全球资深Java技术专家力作，世界级软件开发大师Robert C. Martin与Peter Kriens作序推荐。

系统、全面地讲解如何将模块化设计思想引入开发中，涵盖18个有助于实现模块化软件架构的模式。

PEARSON

HZ BOOKS 华章科技

Java 核心技术系列

Java 应用架构设计

模块化模式与OSGi

Java Application Architecture

Modularity Patterns with Examples Using OSGi

(美) Kirk Knoernschild 著

张卫滨 译

机械工业出版社
China Machine Press

Java 核心技术系列

Java 应用架构设计

模块化模式与OSGi

Java Application Architecture
Modularity Patterns with Examples Using OSGi

(美) Kirk Knoernschild 著
张卫滨 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java 应用架构设计：模块化模式与 OSGi/（美）克内恩席尔德（Knoernschild, K.）著；张卫滨译.
—北京：机械工业出版社，2013.10
（Java 核心技术系列）
书名原文：Java Application Architecture: Modularity Patterns with Examples Using OSGi

ISBN 978-7-111-43768-0

Ⅰ. J… Ⅱ. ①克… ②张… Ⅲ. JAVA 语言-程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2013）第 197705 号

本书版权登记号：图字：01-2012-3793

本书由全球资深专家撰写，世界级软件开发大师 Robert C. Martin 与 Peter Kriens 作序推荐。书中揭示了模块化的重要性，如何实现模块化，以及如何使用 OSGi 实现模块化架构。

本书分为三部分。第一部分（第 1~7 章）描述了需要模块化的理由。在这里，你将会看到模块化在设计软件中所扮演的重要角色，与此同时还会学习到为什么需要设计模块化的软件。第二部分（第 8~12 章）介绍了 18 个模式，这些模式会帮助你设计更为模块化的软件。第三部分（第 13~17 章）引入了 OSGi，然后阐述了如何使用本书中的模式来设计软件系统并充分发挥模块化平台的优势，并使用代码样例论证了我们的观点。

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：谢晓芳
藁城市京瑞印刷有限公司印刷
2013 年 9 月第 1 版第 1 次印刷
186mm×240mm • 17.75 印张
标准书号：ISBN 978-7-111-43768-0
定 价：69.00 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

本书赞誉

“基础永远不会过时。在本书中，Kirk 介绍了如何立足基础，以低成本有效地构建高质量的软件密集型系统。你会发现这本书写得很好、很及时并且全是务实的理念。”

——Grady Booch，IBM 院士

“与 GoF 的《设计模式》一样，Kirk 的这本书应该成为每一位企业级开发人员和架构师的必备品，对 Paremus 的每一位工程师来说，这本书是必备读物。”

——Richard Nicholson，OSGi 联盟主席、Paremus CEO

“通过写这本书，Kirk 为软件社区做出了重要的贡献：他分享了自己关于模块化的大量真知灼见，这些内容新手能理解、在计算机课堂上能讲授并且有经验的程序员也可参考。我希望本书能够有广泛的读者。”

——Glyn Normington，Eclipse Virgo 项目的领导者

“我们的行业需要开始思考模块化这个词——因而需要这本书！”

——Chris Chedgey，Structure 101 创始人兼 CEO

“在本书中，Kirk 为我们提供了在现实世界中进行模块化软件开发所需要的设计模式。尽管模块化确实有助于管理复杂性和创建更容易维护的软件，但是天下没有免费的午餐。如果你想获得模块化所提供的收益，那么购买本书吧。”

——Patrick Paulin，Modular Mind 咨询师和培训师

“Krik 巧妙地记录了使用 OSGi 和 Eclipse 运行时技术的最佳实践。为了更好地理解如何创建优秀的软件，每一位高级 Java 开发人员都需要阅读本书。”

——Mike Milinkovich，Eclipse 基金会执行总监

译者序

“分而治之”是解决复杂问题的有效方式。

面对业务功能复杂的企业级软件，我们会寻找各种方式和标准进行拆分，其目的无非是降低每一部分的复杂性并提高软件重用的便利性。但是，到目前为止，取得的效果并不理想。应用程序依然是庞然大物，难以进行维护和管理，而在重用方面，最常见的方式恐怕还是复制和粘贴。各方面似乎都不尽如人意。

在本书中，作者提供了解决这类问题的另一种方案，那就是模块化。借助模块化技术，我们可以提升软件的架构水平，填补架构师和开发人员在相互理解上的鸿沟，同时又能提升软件的可重用性，控制软件的复杂性。目前，在 Java 平台中，OSGi 是事实上的模块化标准。Java 原生的模块化实现 Jigsaw 已经从 Java SE 7 延期至 Java SE 8，最近又延期至 Java SE 9，而且 Jigsaw 能否得到其他 JDK 厂商的支持还有待观察。换句话说，Java 平台上的模块化技术还在不断发展中，但是本书介绍的模式和理念却具有通用性，虽然在书名上你可以看到 OSGi 的字眼，但是作者在介绍这些模式的时候，却在有意地与特定技术保持距离。只要你对系统进行了良好的设计和拆分，不管使用什么模块化技术，甚至没有模块化运行框架，你都会从中受益。

OSGi 技术在嵌入式领域取得了很大的成功，目前它越来越多地用于企业级应用服务器上，但是在企业级软件开发领域，它的使用并不广泛。这是因为 Java EE 和 OSGi 在诞生之初就是不同应用领域的两种技术，所以二者在理念上以及使用方式上都会有很多不兼容或冲突的地方。但是，这种现象正在发生着变化，随着 OSGi 企业级规范的不断完善，以及像 Apache Aries 和 Eclipse Gemini 这些参考实现的成熟，相信 Java EE 和 OSGi 之间的壁垒会逐渐打破，模块化在企业级软件开发中的发展值得期待。在本书中，有众多设计模块化软件的最佳实践，相信随着模块化技术的不断发展，它会越来越有价值。

在翻译本书的过程中，作者深厚的技术功底和广泛的知识涉猎都令我佩服，尤其是借此机会，更认识到 Bob 大叔那几本名著的价值。这些书值得我们一遍遍地仔细研读，在此推荐给大家。

感谢侯伯薇向我介绍了这本书，并将我引荐给出版社。在本书的翻译过程中，编辑关敏给予了很多的帮助和指导，她的热心和责任心让我很受感染，在此向她表示感谢。

在此，感谢我的家人，没有你们的支持，我很难把这项任务坚持完成，尤其是我的爱人和宝贝儿子。很抱歉在最近几个月中，没有抽出太多的时间陪伴你们。

在翻译的过程中，我尽可能做到准确，但肯定还会有纰漏之处，恳请读者朋友们批评指正，您可以通过电子邮件：levinzhang1981@gmail.com 或新浪微博：@张卫滨 1895 联系到我。希望这本书对您有用！

张卫滨

序 1

我开心得手舞足蹈！我在墙上跳舞，我在天花板上跳舞！我欣喜若狂，我喜出望外。我真的非常开心。

你可能会问："为什么呢？"我来告诉你为什么——因为终于有人读 John Lakos 的书了！

在 20 世纪 90 年代，John Lakos 写过一本名为《Large-Scale C++ Software Design》[⊖]的书。这本书很棒，它是一本开创性的书。这本书为大规模应用架构制订了很好的方案。

John 的书只有一个问题，那就是在书名中有"C++"这样的字眼，并且在出版的时候软件社区正在转向 Java，所以真正需要读那本书的人并没有读到它。

但是当时做 Java 开发的人并没有阅读任何关于软件设计的书籍，因为这些人可能只有 22 岁，整日坐在舒适的办公室里，使用着 Java，按日交易并梦想在 23 岁的时候成为亿万富翁。他们是如此抢手！

所以，十多年之后的现在就是这个样子了。我们成熟了一些，但也有很多失败。这些失败丰富了我们的经验。现在，需要回过头来看看自己所创建的 Java 架构废墟。我们怎么能如此幼稚？我们怎么能忽视 Jacobson、Booch、Rumbaugh、Fowler 以及 Lakos 所提出的原则呢？我们到底哪里做错了？

我要告诉你我们哪里做错了。Web 迷惑了我们，我们的思想过多地受到了 Twitter 的影响。我们认为 Web 是革命性的。我们认为 Web 改变了所有的事情。我们认为 Web 让所有旧的规则过时。我们认为 Web 是如此之新、如此具有革命性、如此具有颠覆性以至于忽略了既有的游戏规则。

为此我们付出了代价。我们已经为此付出了高额的代价。我们为巨大的、无法管理的设计付出了代价。我们为复杂的、混乱的代码付出了代价。我们为误导性的、没有指导性的架构付出了代价。我们为失败的项目、破产的公司以及破碎的梦想付出了代价。我们在付出、不停地付出。

经过了 15 年，我们终于开始意识到原因了。我们开始看到游戏规则一点都没变。我

⊖ 本书中文版书名为《大规模 C++程序设计》。——译者注

们开始意识到 Web 只是另一种交付机制，与其他的并没有什么区别——它只是 IBM 老式绿屏请求/响应技术的一种变体。它就是原来简单的老式软件，我们永远不应该放弃这些游戏规则。

现在，我们意识到应该一直沿用 Parnas、Weinberg、Page-Jones 以及 DeMarco 这些人的智慧结晶。我们永远不能远离 Jacobson 和 Booch 的教导。我们早就应该读 Lakos 那本书！

不过，有人确实读过这本书。他肯定还读过一些其他的书，因为他专门写了一本书来描述 Java 架构的游戏规则，这本书比我以前见过的书都要好。现在你正把这本书拿在手中。写这本书的人名叫 Kirk Knoernschild。

在这本书中，Kirk 超越了 Lakos、Jacobson 以及 Booch。他借鉴了这些大师的原则并创建了关于原则、规则、指导以及模式的全新组合，这些组合是很伟大的。朋友们，这是关于如何构建 Java 应用的。

继续翻阅一下。意识到什么了吗？是的，没有废话！都是核心内容。所有章节都是切题的，都是务实、有用且必要的！都是 Java 应用架构的具体要素——模块化的、解耦的、分层的、可独立部署的以及敏捷的。

如果你是 Java 程序员，如果你是技术领导或团队领导，如果你是架构师，如果你想要或必须让你的开发团队有所改变，那阅读这本书吧。如果你想避免重复过去 15 年的悲剧，那阅读这本书吧。如果你想学习软件架构到底是什么，那阅读这本书吧。

闲言少叙。

——Uncle Bob

序 2

大约在两年前（2010 年 1 月），我收到了 Kirk Knoernschild 的电子邮件，他邀请我为他即将完成的书提供反馈。回头看看随后发生的激烈讨论——大约有 50 封或更多的邮件——我不得不怀疑他对我会产生一些怨恨。我确信我们之间的交流导致他最初的进度出现了严重的延迟。所以，当 Kirk 邀请我为本书作序的时候，我感到又惊又喜；他为这本书付出了巨大的努力却让一个对手来给他写序，这说明他拥有足够强大的内心。

现在，我同意 Kirk 在本书中所说的大部分内容。我们都为模块化的魔力着迷，并且在大多数基础理念上都很有共识。但是，就像通常我们所见到的那样，最热烈的争论往往发生在那些在原则上互相认可但在细节上存在分歧的人群之间。直到在德国达姆施塔特举行的 OSGi 社区大会（OSGi Community Event）上（也就是这篇序言截止日期的两天前），我才突然理解了 Kirk 受到的阻力。

在这次大会上，Graham Charters（IBM）提出了“模块成熟度模型”（Modularity Maturity Model），这个模型来源于 IBM 的 SOA 成熟度模型（SOA Maturity Model），而后者又显然来源于最初的 SEI 能力成熟度模型（Capability Maturity Model，CMM）。这是一场很有见地的演讲，它使我认识到在系统设计方面，因为我在模块化的环境中生活了超过 13 年，我的观点并没有深受其影响。

CMM 的一个重要教训就是不可能直接越过某一步。如果你的公司在 CMM 的第 1 级（混乱的），那么制订计划大跨步直接迈到第 4 级（一般称其为“已管理级”）不会是什么好主意。有的公司这样尝试了，但都惨烈失败了。每个中间步骤的转换都是必要的，它会帮助组织了解不同级别的复杂性。每个级别都有一系列自己的问题，这些问题需要在下一级别来解决。

在 Graham 演讲之后，我清楚地认识到我基本上都是从第 5 级往下看，而 Kirk 则是帮助人们从第 1 级往上看。我们之间争论的具体问题是设计模块化软件时所面临的挑战。当你到达第 2 级或第 3 级时，这些挑战是显而易见的，但当你处于第 1 级时，它们就不那么明显了。我们的大脑有这样一种定式，即只有我们处理过对应的问题，才能理

解其解决方案。我试图让 Kirk 讨论的那些解决方案是他的读者还没有经历和理解的，这些问题位于前面的等级中。

在我的模块成熟度模型中（Graham 的会稍有不同）有如下等级：

1）未管理的/混乱的

2）管理依赖

3）适当隔离

4）修改代码库（code base）以最小化耦合

5）面向服务的架构

在第 1 级中，应用是基于类路径（class path）的，也就是线性的 JAR 列表。应用包含一系列的 JAR 或具有类文件的目录，这些形成了类路径。在这个级别，根本不存在什么模块化。这个级别的问题是缺少类或版本混乱。

在第 2 级中，你已经识别了模块并声明了对其他模块的依赖。模块会有一个名字并且可以进行版本标识。它们依旧是线性查找的，第 1 级的很多问题依然存在，但是系统的可维护性以及结果的可重复性会更好一些。这个层级的主要问题在于，因为过度的传递性依赖，会过多地“从互联网上下载”。这是 Maven 目前所处的级别。

在第 3 级中，模块间会通过导入、导出和私有包真正实现模块间的隔离。依赖现在可以在包级别进行描述，这可以减少“从互联网上下载”。这种隔离为模块提供了内部的命名空间，而这个命名空间是真正在模块内部的，允许存在多个命名空间，这样在一个应用中就可以支持同一个包的不同版本。这个级别的问题通常是由流行 Java 模式引起的，因为它们基于动态类加载机制，很少能与模块边界和多命名空间兼容。

在第 4 级中，你的代码库的修改只是为了最大化内聚或最小化耦合。人们越来越认识到某一行代码可能会导致过多数量的依赖。组合或分隔系统的功能可能会对系统的部署产生重要的影响。在这个级别，已有的 Java 模式使用起来会很痛苦，因为它们通常会依赖中心化的配置，而这个方案更像是平等的点对点（peer-to-peer）模型。在 OSGi 中，μService 变得很有吸引力，因为它们解决了很多问题。

在第 5 级，也就是最后一个级别中，模块化不再像它们所提供的 μService 那样重要。这时设计和依赖方案完全是通过 μService 实现的；模块只是使用和提供 μService 的容器。

在过去的 13 年中，我一直生活在第 5 级中，因为它是在 OSGi 中实现的。这有时会

让我很难设身处地体会那些只使用类路径和简单 JAR 文件的人们。看到 Graham Charter 的演讲后，我认识到 Kirk 的目的是帮助人们理解模块化设计原则的重要性并使他们从第 1 级到达第 2 级，最终为他们打下坚实的基础以便使用 OSGi 实现更好的模块化。我认识到我经常试图把这本书直接提升到第 5 级，因为前面所提的一些教训对设计模块化软件也是很重要的。那样一本书依然是很必要的，希望有一天我能自己来写作。

现在 Kirk 的书很重要，因为它提供了开始模块化思考的模式并且能够让你借助最流行的平台、框架和语言构建模块化的软件。是的，我相信本书中的很多问题会有更好的解决方案，但是我也认为有时候追求更好会适得其反。

假设你使用 Spring、Guice 或者其他流行的依赖注入框架来构建 Java 应用，但依然面临脆弱和僵硬的软件，维护这些软件会很困难并且代价高昂，那么这本书对你来说真的有用。代码的完全耦合会造成很难添加新的功能或者修改已有的代码库。这本书将教会你很多模块化的基础课程并向你展示模块化的魔力。

也就是说，我还希望你特别关注贯穿本书使用 OSGi 的例子。它首先出现在第 3 章结尾处，它向你展示了 OSGi 如何借助 μService 来帮助你实现适当的隔离并尽可能最小化耦合的。随着本书所提供的价值不断增加，我相信按照它的建议将会帮助你构建架构上更加完整的软件并引导你正确地将应用迁移到 OSGi 上。OSGi 是迄今为止最成熟的模块化方案。

一直以来，Kirk 不仅仅是值得尊敬的对手；他敦促我把自己的想法写出来，通过这种方式让我学到了很多，比近几年任何人教我的都要多。在过去的两年中，在和他讨论这本书的过程中，我享受到了很大的乐趣，我真的希望你在读这本书的时候也会有同样的感觉。

——Peter Kriens

OSGi 联盟技术总监

前　言

在 1995 年的时候，设计模式曾经风靡一时。今天，我却发现情况完全相反。模式变得司空见惯，大多数的开发人员在日常工作中会不假思索地使用模式。现在，很少会出现像“四人组”（Gang of Four，GoF）模式[㊀] 那样有影响力的新模式。实际上，这个产业从模式运动以来已经有了很大的改进。模式变得不再那么时尚，它们成了开发人员工具箱中的一部分并用来帮助设计软件系统。

但是，设计模式在过去十多年所扮演的角色不应被低估。它作为催化剂，推动面向对象开发成为主流。它们帮助大批开发人员理解了继承的真正价值以及如何有效使用它。模式提供了如何构建灵活且有弹性软件系统的深刻见解。借助金子般的智慧，如“优先使用组合而不是类继承”以及“面向接口编程而不是面向实现”（Gamma 1995），模式帮助一代软件开发人员接受了一种新的编程范式。

今天，模式还在广泛使用，但对很多开发人员来说，它们是很自然的事情。开发人员不再争论使用策略模式（Strategy pattern）的好处，他们也不必再参考 GoF 的书来识别哪个模式最适合当前的需要。相反，好的开发人员能够本能地设计面向对象的软件系统。

很多模式是永恒的。它们与特定的平台、编程语言或者编程时代无关。做一些细微的修改并且对细节稍加关注，一个模式就能变成适合给定上下文的形式。很多事情是与上下文相关的，包括平台、语言以及要解决问题的复杂程度。随着更多地学习模式，我们提供了在特定语言下如何使用模式的样例，并将其称为习语（idiom）。

我认为本书中的模块化模式也是永恒的。它们与特定的平台或语言无关。不管你是使用 Java 还是.NET、OSGi[㊁] 还是 Jigsaw[㊂]，或者是想构建更加模块化的软件，本书中的模式都会帮助到你。随着时间的推移，我们将会看到一些习语出现，它们将描述如何将这些模式应用到支持模块化的平台上，并且会有工具帮助我们使用这些模式重构软件系

㊀《Design Patterns》（中文版为《设计模式：可复用面向对象软件的基础》）一书中的模式被亲切地称为 GoF 模式。GoF 代表“四人组”，也就是这本书的四位作者。

㊁ OSGi 是 Java 平台的动态模块化系统。它是 OSGi 联盟管理的一个规范。想了解更多信息，可以访问 www.osgi.org。

㊂ Jigsaw 是计划为 Java SE 8 添加的模块化系统。（该功能已经推迟到了 Java SE 9。——译者注）

统。我希望当有工具出现的时候，它们能够不断地进化来帮助开发模块化的软件。但最重要的是，我希望在你们的帮助下，这些模式能够进化并且变成有助于设计更好软件的模式语言——这些软件将会实现模块化的优点。时间将会告诉我们一切。

面向对象的设计

在过去的几年中，出现了一些面向对象的设计原则。很多设计原则体现在设计模式中。Bob 大叔提出的 SOLID 设计原则（参见附录）是最突出的例子。深入分析 GOF 模式，会发现它们中的很多都符合这些原则。

这些形成共识的知识以及所带来的收益有助于指导面向对象开发，但是创建大型的软件系统依然是很困难的。这些大型的系统依旧难以维护、扩展和管理。现有的原则和面向对象开发模式不能帮助管理大型软件系统的复杂性，这是因为它们所解决的是不同的问题。它们有助于解决逻辑设计相关的问题但是无助于解决物理设计方面的挑战。

逻辑设计与物理设计

有些原则和模式可以帮助解决软件设计和架构所面临的问题，这些问题几乎都是关于逻辑设计的。[⊖] 逻辑设计是关于语言结构的，如类、操作符、方法以及包。识别类的方法、类之间的关系以及系统中包的结构都是逻辑设计问题。

毫无意外，因为大多数的原则和模式强调的都是逻辑设计，所以开发人员将大多数时间都用来处理与逻辑设计相关的问题。在设计类及其方法的时候，你是在定义系统的逻辑设计。决定一个类是否为单例（singleton）是逻辑设计问题。确定一个操作是否为抽象的或者决定要继承自一个类还是要包含它，这些同样也是逻辑设计问题。开发人员生存在代码中，就会不断地处理逻辑设计的问题。

能够很好地使用面向对象设计原则和模式是很重要的。要适应大多数业务应用中所需要的复杂行为是很有挑战性的任务，如果不能创建灵活的类结构，将会对未来的增长

⊖ 例外的情况是 John Lakos 的好书《大规模 C++程序设计》。在这本书中，Lakos 为我们展现了多个逻辑设计和物理设计原则，它们有助于开发 C++所编写的软件。

和可扩展性带来负面的影响。但是逻辑设计并不是本书关注的焦点。有很多其他的图书和文章提供了必要的指导，它们能提供足够的智慧以保证创建良好的逻辑设计。逻辑设计只是软件设计和架构所面临挑战的一个方面。挑战的另一方面就是物理设计。如果你没有考虑系统的物理设计，那么不管你的逻辑设计多么漂亮，可能都不会带来预期的收益。换句话说，缺乏物理设计的逻辑设计并不会带来预期的影响。

物理设计表现为软件系统中的物理实体。确定如何将软件系统打包到部署单元中是物理设计问题。决定哪个类属于哪一个部署单元也是物理设计问题。同样，管理可部署实体之间的关系也是物理设计问题。如果我们不说物理设计比逻辑设计更重要的话，起码它们是同等重要的。

例如，定义接口可以使客户端与实现该接口的类解除耦合，这是一个逻辑设计问题。按照这种方式解耦显然能够让你在不影响客户端的情况下，创建这个接口的新实现。但是，要将接口和它的实现类放到物理实体中就是物理设计问题了。如果这个接口有多个不同的实现，并且每个实现类都有底层的依赖，那么如何放置接口和实现就会对系统的整体软件架构质量产生巨大的影响。将接口和实现放在同一个模块中将会引入不必要的部署依赖。如果其中的一个实现依赖复杂的底层结构，那么不管你选择使用哪一个实现，都需要在所有的部署环境中包含这个依赖结构。尽管逻辑设计的质量很高，但是物理实体之间的依赖将会阻碍可重用性、可维护性以及很多在设计时试图要获得的其他收益。

令人遗憾的是，尽管众多团队付出了很大一部分时间在逻辑设计上，但是很少有团队在物理设计上下工夫。物理设计是关于如何将软件系统拆分为模块系统的，物理设计是关于软件模块化的。

模块化

在开发和维护方面，大型的软件系统天生就比小型的系统更复杂。模块化会涉及将大型的系统拆分为单独的物理实体，最终会使系统更易于理解。通过理解模块中的行为以及模块间存在的依赖，我们可以更容易地识别和评估变化所带来的连带影响。

例如，要修改具有很少输入依赖的软件模块比那些具有众多输入依赖的模块更容易。同样，具有很少输出依赖的软件模块比那些具有众多输出依赖的模块更易于重用。当设计软件模块时，重用和可维护性是重要的考虑因素，而依赖在其中扮演了重要的角

色。但是，依赖并不是唯一的因素。

在设计高质量的软件模块方面，模块内聚也扮演着重要的角色。具有很少行为的模块对于其他的模块并没有做太多有用的事情，因此所提供的价值很小。与此相反，如果模块所做的事情太多将会难以重用，因为它所提供的行为超过了其他模块的期望。当设计模块时，识别合适的粒度等级是很重要的。太细粒度的模块提供的价值很小，可能还需要其他的模块协作才能发挥作用。太粗粒度的模块则会难以重用。

本书中的原则为设计模块化的软件提供了指导。它们探讨了一些方式，这些方式可以让你尽可能地减少模块之间的依赖同时又能最大化模块的重用能力。如果没有面向对象设计的原则和模式，其中的很多原则是无法实现的。你将会发现，对模块化系统所做的物理设计决策经常会影响逻辑设计决策。

模块化单元：JAR 文件

在 Java 平台中，物理设计是通过仔细设计 JAR 文件的关系和行为实现的，模块化的单元就是 JAR 文件。尽管这些原则可以应用到其他的单元，如包中，但是将其应用在设计 JAR 文件中会特别有价值（关于模块的定义，参见第 1 章）。

OSGi

OSGi 服务平台（OSGi Service Platform）是 Java 中的动态模块化系统。在 OSGi 的术语中，模块称为 bundle。OSGi 提供了一个框架来管理 bundle，bundle 被打包成普通的 Java JAR 文件，里面包含了清单文件（manifest）。在清单文件中包含了重要的元数据信息，这些信息描述了 bundle 以及对 OSGi 框架的依赖（关于 OSGi，参见第 13 章）。

贯穿本书，你会发现使用 OSGi 的例子。但是，要使用模块化模式，OSGi 并不是先决条件。OSGi 只是提供一个运行时环境，它使得在 Java 平台中实现模块化成为可能，同时它也会强制要求这种模块化。OSGi 提供了如下功能。

- **模块化：**使得在 Java 平台中实现模块化成为可能，同时也会强制要求模块化。
- **版本管理：**支持相同软件模块的多个版本部署在同一个 Java 虚拟机（JVM）实例中。

- **热部署**：允许在运行时系统中进行部署和更新，不必重新启动应用或 JVM。
- **封装**：允许模块对它们的使用者隐藏实现细节。
- **面向服务**：鼓励在更细的粒度上，在同一个 JVM 中使用面向服务的设计原则。为了做到这一点，OSGi 使用的是 μService。
- **依赖管理**：需要明确声明模块之间的依赖。

本书所面向的读者

本书面向负责开发软件应用的开发人员和架构师。如果你对提升系统的设计感兴趣，那么本书也适合你。

本书并不只针对那些使用原生模块化平台的人们。例如，如果你正在使用 OSGi，那么本书会帮助你使用 OSGi 设计更为模块化的软件。但是如果没有使用 OSGi，本书所讨论的技术依然是很有价值的，它会帮助你使用这些技术来提升软件系统的模块化水平。本书也不是仅仅面向 Java 开发人员的。尽管本书中的例子使用的都是 Java，但是所讨论的技术却可以很容易地用于其他平台，如.NET。

如果你想更深入地理解模块化的好处并且要开始设计模块化的软件系统，本书就是为你而作！本书为以下问题提供了答案。

- 模块化所能带来的收益是什么以及它为何如此重要？
- 如何让其他开发人员相信模块化的重要性？
- 要增加软件系统的模块化程度，可以采用什么技术？
- 在没有 OSGi 这样的原生模块化平台上进行开发，如何实现模块化？
- 如何将大规模的整体应用迁移为具备模块化架构的应用？

本书是如何组织的

本书分为三部分。第一部分描述模块化的理由。在这里，你将会看到模块化在设计软件时所扮演的重要角色，与此同时还会学习需要设计模块化软件的原因。第二部分是 18 个模式的列表，这些模式会帮助你设计更为模块化的软件。第三部分引入 OSGi，然后

阐述如何使用本书中的模式来设计软件系统并充分发挥模块化平台的优势。第三部分使用代码样例论证我们的观点。

显然，我建议你逐页阅读本书。但是，在阅读的时候你可能会愿意从一章跳到另一章。尽可以这么做！在本书中，你可能会发现许多与当前主题有关的向前或向后的交叉引用。这会帮助你进行导航并且更容易理解某个理念。以下是每章的概述。

第一部分：模块化的理由

第一部分阐述了模块化为何如此重要。这是采用模块化的理由。第一部分各章的概述如下。

- **第 1 章**：该章引入模块化，正式地定义和识别软件模块的特征。建议每个读者都阅读这个简短的章节。
- **第 2 章**：模块化有两个方面，即运行时模型和开发模型。一直以来，很多的关注集中在运行时模块化支持方面。随着越来越多的平台提供运行时模块化的支持，开发模型的重要性将会得到更多的关注。开发模型由编程模型和设计范式组成。
- **第 3 章**：模块化在软件架构方面扮演着重要的角色。它填补了一项空白，这项空白从团队开发企业级软件系统以来就一直存在。该章将会探讨软件架构的目标以及模块化在实现这一目标时所扮演的重要角色。
- **第 4 章**：企业级软件系统充满复杂性。团队会面临技术债的挑战，因为设计腐化，系统会面临崩溃。该章展现模块化如何帮助我们征服软件系统不断增长的复杂性。
- **第 5 章**：重用是软件开发的灵丹妙药。但是，很少有组织能够真正实现高度的重用。该章考察阻止实现重用的障碍并探讨模块化如何帮助我们提高成功的可能性。
- **第 6 章**：模块化与 SOA 在很多方面都是互补的。该章探讨模块化与 SOA 如何成为强大的组合。
- **第 7 章**：为要讨论的理念提供一些合适的样例是很重要的。这一章有两个目的。首先，它将前 6 章的材料组织在一起，这样就能够看到这些理念是如何得到运用的。其次，它为第二部分要讨论的很多模式奠定了基础。