TURING 图灵程序设计丛书

[PACKT] PUBLISHING

[西] Viktor Farcic Alex Garcia 著 袁国忠 译

Java 测试驱动开发

Test-Driven Java Development

中国工信出版集团

人民邮电出版社 POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Java 测试驱动开发

Test-Driven Java Development

[西] Viktor Farcic Alex Garcia 著
袁国忠 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

Java测试驱动开发 / （西）维克多·法西克
(Viktor Farcic)，（西）阿列克斯·加西亚
(Alex Garcia) 著；袁国忠译. -- 北京：人民邮电出
版社，2017.9
（图灵程序设计丛书）
ISBN 978-7-115-46501-6

Ⅰ. ①J… Ⅱ. ①维… ②阿… ③袁… Ⅲ. ①JAVA语
言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2017)第182978号

内 容 提 要

本书介绍如何将各种 TDD 最佳实践应用于 Java 开发，主要内容包括：用 Java 语言进行 TDD 会用到的各种工具和框架，所需环境搭建；通过实际应用程序，展示 TDD 优点及开发中应注意的主要问题；TDD 是如何通过模拟内部和外部依赖来提升速度的；如何重构既有应用程序；详细介绍所有 TDD 最佳实践。

本书适合所有 Java 开发人员，也适合用其他语言编程的程序员了解 TDD。

◆ 著　[西] Viktor Farcic　Alex Garcia
译　袁国忠
责任编辑　陈　曦
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京圣夫亚美印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：13
字数：307千字　2017年9月第1版
印数：1－3 500册　2017年9月北京第1次印刷
著作权合同登记号　图字：01-2017-4857号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

版权声明

前　言

测试驱动开发面世已有一段时间，但依然未被很多人采用，因为它难以掌握。虽然理论很容易，但要熟练使用，必须经过大量实践。

多年来，本书作者一直在使用TDD，并试图将其经验传授给你。身为开发人员，他们深信学习编码实践的最佳方式是编写代码和不断练习，本书秉承的正是这种理念——通过练习诠释所有TDD概念。本书犹如一次旅行，期间，你有机会将各种TDD最佳实践应用于Java开发。这次旅行结束时，你将成为TDD黑带，你的软件开发工具中也会多一个法宝。

本书内容

第1章阐述我们的目标——成为拥有TDD黑带的Java开发人员。要想知道我们将去往何方，必须对一些描述旅程的问题进行讨论，并找到答案。

第2章介绍并安装本书将用到的所有工具和框架，再搭建所需的环境。对于每个工具和框架，都将通过代码说明其优缺点。

第3章演示如何使用TDD的支柱——“红灯–绿灯–重构”过程开发一个“井字游戏”。我们将编写测试并确定其失败；然后编写实现测试的代码，运行所有测试并确定其通过；最后，重构并完善代码。

第4章开发“遥控军舰”应用程序，以充分展示TDD在单元测试中的威力。你将学习单元测试到底是什么、它与功能测试和集成测试有何不同以及它在测试驱动开发中扮演的角色。

第5章以传统方法开发Connect4游戏。这个开发过程中，没有编写任何测试，而等到开发结束后才编写。通过这样做，你将认识到开发应用程序时，如果不采用使其易于测试的开发方法，将面临什么样的难题。

第6章阐述速度对TDD来说至关重要。为快速演示一些理念和概念，我们将扩展前面开发的“井字游戏”，并使用MongoDB存储数据。所有测试实际上都没有使用MongoDB，因为我们将模拟所有与MongoDB的通信。

第7章讨论如何使用BDD方法开发一个书店应用程序。我们将以BDD方式制定验收标准，分别实现各项功能。通过运行BDD场景确认每项功能都能正常工作，并在必要时重构代码使其达到预期的质量水平。

第8章介绍如何重构既有的应用程序。我们将首先为既有代码创建测试，然后不断重构，直到测试和代码都满足预期。

第9章演示如何开发一个斐波那契数列计算器，以及如何使用功能开关隐藏还未完成或出于商业考虑不应向用户发布的功能。

第10章详细介绍所有TDD最佳实践，并温习通过阅读本书获得的知识和经验。

需要什么

为完成本书的练习，读者必须有一台64位计算机。对于各种需要用到的软件，本书提供了详尽的安装说明。

为谁而写

如果你是经验丰富的开发人员，想学习更有效的系统和应用程序开发方法，那么本书就是为你而写的。

排版约定

为将不同类型的信息区分开来，本书使用了很多文本样式。下面列出其中一些样式及含义。

正文中的代码、数据库表名、文件夹名称、文件名、文件扩展名、路径名、URL、用户输入和Twitter账号，使用如下样式：

“通过使用指令`include`，可包含其他上下文。”

代码块使用如下样式：

```
public class Friendships {
    private final Map<String, List<String>> friendships = new
      HashMap<>();

    public void makeFriends(String person1, String person2) {
        addFriend(person1, person2);
        addFriend(person2, person1);
    }
```

命令行输入或输出使用如下样式：

```
$> vagrant plugin install vagrant-cachier
$> git clone thttps://bitbucket.org/vfarcic/tdd-java-ch02-example-
  vagrant.git
```

新术语和**重要词语**使用粗体。例如，对于出现在屏幕上的菜单或对话框中的词语，表示如下：

“输入查询后，将看到按钮**Go**，请单击。”

警告或重要的注意事项。

提示和技巧。

读者反馈

欢迎提供反馈，请将你对本书的看法告诉我们：哪些方面是你喜欢的，哪些方面你不喜欢。读者反馈对我们来说很重要，因为这可以帮助我们推出更符合读者需求的著作。

要给我们提供反馈，只需向feedback@packtpub.com发送电子邮件，并在邮件主题中指出书名。

如果你有擅长的主题，并有志于写书或撰稿，请参阅www.packtpub.com/authors的撰稿指南。

客户支持

购买本社图书后，你将获得各种帮助，让手中图书最大限度地发挥功效。

勘误

虽然我们力图让图书内容准确无误，但错误仍不可避免。如果你在本社图书中发现错误（包括正文和代码），请告诉我们，我们将感激不尽。你这样做不仅可以让其他读者免遭同样的挫折，还可帮助我们改进该书的后续版本。无论你发现什么错误，都请告诉我们。为此，可以访问http://www.packtpub.com/submit-errata，输入书名，单击链接**Errata Submission Form**，再输入错误详情。提交的勘误得到确认后，将被上传到我们的网站或添加到既有的勘误列表。

要查看已提交的勘误，请访问https://www.packtpub.com/books/content/support，并在搜索框中输入书名，**Errata**栏将列出你搜索的信息。

打击盗版

网上散布的盗版材料是各类媒体屡禁不绝的问题。在保护版权和许可方面，本社的态度非常严肃。如果你在网上看到本社作品的非法复制品，请马上把网址或网站名告诉我们，以便我们能够采取补救措施。

请通过copyright@packtpub.com与我们取得联系，并提供可疑的盗版材料链接。

感谢你为保护我们的作者提供的帮助，也十分感激对于我们提供有价值内容的能力给予的保护。

问题

只要有与本书相关的问题，都可通过questions@packtpub.com与我们联系，我们将尽力解决。

电子书

扫描如下二维码，即可购买本书电子版。

目　录

第1章 为何要关心测试驱动开发

本书的作者是开发人员，针对的读者也是开发人员，因此大部分学习都将通过代码进行。每章都将介绍一个或多个TDD实践，读者将通过完成套路掌握它们。在空手道中，套路（kata）是一种练习，学习者不断重复同样的招式，每次重复都进步一点点。同样，读者阅读每章后，都将有细微但意义重大的进步。你将学习如何改善设计和代码、缩短上市时间、提供与时俱进的文档、通过质量测试提高代码覆盖率以及编写行之有效的清晰代码。

旅程都有起点，本书也不例外。我们的目标是让你成为拥有**测试驱动开发（TDD）**黑带的Java开发人员。

为确定我们将走向何方，必须就一些决定航程的问题进行讨论并找到答案。何为TDD？这是一种测试方法还是别的什么东西？使用TDD有何好处？

本章旨在提供针对TDD的整体概括，帮你了解TDD定义及其优势。

本章涵盖如下主题：

- 理解TDD；
- 何为TDD；
- 测试；
- 模拟；
- 可执行的文档；
- 无需调试。

1.1 为何要使用 TDD

你所处的环境使用的可能是敏捷开发方法，也可能是瀑布开发方法；你们公司可能有明确的规程，这些规程经过了多年艰苦奋斗的洗礼；也可能，你们只是一家刚刚起步的创业公司。无论如何，你都很可能面临过下述一个乃至更多痛点、问题或导致交付失败的原因：

- 部分团队成员无缘参与需求、规范或用户故事的制定；
- 大部分乃至全部测试都是手动的，抑或根本就没有测试；
- 虽然使用了自动化测试，但并未检测出真正的问题；
- 编写并执行自动化测试的时间太晚，无法给项目带来真正的价值；
- 总是有更紧急的问题需要处理，没法腾出专门用于测试的时间；
- 整个团队分为测试、开发和功能分析小组，而这些小组常常不能同步；
- 无法重构代码，因为担心这样做会破坏既有的功能；
- 维护成本高；
- 上市时间过长；
- 客户觉得交付的产品不符合要求；
- 文档从来都不是最新的；
- 害怕部署到生产环境，因为结果无法预料；
- 常常无法部署到生产环境，因为运行回归测试的时间太长；
- 团队为搞清楚某些方法或类的作用花费的时间太多。

测试驱动开发并不能神奇地解决所有这些问题，而只为我们找到解决方案指明方向。世上没有灵丹妙药，但如果有什么开发实践能让众多层面的情况大不相同，那就是TDD。

测试驱动开发可缩短上市时间、简化重构工作、帮助创建更好的设计以及降低耦合程度。

除这些直接的好处外，TDD还是众多其他实践（如持续交付）的前提条件。使用TDD可改善设计和代码的质量、缩短上市时间、确保文档最新、获得极高的代码覆盖率等。

要掌握TDD并不那么容易。即便学习了所有的理论，仔细研究了最佳实践和反模式，旅程也才刚刚开始。要掌握TDD需要很长的时间和大量的实践，这是漫长的过程，绝不是阅读完本书就能结束的。事实上，这个过程根本就没有结束的时候，因为总是有新的方式面世，让你能够更熟练、更快捷地使用TDD。然而，需要付出的代价虽然很高，但带来的好处更多。使用TDD的时间足够长的人都宣称没有其他开发软件的方法，我们就是这样的人，你肯定也会成为其中一员。

学习编码技巧的最佳方式是实践，我们对此深信不疑。要掌握本书介绍的内容，仅在上班的路上翻阅还不够；这不是一本适合躺在床上阅读的书，你必须撸起袖子动手编写代码。

本章将介绍基础知识，但从下一章开始，你将通过阅读、编写和运行代码进行学习。我很想说等阅读完本书后，你就是经验丰富的TDD程序员了，但情况不是这样的。读完本书，你将熟悉TDD，并拥有坚实的理论和实践基础，而剩下的事就全靠你自己了。要想获得更多TDD经验，你必须在日常工作中使用TDD。

1.1.1 理解 TDD

此时你可能正自言自语："我知道TDD会带来一些好处，但测试驱动开发到底是什么呢？"TDD是一种简单的流程，要求你先编写测试，再编写实现代码，这与"编写代码后再测试"的传统方法相反。

1.1.2 红灯–绿灯–重构

测试驱动开发是一个过程，依赖于不断重复极短的开发周期。它基于**极限编程（XP）**的测试优先理念，倡导采用可高度信赖的简单设计。驱动这个流程前行的开发周期称为**"红灯–绿灯–重构"**。

这种流程本身很简单，由几个反复进行的步骤组成：

(1) 编写一个测试；
(2) 运行所有测试；
(3) 编写实现代码；
(4) 运行所有测试；
(5) 重构；
(6) 运行所有测试。

鉴于测试是在实现前编写的，因此它应该不能通过。如果通过了，就说明测试是错误的：要么它描述的功能早已存在，要么编写不正确。编写测试期间处于绿灯状态昭示着存在错报的问题，对于这样的测试，应将其删除或进行重构。

编写测试时，应处于红灯状态。完成测试要求的实现后，所有测试都应通过，此时将处于绿灯状态。

如果最后一个测试未通过，就说明实现不正确，必须修正：要么这个测试不正确，要么实现代码不符合我们制定的规范。如果其他测试未通过，就说明我们破坏了某种功能，必须撤销所做的修改。

在这种情况下，一种自然而然的反应是：花足够的时间修复代码，让所有测试都通过。然而，这样的做法是错误的。如果不能在几分钟内完成修复，最佳的选择是撤销所做的修改。毕竟修改前一切都正常，带来破坏的实现显然是错误的。为何不到原来的地方，重新考虑实现测试的正确方式呢？这样我们只是在错误的实现上浪费了几分钟，而不会为修复一开始就不正确的东西浪费更多时间。原有的测试覆盖率（不包括最后一个测试的实现）应该很高。我们通过有意识地重构来修改既有代码，而不将其作为修复最近编写的代码的方式。

不要试图让最后一个测试的实现完美无缺，而应只编写足以让这个测试通过的代码。

你可以任何喜欢的方式编写代码，但要快。一旦进入绿灯状态，我们就知道存在一个由测试构成的安全网，可接着重构代码了：改进和优化代码，但不引入新功能。重构结束后，所有测试应当在任何情况下都能通过。

如果重构期间有测试未通过，就说明重构破坏了既有功能，应像以前一样撤销所做的修改。在重构阶段，我们不修改任何功能，也不引入新的测试，而只改进代码，并不断运行所有测试，确保没有破坏任何功能。与此同时，我们证明了代码是正确的，并降低了未来的维护成本。

重构结束后，再重复整个过程。这是一个无限循环，每次循环都是一个极短的周期。

1.1.3　速度是关键

想想打乒乓球的情形吧。这项运动的节奏非常快，职业选手玩起来可能让人目不暇接，TDD与这项运动很像。TDD老手通常不会让接球（编写测试或实现的）时间超过一分钟：编写简短的测试并运行所有测试（乒），编写实现并运行所有测试（乓），再编写一个测试（乒），编写该测试的实现（乓），重构并确认所有测试都通过（计分）；然后重复上述过程：乒、乓、乒、乓、计分。不要试图让代码完美无缺，而应力图让球不断运动，直到需要计分（重构）为止。

测试和实现的切换时间应以分钟甚至秒计。

1.1.4　TDD 并非测试方法

TDD中的T常常遭人误解。测试驱动开发是一种设计方法，要求在编写代码前考虑实现以及代码需要提供的功能，且每次只关注一项功能的需求和实现——这有助于理清思路以及更好地组织代码。这并不意味着使用TDD时编写的测试毫无用处，甚至恰好相反：它们很有用，让我们能够以极快的速度进行开发，同时不担心破坏既有功能。对重构来说这显得尤为重要：能够在不担心破坏既有功能的情况下重新组织代码对改善质量大有裨益。

测试驱动开发的主要目标是提供可测试的代码设计，测试只是一项很有用的副产品。

1.2 测试

虽然测试驱动开发主要是一种代码设计方法，但测试也是其中一个很重要的方面，因此我们必须对如下两种测试方法有清晰的认识：

- 黑盒测试；
- 白盒测试。

1.2.1 黑盒测试

黑盒测试（也叫功能测试）将受测软件视为一个黑盒，无需知道其内部构造。这种测试是通过软件界面进行的，旨在确认它们像预期的那样工作。只要界面的功能未变，测试就应通过——即便内部构造发生了变化。测试人员知道程序该做什么，但不知道它是如何做的。黑盒测试是传统组织最常使用的测试类型。这种组织通常将测试人员划归到一个独立的部门——在测试人员不熟悉编程、难以理解代码时尤其如此。这种测试方法提供了外部观察受测软件的结果。

下面是黑盒测试的一些优点：

- 可高效测试大块代码段；
- 无需访问和理解代码，也不要求测试人员知道如何编写代码；
- 将用户角度和开发人员角度分离。

下面是黑盒测试的一些缺点：

- 覆盖率有限，因为只执行部分测试场景；
- 测试效率低下，因为测试人员对软件内部构造一无所知；
- 测试缺乏针对性，因为测试人员对应用程序的了解有限。

用于驱动开发的测试通常是根据验收标准进行的，而验收标准决定了要开发哪些功能。

自动化黑盒测试依赖于某种形式的自动化，如**行为驱动开发**（**BDD**）。

1.2.2 白盒测试

白盒测试（也叫透明盒测试、玻璃盒测试和结构测试）查看受测软件的内部，并将由此获得的知识用于测试过程。例如，如果在特定条件下应引发异常，可能需要在测试中重现这种条件。白盒测试要求测试人员了解系统的内部结构，同时具备编程技能；它提供了从内部观察受测软件的结果。

下面是白盒测试的一些优点：

- 可高效找出错误和问题；
- 知道受测软件的内部构造有助于进行详细测试；
- 能够发现隐藏的错误；
- 可帮助程序员反省；
- 有助于优化代码；
- 由于知道软件的内部构造，因此可最大限度地提高测试覆盖率。

下面是白盒测试的一些缺点：

- 可能无法发现未实现或缺失的功能；
- 需要对受测软件的内部构造有大致认识；
- 需要访问代码；
- 测试通常与产品代码的实现细节紧密耦合，导致重构代码后原本应该通过的测试未能通过。

白盒测试几乎都是自动化测试，且在大多数情况下都是单元测试。

在实现前执行的白盒测试是以**TDD**方式编写的。

1.2.3　质量检查和质量保证的差别

还可根据要达成的目标对测试方法进行分类。要达成的目标通常有两种：**质量检查（QC）**和**质量保证（QA）**。质量检查的重点是发现缺陷，而质量保证力图将缺陷消灭在萌芽状态。QC是面向产品的，旨在确保结果符合预期；而QA更专注于过程以确保制造质量，即力图确保以正确的方式做正确的事情。

质量检查过去扮演的角色更重要，但随着TDD、**验收测试驱动开发（ATDD）**和**行为驱动开发（BDD）**的面世，重点正转向质量保证。

1.2.4　更好的测试

无论使用黑盒测试、白盒测试还是两者兼而有之，编写测试的顺序都非常重要。

需求（规范和用户故事）是在实现需求的代码之前编写的，因此是它们定义了代码，而不是相反。对测试来说亦如此。如果它们是在代码之后编写的，那么从某种意义上说，是代码（及其实现的功能）定义了测试。由既有应用程序定义的测试有失偏颇，倾向于确认代码的功能，而不