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


johnnyshieh.me/posts/unit-test-mockito-2-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...

Junit 4 测试框架可以验证有直接返回值的方法，但是对于没有返回值的 void 方法应该如何测试呢？void 方法的输出结果其实是调用了另外一个方法，所以需要验证该方法是否有被调用，调用时参数是否正确。Mocking 框架可以验证方法的调用，目前流行的 Mocking 框架有 Mockito、JMockit、EasyMock、PowerMock 等。我选择的是 Mockito 框架，原因是：（1）Mockito 是 Java 中最流行的 mocking 框架；（2）Google 的 Google Sample 下的开源库中使用也是 Mockito 框架。下面介绍 Mockito 框架一些概念和用法，以及 Kotlin 中 mockito-kotlin 库的使用。

> 本文是基于 Mockito 2.13.0 版本，Android Studio 3.0 环境

1. Mockito 框架

Gradle 引入

```
testImplementation 'org.mockito:mockito-core:2.13.0'
// 如果需要 mock final 类或方法的话，还要引入 mockito-
inline 依赖
testImplementation 'org.mockito:mockito-inline:2.13.0'
```

先看下 Mockito 的一个简单的例子（选自 Mockito 文档）：

```
//mock creation
//we can not use MutableList<String>::class.java as Class
type
val mockedList = mock(mutableListOf<String>
().javaClass)
//using mock object
mockedList.add("one")
mockedList.clear()
//verification
verify(mockedList).add("one")
verify(mockedList).clear()
```

上面例子中可以看出，使用 Mockito 很容易验证 mock 对象的方法调用，**注意这里的限制是只能验证 mock 对象的方法调用。**

1.1 mock 和 spy

创建 mock 对象是 Mockito 框架生效的基础，有两种方式 `mock` 和 `spy`。`mock 对象` 的属性和方法都是默认的，例如返回 null、默认原始数据类型值（0 对于 int/Integer）或者空的集合，简单来说只有类的空壳子。而 `spy 对象` 的方法是真实的方法，不过会额外记录方法调用信息，所以也可以验证方法调用。

```
val mockedList = mock(mutableListOf<String>
().javaClass)
val spyList = spy(mutableListOf<String>())
// mock object methods actually do nothing
mockedList.add("one")
// spy object call *real* methods
spyList.add("one")
```

Mockito 还提供了 @Mock 等注解来简化创建 mock 对象的工作

```
class CalculatorTest {
    @Mock
    lateinit var calculator:
Calculator
    @Spy
    lateinit var dataBase: Database
    @Spy
    var record =
Record("Calculator")
    @Before
    fun setup() {
        // 必须要调用这行代码初始化
Mock

MockitoAnnotations.initMocks(this)
    }
}
```

除了显式地调用 `MockitoAnnotations.initMocks(this)` 外，还可以使用 `MockitoJUnitRunner` 或者 `MockitoRule` 。使用方式如下：

```
@RunWith(MockitoJUnitRunner.StrictStubs::class)
class CalculatorTest {
    @Mock
    lateinit var calculator: Calculator
}
// or
class CalculatorTest {
    @Rule @JvmField
    val mockitoRule =
MockitoJUnit.rule().strictness(Strictness.STRICT_STUBS)
    @Mock
    lateinit var calculator: Calculator
}
```

还有 @InjectMocks 注解提供构造函数注入、setter 注入或成员注入，具体细节请看官网文档。

1.2 验证方法调用

Mockito 中可以很方便地验证 `mock 对象` 或 `spy 对象` 的方法调用，通过 `verify` 方法即可：

```
val mockedList = Mockito.mock(mutableListOf<String>().javaClass)
mockedList.add("once")
mockedList.add("twice")
mockedList.add("twice")
mockedList.add("three times")
mockedList.add("three times")
mockedList.add("three times")
//following two verifications work exactly the same - times(1) is used by
default
verify(mockedList).add("once");
verify(mockedList, times(1)).add("once")
//exact number of invocations verification
verify(mockedList, times(2)).add("twice")
verify(mockedList, times(3)).add("three times")
//verification using never(). never() is an alias to times(0)
verify(mockedList, never()).add("never happened")
//verification using atLeast()/atMost()
verify(mockedList, atLeastOnce()).add("three times")
verify(mockedList, atLeast(2)).add("three times")
verify(mockedList, atMost(5)).add("three times")
```

verify 方法使用非常简便，但是还是有需要注意的地方，看下面测试代码：

```
val mockedList = Mockito.mock(mutableListOf<String>
().javaClass)
mockedList.add("twice")
verify(mockedList).add("twice")
mockedList.add("twice")
verify(mockedList).add("twice")
```

上面的测试代码在第 5 行中的验证会失败，提示期望一次调用，实际上为两次。第一次验证时成功是没问题的，第二次验证时，此时 `mockedList.add("twice")` 执行了两次，记录为两次，没有随着第一次验证过后就删除 1 次，所以会测试失败。

验证参数值

上面的验证方法调用时，对于参数的校验使用的默认 equals 方法，除此之外也可以使用 argument matchers：

```
verify(mockedList).add(anyString())
verify(mockedList).add(notNull())
verify(mockedList).add(argThat{ argument -> argument.length
> 5 })
```

1.3 stubbing 指定方法的实现

除了验证方法调用之外，Mockito 还有另外一个主要功能：指定方法的返回值或者实现。不过需要使用到 `when` 方法，而在 Kotlin 中 when 属于关键字。

```
val mockedList = mock(mutableListOf<String>().javaClass)
// mockedList[0] 第一次返回 first，之后都会抛出异常
`when`(mockedList[0]).thenReturn("first")
        .thenThrow(IllegalArgumentException())
`when`(mockedList[1]).thenThrow(RuntimeException())
`when`(mockedList.set(anyInt(), anyString())).thenAnswer({
invocation ->
    val args = invocation.arguments
    println("set index ${args[0]} to ${args[1]}")
    args[1]
})
// use doThrow when stubbing void methods with exceptions
doThrow(RuntimeException()).`when`(mockedList).clear()
doReturn("third").`when`(mockedList)[2]
```

需要注意下 stubbing 方法的规则：

- 一旦指定了方法的实现后，不管调用多少次， 该方法都是返回指定的返回值或者执行指定的方法
- 当以相同的参数指定同一个方法多次时，最后一次指定才会生效

指定方法实现通常使用 `thenReturn` 、 `thenThrow` 、 `thenAnswer` 等， 因为这种方式更直观。 但是 上面的例子中还有 `doReturn` 、 `doThrow` 、 `doAnswer` 等 do 系列方法，它可以实现 then 系列方法同样的功能，不过在阅读上没有那么直观。 在下面几种情况下必须使用 do 系列方法：

- 指定 void 方法
- 指定 spy 对象的某些方法时
- 多次指定同一方法，以便在测试中途修改方法实现

其中第二条值得注意，当 使用 then 系列方法，spy 对象的实际方法其实还是会被调用的，然后才执行指定的实现，所以有时使用 then 系列方法会产生异常，这时只能使用 do 系列方法（它会覆盖实际方法实现）。看下面这个例子：

```
val realList = mutableListOf<String>()
val spyList = spy(realList)
// stubbing success
`when`(spyList.size).thenReturn(5)
//Impossible: real method is called so spy.get(0) throws IndexOutOfBoundsException (the list
is yet empty)
`when`(spyList[0]).thenReturn("first")
//You have to use doReturn() for stubbing
doReturn("first").`when`(spyList)[0]
```

2. mockito-kotlin 库

Mockito 在 Kotlin 下使用时会有一些问题，例如 when 属于关键字， 在参数验证使用 `any()` 方法会返回 null， 在传给非空类型参数时 会出错。

Github 上有人已经解决了这个问题，nhaarman 写了一个 小而美的库 Mockito-Kotlin 帮助在 Kotlin 下方便地使用 Mockito。主要使用了顶层函数封装了 Mockito 的常用静态方法，如 mock()、any()、eq() 等。

Gradle 引入

```
testImplementation 'com.nhaarman:mockito-kotlin:x.x.x'
// 使用 Kotlin 1.1 时
testImplementation 'com.nhaarman:mockito-kotlin-
kt1.1:x.x.x'
```

下面是 mockito-kotlin 库 所带来的便利：

whenever 替换 when，避免与 Kotlin 关键字 冲突。

```
whenever(mock.stringValue()).thenReturn("
test")
```

创建 mock 或 spy 对象时， 如果类型可以推倒出来的话，不需要传类型

```
val mock: MyClass = mock()
// if type cannot be inferred
directly
val mock = mock<MyClass>()
// use mock object as
parameter
val instance =
MyClass(mock())
```

可以在 mock 对象时指定方法实现

```
val mock = mock<MyClass> {
   on { stringValue() } doReturn
"test"
}
```

对 Mockito 中 any()、eq() 这些 返回空的方法做了封装，当调用 `any()` 时，会先调用 `Mockito.any()` 更新验证状态，然后返回一个非空的值，避免空指针问题。

更多详细的 内容，可以阅读它的 wiki：mockito-kotlin Wiki

3. 小结

Mockito 框架和 mockito-kotlin 库让我们可以很方便地验证 void 方法的输出结果，即验证方法的调用。 但是 Mockito 框架有一些限制，不能 mock 静态方法，不能指定 final 方法的实现，不过这是利大于弊的， 让我们不会滥用静态方法，其实静态方法应该只存在于 Utils 工具类中。本文只是大体介绍了 Mockito 的主要概念的，具体使用过程中遇到一些问题，推荐大家阅读官方文档。

到目前为止，介绍的 JUnit 4 和 Mockito 测试框架，都是针对 Java 或 Kotlin 代码的测试，如果要在 JVM 中测试 Android 相关逻辑的话，需要利用到 Robolectric 测试框架，所以下一篇文章将介绍 Robolectric 的用法。

参考资料：

- Mockito API
- Unit tests with Mockito - Tutorial
- Android单元测试（四）：Mock以及Mockito的使用