MVP

用户界面设计模式

- MVC：Model，View，Controller
- MVP：Model，View，Presenter
- MVVM：Model，View，ViewModel

为什么选择MVP

- MVC：因为Activity同时处理业务逻辑和View。如果业务复杂or交互复杂，将会非常臃肿。如果只有逻辑有改动，可能牵一发而动全身，容易引起其他错误。

- MVP：通过Presenter将逻辑代码完全抽离出来，通过接口的方式实现，让单元测试变得可行。相比MVVM简单，易于理解，学习成本低。

- MVVM：相比较下，较复杂。ViewModel是基于DataBinding实现，但是目前AndroidDataBinding不支持双向绑定，而且需要深入学习AndroidDataBinding，学习成本较大。

参考官方

- https://github.com/googlesamples/android-architecture

具体实现

- 基类：BasePresenter，BaseView

```
public interface TXPBasePresenter {

    void destroy();
}
```

```
public interface TXPBaseView<T> {

    void setPresenter(T presenter);
}
```

BaseView中含方法setPresenter，该方法作用是在将presenter实例传入view中，调用时机是presenter实现类的构造函数中。

- Contract类： 统一管理view与presenter的所有的接口，使得view与presenter中的功能，一目了然，维护起来也方便。

```
public class TXPHelpCenterContract {

    interface View extends TXPBaseView<Presenter> {

        void showLoading();

        void hideLoading();

        void showListData(List<TXPHelpCenterModel> listData);

        void showError(String message);

        void openDetail(String url);

        boolean isActive();
    }

    interface Presenter extends TXPBasePresenter {

        void loadData(TXPHelpCenterModel model);

        void click(TXPHelpCenterModel model);
    }
}
```

实现原则

- 想清楚，当前界面有那些行为，先确定接口。

- Presenter中不要存在任何Android API。

存在的问题

- 接口粒度不好控制。
- 需要考虑UI界面的生命周期。
- 如果业务复杂，Presenter依旧会臃肿。