

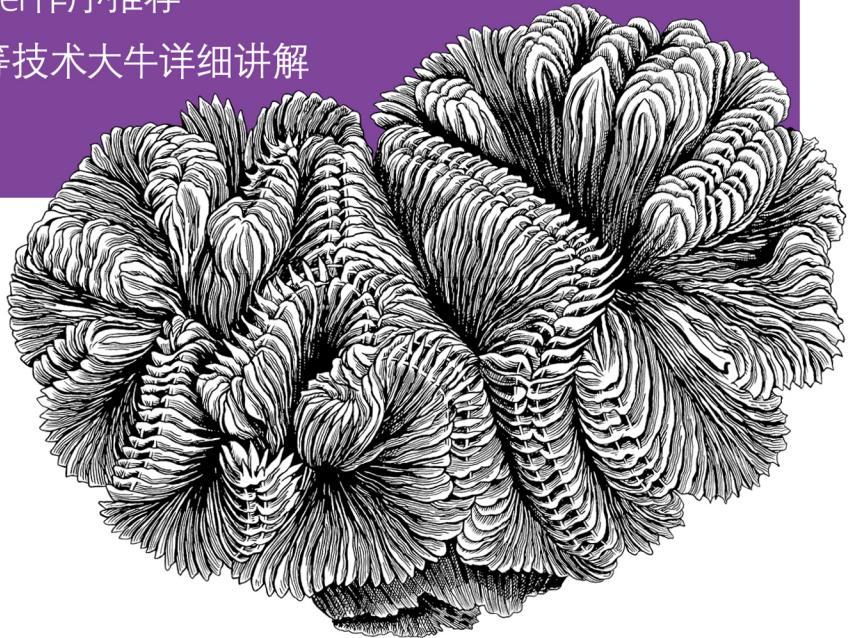
演进式架构

Building Evolutionary Architectures

敏捷之父Martin Fowler作序推荐

ThoughtWorks CTO等技术大牛详细讲解

先进架构思想



[美] 尼尔·福特 丽贝卡·帕森斯 著

[澳] 帕特里克·柯

周训杰 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

译者介绍

周训杰

ThoughtWorks高级咨询师，拥有十余年企业级应用和互联网应用的开发和架构经验，在ThoughtWorks带领多个团队完成不同规模的海外项目交付，目前负责大型企业级项目的服务化平台搭建及技术团队管理。

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。



图灵程序设计丛书

演进式架构

Building Evolutionary Architectures

[美] 尼尔·福特 [美] 丽贝卡·帕森斯 [澳] 帕特里克·柯 著
周训杰 译

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北京

图书在版编目（CIP）数据

演进式架构 / (美) 尼尔·福特 (Neal Ford) ,
(美) 丽贝卡·帕森斯 (Rebecca Parsons) , (澳) 帕特
里克·柯 (Patrick Kua) 著 ; 周训杰译. — 北京 : 人
民邮电出版社, 2019.8
(图灵程序设计丛书)
ISBN 978-7-115-51617-6

I. ①演… II. ①尼… ②丽… ③帕… ④周… III.
①程序设计 IV. ①TP311.1

中国版本图书馆CIP数据核字(2019)第137818号

内 容 提 要

在软件开发流程中,为了尽可能快地响应各种变化,理应把结构渐进改变作为设计的首要原则。本书详尽阐述了演进式架构的必要性、构建方法以及需要注意的问题。各章结合案例分别讨论了软件架构、适应度函数、开展增量变更、架构耦合、演进式数据、构建可演进的架构、演进式架构的陷阱和反模式,以及实践演进式架构。

本书适合所有架构师及开发人员阅读。

-
- ◆ 著 [美] 尼尔·福特 [美] 丽贝卡·帕森斯
[澳] 帕特里克·柯
 - 译 周训杰
 - 责任编辑 朱 巍
 - 责任印制 周昇亮
 - ◆ 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 印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 9.75
 - 字数: 230千字 2019年8月第1版
 - 印数: 1-3 500册 2019年8月北京第1次印刷
 - 著作权合同登记号 图字: 01-2018-8084号
-

定价: 59.00元

读者服务热线: (010)51095183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

© 2017 by Neal Ford, Rebecca Parsons, and Patrick Kua.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2019. Authorized translation of the English edition, 2019 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版，2017。

简体中文版由人民邮电出版社出版，2019。英文原版的翻译得到 O'Reilly Media, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

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、在线服务还是面授课程，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 凭借一系列非凡想法（真希望当初我也想到了）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*

目录

序	ix
前言	xi
第 1 章 软件架构	1
1.1 演进式架构	2
1.1.1 一切都在变化，如何才能长期规划	3
1.1.2 完成架构构建后，如何防止它逐渐退化	4
1.2 增量变更	5
1.3 引导性变更	6
1.4 多个架构维度	6
1.5 康威定律	8
1.6 为何演进	10
1.7 小结	11
第 2 章 适应度函数	13
2.1 什么是适应度函数	15
2.2 适应度函数分类	16
2.2.1 原子适应度函数与整体适应度函数	16
2.2.2 触发式适应度函数与持续式适应度函数	16
2.2.3 静态适应度函数与动态适应度函数	17
2.2.4 自动适应度函数与手动适应度函数	17
2.2.5 临时适应度函数	18
2.2.6 预设式高于应急式	18
2.2.7 针对特定领域的适应度函数	18

2.3 尽早确定适应度函数	18
2.4 审查适应度函数	19
第3章 实施增量变更	21
3.1 构件	24
3.1.1 可测试性	25
3.1.2 部署流水线	26
3.1.3 组合不同类型的适应度函数	30
3.1.4 案例研究：在每天部署 60 次的情况下重建架构	31
3.1.5 目标冲突	33
3.1.6 案例研究：为 PenultimateWidgets 的发票服务添加适应度函数	33
3.2 假设驱动开发和数据驱动开发	36
3.3 案例研究：移植什么	37
第4章 架构耦合	39
4.1 模块化	39
4.2 架构的量子和粒度	40
4.3 不同类型架构的演进能力	42
4.3.1 大泥团架构	42
4.3.2 单体架构	44
4.3.3 事件驱动架构	49
4.3.4 服务导向架构	53
4.3.5 “无服务”架构	62
4.4 控制架构量子大小	63
4.5 案例分析：防止组件循环依赖	64
第5章 演进式数据	67
5.1 演进式数据库设计	67
5.1.1 数据库模式演进	67
5.1.2 共享数据库集成	69
5.2 不当的数据耦合	73
5.2.1 二阶段提交事务	74
5.2.2 数据的年龄和质量	75
5.3 案例研究：PenultimateWidgets 的路由演进	76
第6章 构建可演进的架构	79
6.1 演进机制	79
6.1.1 识别受演进影响的架构维度	79
6.1.2 为每个维度定义适应度函数	80
6.1.3 使用部署流水线自动化适应度函数	80

6.2	全新的项目	80
6.3	改良现有架构	81
6.3.1	适当的耦合和内聚	81
6.3.2	工程实践	81
6.3.3	适应度函数	82
6.3.4	关于商业成品软件	82
6.4	架构迁移	83
6.4.1	迁移步骤	84
6.4.2	演进模块间的交互	86
6.5	演进式架构构建指南	89
6.5.1	去除不必要的可变性	89
6.5.2	让决策可逆	91
6.5.3	演进优于预测	91
6.5.4	构建防腐层	92
6.5.5	案例分析：服务模板	93
6.5.6	构建可牺牲架构	94
6.5.7	应对外部变化	95
6.5.8	更新库与更新框架	97
6.5.9	持续交付优于快照	97
6.5.10	服务内部版本化	98
6.6	案例分析：PenultimateWidgets 的评分服务演进	99
	第7章 演进式架构的陷阱和反模式	103
7.1	技术架构	103
7.1.1	反模式：供应商为王	103
7.1.2	陷阱：抽象泄漏	104
7.1.3	反模式：最后 10% 的陷阱	107
7.1.4	反模式：代码复用和滥用	108
7.1.5	案例研究：PenultimateWidgets 中的复用	109
7.1.6	陷阱：简历驱动开发	110
7.2	增量变更	111
7.2.1	反模式：管理不当	111
7.2.2	案例研究：PenultimateWidgets 的“金发姑娘”管理	112
7.2.3	陷阱：发布过慢	113
7.3	业务问题	114
7.3.1	陷阱：产品定制	114
7.3.2	反模式：报表	115
7.3.3	陷阱：规划视野	116

第8章 实践演进式架构	119
8.1 组织因素	119
8.1.1 全功能团队	119
8.1.2 围绕业务能力组织团队	121
8.1.3 产品高于项目	121
8.1.4 应对外部变化	122
8.1.5 团队成员间的连接数	123
8.2 团队的耦合特征	124
8.2.1 文化	124
8.2.2 试验文化	125
8.3 首席财务官和预算	126
8.4 构建企业适应度函数	128
8.5 从何开始	129
8.5.1 容易实现的目标	129
8.5.2 最高价值优先	129
8.5.3 测试	129
8.5.4 基础设施	130
8.5.5 PenultimateWidgets 的企业架构师	131
8.6 演进式架构的未来	131
8.6.1 基于 AI 的适应度函数	132
8.6.2 生成式测试	132
8.7 为什么（不）呢	132
8.7.1 公司为何决定构建演进式架构	132
8.7.2 案例分析：PenultimateWidgets 选择性伸展	134
8.7.3 企业为何选择不构建演进式架构	135
8.7.4 说服他人	136
8.7.5 案例分析：“咨询柔道”	136
8.8 商业案例	136
8.8.1 未来已来	136
8.8.2 没有后顾之忧地快速前行	137
8.8.3 风险更低	137
8.8.4 新能力	137
8.9 构建演进式架构	137
关于作者	139
封面介绍	140

序

长久以来，软件行业都奉行这样一个理念：在开始编写第一行代码前就应该完成架构开发。受到建筑行业的影响，人们认为成功的软件架构在开发过程中不需要修改，而且重新架构往往会导致高成本的报废和返工。

随着敏捷软件开发方法的兴起，这样的架构愿景受到了很大的挑战。预先规划的架构要求在编码前就确定需求，因而催生了分阶段（即瀑布式）的开发方法——在完成需求分析后才开始设计架构，然后再进入构建（编码）阶段。然而，敏捷软件开发方法并不认同“需求是确定的”这样的观点。它发现现代商业中需求不断变化是必然的，并进一步提供了项目规划技术来拥抱受控的变化。

在这个新的敏捷世界里，很多人质疑架构的作用。当然，预先规划的架构无法适应动态的现代业务，但是，还是会有另外一种架构，它以敏捷的方式拥抱变化。如此看来，架构是不断努力的结果，是一个与开发工作紧密结合的过程，这样它才能同时响应不断变化的需求和开发人员的反馈。我们称之为“演进式架构”，正是要强调当无法预测变化时，该架构仍然可以朝着正确的方向发展。

在 ThoughtWorks，我们无时无刻不秉持演进式架构的理念。Rebecca 在 21 世纪初领导了多个重大项目，作为 CTO，她提升了 ThoughtWorks 的技术领导力。Neal 一直密切关注着我们的工作，将我们学到的经验汇总并传播开来。Pat 在开展项目工作的同时也加强了我们的技术领导力。我们始终认为架构是极其重要的，不容忽视。我们虽然也犯过错误，但我们在错误中学习，更好地理解了如何构建出能正确应对各种变化的系统。

构建演进式架构的核心是采取小步变更，然后通过反馈环让团队的每个成员不断地从系统的发展过程中学习。持续交付的兴起使得演进式架构变得切实可行。三位作者通过适应度函数监控架构的状态。他们探索了架构演进的不同方式，并且重视那些通常被别人所忽视的长存数据。在讨论中也理所当然地会经常提及康威定律。

关于构建演进式软件架构，虽然我们还需要了解很多东西，但本书基于当前的理解给出了基本线路图。随着越来越多的人意识到软件系统在 21 世纪人类社会中的核心地位，每个软件领导者都应该知道如何在发展中以最好的姿态应对变化。

Martin Fowler

前言

排版约定

本书使用如下排版约定。

- **黑体字**
表示新术语或重点强调的内容。
- 等宽字体（*constant width*）
表示程序以及段落内引用的程序元素，如变量、函数名、数据库、数据类型、环境变量、语句和关键字。
- 加粗等宽字体（***constant width bold***）
表示应该由用户输入的命令或其他文本。
- 等宽斜体（*constant width italic*）
表示应该被替换为由用户提供的值或由上下文确定的值的文本。



该图标表示提示或建议。

O'Reilly Safari

 Safari[®] Safari（原来叫 Safari Books Online）是一个会员制的培训和参考咨询平台，面向企业、政府、教育从业者和个人。

会员可以访问来自 250 多家出版商的上千种图书、培训视频、学习路径、互动式教程和精选播放列表，这些出版商包括 O'Reilly Media、Harvard Business Review、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Adobe、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 等。

要了解更多信息，可以访问 <http://oreilly.com/safari>。

联系我们

请把对本书的评价和问题发给出版社。

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

O'Reilly 的每一本书都有专属网页，你可以在那儿找到书的相关信息，包括勘误表¹、示例代码以及其他信息。本书的网站地址是：<http://oreil.ly/2eY9gT6>。

对于本书的评论和技术性问题，请发送电子邮件到 bookquestions@oreilly.com。

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

<http://www.oreilly.com>。

我们在 Facebook 的地址如下：<http://facebook.com/oreilly>。

请关注我们的 Twitter 动态：<http://twitter.com/oreillymedia>。

我们的 YouTube 视频地址如下：<http://www.youtube.com/oreillymedia>。

附加信息

可以通过 <http://evolutionaryarchitecture.com> 访问本书的配套站点。

注 1： 本书中文版的勘误请到 <http://www.ituring.com.cn/book/2440> 查看和提交。——编者注

致谢

Neal 要感谢过去几年里在各种大会倾听过他演讲的所有人，他们帮助打磨和修订了本书。他也要感谢本书的技术审阅人员，他们为本书提供了非常宝贵的建议和意见，特别是 Venkat Subramanian、Eoin Woods、Simon Brown 和 Martin Fowler。Neal 还要感谢他的猫 Winston、Parker 和 Isabella，它们的干扰总能为他带来顿悟。他还想感谢他的朋友 John Drescher、ThoughtWorks 的所有同事、Norman Zapien 敏锐的耳朵、每年的 Pasty Geeks 度假团和附近的鸡尾酒俱乐部，感谢他们的支持和友谊。最后，Neal 要感谢他坚忍的妻子，她用笑容包容了他的经常出差和为事业所做的其他牺牲。

Rebecca 要感谢所有同事、大会出席者及演讲者和作者，感谢他们多年来为演进式架构领域贡献的想法、工具、方法以及提出的澄清问题。她也要感谢本书的技术审阅人员，感谢他们细致的阅读和评注。此外，Rebecca 还要感谢本书的合著者，感谢创作过程中所有启发性的对话和讨论。她要特别感谢 Neal，几年前与他有过一场关于应急式和演进式架构区别的大讨论或者说是辩论，之后这方面的思想才逐渐地成型。

Patrick 要感谢 ThoughtWorks 的所有同事和客户，感谢他们推动了需求，并为阐明演进式架构的构建想法提供了测试平台。他还要同 Neal 和 Rebecca 一起向本书的技术审阅人员表示感谢，他们的反馈极大地提升了本书的质量。最后，他要感谢本书的合著者们。过去几年中，他们位于不同的时区，相距甚远，很少有机会能面对面地一起工作，特别感谢能有机会在这个项目上和他们密切合作。

电子书

扫描如下二维码，即可购买本书电子版。



第1章

软件架构

一直以来，由于软件架构涉及范围广且内涵不断变化，开发人员不断尝试给它一个简洁的定义。Ralph Johnson 就将其定义为“重要的东西（无论那是什么）”。架构师的工作就是理解和权衡那些“重要的东西”（无论它们是什么）。

为了给出解决方案，架构师工作的第一步是理解业务需求，也即领域需求。这些需求是使用软件来解决问题的动机，但终究只是架构师在构建架构时需要考虑的因素之一。架构师还必须考虑其他很多因素，其中一些比较明确（比如清楚地写在性能服务水平协议里），还有一些则隐含在商业活动中不言自明（比如公司正着手并购重组，软件架构显然也要有变动）。所以对于软件架构师来说，架构水平体现了他们在权衡业务需求和其他重要因素后找到最佳方案的能力。软件架构涵盖了所有这些架构因素，如图 1-1 所示。

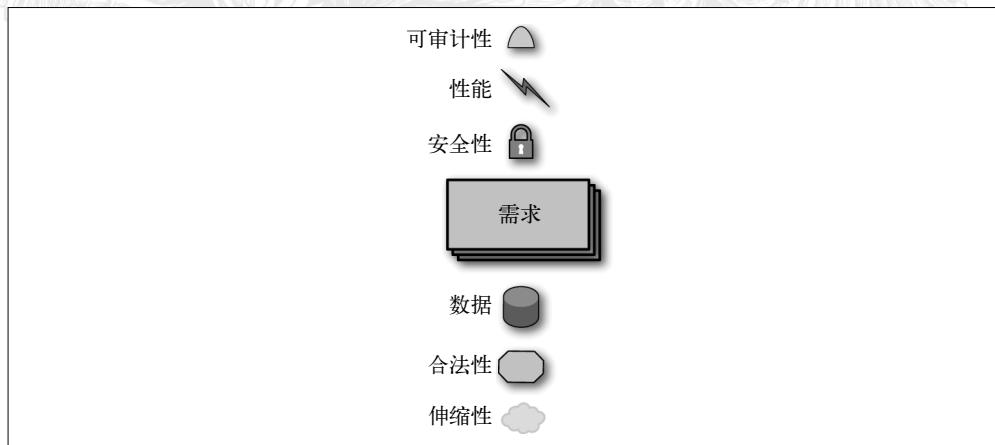


图 1-1：架构的完整概念，包括需求及“各种特征”

如图 1-1 所示，业务需求与其他架构关注点（由架构师定义）并存。这包括各种外部因素，这些因素可以改变构建软件系统的决策过程。表 1-1 列举了一些例子。

表1-1：部分特征列表

可访问性	可问责性	准确性	适应性	可管理性
可负担性	敏捷性	可审计性	自治性	可用性
兼容性	可组合性	可配置性	正确性	可信度
可定制性	可调试性	可降级性	可确定性	可演示性
可信赖性	可部署性	可发现性	分布性	耐久性
有效性	高效性	可用性	可扩展性	故障透明性
容错性	保真性	灵活性	可检查性	可安装性
完整性	互通性	可学习性	可维护性	可管理性
移动性	可修改性	模块性	可操作性	正交性
可移植性	精确性	可预测性	过程能力	可生产性
可证性	可恢复性	相关性	可靠性	可重复性
重现性	弹性	响应性	可复用性	稳健性
安全	伸缩性	无缝性	自我维持性	可服务性
安全性	简单性	稳定性	标准合规性	可生存性
可持续性	可裁剪性	可测试性	及时性	可追溯性

在构建软件时，架构师必须明确哪些特征最重要。然而，许多因素是互相矛盾的。比如，让软件具备高性能的同时还要实现极大的伸缩性就很困难，因为实现这两者需要谨慎地平衡架构、运维及其他诸多因素。因此，在为架构设计做必要分析的同时，又要处理好各个因素之间不可避免的冲突，架构师在权衡每个架构设计方案的利弊时，常常需要做出非常艰难的折中。近年来，软件开发核心工程实践的持续发展给我们提供了条件，使我们得以重新思考架构随时间的推移要如何变化，以及当这样的演进发生时，如何保护重要的架构特征。本书将这些部分联系起来，以一种新的方式思考架构和时间。

我们想为软件架构添加一个新的标准“特征”——演进能力。

1.1 演进式架构

无论我们怎么努力，软件依然变得越来越难以改变。由于各种原因，软件的组成部分不容易变更，而且随着时间推移变得愈发脆弱和难以操作。软件项目的变更通常是由于对功能或范围做了重新评估而导致的，但是还有一些变化是架构师和长期规划者无法控制的。尽管架构师喜欢为未来做战略性规划，但不断变化的软件开发环境使这一切变得困难重重。既然变化是必然的，那么我们就只能因势利导地来利用它。

1.1.1 一切都在变化，如何才能长期规划

生物世界中，环境因自然因素和人为因素而不断变化。例如，20世纪30年代初，澳大利亚的甘蔗受到甲虫危害，导致甘蔗作物严重减产，利润大减。1935年6月，作为应对措施，当时的甘蔗实验站管理总局引入了甘蔗蟾蜍来捕食甲虫，这种蟾蜍原本只产于中美洲和南美洲。短暂喂养后，1935年7月和8月在昆士兰州北部投放了甘蔗蟾蜍。因为它们的皮肤有剧毒，并且在当地没有天敌，很快这种蟾蜍就泛滥成灾了。如今，澳大利亚的甘蔗蟾蜍大约有2亿只。这件事告诉我们：向高度动态的（生态）系统中引入变化，可能会产生无法预料的结果。

软件开发体系由所有的工具、框架、库以及最佳实践（软件开发领域的技术积累）构成。和生态系统一样，软件开发体系实现了平衡，开发人员能够理解这个体系并为其添砖加瓦。然而，这种平衡是动态的，随着新事物不断出现，平衡不断被打破和重建。想象一个脚踏独轮车，手里还拿着盒子的人。他是动态的，因为他需要不断调整来保持挺立；他又是平衡的，因为他保持着身体平衡。在软件开发体系中，每一项创新或新实践都可能打破现状，迫使系统重新建立平衡。就好比我们不断地将更多的盒子抛向骑独轮车的人，迫使他不断寻求新的平衡。

架构师在很多方面都和这个倒霉的独轮车手相似，不断地平衡以适应环境变化。持续交付这项工程实践使得这个平衡过程有了结构性的转变，它将过去孤立的功能（例如运维）合并到了软件开发的生命周期中，这让我们对变化的含义有了新的认识。企业级架构师不能再依赖静态的五年计划了，因为整个软件开发体系在不断变化，任何一个长期计划都可能变得毫无意义。

即便对经验丰富的实践者来说，颠覆性的创新也是难以预测的。比如，像 Docker 这样的容器化技术的崛起就是一个不可预知的行业转变。但我们仍然可以通过一系列小的演进找到一些蛛丝马迹。以前，操作系统、应用服务器和其他基础设施都是商品，需要斥巨资购买使用许可。当时许多架构设计着重于高效利用共享资源。渐渐地，Linux 变得足以支撑企业及应用，使得购买操作系统的费用降为零。接下来，通过 Puppet 和 Chef 等工具自动配置服务器的 DevOps 实践使得 Linux 运维工作也不再需要成本。一旦开发环境免费并得到广泛应用，势必朝着更加通用和便携的方向发展，于是 Docker 应运而生。但如果之前所有的演进过程，容器化就不会发生。

我们所使用的编程平台也在持续演进。新的编程语言提供了更好的应用编程接口（API），提高了对新问题的灵活性和适用性。新的编程语言还提供了不同的范式和概念。例如，引入 Java 替代 C++，降低了编写网络代码的难度并改善了内存管理。回顾过去的 20 年，很多语言一直在持续改进它们的 API，与此同时，新的编程语言往往用于解决新的问题。编程语言的演变如图 1-2 所示。

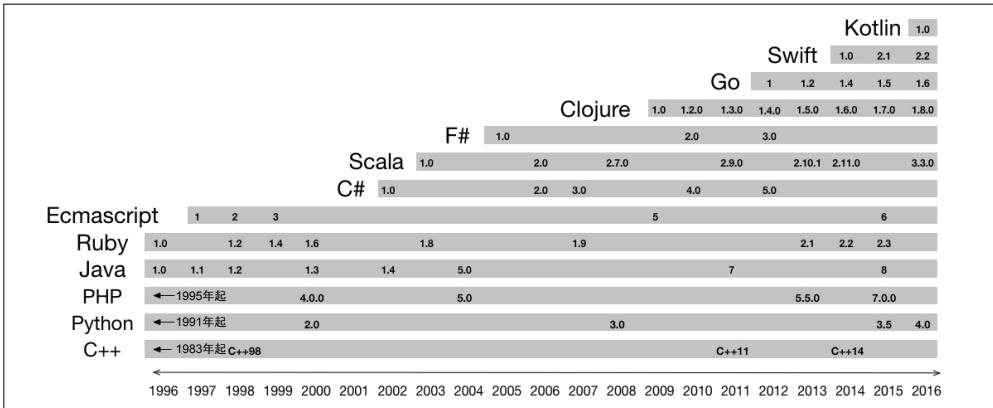


图 1-2：流行编程语言的演进过程

无论是在软件开发的哪个方面，比如编程平台、编程语言、运维环境、持久化技术等，我们都知道改变会持续发生。虽然无法预测技术或领域格局何时会改变，或哪些变化会持续下去，但我们清楚改变是不可避免的。因此，我们应该在构建系统的过程中对这一点保持清醒的认识。

如果整个体系持续地以出乎意料的方式发生变化，预测变化就变得不可能了，那么用什么来替代固定计划呢？企业架构师和开发人员必须学会适应变化。做长期计划有一个隐含的原因是财务上的考虑，因为以前软件变更的成本很高。但是，现代工程实践通过自动化和其他先进实践（例如 DevOps）降低了软件变更的成本。

多年来，一些聪明的开发人员发现系统的某些部分相对而言更难修改。这便是将软件架构定义为“将来难以变更的部分”的原因，这个省事的定义简单地区分了软件系统中真的难以修改的部分和可以轻松修改的部分。但这个定义依然不免走进了盲区，因为开发者预先假设“变更是困难的”，这变成了一个自证式的预言。

几年前，一些富有创新精神的架构师用新的视角审视了“将来难以变更”的问题。使架构具有可变性会怎样呢？换句话说，如果易于改变是架构的基本原则，那么变更将不再困难。反过来，使架构具备演进能力会导致一组全新的行为出现，进而再次打破整个体系的平衡。

即使环境不改变，架构特征出现磨损该怎么办？架构师设计出架构，将其置于纷乱的现实世界，基于架构执行各项事务。架构师要如何保护他们定义的重要部分呢？

1.1.2 完成架构构建后，如何防止它逐渐退化

有一种不幸的退化叫作架构比特衰减，它在很多组织中均有发生。架构师选择特定的架构模式来满足业务需求及让系统具备某些能力，但这些特征常常意外地随着时间推移而退

化。例如，架构师构建了一个包含顶部展现层、底部持久层和一些中间层的分层架构。负责报表功能的开发人员出于性能上的考虑经常会要求绕过中间层，直接从展现层访问持久层。架构师通过分层来隔离变化。而后开发人员绕过这些层，不仅增加了耦合，还使得分层变得毫无价值。

定义了那些重要的架构特征后，架构师如何保护这些特征不磨损呢？答案是添加演进能力作为新的架构特征，使其在系统演进时保护其他特征。比如，架构师追求架构设计的高伸缩性，但不希望在系统演进时削弱该特征。因此，**演进能力**是一种元特征和保护其他所有架构特征的架构封装器。

本书将阐明演进式架构的另一个作用，即它是一种为架构的重要特征提供保护的机制。我们探寻**持续架构**背后的理念。**持续架构**指构建架构的过程没有最终状态，它会随着软件开发体系的不断变化而演进，并保护重要的架构特征。我们不会尝试定义整个软件架构，因为已经存在很多定义了。我们通过引入时间和变化作为头等架构元素来扩展当前的定义。

我们对演进式架构的定义如下。

演进式架构支持跨多个维度的引导性增量变更。

1.2 增量变更

增量变更描述了软件架构的两个方面：如何增量地构建软件和如何部署软件。

在开发阶段，允许小的增量变更的架构更易于演进，因为对于开发者来说，变更范围相对更小。对部署而言，增量变更指业务功能的模块化和解耦水平，以及它们是如何映射到架构中去的。示例如下。

假设有一个小工具商家 PenultimateWidgets，该商家采用微服务架构和一些现代工程实践运营着一个产品目录页面。该页面的一个功能是让用户给出小工具的星级评分。PenultimateWidgets 的其他一些业务也需要评分，比如客服代表、物流服务商评价等，所以这些业务共享星级评分服务。某天，该服务的开发团队发布了新版本，新版本允许用户给出半星评价（一个小而重要的升级）。其他需要评分的服务不需要强制升级，而可以逐渐地在合适的时候迁移到新版本。PenultimateWidgets 的 DevOps 实践采用了架构级别的监控，不仅监控各个服务，还监控服务与服务之间的路由。当运营人员观察到在给定时间内没有请求路由至特定服务时，他们自动将该服务从体系中移除。

这是一个在架构级别进行增量变更的例子：如有需要，原服务和新服务可以并存。团队可以在适当（不太忙或必要）的时候完成迁移，当所有迁移都完成时，旧版本的服务将会作为垃圾被自动回收。

增量变更的成功需要一些持续交付实践的配合。并不是任何情况都需要所有这些实践，但通常它们会一起发生。第3章将讨论实现增量变更的方法。

1.3 引导性变更

一旦架构师选择了重要的架构特征，他们会把变更引导进入架构，以保护这些重要特征。为此，我们借用演化计算中的一个概念：**适应度函数**。该函数是一种目标函数，用于计算潜在的解决方案与既定目标的差距。在演化计算中，适应度函数决定一个算法是否在持续提升。换句话说，随着每个算法变体的产生，基于设计者对算法“适应度”的定义，适应度函数决定每个变体的“适应程度”。

对于演进式架构，随着架构的演进，我们有着类似的需求。我们需要评估机制，来评估变化对架构重要特征的影响，并防止这些特征随着时间的推移而退化。适应度函数的隐喻涵盖多种机制，包括度量、测试和其他检验工具。我们采用这些机制来确保架构不会以不良方式变更。当架构师确定了需要保护的架构特征时，他们会定义一个或多个适应度函数来提供保护。

以往，架构往往要划出一部分作为管理活动，最近架构师才接受了通过架构实现变更的思想。架构适应度函数允许在组织需求和业务功能的上下文中制定决策，并为明晰且可测试的决策奠定了基础。演进式架构并不是毫无约束或不负责任的软件开发方式。相反，它可以在高速变迁的业务、严谨的系统需求和架构特征间找到平衡。适应度函数驱动架构设计决策，并引导架构变更适应业务和技术环境的变化。

第2章将详细介绍使用适应度函数指导架构演进。

1.4 多个架构维度

不存在单独的系统。世界是一个整体。如何划分系统边界取决于讨论的主题。

——Donella H. Meadows

古希腊的物理学通过固定点分析宇宙，最终发展成了经典力学。但是到了20世纪初，随着仪器越来越精密，现象越来越复杂，人们逐渐从经典力学转向相对论。科学家意识到之前被视为孤立的现象其实是相互影响的。自20世纪90年代起，受到启发的架构师越来越多地将软件架构视作多维的。持续交付也将运维纳入了软件架构的范畴。软件架构师往往关注技术架构，但那只是软件项目的维度之一。如果架构师想构建可演进的架构，就必须考虑系统中所有会受变化影响的部分。正如物理学所讲的，万物都是相关联的，架构师深知软件项目是多维的。

为了构建可以不断演进的软件系统，架构师不能只考虑技术架构。例如，如果项目包含一

个关系型数据库，那么数据库实体之间的结构和关系也会随着时间的推移而不断变化。另外，架构师也不希望系统在演进过程中暴露安全漏洞。这些例子展示了架构的不同维度，它们通常在架构中以正交方式交织在一起。部分维度在常见的架构关注点范围之内（见表 1-1），但是架构维度的意义其实更广泛，它囊括了很多传统意义上技术架构范畴之外的东西。每个项目都有许多维度，架构师在考虑架构演进时必须要想到。下面是一些影响现代软件架构演进能力的常见维度。

□ 技术

架构中的实现部分：框架、依赖的库和实现语言。

□ 数据

数据库模式、表格布局、优化计划等。通常由数据库管理员（DBA）处理这类架构。

□ 安全

定义安全策略、指导方针和指定工具来帮助发现缺陷。

□ 运维与系统

关注架构如何映射到现有的物理或虚拟的技术设施中，包括服务器、机器集群、交换机、云等。

以上每个视角构成一个架构维度——为了支持特定视角而有意进行的划分。这里架构维度的概念涵盖了传统的架构特征和其他有助于构建软件系统的因素。随着问题和周遭环境的改变，我们想保护架构不磨损，每个维度都提供给我们一个审视架构的视角。

从概念上划分架构的方法有很多，比如 IEEE 的软件架构定义中的 4+1 视图模型。它关注不同角色的不同视角，将整个系统划分成了逻辑视图、开发视图、进程视图和物理视图。在著名的《软件系统架构》一书中，作者给出了软件架构的视点目录，它涵盖了更多角色。类似地，Simon Brown 的 C4 建模符号也从概念上帮助组织区分关注点。不过，本书没有尝试创建任何维度分类，而是识别那些现有项目中的维度。从实用角度来看，不论如何对关注点进行分类，架构师都需要保证这些维度不磨损。不同的项目有不同的关注点，这导致每个项目都有特定的维度。对于新项目，以上任何技术都能提供有用的见解，但是对于现有的项目，我们必须处理眼前的实际情况。

按照架构的维度思考，通过评估重要维度对变化的响应，架构师可以分析不同架构的演进能力。随着系统与互相冲突的问题（伸缩性、安全性、分布式、事务性等）关联得越来越紧密，架构师必须跟踪更多的维度。只有结合所有这些重要维度，思考系统将如何演进，才能构建出可以不断演进的系统。

项目的整个架构范围由软件需求和其他维度构成。当架构和整个体系随着时间的推移一起演进时，我们可以使用适应度函数来保护架构特征，如图 1-3 所示。

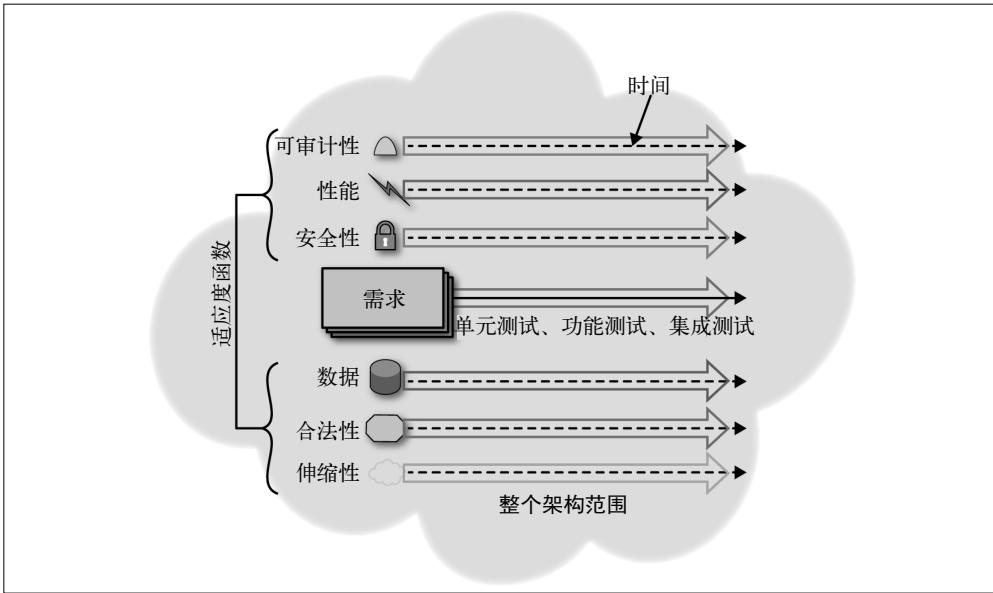


图 1-3：架构由需求和其他维度构成，每个维度都受适应度函数保护

在图 1-3 中，架构师确定了可审计性、数据、安全性、性能、合法性和伸缩性是该应用的关键架构特征。随着业务需求不断变化，每个架构特征都通过适应度函数来保护其完整性。

我们强调架构整体的重要性，但也应意识到，技术架构模式及相关议题也是架构演进的很大一部分，比如耦合和内聚。第 4 章将讨论技术架构耦合对演进能力的影响，第 5 章将讨论数据耦合的影响。

耦合不只和软件项目的结构元素有关。近来，很多软件公司认识到了团队结构对架构的影响。我们将讨论软件中的各类耦合因素，但是团队结构的影响出现得如此早且频繁，我们需要马上就此展开讨论。

1.5 康威定律

1968 年 4 月，梅尔文·康威在《哈佛商业评论》上发表了一篇名为“*How Do Committees Invent?*”的论文。在这篇论文中，康威提出：社会结构，特别是人与人之间的沟通途径，将不可避免地影响最终的产品设计。

康威描述道，在设计的最初阶段，人们首先需要高瞻远瞩地思考如何将职责划分为不同的模式。团队分解问题的方式会左右他们之后的选择，这便是康威定律。

在设计系统时，组织所交付的方案结构将不可避免地与其沟通结构一致。

——梅尔文·康威

正如康威所描述的，当技术人员将问题分解成更小的块，使其更易于委派时，就会产生协调问题。很多组织为了解决协调问题，会设置正式的沟通结构或是建立森严的等级制度，但这样的解决方案往往是僵化的。比如，按照技术职能（用户界面、业务逻辑等）划分的团队，他们在处理常见的跨职能问题时协调的开销就会增加。那些从创业团队跳槽到跨国公司工作的人，很可能会经历两种截然不同的文化：前者非常灵活且适应性很强，而后的沟通结构往往是僵化的。康威定律一个很好的例子就是修改两个服务间的契约。如果一方想修改服务，而这需要另外一方的同意和配合，那么这件事就可能变得很难。

在论文中康威特别提醒软件架构师，不要只关注软件架构和设计，还应关注团队之间委派、分配和协调工作的方式。

在很多组织中，团队是根据职能来划分的。示例如下。

□ 前端开发团队

负责用户界面（UI）技术（如 HTML、移动端和桌面端）。

□ 后端开发团队

专门构建后端服务（有时还包括 API 层）。

□ 数据库开发团队

专门构建存储和逻辑服务。

在这样的组织中，管理层从人力资源的角度简单地按照职能划分团队，没有充分考虑工作效率。虽然每个团队都有其擅长的领域（比如构建一个视图，增加一个后端 API 或服务，或者开发一个新的存储机制），但是当需要发布新的业务功能或特性时，三个团队都要参与其中。各个团队通常都会针对眼前的任务优化效率，而不是针对那些更抽象的战略业务目标（特别是有工期压力时）。这会导致各团队往往专注于交付各自的组件，而不关注端到端的特性价值，导致这些组件可能无法高效协作。

在这样的团队编制下，由于每个团队都在不同的时间忙于自己的组件，因此那些依赖所有团队的特性需要花费更长的时间。例如，修改目录页这样常见的业务变更涉及 UI、业务规则和数据库模式的变更。如果每个团队都各自为战，那么他们必须协调时间表，这将增加实现该特性所需的时间。这个例子很好地解释了团队结构对架构和演进能力的影响。

康威在论文里提到：“每当新的团队组建，其他团队的职责范围会缩小，能够有效执行的可选设计方案也会随之变少。”换句话说，人们很难改变其职责范围外的事情。软件架构师需要时刻关注团队的分工模式，从而使架构目标和团队结构保持一致。

很多构建架构（如微服务）的公司围绕服务边界构建团队，而不是按孤立的技术架构来划分。在 ThoughtWorks 技术雷达中，我们称之为“康威逆定律”。以这种方式组织团队是理

想的，因为团队结构会影响软件开发的很多维度，并且会反映问题的大小和范围。比如，在构建微服务架构时，企业通常会构建与架构相仿的团队结构，通过打破功能筒仓使每个团队都有人能考虑到业务的各个角度和技术的方方面面。



构建与目标系统架构相仿的团队结构，这样项目会更容易实现。

PenultimateWidgets 及其逆康威时刻

本书以 PenultimateWidgets 公司为例。它是排行倒数第二的小工具经销商，还是一家大型在线小工具（各种小商品）商家。这家公司正在更新其大部分 IT 基础设施，包括一些想暂时保留一段时间的遗留系统和正在迭代开发中的新战略系统。本书将重点介绍 PenultimateWidgets 所遇到的问题以及采取的解决方案。

他们的架构师首先观察了软件开发团队。旧的单体应用采用了分层架构，将展现、业务逻辑、持久化和运维分离开来。而他们的团队设置也和架构如出一辙：所有前端开发人员在一起工作，后端开发人员和数据库管理员也各自工作，运维则外包给了第三方。

当开发人员开始在架构（基于具有细粒度服务的微服务架构）上工作时，协调成本急剧增加。因为服务是围绕领域（例如用户结算）而不是技术架构来构建的，所以在变更单个领域时将需要各团队间进行大量的协调。

于是，PenultimateWidgets 采取了康威逆定律，建立了和服务范围相匹配的跨职能团队：每个服务团队包括服务负责人、几位开发人员、一位业务分析师、一位 DBA、一位质量保障人员和一位运维人员。

本书多处提到了团队的影响，并会通过示例说明它带来的结果。

1.6 为何演进

对于演进式架构，一个常见的问题是关于其名称的：为什么叫作演进式架构而不是别的？比如增量式、持续式、敏捷式、响应式或应急式，等等。这些术语都不够准确达意。这里所说的演进式架构包含了两个关键特征：增量和引导。

持续式、敏捷式和应急式都表达了随时间不断变化的概念，这确实是演进式架构的关键特征，但是这些术语都没能准确地表达出架构将如何变化，或者说期望的架构最终是什么样子的。虽然这些术语都隐含着环境变化，但是都没能涵盖架构应有的样子。而在演进式架构的定义中，引导的含义反映了我们想实现的架构，即我们的最终目标。

相比可适应这个词，我们倾向于演进式，因为我们对经历根本性演变的架构感兴趣，而不是那些经历了修修补补后变得越来越难以理解、复杂问题频发的架构。适应意味着解决方案不求优雅或长久，只要能找到方法使系统运作就行。为了构建能实实在在演进的架构，架构师必须支持真正的变化，而不是权宜之计的考虑。回到我们的生物学隐喻，演进是这样一个过程：建立一个适用的并能在其所处的不断变化的环境中持续运行的系统。系统可能存在个别的适应性，但是架构师应该关心整体可演进的系统。

1.7 小结

演进式架构主要由三方面构成：增量变化、适应度函数和适当的耦合。本书余下的部分将分别讨论它们，然后再将它们结合起来，帮助我们构建和维护支持不断变化的架构。



微信连接



回复“架构”查看相关书单



微博连接

关注@图灵教育 每日分享IT好书



QQ连接

图灵读者官方群I: 218139230

图灵读者官方群II: 164939616

图灵社区
iTuring.cn

在线出版,电子书,《码农》杂志,图灵访谈

演进式架构

企业架构师不能再依赖静态计划了。软件开发体系在持续变化，新的工具、框架、技术和范式不断涌现。这给脆弱的系统带来了挑战，但也提供了更好的解决方案。近年来，核心软件工程实践中的快速变化让我们重新思考如何更改架构，使其与时俱进。本书结合相关实践，给出了让架构适应变化的新思路。

构建演进式架构主要涉及3个方面：适用度函数、增量变更和适当的耦合。ThoughtWorks的3位专家各讲一个方面，然后综述如何构建支持持续变更的架构。

- 适应度函数：架构呈现或前进的目标
- 增量变更：在开发和运维中实现渐进改变
- 架构耦合：确定适当的架构耦合以支持无瑕变更
- 演进式数据：随时间推移按要求和架构转变演进数据库
- 构建可演进的架构：结合以上各方面构建演进式架构
- 实践演进式架构：助你起步的实践指南

尼尔·福特 (Neal Ford) 是ThoughtWorks软件架构师、Meme Wrangler，持续集成与交付领域专家。

丽贝卡·帕森斯 (Rebecca Parsons) 是ThoughtWorks CTO，在大规模分布式对象应用开发和集成方面拥有丰富经验。

帕特里克·柯 (Patrick Kua) 是ThoughtWorks前首席科学家，在敏捷和精益开发方面拥有丰富经验。

封面设计：**Karen Montgomery** 张健

图灵社区：iTuring.cn

热线：(010)51095183转600

分类建议 计算机/软件开发/微服务

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

“这本书着眼于理论、立足于实践，架构师们定会受益匪浅。几十年前我若能读到它该多好，当然现在也不晚。”

——Venkat Subramaniam博士

Agile Developer公司创始人

ISBN 978-7-115-51617-6



ISBN 978-7-115-51617-6

定价：59.00元

欢迎加入

图灵社区

最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。

ituring.com.cn