

# [译] 使用强大的 Mockito 来测试你的代码



曾梓华 (/u/b7942bef8847) (+关注)

2016.07.22 10:43\* 字数 2744 阅读 6879 评论 0 喜欢 28 赞赏 1

(/u/b7942bef8847)

- 原文链接 : Unit tests with Mockito - Tutorial (<https://link.jianshu.com?t=http://www.vogella.com/tutorials/Mockito/article.html>)

- 原文作者 : vogella (<https://link.jianshu.com?t=http://www.vogella.com/>)
- 译者 : edvardhua (<https://link.jianshu.com?t=https://github.com/edvardHua/>)
- 校对者: hackerkevin (<https://link.jianshu.com?t=https://github.com/hackerkevin>), futureshine (<https://link.jianshu.com?t=https://github.com/futureshine>)

这篇教程介绍了如何使用 Mockito 框架来给软件写测试用例

## 1. 预备知识

如果需要往下学习，你需要先理解 JUnit 框架中的单元测试。

如果你不熟悉 JUnit，请查看下面的教程：

<http://www.vogella.com/tutorials/JUnit/article.html> (<https://link.jianshu.com?t=http://www.vogella.com/tutorials/JUnit/article.html>)

## 2. 使用mock对象来进行测试

### 2.1. 单元测试的目标和挑战

单元测试的思路是在不涉及依赖关系的情况下测试代码（隔离性），所以测试代码与其他类或者系统的关系应该尽量被消除。一个可行的消除方法是替换掉依赖类（测试替换），也就是说我们可以使用替身来替换掉真正的依赖对象。

### 2.2. 测试类的分类

*dummy object* 做为参数传递给方法但是绝对不会被使用。譬如说，这种测试类内部的方法不会被调用，或者是用来填充某个方法的参数。

*Fake* 是真正接口或抽象类的实现体，但给对象内部实现很简单。譬如说，它存在内存中而不是真正的数据库中。（译者注：*Fake* 实现了真正的逻辑，但它的存在只是为了测试，而不适合于用在产品中。）



*stub* 类是依赖类的部分方法实现，而这些方法在你测试类和接口的时候会被用到，也就是说 *stub* 类在测试中会被实例化。*stub* 类会回应任何外部测试的调用。*stub* 类有时候还会记录调用的一些信息。

*mock object* 是指类或者接口的模拟实现，你可以自定义这个对象中某个方法的输出结果。

测试替代技术能够在测试中模拟测试类以外对象。因此你可以验证测试类是否响应正常。譬如说，你可以验证在 Mock 对象的某一个方法是否被调用。这可以确保隔离了外部依赖的干扰只测试测试类。

我们选择 Mock 对象的原因是因为 Mock 对象只需要少量代码的配置。

## 2.3. Mock 对象的产生

你可以手动创建一个 Mock 对象或者使用 Mock 框架来模拟这些类，Mock 框架允许你在运行时创建 Mock 对象并且定义它的行为。

一个典型的例子是把 Mock 对象模拟成数据的提供者。在正式的生产环境中它会被实现用来连接数据源。但是我们在测试的时候 Mock 对象将会模拟成数据提供者来确保我们的测试环境始终是相同的。

Mock 对象可以被提供来进行测试。因此，我们测试的类应该避免任何外部数据的强依赖。

通过 Mock 对象或者 Mock 框架，我们可以测试代码中期望的行为。譬如说，验证只有某个存在 Mock 对象的方法是否被调用了。

## 2.4. 使用 Mockito 生成 Mock 对象

*Mockito* 是一个流行 mock 框架，可以和JUnit结合起来使用。*Mockito* 允许你创建和配置 mock 对象。使用*Mockito*可以明显的简化对外部依赖的测试类的开发。

一般使用 *Mockito* 需要执行下面三步

- 模拟并替换测试代码中外部依赖。
- 执行测试代码
- 验证测试代码是否被正确的执行

mockitousagevisualization

---

## 3. 为自己的项目添加 Mockito 依赖

### 3.1. 在 Gradle 添加 Mockito 依赖

如果你的项目使用 Gradle 构建，将下面代码加入 Gradle 的构建文件中为自己项目添加 Mockito 依赖

```
repositories { jcenter() }  
dependencies { testCompile "org.mockito:mockito-core:2.0.57-beta" }
```

### 3.2. 在 Maven 添加 Mockito 依赖

需要在 Maven 声明依赖，您可以在 <http://search.maven.org> (<https://link.jianshu.com?t=http://search.maven.org>) 网站中搜索 g:"org.mockito", a:"mockito-core" 来得到具体的声明方式。

### 3.3. 在 Eclipse IDE 使用 Mockito

Eclipse IDE 支持 Gradle 和 Maven 两种构建工具，所以在 Eclipse IDE 添加依赖取决于你使用的是哪一个构建工具。

### 3.4. 以 OSGi 或者 Eclipse 插件形式添加 Mockito 依赖

在 Eclipse RCP 应用依赖通常可以在 p2 update 上得到。Orbit 是一个很好的第三方仓库，我们可以在里面寻找能在 Eclipse 上使用的应用和插件。

Orbit 仓库地址 <http://download.eclipse.org/tools/orbit/downloads>  
(<https://link.jianshu.com?t=http://download.eclipse.org/tools/orbit/downloads>)

orbit p2 mockito

## 4. 使用Mockito API

### 4.1. 静态引用

如果在代码中静态引用了 `org.mockito.Mockito.*`，那你你就可以直接调用静态方法和静态变量而不用创建对象，譬如直接调用 `mock()` 方法。

### 4.2. 使用 Mockito 创建和配置 mock 对象

除了上面所说的使用 `mock()` 静态方法外，Mockito 还支持通过 `@Mock` 注解的方式来创建 mock 对象。

如果你使用注解，那么必须要实例化 mock 对象。Mockito 在遇到使用注解的字段的时候，会调用 `MockitoAnnotations.initMocks(this)` 来初始化该 mock 对象。另外也可以通过使用 `@RunWith(MockitoJUnitRunner.class)` 来达到相同的效果。

通过下面的例子我们可以了解到使用 `@Mock` 的方法和 `MockitoRule` 规则。

```
import static org.mockito.Mockito.*;

public class MockitoTest {

    @Mock
    MyDatabase databaseMock; (1)

    @Rule public MockitoRule mockitoRule = MockitoJUnit.rule(); (2)

    @Test
    public void testQuery() {
        ClassToTest t = new ClassToTest(databaseMock); (3)
        boolean check = t.query("* from t"); (4)
        assertTrue(check); (5)
        verify(databaseMock).query("* from t"); (6)
    }
}
```

1. 告诉 Mockito 模拟 databaseMock 实例
2. Mockito 通过 `@mock` 注解创建 mock 对象
3. 使用已经创建的mock初始化这个类
4. 在测试环境下，执行测试类中的代码
5. 使用断言确保调用的方法返回值为 true
6. 验证 query 方法是否被 MyDatabase 的 mock 对象调用

### 4.3. 配置 mock

当我们需要配置某个方法的返回值的时候，Mockito 提供了链式的 API 供我们方便的调用

`when(...).thenReturn(...)` 可以被用来定义当条件满足时函数的返回值，如果你需要定义多个返回值，可以多次定义。当你多次调用函数的时候，Mockito 会根据你定义的先后顺序来返回返回值。Mocks 还可以根据传入参数的不同来定义不同的返回值。譬如说

你的函数可以将 `anyString` 或者 `anyInt` 作为输入参数，然后定义其特定的返回值。

```
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;

@Test
public void test1() {
    // 创建 mock
    MyClass test = Mockito.mock(MyClass.class);

    // 自定义 getId() 的返回值
    when(test.getId()).thenReturn(43);

    // 在测试中使用mock对象
    assertEquals(test.getId(), 43);
}

// 返回多个值
@Test
public void testMoreThanOneReturnValue() {
    Iterator i= mock(Iterator.class);
    when(i.next()).thenReturn("Mockito").thenReturn("rocks");
    String result=i.next()+" "+i.next();
    // 断言
    assertEquals("Mockito rocks", result);
}

// 如何根据输入来返回值
@Test
public void testReturnValueDependentOnMethodParameter() {
    Comparable c= mock(Comparable.class);
    when(c.compareTo("Mockito")).thenReturn(1);
    when(c.compareTo("Eclipse")).thenReturn(2);
    // 断言
    assertEquals(1,c.compareTo("Mockito"));
}

// 如何让返回值不依赖于输入
@Test
public void testReturnValueIndependentOnMethodParameter() {
    Comparable c= mock(Comparable.class);
    when(c.compareTo(anyInt())).thenReturn(-1);
    // 断言
    assertEquals(-1 ,c.compareTo(9));
}

// 根据参数类型来返回值
@Test
public void testReturnValueIndependentOnMethodParameter() {
    Comparable c= mock(Comparable.class);
    when(c.compareTo(isA(Todo.class))).thenReturn(0);
    // 断言
    Todo todo = new Todo(5);
    assertEquals(todo ,c.compareTo(new Todo(1)));
}
```

对于无返回值的函数，我们可以使用 `doReturn(...).when(...).methodCall` 来获得类似的效果。例如我们想在调用某些无返回值函数的时候抛出异常，那么可以使用 `doThrow` 方法。如下面代码片段所示

```
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;

// 下面测试用例描述了如何使用doThrow()方法

@Test(expected=IOException.class)
public void testForIOException() {
    // 创建并配置 mock 对象
    OutputStream mockStream = mock(OutputStream.class);
    doThrow(new IOException()).when(mockStream).close();

    // 使用 mock
    OutputStreamWriter streamWriter= new OutputStreamWriter(mockStream);
    streamWriter.close();
}
```

#### 4.4. 验证 mock 对象方法是否被调用

Mockito 会跟踪 mock 对象里面所有的方法和变量。所以我们可以用来验证函数在传入特定参数的时候是否被调用。这种方式的测试称为行为测试，行为测试并不会检查函数的返回值，而是检查在传入正确参数时候函数是否被调用。

```
import static org.mockito.Mockito.*;

@Test
public void testVerify() {
    // 创建并配置 mock 对象
    MyClass test = Mockito.mock(MyClass.class);
    when(test.getUniqueId()).thenReturn(43);

    // 调用mock对象里面的方法并传入参数为12
    test.testing(12);
    test.getUniqueId();
    test.getUniqueId();

    // 查看在传入参数为12的时候方法是否被调用
    verify(test).testing(Matchers.eq(12));

    // 方法是否被调用两次
    verify(test, times(2)).getUniqueId();

    // 其他用来验证函数是否被调用的方法
    verify(mock, never()).someMethod("never called");
    verify(mock, atLeastOnce()).someMethod("called at least once");
    verify(mock, atLeast(2)).someMethod("called at least twice");
    verify(mock, times(5)).someMethod("called five times");
    verify(mock, atMost(3)).someMethod("called at most 3 times");
}
```

#### 4.5. 使用 Spy 封装 java 对象

@Spy或者 spy() 方法可以被用来封装 java 对象。被封装后，除非特殊声明（打桩 stub），否则都会真正的调用对象里面的每一个方法

```
import static org.mockito.Mockito.*;

// Lets mock a LinkedList
List list = new LinkedList();
List spy = spy(list);

// 可用 doReturn() 来打桩
doReturn("foo").when(spy).get(0);

// 下面代码不生效
// 真正的方法会被调用
// 将会抛出 IndexOutOfBoundsException 的异常, 因为 List 为空
when(spy.get(0)).thenReturn("foo");
```

方法 `verifyNoMoreInteractions()` 允许你检查没有其他的方法被调用了。

## 4.6. 使用 @InjectMocks 在 Mockito 中进行依赖注入

我们也可以使用 `@InjectMocks` 注解来创建对象, 它会根据类型来注入对象里面的成员方法和变量。假定我们有 `ArticleManager` 类

```
public class ArticleManager {
    private User user;
    private ArticleDatabase database;

    ArticleManager(User user) {
        this.user = user;
    }

    void setDatabase(ArticleDatabase database) { }
}
```

这个类会被 Mockito 构造, 而类的成员方法和变量都会被 mock 对象所代替, 正如下面的代码片段所示:

```
@RunWith(MockitoJUnitRunner.class)
public class ArticleManagerTest {

    @Mock ArticleCalculator calculator;
    @Mock ArticleDatabase database;
    @Most User user;

    @Spy private UserProvider userProvider = new ConsumerUserProvider();

    @InjectMocks private ArticleManager manager; (1)

    @Test public void shouldDoSomething() {
        // 假定 ArticleManager 有一个叫 initialize() 的方法被调用了
        // 使用 ArticleListener 来调用 addListener 方法
        manager.initialize();

        // 验证 addListener 方法被调用
        verify(database).addListener(any(ArticleListener.class));
    }
}
```

### 1. 创建ArticleManager实例并注入Mock对象

更多的详情可以查看

<http://docs.mockito.googlecode.com/hg/1.9.5/org/mockito/InjectMocks.html>

(<https://link.jianshu.com?>

t=<http://docs.mockito.googlecode.com/hg/1.9.5/org/mockito/InjectMocks.html>).

## 4.7. 捕捉参数

ArgumentCaptor 类允许我们在verification期间访问方法的参数。得到方法的参数后我们可以使用它进行测试。

```
import static org.hamcrest.Matchers.hasItem;
import static org.junit.Assert.assertThat;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;

import java.util.Arrays;
import java.util.List;

import org.junit.Rule;
import org.junit.Test;
import org.mockito.ArgumentCaptor;
import org.mockito.Captor;
import org.mockito.junit.MockitoJUnit;
import org.mockito.junit.MockitoRule;

public class MockitoTests {

    @Rule public MockitoRule rule = MockitoJUnit.rule();

    @Captor
    private ArgumentCaptor< > captor;

    @Test
    public final void shouldContainCertainListItem() {
        List asList = Arrays.asList("someElement_test", "someElement");
        final List mockedList = mock(List.class);
        mockedList.addAll(asList);

        verify(mockedList).addAll(captor.capture());
        final List capturedArgument = captor.>getValue();
        assertThat(capturedArgument, hasItem("someElement"));
    }
}
```

## 4.8. Mockito的限制

Mockito当然也有一定的限制。而下面三种数据类型则不能够被测试

- final classes
- anonymous classes
- primitive types

## 5. 在Android中使用Mockito

在 Android 中的 Gradle 构建文件中加入 Mockito 依赖后就可以直接使用 Mockito 了。若想使用 Android Instrumented tests 的话，还需要添加 dexmaker 和 dexmaker-mockito 依赖到 Gradle 的构建文件中。（需要 Mockito 1.9.5版本以上）



```
dependencies {
    testCompile 'junit:junit:4.12'
    // Mockito unit test 的依赖
    testCompile 'org.mockito:mockito-core:1.+'
    // Mockito Android instrumentation tests 的依赖
    androidTestCompile 'org.mockito:mockito-core:1.+'
    androidTestCompile "com.google.dexmaker:dexmaker:1.2"
    androidTestCompile "com.google.dexmaker:dexmaker-mockito:1.2"
}
```

## 6. 实例：使用Mockito写一个Instrumented Unit Test

### 6.1. 创建一个测试的Android 应用

创建一个包名为 `com.vogella.android.testing.mockito.contextmock` 的Android应用，添加一个静态方法

，方法里面创建一个包含参数的Intent，如下代码所示：

```
public static Intent createQuery(Context context, String query, String value) {
    // 简单起见，重用MainActivity
    Intent i = new Intent(context, MainActivity.class);
    i.putExtra("QUERY", query);
    i.putExtra("VALUE", value);
    return i;
}
```

### 6.2. 在app/build.gradle文件中添加Mockito依赖

```
dependencies {
    // Mockito 和 JUnit 的依赖
    // instrumentation unit tests on the JVM
    androidTestCompile 'junit:junit:4.12'
    androidTestCompile 'org.mockito:mockito-core:2.0.57-beta'
    androidTestCompile 'com.android.support.test:runner:0.3'
    androidTestCompile "com.google.dexmaker:dexmaker:1.2"
    androidTestCompile "com.google.dexmaker:dexmaker-mockito:1.2"

    // Mockito 和 JUnit 的依赖
    // tests on the JVM
    testCompile 'junit:junit:4.12'
    testCompile 'org.mockito:mockito-core:1.+'
}
```

### 6.3. 创建测试

使用 Mockito 创建一个单元测试来验证在传递正确 extra data 的情况下，intent 是否被触发。

因此我们需要使用 Mockito 来 mock 一个 Context 对象，如下代码所示：

```
package com.vogella.android.testing.mockitocontextmock;

import android.content.Context;
import android.content.Intent;
import android.os.Bundle;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mockito;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

public class TextIntentCreation {

    @Test
    public void testIntentShouldBeCreated() {
        Context context = Mockito.mock(Context.class);
        Intent intent = MainActivity.createQuery(context, "query", "value");
        assertNotNull(intent);
        Bundle extras = intent.getExtras();
        assertNotNull(extras);
        assertEquals("query", extras.getString("QUERY"));
        assertEquals("value", extras.getString("VALUE"));
    }
}
```

## 7. 实例：使用 Mockito 创建一个 mock 对象

### 7.1. 目标

创建一个 Api，它可以被 Mockito 来模拟并做一些工作

### 7.2. 创建一个 Twitter API 的例子

实现 `TwitterClient` 类，它内部使用到了 `ITweet` 的实现。但是 `ITweet` 实例很难得到，譬如说他需要启动一个很复杂的服务来得到。

```
public interface ITweet {

    String getMessage();
}

public class TwitterClient {

    public void sendTweet(ITweet tweet) {
        String message = tweet.getMessage();

        // send the message to Twitter
    }
}
```

### 7.3. 模拟 ITweet 的实例

为了能够不启动复杂的服务来得到 `ITweet`，我们可以使用 Mockito 来模拟得到该实例。

```
@Test
public void testSendingTweet() {
    TwitterClient twitterClient = new TwitterClient();

    ITweet iTweet = mock(ITweet.class);

    when(iTweet.getMessage()).thenReturn("Using mockito is great");

    twitterClient.sendTweet(iTweet);
}
```

现在 `TwitterClient` 可以使用 `ITweet` 接口的实现，当调用 `getMessage()` 方法的时候会打印 "Using Mockito is great" 信息。

## 7.4. 验证方法调用

确保 `getMessage()` 方法至少调用一次。

```
@Test
public void testSendingTweet() {
    TwitterClient twitterClient = new TwitterClient();

    ITweet iTweet = mock(ITweet.class);

    when(iTweet.getMessage()).thenReturn("Using mockito is great");

    twitterClient.sendTweet(iTweet);

    verify(iTweet, atLeastOnce()).getMessage();
}
```

## 7.5. 验证

运行测试，查看代码是否测试通过。

## 8. 模拟静态方法

### 8.1. 使用 Powermock 来模拟静态方法

因为 Mockito 不能够 mock 静态方法，因此我们可以使用 Powermock。

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public final class NetworkReader {
    public static String getLocalHostname() {
        String hostname = "";
        try {
            InetAddress addr = InetAddress.getLocalHost();
            // Get hostname
            hostname = addr.getHostName();
        } catch ( UnknownHostException e ) {
        }
        return hostname;
    }
}
```

我们模拟了 `NetworkReader` 的依赖，如下代码所示：

```
import org.junit.runner.RunWith;
import org.powermock.core.classloader.annotations.PrepareForTest;

@RunWith( PowerMockRunner.class )
@PrepareForTest( NetworkReader.class )
public class MyTest {

    // 测试代码

    @Test
    public void testSomething() {
        mockStatic( NetworkUtil.class );
        when( NetworkReader.getLocalHostname() ).andReturn( "localhost" );

        // 与 NetworkReader 协作的测试
    }
}
```

## 8.2.用封装的方法代替Powermock

有时候我们可以在静态方法周围包含非静态的方法来达到和 Powermock 同样的效果。

```
class FooWrapper {
    void someMethod() {
        Foo.someStaticMethod()
    }
}
```

## 9. Mockito 参考资料

<http://site.mockito.org> (<https://link.jianshu.com?t=http://site.mockito.org>) - Mockito 官网

<https://github.com/mockito/mockito-> (<https://link.jianshu.com?t=https://github.com/mockito/mockito->) Mockito Github

<https://github.com/mockito/mockito/blob/master/doc/release-notes/official.md>  
(<https://link.jianshu.com?t=https://github.com/mockito/mockito/blob/master/doc/release-notes/official.md>) - Mockito 发行说明

<http://martinfowler.com/articles/mocksArentStubs.html> (<https://link.jianshu.com?t=http://martinfowler.com/articles/mocksArentStubs.html>) 与 Mocks, Stub有关的文章

<http://chiuki.github.io/advanced-android-espresso/> (<https://link.jianshu.com?t=http://chiuki.github.io/advanced-android-espresso/>) 高级android教程 (竟然是个妹子)


小礼物走一走，来简书关注我

赞赏支持

(/u/4a4eb4feee62)

📖 软件工程 (/nb/5012049)

举报文章 © 著作权归作者所有



曾梓华 (/u/b7942bef8847) ♂

写了 30483 字, 被 249 人关注, 获得了 241 个喜欢

</u/b7942bef8847>

+ 关注

虎牙直播机器学习工程师一枚。不定期总结和分享工作上遇到的问题。不定期分享前沿技术。交流想法...

喜欢 | 28



更多分享

(http://cwb.assets.jianshu.io/notes/images/4725867/w)



下载简书 App ▶

随时随地发现和创作内容



(/apps/download?utm\_source=nbc)



登录 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-comment-form)

评论

智慧如你, 不想发表一点想法 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-nocomments-text)  
咩~

被以下专题收入, 发现更多相似内容

Android... (/c/5139d555c94d?utm\_source=desktop&utm\_medium=notes-included-collection)

软件测试 (/c/3f7695959dda?utm\_source=desktop&utm\_medium=notes-included-collection)

Android开发 (/c/0dc880a2c73c?utm\_source=desktop&utm\_medium=notes-included-collection)

android... (/c/eb5e5dc587f1?utm\_source=desktop&utm\_medium=notes-included-collection)

程序员 (/c/NEt52a?utm\_source=desktop&utm\_medium=notes-included-collection)

iOS && ... (/c/2b91f4eeade5?utm\_source=desktop&utm\_medium=notes-included-collection)

JAVA (/c/60548045623d?utm\_source=desktop&utm\_medium=notes-included-collection)

展开更多 ∨

推荐阅读

更多精彩内容 > (/)

### 关联规则挖掘算法 (/p/7d459ace31ab?utm\_campaign=...

(/p/7d459ace31ab?

关联规则挖掘是一种基于规则的机器学习算法，该算法可以在大数据库中发现感兴趣的关系。它的目的是利用一些度量指标来分辨数据库中存在的强规则。

utm\_campaign=maleskine&utm\_content=note&utm\_m

曾梓华 (/u/b7942bef8847?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

### [译] 简明 TensorFlow 教程 — 第三部分: 所有的模型 (/...

(/p/67977f1fa535?

原文地址: TensorFlow in a Nutshell—Part Three: All the Models 原文作者: Camron Godbout 译者: edvardhua 校对者: marcmoore, cdpath 简明

utm\_campaign=maleskine&utm\_content=note&utm\_m

曾梓华 (/u/b7942bef8847?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

### 个人品牌之死 (/p/25b95cb07a22?utm\_campaign=...

(/p/25b95cb07a22?

1. 小K是一个我非常喜欢的手绘达人，今天下午我看到她在朋友圈和我的社群微信群发了一张海报和这样一段话：【免费赠送书友】转发海报到朋友圈或者

utm\_campaign=maleskine&utm\_content=note&utm\_m

彭小六 (/u/1441f4ae075d?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

### 杀手没有性生活 (/p/b83c320ce0f6?utm\_campaign=...

(/p/b83c320ce0f6?

1 漆七刚入行的时候，师傅就跟他说过，干这一行，有两样东西不能沾。一是酒，二是色。酒能麻痹人的神经，打乱人的性情，色会腐蚀人的心智，降低人

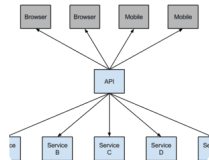
utm\_campaign=maleskine&utm\_content=note&utm\_m

黄咚咚 (/u/cb272a36d4d7?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

越长大越明白，取悦自己真的是件很酷的事情 (/p/05c... (/p/05c9261f753a?  
我们认为最好的生命状态是合群，不愿意把自己打磨得熠熠闪光。即使是金子 utm\_campaign=maleskine&utm\_content=note&utm\_m  
也只能甘于平庸，混然于世。01 为了生存，我们选择合群，但不要忘了迎  
羊达令 (/u/ce94d617e045?  
utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)


(/p/46fd0faecac1?



utm\_campaign=maleskine&utm\_content=note&utm\_medium=seo\_notes&utm\_source=recommendation)

**Spring Cloud (/p/46fd0faecac1?utm\_campaign=maleskine&utm\_cont...**

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智能路由，微代理，控制总线）。分布式系统的协调导致了样板模式，使用Spring Cloud开发人员

 卡卡罗2017 (/u/d90908cb0d85?  
utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)


**Android单元测试（四）：Mock以及Mockito的使用 (/p/b6e0cf81641b?...**

几点说明：代码中的 //<== 表示跟上面的相比，这是新增的，或者是修改的代码，不知道怎么样在代码块里面再强调几行代码\_T.T。。。很多时候，为了避免中文歧义，我会用英文表述 在第一篇文章里面我们提

 邹小创 (/u/b9c5c66b6a08?  
utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

**从百变怪Mockito到单元测试 (/p/88403716c5c2?utm\_campaign=males...**

一、百变怪 Mockito Mockito可谓是Java世界的百变怪，使用它，可以轻易的复制出各种类型的对象，并与之进行交互。1.1 对象“复制” 1.2 技能复制 虽然复制出来的对象上的所有方法都能被调用，但好像这个百

 罗力 (/u/e83d53e769d0?  
utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

(/p/b4d6e9bbcfce?




utm\_campaign=maleskine&utm\_content=note&utm\_medium=seo\_notes&utm\_source=recommendation)

**Android自动化测试 (/p/b4d6e9bbcfce?utm\_campaign=maleskine&ut...**

Instrumentation介绍 Instrumentation是个什么东西？ Instrumentation测试 Instrumentation原理介绍

一、Instrumentation是个什么东西？ Instrumentation是位于android.app包下，与...

 qiangzier (/u/09547a87c1c9?  
utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)


(/p/9f7a992fe9ec?



utm\_campaign=maleskine&utm\_content=note&utm\_medium=seo\_notes&utm\_source=recommendation)

## Android单元测试在蘑菇街支付金融部门的实践 (/p/9f7a992fe9ec?utm\_c...

这是我在InfoQ组织的移动开发前线微信群里面做的一次分享，发到简书上面一份，希望给在简书有需要的读者也能提供一点帮助，以下是原文：大家好，我是蘑菇街支付金融部门的小创。今天很高兴跟大家分享

 邹小创 (/u/b9c5c66b6a08?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

(/p/3c2452644e5a?


240



utm\_campaign=maleskine&utm\_content=note&utm\_medium=seo\_notes&utm\_source=recommendation)

## OTT/IPTV互联网电视运营平台 (/p/3c2452644e5a?utm\_campaign=mal...


一、产品介绍：800Li OTT-TV System是基于开放互联网的視頻服务系统，终端可以是电视机、电脑、机顶盒、PAD、智能手机、智能投影仪等等。OTT-TV通过互联网传输的視頻节目，不受物理网络局限，全球

 微微笑i (/u/33e0c94c5bfe?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

## 黄饭与白饭 (/p/5161b61925b6?utm\_campaign=maleskine&utm\_conte...


春桃一天要吃两碗饭。一碗是净米，白丝丝的看着好看，一碗是拌了酱油的，黄澄澄的闻着诱人。吃饭很重要，尤其是对春桃这样不太专注生活里其余乐趣的人来说。且，将温饱与喜好合二为一，对于很多人来

 深耀 (/u/1f271f8ebac8?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

## 戒撸 (/p/d4475085e33b?utm\_campaign=maleskine&utm\_content=no...


8.14晴 今天去玩电脑好不爽，一玩游戏就不想站起来，现在到了最关键的阶段，而且自己的理解能力并不好，周日应该待在教室温习一下这周内容的，不能再去了，至少这个月不能玩游戏了，以后就是去网

 愿明天安好 (/u/3ee86844cc75?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

## 网易云歌单 (/p/61e2edd8417f?utm\_campaign=maleskine&utm\_conten...


A面我自己的 B面网上的 A面 <http://music.163.com/#/m/playlist?id=130045209&userid=106976093> 去您妈的 真麻烦 不分享了再见

 LunarShade (/u/152d2b016dc5?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)

## 如何建立个人品牌? (/p/167d419723d5?utm\_campaign=maleskine&ut...

一、个人品牌的“品”什么是职业？职业就是通过满足他人需求获得恰当收益，先理解职业的本质是社会交换，再理解个人品牌。把自己当成品牌，把自己交换出去，从而体现出来的价值。网上那些解释不对的，

 Darren\_Lin (/u/22ab9c8d11e8?

utm\_campaign=maleskine&utm\_content=user&utm\_medium=seo\_notes&utm\_source=recommendation)