基于多线程并发的 JUnit 与 TestNG 测试框架比较①

汪添生[1,2], 马朝晗[3], 崔 蔚[1], 刘 迪[1,4], 赵俊峰[5], 夏 飞[5]

[1](国网信息通信产业集团有限公司, 北京 100761)
[2](厦门亿力吉奥信息科技有限公司, 厦门 361000)
[3](国网信通产业集团亿力科技公司, 北京 100053)
[4](北京中电普华信息技术有限公司, 北京 100085)
[5](国网江苏省电力公司通信分公司，南京 210008)

摘 要: 在多线程并发测试需求面前, Junit 和 TestNG 这两个简单的单元测试框架一直被拿来作比较, 而用户一般更偏向于配置简单、灵活、易操作且满足测试要求的测试框架. 本文对 Junit 和 TestNG 两种多线程并发测试方式进行实验对比, 目的在于验证两种框架对多线程并发测试的实现方式, 为用户在实际测试场景中选择工具提出了依据.

关键词: 多线程; 并发测试; JUnit; TestNG

Comparison of JUnit and TestNG Testing Framework Based on Concurrent Multi-Thread

WANG Tian-Sheng[1,2], MA Zhao-Han[3], CUI Wei[1], LIU Di[1,4], ZHAO Jun-Feng[5], XIA Fei[5]

[1](State Grid Information & Telecommunication Co. Ltd., Beijing 100761, China)
[2](Xiamen Great Power Geo Information Technology Co. Ltd., Xiamen 361000, China)
[3](Great Power Science and Technology Corporation State Grid Information & Telecommunication Group, Beijing100053, China)
[4](Beijing China-Power Information Technology Co. Ltd., Beijing 100085, China)
[5](State Grid Jiangsu Electric Power Company & Telecommunication Branch, Nanjing 210008, China)

Abstract: In front of multi-threaded concurrent test requirements, Junit and TestNG these two simple unit test framework has been used to compare, but users generally prefer to the test framework that configuration is simple, flexible, and easy to operate and satisfied the test requirements. This thesis has made an experimental comparison of Junit and TestNG on multi-threaded concurrent test methods, to verify the two frameworks implementation for multithreaded concurrent test, and provides the basis for the user to select the tool in the actual test scenario for users.

Key words: multi-thread; concurrent testing; JUnit; TestNG

随着社会的信息化气味越来越浓重, 催促着 IT 业的快速发展, 软件的需求已经融入各个行业, 用来辅助业务的信息化, 规范化和正规化发展. 软件的测试技术将首先解决软件开发过程的各种问题, 而软件单元测试在编码完成后用来对软件接口等方法进行验证, 是整个软件生命周期中一个必不可少的环节, 降低软件使用过程中的出错率[1]. 软件开发人员更喜欢在软件投入集成测试前进行自我验证, 减少软件的缺陷, 但这其实更是一个理所应当的步骤, 后面进行集成测试或者系统测试的测试人员都不会花时间去测试一个编码本身都存在问题的软件.

大多数的在单元测试主要由开发人员来进行, 一般主要包括五个方面的测试任务: 边界条件测试、模块中所有独立路径测试、模块的各条错误处理通路测试、局部数据结构测试、模块接口测试[2]. 这些测试任务中涉及到的代码覆盖率、重复率、并发性等问题, 促使一些优秀自动化测试框架产品的诞生, 方便了开发人员的单元测试工作. 目前比较常用的自动化测试框架有 C++的 CppUnit、CXXTest, Java 的 Junit、TestNG 等[3-5]. 针对 JAVA 语言开发的软件单元测试, Junit 和 TestNG 在集成开发工具 Eclipse 中提供了插件的支持, 受到很开发人员的喜爱, 它提高了单元测试的效率, 大大节约的开发人员编写测试用例的时间, 通过最后测试分析, 更容易发现编码当中存在缺陷[6-11].

① 收稿时间:2016-09-02;收到修改稿时间:2016-11-07 [doi:10.15888/j.cnki.csa.005774]

1 测试工具多线程简介

在最新的 Junit 版本 Junit4 中已经不需要测试方法以 Test 开头, 不用再继承 TestCase 类等, 同时此版本提供了注解, 注解的使用简化的测试用例的代码量, 提高测试的效率, 但 Junit4 本身支持多线程并发测试, 却没有提供的相应注解, 相比 Junit4, TestNG 在多线程测试方面则提供了相应的注解. 最简单的方式是在要测试的方法前加上注解, 如:

@Test(threadPoolSize = 10, invocationCount = 3, timeOut = 1000), 通过在@Test 注解中配置 threadPoolSize 这个属性来进入多线程模式的. 属性 threadPoolSize 的值代表该测试方法将会在 10 个不同的线程中同时执行, 属性 invocationCount 配置的是该测试方法将被执行的总次数, timeOut 的值将规定每次执行该测试方法所耗费时间的阈值, 超时则测试失败[12-14], 这是最简单的方法级别的多线程配置.

1.1 Junit 多线程简介

对于传统的测试方式, 大家都会使用 main 方法来执行测试的方法, 对于要执行多线程的方法, 创建一个线程类, 再循环执行线程类, 以此来做到测试多线程并发. Junit4 支持通过在要进行测试多线程的类前加上注解: @RunWith(TestRunner.class), TestRunner 是自定义的类, 继承了 BlockJUnit4ClassRunner, 而扩展的 TestRunner 内部则实现了对多线程的定义, 使得加上了该注解的类按 TestRunner 中的实现来跑多线程.

尽管 Junit4 对比之前几个版本的 Junit, 最大的一个亮点是提供了注解, 方便很多测试需要, 但遗憾的是没有提供多线程并发测试的注解. 而 TestNG 则弥补了 Junit 的不足, 支持的多线程并发测试的 XML 配置.

1.2 TestNG 多线程简介

TestNG 支持在多线程环境中执行的测试方法, 这种灵活的配置只需在所需测试的方法前加上一个注解, 如前面介绍的@Test(threadPoolSize = 10, invocationCount = 3, timeOut = 1000), 不仅如此, 通过 XML 的配置, TestNG 更能支持并行执行测试方法, 测试类, 测试组件. testng.xml 文件的节点属性 parallel 可以设置多线程并发运行测试的类型, 属性 thread-count 可以设置并发执行时的线程池数量.

相对于传统的单线程执行测试的方式, 这种多线程方式拥有很大的优势, 由于是并发测试, 总体测试的运行时间则可以大大减少, 并且可以单独验证某段代码在多线程环境中运行的正确性.

2 实验场景设计

2.1 Junit4 多线程实现

2.1.1 通过 main 方法实现多线程

现在我们用传统的 main 方法来执行线程, 首先设计一个类, 其中有两个待测试的方法, 方法的内容都是打印出一行信息, 打印的信息后面拼接一个线程 ID, 主要代码如表 1 所示.

表 1 Junit 测试类设计代码

```
publicclass JunitA {//测试类
@Test
    publicvoid junitAMethodOne(){
      long id = Thread.currentThread().getId();//获取一个线程ID
    System.out.println("junitAMethodOne" + id);//方法一打印带有线程ID的一条信息
    }
  @Test
    publicvoid junitAMethodTwo(){
    long id = Thread.currentThread().getId();//获取一个线程ID
    System.out.println("junitAMethodTwo" + id); //方法二打印带有线程ID的一条信息
    }
}
```

通过表 1 的代码, 我们可以知道测试类 JunitA 中有两个方法 junitAMethodOne 和 junitAMethodTwo, 这两个方法前面标注了@Test, 说明两个方法是要借助 Junit 框架进行测试的方法. 在方法的内部, 做的主要工作就是打印出一句信息, 如: System.out.println ("junitAMethodOne" + id), 打印的信息拼接了一个线程 ID, 此线程 ID 可以帮助我们知道该方法是被哪个线程所执行. 线程 ID 的线程的标识, 通过它我们就能判断两个方法是在不同的线程中执行还是在同一个线程中执行.

设计好测试类后, 通过在另一个拥有 main 方法的类中来测试上述的两个方法, 主要代码如表 2 所示. 在 mian 方法中使用循环来开启多个线程 Thread, 这里开启了 2 个线程. 线程 run 方法是执行任务的地方, 这里使用 JUnitCore.runClasses 来执行要测试的类, 这样会以线程模式来运行测试类中所有标注了@Test 的方法. JUnitCore 是执行所有测试类的核心入口类, 通过 JUnitCore 调用 runClassses()方法, 传递一个我们要进行的测试类数组, 数组有一个值, 即 JunitA.class, 运

行后, 将会有 2 个线程来并发执行测试类 JunitA 中的两个方法.

表 2 main 方法内部主要代码

```
public static void main(String[] args) {
    for(int i=0; i<2; i++) {//开启2个线程
        new Thread() {public void run() {
            JUnitCore.runClasses(new Class[] { JunitA .class });
//Junit框架运行测试类
            }}.start();//执行线程
    }
}
```

运行表 2 中的代码得到如表 3 所示的运行结果, 从表 3 的运行结果可以看出, 有两个线程 ID 分别为 8 和 9 的线程执行了测试类 JunitA 中的方法, 线程 8 执行后打印出: junitAMethodOne 8 和 junitAMethodTwo 8; 线程 9 执行后打印出: junitAMethodOne 9 和 junitAMethodTwo 9.

表 3 通过 main 方法运行结果

```
junitAMethodOne 9
junitAMethodOne 8
junitAMethodTwo 9
junitAMethodTwo 8
```

2.1.2 自定义 Runner 实现多线程

Junit4 框架中引入了扩展的机制, 它允许我们通过自定义 Runner 来个性化我们自己的测试需求. Junit 中的测试用例是交由 Runner 去执行的, 所以扩展这个 Runner, 我们也达到多程线并发测试需求. 默认情况下 Junit4 使用 BlockJUnit4ClassRunner, 在开发我们自己的 Runner 类时, 需要继承 BlockJUnit4ClassRunner, 自定义 Runner 类关键代码如表 4 所示.

表 4 自定义 Runner 类

```
public class MultiThreadedRunner extends BlockJUnit4ClassRunner { //
继承BlockJUnit4ClassRunner的自定义Runner类
 @Override
    protected void runChild(final FrameworkMethod method, final
RunNotifier notifier) {
        while (numThreads.get() <thread_count) {//开启线程的次数
      numThreads.incrementAndGet();
      new Thread (new Test(method, notifier)).start();//运行线程
        }
}
 class Test implements Runnable {//内部线程类
        private final FrameworkMethod method;
        private final RunNotifier notifier;
 @Override
        public void run ()
{      MultiThreadedRunner.super.runChild(method, notifier);//通过参
数method的传递运行测试类的方法
            numThreads.decrementAndGet();
        }
    }
 }
}
```

从表 4 代码可以看出, 自定义的 MultiThreadedRunner 内部嵌套了一个实现了 Runnable 接口的内部类, 而这个内部类是实现多程线的关键, 通过实现 Runnable 接口中的 run 方法来执行 MultiThreadedRunner 的 runChild 方法, runChild 是执行测试类的测试方法, 在该方法中, 由变量 thread_count 控件着启动线程的个数, 这里设置为 4, 然后通过 new Thread (new Test(method, notifier)).start()运行线程, 从而做到多线程并发测试的目的, 所以内部关键代码其实和通过 main 方法的实现方式是类似的, 只是自定义的 Runner 类如何运用于测试类呢?

扩展了 Runner 后, 在要进行测试的 JunitA 类前加上注解 @RunWith(MultiThreadedRunner.class), 则 JunitA 运行时会使用 MultiThreadedRunner 来跑多线程. 其运行结果如表 5 所示.

表 5 通过自定义 Runner 运行结果

```
junitAMethodOne 12
junitAMethodOne 9
junitAMethodOne 10
junitAMethodOne 11
```

从表 5 的运行结果可以看出, 此方式只针对测试类的第一个方法运行多线程并发测试, 因此对于实际业务测试过程中需要对某个特定的重要接口进行多线程并发测试可以使用此方式, 只要设置好 thread_count 变量的值, 可以任意进行测试.

2.2 TestNG 多线程实现

TestNG 的多线程一直是开发人员在执行方法并发测试时的首选, 基于 XML 文件的配置简单, 灵活, 容易上手, 满足的场景多样性, 而且 TestNG 的多线程并发测试是安全的, XML 配置文件的结构图如图 1 所示.

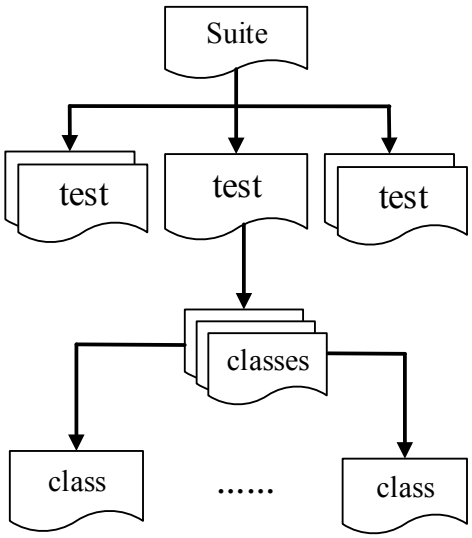

图 1　XML 多线程配置文件结构图

表 6 说明了 XML 涉及多线程配置时各节点的意义. 通过在 XML 文件中的相关节点的配置, 我们可以只对一个类中的所有方法进行并发测试; 对多个类进行并发测试; 对同一个测试套件内的多个测试组件进行并发测试.

表 6　XML 多线程配置文件属性节点说明

属性	说明
suite	一个测试套件，下面包含多个 test 组件
test	一个测试组件，下面可包含多个 class
classes	一个类集合，下面可包含多个 class
class	一个具体的要测试的类

接下来我们运用 TestNG 通过基于 XML 的配置方式来实现对测试类的多线程实现设计. 首先设计一个测试类 TestngA, 类中有两个待测试的方法, 方法的内容都是打印出一行信息, 打印的信息后面拼接一个线程 ID, 主要代码如表 7 所示.

表 7　TestNG 测试类设计代码

```
public class TestngA { //测试类
    @Test
    public void testngAMethodOne(){
            long id = Thread.currentThread().getId();//获取一个线程ID
    System.out.println("testngAMethodOne " + id); //方法一打印带有线程ID的一条信息
    }
    @Test
    public void testngAMethodTwo(){
            long id = Thread.currentThread().getId();//获取一个线程ID
    System.out.println("testngAMethodTwo " + id); //方法二打印带有线程ID的一条信息
    }
}
```

2.2.1 并发执行测试方法

现在我们通过基于 XML 文件的配置方式来测试多线程的并发测试实验. 并发执行测试类中的所有测试方法对应的 XML 配置文件如表 8 所示.

表 8　并发测试方法的 XML 文件配置

```
<suite name="test-method Suite" parallel="methods" thread-count="2" >
 <test name="test1">
  <classes>
   <class name="com.sgcc.uap.testng.TestngA"/>
    </classes>
 </test>
</suite>
```

从表 8 中我们看到 suite 这个节点有个 parallel 属性设置成了“methods”, 它代表 TestNG 将在不同的线程内运行测试方法. “thread-count”代表启用的线程个数, 此处设置成 2, 代表将有 2 个线程被分配来执行测试方法. 因此针对测试类 TestngA 中的两个方法, 将会在 2 个不同的线程内被执行.

通过右击此 XML 配置文件, 选择 Run as->TestNG Suite, 可以看到运行结果如表 9 所示, 从输出结果后面拼接的线程 ID 可以看出, 2 个方法是在不同的线程内执行的.

表 9　TestNG 并发执行测试方法的运行结果

```
TestngAMethodOne 10
TestngBMethodTwo 11
```

2.2.2 并发执行测试类

由于本节要以多线程的模式并发执行测试类, 因此需要再设计一个测试类 TestngB, 该类也有两个测试方法: testngBMethodOne 和 testngBMethodTwo, 其它和测试类 TestngA 相同, 方法内部也是打印出带有线程 ID 的信息. 现在修改上节的 XML 配置文件, 使它能够在测试类之间并发执行, 修改后的 XML 配置文件如表 10 所示.

表 10 并发测试类的 XML 文件配置

```
<suite name="test-method Suite" parallel="methods"
thread-count="2" >
 <test name="test1">
 <classes>
 <class name="com.sgcc.uap.testng.TestngA" />
<class name="com.sgcc.uap.testng.TestngB" />
 </classes>
 </test>
</suite>
```

对比表 8 的配置文件, 我们可以看出, 唯一不同的是为满足并发执行测试类, 需要在 classes 节点下再增加要并发执行的测试类的信息, 我们在本例中配置了 TestngA 和 TestngB 两个测试类, 运行后的结果如表 11 所示.

表 11 TestNG 并发执行测试类的运行结果

```
testngAMethodOne 9
testngAMethodTwo 10
testngBMethodOne 9
testngBMethodTwo 10
```

从表 11 的运行结果可以看出, TestngA 的方法 testngAMethodOne 和 TestngB 的方法 testngBMethodOne 是在线程9内执行的, 而TestngA的方法testngAMethodTwo 和 TestngB 的方法 testngBMethodTwo 是在线程 10 内执行的, 也就是说 TestngA 和 TestngB 两个测试类是并发被执行的.

2.2.3 并发执行测试组件

XML 配置文件的 test 节点是用来配置测试组件, 前面 2 节的配置文件中, 测试类都是配置在同一个名为“test1”测试组件中的, 现在我们修改 XML 配置文件, 将 TestngA 分配在名为“test1”的测试组件中, 将 TestngB 分配在名为“test2”的测试组件中, 修改后的 XML 配置文件如表 12 所示.

表 12 并发执行测试组件的 XML 文件配置

```
<suite name="test-method Suite" parallel="methods"
thread-count="2" >
 <test name="test1">
  <classes>
 <class name="com.sgcc.uap.testng.TestngA" />
 </classes>
</test>
 <test name="test2">
  <classes>
 <class name="com.sgcc.uap.testng.TestngB" />
 </classes>
 </test>
</suite>
```

运行表 12 的配置文件得到的运行结果如表 13 所示.

表 13 TestNG 并发执行测试组件的运行结果

```
testngAMethodTwo 10
testngAMethodOne 9
testngBMethodOne 11
testngBMethodTwo 12
```

从表 13 的运行结果可以看出, 虽然配置文件中 thread-count 设置成 2, 表示用 2 个线程来运行两个测试组件, 但对于 TestngA 和 TestngB 中的 2 个方法, TestNG 又分别分配了 2 个不同的进程来执行. 由此得到的结论是, 不同的组件中的测试类 TestNG 保证了它们在独立的进程中执行, 同时根据 suite 节点中的 thread-count 属性的设置分配了所设置个数的线程数量来运行各个测试组件中的测试方法.

2.3 JUnit 和 TestNG 对比分析

通过实验得出的基于多线程并发测试的 JUnit 和 TestNG 两种测试框架的技术对比结果如表 14 所示.

表 14 多线程并发的 JUnit 和 TestNG 框架对比

	并发方法	并发测试类	提供注解	支持配置文件	手动写代码
JUnit	是	是	否	否	是
TestNG	是	是	是	是	否

从表 14 的框架对比内容可以看出, 两种测试框架中都可以达到大家的多线程并发测试的需求, Junit 通过 main 方法或者自定义的 Runner 来运行测试方法, 方法内部都是执行定义的线程类达到并发测试; TestNG 则更加方便的提供了方法级的注解以及简单的 XML 配置, 就可以轻松并发测试, 同时无需再去通过编代码来达到目的.

执行并发测试时, Junit 相比 TestNG 来说, 更多的是我们在控制着执行的代码, TestNG 则相对偏于配置的组合, 一个简单的 XML 配置文件便能撑起整个并发测试的需求. 因此结合这几个点来说, TestNG 在多线程并发测试方面的优势则比 Junit 更为突出.

3 结语

在 java 单元测试方面, JUnit 和 TestNG 一直是开发人员最常用的开源自动化测试框架, 两者各有不同的特性, 而 TestNG 则优于 JUnit, TestNG 创建的灵感来自于 JUnit 和 NUnit, 但它却不是一个 JUnit 的扩展. 从支持多程线方面, 可以看出, JUnit 不管是否实现自定义的 Runner, 还是从传统的定义线程类开始, 都需要用户手动编写代码来达到测试的目的, 同时对特殊场景, 还得考虑它的线程安全问题; TestNG 基于 XML 配置的方式来运行多线程或者通过方法级上的注解便可支持多种并发测试的需求, 而且配置文件简单、灵活、方便, 无需用户多花时间研究它的代码即可进行测试. 所以在进行多线程并发测试方面, 选择 TestNG 单元测试框架将是一个正确的决定.

在其他测试需求方面, TestNG 还有 JUnit 框架无法比拟的优点, 但两者都有自己适用的场景, 开发人员最需要的是选择一个适用自己场景的单元测试框架, 选择最合适的会比选择功能最多的给用户带来更意想不到的结果, 灵活的运用测试框架才会完成高效的测试工作.

参考文献

1 周瑞阳,王猛.基于三层体系结构的单元测试框架研究与实现.计算机应用,2010,30(8):29–37.

2 白凯,崔冬华.基于 JUnit 自动化单元测试的研究.计算机与数字工程,2010,38(2):52–54.

3 孔亮亮,殷兆麟.Java 类测试工具 JUnit 的分析与扩展.计算机工程与设计,2005,26(12):3413–3416.

4 董晓霞.相邻因素组合测试用例集的最优生成方法闭.计算机学报,2007,30(2):200–210.

5 Denis A, Perez C, Priol T, et al. Padico: A component-based software infrastructure for Grid computing. Proc. of lnternational Parallel and Distributed Processing Symposium. ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-4974.pdf. [2010-03-01].

6 Bradley S. Green. Software test automation. ACM, 2000, 17 (9): 9–45.

7 Zhu H, Wong WE, Belli F. Advancing test automation technologyto meet the challenges of model-driven software development. ACM, 2008, 19(15): 5–10.

8 Beust C, Suleiman H. Next generation Java testing: TestNG and advanced concepts. Pearson Education, 2007.

9 Wang XC, Xu PJ. Build an Auto Testing Framework Based on Seleniumand FitNesse. 2009 International Conference on Information Technology and Computer Science. 2009, 2. 436–439.

10 余波,王树林,张大方.基于 JUnit 自动生成类测试案例框架的实现.计算机工程与应用,2006,42(10):60–98.

11 白凯,崔冬华.基于 JUnit 自动化单元测试的研究.计算机与数字工程,2010,38(2):32–35.

12 王晶,樊晓娅,张盛兵,等.同时多线程结构的 2 级调度策略.西北工业大学学报,2007,(3):433–437.

13 肖明清,朱小平,夏锐.并行测试技术综述.空军工程大学学报(自然科学版),2005,(3):22–25.

14 夏锐,肖明清,朱小平,等.并行测试技术在自动系统中的应用.计算机测量与控制,2005,13(1):7–10.