Android 中的 MVP 框架

A Fool

2019

目录

1 MVP

1.1 MVP 概述

MVP 全称：Model-View-Presenter ；MVP 是从经典的模式 MVC 演变而来，它们的基本思想有相通的地方：Controller/Presenter 负责逻辑的处理，Model 提供数据，View 负责显示。

- Model 定义用户界面所需要被显示的数据模型，一个模型包含着相关的业务逻辑。
- View 视图为呈现用户界面的终端，用以表现来自 Model 的数据，和用户命令路由再经过 Presenter 对事件处理后的数据。

- Presenter 包含着组件的事件处理，负责检索 Model 获取数据，和将获取的数据经过格式转换与 View 进行沟通。

MVP 从 MVC 演变而来，通过表示器将视图与模型巧妙地分开。在该模式中，视图通常由表示器初始化，它呈现用户界面（UI）并接受用户所发出命令，但不对用户的输入作任何逻辑处理，而仅仅是将用户输入转发给表示器。通常每一个视图对应一个表示器，但是也可能一个拥有较复杂业务逻辑的视图会对应多个表示器，每个表示器完成该视图的一部分业务处理工作，降低了单个表示器的复杂程度，一个表示器也能被多个有着相同业务需求的视图复用，增加单个表示器的复用度。表示器包含大多数表示逻辑，用以处理视图，与模型交互以获取或更新数据等。模型描述了系统的处理逻辑，模型对于表示器和视图一无所知。

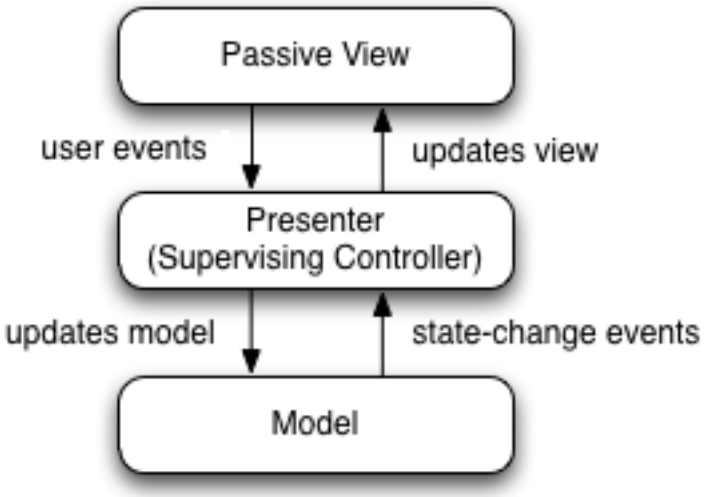

图 1: MVP 原理图

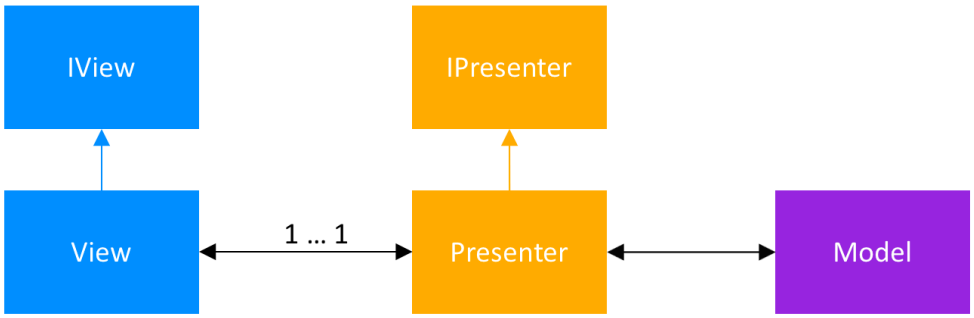

图 2: Model-View-Presenter class structure

1.2 优缺点

优点：

1. View 与 Model 完全隔离，如果 Model 或 View 中的一方发生变化，只要交互接口不变，另一方就没必要对上述变化做出改变。这使得 Model 层的业务逻辑具有很好的灵活性和可重用性。

2. Presenter 与 View 的具体实现技术无关，采用诸如 Windows 表单、WPF、Web 表单等用户界面构建技术中的任意一种来实现 View 层，都无需改变系统的其他部分。甚至为了使 B/S，C/S 部署架构能够被同时支持，应用程序可以用同一个 Model 层适配多种技术构建的 View 层。

3. 可以进行 View 的模拟测试，在 MVP 模式中，View 和 Model 之间没有直接依赖，开发者能够借助模拟对象注入测试两者中的任一方。

4. 视图的变化总是比较频繁，将业务逻辑抽取出来，放在表示器中实现，使模块职责划分明显，层次清晰，一个表示器能复用于多个视图，而不需要更改表示器的逻辑，这增加了程序的复用性。

5. 数据的处理由模型层完成，隐藏了数据，在数据显示时，表示器可以对数据进行访问控制，提高数据的安全性。

缺点：

增加了代码的复杂度，特别是针对小型 Android 应用的开发，会使程序冗余。

1. Presenter 中除了应用逻辑以外，还有大量的 View->Model，Model->View 的手动同步逻辑，会导致 Presenter 臃肿，维护困难。

2. 视图的渲染过程也会放在 Presenter 中，造成视图与 Presenter 交互过于频繁，如果某特定视图的渲染很多，就会造成 Presenter 与该视图联系过于紧密，一旦该视图需要变更，那么 Presenter 也需要变更了，不能如预期的那样降低耦合度和增加复用性。

注：

- 如果要实现的 UI 比较复杂，而且相关的显示逻辑还跟 Model 有关系，就可以在 View 和 Presenter 之间放置一个 Adapter。由这个 Adapter 来访问 Model 和 View，避免两者之间的关联。而同时，因为 Adapter 实现了 View 的接口，从而可以保证与 Presenter 之间接口的不变。这样就可以保证 View 和 Presenter 之间接口的简洁，又不失去 UI 的灵活性。

- 在 MVP 模式里，View 只应该有简单的 Set/Get 的方法，用户输入和设置界面显示的内容，除此就不应该有更多的内容，绝不容许直接访问 Model–这就是与 MVC 很大的不同之处。

为什么 MVP 模式利于单元测试?

Presenter 将逻辑和 UI 分开了，里面没有 Android 代码，都是纯纯的 java 代码。我们可以直接对 Presenter 写 Junit 测试。

2 Android 中的 MVP

2.1 标准 MVP 模式

- BaseActivity: 提供 Activity 的抽象类；
- BasePresenter: Presenter 的接口；
- BookBean: 实体 Bean；
- BookCallBack: 返回 CallBack 接口，传递返回结果；
- BuyBookActivity: View 层，负责数据显示；
- BuyBookAdapter: ListView 适配器；
- BuyBookContract: 包含 View 和 Presenter 更接近本层功能的接口；
- BuyBookModel: Model 层，负责网络请求；
- BuyBookPresenter: Presenter 层，负责处理 View 和 Model 之间的交互。

```
1  //BaseActivity
2  public abstract class BaseActivity<T extends BasePresenter> extends
       Activity {
3      protected T basepresenter;
4      @Override
5      protected void onCreate(@Nullable Bundle savedInstanceState) {
6          super.onCreate(savedInstanceState);
7          setContentView(getLayout());
8          initView();
9          basepresenter = initPresent();
10         onPrepare();
11     }
12     abstract T initPresent();
13     abstract int getLayout();
14     abstract void initView();
15     abstract void onPrepare();
16 }
```

```
//BasePresenter
public abstract class BasePresenter<T extends BaseActivity> {
    abstract void initData();
}
```

```
//BookBean
public class BookBean {
    private String name;
    private int number;
    private String time;
    //setter getter 构造器
}
```

```
//BookCallBack
public interface BookCallBack<T> {
    void onSuccess(T t);
    void onFail(String code);
}
```

```
//BuyBookActivity
public class BuyBookActivity extends BaseActivity<BuyBookPresenter>
    implements BuyBookContract.IBuyBookView {
    private ListView listView;
    private BuyBookAdapter adapter;
    @Override
    BuyBookPresenter initPresent() {
        return new BuyBookPresenter(this);
    }
    @Override
    int getLayout() {
        return R.layout.activity_buy;
    }
```

```
13      @Override
14      void initView() {
15          listView = findViewById(R.id.list_view);
16      }
17      @Override
18      void onPrepare() {
19          adapter = new BuyBookAdapter(basepresenter.getAdapterData()
                , this);
20          listView.setAdapter(adapter);
21          basepresenter.initData();
22      }
23      @Override
24      public void showToast(String msg) {
25          Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
26      }
27      @Override
28      public void refreshAdapter() {
29          adapter.notifyDataSetChanged();
30      }
31      @Override
32      public void onEmpty() {
33          listView.setEmptyView(null);
34      }
35  }
```

```
1   //BuyBookAdapter
2   public class BuyBookAdapter extends BaseAdapter{
3       private List<BookBean> list;
4       private Context context;
5       public BuyBookAdapter(List<BookBean> list, Context context) {
6           this.list = list;
7           this.context = context;
8       }
9       @Override
10      public int getCount() {
```

```
11          return list == null ? 0 : list.size();
12      }
13      @Override
14      public Object getItem(int position) {
15          return list == null ? null : list.get(position);
16      }
17      @Override
18      public long getItemId(int position) {
19          return position;
20      }
21      @Override
22      public View getView(int position, View convertView, ViewGroup
           parent) {
23          MyHolder myHolder = null;
24          if (convertView == null) {
25              convertView = View.inflate(context, R.layout.lst_item,
                   null);
26              myHolder = new MyHolder(convertView);
27              convertView.setTag(myHolder);
28          } else {
29              myHolder = (MyHolder) convertView.getTag();
30          }
31          myHolder.name.setText(list.get(position).getName());
32          myHolder.number.setText("(" + list.get(position).getNumber
               () + "人)");
33          myHolder.time.setText(list.get(position).getTime());
34          return convertView;
35      }
36      class MyHolder {
37          TextView name;
38          TextView number;
39          TextView time;
40          public MyHolder(View v) {
41              name = v.findViewById(R.id.name);
42              number = v.findViewById(R.id.number);
43              time = v.findViewById(R.id.time);
```

```
44          }
45      }
46  }
```

```
1  //BuyBookContract
2  public class BuyBookContract {
3      interface IBuyBookModel {
4          void getTestData(BookCallBack<List<BookBean>> callBack);
5          List<BookBean> getAdapterData();
6      }
7
8      interface IBuyBookPresenter {
9          List<BookBean> getAdapterData();
10     }
11     interface IBuyBookView {
12         void showToast(String msg);
13         void refreshAdapter();
14         void onEmpty();
15     }
16 }
```

```
1  //BuyBookModel
2  public class BuyBookModel implements BuyBookContract.IBuyBookModel
       {
3      private List<BookBean> list;
4      public BuyBookModel() {
5          this.list = new ArrayList<>();
6      }
7      @Override
8      public void getTestData(final BookCallBack<List<BookBean>>
           callBack) {
9          new Handler().postDelayed(new Runnable() {
10             @Override
11             public void run() {
```

```
12                  List<BookBean> tlist = new ArrayList<>();
13                  tlist.add(new BookBean("赵云", 1, "09-27 09:11"));
14                  tlist.add(new BookBean("赵云、隔壁老王、小王、典
                        韦、貂蝉、林芳、曹操、刘备、关羽、黄忠、张飞、诸
                        葛孔明", 10, "09-27 09:11"));
15                  tlist.add(new BookBean("黄忠、孙权、大乔", 50, "
                        09-27 09:11"));
16                  tlist.add(new BookBean("大乔、小乔、貂蝉、孙尚香",
                        300, "09-27 09:11"));
17                  Random rd = new Random();
18                  int N = rd.nextInt(10);
19                  if (N > 5) {
20                      callBack.onSuccess(tlist);
21                  } else {
22                      callBack.onFail("请求失败");
23                  }
24              }
25          }, 1000);
26      }
27
28      @Override
29      public List<BookBean> getAdapterData() {
30          return list;
31      }
32  }
```

```
1   //BuyBookPresenter
2   public class BuyBookPresenter extends BasePresenter<BuyBookActivity
        > implements BuyBookContract.IBuyBookPresenter {
3       private BuyBookContract.IBuyBookView view;
4       private BuyBookModel model;
5       public BuyBookPresenter(BuyBookContract.IBuyBookView view) {
6           this.view = view;
7           this.model = new BuyBookModel();
8       }
```

```
9       @Override
10      void initData() {
11          model.getTestData(new BookCallBack<List<BookBean>>() {
12              @Override
13              public void onSuccess(List<BookBean> bookBeans) {
14                  model.getAdapterData().addAll(bookBeans);
15                  view.refreshAdapter();
16              }
17              @Override
18              public void onFail(String code) {
19                  view.showToast(code);
20                  view.onEmpty();
21              }
22          });
23      }
24      @Override
25      public List<BookBean> getAdapterData() {
26          return model.getAdapterData();
27      }
28  }
```

2.2 MVP 常用实例二

- MainActivity：View 层，专注视图处理；
- MVPCallback：用于返回结果；
- MVPContract：封装 View、Presenter 接口；
- MVPModel：Mode 层，网络交互；
- MVPPresenter：Presenter 层，处理 View 请求，与 Model 交互。

```
1  //MainActivity
2  public class MainActivity extends AppCompatActivity implements
       MVPContract.IMVPView {
3
```

```
4       private ProgressDialog progressDialog;
5       private MVPContract.IMVPPresenter presenter;
6
7       @Override
8       protected void onCreate(Bundle savedInstanceState) {
9           super.onCreate(savedInstanceState);
10          setContentView(R.layout.activity_main);
11
12          initView();
13          setPresenter();
14      }
15      protected void initView() {
16          progressDialog = new ProgressDialog(this);
17          progressDialog.setCancelable(false);
18          progressDialog.setMessage("正在加载");
19      }
20
21      @Override
22      public void setPresenter() {
23          presenter = new MVPPresenter(this);
24      }
25
26      @Override
27      public void showLoading() {
28          if (!progressDialog.isShowing()) {
29              progressDialog.show();
30          }
31      }
32
33      @Override
34      public void hideLoading() {
35          if (progressDialog.isShowing()) {
36              progressDialog.dismiss();
37          }
38      }
39
```

```
40      @Override
41      public void showData(String data) {
42          Toast.makeText(this, data, Toast.LENGTH_SHORT).show();
43      }
44
45      @Override
46      public void showFailureMessage(String msg) {
47          Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
48      }
49
50      @Override
51      public void showErrorMessage() {
52          Toast.makeText(this, "网络请求数据出现异常", Toast.
                LENGTH_SHORT).show();
53      }
54
55      public void getData(View view){
56          presenter.getData("normal");
57      }
58
59      public void getDataForFailure(View view){
60          presenter.getData("failure");
61      }
62
63      public void getDataForError(View view){
64          presenter.getData("error");
65      }
66  }
```

```
1  //MVPCallback
2  public interface MVPCallback {
3      void onSuccess(String data);
4      void onFailure(String msg);
5      void onError();
6      void onComplete();
```

```
}
```

```
//MVPContract
public class MVPContract {
    interface IMVPView {
        void setPresenter();
        void showLoading();
        void hideLoading();
        void showData(String data);
        void showFailureMessage(String msg);
        void showErrorMessage();
    }
    interface IMVPPresenter {
        void start();
        void getData(String params);
    }
}
```

```
//MVPModel
public class MVPModel {
    public void getNetData(final String param, final MVPCallback
       callback) {
        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                switch (param) {
                case "normal":
                callback.onSuccess("请求成功");
                break;
                case "failure":
                callback.onFailure("请求失败");
                break;
                case "error":
                callback.onError();
```

```
                break;
            }
            callback.onComplete();
            }
        }, 2000);
    }
}
```

```
//MVPPresenter
public class MVPPresenter implements MVPContract.IMVPPresenter{
    private MVPContract.IMVPView view;
    private MVPModel model;

    public MVPPresenter(MVPContract.IMVPView view) {
        this.view = view;
        model = new MVPModel();
    }

    @Override
    public void start() {
    }

    public void getData(String params) {
        view.showLoading();
        model.getNetData(params, new MVPCallback() {
            @Override
            public void onSuccess(String data) {
                view.showData(data);
            }

            @Override
            public void onFailure(String msg) {
                view.showFailureMessage(msg);
            }

```

```
28              @Override
29              public void onError() {
30                  view.showErrorMessage();
31              }
32
33              @Override
34              public void onComplete() {
35                  view.hideLoading();
36              }
37          });
38      }
39  }
```

2.3 MVP 常用实例改造

实体 Bean：

```
1  //请求返回解析Gson辅助类
2  public class ResponseBean {
3      private WeatherBean weatherinfo;
4      public ResponseBean(WeatherBean weatherBean) {
5          this.weatherinfo = weatherBean;
6      }
7      public WeatherBean getWeatherBean() {
8          return weatherinfo;
9      }
10     public void setWeatherBean(WeatherBean weatherBean) {
11         this.weatherinfo = weatherBean;
12     }
13 }
14 //真正实体Bean
15 public class WeatherBean {
16     private String city;
17     private String cityid;
18     private String temp1;
19     private String temp2;
```

```
20      private String weather;
21      private String img1;
22      private String img2;
23      private String ptime;
24      // setter getter 构造器...
25  }
```

2.3.1 普通实例一

通过简单地 OkHttp 获取网络数据，Gson 解析后通过 Handler 显示到 View，这里一气呵成，将 Controller 和 View 混淆在一起，随着功能的增加，Activity 的体量会迅速上升。

```
1   public class MainActivity extends AppCompatActivity {
2       private TextView textView;
3       String temp;
4       private Handler handler = new Handler(){
5           @Override
6           public void handleMessage(Message msg) {
7               switch (msg.what) {
8                   case 1:
9                       textView.setText(temp);
10                      break;
11              }
12          }
13      };
14
15      @Override
16      protected void onCreate(Bundle savedInstanceState) {
17          super.onCreate(savedInstanceState);
18          setContentView(R.layout.activity_main);
19          textView = findViewById(R.id.text_weather);
20          getTextInfo();
21      }
22      protected void getTextInfo(){
23          OkHttpClient client = new OkHttpClient();
24          Request request = new Request.Builder()
```

```
25              .url("http://www.weather.com.cn/data/cityinfo
                    /101010100.html")
26              .build();
27          Call call = client.newCall(request);
28          call.enqueue(new Callback() {
29          @Override
30          public void onFailure(Call call, IOException e) {
31              textView.setText("失败");
32          }
33
34          @Override
35          public void onResponse(Call call, Response response) throws
                IOException {
36              Gson gson = new Gson();
37              String t = response.body().string();
38              ResponseBean responseBean = gson.fromJson(t,
                    ResponseBean.class);
39              WeatherBean weatherBean = responseBean.getWeatherBean()
                    ;
40              temp = weatherBean.getCity() + " " + weatherBean.
                    getTemp1() + " " + weatherBean.getTemp2() + " " +
                    weatherBean.getWeather();
41              Message msg = new Message();
42              msg.what = 1;
43              handler.sendMessage(msg);
44          }
45          });
46      }
47  }
```

2.3.2 简单 MVP 实现

1. 先定义一个接口 RequestView1，用来针对 View 需要做出的动作；
2. 让 Activity 实现 RequestView1，构成 View 层；
3. 再创建一个类，封装网络请求过程，与 Bean 构成 Model 层；

4. 再创建一个类，处理 Model 层和 View 层交互，构成 Presenter 层。

```
1  public interface RequestView1 {
2      //请求时展示加载
3      void requestLoading();
4      //请求成功
5      void resultSuccess(WeatherBean result);
6      //请求失败
7      void resultFailure(String result);
8  }
```

```
1  public class MainActivity extends AppCompatActivity implements
       RequestView1 {
2
3      @FieldView(R.id.tv_text)
4      private TextView textView;
5      private RequestPresenter1 presenter;
6
7      @Override
8      protected void onCreate(Bundle savedInstanceState) {
9          super.onCreate(savedInstanceState);
10         setContentView(R.layout.activity_main);
11         ViewFind.bind(this);
12
13         //创建Presenter
14         presenter = new RequestPresenter1(this);
15     }
16
17     //点击事件
18     public void request(View view) {
19         presenter.clickRequest("101010100");
20     }
21
22     //请求时加载
```

```
23      @Override
24      public void requestLoading() {
25          textView.setText("请求中,请稍后...");
26      }
27
28      //请求成功
29      @Override
30      public void resultSuccess(WeatherBean result) {
31          //成功
32          textView.setText(result.getWeatherinfo().toString());
33      }
34
35      //请求失败
36      @Override
37      public void resultFailure(String result) {
38          //失败
39          textView.setText(result);
40      }
41  }
```

```
1   public class RequestMode1 {
2
3       private static final String BASE_URL = "http://www.weather.com.
            cn/";
4
5       public void request(String detailId, Callback<WeatherBean>
            callback){
6           //请求接口
7           Retrofit retrofit  = new Retrofit.Builder()
8           //代表root地址
9           .baseUrl(BASE_URL)
10          .addConverterFactory(ScalarsConverterFactory.create())
11          .addConverterFactory(GsonConverterFactory.create())
12          .build();
13
```

```
14          ApiService apiService = retrofit.create(ApiService.class);
15
16          //请求
17          Call<WeatherBean> weatherBeanCall = apiService.
                requestWeather(detailId);
18
19          weatherBeanCall.enqueue(callback);
20      }
21  }
```

```
1   //需要View和Model层的引用
2   public class RequestPresenter1 {
3
4       private final RequestView1 mRequestView;
5       private final RequestMode1 mRequestMode;
6
7       public RequestPresenter1(RequestView1 requestView) {
8           this.mRequestView = requestView;
9           this.mRequestMode = new RequestMode1();
10      }
11
12      public void clickRequest(final String cityId){
13          //请求时显示加载
14          mRequestView.requestLoading();
15
16          //模拟耗时，可以展示出loading
17          new Handler().postDelayed(new Runnable() {
18              @Override
19              public void run() {
20                  mRequestMode.request(cityId, new Callback<
                        WeatherBean>() {
21                  @Override
22                  public void onResponse(Call<WeatherBean> call,
                        Response<WeatherBean> response) {
23                      mRequestView.resultSuccess(response.body());
```

```
24                  }
25
26                  @Override
27                  public void onFailure(Call<WeatherBean> call,
                        Throwable t) {
28                      mRequestView.resultFailure(Log.
                            getStackTraceString(t));
29                  }
30                  });
31              }
32          },1000);
33      }
34  }
```

2.3.3 改进-解决内存泄露

因为如果在网络请求的过程中 Activity 就关闭了，Presenter 还持有了 V 层的引用，也就是 MainActivity，就会内存泄露。

解决办法：将 P 层和 V 层的关联抽出两个方法，一个绑定，一个解绑，在需要的时候进行绑定 V 层，不需要的时候进行解绑就可以了。只需要修改上面 Presenter 中的构造代码，不需要在构造中传递 V 层了，然后再写一个绑定和解绑的方法，最后修改 Activity 创建 Presenter 时进行绑定，在 onDestroy 中进行解绑。

```
1   //Presenter层
2   public class RequestPresenter2 {
3
4       private RequestView2 mView;
5       private RequestMode2 mMode;
6
7       public RequestPresenter2() {
8           mMode = new RequestMode2();
9       }
10      public void clickRequest(final String cityId) {
11          //...
12      }
13      //绑定
```

```
14      public void attach( RequestView2 view) {
15          this.mView = view;
16      }
17      //解绑
18      public void detach() {
19          mView = null;
20      }
21      //取消网络请求
22      public void interruptHttp(){
23          mMode.interruptHttp();
24      }
25  }
```

```
1   //1. 是会内存泄露，因为persenter一直持有Activity，如果一个发了一个
        请求，但是网络有点慢，这个时候退出Activity，那么请求回来后还是会
        调用Activity的回调方法，这里还是因为一直持有的问题。
2   //2. 如果已经退出了当前界面，这个请求也没有用了，这个时候我们可以断
        开请求
3   /解决:
4   //1. 增加绑定和解绑的方法来解决内存泄露和退出后还会回调的问题;
5   //2. 增加断开网络连接的方法。
6   public class MainActivity extends AppCompatActivity implements
        RequestView2 {
7       //...
8       protected void onCreate(Bundle savedInstanceState) {
9           //...
10          presenter = new RequestPresenter2();
11          presenter.attach(this);
12      }
13      //...
14      protected void onDestroy() {
15          super.onDestroy();
16          presenter.detach();
17          presenter.interruptHttp();
18      }
```

```
19 }
```

仍存在问题：应用中肯定不可能只有一个模块，每个模块都对应着一个 V 层和 P 层，那这样的话每个 Presenter 中都要定义绑定和解绑的方法，而 Activity 中对应的也要调用这绑定和解绑的两个方法，代码冗余。

2.3.4 再次改进-简单抽象

1. 创建一个基类 View，让所有 View 接口都必须实现, 这个 View 可以什么都不做只是用来约束类型的；
2. 创建一个基类的 Presenter，在类上规定 View 泛型，然后定义绑定和解绑的抽象方法，让子类去实现，对外在提供一个获取 View 的方法，，让子类直接通过方法获取 View。
3. 创建一个基类的 Activity，声明一个创建 Presenter 的抽象方法，因为要帮子类去绑定和解绑那么就需要拿到子类的 Presenter 才行，但是又不能随便一个类都能绑定的，因为只有基类的 Presenter 中才定义了绑定和解绑的方法，所以同样的在类上可以声明泛型在，方法上使用泛型来达到目的。
4. 修改 Presenter 和 Activity 中的代码，各自继承自己的基类并去除重复代码。

```
1 public interface IMvpBaseView4 {
2 }
```

```
1  public abstract class AbstractMvpPersenter4<V extends IMvpBaseView4
      > {
2      private V mMvpView;
3
4      public void attachMvpView(V view){
5          this.mMvpView = view;
6      }
7
8      public void detachMvpView(){
9          mMvpView = null;
10     }
11
12     public V getmMvpView() {
```

```
        return mMvpView;
    }
}
```

```
public abstract class AbstractMvpActivity<V extends IMvpBaseView4,
    P extends AbstractMvpPersenter4<V>> extends AppCompatActivity
    implements IMvpBaseView4 {
    private P presenter;
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //创建Presenter
        if (presenter == null) {
            presenter = createPresenter();
        }

        if (presenter == null) {
            throw new NullPointerException("presenter 不能为空!");
        }
        //绑定view
        presenter.attachMvpView((V) this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        //解除绑定
        if (presenter != null) {
            presenter.detachMvpView();
        }
    }

    /**
    * 创建Presenter
```

```
30      * @return 子类自己需要的Presenter
31      */
32     protected abstract P createPresenter();
33 
34     /**
35      * 获取Presenter
36      * @return 返回子类创建的Presenter
37      */
38     public P getPresenter() {
39         return presenter;
40     }
41 }
```

```
1  public class RequestPresenter4 extends AbstractMvpPersenter4<
       RequestView4> {
2  
3      private final RequestMode4 mRequestMode;
4  
5      public RequestPresenter4() {
6          this.mRequestMode = new RequestMode4();
7      }
8  
9      public void clickRequest(final String cityId){
10     //...
11     }
12     public void interruptHttp(){
13         mRequestMode.interruptHttp();
14     }
15 }
```

```
1  public interface RequestView4  extends IMvpBaseView4{
2      void requestLoading();
3      void resultSuccess(WeatherBean result);
4      void resultFailure(String result);
```

```
5  }
6
7  public class MainActivity extends AbstractMvpActivity<RequestView4,
       RequestPresenter4> implements RequestView4 {
8      private RequestPresenter4 presenter;
9      //...
10     protected RequestPresenter4 createPresenter() {
11         return new RequestPresenter4();
12     }
13     //...
14 }
```

2.3.5 高级抽象-使用注解、工厂模式、代理模式、策略模式

http://www.androidchina.net/7961.html