安卓单元测试（九）：使用Mockito Annotation快速创建Mock

chriszou.com/2016/07/16/mockito-annotation

July 16, 2016

注：
如果你还不了解Mock的概念或Mockito框架的使用，请先看这篇文章。

@Mock的基本用法

如果你follow了这个安卓单元测试系列文章，那么到现在为止，你应该很清楚mock的概念和使用了，创建Mock的方法我们都知道：

```
YourClass yourInstance = Mockito.mock(YourClass.class);
```

比如：

```
public class LoginPresenterTest {
   @Test
   public void testLogin() {
      UserManager mockUserManager = mock(UserManager.class);
      PasswordValidator mockValidator = mock(PasswordValidator.class);
      Mockito.when(mockValidator.verifyPassword("xiaochuang is handsome")).thenReturn(true);
      LoginPresenter presenter = new LoginPresenter(mockUserManager, mockValidator);
      presenter.login("xiaochuang", "xiaochuang is handsome");
      verify(mockUserManager).performLogin("xiaochuang", "xiaochuang is handsome");
   }
}
```

虽然很简单，但是如果一个测试类里面很多测试方法都要用到mock，那写起来就会有点麻烦，这时候我们可以写一个 `@Before` 方法来作这个setup工作：

```
public class LoginPresenterTest {
    UserManager mockUserManager;
    PasswordValidator mockValidator;
    LoginPresenter loginPresenter;
    @Before
    public void setup() {
        mockUserManager = mock(UserManager.class);
        mockValidator = mock(PasswordValidator.class);
        loginPresenter = new LoginPresenter(mockUserManager, mockValidator);
    }
    @Test
    public void testLogin() {
        Mockito.when(mockValidator.verifyPassword("xiaochuang is handsome")).thenReturn(true);
        loginPresenter.login("xiaochuang", "xiaochuang is handsome");
        verify(mockUserManager).performLogin("xiaochuang", "xiaochuang is handsome");
    }
}
```

这样可以部分上减少Mock的创建，然而Mock写多了，你也会觉得有点烦，因为完全是Boilerplate code。这里有个更简便的方法，那就是结合Mockito的Annotation和JUnit Rule，达到以下的效果：

```
public class LoginPresenterTest {
    @Rule
    public MockitoRule mockitoRule = MockitoJUnit.rule();
    @Mock
    UserManager mockUserManager;
    @Mock
    PasswordValidator mockValidator;
    LoginPresenter loginPresenter;
    @Before
    public void setup() {
        loginPresenter = new LoginPresenter(mockUserManager, mockValidator);
    }
    @Test
    public void testLogin() {
        Mockito.when(mockValidator.verifyPassword("xiaochuang is handsome")).thenReturn(true);
        loginPresenter.login("xiaochuang", "xiaochuang is handsome");
        verify(mockUserManager).performLogin("xiaochuang", "xiaochuang is handsome");
    }
}
```

也就是说，在测试方法运行之前，自动把用 `@Mock` 标注过的field实例化成Mock对象，这样在测试方法里面就可以直接用了，而不用手动通过 `Mockito.mock(YourClass.class)` 的方法来创建。这个实现化的过程是在 `MockitoRule` 这个Rule里面进行的。我们知道(不知道？)一个JUnit Rule会在每个测试方法运行之前执行一些代码，而这个Rule实现的效果就是在每个测试方法运行之前将这个类的用了 `@Mock` 修饰过的field初始化成mock对象。如果你去看这个Rule的源代码的话，其实重点就在一行代码：

```
MockitoAnnotations.initMocks(target);
```

上面的 `target` 就是我们的测试类（ `LoginPresenterTest` ）。所以，在上面的例子中，如果你不使用这个Rule的话，你可以在 `@Before` 里面加一行代码：

```
@Before
public void setup() {
    MockitoAnnotations.initMocks(this);
    loginPresenter = new LoginPresenter(mockUserManager, mockValidator);
}
```

也能达到一样的效果。

使用@InjectMocks

其实在上面的代码中，我们还可以进一步的简化。我们可以使用 `@InjectMocks` 来让Mockito自动使用mock出来的 `mockUserManager` 和 `mockValidator` 构造出一个 `LoginPresenter` ：

```
public class LoginPresenterTest {
    @Rule
    public MockitoRule mockitoRule = MockitoJUnit.rule();
    @Mock
    UserManager mockUserManager;
    @Mock
    PasswordValidator mockValidator;
    @InjectMocks
    LoginPresenter loginPresenter;
    @Test
    public void testLogin() {
        Mockito.when(mockValidator.verifyPassword("xiaochuang is handsome")).thenReturn(true);
        loginPresenter.login("xiaochuang", "xiaochuang is handsome");
        verify(mockUserManager).performLogin("xiaochuang", "xiaochuang is handsome");
    }
}
```

这是因为Mockito在 `MockitoAnnotations.initMocks(this);` 是时候，看到这个被 `@InjectMocks` 的 `LoginPresenter` 有一个构造方法：

```
    public LoginPresenter(UserManager userManager, PasswordValidator passwordValidator) {
        this.mUserManager = userManager;
        this.mPasswordValidator = passwordValidator;
    }
```

这个构造方法所需要的两个参数正好这个测试类里面有这样的field被 `@Mock` 修饰过。于是就用这两个field来作为 `LoginPresenter` 的构造参数，将 `LoginPresenter` 构造出来了。这个叫做*Constructor Injection* 。
field的声明顺序其实是无所谓的，你完全可以把

```
    @InjectMocks
    LoginPresenter loginPresenter;
```

放在两个 `@Mock` field前面。此外，如果你觉得每个测试类里面都要写这个Rule有点麻烦，你可以建一个Test基类，然后把这个Rule的定义放在基类里面，每个Test类继承这个基类，这样就不用每个测试类自己写这个Rule了。

诡异的@InjectMocks

其实， `@InjectMocks` 是比较tricky的一个东西。比如说，如果 `LoginPresenter` 的构造方法是空的，就是没有参数：

```
public class LoginPresenter {
    private UserManager mUserManager;
    private PasswordValidator mPasswordValidator;
    public LoginPresenter() {
    }
}
```

那么Mockito依然会将 `LoginPresenter` 里面的 `mUserManager` 和 `mPasswordValidator` 初始化为 `LoginPresenterTest` 里面的两个mock对象， `mockUserManager` 和 `mockValidator` 。这个效果跟 `LoginPresenter` 有两个构造参数是一样的。也就是说，如果被 `@InjectMocks` 修饰的field只有一个默认的构造方法，那么在inject mocks的时候，Mockito会去找被 `@InjectMocks` 修饰的field的类（这里是 `LoginPresenter` ）的field，然后根据类型对应的初始化为测试类里面用 `@Mock` 修饰过的field，这个叫*Field Injection*。然而如果 `LoginPresenter` 只有一个 `UserManager` 的构造方法：

```
public class LoginPresenter {
    private UserManager mUserManager;
    private PasswordValidator mPasswordValidator;
    public LoginPresenter(UserManager userManager) {
        this.mUserManager = userManager;
    }
}
```

那么，在上面的 `LoginPresenterTest` 例子里面，只有 `mUserManager` 会被初始化为 `mockUserManager` ，而 `mPasswordValidator` 是不会初始化为 `mockValidator` 的。这就是tricky的地方。
此外还有*Property Setter Injection*，也就是通过setter方法，自动的初始化 `@InjectMocks` 对象。这个要搞清楚具体的工作流程还是有点小复杂的。这三种injection的优先级顺序分别为：
Constructor Injection > Property Setter Injection > Field Injection。 具体情况可以在这里看到。很多人其实都不推荐使用 `@InjectMocks` ，因为很难弄清楚到底会通过哪种方式inject，我个人对这点也感觉有点别扭，我宁愿通过 `new LoginPresenter(mockUserMananger, mockValiator)` 这种方式来创建LoginPresenter对象，而不是使用 `@InjectMocks` 。

使用@Spy创建Spy对象

在介绍Mock的时候，我们还介绍了Spy，同样的，我们可以使用 `@Spy` 来快速创建spy对象。

```
@Spy PasswordValidator spyValidator;
//or
@Spy PasswordValidator spyValidator = new PasswordValidator();
```

当然，就跟使用 `Mockito.spy()` 方法创建Spy对象一样，要么用 `@Spy` 修饰的field的类有默认的Constructor，要么是一个对象。如果 `PasswordValidator` 没有默认无参的构造方法，那么 `@Spy PasswordValidator spyValidator;` 这个方式是会报错的。

小结

本以为这篇文章应该会很短，没想到写出来也不短了，这就跟做一个新feature一样，乍一看好像很简单，一个小时搞定，结果两个小时以后，呃。。。

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。因为群成员超过100人，没办法扫码加入，请关注公众号获取加入方法。