Android单元测试(十)：DaggerMock：The Power of Dagger2, The Ease of Mockito

chriszou.com/2016/07/24/android-unit-testing-daggermock

July 24, 2016

The Old Way

我们在系列的第六篇文章前面介绍了Dagger2在单元测试里面的使用姿势。大致过程是这样的，首先，你要mock出一个Module，让它的某个Provider方法在被调用的时候，返回你想到的mock的Dependency。然后使用这个mock的module来build出一个Component，再把这个Component放到你的 `ComponentHolder` 。举个例子说明一下，假设你有一个 `LoginActivity` ，里面有一个 `LoginPresenter` ，是通过Dagger2 inject进去的，如下：

```
public class LoginActivity extends AppCompatActivity {
    @Inject
    LoginPresenter mLoginPresenter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...other code
        ComponentHolder.getAppComponent().inject(this);
    }
}
//对应的Test类如下：
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class, sdk = 21)
public class LoginActivityTest {
    @Test
    public void testLogin() {
        AppModule mockAppModule = Mockito.mock(AppModule.class);
        LoginPresenter mockLoginPresenter = mock(LoginPresenter.class);
        Mockito.when(mockAppModule.provideLoginPresenter(any(UserManager.class),
any(PasswordValidator.class))).thenReturn(mockLoginPresenter);  //当mockAppModule的
provideLoginPresenter()方法被调用时，让它返回mockLoginPresenter
        AppComponent appComponent =
DaggerAppComponent.builder().appModule(mockAppModule).build();  //用mockAppModule来创建
DaggerAppComponent
        ComponentHolder.setAppComponent(appComponent);  //假设你的Component是放在
ComponentHolder里面的
        LoginActivity loginActivity = Robolectric.setupActivity(LoginActivity.class);
        ((EditText) loginActivity.findViewById(R.id.username)).setText("xiaochuang");
        ((EditText) loginActivity.findViewById(R.id.password)).setText("xiaochuang is handsome");
        loginActivity.findViewById(R.id.login).performClick();
        verify(mockLoginPresenter).login("xiaochuang", "xiaochuang is handsome");
    }
}
```

可以看到，为了让Dagger2返回一个Mock对象，我们需要写5行代码。再多写几个测试，我

保证你一定会觉得繁琐的。当然，我们可以使用前一篇文章里面说的方式，和其它的一些手段，来简化代码，以下是我作出的一些努力，应该说，代码已经比较简洁了：

```
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class, sdk = 21)
public class LoginActivityTest {
    @Rule
    public MockitoRule mockitoRule = MockitoJUnit.rule();
    @Mock
    LoginPresenter loginPresenter;
    @Test
    public void testLogin() {
        Mockito.when(TestUtils.appModule.provideLoginPresenter(any(UserManager.class),
any(PasswordValidator.class))).thenReturn(loginPresenter);  //当mockAppModule的
provideLoginPresenter()方法被调用时，让它返回mockLoginPresenter
        TestUtils.setupDagger();
        LoginActivity loginActivity = Robolectric.setupActivity(LoginActivity.class);
        ((EditText) loginActivity.findViewById(R.id.username)).setText("xiaochuang");
        ((EditText) loginActivity.findViewById(R.id.password)).setText("xiaochuang is handsome");
        loginActivity.findViewById(R.id.login).performClick();
        verify(loginPresenter).login("xiaochuang", "xiaochuang is handsome");
    }
}
public class TestUtils {
    public static final AppModule appModule = spy(new
AppModule(RuntimeEnvironment.application));
    public static void setupDagger() {
        AppComponent appComponent =
DaggerAppComponent.builder().appModule(appModule).build();
        ComponentHolder.setAppComponent(appComponent);
    }
}
```

上面把dagger设置相关的代码减少到了两行，应该说，已经不再是一个负担了。然而哪怕是这样，如果写多了的话，依然会让人感觉略烦，因为这也完全是Boilerplate code（这里为什么要用“也”？）。再多写一点，你就会自然而然的想，如果能有一个工具，能达到这样的效果就好了：我们在Test类里面定义一个@Mock field（比如上面的 `loginPresenter` ），这个工具就能自动把这个field作为dagger的module对应的provider方法（ `provideLoginPresenter(...)` ）的返回值。也就是说，自动的mock module，让它返回这个@Mock field，然后用这个mock的module来build一个component，并放到ComponentHolder里面去。

New Hope

Well，我写这篇文章，就是想告诉大家，还真有人写了这样的一个工具，这就是这篇文章要介绍的DaggerMock。它就能达到我们上面描述的那种效果，让我们像使用Mockito Annotation一样来定义Mock，却能自动把它们作为Dagger2生产的Dependency。达到的效果如下：

```
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class, sdk = 21)
public class LoginActivityTest {
    @Rule public DaggerRule daggerRule = new DaggerRule();
    @Mock
    LoginPresenter loginPresenter;
    @Test
    public void testLogin_shinny_way() {
        LoginActivity loginActivity = Robolectric.setupActivity(LoginActivity.class);
        ((EditText) loginActivity.findViewById(R.id.username)).setText("xiaochuang");
        ((EditText) loginActivity.findViewById(R.id.password)).setText("xiaochuang is handsome");
        loginActivity.findViewById(R.id.login).performClick();
        verify(loginPresenter).login("xiaochuang", "xiaochuang is handsome");
    }
}
```

在上面的代码中，已经没有多余的Boilerplate code，要写的代码，基本是必需写的了。上面起作用的是 `@Rule public DaggerRule daggerRule = new DaggerRule();` 这行代码。可见，它是通过JUnit Rule来实现的。如果你熟悉JUnit Rule的工作原理，那么你很容易猜到这个 `DaggerRule` 的工作原理：

1. 初始化一个测试类里面的所有用 `@Mock` field为mock对象(`loginPresenter`)
2. mock `AppModule` ，通过反射的方式得到 `AppModule` 的所有provider方法，如果有某个方法的返回值是一个 `LoginPresenter` ，那么就使用Mockito，让这个方法(`provideLoginPresenter(...)`)被调用时，返回我们在测试类里面定义的mock `loginPresenter` 。
3. 使用这个mock AppModule来构建一个Component，并且放到 `ComponentHolder` 里面去。

我相信看到这里，你一定有很多疑问：

1. 它怎么知道要使用 `AppModule`
2. 它怎么知道要build什么样的Componant
3. 它怎么知道要把build出来的Component放到哪？

好吧，其实上面的 `DaggerRule` ，不是DaggerMock这个library自带的，是我们自己实现的。然而别着急，DaggerMock给了我们提供了一个父类Rule： `DaggerMockRule` ，这个Rule已经帮我们做了绝大多数事情了。我们自定义的 `DaggerRule` ，其实也是继承自 `DaggerMockRule` 的，而我们在自定义Rule里面做的事情，也只不过是告诉DaggerMock，上面说到的三个问题的答案：要使用哪个Module、要build哪个Component、要把build好的Component放到哪，仅此而已。不信请看代码：

```
public class DaggerRule extends DaggerMockRule<AppComponent> {
    public DaggerRule() {
        //告诉DaggerMock要build什么样的Component，使用哪个module
        super(AppComponent.class, new AppModule(RuntimeEnvironment.application));
        //告诉DaggerMock把build好的Component放到哪
        set(new ComponentSetter<AppComponent>() {
            @Override
            public void setComponent(AppComponent appComponent) {
                ComponentHolder.setAppComponent(appComponent);
            }
        });
    }
}
```

怎么样，很简单吧？这个DaggerRule是可以重复使用的，一般来说，一个Component类对应于一个这样的 `DaggerRule` 就好了。自此，你可以只负责使用 `@Mock` 来定义mock了，dagger的事情就交给这个 `DaggerRule` 就好了。
是不是很爽！

哦对了，将这个library加到项目里面的姿势说一下，在build.gradle文件里面加入：

```
repositories {
    jcenter()
    maven { url "https://jitpack.io" }
}
```

和

```
dependencies {
    //...others dependencies
    testCompile 'com.github.fabioCollini:DaggerMock:0.6.1'
    androidTestCompile 'com.github.fabioCollini:DaggerMock:0.6.1' //如果你需要在Instrumentation、Espresso、UiAutomator里面使用的话
}
```

我刚开始使用这个lib的时候，还是花了点时间来理解的，个人认为作者的README和对应的文章写得都不算是很容易看懂，希望这篇文章能让帮助到各位一点点。

照例文中的代码在github的这个repo。

如果你也对安卓单元测试感兴趣，欢迎加入我们的交流群，关注公众号查看怎么加入。