Android单元测试

0.准备

单元测试和集成测试的区别

- **单元测试**：JVM上面直接运行的，验证调用了某个的方法，同时参数是正确的。
- **集成测试:**
  除去单元测试其他测试，需要在模拟器上运行

名词介绍

JUnit4：

java代码单元测试框架

mock和Mockito：

mock就是创建一个虚假的、模拟的对象。在测试环境下，用来替换掉真实的对象。这样就能达到两个目的：1. 可以随时指定mock对象的某个方法返回什么样的值，或执行什么样的动作。 2. 可以验证mock对象的某个方法有没有得到调用，或者是调用了多少次，参数是什么等等。mock就是这样一个框架

Dagger2：

依赖注入：一个类用到了另外一个类，那么前者叫Client，后者叫Dependency
由于类的相互依赖，有时候我们需要将mock出的对象依赖注入到另一个对象中，以实现测试。
Dagger2是一个依赖注入的框架

Robolectric：

可以让jvm运行android代码。Junit不能调用android系统提供的方法，Robolectric解决了这个问题

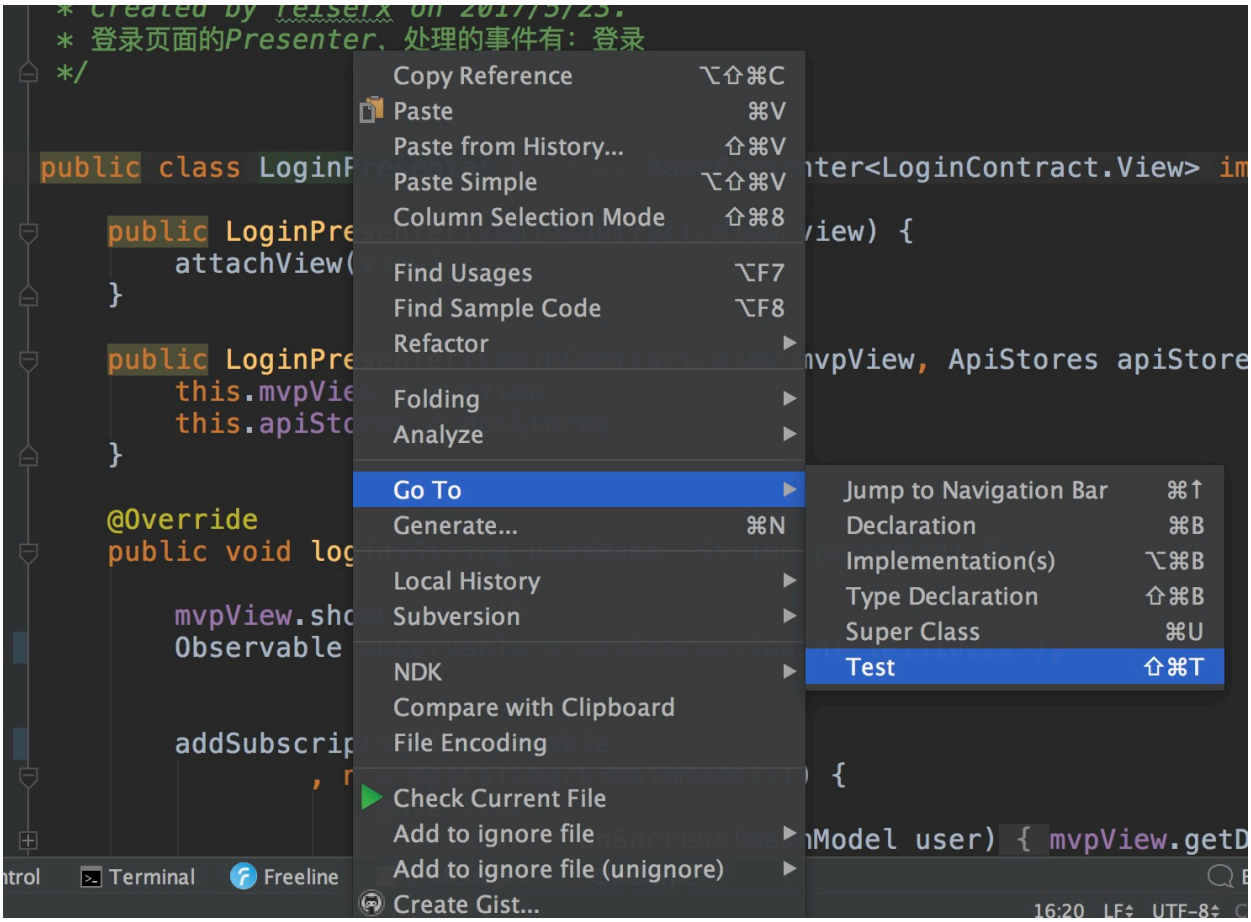
1.目前项目测试框架选用：

JUnit4+ Mockito

> 由于项目采用MVP模式开发，所有的逻辑代码都放入到Presenter，Presenter内只有java代码，所以只使用JUnit4+ Mockito就可完成绝大多数单元测试。所以Dagger2和Robolectric并不作强制要求，但项目会添加相应的依赖，以在特殊情况下可以处理

2.添加测试

添加测试文件



> 直接通过AndroidStudio就可以生成测试文件，位置在main目录同级的test目录下

添加测试代码

测试的方法有两种情况

1）有返回值

使用Junit就可以完成测试

1. setup：即new 出待测试的类，设置一些前提条件（通过@Before注解指定方法）
2. 执行动作：即调用被测类的被测方法，并获取返回结果（在@Test方法中）
3. 验证结果：验证获取的结果跟预期的结果是一样的，assertEquals比较结果，还有assertTrue, assertNotNull等等

```
public class CalculatorTest {

    Calculator calculator;

    @Before//第一步，setup
    public void setUp() throws Exception {
        calculator = new Calculator();
    }

    @Test
    public void add() throws Exception {
        int out = calculator.add(2, 2);//第二步，执行动作
        assertEquals(4,out);//第三步,验证结果
    }

}
```

> 上面的例子是测试一个计算类做加法的功能。第一步，在@Before标签初始化计算类，@Before会在每次测试方法执行前执行。第二步，@Test标签下执行动作，并取得测试的结果。第三步，使用juint的assertEquals判断方法执行的结果和自己预期的是否相等，如例子：calculator.add(2, 2)的结果是否合4相等。若assert通过则测试通过，否则失败。

2）无返回值

当测试测方法返回为空时，我们就没办法使用assert来判断方法是否执行了，这时就需要使用Junit+Mockito，我们将mock出的对象依赖注入进去，就可以判断方法是否执行了。

例如：View和Prsenter代码如下

```
public interface FeedBackContract {

    interface View extends BaseView<Presenter> {

        void addFeedbackSuccess();

    }

    interface Presenter extends BasePresenter {

        void addFeedback(String content, String email);

    }
}
```

```
public class FeedBackPresenter implements FeedBackContract.Presenter {

    private final FeedBackContract.View view;
    private final HttpRequest.ApiService api;

    public FeedBackPresenter(FeedBackContract.View view,
HttpRequest.ApiService api) {
        this.view = view;
        this.api = api;
        this.view.setPresenter(this);
    }

    @Override
    public void addFeedback(String content, String email) {
        view.showProgress();
        Observable<BaseEntity> observable =
ObservableDecorator.decorate(api.addFeedBack(fb));
        observable.subscribe(new Subscriber<BaseEntity>() {
            @Override
            public void onCompleted() {

            }

            @Override
            public void onError(Throwable e) {
                if (!view.isActive()) {
                    return;
                }
                view.dismissProgress();

                view.showTip("反馈提交失败");
            }

            @Override
            public void onNext(BaseEntity entity) {
                if (!view.isActive()) {
                    return;
                }
                view.dismissProgress();

                view.addFeedbackSuccess();
            }
        });
    }
}
```

> 对于上面的presenter我们需要测试addFeedback方法，其中使用了Retrofit作为网络通讯，并回调执行View，由于网络通讯方法是没有返回参数的，view的方法返回也是void。所以没有办法像上面有参数返回时直接判断。当我们需要判断view的方法是否执行时，我门就需要mock一个view并依赖注入到presenter，这样就可以监听到view的方法执行状态了

```
public class FeedBackPresenterTest {

    // 用于测试真实接口返回数据
    private FeedBackPresenter presenter;

    // 用于测试模拟接口返回数据
    private FeedBackPresenter mockPresenter;

    @Mock
    private FeedBackContract.View view;

    @Mock
    private HttpRequest.ApiService api;

    @Before
    public void setupMocksAndView() {
        // 使用@Mock标签等需要先init初始化一下
        MockitoAnnotations.initMocks(this);
        // 用真实接口创建反馈Presenter
        presenter = new FeedBackPresenter(view,
HttpRequest.getInstance().service);
        // 用mock模拟接口创建反馈Presenter
        mockPresenter = new FeedBackPresenter(view, api);
    }

    @Test
    public void testAddFeedback_Success() throws Exception {
        // 真实数据，调用实际接口
        String content = "这天气真是好！";
        String email = "110@qq.com";
        presenter.addFeedback(content, email);

        verify(view).showProgress();
        verify(view).dismissProgress();
        verify(view).addFeedbackSuccess();
    }

    @Test
    public void testAddFeedback_Mock_Success() throws Exception {
        // 模拟数据，当api调用addFeedBack接口传入任意值时，就返回成功的基本数据
BaseEntity

when(api.addFeedBack(any(FeedBack.class))).thenReturn(Observable.just(new
BaseEntity()));
```

```
        String content = "这天气真不错！";
        String email = "119@qq.com";
        mockPresenter.addFeedback(content, email);

        verify(view).showProgress();
        verify(view).dismissProgress();
        verify(view).addFeedbackSuccess();
    }
}
```

> 这里有几个语法需要说明一下
>
> **@Mock**：标记该对象是需要mock的，在 @Before 中执行MockitoAnnotations.initMocks(this);会将该标记下的对象生成mock对象
> **when:**完整为Mockito.when(mockObject.targetMethod(args)).thenReturn(desiredReturnValue);表示当执行某个mock对象的方法时，返回自己指定的值。
> **any:**表示任意的值的该类型。
> **verify:**Mockito.verify(objectToVerify).methodToVerify(arguments);这就是验证操作，他会验证方法是否执行且参数正确

运行测试

- **运行整个项目的测试：**控制台运行在项目根目录下，执行gradle testReleaseUnitTest
- **运行测试文件的测试:**右键点击测试文件，选择Run
- **运行单个测试方法:**右键该方法内部，选择Run

3.需要测试的类和方法

1. 所有的Presenter等类的public方法
2. 所有的Api类的public方法
3. 所有的Utils等类的public方法