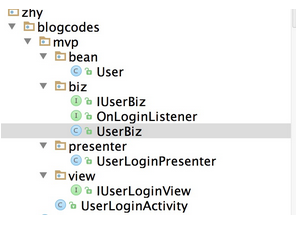
MVP 设计模式  
  
View 对应于Activity，负责View的绘制,数据展示以及与用户交互  
Model 依然是业务逻辑和实体模型,例如联网操作,查询数据操作,逻辑计算....  
Presenter 负责完成View于Model间的交互  
减少了Activity的职责，简化了Activity中的代码，将复杂的逻辑代码提取到了Presenter中进行处理。与之对应的好处就是，耦合度更低，更方便的进行测试  
  
MVC 设计模式  
  
View：对应于布局文件  
Model：业务逻辑和实体模型  
Controllor：对应于Activity  
实际上关于该布局文件中的数据绑定的操作，事件处理的代码都在Activity中，造成了Activity既像View又像Controller  
例子

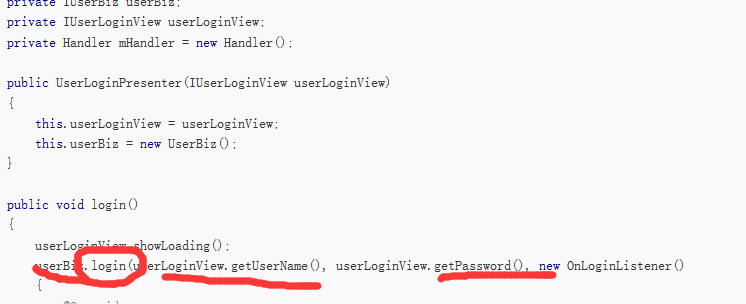
modle 

User 实体类

Biz 逻辑处理类 登录的逻辑处理 login,loginout,,,

Presenter

userLogin 链接 view和 modle



View 就是控件获取数据和显示数据动作,抽取出来,可以直接放到activity中

<http://blog.csdn.net/u013134722/article/details/55098683> mvp解释

单例模式

一个类只能有一个实例；  
2，自己创建这个实例；  
3，整个系统都要使用这个实例

单例模式，能避免实例重复创建；  
单例模式，应用于避免存在多个实例引起程序逻辑错误的场合；  
单例模式，较节约内存



观察着模式 一个接口多个实现,在有变化时,通知到所有一来该接口的实现类

工厂模式 你只需要修改工厂代码。其他地方引用工厂，可以做到只修改一个地方，其他代码都不动，就是解耦了，也就是降低了代码的耦合性

适配器模式

就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。

多个工厂方法模式，是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不

能正确创建对象，而多个工厂方法模式是提供多个工厂方法，分别创建对象。

静态工厂方法模式，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可

\*\*\*\*一个接口多个实现,根据 判断返回他的实现累

Error:Execution failed for task ':app:mergeDebugResources'.

> Error: java.lang.RuntimeException: Some file crunching failed, see logs for details