Simple Unit Testing

Created by

Zhang Kai (@WalterInSH)

Zhang Wenbin (@zhwbqd)

Code

https://github.com/WalterInSH/unit-testing

更全的示例

由于本文受排版限制，只截取示例代码的一部分，点击每页链接可以查看更全示例

页数

北京同事注意右下角页数，我们保持同步

一个常见错误

```
@Test
public void case() throws Exception {
  int correctResult = Method.returnsOne();
  System.out.println("My result is " + correctResult);

  int incorrectResult = Method.shouldHaveReturnedOne();
  System.out.println("My result is " + incorrectResult);
}
```

```
public static void main(String[] args){
  int correctResult = Method.returnsOne();
  System.out.println("My result is " + correctResult);
}
```

基础

单元测试的名字一般为 `*Test.java` `Test*.java`

`*TestCase.java`

Maven默认会运行test/src目录下所有符合上述名称规则的测试类

```
public class BasicAssertionTest {
    @Test
    public void exampleOne() throws Exception {
        Assert.assertNull(Method.returnsNull());

        Assert.assertTrue(Method.returnsTrue());
        Assert.assertFalse(!Method.returnsTrue());

        Assert.assertEquals(Method.returnsOne(), 1);
        Assert.assertNotEquals(Method.returnsOne(), 2);
    }
}
```

使用Java static import省去 `Assert.`，让代码更易读

```
    @Test
    public void useStaticImport() throws Exception {
        assertNull(Method.returnsNull());

        assertTrue(Method.returnsTrue());
        assertFalse(!Method.returnsTrue());

        assertEquals(Method.returnsOne(), 1);
        assertNotEquals(Method.returnsOne(), 2);
    }
```

我的方法应该抛一个异常

```
@Test(
    expectedExceptions = IllegalArgumentException.class,
    expectedExceptionsMessageRegExp = "message")
public void expectAnExceptionWithAMessage()
    throws Exception {
    Method.onlyAcceptPositiveNum(-1);
}
```

为同一测试准备多份测试数据

```
@DataProvider(name = "number set")
public Object[][] numberSet(){
    return new Object[][] {
            {"-1"},
            {"123"},
            {"1.1"}};
}

@Test(dataProvider = "number set")
public void test(String data) throws Exception {
...
}
```

单元测试的热身和收尾

```
@BeforeClass
public void beforeClass() throws Exception {
    System.out.println("Running code before class");
}

@BeforeMethod
public void beforeMethod() throws Exception {
    System.out.println("Running code before method");
}

@AfterMethod
public void afterMethod() throws Exception {
    System.out.println("Running code after method");
}

@AfterClass
public void afterClass() throws Exception {
    System.out.println("Running code after class");
}
```

让单元测试有序进行

```
    @Test(priority = 1)
    public void testOne() throws Exception {
        SharedClass.sharedFiled = 2;
        assertEquals(SharedClass.sharedFiled, 2);
    }

    @Test(priority = 0)
    public void testTwo() throws Exception {
        assertEquals(SharedClass.sharedFiled, 0);
    }
```

Google Truth

像读书一样读单元测试

> Google Truth is an open source, fluent testing framework for Java that is designed to make your test assertions and failure messages more readable, as well as other benefits.

```
String string = "awesome";

assertThat(string).startsWith("awe");

assertWithMessage("Without me, it's just aweso")
    .that(string).contains("me");

Iterable<Color> googleColors = googleLogo.getColors();
assertThat(googleColors)
    .containsExactly(BLUE, RED, YELLOW)
    .inOrder();
```

Google Truth's faulure message

```
Not true that <[1, 3, 2]> is ordered <3> <2>

Not true that <{Tom=Level 6, Jim=Level 6}> contains
entry <Jim=Level 1>
```

更多Google Truth示例

Behavior-driven development

TDD and BDD Differences

- BDD is from customers point of view and focuses on excpected behavior of the whole system.
- TDD is from developpers point of view and focuses on the implementation of one unit/class/feature. It benefits among others from better architecture (Design for testability, less coupling between modules).
- From technical point of view (how to write the "test") they are similar.

```
public class Story {

    @Given("give a $parameter")
    public void init(String parameter)
        throws Exception {}

    @When("do something with $input")
    public void doSomething(String input)
        throws Exception {}

    @Then("I should get $result")
    public void shouldGet(String expectedResult)
        throws Exception {}
}
```

```
Scenario: need uppercase
Given give a parameter
When do something with input
Then I should get result
```

一个可以运行的Jbehave 例子

https://github.com/WalterInSH/unit-testing/tree/master/src/test/java/io/github/walterinsh/bdd

https://github.com/WalterInSH/unit-testing/blob/master/src/test/resources/io/github/walterinsh/bdd/string_reverser.story

Mock

定制测试场景

定制不同返回值的场景

```
TargetClass target = Mockito.mock(TargetClass.class);

//it means: when we invoke the method with parameter ""
//it should return "A".
Mockito.when(target.targetMethod(""))
    .thenReturn("A");

assertEquals(target.targetMethod(""), "A");
```

更多示例

想知道断网会发生什么？

```
doThrow(new SocketTimeoutException("no response"))
    .when(httpService).callRemoteAPI();
```

你必须执行2次，不能多，不能少

很多时候，得到正确的结果并不意味着过程是对的。例如如何验证一个缓存第二次读取时没有读数据库。

```
TargetClass targetClass = mock(TargetClass.class);

targetClass.targetMethod(null);
targetClass.targetMethod("A");

verify(targetClass, times(1)).targetMethod("A");
verify(targetClass, times(2)).targetMethod(anyString());
```

更多示例

Didn't you get the memo?

```java
@Test
public void trackParam() throws Exception {
    ArgumentCaptor<CharSequence> captor =
        ArgumentCaptor.forClass(CharSequence.class);

    TargetClass targetClass = mock(TargetClass.class);

    targetClass.targetMethod("parameter");

    verify(targetClass).targetMethod(captor.capture());

    List<CharSequence> capturedValues =
        captor.getAllValues();

    assertEquals(capturedValues.get(0), "parameter");
}
```

示例一

https://github.com/WalterInSH/unit-testing/blob/master/src/test/java/io/github/walterinsh/realworld/SuperMarketTest.java

WireMock

其实, 我是个Http服务

服务器，请给我来一个 500

```
stubFor(get(urlEqualTo("/api"))
    .willReturn(aResponse()
    .withStatus(500)));

String data = httpAPIService.requestData();
assertEquals(data, "");
```

更多例子

和Spring 集成

和Spring做朋友

把Spring 的豆子准备好

```
@ContextConfiguration(locations =
    "classpath:spring/application-context.xml")
public class Test
    extends AbstractTestNGSpringContextTests{

    @Autowired
    private SpringBean bean;

    @Test
    public void testBeanLoaded() throws Exception {
        assertNotNull(bean);
    }
}
```

OMG！AOP抢走了我的豆子

```
@Autowired
private AOPSpringBean bean;

@Test
public void testBeanLoaded() throws Exception {
    assertNotNull(bean);

    AOPSpringBean realBean =
        AopTestUtils.getTargetObject(bean);
}
```

完整示例

内存数据库

让我们从头开始

直接基于真实数据库测试的好处

- 运行环境更真实
- 完全不需要因为单元测试而改变dao层代码（H2语法偶有不同）
- 支持事务

直接基于真实数据库测试的劣势

- 每次单元测试的环境不可控
- 单元测试会产生脏数据
- 单元测试甚至可能改变数据库结构
- 极端情况下可能出现多个单元测试间冲突（一次提交触发多次同步构建）

H2

The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- ~~Browser based Console application~~(never seen people use this)
- Small footprint: around 1.5 MB jar file size

```
<jdbc:embedded-database id="dataSource" type="H2">
  <!--schema file defines your tables-->
  <jdbc:script location="h2_schema.sql"/>
  <jdbc:script location="h2_test_data.sql"/>
</jdbc:embedded-database>
```

schema

```
CREATE TABLE student (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(64) NOT NULL,
  PRIMARY KEY (id)
);
```

test data

```
INSERT INTO student(name) VALUES('Jim'),('Jade');
```

MongoDB也可以

```
<dependency>
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo</artifactId>
    <version>1.50.5</version>
</dependency>
```

Before Q & A

How to test a class that has private methods, fields or inner classes?

The best way to test a private method is via another public method. If this cannot be done, then one of the following conditions is true:

- The private method is dead code
- There is a design smell near the class that you are testing
- The method that you are trying to test should not be private

Is Unit Testing worth the effort?

Unit tests and test-driven development (TDD) have so many hidden and personal benefits as well as the obvious ones that you just can't really explain to somebody until they're doing it themselves.

👇

1. Unit Tests allows you to make big changes to code quickly. You know it works now because you've run the tests, when you make the changes you need to make, you need to get the tests working again. This saves hours.
2. TDD helps you to realise when to stop coding. Your tests give you confidence that you've done enough for now and can stop tweaking and move on to the next thing.
3. Unit Tests give you instant visual feedback, we all like the feeling of all those green lights when we've done. It's very satisfying.
4. Contrary to popular belief unit testing does not mean writing twice as much code, or coding slower. It's faster and more robust than coding without tests once you've got the hang of it. Test code itself is usually relatively trivial and doesn't add a big overhead to what you're doing. This is one you'll only believe when you're doing it :)

5. **Imperfect tests, run frequently, are much better than perfect tests that are never written at all**
6. Good unit tests can help document and define what something is supposed to do

要追求高单元测试覆盖率吗

需要的场景

- 单元测试的初学者（变熟练，养成习惯）
- 需要持续维护的外包项目（对质量的强制保证，开发维护人员变更）

不需要的场景

- 熟练且有良好习惯的

什么单元测试一定要写

1. 核心代码（无论复杂与否）
2. 易出错代码
3. 经常变动的代码
4. 可能要重构的代码
5. 线上bug

什么单元测试可以不写

1. 简单读写逻辑
2. 有集中异常捕获的常见参数异常（需要测集中异常捕获）
3. 绝对不会发生的情况

Junit or Testng

Junit和Testng都已经是十分成熟的单元测试框架，基本的功能上两者没有差距。但是在细节上Testng更胜一筹。

总之，**尽量使用Testng**.

One more thing

Marp

Marp is the simplest presentation writer with Markdown.

- Create slides with Markdown
- Cross-platform
- Live preview with 3 modes
- Theme supports (Default & Gaia theme)
- Export slides as PDF!

Enjoy testing! 👍

https://github.com/WalterInSH/unit-testing/