Android单元测试（七）：Robolectric，在JVM上调用安卓的类

chriszou.com/2016/06/05/robolectric-android-on-jvm

June 5, 2016

今天讲讲Android上做单元测试的最后一个难点，那就是在JVM上无法调用安卓相关的类，不然的话，会报类似于下的错误： `java.lang.RuntimeException: Method isEmpty in android.text.TextUtils not mocked.`

关于这个话题，其实我以前是写过的，也许今天我回过头来写这个话题，会采用不一样的形式，不一样的心态来写，然而，作为我写过的第一篇关于单元测试的文章，而且看看时间，是去年的6月15号，再过几天，刚好一周年。想想这篇文章是在我刚开始探索，尝试在安卓上面写单元测试的时候，写的一篇文章，如今因为安卓单元测试的原因，我认识了很多同行，甚至不时有人叫我“大牛大神”之类的，虽然知道大家是客气，我也受之有愧，但怎么滴心里也有点虚荣的开心，哈哈哈。。因此现在回过头去看看当时自己写的东西，不禁觉得有点那啥。。。因此，我决定把之前的文章稍作补充和修改，作为这个系列的第七篇。

----------以下文字写于去年今天----------

作为一只本科非计算机专业的程序猿，手动写单元测试是我从来没接触过的东西，甚至在几个月前，我都不知道单元测试是什么东西。倒不是说没听过这个词，也不是不知道它的大概是什么东西——“用来测试一个方法，或者是一小块代码的测试代码”。然而真正是怎么做的？我并没有一个概念，或者说并没有一个感觉。
记得第一份工作在创新工场的时候，听当时的boss说，公司有个神级的程序员，他会写大量的单元测试，甚至50%以上的代码都是单元测试。当时崇拜之极，却仍然觉得写单元测试是很麻烦的一件事情。

扯远了，话说回来，当你接触多了国外的技术博客，视频之后，你会发现，单元测试甚至TDD，在国外是非常流行的事情。很多人甚至说离开了单元测试，他们便没有办法写代码。这些都让我对单元测试的好感度逐渐的上升。然而，真正让我下定决心，一定要研究一下这个东西的，是前段时间看大名鼎鼎的《重构：改善现有代码的艺术》里面的一段话：

> I've found that writing good tests greatly speeds my programming, even if I'm not refactoring. This was a surprise for me, and it is counterintuitive for many programmers...
> --Martin Fowler 《Refactoring: Improving the Design of Existing Code》

是的，你没看错，他说单元测试可以节约时间，提高开发速度！！！身为一个无可救药的懒癌患者，看了这句话简直就像看到了一道神光似的！既然都可以节省时间，那肯定是要看看的啊！
有趣的是，Martin Fowler在《重构》里面说他最初是因为 Dave Thomas说的一句话，让他走上了单元测试的不归路。而我这几天刚好又在看Dave Thomas写的《Programming Ruby 1.9 & 2.0》，也算是个巧合啊！

Martin Fowler在《重构》里面还解释了为什么单元测试可以节省时间，大意是我们写程序的时候，其实大部分时间不是花在写代码上面，而是花在debug上面，是花在找出问题到底出在哪上面，而单元测试可以最快的发现你的新代码哪里不work，这样你就可以很快的定位到问题所在，然后给以及时的解决，这也可以在很大程度上防止regression（相信QE和QA们一定很喜欢哈哈。。。），这也是个大部分程序员和测试都很痛恨的问题。
之后不久，就开始花了点时间了解了一下Android里面怎么做unit testing，结果却发现那是个非常难办的事情。。。

为什么android unit testing不好做

我们知道安卓的app需要运行在delvik上面，我们开发Android app是在JVM上面，在开发之前我们需要下载各个API-level的SDK的，下载的每个SDK都有一个android.jar的包，这些可以在你的*android_sdk_home*/platforms/下面看到。当我们开发一个项目的时候，我们需要指定一个API-level，其实就是将对应的android.jar 加到这个项目的build path里面去。这样我们的项目就可以编译打包了。然而现在的问题是，我们的代码必须**运行**在emulator或者是device上面，说白了，就是我们的IDE和SDK只提供了开发和编译一个项目的环境，并没有提供运行这个项目的环境，原因是因为android.jar里面的class实现是不完整的，它们只是一些stub，如果你打开android.jar下面的代码去看看，你会发现所有的方法都只有一行实现：`throw RuntimeException("stub!!");`
而运行unit test，说白了还是个运行的过程，所以如果你的unit test代码里面有android相关的代码的话，那运行的时候将会抛出RuntimeException("stub!!")。为了解决这个问题，现在业界提出了很多不同的程序架构，比如MVP、MVVM等等，这些架构的优势之一，就是将其中一层抽出来，变成pure Java实现，这样做unit testing就不会遇到上面这个问题了，因为其中没有android相关的代码。
好奇的童鞋可能会问了，既然android.jar的实现是不完整的，那为什么我们可以编译这个项目呢？那是因为编译代码的过程并没有真正的运行这些代码，它只会检查你的接口有没有定义，以及其他的一些语法是不是正确。举个简单的例子：

```
public class Test {
   public static void main(String[] argv) {
      testMethod();
   }
   public static void testMethod() {
      throw RuntimeException(&amp;quot;stub!!&amp;quot;);
   }
}
```

上面的代码你同样可以编译通过，但你运行的时候，就会抛出异常`RuntimeException("stub!!")`。当我们的项目运行在emulator或者是device上面的时候，android.jar被替换成了emulator或者是device上面的系统的实现，那上面的实现是真正实现了那些方法的，所以运行起来没有问题。
话说回来，MVP、MVVM这些架构模式虽然解决了部分问题，可以测试项目中不含android相关的类的代码，然而一个项目中还是有很大部分是android相关的代码的，所以上面那种解决方案，其实是放弃了其中一大块代码的unit test。

当然，话说回来，android还是提供了他自己的testing framework，叫instrumentation，但是这套框架还是绕不开刚刚提到的问题，他们必须跑在emulator或者是device上面。这是个很慢的过程，因为要打包、dexing、上传到机器、运行起来界面。。。这个相信大家都有体会，尤其是项目大了以后，运行一次甚至需要一两分钟，项目小的话至少也要十几秒或几十秒。以这个速度是没有办法做unit test的。
那么怎么样即可以给android相关的代码做测试，又可以很快的运行这些测试呢？

Robolectric to the rescue

解决的办法就是使用一个开源的framework，叫robolectric，他们的做法是通过实现一套JVM能运行的Android代码，然后在unit test运行的时候去截取android相关的代码调用，然后转到他们的他们实现的代码去执行这个调用的过程。举个例子说明一下，比如android里面有个类叫 `TextView` ，他们实现了一个类叫 `ShadowTextView` 。这个类基本上实现了 `TextView` 的所有公共接口，假设你在unit test里面写到 `String text = textView.getText().toString();` 。在这个unit test运行的时候，Robolectric会自动判断你调用了Android相关的代码 `textView.getText()` ，然后这个调用过程在底层截取了，转到 `ShadowTextView` 的 `getText` 实现。而 `ShadowTextView` 是真正实现了 `getText` 这个方法的，所以这个过程便可以正常执行。
除了实现Android里面的类的现有接口，Robolectric还做了另外一件事情，极大地方便了unit testing的工作。那就是他们给每个Shadow类额外增加了很多接口，可以读取对应的Android类的一些状态。比如我们知道 `ImageView` 有一个方法叫 `setImageResource(resourceId)` ，然而并没有一个对应的getter方法叫 `getImageResourceId()` ，这样你是没有办法测试这个 `ImageView` 是不是显示了你想要的image。而在Robolectric实现的对应的 `ShadowImageView` 里面，则提供了 `getImageResourceId()` 这个接口。你可以用来测试它是不是正确的显示了你想要的Image。

Talk is cheap. Show me the code!

下面简单的介绍一下使用Robolectric来做unit testing。注意：下面的配置方法指的是AndroidStudio上面的，Eclipse用户自行google一下配制方法。
要使用Robolectric，需要做几步配置工作。

1. 首先需要将它和JUnit4加到你项目的dependencies里面，

```
testCompile &amp;#039;junit:junit:4.12&amp;#039;
testCompile 'org.robolectric:robolectric:3.0-rc3'
```

其中的Robolectric的最新版本号可能会变，具体可以上jcenter查看一下当前的最新版本号。

2. 如果你用的是AndroidStudio2.0一下的版本，需要将 `Build Variant` 里面的 `Test Artifact` 选择为Unit Test，如果你找不到 `Build Variant` ，可以在菜单栏选择 `View -> Tool Windows -> Build Variant` . 正常情况下它会出现在左下角。AndroidStudio2.0以上的版本已经不需要了。

3. 如果是Mac的话，还需要配置一个东西，菜单栏选择 `Run -> Edit Configuration -> Defaults -> JUnit` ，在Configuration tab将working directory改成 `$MODULE_DIR$` 。这个配置是Robolectric官方文档提到的，但我用最新的AndroidStudio1.3实验的时候，忘了配置这个，貌似也可以正确运行，anyway，配置一下也无所谓。具体见Robolectric的官方文档，最下面那部分。

到这里，就可以开始code了。
测试代码是放在 `app/src/test` 下面的，test class的位置最好跟target class的位置对应，比如 `MainActivity` 放在
`app/src/main/java/com/domain/appname/MainActivity.java`
那么对应的test class MainActivityTest最好放在
`app/src/test/java/com/domain/appname/MainActivityTest.java`
这里举个简单又稍微有点用的例子，假设app里面有两个Activity： `MainActivity` 和 `SecondActivity` ， `MainActivity` 里面有一个 `TextView` ，点击一下这个 `TextView` 将跳转到 `SecondActivity` ， `MainActivity` 里面的代码大概如下：

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = (TextView)findViewById(R.id.textView1);
        textView.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(MainActivity.this, SecondActivity.class));
            }
        });
    }
}
```

对应的测试类， `MainActivityTest` 的代码：

```
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class, sdk = 21)
public class MainActivityTest {
    @Test
    public void testMainActivity() {
        MainActivity mainActivity = Robolectric.setupActivity(MainActivity.class);
        mainActivity.findViewById(R.id.textView1).performClick();

        Intent expectedIntent = new Intent(mainActivity, SecondActivity.class);
        ShadowActivity shadowActivity = Shadows.shadowOf(mainActivity);
        Intent actualIntent = shadowActivity.getNextStartedActivity();
        Assert.assertEquals(expectedIntent, actualIntent);
    }
}
```

上面的代码测试的就是当用户点击 `textView` 的时候，程序会正确的跳转到 `SecondActivity` 。其中 `@RunWith(RobolectricGradleTestRunner.class)` 表示用Robolectric的TestRunner来跑这些test，这就是为什么Robolectric可以检测到你调用了Android相关的类，然后截取这些调用，转到他们的Shadow类的原因。此外， `@Config` 用来配置一些东西。
代码中的
`MainActivity mainActivity = Robolectric.setupActivity(MainActivity.class);` 用来创建 `MainActivity` 的instance，或者说，用来启动这个Activity，
当 `Robolectric.setupActivity` 返回的时候，这个Activity已经完成了onCreate、onStart、onResume这几个生命周期的回调了。
`mainActivity.findViewById(R.id.textView1).performClick();` 用来触发点击事件。 `ShadowActivity shadowActivity = Shadows.shadowOf(mainActivity);` 用来获取mainActivity对应的ShadowActivity的instance。
`shadowActivity.getNextStartedActivity();` 用来获取mainActivity调用的startActivity的intent。这也是正常的Activity类里面不具有的一个接口。
最后，调用 `Assert.assertEquals` 来assert启动的intent是我们期望的intent。
运行这个unit test，启动命令行，cd到项目的根目录，运行
`./gradlew test` ，几秒钟后，你将看到测试运行的结果

```
...
:app:processDebugJavaRes UP-TO-DATE
:app:compileDebugJava UP-TO-DATE
:app:preCompileDebugUnitTestJava
:app:preDebugUnitTestBuild UP-TO-DATE
:app:prepareDebugUnitTestDependencies
:app:processDebugUnitTestJavaRes UP-TO-DATE
:app:compileDebugUnitTestJava
:app:compileDebugUnitTestSources
:app:mockableAndroidJar UP-TO-DATE
:app:assembleDebugUnitTest
:app:testDebug

BUILD SUCCESSFUL

Total time: 12.884 secs
```

在我的机器上（MacBook Air 2013款，8G内存，算比较低的配置），运行这个test只需要不到12秒钟，如果直接在AndroidStudio里面运行的话，这个速度会更快，一般可以再10秒之内完成，或许没有达到普通JUnit的秒级速度，然而相对于用Instrumentation来说已经是极大的提升了。
注：第一次运行可能需要下载一些library，或者是gradle本身，可能需要花一点时间，这个跟unit test本身没关。
整个项目已经放到github上面：robolectric-demo

小结

总体来说，Robolectric是个非常强大好用的unit testing framework。虽然使用的过程中肯定也会遇到问题，我个人就遇到不少问题，尤其是跟第三方的library比如Retrofit、ActiveAndroid结合使用的时候，会有不少问题，但瑕不掩瑜，我们依然可以用它完成很大部分的unit testing工作。

----------原文结束----------
今天回过头来看，我想强调的是，Robolectric到底应该充当什么样的一个角色。在没有Robolectric的pure JUnit世界，我们是很难对一整个流程进行测试的，因为上层的界面是安卓的类，底层的数据库和Preference等等是安卓的类。因此，我们没有办法对一整个流程做一个完整的测试。然而有了robolectric以后，我们就可以这么做了：启动activity，向网络或数据库请求数据，更新界面。。。因此，有了这个东西以后，我们的第一反应可能就是去测试这整个app流程。所以经常有小伙伴问我，Robolectric到底是做单元测试的框架，还是做集成测试，甚至UI测试的框架？
这就是我想强调的，需要避免的陷阱。对于上面的问题，我的回答是：Robolectric就是一个能够让我们在JVM上跑 **测试** 时够调用安卓的类的框架，至于我们是拿它来做单元测试还是集成测试，完全取决于我们自己。而回到我们强调的 **单元测试**，测一个小的独立的代码单元，Robolectric的角色，应该是一个让我们在做 **单元测试** 的过程中，能够调用安卓的类，测试安卓的类，把安卓的类当做普通的纯java类的一个framework，仅此而已。
这点，谨记。

最后，如果你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，请关注下方公众号获取加入方法。

有任何意见或建议，或者发现文中任何问题，欢迎留言评论！