Kotlin 写 Android 单元测试（一），单元测试是什么以及为什么需要


johnnyshieh.me/posts/unit-test-what-and-why-in-kotlin

Kotlin 写 Android 单元测试系列文章：

Kotlin 写 Android 单元测试（一），单元测试是什么以及为什么需要

Kotlin 写 Android 单元测试（二），JUnit 4 测试框架和 kotlin.test 库的使用

Kotlin 写 Android 单元测试（三），Mockito mocking 框架的使用

Kotlin 写 Android 单元测试（四），Robolectric 在 JVM 上测试安卓相关代码

未完待续...

越来越多的开发者开始使用 Kotlin 开发 Android 项目，Kotlin 逐渐成为 Android 开发的第一语言，公司项目中也开始用 Kotlin 写单元测试，在项目中以单元测试作为开始引入 Kotlin 语言的入口是个不错的选择，大家可以尝试下。但是现在网上关于如何使用 Kotlin 语言写单元测试的文档很少，所以写下这个系列文章记录自己使用 Kotlin 写 Android 项目的单元测试的一些经验。

有些人可以对单元测试的概念比较模糊，更多的人则认为单元测试是完全没有必要的，所以下面介绍什么是单元测试以及为什么需要写单元测试。

什么是单元测试

单元测试的定义

在 WiKi 上的定义是：在计算机编程中，单元测试（英语：Unit Testing）又称为模块测试，是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作。程序单元是应用的最小可测试部件。

简单点说就是对程序单元进行测试，在面向过程开发中，一个单元就是一个函数；在面向对象开发中，一个单元就是类的一个方法。

下面使用 Junit 4 中的 Calculator 例子来说明。

假设我们有一个 Calculator 类：

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    // 计算整数相加的表达式的结果
    for (String summand:
expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

那么为 Calculator 类的 `evaluate` 方法写的单元测试代码如下：

```
import static
org.junit.Assert.assertEquals;
import org.junit.Test;
public class CalculatorTest {
  @Test
  public void testEvaluate() {
    Calculator calculator = new
Calculator();
    int sum =
calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

`testEvaluate` 就是 `evaluate` 的测试方法，测试 `evaluate` 方法是否实现预期功能。

从上面的测试方法中可以发现，一般单元测试分为 3 步：

1.初始化。一般是准备一些测试的前提条件，如新建需要测试的类的实例：`Calculator calculator = new Calculator();`。

2.调用要测试方法。如 `int sum = calculator.evaluate("1+2+3");`。

3.验证结果。验证测试结果是否和预期一致：`assertEquals(6, sum);`。

有的时候还会有第四步：

4.收尾工作。一般释放资源或者删除文件等。

单元测试的重点

Java 和 Android 中程序是由多个对象组成的，对象之间可以互相调用公开的方法，单元测试就是测试对象的公开方法，也就是说测试类的 public 方法，因为 public 方法是公开的，定义了类的行为，其他类可以调用。而 private 方法是类的内部实现细节，所以一般是不测

试的，**单元测试的重点是测试类的 public 方法**。

再进一步的说，面向对象中单元测试是：测试类的 public 方法的输出是否与预期一致。

看下面的例子：

```
public void save(String result) {
    ...
    mStorage.saveToFile(result);    // 把 result 写入到文
件 file
}
```

`save` 方法没有返回值，那么它的输出是什么，可能很多人会说就是 `result 写入到文件 file` ，但是那是 `saveToFile` 方法的输出，而 `save` 方法的输出是调用了 `mStorage.saveToFile()` 方法并且参数是 `result` 。单元测试只是测试方法单元，并不是测试一个流程。

所以方法的输出分为两种：

1. **直接的返回结果**，这种情况可以用 Junit 4 测试框架的 `assert` 语句测试。
2. **间接的方法调用**，这种情况验证方法是否调用，需要利用 mock 的测试框架，例如 Mockito，后续会有介绍。

为什么要写单元测试

前面介绍了什么是单元测试，但是很多人还是不愿意写单元测试，之前我也认为单元测试是浪费时间，自己写的单元测试肯定都能测试通过，没有多大意义。大学时老师讲的单元测试的意义，自己也没当回事，没有单元测试，照样写好代码，而且速度还更快。毕业去第一家公司上班后，同事也是基本不写单元测试的，也没有认识到单元测试的价值。

但是在 Github 上看国外大神的代码时，经常看到他们都会写测试代码，会怀疑单元测试真的有用？真正让我改变看法是不经意间看到 `小创作` 的一篇文章 Android单元测试（二）：再来谈谈为什么，于是开始尝试写单元测试代码，慢慢感受到单元测试的魅力，也开始由路转粉了。下面从自己的感受来说说，为什么需要写单元测试？

写正式代码时会不自觉地分离接口和实现，代码设计更加良好

如果在项目一开始就写单元测试代码，甚至采用测试驱动开发，写出的代码质量会更高。在日常开发中，很多类会引用其他类，例如一个依赖于数据库的类，为了测试它，测试人员通常编写代码去操作数据库。这是不对的，因为单元测试不应超出待测试的类边界。前面有提到这时需要的是验证数据库的某个方法是否被调用，这时需要创建一个数据库连接的接口，然后实现这个接口的 mock 对象，去测试 mock 对象的某个方法是否被调用。这时已经不自觉地使用面向接口设计，分离数据库的接口和实现，扩展性更好。

另外在某一个方法的实现前，先去写它对应的单元测试代码，需要注意该方法的输入和输出，不去想实现细节。也会考虑该方法的一些异常，各种情况的预期结果是什么。这样会对这个方法的本质有更清晰的认识，边界情况也考虑的更加全面，然后再去写正式代码时，我发现更加容易，质量也更高。

节约开发过程中 debug 时间

在没有写测试代码之前，开发过程中需要完成一个流程中所有模块，然后等 app 运行起来后看看有没 Crash 或者手动进入各个界面测试有没有异常，效率很低，而且往往一开始 Crash 的问题都是一些写代码时疏忽导致的低级错误。相信大多数开发者都跟我一样，或多或少总是会不小心犯一些低级错误。然后改一点后再运行一遍 app，这样反复进行。这种痛相信很多开发者都经历过，测试效率真是太低了。

而开始写单元测试代码后，写完一部分代码就执行下测试代码，不用等其他模块也完成，其他模块可以先只定义好接口。这样在完成一个小模块的代码，可以通过单元测试及时发现错误，而且单元测试的方式比运行 app 测试的方式速度更快，也更容易发现问题。

保证代码工作正确，加强自信心

很多时候，写完代码后自己也多大信心，不知道运行起来是否正常，一些特殊情况是否会崩溃，特别修改已有代码，不知道会不会有其他影响。有单元测试后，就不用那么担心，单元测试通过可以一定程度保证代码工作正确，对后续发版本提供信心，不用那么担心啦。人有自信很重要，自信的人通常更帅^_^。

便于后期重构或变更时，确保其他模块工作正确

在项目开发完成后，后期维护或重构的时间通常比较长，这时单元测试的价值也更为凸显。在修改一个地方的代码时，可能会对其他的代码造成影响，但是我们开发者很难知道。经常会出现一种情况，修改一个 bug，会产生另外一个 bug。

单元测试可以保证修改代码后其他模块依然工作正确，一旦变更导致错误发生，借助单元测试可以快速定位错误。这样在变更代码可以确保不影响其他模块，降低重构成本。

还是不想写单元测试

可能看完上面写的这些内容，有些人还是不想写单元测试，主要有下面三个原因：

写单元测试增加时间成本

的确学习单元测试本身就需要额外时间，写单元测试代码也需要时间，但是单元测试可以减少后续的 debug 时间，还能保证代码质量，总的来说其实可以帮我们节约时间。而且学习单元测试让自己多一个技能点，国外大神都有的技能点，何乐而不为呢。

不相信单元测试的价值

人们对于未知的事物总是会持有怀疑态度，我也一样，但是很多时候需要动手尝试，不去行动永远不知道具体是怎样的。所以去尝试开始写单元测试，你会慢慢发现单元测试的魅力的。

在已有项目中加入单元测试比较困难

是的，我在公司项目加入单元测试也遇到一些困难，有些代码不方便测试，需要调整其他的代码或者加上接口，这比较花费时间。但是我也发现一般一个类不好测试时，往往因为这个类的设计有问题。调整后便于测试的代码，设计更加良好，更具有扩展性。所以这是一件非常值得做的事情，相当于重构代码。

也可以在新需求中新增代码时，加入单元测试，这样困难会少一点，在涉及到之前代码时，可能还需要调整，这时不要轻易放弃，在经过一开始的代码适应阶段后，写单元测试会越来越容易。

小结

总的来说，单元测试可以提升代码设计，节约测试时间，增强对代码的信心。希望通过这篇文章让更多的人清楚单元测试的概念，同时开始学习写单元测试，后面会介绍如何用 Kotlin 语言写 Android 项目的单元测试。

参考文章：

- Android 单元测试: 首先，从是什么开始
- Android单元测试（二）：再来谈谈为什么