Android 单元测试: 首先，从是什么开始

gold.xitu.io

这是一系列安卓单元测试的文章，目测主要会cover以下的主题：

1. 什么是单元测试
2. 为什么要做单元测试
3. JUnit
4. Mockito
5. Robolectric
6. Dagger2
7. 一个具体的app例子实践
8. 神秘的bonus

什么是单元测试

首先需要介绍一下什么是单元测试。很多人像我一样，本科并不是计算机专业出身的，如果在职的公司不要求做单元测试的话，可能对这个词并没有一个确切的概念。而即使是计算机专业出身，如果毕业以后写的不多的话，可能对这个词的含义也不是很清楚。从名字上看，单元测试是为了测试某一个代码单元而写的测试代码。但是什么叫“一个代码单元”呢？是一个模块、还是一个类、还是一个方法（函数）呢？不同的人、不同的语言，都有不同的理解。一般的定义，尤其是是在OOP领域，是一个类的一个方法。在此，我们也这样理解：单元测试，是为了测试某一个类的某一个方法能否正常工作，而写的测试代码。
我们举一个例子说明一下，假如你有一个类，定义如下：

```
public class Calculator {
    public int add(int one, int another) {
        //为了简单起见，暂不考虑溢出等情况。
        return one + another;
```

```
    }
}
```

那么为了测试这个Calculator类的add()方法，我们可以写如下的单元测试代码：

```
public class CalculatorTest {
    public void testAdd() throws Exception {
        Calculator calculator = new Calculator();
        int sum = calculator.add(1, 2);
        Assert.assertEquals(3, sum);
    }
}
```

这里的CalculatorTest是Calculator对应的测试类。而这里的testAdd()就是add()这个方法对应的测试方法。所以，写单元测试，就是给你的每个类的每个public方法写对于的测试方法。非public方法我们一般是不测试的，虽然可以通过反射等手段去做，但是一般看来，非public方法是这个类的实现细节，我们并不关心，我们只关心某一个public方法的输入、输出。
一般来说，一个方法对应的测试方法主要分为3部分，以上面的测试方法为例：

1. setup。一般是new出你要测试的那个类，以及其他一些前提条件的设置：Calculator calculator = new Calculator();
2. 执行操作。一般是调用你要测试的那个方法，获得运行结果：int sum = calculator.add(1, 2);
3. 验证结果。验证得到的结果跟预期中是一样的：Assert.assertEquals(3, sum);

一般来说，我们写单元测试，会用到一些单元测试框架。常见的Java单元测试框架有JUnit、TestNG等等。在这个系列的文章中，我采用JUnit 4，这也是用的最多的一个测试框架。上面的第三部，Assert.assertEquals(3, sum); 用的就是JUnit里面的验证结果的方法，最常见的就是调用Assert类的一些assert方法。除了上面用到的assertEquals，还有assertTrue, assertNotNull等等。关于JUnit，我会在后面的系列文章中专门介绍。

如何在一个android project里面运行单元测试

我们知道，在一个android gradle project中，源代码默认是放在src/main/java下面的。而对应的单元测试代码则是放在src/test/java下面的，如下图所示：

其中的package name可以随意定，很多人喜欢跟src package name保持一致，我个人习惯在最

后加上.test后缀，因为AndroidStudio太智能了，经常我需要重命名单元测试的package的时候，AndroidStudio会把src的package也给重命名了。
打开CalculatorTest，用鼠标右键点击testAdd()方法，选择Run testAdd(), 如下图所示：

从图中你可以看出，你可以按快捷键Ctrl+Shift+R快速运行，当然，这要求你的光标当前焦点是在这个方法内部的。如果你的焦点是在类内部，而不在某一个测试方法内部，那么Ctrl+Shift+R将运行这个测试类的所有测试方法。当然，你也可以通过右键点击测试类名来运行这个测试类里面的所有测试方法。
运行结束以后，你会在底部的“Run”这个tab看到运行的结果，如下图所示：

除了在AndroidStudio里面运行，你还可以在命令行通过gradle testDebugUnitTest，或者是gradle testReleaseUnitTest，分别运行debug和release版本的unit testing，这种方式可以一次性运行所有测试类的所有测试方法。 这种方式的运行结果如下图所示：

每个test case的报告可以在project_root/app/build/reports/tests/debug/index.html 这个xml里面看到。大致如下图：

这篇文章的代码在github上，感兴趣的可以clone下来自己试试。

单元测试不是集成测试

这里需要强调一个观念，那就是单元测试只是测试一个方法单元，它不是测试一整个流程。举个例子来说，一个Login页面，上面有两个输入框和一个button。两个输入框分别用于输入用户名和密码。点击button以后，有一个UserManager会去执行performlogin操作，然后将结果返回，更新页面。
那么我们给这个东西做单元测试的时候，不是测这一整个login流程。这种整个流程的测试：给两个输入框设置正确的用户名和密码，点击login button, 最后页面得到更新。叫做集成测试，而不是单元测试。当然，集成测试也是有他的必要性的，然而这不是我们每个程序员应该花多少精力所在的地方。在这方面，有一个理论叫做Test Pyramid，如下图所示：

Test Pyramid理论基本大意是，单元测试是基础，是我们应该花绝大多数时间去写的部分，而集成测试等应该是冰山上面能看见的那一小部分。
为什么是这样呢？因为集成测试设置起来很麻烦，运行起来很慢，发现的bug少，在保证代码质量、改善代码设计方面更起不到任何作用，因此它的重要程度并不是那么高，也无法将它纳入我们正常的工作流程中。
而单元测试则刚好相反，它运行速度超快，能发现的bug更多，在开发时能引导更好的代码设计，在重构时能保证重构的正确性，因此它能保证我们的代码在一个比较高的质量水平上。同时因为运行速度快，我们很容易把它纳入到我们正常的开发流程中。
至于为什么集成测试发现的bug少，而单元测试发现的bug多，这里也稍作解释，因为集成测

试不能测试到其中每个环节的每个方面，某一个集成测试运行正确了，不代表另一个集成测试也能运行正确。而单元测试会比较完整的测试每个单元的各种不同的状况、临界条件等等。一般来说，如果每一个环节是对的，那么在很大的概率上，整个流程就是对的。虽然不能保证整个流程100%一定是对的。所以，集成测试需要有，但应该是少量，单元测试是我们应该花重点去做的事情。
那对于这个例子，单元测试是怎么样的呢？这个请看下一小节。

两种函数（方法），两种不同的测试方式

一个类的方法可以分为两种，一种是有返回值的，另一种是没有返回值的。对于有返回值的方法，我们要测起来比较容易，就跟上面的Calculator例子那样，输入相应的参数，得到相应的返回值，然后验证得到的返回值跟我们预期的值一样，就好了。
但是没有返回值的方法，要怎么测试呢？比如说刚刚login的例子，点击那个按钮，会执行Activity的login()方法，它的定义如下：

```
public void login() {
    String username = ...//get username from username EditText
    String password = ...//get password from password EditText
    //do other operation like validation, etc
    ...

    mUserManager.performlogin(username, password);
}
```

这个方法是void的，那么怎么验证这个方法是正确的呢？其实仔细想想，这个方法也是有输出的，它的输出就是，调用了mUserManager的performLogin方法，同时传给他两个参数。所以只要验证mUserManager的performLogin方法得到了调用，同时传给他的参数是正确的，就说明这个方法是能正常工作的。
那怎么样验证这个Activity的login()方法，会调用mUserManager的performLogin方法呢？这里涉及到mock的概念，在后面的文章关于Mockito的使用的时候，会介绍到，这里简单解释下，那就是在测试环境下，我们会使用一套mock framework（Mockito），生成一个mock的UserManager然后赋值给mUserManager，因为这个mUserManager是通过mock framework生成的，所以这个mock framework可以验证它的什么方法被调用了，参数是什么，等等。

小结

上面讲述了单元测试的定义，以及与集成测试的区别，一般来说，单元测试不会接触到数据库，不会接触到网络，不会接触到一些复杂的外部环境，如果有的话，那可能是你测试的方式有误，测试的粒度不够“单元”，希望这篇文章能将这两者的区别解释清楚。

如果文中有任何的错误或疑问欢迎留言。

有任何意见或建议，或者发现文中任何问题，欢迎留言！

更多文章请访问个人网站
作者：邹小创
Github：https://github.com/ChrisZou
邮件：happystriving@126.com