒店MVP详解

```
public class HotelInteractor {
    //点击详情的事件处理器
    public View.OnClickListener mDetail;
}
```

```
public class HotelActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        HotelInteractor interactor = new HotelInteractor();

        interactor.mDetail = new View.OnClickListener() {
            public void onClick(View view) {
                viewHotelDetail();//处理详情业务;
            }
        };
        HotelModel model= HotelLoader.loadHotelById(1000);
        HotelPresenter presenter = new HotelPresenter (interactor);
        View view= findViewById(R.id.hotel_view);
        presenter.setView(view);
        presenter.setData(hotel);
    }
}
```

```
public class HotelPresenter {
    private View hotelView;
    private HotelInteractor mInteractor;
    private Button mDetailButton;

    public HotelPresenter(HotelInteractor interactor) {
        this.mInteractor = interactor;
    }

    private interface IView {
        int DETAIL= R.id.detail_button;
        ……
    }
    public void setView(View hotelView) {
        this.hotelView  = hotelView;
        mDetailButton= (Button)findViewById(IView. DETAIL );
     }
    public void setData(HotelModel hotel) {
        if (hotel.showButton) {
            mDetailButton.setVisibility(View.Visibile);
            mDetailButton.setOnClickListener(mInteractor. mDetail);
        }
     }
}
```

携程酒店框架模式发展规划

1、进一步完善和推广MVCPI，优化各个元素的定位，识别新元素

2、探索适合不同页面属性的新的框架模式

3、从 框架模式 演进到 框架，推出应用上层框架

ctrip tech 携程技术中心

ctrip tech 携程技术中心 | IT大咖说 知识分享平台

THANK YOU!

Q&A