|  |
| --- |
| 1、 MVC：将视图和数据分离，在WEB领域中应用的很广泛。  M对应Model，代表数据；  V对应View，代表视图；  C对应Controller，代表控制器。  流程：用户通过界面组件进行操作，也就是View层，相应的动作会传递给控制器也就是Controller层，而Controller根据自己的业务逻辑去操作数据层也就是Model，而最终数据层的变化会同步更新到视图层。  好处：分层思想，解耦；模块划分，分组开发，方便测试，高可扩展性。  2、MVC在Android中的问题  Activity中的layout.xml布局文件对应视图界面也就是View层。 数据库文件，Sharedprefrence,内存缓冲，磁盘缓冲等数据内容对应Model层。 而Controller控制层基本上也由Activity层面来进行。因为Android的特殊性，使得Activity对应了MVC中的V和C,同时担任两个角色，这显然就不符合软件设计原则的“单一职责”原则。随着项目的深入发展，很多逻辑很越来越复杂，Activity处理的东西也会越来越多，代码越来越臃肿。  3、MVP  View 对应于Activity，负责布局的绘制以及与用户交互；  Model 依然是业务逻辑和实体模型；  Presenter 负责完成View于Model间的交互、连接；  MVP中Presenter取代了MVC中的Controller。  MVC中Model、View、Controller之间相互发生通信，而MVP中Model与Presenter相互通信，View与Presenter相互通信，而Model与View之间没有通信。  MVP的出现实际上就是将MVC进行升级，对应Android开发中就是帮助Activity解压。  4、MVP开发在Android中的基本流程 ，遵循面向接口编程。  1） View层定义View.interface，用来定义View的行为。一般由Activity或者是Fragment来实现这个接口，它定义了View视图的各种变化，如设置Textview,加载对话框，更新进度条等。  2） Model层定义Modle.interface,这个是用来定义数据层发生变化时的通知接口。  3）Presenter翻译的意思是主持人的意思。在这时Presenter就负责具体的业务逻辑，把请求数据送到Model，或者监听Model的数据变化，负责通过通知View层的视图变化。 |

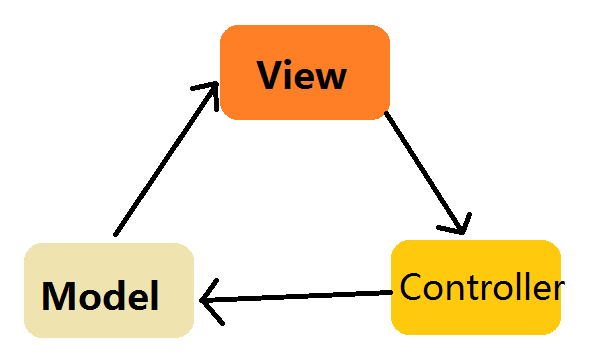
# 1、MVC与MVP

## 1.1 MVC

M对应Model，代表业务数据；

V对应View，代表视图；

C对应Controller，代表控制器。



（1）MVC架构将视图和数据分离，在WEB领域中应用的很广泛。   
 用户通过界面组件进行操作，也就是View层，相应的动作会传递给控制器也就是Controller层，而Controller根据自己的业务逻辑去操作数据层也就是Model，而最终数据层的变化会同步更新到视图层。

（2）MVC好处

**分层思想，进行解耦；便于对软件进行模块划分，方便分组开发，方便测试，提高可扩展性，提高了额编码速度与质量。分工合作容易，划分的更小，容易协调，相互之间的影响小了。**

MVC 分层有助于管理复杂的应用程序，因为您可以在一个时间内专门关注一个方面。例如，您可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。

MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

可以看到MVC的主要目的是为了视图和数据分离，这对于开发大型软件来说更方便进行模块的划分，提高编码速度与质量。

（3）**Android中的MVC**

Android世界中也经常运用到MVC模式。

Activity中的layout.xml布局文件对应视图界面也就是View层。

数据库文件，Sharedprefrence,内存缓冲，磁盘缓冲等数据内容对应Model层。

而Controller控制层基本上也由Activity层面来进行。（导致C、V都集中到了Activity中了。）

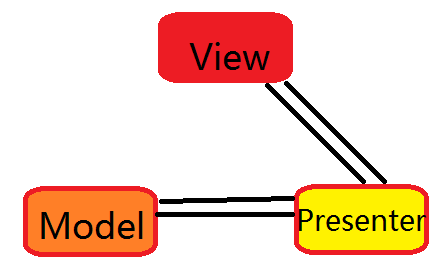
## 1.2 MVP

上面讲解了MVC的基础知识，大家可能觉得MVC挺好的啊？怎么还要整一个MVP。是的MVC是挺好的，但是它也有它的缺点，特别是针对Android开发。

**因为Android的特殊性，使得Activity对应了MVC中的V和C,同时担任两个角色，就有了类似“既当爹又当妈”的感觉，这显然就不符合软件设计原则的“单一职责”原则。**但现实中是很多的APP代码中有这么的处境，特别是Androi原生的很多系统APK，某些Activity动则几千行代码。

况且，随着项目的深入发展，很多逻辑很越来越复杂，Activity处理的东西也会越来越多，代码越来越臃肿。**这样一来维护起来的代价就会越来越高，**这是因为View的变化会引起Controller的很多变化，反之亦然。用一句大白话来说明就是–某一段代码的变动会引起很多其他相关联的代码的改动，而程序员都是懒惰的，所以会恨死这样的代码。

而MVP就是要减轻在Android中的这种困惑。 MVP是基于MVC的，它的架构图如下：



M(Model) 数据相关层，

V(View) 视图层，如Activity上的布局，

P(Presenter) 纽带层，用来连接Model与View。

如果跟MVC的架构图对比的话，可以发现它们有相似之处也有不同。

相似之处

模块划分的相似

MVC由Model、View、Controller构成。

MVP由Model、View、Presenter构成。

不同的地方

MVP中Presenter取代了MVC中的Controller

MVC中Model、View、Controller之间相互发生通信，而MVP中Model与Presenter相互通信，View与Presenter相互通信，而Model与View之间没有通信。

Android中MVP的好处？就Android层面上来讲MVC架构虽然好，但不是最好，情况前面有讲过。用一句话概括就是“模块界限很模糊”。而MVP的出现实际上就是将MVC进行升级，对应Android开发中就是帮助Activity解压。

MVC中Activity同时充当了V和C的角色，这就属于界限划分不清楚。而MVP则划分的很清楚，Activity只充当V的角色，业务逻辑控制交给了Presenter。

## 1.3 MVP开发在Android中的基本流程

1. View层定义View.interface，用来定义View的行为。一般由Activity或者是Fragment来实现这个接口，它定义了View视图的各种变化，如设置Textview,加载对话框，更新进度条等。

2. Model层定义Modle.interface,这个是用来定义数据层发生变化时的通知接口，因为Model不能直接与View交互，所以它与Presenter交互，然后再通过Presenter间接达到与View的交互。

3. Presenter翻译的意思是主持人，也就是主持场合，控制节奏的意思。在这时Presenter就负责具体的业务逻辑，请求数据，把数据送到Model，或者监听Model的数据变化，接受View层的动作，负责通过通知View层的视图变化。

当将架构改为MVP以后，Presenter的出现，将Actvity视为View层，Presenter负责完成View层与Model层的交互。现在是这样的：

View 对应于Activity，负责View的绘制以及与用户交互

Model 依然是业务逻辑和实体模型

Presenter 负责完成View于Model间的交互

---------------------

# 2、Simple Login Demo

## 2.1 Model

首先实体类User不用考虑这个肯定有，其次从效果图可以看到至少有一个业务方法login()，这两点没什么难度，我们首先完成：

package com.zhy.blogcodes.mvp.bean;

/\*\*

\* Created by zhy on 15/6/18.

\*/

public class User

{

private String username ;

private String password ;

public String getUsername()

{

return username;

}

public void setUsername(String username)

{

this.username = username;

}

public String getPassword()

{

return password;

}

public void setPassword(String password)

{

this.password = password;

}

}

package com.zhy.blogcodes.mvp.biz;

/\*\*

\* Created by zhy on 15/6/19.

\*/

public interface IUserBiz

{

public void login(String username, String password, OnLoginListener loginListener);

}

package com.zhy.blogcodes.mvp.biz;

import com.zhy.blogcodes.mvp.bean.User;

/\*\*

\* Created by zhy on 15/6/19.

\*/

public class UserBiz implements IUserBiz

{

@Override

public void login(final String username, final String password, final OnLoginListener loginListener)

{

//模拟子线程耗时操作

new Thread()

{

@Override

public void run()

{

try

{

Thread.sleep(2000);

} catch (InterruptedException e)

{

e.printStackTrace();

}

//模拟登录成功

if ("zhy".equals(username) && "123".equals(password))

{

User user = new User();

user.setUsername(username);

user.setPassword(password);

loginListener.loginSuccess(user);

} else

{

loginListener.loginFailed();

}

}

}.start();

}

}

package com.zhy.blogcodes.mvp.biz;

import com.zhy.blogcodes.mvp.bean.User;

/\*\*

\* Created by zhy on 15/6/19.

\*/

public interface OnLoginListener

{

void loginSuccess(User user);

void loginFailed();

}

实体类不用说，至于业务类，我们抽取了一个接口，一个实现类这也很常见~~login方法，一般肯定是连接服务器的，是个耗时操作，所以我们开辟了子线程，Thread.sleep(2000)模拟了耗时，由于是耗时操作，所以我们通过一个回调接口来通知登录的状态。

其实这里还是比较好写的，因为和传统写法没区别。

## 2.2 View

上面我们说过，Presenter与View交互是通过接口。所以我们这里需要定义一个ILoginView，难点就在于应该有哪些方法，我们看一眼效果图：

可以看到我们有两个按钮，一个是login，一个是clear；

login说明了要有用户名、密码，那么对应两个方法：

String getUserName();

String getPassword();

再者login是个耗时操作，我们需要给用户一个友好的提示，一般就是操作ProgressBar，所以再两个：

void showLoading();

void hideLoading();

login当然存在登录成功与失败的处理，我们主要看成功我们是跳转Activity，而失败可能是去给个提醒：

void toMainActivity(User user);

void showFailedError();

ok，login这个方法我们分析完了~~还剩个clear那就简单了：

void clearUserName();

void clearPassword();

综上，接口完整为：

package com.zhy.blogcodes.mvp.view;

import com.zhy.blogcodes.mvp.bean.User;

/\*\*

\* Created by zhy on 15/6/19.

\*/

public interface IUserLoginView

{

String getUserName();

String getPassword();

void clearUserName();

void clearPassword();

void showLoading();

void hideLoading();

void toMainActivity(User user);

void showFailedError();

}

有了接口，实现就太好写了~~~

总结下，对于View的接口，去观察功能上的操作，然后考虑：

该操作需要什么？（getUserName, getPassword）

该操作的结果，对应的反馈？(toMainActivity, showFailedError)

该操作过程中对应的友好的交互？(showLoading, hideLoading)

下面贴一下我们的View的实现类，哈，其实就是Activity，文章开始就说过，MVP中的View其实就是Activity。

package com.zhy.blogcodes.mvp;

import android.os.Bundle;

import android.support.v7.app.ActionBarActivity;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ProgressBar;

import android.widget.Toast;

import com.zhy.blogcodes.R;

import com.zhy.blogcodes.mvp.bean.User;

import com.zhy.blogcodes.mvp.presenter.UserLoginPresenter;

import com.zhy.blogcodes.mvp.view.IUserLoginView;

public class UserLoginActivity extends ActionBarActivity implements IUserLoginView

{

private EditText mEtUsername, mEtPassword;

private Button mBtnLogin, mBtnClear;

private ProgressBar mPbLoading;

private UserLoginPresenter mUserLoginPresenter = new UserLoginPresenter(this);

@Override

protected void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_user\_login);

initViews();

}

private void initViews()

{

mEtUsername = (EditText) findViewById(R.id.id\_et\_username);

mEtPassword = (EditText) findViewById(R.id.id\_et\_password);

mBtnClear = (Button) findViewById(R.id.id\_btn\_clear);

mBtnLogin = (Button) findViewById(R.id.id\_btn\_login);

mPbLoading = (ProgressBar) findViewById(R.id.id\_pb\_loading);

mBtnLogin.setOnClickListener(new View.OnClickListener()

{

@Override

public void onClick(View v)

{

mUserLoginPresenter.login();

}

});

mBtnClear.setOnClickListener(new View.OnClickListener()

{

@Override

public void onClick(View v)

{

mUserLoginPresenter.clear();

}

});

}

@Override

public String getUserName()

{

return mEtUsername.getText().toString();

}

@Override

public String getPassword()

{

return mEtPassword.getText().toString();

}

@Override

public void clearUserName()

{

mEtUsername.setText("");

}

@Override

public void clearPassword()

{

mEtPassword.setText("");

}

@Override

public void showLoading()

{

mPbLoading.setVisibility(View.VISIBLE);

}

@Override

public void hideLoading()

{

mPbLoading.setVisibility(View.GONE);

}

@Override

public void toMainActivity(User user)

{

Toast.makeText(this, user.getUsername() +

" login success , to MainActivity", Toast.LENGTH\_SHORT).show();

}

@Override

public void showFailedError()

{

Toast.makeText(this,

"login failed", Toast.LENGTH\_SHORT).show();

}

}

对于在Activity中实现我们上述定义的接口，是一件很容易的事，毕竟接口引导我们去完成。

最后看我们的Presenter。

## 2.3 Presenter

Presenter是用作Model和View之间交互的桥梁，那么应该有什么方法呢？

其实也是主要看该功能有什么操作，比如本例，两个操作:login和clear。

package com.zhy.blogcodes.mvp.presenter;

import android.os.Handler;

import com.zhy.blogcodes.mvp.bean.User;

import com.zhy.blogcodes.mvp.biz.IUserBiz;

import com.zhy.blogcodes.mvp.biz.OnLoginListener;

import com.zhy.blogcodes.mvp.biz.UserBiz;

import com.zhy.blogcodes.mvp.view.IUserLoginView;

/\*\*

\* Created by zhy on 15/6/19.

\*/

public class UserLoginPresenter

{

private IUserBiz userBiz;

private IUserLoginView userLoginView;

private Handler mHandler = new Handler();

public UserLoginPresenter(IUserLoginView userLoginView)

{

this.userLoginView = userLoginView;

this.userBiz = new UserBiz();

}

public void login()

{

userLoginView.showLoading();

userBiz.login(userLoginView.getUserName(), userLoginView.getPassword(), new OnLoginListener()

{

@Override

public void loginSuccess(final User user)

{

//需要在UI线程执行

mHandler.post(new Runnable()

{

@Override

public void run()

{

userLoginView.toMainActivity(user);

userLoginView.hideLoading();

}

});

}

@Override

public void loginFailed()

{

//需要在UI线程执行

mHandler.post(new Runnable()

{

@Override

public void run()

{

userLoginView.showFailedError();

userLoginView.hideLoading();

}

});

}

});

}

public void clear()

{

userLoginView.clearUserName();

userLoginView.clearPassword();

}

注意上述代码，我们的presenter完成二者的交互，那么肯定需要二者的实现类。大致就是从View中获取需要的参数，交给Model去执行业务方法，执行的过程中需要的反馈，以及结果，再让View进行做对应的显示。