链滴

JUnit4 快速入门

作者：Zing

原文链接：https://ld246.com/article/1457751464812

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

引言

接触JUnit好长一段时间了，然而只是在工程中画葫芦，并没有好好地学过。借这个机会总结学习一，也方便有需要的同学快速入门。

JUnit是最常见且易用的用于单元测试的框架。JUnit4之前对测试类测试方法命名限制比较多，到了4入了注解，一切变得更加灵活，甚至无需继承TestCase。只需要导入jar包，注解会帮我们解决一切虽然在SSH中我更喜欢手写配置，然而不得不承认注解会减轻大量的工作）。

执行顺序

对于每一个测试类，都会有这样的一个执行流程：

BeforeClass->(Before->TestMethod1->After)->(Before->TestMethod2->After)->...->AfterClas

先普及以下Before和After的含义（在4之前更多称为SetUp和TearDown），Before为抽取出来的当行该测试类中所有测试方法之前都必须要的相同操作，如初始化、数据准备等等，Before在每个测试法执行前都会执行，而BeforeClass只会在所有方法执行前执行一次。After为抽出出来的当执行该测类中所有测试方法之后都必须要的相同操作，如清理脏数据等等，After在每个测试方法执行之后都执行，而AfterClass只会在所以方法执行完成后执行一次。

JUnit常用注解

在JUnit4中对类名称和方法名称不再限制，可以根据各人的编程习惯命名，只需要加上对应的注解就以了。

方法注解

- @BeforeClass

该方法必须是public static void

在测试类中所有测试方法执行之前，只执行一次该方法。例如创建数据库连接或读取文件等。

- @Before

该方法必须是public void，不允许为static

在该类中每个测试方法执行之前都会执行该方法，主要用于测试数据准备。

- @Test

该方法必须是public void，不允许为static，可以抛出异常

  - @Test(expected=XXXException.class)：测试抛出异常，在4之前只能通过catch中手动fail来试异常。
  - @Test(timeout=1000)：限时测试，若在指定时间内未执行完成，则认为测试失败。

- @Ignore("message")

该注解标记的测试方法会被忽略。可以备注为什么忽略的message。

- @After

该方法必须是public void，不允许为static

在该类中每个测试方法执行之后都会执行该方法，主要用于脏数据清洗。

- @AfterClass

该方法必须是public static void

在测试类中所有测试方法执行之后，只执行一次该方法。例如释放IO连接资源，恢复现场等。

- @Parameters

该方法必须是public static Collection，无传入参数

用于参数化功能，该注解用于准备数据的方法。

类注解

@RunWith注解放在测试类名之前，用于指定该测试类的运行方式，若不使用注解，则采用默认注解RunWith(JUnit4.class)

常见的运行器：

- 默认运行器　JUnit4.class：

JUnit4的默认运行器，可缺省

- 兼容运行器　JUnit38ClassRunner.class：

兼容JUnit3.8版本的运行器

- 打包测试运行器　Suite.class：

JUnit测试的入口，用于指明需要执行的测试类。

```
@RunWith(Suite.class)
@SuiteClasses({ATest.class,BTest.class,CTest.class})
public class whatever{}
```

- 参数化运行器　Parameterized.class：

我们经常会遇到这样的需求对同一个测试方法需要执行多组测试数据。面对这样的需求我们可以方便使用参数化运行期和@Parameters 注解配合使用测试集功能。

```
//0.指定使用参数化运行期
@RunWith(Parameterized.class)
public class OpTest {
    private static Op op = new Op();
    //1.声明用于存放参数和期望值的变量
    private int param;
    private int result;

    //2.构造函数，对1声明的变量进行初始化
    public OpTest(int param, int result) {
        this.param = param;
        this.result = result;
    }
    // 3.使用@Parameter方法准备数据
    @Parameters
```

```
    public static Collection data() {
        return Arrays.asList(new Object[][]{{1, 1},{0, 0},{1, -1}});
    }
    //4.在测试方法中直接使用1定义的变量作为参数
    @Test
    public void testAdd() {
        assertEquals(result, op.add(param));
    }
}
```

断言

断言是JUnit用于检测actuals是否符合expecteds的重要方法，若符合则意味着该测试用例通过，否则不通过，会产生相应的报告。

message是每个方法的可选字段，当检测失败后，message会在报告中展示。

相当于伪代码段：

```
if xxx:
    do nothing
else:
    report fail(message)
```

- assertEquals([String message,] expected,actual)：断言两个对象相等
- assertNotEquals([String message,] expected,actual)：断言两个对象不相等
- assertTrue ([String message,] condition)：断言condition为真
- assertFalse([String message,] condition)：断言condition为假
- assertNull([String message,] object)：断言object为空
- assertNotNull([String message,] Object object)：断言object不为空
- assertArrayEquals([String message,] expecteds, actuals)：断言两个数组相等
- assertSame([String message,]expected, actual)：断言两个对象的引用相等
- assertNotSame([String message,]expected, actual)：断言两个对象的引用不相等
- fail([String message])：直接让测试失败
- assertThat([String message,] actual, matcher)：断言actual满足matcher

that是JUnit4的新断言方法，eg： `assertThat(result, greaterThan(100));`

that的具体使用方法可以参考狂奔的瘦子