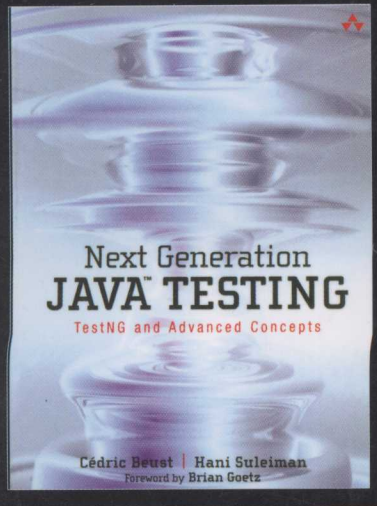
HZ BOOKS
PEARSON Addison Wesley
华章程序员书库

Java测试新技术

TestNG和高级概念

Next Generation Java Testing

TestNG and Advanced Concepts



Cédric Beust
Hani Suleiman 著
王海鹏 译

- TestNG创始人最新力作。
- 介绍了大量新的测试模式和一些新工具。
- 以实例展示测试模式。

机械工业出版社
China Machine Press

华章程序员书库

Java测试新技术

TestNG和高级概念

Next Generation Java Testing

TestNG and Advanced Concepts

Cédric Beust
Hani Suleiman 著

王海鹏 译

本书介绍Java测试的新技术，主要内容包括：基本概念、测试设计模式、企业级测试、Java EE测试、集成和扩展TestNG等。本书通过针对有效测试Java应用程序以及围绕可测试性来设计应用程序和组件，以实用的方式展示了这些有效的测试技术，并给出了每种测试方法的优点和不足，展示了解决常见问题的不同选择。

本书注重实际应用，适合对测试感兴趣的Java开发者参考阅读。

本书版权登记号：图字：01-2008-1791

图书在版编目（CIP）数据

Java测试新技术：TestNG和高级概念/伯斯特（Beust，C.），苏雷曼（Suleiman，H.）著；王海鹏译. —北京：机械工业出版社，2008.8

（华章程序员书库）

书名原文：Next Generation Java Testing：TestNG and Advanced Concepts

ISBN 978-7-111-24550-6

Ⅰ.J… Ⅱ.①伯… ②苏… ③王… Ⅲ.JAVA语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2008）第098934号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：王春华

北京京北印刷有限公司印刷 · 新华书店北京发行所发行

2009年1月第1版第1次印刷

186mm×240mm · 21.25印张

标准书号：ISBN 978-7-111-24550-6

定价：49.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

软件开发是一项风险事业。测试则是缓解项目风险最重要的手段之一。一般来说，我们应该让需求可测试，让测试自动化，让自动化测试变得容易。

本书作者采用的是实用主义的方式，这一点对于真实项目的开发者帮助特别大。没有教条式的金科玉律，有的是更多实际可行的平衡和折衷。作者在我们面前展现了多姿多彩的Java企业级应用开发的实景，介绍了他们以及TestNG用户社区的实际开发经验和测试经验。在开发中，我们也许要和300万行遗留代码打交道，要和启动缓慢的应用服务器、数据库服务器打交道，要和各式各样、不断涌现的复用组件和库打交道，我们的生活充满了挑战。在本书中，您会看到世界一流的开发者是如何应对这些挑战的。

Java的开发社区充满了创新。这些创新者都有一个良好的愿望，让好的思想和工具为尽可能多的人提供帮助。TestNG的作者也是一样，所以本书既包含了理性的思考，也包含了善良的祝福：您可以更高效地完成项目，然后有更多的时间来锻炼身体或陪伴家人（或玩魔兽世界）。

理念一定要先进，工具一定要先进。将这些先进的理念和工具应用于项目中，超过社会平均的生产效率，这就是创新的意义所在。

JUnit让开发者编写测试的概念深入人心，TestNG则将我们的视野扩展到所有的测试，不仅仅是单元测试，还有集成测试、系统测试、功能测试、验收测试、压力测试……我相信，这本书将会给Java开发者带来诸多帮助。

本书由王海鹏负责翻译，参加本书翻译工作的人员还有：王海燕、李国安、周建鸣、范俊、张海洲、谢伟奇、林冀、钱立强、甘莉萍。在本书的翻译过程中，我学到了很多，因此郑重地向大家推荐它。如果这本书对于您改进软件开发实践有所帮助，我将十分高兴。

王海鹏

戊子年初夏于上海

序

做正确的事情从来就不容易。我们大多数人可能应该更合理地饮食，进行更多的锻炼，花更多的时间和家人在一起。但是每天，当面对做容易的事还是做正确的事的选择时，我们常常选择容易的事，因为这些事更容易一些。

对于软件测试也是一样。虽然我们都知道更多的测试会让我们的代码更可靠、更可维护，我们应该在测试上花费更多的精力，而且如果这么做，用户也会感谢我们，从而帮助我们更好地理解自己的程序，但是每天当我们坐在计算机前，面对编写更多测试还是更多应用程序代码的选择时，还是很想选择更容易的事情做。

今天，大多数人都已承认单元测试是开发者的职责，而不是 QA 的职责。对此，我们要感谢 JUnit 测试框架。JUnit 之所以产生这样大的冲击，是因为它实在没有太多的东西——它是一个简单的框架，没有多少代码。JUnit 改变了开发者的行为，而成年累月的说教和过错都没有做到这一点，这其中最重要的原因就是，编写单元测试的痛苦被减少到了可以忍受的程度，让我们实际上有可能将单元测试包含在日常编码工作中。JUnit 没有激发编写测试的渴望，这是不太容易让人接受的，它只是让我们能够在做正确的事情时更容易。

随着新形成的这些正确观点，许多开发者宣称他们对测试充满热情，自豪地将自己称为“受测试感染的人”。这都很好，很少有人会说软件开发者做了太多的测试，所以更多的测试可能是一种进步。但是，这只是第一步。除了单元测试之外，还有更多的测试，如果希望开发者再深入一步，我们必须提供一些测试工具来减少创建这些测试的痛苦，并且将可测试性作为一项基本的设计要求。如果软件工程打算成为真正的工程实践，测试就将成为支撑它的一根关键支柱。（也许有一天，编写没有测试的代码会被认为是没有职业负责精神的做法，就像造桥时不进行结构分析一样。）

这本书基于的观点是，我们刚刚才开始进行负责任的测试。TestNG（NG 的意思是“下一代”）项目的目标是帮助开发者再深入一步，让我们能进行范围更广、更深入的测试，不仅包括单元测试，还包括验收测试、功能测试和集成测试。它提供了许多有用的特征，其中包括指定测试套件和向测试套件传递参数的丰富机制，支持并发测试，以及解除测试代码及其数据源的耦合的机制。它的一些功能被最近的 JUnit 版本吸收了，这也证明了 TestNG 的成功。

不论提供什么工具，进行更有效的开发者测试存在一项挑战，即对于编写有效的测试与编写有效的产品代码来说，对技能的要求是不一样的。但是和大多数技能一样，测试技能是可以习得的，最好的学习方式就是看看更有经验的人是如何做的。在这本书中，Hani 和 Cédric 针对有效测试 Java 应用程序以及围绕可测试性来设计应用程序和组件，分享了他们喜爱的技术。最后一项技术（围绕可测试性来设计）可能是这本书中最有价值的经验之一。围

绕可测试性来设计代码迫使您更早地思考组件间的交互和依赖关系，从而鼓励您创建更干净、更松耦合的代码。当然，书中利用了 TestNG 来展示这些技术，但即使您不是 TestNG 的用户（也不想成为一名 TestNG 的用户），这里展现的实用技术也将帮助您更好地测试，从而更好地进行软件工程。

Brian Goetz
Senior Staff Engineer, Sun Microsystems

前　言

我们都在进行软件开发：编写代码，然后部署。当部署了代码之后，客户会表现高兴或不高兴。令人沮丧的是，他们经常表现得不高兴。

在过去的几十年里，人们为了减少这种不高兴编写了许多东西。我们有无数种语言、方法学、工具、管理技术和老式秘方，来帮助处理这个问题。

其中有些方法很有效。最近，人们将关注的重点重新放到了测试上，据说测试将给开发者和用户带来欢乐。

人们写了很多东西来赞美测试的优点。它可以让您的代码更好、更快、更轻。它可以为编码这样的乏味工作带来某种极为需要的调味剂。它既令人激动又很新（因此值得尝试），更别提它带来的那种责任感和可靠感。添加一项功能，然后有一个测试证明您做对了，这带来了某种神奇的成就感。

遗憾的是，宗教也渗入到了测试科学之中。您不必费力就能发现一些神圣的戒律或权威人士发出的指令，赞赏或斥责某些测试行为。

这本书试图提炼近几年来在 Java 测试领域中出现的一些精华。我们不曾领取薪水来推销测试，也没人强迫我们测试。在有的地方，一种方法学被宣布为获胜者，必须虔诚地奉行，我们都没在这样的地方工作过。

相反，我们是实用主义的测试者。测试对我们来说只是有价值的工具，帮助我们完成软件开发生命周期中的一部分工作。我们不是“受测试感染的人”，这个由 JUnit 在早期声明的术语已经广为流传。当我们觉得有意义的时候，就编写测试。对我们来说，测试是一种选择，不是一种传染病。

这种方式的结果是，我们注意到测试工具库中有一块很大的空白：很少有工具是从实际出发，能够方便地编写我们想写的那些测试。Java 测试中的主要工具是 JUnit，在许多情况下，它让我们很容易、很直接地考虑我们想执行的测试。但是，一个主要的障碍使它不能成为我们的工具，它不能反映我们想测试的代码中的一些更为深入的概念，例如，封装、状态共享、范围和顺序等。

尽管有许多缺点，但 JUnit 确实将测试的概念摆在了我们面前。它不再是一种随意而为的方法。相反，测试有了一个框架，并且有适用的测量标准。利用各种可视化工具，基于 JUnit 的测试可以很容易自动化，在多种环境下重复执行。这种易用性使得人们大量采用它，从总体上增强了 Java 测试的意识。

它的成功也扩展到了其他一些语言，它被移植到另一些语言上，底层的概念是相同的。

但是与任何其他成功的工具一样，成功是有代价的。微妙的焦点转移发生了，人们不再关注 JUnit 作为一项工具所支持的测试，而开始关注 JUnit，如果有测试不能够符合它的狭

小范围，人们会怀疑测试，而不是怀疑工具。

很多人会宣称，不能很容易地表达为一个简单“单元”的测试是有问题的测试。由于它的要求超出了JUnit所提供的简单功能，所以它不是一个单元测试。它是一项功能测试，应该在我们完成了单元构建块之后才发生。我们觉得这种论调让人迷惑不解。说到底，没有哪一种方法是进行测试的最佳方法。有人宣称开发必须从实现较小的完整单元开始，然后再思考更高层面的概念，这也是同样可笑。在有些情况下，这样做确实很有意义，但在另外一些情况下，这样做是没有意义的。测试是实现目标的手段，而目标是更好的软件。时刻记住这一点是很关键的。

为什么再写一本关于测试的书

这本书是关于Java测试的。您接下来读到的每一章每一节都会讨论这样或那样的测试。不论您使用哪种测试框架，或者使用了我们没有提到的工具，我们的目标都是向您展示我们试过有效的一些实践。我们也尝试根据自己的经验总结一些一般结论，并利用这些一般结论来推测将来可能出现的一些情况。

虽然我们在本书中利用TestNG来表达我们的思想，但是我们坚信，不论您是否使用JUnit，都会发现一些有用的内容，即使您不在Java平台上编程。有许多针对其他语言（如C#和C++）的类似TestNG和JUnit的框架，测试框架中包含的思想通常是普遍适用的，可以超越您所遇到的实现细节。

这本书是关于实用主义测试的。在本书中，您不会看到绝对的说法，没有根据的、宗教式的宣告，以及确保健壮代码的黄金法则。相反，我们总是试图展现每种情况的优缺点，因为归根到底，作为开发者的您才是对您面对的系统具有知识和经验的人。我们不能够在具体的事情上帮助您，但我们肯定可以向您展示解决常见问题的不同选择，让您来决定最合适的方法。

了解了这一点，让我们回到上面的问题：为什么再写一本关于测试的书？

关于Java测试有很多书，但仔细看时，我们认为几乎没有一本书包含了我们在日常工作中发现的、非常重要的一个宏大主题：现代Java测试。

是的，在一本书中使用“现代”这个形容词是很危险的，因为从本质上说，书籍不能够长久地保持“现代”。我们不认为这本书能逃过这一法则，但我们很清楚，已有的那些关于Java测试的书籍没有很好地处理Java开发者今天所面临的挑战。您可以在本书的目录中看到，我们介绍了大量的框架，其中许多是在最近三年内才出现的。

在我们对已有书籍的调查中，我们也意识到大多数Java测试的书籍使用了JUnit，它虽然是一个品质不错的测试框架，但是自从2001年以来，它就几乎没有做过任何改进[⊖]。我们不仅发现JUnit的年龄带来了一些局限性，而且发现它的设计目标也有局限性：JUnit是一

⊖ JUnit 4在2006年发布，是5年以来JUnit的首次更新，但在本书编写时，它的采用仍然相当有限，因为大多数Java项目仍在使用JUnit 3。

个单元测试框架。如果试图用 JUnit 来做超出单元测试的工作（如测试在应用服务器中的一个真实 servlet 的部署），您可能就用错了工具。

最后，我们还介绍了一些刚刚开始被采用的新工具（如 Guice）。我们相信这些工具与 TestNG 这样的现代测试框架一起使用时，具有很大的潜力，为我们打开了许多扇门，所以我们无法拒绝在书中介绍它们。希望我们对这些崭新框架的介绍能够为您的尝试带来一些方便。

在整本书中，我们都试图展示一种实用主义的应用程序测试方法。这本书介绍了许多模式。它们不是以清单的形式列出的，我们不希望读者将它们背下来，相反，它们仅仅是一组例子，确保您在面对测试代码时，找到正确的思考方法。

我们是通过两种独立的途径来做到这一点的。第一种途径是 TestNG 的用法说明。我们介绍它的大多数功能，解释这些功能是如何出现的、为何会出现，以及应用这些功能的实际例子。通过这种讨论，我们可以看到测试模式是如何通过测试框架来反映的，什么是健壮的、可维护的测试套件。

第二种途径是展示 TestNG 如何与现有的代码以及大量的 Java 框架和库集成。很少有人会有幸做一个完全从头开始的项目。总有一些组件需要复用或集成，有一些遗留的子系统需要调用，或者需要考虑向下兼容的问题。为了支持测试而要求重新设计或重写是很愚蠢的。相反，我们试图展示如何能够与已有的代码集成，小的增量式的改动如何能够让代码变得更加可测试，随时间推移而变得更健壮。同样，通过这种方式，一些模式出现了，其中也包含了关于如何编写测试更多实践和进行测试的一般方法。

我们希望您在阅读本书时得到享受，就像我们在编写本书时得到享受一样。我们对测试有强烈的感情，但同样强烈地感受到，在各种钉子的世界里不存在一把金锤子。虽然有些人宁愿相信这一点，但是实际上没有哪种方法或解决方案能够让您不必思考，不必理解您的目标，就确保您的测试过程是理性的、考虑周全的。

本书读者

那么，这本书是写给谁的呢？简而言之，它是写给对测试感兴趣的 Java 开发者的。

我们的目标也包括这样一些开发者，他们已经对代码进行了一段时间的测试（使用 JUnit 或其他框架），但仍然发现自己有时候被代码明显的复杂性、范围或测试的工作量吓住了。在 TestNG 社区的帮助下，这些年来，我们对测试各种 Java 代码时形成的一些最佳实践的理解有了很大的提高。我们希望这本书充分地讨论这些最佳实践，这样大家在遇到测试问题时不会不知所措。

这本书使用了 TestNG 来编写示例代码，如果您还不是 TestNG 的用户，不要被吓住了：许多的原则很容易在最新版本的 JUnit 中采用（或移植）。

不论您是否使用 TestNG，我们都希望您读完这本书时，会学到一些测试 Java 代码的新技术，将来能够正确地应用。

致　谢

在编写这本书的过程中，许多人为我们提供了帮助，在此对他们表示感谢。

首先，我们要感谢 Alexandru Popescu，感谢他从很早就开始在 TestNG 上不知疲倦地工作，感谢他对用户的绝对奉献精神。没有他，TestNG 不会像今天这样。

有些人帮助我们检查了本书的草稿，他们发现了许多错误和不准确的地方，如果不是他们，我们就会漏过这些问题。按照字母顺序，我们要向以下人员表达特殊的“QA 感谢”：Tom Adams、Mike Aizatsky、Kevin Bourrillion、Brian Goetz、Erik Koerber、Tim McNerney、Bill Michell、 Brian Slesinsky、James Strachan 和 Joe Walnes。

我们也要特别感谢 Bob Lee 和 Patrick Linskey，在过去的几年里，他们经常与我们讨论许多问题，这些讨论最终以某种方式影响了 TestNG（和这本书）。

如果没有 TestNG 邮件列表中的这些人，这一切都不可能发生了。这些年来，他们提供了鼓励、洞见、补丁和用例，以及足够多的建设性批评意见，帮助我们优化在测试领域的想法和思考，这一切对我们的帮助无法估量。

Hani 要感谢他的父母、兄弟姐妹（Sara、Ghalib、Khalid 和 May）、妻子（Terry）和那帮 Lost 剧迷们——这些人忍住嘲笑指出写完这本书是多么不可能的一件事情。

Cédric 要感谢他的妻子 Anne Marie，在编写这本书的过程中，她付出了耐心和支持。

最后，特别提一下，“不要感谢”暴雪公司，魔兽世界的发布者，没有它这本书要容易写得多。

Cédric 和 Hani

目　录

第1章 起 步

这里先介绍本书作者的一段轶事，我们将在本章余下的部分利用这段轶事来引入本书中讨论的一些概念。

几年前，我曾遇到一个似乎很简单的问题：在我开发的产品中，一个测试在那一天早些时候失败了，我花了几个小时试图找出问题所在。我几乎查遍了这个测试执行到的所有代码。我在代码中的不同位置加了许多断点，但变量总是包含正确的值，代码执行的路径也和设想的一样。

我无计可施了，我逐渐认为，如果问题不在我的代码中，可能是在工具中。

在面对一个软件问题时，我已经学会尽量控制自己不要去责备工具或其他层次的、不属于我的代码，特别是它们已经工作了很长时间，又从来没出过问题。但我想不到任何其他方法能够解释我所看到的失败。

因此，我把我的调试器指向了 JUnit，开始了又一次的断点跟踪过程。

然而我有了惊人的发现。

我不记得当时导致这种现象的确切代码了，但我知道它基本上可以简化为程序清单 1-1 中的代码。

程序清单 1-1 基本 JUnit 测试

```
public class MyTest extends TestCase {
  private int count = 0;

  public void test1() {
    count++;
    assertEquals(1, count);
  }
  public void test2() {
    count++;
    assertEquals(1, count);
  }
}
```

快速看一下这个测试用例，然后想象一下用 JUnit 执行它。您认为会发生什么情况？通过还是失败？

答案是：它通过了。

我记得当我看到这个结果时，难以置信地摇着头。我真的不敢相信自己的眼睛，我断定自己是画蛇添足了，以至于脑子转不过弯了。但对 JUnit 代码的调试证实了这一点：JUnit 在执行每个测试方法之前，都会重新实例化测试类，这就解释了为什么 count 字段每次都被

重置为0。

当然，我的下一个问题就是这到底是JUnit的一个缺陷，还是设计的行为。从内心深处，我觉得这不可能是一个缺陷，因为JUnit是一个无数Java项目都使用的框架，如果这样的行为不是有意设计的，肯定已经有人报告过了。

所以我进行了一些研究，并与Kent Beck和Eric Gamma交换了几封电子邮件。他们确认这是有意设计的行为，并说这种想法背后的原因是确保每个测试开始时状态都被重置，从而确保测试彼此之间不会产生依赖关系，或者一个测试对另一个测试产生副作用。

一方面我确实看到了这种想法的价值。毕竟，测试之间越独立，分别执行它们就越容易。但是另一方面我也被这个行为弄得大吃一惊，如果有人说JUnit的实现是有道理的，那么他也很难否认这是非常违反直觉的。如果Java或C++采用了同样的语义，我肯定会觉得不舒服。

工作中这个无害的小插曲只是一个开始，它使得我花了很长时间回顾我的测试习惯，全面回顾了多年来我一直不假思索地使用的测试实践。JUnit无疑是Java世界中事实上的标准测试工具，但我逐渐开始意识到这种广泛的应用也有一些代价，我们的测试习惯有一些过于自动化了，没有经过仔细的反思。

在这短短的一章中，我们将向您介绍本书的主要内容，以及本书目标之后的动机。

1.1 超越JUnit 3

像大多数Java开发者一样，我们使用JUnit[⊖]的历史已经很长了，我们当然相信它使我们的测试更可靠健壮。但是这些年来，我们也遇到了这个框架中的一些不足，至少我们认为是这样的。我们报怨的许多行为实际上是JUnit的作者有意设计并实现的。

我们将从一些例子开始，然后通过说明我们建议如何绕过这些不足来总结这部分内容，帮助读者投入到下一代的测试中去。

1.1.1 有状态的类

让我们回到本章最初讨论的问题。之所以发生这个问题，是因为我们试图在两个测试方法之间维持状态。我们在一个测试方法中将一个字段设置为特定的值，然后我们期望这个值出现在另一个测试方法中。

JUnit似乎是在说，我们不应该做这样的事情，而且为了强制执行这样的实践，它在每个方法调用之间将测试类重新实例化。

我们会感觉试图在测试类中维持状态是件坏事吗？

如果您曾在书中或文章中读过有关测试实践的内容，会知道这样的实践确实是让人皱眉头的事情，尽管如此，我们仍然发现自己在测试活动中经常有这样的需要。那么为什么

⊖ 除非特别指明，本书中的JUnit都是指JUnit 3。

JUnit 要阻止我们这么做?

在 JUnit 中也有一个绕过去的办法，它作为一种设计模式而推荐：使用静态变量来保存值，这样它就不会每次都重新实例化。

不错。但静态变量有一些代价。

- 它们不太适合在同一个 JVM 上执行。在同一个 JVM 上执行几次测试（如使用 <java>ant 任务而不指定 fork= "yes"），静态变量可能从一个执行到另一个执行。
- 静态变量也引入了一些线程安全缺陷，在可能的情况下最好避免。

总之，我们觉得 JUnit 有点僵化，它迫使我们修改原来编写的代码，只为绕过框架本身的不足。

我们将在第 2 章和第 7 章中更详细地讨论测试中的状态。

1.1.2 参数

JUnit 中的测试方法有以下约束：

- 它们的名称必须以 test 开始。
- 它们不返回任何值。
- 它们不能带任何参数。

最后一条限制经常使我们觉得难受。为什么我们不能够向测试方法传递参数？毕竟，向方法和函数传递参数是在编程语言中出现了数十年的思想，我们想不出不熟悉这一思想的程序员。

向测试方法传递参数似乎是非常自然的，有时候还特别实用。同样，我们发现您无法利用 JUnit 做到这一点，模拟这种结果的唯一方法是利用一种绕过去的设计模式，我们将在 2.3 节详细讨论。

1.1.3 基类

要作为 JUnit 的测试类，Java 类需要从 TestCase 类继承。

这个类实现了几个目标：它提供了一组断言方法，同时它提供了 setUp/tearDown 等生命周期方法，这对于测试的正确执行是至关重要的。

同样，强制从一个基类扩展也让人觉得受到了强迫，特别是在 JDK1.4（断言）和 JDK1.5（静态 import）中出现了一些新功能，使得这些约束变得没有必要了。

1.1.4 异常并非偶然

所有严肃对待测试的开发者都会花费大量的时间来确保他们的代码能够很好地处理异常。确保您的程序提供了预期的功能，这一点很重要，但是验证预期的错误情况要么得到处理，要么以友好的方式告诉用户也同样重要。

JUnit 没有我们所谓的异常测试支持，当您面对这种情况时，会发现自己要编写绕弯的