HZ BOOKS
华章科技

MANNING

资深敏捷技术实践专家撰写，系统且深入地阐释单元测试用于软件设计的工具、方法、原则和最佳实践

深入剖析各种测试常见问题，包含大量实践案例，可操作性强，能为用户高效编写优秀测试提供系统实践指南

华章程序员书库

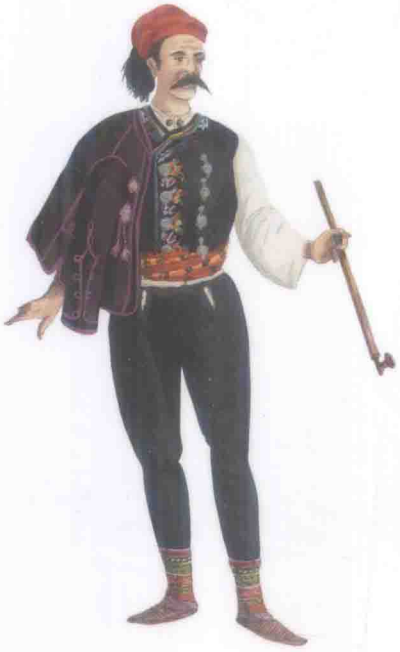
Effective Unit Testing

A guide for Java Developers

有效的单元测试

（芬） Lasse Koskela 著

申健 译



机械工业出版社
China Machine Press

华章程序员书库

Effective Unit Testing

A guide for Java Developers

有效的单元测试

（芬） Lasse Koskela 著

申健 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

有效的单元测试 /（芬）科斯凯拉（Koskela，L.）著；申健译．—北京：机械工业出版社，2014.11
（华章程序员书库）
书名原文：Effective Unit Testing: A guide for Java Developers

ISBN 978-7-111-48343-4

I. 有… II. ①科… ②申… III. JAVA 语言－程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2014）第 246045 号

本书版权登记号：图字：01-2014-0807

有效的单元测试

出版发行：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：秦　健　　责任校对：董纪丽
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2014 年 11 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：13.5
书　　号：ISBN 978-7-111-48343-4　　定　　价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

The Translator's Words 译 者 序

单元测试最初兴起于敏捷社区。1997 年，设计模式四巨头之一 Erich Gamma 和极限编程发明人 Kent Beck 共同开发了 JUnit，而 JUnit 框架在此之后又引领了 xUnit 家族的发展，深刻地影响了单元测试在各种编程语言中的普及。

随着敏捷开发大潮的流行，单元测试也成了现代软件开发中必不可少的工具之一。古人云，流水不腐，户枢不蠹。越来越多的程序员推崇自动化测试的理念，作为经济合理的回归测试手段，以适应迭代开发的需要。然而有些时候，这些测试对生产力并无明显改善，人们盲目地追求测试覆盖率，却忽视了测试代码的质量，各种无效单元测试反而带来了沉重的维护负担。

2013 年年初，一位远在新加坡的印度朋友 Ram 就推荐我看过本书的英文版，当时完整读了一遍，觉得该书深入浅出，覆盖了很多关于编写优秀单元测试的内容，而且总结得很有条理。没想到的是，2013 年下半年，出版社的编辑找到我说有一本书愿不愿翻译，没想到正是本书！既然有缘，当时就答应了下来。

本以为半年可以完成翻译，却没想到过去的 12 个月里生活和工作出现各种变化与惊喜，于是翻译工作一拖再拖，进展不快。

2014 年 6 月 Scrum Gathering 大会在上海召开，作为话题演讲的总制作人和大会讲师，原想书能尽快出版，就可以带去一些回馈社区，只可惜没能实现，又让朋友们多等了几个月。

目前，我从事敏捷教练和培训的工作，同时通过动手实践仍在不断提高自己的编程水平。希望借此机会，将我过去几年的敏捷实践经验分享给更多人。

个人喜欢和敏捷社区的软件匠友动手切磋，一起编写高质量代码，另外在讲授 CSD（认证 Scrum 开发者或称敏捷技术实践）课程时，也经常会接触来自不同行业的软件开发者。在这个过程中，我们发现，审美之前有必要先学审丑。好的编码模式各有千秋，能抓到老鼠的

就是好猫。然而，坏的模式却是有限的。

本书作者 Lasse 在敏捷技术实践领域一直走在前沿，在 TDD 和单元测试领域颇有研究。他将本书分为三部分，首先分析了编写测试的目的，以及优秀测试的特征，然后是本书的核心部分，从三个角度一一阐述了测试的坏味道。此外，本书还介绍了测试替身的概念与用法，如何用另类 JVM 语言为 Java 代码编写测试，如何提高代码的可测性，以及如何加速测试和构建的速度，从而加快反馈的速度。

翻译是个耗时的差事，时间投入与经济回报不成比例。于是，有人不禁问你为什么还要坚持翻译。在我看来，既然自己从社区中获得很多养分，也就有义务将更多丰厚的知识和实践经验分享给大家。同样的原因，几年来我也一直坚持参与组织各种社区活动，回馈社区。

作为译者，因为能力水平有限，难免会有疏漏，在此恳请读者见谅，并给予批评指正。

在本书的翻译过程中得到了多位软件匠友大力协助，非常感谢：尹哲、伍斌、王洪亮（排名不分先后）。

感谢所有关心和鼓励我的人。特别是我的家人，在那些挑灯奋战的夜晚里默默地给予着支持。

今宵梦霓虹　明空浩荡

残思追穹方　月已西徜

Preface 序　言

2009 年 6 月 10 日夜里，我收到来自 Manning 出版社 Christina Rudloff 的电子邮件，问我是否认识合适的人选，希望可以按照 Roy Osherove 的《 .NET 单元测试艺术》来写一本 Java 版的书。我跟她说，我来。

那是很久以前的事情了，而你现在看到的书与 Roy 的书已经大不相同了。我来解释一下。

项目一开始只是直接地从 .NET 翻译成 Java，仅仅会为了符合技术平台、工具和读者的变化而重写一些必要之处。我完成了第 1 ~ 3 章，然后突然发现自己不只重写了一些段落，而是整个章节。那书不是我的调调；有时我不同意 Roy 的看法或者意见有偏差，有时我希望表达自己的想法，直截了当，大步前进。

最终，我决定从头开始。

很显然这不是一个翻译项目。这是全新的题目——一本帮助 Java 开发者改善测试的书，让人深入了解何为优秀的测试，以及避免落入各种陷阱。你仍然会在本书中以各种方式见到 Roy 的思想。例如，第二部分的章节标题就明显地从 Roy 那里借鉴而来，而第 7 章在很大程度上要感谢 Roy 的《 .NET 单元测试艺术》书中的相应部分。

本书是针对 Java 程序员的。但我并不想将想法束缚在本书中，因此即便模式目录中的所有代码例子都是用 Java 写的，我也尽量避免特定于语言的内容。书写优秀测试是个与语言无关的问题，即使你的大部分工作时间都花在另一种编程语言上，但是我也真心地推荐你带着思考来阅读本书。

与此同时，我不希望写成我喜欢的 mock 对象库或 JUnit 的一个教程。除了因为这些技术更新得太快以至于出版几个月就会过时，我也是希望写一本自己乐意去阅读的书。我喜欢专注的书籍，它不会强加我早就了解的一个测试框架或者用不到的 mock 对象库，而造成额

外负担。基于这些原因，我尽量减少特定于技术的建议。虽然还是会有一些，但是我希望你知道我已经尽最大努力来减少它们了——只是用来有意义地谈论底层概念，那些概念我认为是书写、运行、维护和改善测试的基础。

我试着写了自己愿意读的一本书。我希望你也喜欢，而且最重要的是能将其中一些想法融入到你的实践中。

Introduction 前　言

过去 10 余年间，Java 开发者显著地青睐开发者测试。如今，计算机科学专业的毕业生无人不知自动化单元测试及其在软件开发中的重要性。这个想法很简单——确保我们的代码能工作并且一直能工作——但是该技能需要花很大力气去学习。

编写测试、学习 JUnit 的测试框架，这些都不难。要真正地掌握编写自动化单元测试实践，需要花大量时间在阅读并改善测试代码上。这种持续的测试重构能够尝试用不同方式来表达意图、组织测试的不同行为、用测试构建各种用到的对象——这才是一种务实的方式，用来自我学习和培养对单元测试的感觉。

这种感觉是关于哪些是优秀的单元测试，而哪些不那么优秀。有些是绝对的真理（比如完全在重复代码内容的注释就是冗余的，应该被删除），但大多数关于单元测试的知识都取决于上下文。通常意义上的优秀在特定条件下可能却很糟糕。同样，一般认为糟糕和应当避免的想法有时却是正确的做法。

原来，找到优秀方案的最好方式就是尝试一个看似可行的方法，识别该方法的问题，然后改变该方法从而消除讨厌的部分。通过重复这个过程，不断地评估和进化，最终你会找到一个可行的方案，它闻起来没那么臭。你甚至会说那是相当优秀的方式！

考虑到这种复杂性，本书采用了一种风格和结构，那就是我们不会告诉你怎么做，也不会告诉你怎么编写单元测试。相反，我打算给你一个坚实的基础，让你知道哪些是我们希望测试表现出的属性，然后给你尽可能多的例子来培养你对测试坏味道的感觉——帮你注意到测试中的不合时宜之处。

受众

本书针对想要提高单元测试编写质量的各个层次的 Java 程序员。虽然我们提供了附录来教给你测试框架（JUnit），但我们的主要目标是帮助已经了解单元测试的 Java 程序员，用其

喜欢的测试框架来编写更好的单元测试。不管你已经写了多少单元测试，我们肯定你仍然可以做得更好，阅读本书或许能带你揭示一些难以言喻的想法。

路线图

本书提到了多个方面的挑战，因此需要良好的结构来支撑这些方面。深思熟虑后（源自于多次迭代中的失败尝试），我们决定将本书分为三个部分。

关于更好的测试，第一部分先介绍了我们要达到的目标，以及它们为什么是可取的。其中的三章展示了编写优秀测试的基本工具和简单指南。

第 1 章从自动化单元测试的价值主张开始。我们通过考虑程序员生产力的多种因素来建立价值，以及编写优秀自动化单元测试如何有助于生产力，或避免耽误我们。

第 2 章提高了标准，尝试定义什么是优秀的测试。该章中的属性和注意事项是第二部分的核心基础，涉及测试的可读性、可维护性和可靠性。

第 3 章进一步介绍测试替身，作为编写优秀测试的重要工具。我们并不是为了使用测试替身而使用，而是深思熟虑地用好它们。（它们不是你想要的银弹。）

第二部分与第一部分形成鲜明对比，把一个测试坏味道的目录摆在你面前。除了描述测试代码中的可疑模式外，我们也给出了面对这些坏味道时的尝试方案。这些章节分为三个主题：令可读性退化的坏味道、暗示着潜在维护性梦魇的坏味道，以及带来信任危机的坏味道。第二部分的许多坏味道可以归为三大章节中的任何一个，但我们尝试根据其主要影响来归类。

第 4 章中的坏味道，主要侧重于过分不透明的测试意图或实现。我们涉及诸如模糊的断言、不恰当的抽象层次和测试代码的信息分散。

第 5 章中的坏味道会导致在办公室加班，因为要针对一个小的变更而没完没了地修改某个单元测试，或者为了一个小的变更要修改数百个测试。我们谈及了代码重复、测试代码中的逻辑和阐述触及文件系统的恐怖。我们也不会让缓慢的测试从眼皮底下溜过去，因为时间就是金钱。

第 6 章是测试坏味道目录的最后一部分，谈及一系列关于假设的陷阱。有些假设是由于测试代码中麻烦的注释，有些则是由于未能清楚地表达自己而带来的失败和不幸。

第三部分可以称为“高级话题”。然而并非如此，因为这里的话题与第一部分和第二部分并无关联。相反，这些是 Java 程序员在编写测试时随时可能碰到的话题。毕竟，无论是关于单元测试类的继承关系，还是用来编写测试的编程语言，或是构建设施运行测试的方式，几乎所有关于“优秀”单元测试的内容都依赖于上下文，所以对于造成某些程序员紧迫的话

题，却无法打动另外一些人，这毫不奇怪。

第 7 章紧接第 2 章，探讨什么是可测的设计。先大致阐述了有用的原则，并澄清我们其实是在寻找模块化设计，然后我们学习了基本的可测性问题，它会导致代码不可测。该章最后给出一些简单的指南，来保证我们走在可测设计的康庄大道上。

第 8 章抛出一个突如其来的问题，如果我们用 Java 之外的编程语言来编写单元测试如何？ Java 虚拟机允许当今的程序员采用一些另类的编程语言，并将它们与普通 Java 代码集成到一起。

第 9 章回到现实的挑战，应对逐渐缓慢的构建及延迟的测试结果。我们从两方面同时寻找方案，一方面是测试代码，考虑如何令作为构建一部分的代码加速，另一方面是基础设施，考虑如何从更快的硬件或不同的硬件分配方式中获得额外的吸引力。

尽管 JUnit 的声望和地位已经是 Java 社区中事实上的单元测试框架，但并非所有的 Java 程序员都熟悉这个伟大的开源库。我们附加了两个附录，来帮助那些还没有体会到 JUnit 高级特性的强大之处的人。

附录 A 提供用 JUnit 编写测试的入门介绍，以及当你告诉 JUnit 运行测试时，JUnit 如何来操作它们。浏览完这个附录后，你会更懂得编写测试，以及如何用 JUnit 的 API 来进行断言。

附录 B 旨在扩展 JUnit 的内置功能，于是深入地探讨其 API。我们不会试图细致地涵盖 JUnit 的所有方面，而是带你浏览两个扩展 JUnit 的常见方式——规则和运行器——这个附录会展示一些内置规则，它们不仅有用，而且让你知道如何与自定义插件打交道。

代码规范

本书中的代码示例包含 Java 源代码和大量标记语言以及输出代码清单。以代码清单的方式来展示较长的代码。较短的代码就内联在文字中。我们经常会在文字中提到代码清单。许多较长的代码清单具有带编号的注解，它们会在文字中提及。

下载代码

Manning 的网站 www.manning.com/EffectiveUnitTesting 上提供了本书的源代码打包下载。其中包含了本书的部分源代码，供你进一步钻研。

下载包中包括 Apache Maven 2 的 POM 文件，以及 Maven(http://maven.apache.org) 的安装使用说明，用来编译和运行示例。注意，下载包不包含各种依赖，而首次运行 Maven 时你需要连接互联网——Maven 会从互联网下载所需依赖。随后，你就可以断开网络，离线地运

行示例了。

代码示例用 Java 6 编写，你需要安装它以编译和运行示例。你可以从 www.oracle.com 下载合适的 Java 环境。（为了编译代码，你需要下载 JDK，而不是 JRE。）

我们还推荐安装合适的 IDE。你可以下载安装最新版的 Eclipse(www.eclipse.org) 或其他主流工具比如 IntelliJ IDEA(www.jetbrains.com) 或 NetBeans(www.netbeans.org)。只要你熟悉，这些工具都可以使用。

何去何从

这本书应当给你足够的洞察力，从而明显地提高你的单元测试能力。这是一条漫漫长路，肯定会有我们想不到或回答不了的问题。幸运的是，你会遇到很多同行，许多人会乐于分享和在线讨论测试代码的细微差别。

Manning 建立了在线论坛，你可以和 Manning 图书的作者们交谈。本书作者就在 Author Online 论坛上，网址为 www.manning.com/EffectiveUnitTesting。

在 testdrivendevelopment 和 extremeprogramming Yahoo! Groups 中，也有测试感染程序员的活跃社区。这些论坛是优秀的讨论区，不仅仅讨论单元测试。与此同时，或许你也能收获测试代码之外的一些新想法。

如果你寻找关于开发者测试的更加集中的论坛，那就去看看 CodeRanch（http://www.coderanch.com）及其优秀的 Testing 论坛。那儿有一群可爱的牛人。

最重要的是，我建议你积极地与工作伙伴讨论测试代码。我自己的某些最佳见解就是通过让别人看我的代码才得出的。

Acknowledgements 致　谢

当我签约写这本书时，我以为是个小工程。一切都那么简单，没什么不确定的。我太天真了。随着几周变成几个月，然后变成几年，我的一厢情愿也变得荡然无存。如果没有大家的帮助，这本书不可能问世，恐怕它现在还在进行中呢。

项目最初始于与 Manning 出版社的 Christina Rudloff 的邮件交谈，从那时起，我获得了大量帮助，非常感谢，也非常需要。

我要感谢 Manning 团队给予的支持和不懈努力（排名不分先后），Michael Stephens、Elle Suzuki、Steven Hong、Nick Chase、Karen Tegtmeyer、Sebastian Stirling、Candace M. Gillhoolley、Maureen Spencer、Ozren Harlovic、Frank Pohlmann、Benjamin Berg、Elizabeth Martin、Dottie Marsico、Janet Vail 和 Mary Piergies。

特别感谢各位领域专家和评审者为了改进本书而做出的努力，他们为本书带来了专业经验和知识。我要向他们致以最真挚的感谢（排名不分先后），Jeremy Anderson、Christopher Bartling、Jedidja Bourgeois、Kevin Con- away、Roger Cornejo、Frank Crow、Chad Davis、Gordon Dickens、Martyn Fletcher、Paul Holser、Andy Kirsch、Antti Koivisto、Paul Kronquist、Teppo Kurki、Franco Lombardo、Saicharan Manga、Dave Nicolette、Gabor Paller、J. B. Rainsberger、Joonas Reynders、Adam Taft、Federico Tomassetti、Jacob Tomaw、Bas Vodde、Deepak Vohra、Rick Wagner、Doug Warren、James Warren、Robert Wenner、Michael Williams 和 Scott Sauyet。

特别感谢 Phil Hanna 在印刷之前对手稿进行的技术审校。

最后，但也是同样重要的，我要感谢家人的长期支持。有时我觉得这本书的编写过程永无休止。在那些敲击着键盘的夜晚里，谢谢你们的理解，伴我走过艰难时期。

Contents 目　录

第二部分 目录

第一部分 Part 1

基　础

本书第一部分打算利用各个章节，建立读者和作者之间也就是你我之间共同的上下文。本书最终的目的是帮助你提高编写优秀测试的能力，第1章先从整体来看测试先行所带来的价值。然后讨论程序员生产力的动力学，以及各种对测试和测试质量的影响，最后会简单地介绍两种与自动化测试紧密相关的方法：**测试驱动开发**（Test-Driven Development，TDD）和**行为驱动开发**（Behavior-Driven Development，BDD）。

第2章挑起重担，定义了如何才能写出优秀的测试。简而言之，我们希望写出可读、可维护、可靠的测试。第2部分将会深入兔子洞，反转问题，检视一系列我们不希望看到的反例。

第一部分的末尾是第3章，它会谈及现代程序员最基本的工具之一——测试替身（test double）。我们将建立合理的用法，比如隔离代码以使其能够被恰当地测试，并且区分各种可能用到的测试替身的类型。最后，我们指导如何用好测试替身，帮助你绕开常见的坑，同时又能享受替身的好处。

读完前三章，你应当明白哪种测试是你希望编写的，以及为什么是那样的。你应当清楚地理解测试替身，将其作为常用工具。本书其余部分将根据这些基础展开，扶上马，送你一程。