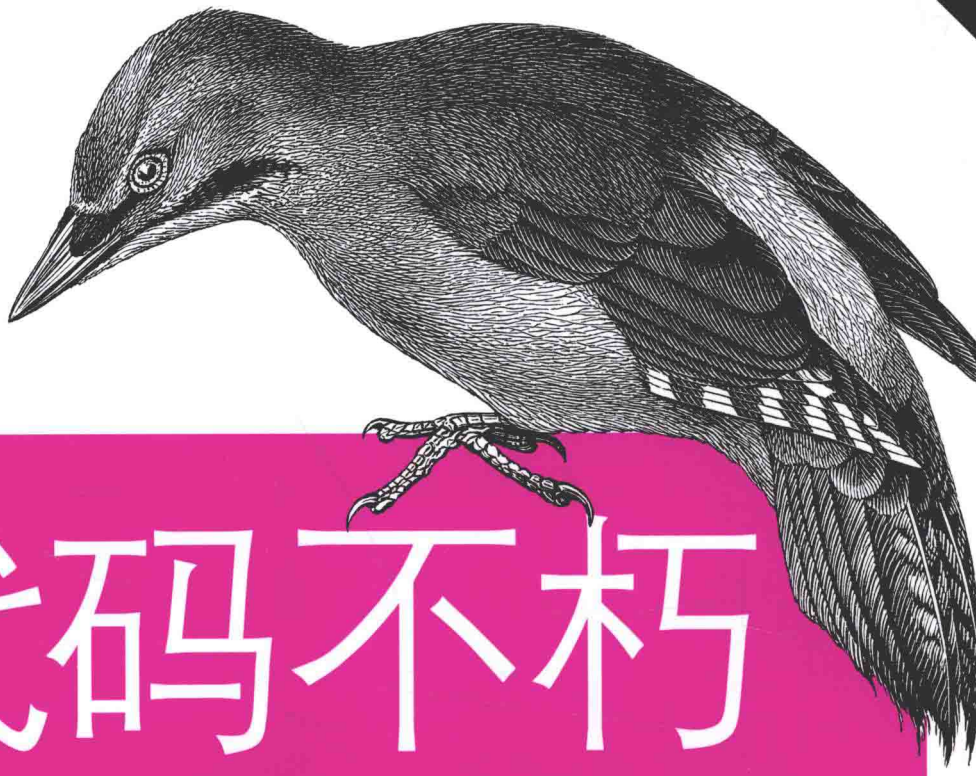
O'REILLY®

Broadview®
www.broadview.com.cn

Java版



# 代码不朽

## 编写可维护软件的10大要则

Building Maintainable Software

[荷] Joost Visser 著
张若飞 译

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

# 代码不朽

## 编写可维护软件的10大要则（Java版）

Building Maintainable Software

[荷] Joost Visser 著
张若飞 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

## 内 容 简 介

人类到目前为止已经能够度量越来越多的东西，例如时间、长度等，但是在软件开发领域，我们依然很难去评估一个软件系统的质量，以及维护它的难易程度。可维护性越差，意味着开发成本越高、开发速度越慢，以及由于改动带来的缺陷也越多。在现实中，我们经常会面对代码混乱、模块紧耦合的遗留系统，持续攀升的维护难度会最终导致系统不可维护，从而推倒重来。来自软件改进组织（Software Improvement Group）的咨询师们，从大量实践项目中提取出了编写可维护软件的10个最佳原则，不仅可以用来测量软件的质量和可维护性，还可以指导我们如何编写出高质量的代码。本书会一一介绍这些原则，并且提供了翔实的代码示例，能够让读者一步步了解到如何对代码进行重构，从而达到满足原则、提高可维护性。本书中的代码示例都采用Java语言编写，但是背后的原则也适用于使用其他语言的开发人员。

希望各位读者在阅读完本书后，能够了解和掌握如何对软件系统的质量进行评估和测量，以及如何在实践中遵循书中的原则，编写出高质量、简洁的代码，开发出松耦合、高可维护性的系统。



版权贸易合同登记号 图字：01-2016-4316

**图书在版编目（CIP）数据**

代码不朽：编写可维护软件的10大要则：Java版 /（荷）约斯特·维瑟（Joost Visser）著；张若飞译. —北京：电子工业出版社，2016.10
书名原文：Building Maintainable Software
ISBN 978-7-121-29704-5
Ⅰ. ①代… Ⅱ. ①约… ②张… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312.8
中国版本图书馆CIP数据核字(2016)第190897号

责任编辑：张春雨
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
开　　本：787×980　1/16　印张：10.25　字数：224千字
版　　次：2016年10月第1版
印　　次：2016年10月第1次印刷
定　　价：69.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819 faq@phei.com.cn。

# O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

## 业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

# 序言

“上医治未病，中医治欲病，下医治已病。”

——源自《黄帝内经》

软件是当今信息社会的 DNA。DNA 虽然小到肉眼无法察觉，它却决定着世界的一切。同样，软件虽然无形且深奥莫测，它却主宰着信息的动向。我们生活中对软件的依赖无处不在：教育、管理、生产、贸易、旅行、娱乐、社交等。它是如此普遍，以至于我们理所当然地认为：软件会一如既往地提供着已有的一切功能，并且会不断发展以满足我们日益增长的需求。

不仅仅中国在依赖着软件，整个世界都在依赖着中国制造的软件。毫无疑问，在不久的将来，中国会成为制造日益精进的软件的核心大国。

我和我来自 Software Improvement Group（SIG）的同事们期望通过这本书，将我们在过去十五年中积累的软件分析经验分享出来，以回馈软件工程界。 我们荣幸地为全球，包括中国在内的客户们，诊断软件系统的健康程度，并提出改进的意见与建议，以确保软件代码能够经受时间的考验，在一个良好健康的系统环境下扩展。

在过去的十五年中，我们看到了千百万行优美的代码，同时也看到了不计其数的拙劣的代码。我们从中学会了诊断代码并且对症下药。现在，我们把所学到的经验总结成十个关于构建可维护软件的基本原则回馈给社会。

这十个基本原则旨在帮助软件开发人员编写出能经受时间考验的代码。尽管软件近几十年才开始存在，中国传统的哲学理念依旧适用。防患于未然永远好过亡羊补牢。从写下第一行代码时，可维护性就应该获得开发人员的重视，并成为贯穿始终的基本理念。

——Joost Visser, 阿姆斯特丹 , 2016 年 6 月

# 关于作者

Joost Visser 是 Software Improvement Group（SIG）的研究负责人。SIG 提供了许多用来测量和精通软件开发的方法和工具，而他的职责是发现它们背后的科学理论。Joost 也是荷兰内梅亨拉德伯德大学（Radboud University Nijmegen）“大规模软件系统”专业的教授。他在阿姆斯特丹大学获得博士学位，并发表了 100 多篇有关泛型编程、程序变换、绿色计算、软件质量以及软件演化等方面的论文。Joost 认为软件工程是一门社会化的技术学科，并且相信懂得如何测量软件是开发团队和产品负责人必须要学会的技能。

Pascal van Eck 在 2013 年作为一名软件质量顾问加入 SIG。在加入 SIG 之前，他在荷兰屯特大学信息系统专业做了 12 年的助理教授。Pascal 拥有阿姆斯特丹自由大学（Vrije Universiteit Amsterdam）计算机科学专业的博士学位，发表过 80 多篇有关企业级架构、IT 安全以及软件测量指标的论文。Pascal 还是荷兰数码国际架构会议程序委员会的主席。

Rob van der Leek 在 2015 年获得代尔夫特理工大学（Delft University of Technology）软件工程专业的硕士学位，随后加入 SIG 成为了一名软件质量顾问。对于 Rob 来说，在 SIG 的工作就像是在做一名软件医生。作为顾问，他将软件工程和软件技术方面的专业知识结合起来，向客户提供如何保持系统健康的建议。之后，Rob 成为了 SIG 内部开发团队的负责人。这个团队负责开发和维护 SIG 的软件分析工具。Rob 的理想是通过自己的努力，让 IT 行业逐渐变得更好。

Sylvan Rigal 从 2008 年起就从事 IT 咨询工作，并于 2011 年加入 SIG，成为一名软件质量顾问。他帮助许多客户降低了软件维护的成本，并且通过调整软件设计和开发过程中相关改进点的优先级，提高了软件整体的安全性。他拥有荷兰马斯特里赫特大学（Maastricht University）的硕士学位。作为 SIG 软件安全团队中积极的一份子，Sylvan 会培训其他顾问如何分析软件安全风险。在工作之余，他是一名巴西柔术教练，喜欢去阿姆斯特丹的餐馆品尝美食，或者在亚洲各处旅游。

Gijs Wijnholds 于 2015 年加入 SIG，是公共管理方面的一名软件质量顾问。他擅长通过优化开发过程，将技术风险转化成策略决策，来帮助客户更好地控制软件工程。Gijs 拥有乌得勒支大学（Utrecht University）人工智能专业的学士学位，以及阿姆斯特丹大学（University of Amsterdam）的逻辑学硕士学位。他还是一名 Haskell 语言和数学语言学（mathematical linguistics）方面的专家。

# 前言

简单出真知。

——歌德[1]

在 SIG 经历了长达 15 年有关软件质量的咨询工作后，我们在可维护性方面学习到了不少经验。

首先，在软件开发过程中，对可维护性的忽视是一个确实存在的问题。可维护性低意味着开发人员需要在维护和修复旧代码方面花费更多的时间和精力。相应地，留给软件开发中最有价值的部分——编写新代码的时间就少了。我们的经验和收集到的数据都表明，低于平均值与高于平均值的可维护性相比，在维护源代码方面至少相差两倍的工作量。我们会在附录 A 中介绍如何来测量可维护性。

第二，很大程度上，可维护性随着一些小问题的不断发生而变得越来越低。因此，提升可维护性的最高效、最有效的方式，就是找出这些小问题。提升可维护性不需要什么魔法或者高科技，一些简单的技能和知识，再加上对它们的遵守和使用环境，就可以让可维护性有一个飞跃的提升。

在 SIG，我们见过已完全无法再维护的系统。在这些系统中，bug 没有被修复，功能没有被修改或扩展，最终原因都是因为时间太长、风险太大。不幸的是，这种情况在今天的 IT 行业中非常普遍，但是本不该如此。

这也是为什么我们编写这 10 条原则的原因。我们希望将每一个一线开发人员都应该掌握的知识和技能分享出来，让每个人都能不断写出高可维护性的代码。我们相信，作为软

1 约翰·沃尔夫冈·冯·歌德（1749 年 8 月 28 日—1832 年 3 月 22 日），出生于德国法兰克福，戏剧家、诗人、自然科学家、文艺理论家和政治人物。

件开发者的你，在阅读并理解这10条原则后，一定能写出高可维护性的代码。剩下的就是，创造出让这些技能发挥最大效果的开发环境，包括互相共享的开发实践、适合的工具等。我们将在第二本书《构建软件团队》（*Building Software Teams*）中介绍这些开发环境。

# 本书的主题：构建可维护软件的十个原则

后续章节中所介绍的原则与系统的类型无关。这些原则都是关于代码单元（Java中的方法）大小及参数数量、代码中决策点的数量，以及源代码等方面的讨论。可能许多开发人员都已经对它们广为熟悉，至少在培训中应该或多或少都听说过一些。这些章节还会提供示例代码（大多数以重构的形式），来帮助读者在实践中掌握这些原则。虽然这些原则都是基于Java语言介绍的，但是它们不受所使用的编程语言的限制。这些原则其中的4/5，大概都来自SIG/TÜViT[2]认证产品可维护性评估标准（Evaluation Criteria for Trusted Product Maintainability[3]），它是一组用于系统化评估源代码可用性的指标集合。

# 你为什么应该阅读本书

单独来看本书中的每一条原则，可能都已经被开发人员广为熟知。事实上，许多常用的代码分析工具，都会检查其中的一部分原则。例如，Checkstyle（*http://checkstyle.sourceforge.net*）（用于Java）、Style-Cop+（*http://stylecopplus.codeplex.com*）（用于C#）、Pylint（*http://www.pylint.org*）（用于Python）、JSHint（*http://jshint.com*）（用于JavaScript）、RuboCop（*https://github.com/bbat sov/rubocop*）（用于Ruby），以及PMD（*https://pmd.github.io*）（可用于多种语言，包括C#和Java），这些工具都会检查代码是否符合第2章中所介绍的原则。但是，以下3个特点是本书区别于其他一些软件开发书籍的地方：

我们根据经验选择了10条最重要的原则

> 代码风格工具和静态代码分析工具可能会让人望而却步。Checkstyle 6.9包含了近150个规则，每个都代表着一条原则。虽然它们都有意义，但是对提高可维护性的效果却不相同。我们已经从中选出了10个对提升可维护性最有效的原则。下一页中的“为什么选择这十条原则？”中解释了我们是如何来选择这些原则的。

我们会告诉你如何才能符合这10条原则

> 告诉一个开发人员什么应该做或者什么不应该做是一回事（正如很多工具做的那样），教会他们如何做到是另一回事。在本书中，对于每一个原则，我们都提供了

2 TÜViT是TÜV（德国的一个从事技术质量管理的全球性公司）的一部分。它主要专注于IT和安全方面的认证咨询。

3 请参考可维护性评估标准（*http://bit.ly/eval_criteria*）。

翔实的示例代码，一步一步讲解如何编写符合原则的代码。

我们提供来自于真实系统的统计数据及示例代码

在SIG，我们已经见过了大量由开发人员在各种实际限制条件下编写的源代码，其中包含了各种妥协的处理。因此，我们将自己的基准测试数据分享出来，让读者了解到现实中的代码与理想中的差距。

# 谁应该阅读本书

本书的目标读者是使用Java语言编程的软件开发人员。在这些读者中，本书又针对两个不同的人群各有侧重点。第一个人群是那些接受过计算机科学或软件工程方面专业教育的开发人员（例如，大学主修这两个专业之一的）。对于这样的开发人员，本书可以巩固他们在专业编程课程上所学的基础知识。

第二种是没有进行过计算机科学或软件工程专业学习的软件开发人员。我们认为这些开发人员主要是一些进行自学或者大学主修完全是其他专业的人员，但是他们后来又从事了软件开发这个行业。我们的经验是，这类人员除了熟悉所用语言的语法和语义之外，很少接受其他的专业培训。这也是我们在编写本书时特别考虑的人群。

## 为什么选择这10条原则？

本书包含了10条原则。前8条与SIG/TÜViT认证产品的可维护性评估标准（它是SIG评估可维护性的理论基础）中的**系统属性**一一对应。对于SIG/TÜViT评估标准，我们按照如下原则来选择评估指标：

- 数量尽可能少
- 与技术无关
- 易于测量
- 可以与实际的企业软件系统进行有意义的比较

因此，我们就选出SIG/TÜViT评估标准中的这8个系统属性。而添加另外两个原则（关于整洁代码和自动化测试），是考虑到它们是最关键的，并且可以由开发人员直接控制。

计算机科学和软件工程中的研究人员已经定义了非常多的源代码指标。不管你怎么数，几十个指标总是有的。因此我们这里提炼出的8个系统属性，显然只是所有可维护性指标中的很小一部分。

但是，我们想说的是，这 8 个 SIG/TÜViT 指标是完全适合并且足够测量可维护性的，因为它们解决了以下几个问题：

**依赖于具体技术实现的指标**

有些指标（例如，继承深度）与具体使用的技术（例如，只有在面向对象的语言中才存在继承关系）有很大的关系。但是在现实中，面向对象还远没有达到完全统治的地位，因此我们也需要考虑评估大量非面向对象代码（例如用 Cobol、RPG、C 和 Pascal 编写的系统）的可维护性。

**与其他指标紧密相关的指标**

有些指标与其他指标之间有非常紧密的关系，系统中的决策点总数就是一个例子。实验证明，这个指标与代码量有直接的关系。这意味着一旦你知道系统中代码行的总数（这很容易测量），那么就几乎可以非常准确地预测出决策点的数量。我们没理由去选择那些较难于测量的指标，因为与较容易测量的指标相比，你不得不花费更多的精力来执行并统计结果，但是又得不到的内容和价值。

**在实践中没有区别的指标**

有些指标从理论角度看很好，但是在软件开发实践中，它在所有系统上的表现都几乎一样。我们没理由将这些指标作为评估标准，因为无法用它们的结果来区分各个系统。

# 本书不包括哪些内容

本书使用 Java 语言（本书中的唯一一种语言）来阐述和解释我们的原则。但是我们并不是要教大家如何使用 Java。我们会假设读者至少可以阅读 Java 代码和 Java 标准库的 API，并且尽可能地保证示例代码足够简单，只使用 Java 语言的基本特性。

这也不是一本介绍 Java 习惯用法的书，也不是要告诉大家什么才是符合 Java 习惯的代码。我们不相信熟练使用某种语言就可以达到高可维护性。相反，本书中的原则在很大程度上都与语言无关，因此也与语言的习惯用法无关。

虽然我们在书中会介绍或解释许多重构模式，但我们并不是想写一本关于这些模式的书。市场上已经存在了很多关于模式的优秀书籍和网站。我们这本书只关注为何选择这些重构模式，以及它们如何能提高可维护性。因此，这本书只是学习重构模式的一个起点。

# 下一本书

我们知道，单个开发人员并不能控制开发流程的方方面面。使用哪些开发工具、质量控制如何管理、如何搭建部署环境等，这些都是影响软件质量的重要因素，但它们同时也是一个“团队”的责任。因此，这些主题已经超出了本书的范围，我们会在下一本书《构建软件团队》中介绍这方面的最佳实践，以及如何测量它们的结果。

# 关于 SIG

虽然本书封面只列出了一个作者名字，但是本书的真正作者不止一人。真正的作者是SIG——一个软件管理咨询公司。可以说，这本书提炼了 SIG 所有顾问从 2000 年以来，在测量软件质量和提出建议过程中总结的集体经验和知识。我们运营着唯一一个经过认证[4]的软件分析实验室，可以按照 ISO 25010 国际标准，对软件产品的质量进行标准化的检测。

SIG 提供的服务之一是我们的“软件风险监控”服务。我们的客户通过该服务，定期（通常每周）上传他们的源代码。然后我们的软件实验室会自动对上传源代码进行检测。SIG 顾问会评估所有自动分析出的异常情况，并与客户讨论。在本书编写时，SIG 已经总共分析了 71 亿行代码，每周有将近 7 千多万行代码被上传到 SIG。

SIG 成立于 2000 年。它的历史可以追溯到荷兰国家数学和计算机科学研究院。15 年之后的现在，我们仍然保持并重视与软件工程学术界的联系。SIG 顾问会定期在学术期刊上发表文章，并且许多博士论文都是基于对开发和提高 SIG 质量模型的研究。

# 关于此版本

这是本书的 Java 版本。所有代码示例都用 Java 编写（并且只用 Java 编写），文中经常提到的工具和名词都是在 Java 社区中所广泛使用的。我们假定读者都拥有一定的 Java 编程经验。如前文所述，本书中用 Java 展示的各条原则，实际上是与语言无关的。本书的 C# 版本由 O’Reilly 出版社负责出版发行。

# 相关书籍

我们列举了 10 条实现高可维护性的基本原则。虽然这可能是许多人关于可维护性方面阅读的第一本书籍，但是我们希望它不要成为最后一本。因此我们推荐读者继续阅读以下图书：

4 即 ISO/IEC 17025 认证。

《构建软件团队》，SIG 著

这是同一批作者编写的配套书籍。与本书关注于构建可维护软件的开发原则不同，这本书关注于软件开发流程的最佳实践以及如何使用“目标—问题—指标”的方法来有效地测量它们。《构建软件团队》一书将于 2016 年出版。

《重构：改善现有代码的设计》，Martin Fowler 著

这本书关注于提升现有代码的可维护性（以及其他质量特征）。

《代码整洁之道》，Robert C. Martin（也称为 Bob 大叔）著

与本书不同，这本书讲述的是如何编写高质量的软件源代码，但是它介绍了更高抽象层次的原则。

《代码质量》，Diomidis Spinellis 著

这本书也介绍了关于代码质量的原则，但是同《代码整洁之道》一样，它们都处于更高的抽象层次。

《设计模式：可复用面向对象软件的基础》，Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides（也称为四人帮）著

推荐那些想成为软件架构师的开发人员一定要阅读该书。

# 本书中使用的约定

本书使用以下印刷排版约定：

斜体（*Italic*）

表示新名词、URL、邮件地址、文件名及文件扩展名。

等宽字体（`Constant width`）

用于程序列表（包括段落中的），表示程序元素，例如变量或者函数名、数据库、数据类型、环境变量、语句和关键字。

该图标表示一个提示或建议。

该图标表示一个一般说明。

该图标表示一个提醒或警告。

该图标表示一个重要的备注。

# 源代码中各元素的命名方式

虽然本书中我们使用 Java 来示范可维护性的原则，但是它们并不是只能用于 Java。这些原则来源于 SIG 的可维护性模型，与具体技术无关，并且已经被应用于大约 100 种编程语言及相关技术中（例如 JSP）。

各种编程语言（理所当然的）在功能和语法上各不相同。例如，Java 使用关键字 `final` 来表示常量，而 C# 使用 `readonly`。另一个例子是 C# 提供了一个称为“分部类”的功能，而 Java（当前）不支持该功能。

除了语法方面的不同之处，编程语言在图书、教程以及规范中使用的术语也不一样。例如，几乎每种编程语言都有将一组代码行作为整体执行的概念。在 Java 和 C# 中，该概念被称为一个*方法*（method）。而在 Visual Basic 中，它被称为一个子程序（subroutine）。在 JavaScript 和 C 中，它被称为一个*函数*（function）。在 Pascal 中，它被称为一个*过程*（procedure）。

因此，与技术无关的模型需要通用名来对这些概念进行分类。表 P-1 展示了我们在本书中使用的通用名。

表 P-1. 对概念的通用分类以及在Java中的表示

| 通用名 | 通用定义 | Java 中的表示 |
|---|---|---|
| 单元 | 最小可独立执行的一段代码 | 方法或构造函数 |
| 模块 | 最小的单元集合 | 顶层类、接口或者枚举 |
| 组件 | 一个系统按照其软件架构定义的顶层划分 | 编程语言中没有相关定义 |
| 系统 | 整个代码库 | 编程语言中没有相关定义 |

以下列表进一步阐释了表 P-1 中的分类结构与 Java 编程实践之间的关系：

从最小到最大

表 P-1 中的分类概念按照从最小到最大的顺序排列。虽然单元本身由多个语句组成，但是语句不是一个可分类的结构。

同许多其他编程语言一样，Java 的语句和行数之间也有着复杂的关系。一行代码可能是一条或多条语句，但是一条语句也可能跨越多行代码。为了方便起见，我们只关心**代码行**（line of code, LOC），即源代码中任何以回车 / 换行结束的行，不包括空行以及只含有注释的行。

特定语言中没有定义某些概念

如表 P-1 所示，Java 中没有某些通用的概念。例如，在 Java 中，没有语法能用来描述一个系统的边界。之所以定义这个概念，是因为我们需要它。这并不是 Java 缺少语法的问题，而是实际中我们会通过其他方式来确定这些边界。

一些众所周知的通用概念无影响

你可能会奇怪，为什么表 P-1 中没有定义像*子组件*或者*子系统*这样众所周知的概念。原因很简单：我们在阐述本书中原则时用不到它们。

并不代表一个语言的所有分类结构

Java 中的分类结构远比表 P-1 中的要多。Java 有用来对类分组的包、接口，还有内部类。因为我们不需要在原则中使用它们，所以没有列举在表 P-1 中。例如，我们不需要用类和内部类的区别，来说明有关耦合的原则。

Java 中的包与我们在表 P-1 中提到的组件不同。在一个非常小的 Java 系统中，组件和包之间可能是一对一的关系。而在更大的 Java 系统中，通常包的数量要远比组件要多。

构建工具对通用术语无影响

在 Java 开发中，使用 Apache Ant 的团队有另一个分类概念：目标（target）。同样，使用 Maven 的团队可能会使用一些 POM 文件，每个文件对应系统中的某一部分。但是，在我们的原则中都不会涉及 Ant 目标或者 Maven POM 文件。虽然组件与 Ant 目标或 Maven POM 文件之间，可能会存在一对一的关系，但这并不是一定的。

组件并不是一个只有在 Java 中才有的概念。组件并不是 Java 中的包，也不是 Ant 的目标或者 Maven POM 文件。相反，组件是由系统的软件架构而决定的最高层的构建模块。这些模块只有在系统的架构设计图中才会出现。第 7 章会进一步解释组件的概念，并提供一些示例程序。

# 如何使用书中的代码示例

你可以在 *https://github.com/oreillymedia/building_maintainable_software* 下载本书的补充资料（代码示例、练习题等）。

本书的目的是帮助你完成工作。一般来说，如果本书提供了示例代码，你就可以用在自己的程序或者文档中。除非你改写了代码中的很大一部分，否则不需要联系我们申请使用权。例如，编写一段使用了本书中部分代码的程序不需要授权，但是售卖 O'Reilly 书中的代码或者分发含有代码的存储媒介（例如 CD-ROM）需要授权。引用本书及书中示例代码来回答问题不需要授权，但是在你的产品文档中引用大量的本书示例代码需要授权。

我们感谢但不强制你使用署名。通常，署名包括书籍名称、作者、出版商和 ISBN 号。例如”*Building Maintainable Software: Ten Guidelines for Future-Proof Code* by Joost Visser. Copyright 2016 Software Improvement Group B.V., 978-1-4919-5352-5“。

如果你觉得使用代码示例的目的，不在我们上面提到的授权范围之内，可以联系 *permissions@oreilly.com* 获得特别授权。

# Safari® Books Online

*Safari Books* Online（www.safaribooksonline.com）是一家应需而变的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。

Safari Books Online 是技术专家、软件开发人员、Web 设计师、商务人士和创意人士开展调研、解决问题、学习和认证培训的第一手资料。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit

Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，请访问我们的网站。

# 联系我们

请将对本书的评价和发现的问题通过如下地址告知出版者。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

我们在 *http://bit.ly/info_architecture_4e* 上列出了勘误表、示例和所有额外的信息。

要评论或者询问关于本书的任何技术问题，请发邮件到 *bookquestions@oreilly.com*。

要了解 O'Reilly 更多的图书、课程、会议和新闻，请访问我们的网站 *http://www.oreilly.com*。

我们的 Facebook 账号：*http://facebook.com/oreilly*

我们的 Twitter 账号：*http://twitter.com/oreillymedia*

YouTube 的观看网址：*http://www.youtube.com/oreillymedia*

# 感谢

我们想要感谢参与编写本书的以下人员：

- 感谢 Yiannis Kanellopoulos (SIG)，他是我们的项目经理，负责各个方面。
- 感谢 Tobias Kuipers (SIG)，他是本项目的发起人。