The Pragmatic Programmers

TURING 图灵程序设计丛书

Dr.Dobb's Jolt 第18届Jolt生产效率大奖图书

发布！软件的设计与部署

通过实例探讨如何构建稳健、务实的软件架构

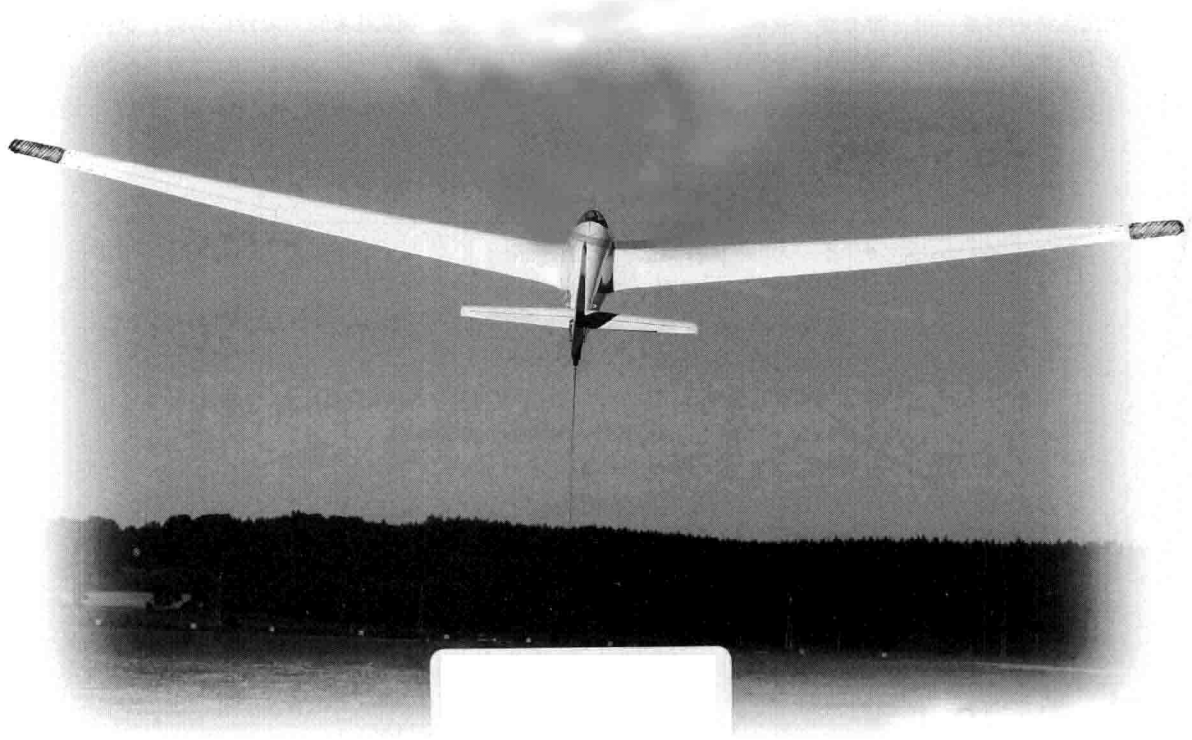
【美】Michael T. Nygard 著　涂鸣 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

发布！软件的设计与部署

【美】Michael T. Nygard 著　涂鸣 译



人民邮电出版社
北　京

图书在版编目（CIP）数据

发布！软件的设计与部署 /（美）尼加德（Nygard,M.T.）著；涂鸣译. -- 北京：人民邮电出版社，2015.2
（图灵程序设计丛书）
ISBN 978-7-115-38045-6

Ⅰ. ①发… Ⅱ. ①尼… ②涂… Ⅲ. ①软件设计 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2014)第300320号

内 容 提 要

作者根据自己的亲身经历和某些大型企业的案例，讲述了如何创建高稳定性的软件系统，分析设计和实现中导致系统出现问题的原因。本书分为四个部分，每部分内容都由一个研究案例引出。第一部分介绍了如何保证系统的生存，即维持系统正常运行。第二部分介绍了如何衡量系统的容量，以及如何随时间来优化系统的容量。第三部分讲述了架构师在为数据中心构建软件时应该思考的一般设计问题。第四部分讨论了系统的运行寿命，将其作为整个生态环境的一部分。

本书适合面向企业级软件的架构师、设计师和开发人员阅读参考。

◆ 著 [美] Michael T.Nygard
译 涂 鸣
责任编辑 朱 巍
责任印制 杨林杰
◆ 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
北京鑫正大印刷有限公司印刷
◆ 开本：800×1000 1/16
印张：14.75
字数：349千字 2015年2月第1版
印数：1-3 000册 2015年2月北京第1次印刷
著作权合同登记号 图字：01-2011-1377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：京崇工商广字第0021号

版权声明

前　言

你为了自己的项目勤奋工作了一年多。最终，所有的功能似乎都已经完成，其中的大部分还进行了单元测试。你如释重负，终于大功告成了。

真是这样吗？

“功能完成”是否真的就意味着“产品就绪”了？系统确实可以部署了吗？操作员可以运行它，并独立面对大量实际用户了吗？你是否会有不祥预感，觉得半夜就会响起应急电话？其实，功能完成之外，还有很多开发工作要做。

通常，项目团队的目标是通过QA测试，而不是保障产品（注意这里说的是产品）的生命周期。也就是说，你的大部分工作是为了通过测试。测试，即便是敏捷、注重实效的自动化测试，也并不足以证明软件已经可以进行实际应用了。狂热的用户、全球涌来的超大流量，还有你闻所未闻的病毒制造者，这些残酷的现实都大大超出了测试时的预期。

为了确保软件能够应对自如，你的确需要做好充分的准备。在这本书里，我会为你展示问题所在，并帮助你了解如何解决它们。开始讲解之前，我会先讨论一些普遍存在的误解。

首先，你得接受这样的事实：尽管你的计划很棒，但是糟糕的状况仍然会出现。当然，如果可能，能够防范这些问题总是好的。不过最要命的情况就是，你以为自己总能预料并排除所有可能发生的错误事件。正确的做法应该是，对于自己确实能够预料到的问题采取预防措施，同时要确保整个系统能从那些无法预料的严重灾难中恢复过来。

其次，你要了解，发布1.0版并不是项目开发的终点，反而是系统自身生命周期的起点。这就好比刚长大成人的孩子第一次离开父母。你肯定不希望看到已经成年的孩子又搬回来和你一起住，特别是他们还带着另一半，加上四个孩子、两条狗和一只澳洲鹦鹉。

同样，你在项目开发过程中所做的决策将会极大影响你在1.0版发布之后的生活质量。如果你未能针对产品的环境来设计系统，那么你的生活会在项目发布后充满“刺激”，当然，那绝不会是惊喜。在本书中，你将会看到一些至关重要的设计折中，并且了解如何更明智地权衡利弊。

最后，尽管我们都热爱技术、看上去很炫的新技巧以及那些超酷的系统，但是你必须明白，这些都不重要。在现实的商业社会中（正是这个社会来为我们的系统买单），一切都归结到金钱。我

们打造的系统要花钱。为此，无论是开源还是节流，我们必须确保这些系统能够生钱。其他工作也要花钱，即便停工时也一样。所以，低效的代码会增加运营开销，从而浪费大量金钱。要理解一个运行系统，你必须了解钱的走向。同时，要想在行业中站有一席之地，你得赚钱，至少不能赔钱。

我真的希望这本书能使这些情况有所改变，切实地帮助你和你的公司避免传统企业软件所面临的问题：巨大损耗和过度支出。

谁来读这本书

本书面向企业级软件的架构师、设计师和开发人员，这些软件包括网站、网络服务、EAI系统等。在我看来，企业级意味着软件必须可用，否则公司就会赔钱。这些软件可以是通过销售直接获得收入的电子商务系统，也可以是雇员用于工作的重要内部系统。如果有人因为你设计的软件不能工作，不得不回家闲呆着，那么你需要好好读读这本书。

本书结构

本书分为四个部分，每部分内容都由一个研究案例引出。第一部分介绍如何保证系统的生存，即维护系统正常运行。尽管分布式系统通过冗余来保证可靠性，但是它更可能是以“两个8”而不是让人梦寐以求的“5个9”来展示其有效性[①]。相对任何其他考量，稳定性是必要的先决条件。试想，如果你的系统每天都会宕机，那么还有谁会关心这个系统的未来呢。在这种情形下，短期修复以及短期的思考将是主要手段。没有稳定性，未来就面临生存问题，因此你一开始就得找到方法，确保自己拥有一个可运转的稳定的基础系统。

一旦做到了稳定性，下一步要思考的就是容量。在第二部分，你会看到如何衡量系统的容量，了解容量的真正含义，并学习如何随时间来优化系统的容量。我会告诉你各种模式与反模式，借此诠释好的设计和坏的设计，以及它们如何对系统的容量产生显著的效果，当然也会影响半夜响起的应急电话的次数。

在第三部分，你会了解架构师在为数据中心构建软件时应该思考的一般设计问题。在过去的十年里，硬件和基础设施的设计已经发生了巨大的变化。例如，过去罕见的多宿主实现已经非常普遍了。网络变得日益复杂，它们分层设计并且更加智能。存储区域网络司空见惯。要让软件能够在数据中心里更加流畅地运行，软件设计必须考虑并利用这些变化。

在第四部分，你需要考察系统的运行寿命，将其作为整个信息生态环境的一部分。很多产品系统就像薛定谔的猫[②]，被锁在一只盒子中而无法观察其真实的状态。这不利于构筑一个健康的

① 即88%的正常运行时间，而非99.999%的正常运行时间。

② 薛定谔的猫是奥地利物理学家埃尔温·薛定谔为证明量子力学在宏观条件下的不完备性而提出的一个思想实验。——编者注

生态系统。缺乏信息，就不可能做出深思熟虑的改进[①]。第17章讨论了需要从开发中的系统学到的动机、技术和过程（这是你唯一可能获得某些经验的地方）。一旦系统的健康状况、性能和特性被暴露，你就可以针对这些信息采取行动。事实上，别无选择，你必须根据新的认识采取行动。不过，说来容易做来难，在第18章，你会看到变化带来的障碍，以及减少并克服那些障碍的方法。

关于案例

我加入了一些研究案例，用以阐述本书的主题。这些案例取自我所了解的真实事件及真正的系统故障。这些故障的代价非常高，让涉及其间的人感到非常尴尬。因此，为了保护这些公司和人，我故意模糊了一些信息。我修改了系统、类和方法的名称，但只改变了一些不重要的细节。在每个案例中，我都保留了行业、事件发生顺序、故障模式、错误传播和结果。这些故障的损失也没有夸大。这些案例都来自真实的公司，都涉及一大笔钱。我保留了这些数字，以强调这份材料的重要性。系统一旦发生故障，很可能就会损失一大笔钱。

致谢

这本书产生于我为对象技术用户组[②]所做的一次演讲。因此，感谢Kyle Larson和Clyde Cutting，他们推荐并认可了我的演讲。Poppendieck夫妇（Tom和Mary）是两本“精益软件开发”[③]图书的作者，他们给了我极大的鼓励，说服我把这个演讲内容整理成书。我还要特别感谢我的朋友和同事Dion Stewart，他不断地为这本书的初稿提供非常宝贵的反馈。

当然，还要感谢我的妻子和女儿们。我的小女儿出生后的一半时光都在看着我为此工作。她们如此耐心，陪伴我度过了无数个写书的周末。Marie、Anne、Elizabeth、Laura和Sarah，谢谢你们！

① 胡乱猜测可能碰巧取得改进，但更有可能增加而不是减少不确定性。

② 参见http://www.otug.org。

③ 参见*Lean Software Development*[PP03]和*Implementing Lean Software Development*[MP06]。

TURING 图灵教育

延展阅读

本书简明实用、见解深刻，总结了高效程序员在开发过程中的45个个人习惯、思想观念和方法，有助于开发人员在开发进程、编码工作、开发者态度、项目和团队管理以及持续学习这5个方面积极修炼。通过学习这些内容，养成这些好的习惯，你可以极大地提升自己的编程实力，更快速、更可靠地交付更高质量的软件，从而成为真正的高效程序员。

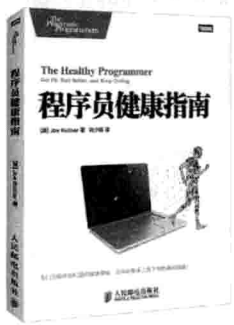
书名：高效程序员的45个习惯：敏捷开发修炼之道（修订版）
书号：978-7-115-37036-5
定价：45.00元

本书从认知科学、神经学、学习理论和行为理论角度，深入探讨了如何才能具备优秀的学习能力和思考能力，阐述了成为一名专家级程序员的关键要素，具体包括：大脑运行机制简介，如何正确使用和调试大脑，改进学习能力的具体技巧，如何通过自我引导积累经验，控制注意力的方法。为了让读者加深印象，作者还特别设立了一个“实践单元”，其中包括具体的练习和实验，旨在让读者真正掌握所学内容。

书名：程序员思维修炼（修订版）
书号：978-7-115-37493-6
定价：49.00元

作者多年来帮助许多高科技公司成功地解决了各种有关产品开发管理的棘手问题，本书正是她宝贵实战经验的提炼。书中从应对实际风险的角度出发，讲述了从项目启动、项目规划到项目结束的整个管理流程；展示了作者的思考过程，从评估项目背景，选择生命周期，直到为项目建立清晰的条件；同时穿插了丰富的提示和真实案例，介绍了可能遇到的常见问题。这些真知灼见不仅适用于软件项目管理，同样适用于其他产品的开发项目。

书名：项目管理修炼之道（修订版）
书号：978-7-115-36978-9
定价：59.00元



众所周知，程序员工作压力大、无暇照顾自己，去健身房更是奢侈行为。颈椎病、手腕疼痛、干眼症等问题困扰着几乎所有程序员和长期伏案的工作人员。在久坐不动的室内工作中，在没有大块时间去健身的客观条件下，如何保持身体健康？本书在科学研究和实践的基础上，总结了一些简单易行的健康指导，帮助读者从周身病痛、病弱且极有可能脾气暴躁的黑客，蜕变为快乐、高效的程序员。

书名：程序员健康指南
书号：978-7-115-36716-7
定价：39.00元

目　录

第1章 引 言

学校或社会开设的软件设计课程极其不完整，它只谈了系统该做什么，而没有涉及系统不该做什么。系统不应该崩溃、宕机、丢失数据、侵犯隐私、损失金钱、损害公司，或者导致客户流失。

在本书中，我们会探讨为这个混乱、扭曲的真实世界架构、设计和构建软件，特别是构建分布式系统的方法。为应对那些行事疯狂、难以预测的用户，我们要做好准备。软件从发布的那一刻起就会遭受攻击，它需要抵抗台风般的快闪族、Slashdotting，抑或 Fark 或者 Digg[①]上的链接。我们将仔细研究无法通过测试的软件，找到方法确保软件能经受现实世界的考验。

现今的软件设计就像 20 世纪 90 年代早期的汽车设计，完全脱离现实世界。那时候，汽车只是在凉爽舒适的实验室中设计，从模型和 CAD 系统看，它们都很棒。在几个巨大风扇前，曲线完美的汽车闪闪发光，轰隆着冲破层层气流。这些设计者们生活在这样平静的空间里，进行着优雅、复杂、精巧，但也脆弱、不能让人满意并且最终短命的设计。大多数的软件架构和设计也同样在纯净、与世隔绝的环境中进行。

你想拥有一辆为现实世界设计的汽车，你希望这位设计者懂得：机油要在汽车每行驶 3000 英里后更换；轮胎胎面磨损至 1/16 英寸时，性能必须与开始时一样好；在你一手拿着腌肉蛋汉堡，一手打着电话时，你保不齐就会猛踩刹车。

1.1 瞄准正确的目标

大多数软件是为研发实验室或者质量保障（QA）部门的测试人员设计的。设计与构建软件

① 快闪族是指一群素昧平生的人通过网络、手机短信等途径事先约定活动主题、时间和地点，然后一起做出夸张举动，这种活动通常短得令旁人来不及反应。Slashdotting 来源于 Slashdot.org 网站，该网站文章中的网址链接有可能在一瞬间被点击上千次，甚至上万次，造成被链接网站负荷不了，产生画面宕掉或带宽被完全占用的现象。Fark 和 Digg 的网址分别是 fark.com 和 digg.com。——译者注

的目标是为了通过测试，比如“必须输入客户的姓和名，中间名可选择输入”。它的目的是为了能在 QA 的人造场景，而非真实的产品世界中使用。

当系统通过了 QA 这一关，我就能自信满满地说它已经能够作为产品发布了吗？仅仅通过了 QA 测试，并不能让我了解，系统在今后的 3 ~ 10 年中是否能稳定工作。它可能是软件中的丰田凯美瑞，连续运行上千个小时；也可能是雪佛兰织女星（该车的前端曾经在公司自己的测试跑道上解体）或者福特平托，会在正常的碰撞测试时发生爆炸。几天或几个星期的 QA 测试并不能说明未来数年会发生什么。

长久以来，制造业的产品设计者一直在追求“可制造性设计”，这种设计产品的工程方法能够保证低成本、高质量地制造产品。在此之前，产品设计者与制造者自行其是，各自为战。设计部门“隔墙”丢给生产部门的方案，竟然包含了不合格的螺钉、易混淆的零件，以及需要用现成部件定制的零件。质量低劣、制造成本高昂自然是不可避免的。

这听来熟悉吗？今天，软件业也处在这样的状态。我们总是无法按期完成新的系统，因为要不断接听电话，为已经被我们甩掉的上一个未完成项目提供技术支持。我们以为“可制造性设计”就是“为生产而设计”，但其实我们不是为制造者设计软件，而是要将完成的软件交付 IT 运营部门。我们需要设计出一个个独立的软件系统，以及相互依赖的多个系统所构成的整个生态系统，以此带来低成本和高质量的 IT 运营。

1.2 使用决断力

早期决策对系统的最终形态影响最大。最初的决策最难以更改。这些关于系统边界和子系统划分的早期决策，渗入到了团队结构、经费分配、程序管理结构甚至工时表中。团队分配是架构的第一稿（参见 7.2 节知识框）。极具讽刺意味的是，这些早期决策也是在信息获取最少的情况下做出的。这就是说，你的团队在这个时候对软件的最终结构最无知，但你必须要做出某些不可更改的决策。

即使是“敏捷”项目[①]，也最好有远见地做出决策。为了选择最健壮的设计，设计者预测未来的时候似乎必须有决断力。由于不同的选择通常会有相似的实现成本，却带来完全不同的生命周期开销，因而考虑每种决策对可用性、能力和灵活性的影响就显得格外重要。我将结合较好和较差方法的具体例子，阐述各种设计选择的效果 。这些例子都来自于我接手过的真实系统，大多数都曾让我牺牲了宝贵的睡眠。

① 我声明，我是敏捷方法的强烈推崇者。它强调及早发布并持续改进，意味着软件可以尽快投入使用。因为软件投入使用是唯一得知软件如何响应真实世界刺激的途径，我提倡不管使用什么方法，都要尽早开始这个学习过程。

1.3 生活的质量

发布 1.0 版是软件生命的开始，而不是项目的结束。1.0 版发布后，你的生活质量取决于你在这个重要里程碑之前所做的选择。

不管你是按小时收费的技术支持工程师，还是雇用这些人的雇主，都需要知道你要维护的是一个经得起严格测试、坚不可摧，并能推进你的事业向前发展的汽车，而不是一个只在卖场摆放的花瓶。

1.4 挑战的范围

“软件危机”已经超过 30 年了。就金主而言，软件耗资依然太大（参见 *Why Does Software Cost So Much?* [DeM95]）。就目标捐助者而言，即便进度表是以月而非年来衡量的，软件的开发周期依然很长。很明显，过去 30 年设想的生产效率提高只是一场幻影。

这些术语来自于敏捷社区。金主（gold owner）是指为软件付费的人。目标捐助者（goal donor）是指你正在满足其需求的人。他们一般都不是同一个人。

另一方面，我们获得的某些实际的生产效率提高方法，可能被用来攻克更大的问题，而不是快速廉价地开发同样的软件。在过去的十年里，系统的规模呈指数级膨胀。

在安逸懒散的客户端/服务器系统时代，系统的用户基数是以十或百来衡量的，并发用户数顶多也就几十个。可现在，资助人要求系统要能承受“25 000 个并发用户”和“每天 400 万独立访问者”。

正常运行时间的要求也提高了。然而，著名的“5 个 9”（99.999%）运行时间曾经是大型机及其维护者给出的承诺，而现在，用户希望普通的商业网站也能够提供 24×7×365 的服务[①]。显然，从今天构建的软件系统的规模来说，我们已经向前迈了一大步，但是系统范围和规模的增大也导致新的系统崩溃方式层出不穷、环境更加恶劣，以及用户对系统缺陷的容忍度降低。

快速廉价地构建软件，对用户更友好，操作更简便，这些新的挑战要求我们持续地改进软件架构和设计方法。对小型网站来说可行的设计方法，对有上千用户的事务型分布式系统是不堪一用的，我们将会分析一些恐怖故障的案例。

1.5 随手一松就是一百万

项目的成功、股票期权或利润分红、公司的生存，甚至你的工作，所有这些都岌岌可危。为

① 这个短语总是在困扰我。作为一名工程师，我希望它要么是“24×365”，要么是“24×7×52”。

QA测试而构建的系统，通常在操作成本、宕机时间、软件维护等方面，都需要很大的持续支出，它们根本没法盈利，更别说有什么净现金收益了（这是指系统减去了构建成本之后的利润）。这些系统的可用性低，直接导致了收入减少，有时甚至因损害品牌形象而产生更大的间接损失。对于我的很多客户来说，宕机的直接损失每小时在10万美元以上。

在一年里，98%的正常运行时间和99.99%的正常运行时间，二者在收益上的差别将超过1700万美元[①]。请想象一下，仅仅通过更好的设计就能为账户添加1700万美元！

在繁忙紧张的项目开发期间，你很可能会决定以牺牲运营成本来优化开发成本。这只是在项目团队有固定预算和交付时间的情况下才有意义。然而，这对为软件付费的组织来说，是个错误的选择。系统的运营时间要远远多于开发时间，至少对那些还没有被取消或废弃的系统来说是如此。靠承担经常性运营成本来避免一次性成本，是没有意义的。事实上，通过一次性投入减少经常性运营成本的经济效益更大。如果为了避免版本发布期间的宕机，你愿意在一个自动构建和发布版本的系统上花费5000美元，那么公司将会少损失20万美元[②]。我相信，大多数CFO都会因这4000%的投资回报率而批准这笔经费的。

系统设计与架构上的决策也是财务决策，制定这些决策都必须着眼于实现成本和宕机成本。这种技术和财务相融合的观点，是本书反复探究的一个最重要的主题。

> 不要为了避免一次性开发成本，而以经常性运营成本为代价。

1.6 务实的架构

两种不同的活动都归结到了架构这一术语。一种类型的架构追求高度抽象，以做到更灵活的跨平台移植，并更少地关联硬件、网络、电子和光子等复杂细节。这种方法的极端形式会造就一座“象牙塔”——一间Kubrickesque式的整洁房间，盒子和箭矢装饰着每面墙，超然不俗的大师居住在这里。命令从象牙塔发出，降临在劳苦的程序员身上。“使用EJB的CMP完成数据库操作！”“所有UI使用JSF构建！”“所有过去、现在和将来的数据都要保存在Oracle数据库中！”如果你曾经为了遵守这些“公司标准”而在编码时咬紧牙关（用其他的技术完成会容易十倍），那你就是象牙塔架构师的受害者。我敢保证，一个不愿倾听团队成员意见的架构师，也一定不愿倾听用户的声音。你已经看到结果了：当系统崩溃时，用户们欢欣鼓舞，因为他们可以暂时不受这个烂系统的折磨了。

① 对于一级零售商，平均宕机成本在每小时10万美元。

② 这是假设每次发布需要1万美元（劳动力加上计划宕机的成本），每年有4次发布，计划5年。大多数公司都希望每年有超过4次发布，但是我比较保守。

另一类架构则是架构师与程序员们同心协力，甚至成为其中一员。这些架构师会毫不犹豫地剥去抽象或者摒弃那些不适合的部分。务实的架构师更喜欢讨论诸如内存使用、CPU需求、带宽要求，以及超线程和CPU绑定的优缺点等问题。

象牙塔架构师非常喜欢一种完美的终极愿景，而务实的架构师会不断地思考变化的原因。“我们该如何不重启系统而完成部署？”“我们需要收集什么数据，以及该如何分析它们？”“系统的哪部分最需要改进？”象牙塔架构一旦完成，系统就不容许任何改进，每部分都将完美地适应其角色。相反，务实架构师构建的每个组件都可以轻松地应对当前的问题，他们知道根据系统压力随时间变化的方式，哪些模块将被替换掉。

如果你已经是一个务实的架构师，书里为你准备了各种强力“弹药”。如果你是一个象牙塔架构师，并且还在读这本书，那么本书可能会“怂恿”你降低一些抽象层次，来重新思考软件、硬件和用户间的重要交叉点：活在产品中。这样，在产品最终发布的时候，你、用户以及公司都会高兴很多！

Part 1

第一部分

稳定性

本部分内容