# 使用 Mockito 单元测试 – 教程

### tanyuanji@126.com

|  |  |  |  |
| --- | --- | --- | --- |
| **版本历史** | | | |
| - | - | - | - |

## 使用 Mockito 进行测试

该教程主要讲解 Mockito 框架在Eclipse IDE 中的使用

**目录**

[使用 Mockito 单元测试 – 教程 1](#_Toc372637079)

[tanyuanji@126.com 1](#_Toc372637080)

[使用 Mockito 进行测试 1](#_Toc372637081)

[1. 需求知识 2](#_Toc372637082)

[2. 使用 存根(Stub) 和 模拟对象(Mock Object) 进行测试 2](#_Toc372637083)

[2.1. 为什么需要模拟？ 2](#_Toc372637084)

[2.2. 存根(Stub) vs. 模拟对象 (Mock) 2](#_Toc372637085)

[2.3. 行为测试 vs. 状态测试 4](#_Toc372637086)

[2.4. 生成模拟对象 4](#_Toc372637087)

[3. 模拟框架( Mock Framework) 5](#_Toc372637088)

[4. Mockito 6](#_Toc372637089)

[4.1. 使用 Mockito 模拟对象 6](#_Toc372637090)

[4.2. 使用 Mockito 6](#_Toc372637091)

[4.3. Mockito的限制 7](#_Toc372637092)

[4.4. 模拟对象的配置 7](#_Toc372637093)

[4.5. 验证模拟对象的行为 7](#_Toc372637094)

[4.6. Spy 7](#_Toc372637095)

[5. Mockito 在 Android 平台测试 9](#_Toc372637096)

[5.1. 在 Android 使用 Mockito 9](#_Toc372637097)

[5.2. 安装 9](#_Toc372637098)

[6. 感谢 10](#_Toc372637099)

[7. 链接和参考 10](#_Toc372637100)

[7.1. Mockito 相关链接 10](#_Toc372637101)

## 1. 需求知识

该教程需要理解单元测试和熟悉JUnit 框架的使用。

如果您不熟悉 JUnit，请阅读 JUnit 教程。

## 2. 使用 存根(Stub) 和 模拟对象(Mock Object) 进行测试

### 2.1. 为什么需要模拟？

一个单元测试需要在隔离的环境下执行。如果可以的话需要消除其他依赖的服务影响。但实际上,软件中是充满依赖关系的.我们会基于service类写操作类,而service类又是基于数据访问类(DAOs)的，依次下去.

为了解决这个问题, 可以使用 存根 (Stub) 或者 模拟 (Mock) 对象的方法进行测试。

### 2.2. 存根(Stub) vs. 模拟对象 (Mock)

存根(Stub)类是实现了一个接口或者抽象类的类，可以在测试过程中使用该类，例如：

**package** cn.nsccsz.stub;

**public** **class** TestStub {

**static** **interface** USB {

**void** work();

}

**static** **class** Mp3Stub **implements** USB {

@Override

**public** **void** work() {

// code.

}

}

**static** **class** Mp4Stub **implements** USB {

@Override

**public** **void** work() {

// code.

}

}

}

一个模拟对象(mock object)是一个接口或者抽象类的虚拟实现。例如：

**public** **class** TestMock {

**static** **interface** USB {

**void** work();

}

@Test

**public** **void** testMockObject() {

USB usb = Mockito.*mock*( USB.**class** );

usb.work();

}

}

存根和模拟对象都可以传递给其他的对象进行测试。你的一些单元测试可以测这些类的正确性等。利用存根对象或者模拟对象可以保证测试过程中不受到其他的影响。

存根对象需要自定义实现方法；

模拟对象只需要更少的代码和简单的配置。

以下的内容将详细介绍模拟对象的使用方法。

### 2.3. 行为测试 vs. 状态测试

Mock 对象允许你对行为进行测试。有一些测试不需要验证结果，但是需要检查某些方法是否被正确的参数调用过。这种测试为行为测试。

状态测试只是关注与结果是否正确，而行为测试能够判断一个应用调用结构以及层次。

### 2.4. 生成模拟对象

你们可以使用Mock 框架来生成模拟对象。Mock 框架允许你在运行期间创建对象，并且定义它的一些行为。

一个典型的例子就是使用模拟对象来模拟数据库DAO层。在生产环境上是使用运行的数据库，但是在单元测试环境中完全可以用模拟对象来模拟数据，确保单元测试的正确条件。这样就不需要依赖于外部的数据。

## 3. 模拟框架( Mock Framework)

比较流行的模拟框架有 EasyMock、jMock 和 Mockito。下面的列表是这些框架的链接。

# jMock

http://jmock.org/

# EasyMock

http://easymock.org/

# Mockito

http://code.google.com/p/mockito/

## 4. Mockito

### 4.1. 使用 Mockito 模拟对象

Mockito 是比较流行的模拟框架，可以与JUnit 联合起来测试。它允许你进行创建和配置模拟对象。

Mockito的官方网站: [**Mockito 主页**](http://code.google.com/p/mockito/).

### 4.2. 使用 Mockito

Mockito 支持使用 mock() 静态方法创建模拟对象。

同样也支持 @Mock注解方式,如果使用注解的方式，需要使用在初始化方法调用 MockitoAnnotation.InitMock( this ) 方法

For example, the following code demonstrate the usage of Mockito to test a class called ClassToTest.

例如，下面的例子就是使用 Mockito 进行对类 ClassToTest 的单元测试。

**public** **class** MockitoTest {

@Mock

MyDatabase databaseMock;

@Before

**public** **void** setUp() **throws** Exception {

MockitoAnnotations.*initMocks*(**this**);

}

@Test

**public** **void** testQuery() {

// 需要测试的类

ClassToTest t = **new** ClassToTest(databaseMock);

// 调用方法

**boolean** check = t.query("\* from t");

// 验证结果

*assertTrue*(check);

// 模拟对象是否调用了该方法

Mockito.*verify*( databaseMock ).query("\* from t");

}

#### 提示

可以使用静态导入方法调用方法 mock()

### 4.3. Mockito的限制

Mockito 以下的类型不能进行构造：

* 终态类(final classes)
* 匿名类(anonymous classes)
* 基本数据类型(primitive types)

### 4.4. 模拟对象的配置

Mockito 可以使用 verify() 方法来确认某些方法是否被调用过.

when(....).thenReturn(....) 结构可以为某些条件给定一个预期的返回值.

@Test

**public** **void** testList() {

List mock = Mockito.*mock*( List.**class** );

Mockito.*when*( mock.get( 0 ) ).thenReturn( 1 );

*assertEquals*( "预期返回1", 1, mock.get( 0 ) );

}

同样可以使用doReturn(object).when(kdskfsk).methodCall 结构

### 4.5. 验证模拟对象的行为

Mockito 跟踪了所有的方法调用和参数的调用情况。verify()可以验证方法的行为。

查看下面的例子：

@Test

**public** **void** testMap() {

Map mock = Mockito.*mock*( Map.**class** );

Mockito.*when*( mock.get( "city" ) ).thenReturn( "深圳" );

// test code

*assertEquals*( "城市测试", "深圳", mock.get( "city" ) );

Mockito.*verify*(mock).get( Matchers.*eq*( "city" ) );

Mockito.*verify*( mock, Mockito.*times*( 2 ) );

}

### 4.6. Spy

@Spy 或者方法 spy() 可以包含一个真实的对象. 每次调用，除非特出指定，委托给改真实对象的调用.

@Test

**public** **void** testSpy() {

// Lets mock a LinkedList

List list = **new** LinkedList();

list.add( "yes" );

List spy = Mockito.*spy*(list);

//You have to use doReturn() for stubbing

*assertEquals*( "yes", spy.get( 0 ) );

Mockito.*doReturn*("foo").when(spy).get(0);

*assertEquals*( "foo", spy.get( 0 ) );

}

@Test( expected = IndexOutOfBoundsException.**class**)

**public** **void** testSpy2() {

// Lets mock a LinkedList

List list = **new** LinkedList();

List spy = Mockito.*spy*(list);

// this would not work

// real method is called so spy.get(0)

// throws IndexOutOfBoundsException (list is still empty)

Mockito.*when*(spy.get(0)).thenReturn("foo");

*assertEquals*( "foo", spy.get( 0 ) );

}

The verifyNoMoreInteractions() allows you to check that no other method was called.

You also have the @InjectMocks annotation which tries to do constructor dependency injection based on the type. The following code is a slightly modified example from the Javadoc.

public class ArticleManagerTest {

*@Mock* private ArticleCalculator calculator;

*@Mock* private ArticleDatabase database;

*@Spy* private UserProvider userProvider = new ConsumerUserProvider();

// creates instance of ArticleManager

// and performs constructor injection on it

*@InjectMocks* private ArticleManager manager = new ArticleManager();

*@Test* public void shouldDoSomething() {

MockitoAnnotations.initMocks(this);

verify(database).addListener(any(ArticleListener.class));

}

}

## 5. Mockito 在 Android 平台测试

### 5.1. 在 Android 使用 Mockito

Mockito 同样也可以在安卓平台上进行测试。

### 5.2. 安装

在 Android 测试项目中使用 Mockito。添加下面的包到Android 测试项目的 *libs* 目录

https://mockito.googlecode.com/files/mockito-all-1.9.5.jar

http://dexmaker.googlecode.com/files/dexmaker-1.0.jar

http://dexmaker.googlecode.com/files/dexmaker-mockito-1.0.jar

After you make the libraries available you can use Mockito in your tests.

接下来可以在你的测试项目中使用 Mockito 。

## 6. 感谢

感谢阅读该教程:

## 7. 链接和参考

### 7.1. Mockito 相关链接

**Mockito 项目主页**

**[Mockito 的依赖注入功能](http://docs.mockito.googlecode.com/hg/org/mockito/InjectMocks.html" \t "_top)**

[**Unit tests with Mockito - Tutorial**](http://www.vogella.com/articles/Mockito/article.html)