

第6章 商业模式与加速发展

许多人第一次听到以销售开源软件为企业时，他们的第一个问题是“你如何售卖‘free’^①（免费、自由）的软件？”本章将给出这个问题的答案。

即使把讨论限制在“free=免费”的语境，也不能将答案简化多少。在商业世界中，免费和付费服务或产品以各种方式相互支持和补贴的例子比比皆是。虽然这种方法很常见，但是很难做得恰当。而且，随着技术和消费者习惯的变化，基于这种方法的商业模式可能也会迅速转变。

在一般商业关系中，金钱被用来换取有价值的产品或服务。但如果我们将开源软件视为一种存在于一般商业关系之外的“开源商业模式”，可能有些简单化了，甚至很可能是有误导性的。

这就引出了我们将要讨论的另一个重要主题：开源软件和帮助塑造它的力量并不是独立游离存在的，而是软件开发、业务交互和组织文化中更广泛趋势的组成部分。

^① free一词在英国有“免费”和“自由”的双重意义，在本章提及的某些一词多义的场景中，我们会直接引用free一词并对其意义进行说明。——译者注

6.1 如何出售自己捐赠的东西

free 是个多义词。在某些语言中，它的含义不那么混乱。拉丁语和一些由此衍生出来的语言中使用不同的词对应“自由”（拉丁语为 *liber*）和“免费”（拉丁语为 *gratis*）。而现代英语中的 free 似乎来源于古英语或盎格鲁-撒克逊语系分支，在那里 *freagon* 同时体现了自由和免费两种含义。（当然，英语也有类似于诺曼法语的拉丁语词根，例如 *liber* 和 *gratis* 这两个拉丁语词根也都出现在 *liberty* 和 *gratuity* 等单词中。）

正如我们所看到的，我们在 free software 这里所讨论的 free 一直是“自由”的意思，而不是“免费”的意思。正如 Stallman 在 GNU 宣言的第一个脚注中写道：

这里的措辞很草率。起初的目的是任何人都不需要为 GNU 系统的使用许可付费，但是文字并没有说明这一点，人们经常把它们理解为“GNU 的副本应该总是以极低价或免费的方式分发”，而这从来都不是我们的本意；后来，宣言提到公司为盈利提供分销服务的可能性。随后，我学会了仔细区分自由意义上的“免费”和价格意义上的“免费”。自由软件是用户可以自由发布和更改的软件。有些用户可能免费获得副本，而有些用户则付费获得副本——如果这些资金有助于改进软件，善莫大焉。重要的是，每个拥有副本的人都有与他人合作使用副本的自由。

但自由并不能支付账单。

GNU 宣言提供了一些主要适用于个人的可能性。例如，“有新想法的人可以将程序作为免费软件分发，从满意的用户那里寻求捐赠，或者提供相关服务”。Stallman 还提出了一些更难实现的建议，例如“假设每个购买计算机的人都必须支付一定的软件税，政府再把这些钱交给国家科学基金会这样的机构用于软件开发”。

这些看起来都不像是可扩展的商业模式。我们有充分的理由怀疑甚至批评这些所谓的“商业模式”，因为它们似乎更接近于赞助或依赖他人的善意，而不是用有价值的东西换取收入。

Linux 基金会的 Chris Aniszczyk 认为：

……它甚至可以实现我所谓的以开源开发者为中心的零工经济。在这种经济中，人们期待用户捐款，但靠捐款却赚不到足够的钱。实际上，我做了很多的研究，很少有开发者是靠用户捐款维持生计的。我认为这是一个糟糕的模式。相反，我们应该教开发者如何找到工作，或者如何围绕他们所做的“很酷的事情”来创建企业。这样他们就可以通过成立一家很棒的公司，或通过工资、福利等所有的好处来维持自己的生活。

那么，如何围绕开源建立业务呢？

6.2 是否存在“开源商业模式”

在 *Free: The Future of a Radical Price*(Hyperion, 2009)一书中，*Wired* 杂志前主编 Chris Anderson 将 free 称为“最容易被误解的词”，同时描述了许多通过免费赠予依然可以获利的方式。总之，他描述了 50 种商业模式，并将其分为三大类。

6.2.1 商业模式的类别

第一种商业模式是直接补贴。例如，Apple 公司赠送许多类型的 Apple Store Genius Bar 技术支持作为购买 iPhone 或 Mac 计算机时获得的部分套餐。

第二种商业模式是通过三方或双方市场，其中一个客户阶层补贴另一个。包括像 Facebook 这样的以广告为支持的媒体公司，都属于这一类。它们既向消费者提供服务，同时也向想要接触这些消费者的企业收取费用（从而允许消费者通过“出售”他们的注意力来换取免费服务）。

第三种商业模式是免费增值模式。某类服务或产品是免费的，但你需要付费升级。免费增值是销售多种类型软件的常见方法。iPhone 和 Android 智能手机上的应用内购就是这种方式的典型例子。你可以免费下载基本应用，但你需要付费来去除广告或获得更多功能。

这些模式都存在不同种类的问题。

首先，只有在没人能拆解你的打包产品时，补贴模式才会起作用。例如，当 Craigslist 出现并淘汰了传统报纸的分类广告业务时，报纸再也不能用分类广告来补贴外国新闻机构或调查性新闻团队了，尽管这些机构或团队给报纸带来了良好声望和大量读者。但由于成本高昂，它们作为独立企业是不可行的。

一般来说，广告，尤其是那些侵入性强的、有针对性的数字广告，会让用户的许多体验变得极差，而且越来越难以与党派性新闻和非新闻进行区分。

就本章的讨论目的而言，我们将聚焦于那些不使用广告的以及那些将开源项目作为必要组成部分的方式。

6.2.2 寻找平衡点

我们从一个为开源项目提供了额外功能和价值的商业产品开始聊。

我们中的许多人都熟悉软件的“免费增值”（freemium）模式。除

除了前面提到的应用商店模式以外，我们经常看到软件和软件服务以越来越高的价格提供多个定价阶梯，并且在一个或多个维度上的功能越来越强大。

例如，更高级的在线协作产品可能支持更多用户并提供更多存储空间。其他产品可能专注于大型企业经常需要的功能，如 Microsoft Active Directory 集成等，但个人用户通常不关心这些功能。

“免费增值”面临的一个普遍挑战是如何在免费和付费之间取得平衡。如果免费版本做得太好，付费转换率可能不足以盈利，即使是那些已经在使用该产品并愿意付费的人也不一定会支持。这些人的基本想法是：我使用各种各样的专有程序和软件服务，它们对我来说非常有用，如果我必须使用，我可能愿意为它们付费；然而，如果免费版本已满足了我的需求，我甚至可能根本不会觉得付费版本的增强功能有价值。

另外，如果过度削弱免费版本，软件又会变得无趣。如果发生这种情况，很少有人会去尝试这些软件。

这里举一个真实存在的在线存储服务的例子。某公司试图通过“提供免费试用”来扩大客户群。“免费试用”是有利有弊的，在这种情况下可能尤为不合适。想象一下，当试用期结束时，一些客户会丢失之前上传的内容，而且这可能是他们的唯一副本，这些客户会发疯的。这不会是一个好计划。

另一种选择是，该公司允许用户自由注册，获得一些他们可以永久使用的免费存储空间。但是，如果用户想要更多的存储空间，他们就必须根据需要的数量按月支付一定的费用。但是你应该给他们多大的存储空间呢？

你可以小气到给他们 10MB。真的吗？你可以用这个存储量存储一首

mp3 格式的歌曲。这样的免费版本没人会用，也不值得考虑。好的，那么 10TB 呢？这超出了几乎所有人所需的存储空间，只有极少数人需要这么大的存储空间。你会获得大量的注册用户（假设这项服务在其他方面也是有用的），但也只有极少数用户会需要更多空间并为之付费。

找到正确的平衡点是很困难的，特别是考虑到你的用户群的需求和价格敏感度可能完全不同。

6.2.3 用免费软件构建销售漏斗

“免费增值”模式的好处在于，这是一种用产品本身获得用户的好方法。虽然他们还不是付费客户，但用营销术语来说，他们已经进入了你的销售漏斗。准确地说，他们可能正处于“评估”阶段，这通常被列为“知道”和“产生兴趣”之后的第三阶段。“知道” → “产生兴趣” → “评估” → “决策” → “购买”。

他们在用你的产品就是一个好的开始。让潜在客户“评估”你的产品是极为重要的一步。软件的“免费增值”模式，尤其是相对简单易用的软件，能将从“知道”到“评估”阶段的阻力降到最低（当然前提条件是新用户的体验很靠谱）。

6.2.4 这对开源意味着什么

许多包含开源软件的商业模式都可以被认为是“免费增值”模式的变体。社区项目是免费的部分，而产品则是免费之外的部分。

但是，要小心这种过于简单的框架。

如果“免费之外的部分”只是基本的支持，那么这并没有增加很多

价值，而且总的来说也不是一种非常成功的方法。更好的做法是：将项目与使其成为商业产品的所有其他功能结合起来。至少，如果你提供支持，客户通常会希望你提供专业知识和能力，这是一般支持组织无法提供的。在开源环境下，这意味着，即使你不是这个项目的最初创建者，你也是上游社区的积极参与者。

另外，为那些想要使用项目的公司提供咨询是另一种常见的途径。但是，就咨询模式而言，真正的问题是：如果碰巧涉及一个开源项目或者你可能需要某些特定专业知识的项目，这究竟是否是一个好的咨询业务。

6.2.5 核心开源与开放源代码

“核心开源”(open core)是另一种已经存在一段时间，但近年来受到一些批评的开源方法。

通过核心开源，公司会提供免费的开源产品，但随后出售额外的专有软件，以某种方式对开源产品进行补充。这通常采取类似免费的社区版和需要许可证或订阅费的企业版的形式。两个版本的典型区别是，企业版将包含一些对运行软件的大型组织来说很重要，但对个人或者临时用户来说则不那么重要的功能。核心开源这一术语是Andrew Lampitt创造的。

Oracle公司在收购Sun公司时得到的MySQL数据库就是一个这方面的典型例子。MySQL企业版：

……包括最全面的一套高级功能，以及管理工具和技术支持，以实现最高级别的MySQL可伸缩性、安全性、可靠性和正常运行时间。它降低了开发、部署和管理业务关键的MySQL应用程序的风险、成本和复杂性。

因此，即使你可以免费使用基本的 MySQL 数据库，但企业用户想要的许多功能都需要付费。

由于追加销售的功能通常没有从核心项目中清晰地分割出来，许多核心开源产品要求其贡献者签署贡献者许可协议（Contributor License Agreement, CLA）——该协议将权利分配给产品的商业所有者。（它可以转让版权，也可以不转让版权，但在任何情况下，它通常会授予所有者在专有许可下使用贡献的代码的权利。）

纯开源项目也可以使用 CLA，但在这种情况下，它们的用途有所不同。例如，Eclipse Contributor Agreement 给出了以下理由：“CLA 想对 Eclipse 中所有知识产权的来源做个记录。我们希望有一个清晰的记录，表明您已经同意 Eclipse 社区接受贡献的条款。”在这种情况下，另一些人建议使用“开发人员原产地证书”(Developer Certificate of Origin, DCO) 而不是 CLA。

许多供应商被“核心开源”的模式所吸引，因为它是一种利用开源的商业模式，但往往避开了直接采用开源项目开发模式所带来的供应链风险以及需要履行的开源协议义务。销售商出售的是专有附加组件，希望能在“免费”和“付费”中找到平衡——免费软件足以吸引用户甚至是外部贡献者，但大多数愿意并且有能力支付费用的客户都会购买高级版本。

作为 OSI 的前主席，Simon Phipps 写道：

核心开源并不是真正追求“软件自由”，而更像是玩了一个文字游戏，因为它没有为软件用户“提供”真正的软件自由……为了在生产中有效地使用软件包，企业可能会觉得核心软件包的功能并不够，即使在核心软件包能力很强的（通常）情况下也是如此。它们会发现，如果没有某些“附加功能”，核心软件包在很大程度上是不够有效的。这些“附加功能”只在

不开源的“企业版”中可用。要使用这些功能，你必须成为软件公司的付费客户。即使觉得软件定价不合理，或者时间充裕但现金匮乏，你也没有其他选择或办法自己去开发。更糟糕的是，使用该软件包会将你锁定在该供应商手中。

6.2.6 你是否从开源开发模式中获益

以上说法是来自软件用户的观点，但从供应商的角度来看呢？这是否只是一个观点？这反映了“核心开源”并不是一种在意识形态上足够纯粹的打造开源商业公司的方法。

对某些人来说，让他们恼火的是“核心开源”中“开放”一词的用法。正如我们在讨论许可证时所看到的那样，“核心开源”中所指的“开放”，与传统意义上“开源”一词所指的“开放”是不同的。使用这个术语的供应商似乎是在试图从公司和政府对购买开源产品日益增长的需求中谋利，甚至有时通过立法授权来达成相应目标。

从客户的角度来看，如果你需要企业版的功能，那么你应该采购专有产品。开源版本缺少所需要的功能，其实并不是那么重要。本质的问题在于：“核心开源”与传统的私有软件或“软件即服务”的“免费增值”方法其实没有什么不同。我们依然不能尝试那些不开放的部分。“免费增值”模式在吸引潜在用户方面能力显著，而“核心开源”与使用“免费增值”的专有软件销售方法并没有什么不同之处，只不过加入了“开源软件”这一卖点而已。

但是不考虑开源软件带来的意识形态和客户灵活性，“核心开源”模式从开源开发模式中的获益是有限的。

开源作为一种开发方法的核心关键点并不在于“代码任何人可见”

这一点，而是“个人和公司可以通过合作和工作协同来创造更好的软件”这一点。然而，由于依赖于核心开源的版权扩展，“核心开源”不禁给人一种“供应商拥有开源项目及其社区”的感觉。在这种情况下，很难吸引外部贡献者——即使在最好的情况下，这也是社区管理的一个挑战。而更常见的情况是：该开源项目只是名义上的开源而已。

总而言之，在某些情况下，“核心开源”仍然是一种明智的方法。另外，采用这种方法也可以避免在开源社区开发中可能涉及的一些权衡。

6.3 采用“开源发展模式”的企业软件

在 Red Hat 公司，我们将自己描述为一家使用“开源发展模式”的企业软件公司。笔者认为这很好地抓住了问题关键，因为它既描述了目标，也描述了实现目标所需的众多手段（之一）。那么，开源是如何走到这一步的？

6.3.1 独立软件供应商的崛起

我们很难指出第一家不卖硬件而只单纯销售软件（即第一家独立软件供应商）的公司。正确答案很可能是成立于 1968 年的 Cincom 系统公司。它售卖的商业数据库管理系统是第一个由非 IBM 公司等系统制造商开发的管理系统。补充个有趣的事，时至 2021 年，Cincom 公司仍然是一家私营公司，而且它的创始人之一 Thomas Nies 仍然担任首席执行官。

随着时间的推移，客户购买的大部分软件是由单一业务或基本上是单一业务的软件公司打包并出售的。正如我们所看到的，像 Microsoft 公司这样销售封闭源专有软件的独立软件供应商甚至成为操作系统和其他

平台软件的主要供应商，这些软件过去都是由供应商与他们的硬件捆绑提供的。

当开源软件出现时，它并不需要像专有软件一样，在使用前购买软件许可。然而，许多用户仍然希望与商业实体建立关系，因为这会带来一些具体的好处。

6.3.2 开源支持的出现

Cygnus Solutions 很可能是第一家以正式形式系统地提供开源软件支持的公司。它由 John Gilmore、Michael Tiemann 和 David Henkel-Wallace 于 1989 年建立，最初名为 Cygnus Support。它的口号是：让自由软件价格合理。

Cygnus Solutions 公司维护了许多关键的 GNU 软件产品，包括 GNU 调试器等。它也是 GCC 项目 GNU C 编译器的主要贡献者。

正如 Tiemann 在 1999 年的 *Open Sources: Voices from the Open Source Revolution* (O'Reilly Media) 一书中描述的那样：

我们在 1990 年 2 月签订了第一份合同。到 4 月底，我们已经签订了价值超过 15 万美元（1 美元约合 6.7 人民币）的合同。5 月，我们向 50 位可能对我们的支持感兴趣的潜在客户发送了信函，6 月又向另外 100 位发送了信函。突然间，这个商业模式看起来真的可行。到第一年年底，我们已经签订了价值 72.5 万美元的支持和开发合同，我们看到的每一个地方都有更多的机会。

在 Tiemann 的那本书中，他还谈到其他一些对成功的开源企业非常重要的东西。他写道：

除非竞争对手能够招到与我们现有的 100 余名工程师匹敌的人力（这些工程师大多数是我们支持的软件的主要作者或维护者），否则他们无法从“真正的 GNU”来源的地位上取代我们（我们提供了对 GCC、GDB 和相关实用程序的所有更改的 80% 以上）。

在 Tiemann 写下这些话后不久，1999 年 8 月刚刚上市的 Red Hat 公司收购了 Cygnus 公司。Red Hat 公司的历史可追溯到 1993 年，当时成立了 ACC 公司的 Bob Young 开始了一项邮购目录业务，即销售 Linux 操作系统和 UNIX 操作系统的软件副本。第二年，Marc Ewing 开始发行自己策划的 Linux 版本，并且选择了“Red Hat”作为名字。之所以选择这个不寻常的名字，是因为他在卡内基·梅隆大学计算机实验室帮助同学时，因戴着祖父的红色康奈尔大学曲棍球帽而闻名。

Young 发现自己卖了很多套 Ewing 的产品。1995 年，他们联合成立 Red Hat 软件公司，出售盒装版本的 Red Hat Linux 操作系统（一种早期的 Linux 发行版）。

6.3.3 Linux 发行版出现

Linux 发行版（distributions 或 distros）最早出现在 1992 年，而且于第二年出现了更多的活跃版本。同时期，Patrick Volkerding 发布了基于早期维护不善的 SLS 版本的 Slackware。1993 年，Ian Murdock 创立了 Debian 公司并在年底发行了 Debian 发行版。

Linux 发行版将核心操作系统组件（包括内核）结合在一起，同时将它们与其他组件（如实用程序、编程工具和 Web 服务器）结合起来，以创建适合运行应用程序的工作环境。Linux 发行版的出现体现了一种共识：一个操作系统内核，甚至内核加上一组核心实用程序（例如 Linux 操作系统中组成 GNU 的实用程序）本身有明显的局限。

在接下来的 10 年中，尽管只有少数 Linux 发行版通过商业渠道卖出，但是 Linux 发行版的数量依旧保持了爆炸式增长。

技术支持是商业 Linux 发行版中最先增加的内容之一。这种技术支持最初与传统零售盒装软件的情况类似：如果你在安装过程中遇到问题，或者软件无法按承诺工作，或者你想报告一个软件错误，可以随时致电帮助热线。然而，技术支持本身只是商业软件产品的许多方面的一部分。正如 Steven Weber 所写的那样，这一切都是关于“围绕开源过程构建可盈利的经济模型”。

6.3.4 订阅——不仅仅是“支持”

一款开源软件的产品订阅形式是什么样的？与 Adobe Creative Cloud 类型的产品不同，即使你的订阅失效，你也不会失去访问代码的权限，毕竟，它是开源的。

在研究开源开发模式时，我们讨论了“基于开源项目打造商业产品”的典型模式。这种模式建立在社区（共同体）基础之上，通过实验、创新和完善并随着时间的推移迭代发展。通过这种模式，开发者可以更好地参与社区，增加客户想要的特性和功能，然后测试、加固、编译以向客户分发稳定可行的版本。

Michael Tiemann 在 1999 年写的一篇文章中提到，开源软件商业模式的一部分是在公司内部建立产品的设计、优化和维护方面的专业知识。对售卖专有软件的公司来说，这似乎是再明确不过的。然而，对开源软件来说，如果没有社区的积极参与，就很难提供有效的支持。这种专业知识的沉淀可以有效解决在社区参与不够及时的情况下困难的软件支持问题。

沉淀专业知识并不仅仅在“软件支持”的场景下有意义。软件用户通常希望影响产品方向或新功能的开发。使用开源软件，用户可以直接这样做。然而，对不熟悉社区的商业公司来说，与开源软件的社区合作并不一定容易或显而易见。就我们所了解，这种合作还没有通用的模板。社区有不同的治理模式、习惯和流程，甚至有时候会有些怪癖。即使那些直接参与设置它们所使用软件方向的组织，也可以从这种专业沉淀中受益。

6.3.5 专注于“核心竞争力”

许多组织不想（或不应该）花费金钱和精力开发它们自己使用的所有软件。21世纪初，作为一名行业分析师，笔者会与银行和其他金融机构交谈。它们是继互联网基础设施供应商之后较早一批采用Linux操作系统的公司之一。大型银行拥有“内核工程总监”之类的技术专家，但实际情况是，银行实际上并不需要自己编写和支持操作系统。它们只需要“有操作系统可用”就行。正如银行需要“有银行网点可用”但并不需要自己作为建筑公司来建造网点是一样的道理。

随着时间的推移，银行和其他终端用户确实越来越多地参与了基于社区的开源开发。他们制定了共同利益的标准，如AMQP。今天，终端用户越来越多地参与到许多软件基金会中。然而，特别是对于主要是基础设施的平台，大多数企业更愿意让专门从事计算机硬件和通用软件的公司承担大部分繁重的工作，而它们自己则可以专注于开发与竞争对手差异化的技术和能力。

最终，订阅用户可以选择他们参与和影响技术和行业创新的程度。他们既可以选择像使用其他产品一样使用开源产品，也可以积极地在设定开发方向方面发挥作用，而使用专有产品则很难做到类似的

程度。

“开源软件订阅”完全可以提供商业软件产品所提供的修复、更新、帮助和认证，这对许多客户来说可能已经足够了。而可以参与开源开发模式这一额外特色，为开源软件订阅创造了专有软件所无法提供的全新机会。

6.3.6 订阅与激励相结合

此外，订阅为供应商创造了不同于预先许可证的激励机制。专有软件许可证的常用商业模式是：首先在售卖时收取许可费用；其次在主要的新版本发布的时候，收取升级费用；之后持续收取维护费用。而如果软件比较复杂，可能还会加入重要的咨询收费组件。因此，供应商有强烈的动机鼓励升级。在实践中，这意味着，虽然销售软件的公司有合同义务修复程序缺陷和安全漏洞，但它们实际上更希望客户在有新版本可用时，升级到新版本。因此，软件供应商会非常不愿意在现有软件中添加新功能特性。

而订阅模式则不同。只要客户继续订阅，是否升级到新版本对供应商来说并不重要。随着时间的推移，一些动力将促使客户升级。向旧版本添加新功能所需要付出的努力会增加，而且在某些时候，继续为旧版本提供支持可能会变得非常困难，还不如升级到新版本。但是，在这个过程中没有任何成本方面的刺激，因此不会给客户造成“强制升级”的人为紧迫感。

当然，关于订阅的评价有时的确不好，尤其是在专有消费者软件和服务领域。但这主要是因为，对于大多数此类订阅，我们只有持续支付订阅费用后才能继续使用软件。对于那些只是偶尔使用的软件场景，与

其持续付钱订阅，还不如用一个 5 年前的已经不被支持但依然可以完成手头任务的软件包划算。

而开源订阅是不同的。因为即使订阅失效，客户仍然可以完全控制和访问相应的软件代码。这是自由软件和开源软件的基础：你爱怎么做就怎么做。这是该商业模式的核心——使得围绕开源建立盈利公司成为可能。

6.3.7 云服务带来的转变

到目前为止，关于商业模式的讨论主要集中在传统软件环境下销售开源软件。这些传统软件是你在自己的计算机上安装和运行的。当然，如今情况已经有了变化。

单纯地在别人拥有和操作的计算机上或多或少地运行同样的软件，并不会改变基本原理。即使硬件和电力保障是别人的责任，你仍然要获得并承担维护软件的主要责任。

然而，一旦我们开始谈论以第三方供应商的服务形式来消费软件，许多东西会动态变化，尤其是与开源软件相关的软件商业模式。

从内部部署软件到“软件即服务”的转变是当今 IT 行业发生的较深刻的转变之一。第 7 章将深入讨论这个话题。然而，本章的其余部分将着眼于商业合作和软件开发的一些方式。这些方式随着开源软件的兴起以及商业模式的相应变化而普遍发生了转变。

6.4 从竞争到合作

有些变化可以说是直接从开源开发模式中得到启示的。其他变化更

可能是由一些相同的影响导致的，这些影响使开源软件的广泛采用成为可能，其中就包括互联网和廉价计算机。

随着开源的兴起，一个广泛的变化是“合作竞争”的情况越来越普遍。

6.4.1 合作竞争概念的产生

合作竞争一词最早可以追溯到 20 世纪初，但直到 Novell 公司的 Ray Noorda 在 20 世纪 90 年代用这个词描述公司的业务战略时，它才开始得到广泛使用。当时，Novell 公司正计划进军互联网门户业务，这要求其与一些相同的搜索引擎供应商以及其他公司建立合作伙伴关系，而这些公司也将与 Novell 公司展开业务竞争。

1996 年，哈佛商学院的 Adam Brandenburger 和耶鲁大学的 Barry Nalebuff 就这一主题写了一本《纽约时报》(*The New York Times*) 畅销书 *Co-opetition* (Harpercollins)，其中采用了 Noorda 的术语并通过博弈论的视角检验了这一概念。他们的描述如下：

有些人把商业完全视为竞争。他们认为做生意是在发动战争，除非别人输了，否则自己不会赢。另一些人则将商业完全视为合作团队和伙伴关系。但是商业既是合作也是竞争。这就是“合作竞争”。

这些基本原则一直存在。马歇尔大学的 Robert Deal 在《捕鲸法：争议解决、物权法和美国捕鲸者，1780—1880》(*The Law of the Whale Hunt: Dispute Resolution, Property Law, and American Whalers, 1780–1880*) (Cambridge University Press, 2016) 中描述了如何解决：

远离法庭和执法部门，粗俗剽悍的美国捕鲸船团队往往会在海上直接解决关于鲸鱼所有权的争端。由于只能靠自己解决争端，捕鲸者制定了规范和

习俗来决定由多个捕鲸船团队共同捕获的鲸鱼的所有权。

许多情况下，解决方案是在“完全无情的竞争”和“完全利他的合作”之间的某个平衡点。

6.4.2 为什么合作竞争不断增长

合作竞争背后的理论并不是很成熟，关于合作竞争在哪里最有效以及什么是最有效的策略仍然存在争议。Paavo Ritala 在 2012 年的一篇论文中指出，“有人认为这种情况发生在知识密集型行业，在这些行业中，竞争对手们在制定可互操作的解决方案和标准、研究开发和分担风险方面进行合作”。

这是对 IT 行业的一个很好的描述，但它也是对许多行业的一个越来越好的描述，这些行业越来越多地依赖于由软件支持的产品和服务，或者仅仅是软件。风险资本家（也是第一款广泛使用的网络浏览器的合著者）Marc Andreessen 在 2011 年《华尔街日报》(*Wall Street Journal*) 的一篇文章中的一句名言为“软件正在吞噬世界”。最近合作竞争的高调似乎至少是合理的。由于复杂性和客户需求而导致不合作，企业将会面对更多的困难。

相比之下，笔者还记得 20 世纪 90 年代初的一天收到一封来自一位销售代表的电子邮件。他很生气，因为他得知由笔者担任产品经理的新计算机系统中的网卡是由主要竞争对手 Digital Equipment 公司制造的。这位销售代表的措辞包括“我每天都和这些家伙打架，你却在背后捅我一刀”。

笔者讲这个故事是因为它很好地说明了计算机系统市场的变化程度。如今，还认为与竞争对手建立商业关系以提供某些零件或服务是可

耻的或者扎眼的，这种想法反而看起来很奇怪。Apple 公司、Google 公司和 Amazon 公司等在智能手机、语音助理和应用商店的混战中也有一些类似行为。但这些行为之所以引人注目，主要是因为它们并不是真正的常态。

合作竞争是大多数大型开源项目的核心。在这些项目中，参与者大多是从事软件开发的开发人员，这是他们日常工作的一部分。纵观 Linux 内核的主要贡献者，你将看到多家半导体公司、具有 Linux 发行版的软件公司、计算机系统供应商和云服务供应商。这些集团中的每一家公司通常都是直接竞争对手。之所以成立 OpenStack 基金会，很大程度上是为了明确创建一个能够容许竞争公司以利益方式参与的机构。大多数开源基金会的职能与标准组织和行业商会（如美国的 501(c)(6) 组织）类似。

6.4.3 开源——受益者和催化剂

开源软件开发既受益于合作竞争，也是合作竞争的催化剂。我们很容易忽略多家公司在同一个开源项目上合作这一方式的新奇之处。毕竟，通常这样的合作是以合资企业或其他类型的“合伙关系”来完成的。

我们发现开源项目可以大幅降低合作相关的间接开销。公司的合作方式有多种，其中很多都涉及合同、保密协议以及其他法律细节。虽然开源项目可能有一些需要参与者许可协议的地方，但在大多数情况下，开始处理项目非常简单。只需提交一个 pull 请求，让其他人知道你已将代码推送到存储库并且希望与现有代码库合并，合作就可以推进。

当然，广泛参与重大项目往往更正式、更有条理，具体细节将取决

于项目的治理模型。尽管如此，开源项目的协同方式往往比过去的公司协同工作的过程更轻量，开销更低，而且带来更快的速度。

我们已经看到一个具体的变化：现在经常开发软件标准。

6.4.4 合作竞争与标准

通常，我们讨论两种类型的标准（法律标准和事实标准）。一种类型是法律标准，或根据法律制定的标准。行业代表们（其中一些可能是竞争对手）作为标准组织或其他贸易组织的部分成员，坐下来创建了这些标准。这一过程可能非常漫长和艰难，而且常常以一种学术的方式制定冗长的、技术严苛的规范，而这些规范在现实世界中并没有得到太多应用。

追溯到 20 世纪 70 年代末的开放系统互连（Open Systems Interconnection, OSI）模型（注意不要与同样缩写的“开放源代码组织”混淆）就是一个例子。虽然 OSI 的概念性的七层模型已被广泛用作讨论网络软件栈，但以高成本开发的、直接实现 OSI 的大量软件从未被广泛使用。

相比之下，TCP/IP〔传输控制协议（TCP）和互联网协议（IP）〕于 20 世纪 60 年代末由美国国防高级研究计划局（Defense Advanced Research Projects Agency, DARPA）研究和开发。如今，它们是互联网使用的核心通信协议之一。尽管 TCP/IP 随后被批准为正式标准，但由于其广泛使用，在这之前就已经是一个事实标准了。另外，广泛使用的专有产品，例如 Microsoft 公司的 Windows 操作系统或 x86 处理器，也可以被视为事实标准。

开源已经发展成为一种倾向于事实标准的操作过程。它通过“代码优先”的合作竞争来形成代码层面的“事实标准化”。

我们已经在软件容器领域多次看到这种情况。虽然相关领域基于“开

放容器倡议”（Open Container Initiative，OCI）制定标准，但在其正式成为标准之前，图像运行时和图像格式等代码早已作为实现存在。容器编排也是如此，Kubernetes 逐渐成为编排和管理容器集群的常见方式之一。这是一个基于社区规模和采用情况的标准，而不完全依托于标准机构的行动。

这种方法允许公司合作开发软件，然后根据需要进行迭代，所以非常灵活。此外，事实标准还有其他优势。由于行业制定的“规范”往往很少完全指定所有内容，因此，基于标准的软件通常必须对未指定的细节和行为进行假设，而这可能会导致互操作性问题。（正式标准姗姗来迟，还可能导致供应商基于规范草案进行开发，从而引发进一步的问题。例如 20 世纪 90 年代存储网络领域的光纤通道标准。）

相比之下，通过“代码优先”方法落实的标准，已经是它自己的参考实现。因此，这是一种更有效的合作竞争方法，而不是在委员会中制定一个规范，之后让各方开始开发各自的实现。制定规范很容易带来一个现象——各方的实现中都有些微小的不兼容，而这种不兼容放在一起之后，会带来全局层面的互操作性问题。

6.5 对速度的需要

开源开发模式作为协作和创新方法的优势并不是 20 世纪中后期唯一有趣的 IT 趋势。许多事情以某种方式结合在一起，带来了新的平台和新的开发实践。

6.5.1 从物理到虚拟

服务器虚拟化正在成为主流，IT 部门也越来越适应它。虚拟化是用

于将物理服务器拆分成多台虚拟服务器的技术。服务器虚拟化最初的重点是减少所需的物理服务器数量，从而降低成本。降低成本是 21 世纪初技术低迷时期的一个重要考虑因素，但虚拟化的用处不止于此。无处不在的虚拟化意味着 IT 组织越来越接受这样一种观点，即它们不一定知道自己的应用程序实际在哪里运行。换句话说，另一个抽象层次正在成为常态，这在计算历史上发生过多次。

一个供应商和软件的生态系统正在虚拟化的基础上发展起来。该生态系统解决的一个具体痛点是“虚拟化蔓延”，这是一个由于虚拟化使得启动新系统变得轻而易举，从而更容易导致管理失控的问题。自动化、基于策略的管理、标准操作环境和自助服务管理等概念正开始取代过去的系统管理流程。这些流程不是通过手动进行处理，而是由一次性脚本来处理。

6.5.2 信息技术的消费化

IT 业也在消费化并更加移动化。到 21 世纪初，许多专业人士不再使用物理连接局域网的 PC。取而代之的是，他们使用带有无线网络移动连接的笔记本电脑。之后，Apple 公司在 2007 年推出 iPhone。不久后，智能手机已经无处不在了。员工购买的智能手机会用于个人和商业目的，而不再像黑莓时代那样主要作为商务使用。与此同时，在软件方面，在后网络时代的第二阶段，用户已经习惯了像 Amazon 和 Netflix 这样响应迅速流畅的消费者网站。单调乏味和难以使用的企业软件看起来比以往任何时代都缺乏吸引力。

公司的业务线用户也开始注意到 IT 部门响应请求的速度有多慢。企业 IT 部门进行了准确反驳，它们要在很多限制条件下运营，比如数据安全、详细的业务需求或正常运行时间等，而免费的社交媒体网站则没有

这些限制。尽管如此，消费者网络越来越多地设定了一个期望，如果 IT 部门不能或不愿满足这个期望，用户就会转向在线服务，或者购买计算资源，或者直接购买一个完整的在线应用程序。

这并不是因为企业 IT 部门比过去做得差了，而是因为 Amazon 和 Netflix 等大型互联网企业提升、更新和调整面向客户服务的速度，与传统 IT 能够做到的程度完全不同。虽然一些微小的部署问题可能会带来业务中断，但这些公司在许多方面不同于传统的企业，它们展示了未来的可能性。

这就引出了 DevOps。

6.6 DevOps 的崛起

笔者在前面安全方面谈及 DevOps。DevOps 越来越多地被 DevSecOps 作为一种提醒，提醒人们在整个 IT 开发和运营过程中安全是多么重要。但其实 DevOps 涉及软件开发、交付和操作过程的许多不同方面。在较高的层次上，我们可以把它看作将开源原则和实践应用于自动化、平台设计和文化。目标是使与软件相关的整个过程实现更灵活的小步快跑。DevOps 概念的核心思想是基于指标和数据的持续改进，这一核心思想改变了许多行业的生产方式。DevOps 是 Amazon 和 Netflix 成功的关键因素之一。

6.6.1 DevOps 起源故事

DevOps 起源于“敏捷软件开发方法论”。该方法论在 2001 年的一份宣言中被正式阐述，尽管它们的根源可以追溯得更久远。例如，在汽车行业和其他地方广泛采用的精益制造和持续改进方法中，就有敏捷和

DevOps 的先例。这种对应关系并不完美：精益方法在很大程度上侧重于减少库存，这并不能很好地映射到软件开发。但是，DevOps 与丰田方式（丰田生产系统的基础）的原则有不少共鸣，比如“尊重人”“正确的流程将产生正确的结果”以及“不断解决根本问题”等。欣赏这一传承也有助于我们理解：适当的工具和平台固然很重要，但文化和流程在 DevOps 中同样重要。

DevOps 一词是由比利时政府顾问 Patrick Debois 创造的。他在接受比利时政府的任务时，对应用程序实践和基础设施实践之间的隔离墙和缺乏凝聚力感到沮丧。John Allspaw 和 Paul Hammond 在 O'Reilly Velocity 09 会议上发表了题为《每天 10 次部署：Flickr 的开发和运营合作》的演讲。该演讲为 Debois 提供了灵感，他于 2009 年在比利时根特召开了自己的大会，名为 DevOpsDays，以讨论此类问题。DevOpsDays 也逐渐从一个基层社区成长为每年在世界各地举办几十次这样规模的活动。

Frederic Paul 在 2012 年 4 月的一次 InfoQ 视频采访中提到，Debois 承认命名这场运动为无心插柳，并非有意为之。“我选择‘DevOpsDays’来展现这是开发人员和运营人员的协同工作，因为‘敏捷系统管理’太长了。”Debois 说，“DevOps 作为一个词，从来没有一个宏伟的计划。”

另一个值得注意的 DevOps 时刻是 2013 年由 Gene Kim、Kevin Behr 和 George Spafford 撰写的图书 *The Phoenix Project: A Novel About IT* (IT Revolution Press)。这本书是一个关于 IT 经理的寓言——他不得不挽救一个陷入困境并导致前任被解雇的关键项目。一位董事会成员的导师指导他以全新的方式思考 IT、应用程序开发和安全问题，同时在此过程中引入 DevOps。尽管自那时以来，DevOps 已经得到发展并更加系统化〔包括 Gene Kim、Patrick Debois、Jez Humble 和 John Willis 撰写的《DevOps 手册：如何在技术组织中创造世界级的敏捷性、可靠性和安全性》(*The DevOps Handbook: How to Create World-Class Agility, Reliability, and*

Security in Technology Organizations) (IT Revolution Press, 2016)], 凤凰计划仍然是这场运动的一个有影响力文本。Kim 的《独角兽项目：关于开发者、数字颠覆和数据时代的繁荣的小说》(*The Unicorn Project: A Novel about Developers, Digital Disruption, and Thriving in the Age of Data*) (IT Revolution Press, 2019) 是最近的一次重述。

6.6.2 DevOps——不仅仅是敏捷

DevOps 扩展了敏捷原则，涵盖了包括生产在内的整个应用程序生命周期。因此，我们需要向包括设计人员、测试人员和开发人员在内的跨职能团队增添运维和安全技能。提高协作、沟通和跨职能技能水平是 DevOps 的重要宗旨。

在这里我们做一个极端思考，也许之后可能不再有专门的开发人员和运维人员，而是一群有 DevOps 技能集的人。更常见的是，DevOps 的这种观点关注的是“两批”跨职能团队——小型多学科的团队，他们从服务的初始阶段管控到整个服务生命周期。能做到这一点的一部分原因是这样的服务只要遵守 API 合同，就是自治的、上下游可控的，并且可以独立于其他服务和组进行开发的。另一个要求是这些“多面手”团队要具有操作底层平台所需的技能。

6.6.3 将不同的关注概念进行抽象

特别是在大型组织中，DevOps 已经演变为与现有跨职能团队、负责实时在线系统的待命工程师或编写代码的传统系统管理员都有所不同的含义。虽然这些模式可能仍会或多或少地被遵循，但更重要的关注点是将不同的“关注概念”进行清晰分离。核心工作是运维人员为开发人员

提供一个环境，然后尽可能不做任何干涉。

这就是 Netflix 前云计算和 DevOps 大师 Adrian Cockcroft 在 AWS 工作时写下“无需运营”这个词时的用意。由于 Netflix 是个有独特需求的、流量极高的视频网站，过去和现在也都会是一个特例。但是 Cockcroft 暗示了一个广泛适用的理念：在进化的 DevOps 中，运维人员做的大多事情都是将核心服务放在适当的位置，然后让路。创建基础设施、流程和工具的方式有其价值，这意味着开发人员不需要与运维人员进行太多的交互，亦可保持高效。（Netflix 主要使用亚马逊云服务运营，所以，尽管规模巨大，但它们自己运营的基础设施相对较少。）

正如之前所讨论的，降低开发人员和运维人员之间的沟通难度并不意味着需要“让交流变得更容易”，我们也可以“使交流变得不必要”。正如我们需要的可能不是与银行出纳员更有效的沟通，而是自助服务，是自动取款机。

使用这种 DevOps 模式，操作的关键方面发生在应用程序开发过程之外，而且独立于应用程序开发过程。

当然，开发人员和运维人员（以及其他团队，例如安全团队）之间的交流仍然很重要，但这种沟通不应该成为瓶颈。持续沟通是非常有效的方法，可以促进更好的协作，从而让团队：能在失败发生之前预警；通过反馈来不断完善和提升；获得足够的透明度。

6.6.4 站点可靠性工程师

我们接下来会展开来介绍站点可靠性工程。该概念由 Google 公司于 2003 年提出。当时由 Ben Treynor 领导的一个团队负责，目标是让 Google 公司的网站平稳、高效、可靠地运行。与其他拥有大规模基础设施的公

司一样，Google 公司发现现有的系统管理模式既不能提供自己所需的可靠性，也不能提供部署新特性的能力。

相应的解决思路是：站点可靠性工程师（Site Reliability Engineer, SRE）将把大约一半的时间花在运维相关的任务上，如手动干预和清理问题等。由于目标是使底层系统尽可能地自动化和自愈，SRE 也会投入大量时间编写软件，以减少手动干预或添加新功能的需要。从概念上说，这有点像传统的系统管理员在多次执行相同任务后不得不编写脚本。但是 SRE 的概念将这种做法发挥了更大的作用，同时将运维的角色转变为具有更大软件开发组件的角色。

Google 公司的 Seth Vargo 和 Liz Fong-Jones 认为 SRE 是 DevOps 的变体，或者“DevOps 就像编程中的一个抽象类，而 SRE 是该类的一种可能实现”。考虑到 SRE 团队支持实际开发软件服务的团队，笔者认为它更像是一种用于分离关注点的 DevOps 模式的进化形式。也就是说，SRE 方法可能确实会改变以运维为中心的角色和以开发为中心的角色之间的边界。一个具体的例子是“将应用程序的操作领域知识嵌入到一个容器集群中”，这与最初由 CoreOS 创建的操作员一样。总而言之，笔者认为 DevOps 仍然以专业运维为核心职能。

6.7 开源和 DevOps

开源与 DevOps 相关的部分涉及平台和工具、流程和自动化以及文化等多个方面。

6.7.1 平台和工具

DevOps 方法几乎可以应用于使用任何工具的任何平台上。它甚至可

以成为现有系统、应用程序和开发过程（以及新开发过程）之间的良好桥梁。虽然说世界上最好的工具也无法弥补“破碎的过程”或“有毒的文化”。但是，使用正确的平台和工具可以容易有效地简化 DevOps 工作流程。

开源工具是 DevOps 中的默认工具。市场研究机构 IDC 在 2015 年年初进行的“DevOps 思想领导力调研”中发现，高达 82% 的 DevOps 早期使用者表示，开源是“他们的 DevOps 战略的关键或重要推动者”。更重要的是，在执行 DevOps 计划的调查中，受访者认为开源和 DevOps 开源工具越来越重要。

在平台层面，推动新技术使用的一个关键趋势是从“静态平台”向“动态、可编程”的软件定义平台的转变，通过 API 进行相应控制。

容器及其相关技术是现代分布式应用程序平台的另一个重要元素。容器使 IT 环境和流程现代化，同时为实现 DevOps 提供了灵活的基础。从组织层面来说，容器使得“合理管理技术栈和进程的所有权”成为可能，可以减少所有权传递和随之而来的高开销的变更协调。从而使得应用程序团队拥有容器映像（包括所有依赖项），同时允许运维团队保留生产平台的完全所有权。

有了标准化的容器基础设施，IT 运营团队可以专注于构建和管理容器集群，以满足他们的安全标准、自动化需求、高可用性需求并最终满足他们的成本配置。

当考虑与 DevOps 相关的工具链时，一个很好的起点是持续集成/持续部署（Continuous Integration/Continuous Deployment, CI/CD）管道的自动化。最终目标是在传统 IT 和云本地 IT 之间使用一种通用语言，使得自动化普及和一致。举个例子，Ansible 是一个开源项目和产品，它允许将配置以一种人类和机器都可以读取的数据格式表示为“操作手册”。这

是基础架构代码化的一个例子。以这种方式为部署模式编写文档，便于与其他程序一起审核，也便于非开发人员阅读和理解。自动化通常是 DevOps 的一个关键组件，正如我们之前在安全自动化方面看到的，原因是自动化是确保可重复性所需的组件之一。

在 DevOps 环境中，其他广泛的开源工具也很常见，包括代码库（如 Git）、监控软件（如 Prometheus）、日志工具（如 Fluentd）和容器内容工具（如 Buildah）等。

6.7.2 流程

我们还看到开源开发流程与 DevOps 开发流程的一致性。虽然并不是每个开源项目都投入到全面实现 DevOps 工作流的前期工作中，但很多项目都这样做了。

例如，Edward Fry 讲述了一个社区的故事：

.....兼职的社区团队会带来一些巨大的好处。“工作计划”从漫长而艰巨的设计过程方式转变成快速“原型设计”或“分镜脚本”的方式，从而使得构建变得自动化、可靠和有弹性。测试和错误检测从被动变为主动，给客户带来了愉悦感。该团队现在由一个兼职经理监督项目，从前的多名全职项目经理被懂得如何自我管理的团队所取代。团队变得更小、更高效，这意味着更高的生产率和更高质量的项目交付。DevOps 带来的优质变化是喜闻乐见的，鲜有反对。

无论多少，重要的开源项目几乎都或多或少会有一些 DevOps 的特征。

例如，它们需要一个公共的、一致的代码视图。DevOps 和开源项目

都很适合使用分布式方法，即每个开发人员直接使用他们自己的本地存储库工作，存储库之间的更改作为单独的步骤共享。事实上，在 DevOps 平台上广泛使用的 Git 是由 Linus Torvalds 根据 Linux 内核项目的需要设计的，它具有去中心化、快速、灵活和鲁棒等特色。

还记得之前讨论过的通过向新贡献者提供快速反馈并在代码准备就绪时整合他们的代码，从而为新贡献者创造良好的体验这一点吗？自动化和 CI/CD 系统恰恰是实现自动化测试、更快构建软件和更频繁发布版本的一种极好方法。

1. 迭代、实验以及失败

从更高层面来看，DevOps 支持快速迭代，虽然这听起来很像软件开发的集市方法，但它们并不完全相关联。使用 DevOps 方法开发和操作的软件仍然可以通过系统精密的架构设计来完成。DevOps 是一种更普遍的理念，其包含增量更改、模块化和实验等特性。

接下来我们谈一下实验这一特性。通过实验，我们可以展现出故障。

故障这个词有一种负面的感觉。在工程和建筑项目中，它会让人联想到泰坦尼克号的沉没、塔科马海峡大桥在风中扭曲或者挑战者号航天飞机爆炸这些场景。这些都是工程设计或工程管理不善带来的系统性故障。

纯软件领域中的大多数故障不会像前面介绍的这样具象化，但它们同样会造成广泛的财务和人力成本消耗。想想美国医保系统网站项目的失败，塔吉特连锁超市的数据泄露事件或者其他那些数百万美元但没什么效果的软件项目。另一个例子是，2012 年美国空军在累计花费了 10 亿美元后，取消了一个企业资源规划（Enterprise Resource Planning, ERP）软件项目。以上这些都不能认为是孤立的失败事件。

在这种情况下，推卸责任是惯例。即使大多数参与其中的人并没有像泰坦尼克号那样真的随船沉没，但人们还是会被解雇，职业生涯也会中断。互联网对个人和组织都有很大的影响。

但我们如何将这与 DevOps 中频繁出现的“直面故障”警告联系起来呢？如果这些失败故障无法避免，我们如何惩罚应对？

并非所有的故障都是类似的。理解不同类型的故障并构建环境和流程来尽量减少不良的类型是成功的关键。正如 Megan McCardle 在 *The Upside of Down: Why Failure Well is The Key to Success* (Penguin Books, 2015) 一书中所写的那样，关键在于如何“优雅地失败”。

在那本书中，McArdle 描述了“棉花糖挑战”——一个最初由 Palm 前设计副总裁 Peter Skillman 设计的实验。在这项挑战中，每个小组会收到 20 根意大利面条、一盘胶带、一捆绳子和一个棉花糖。他们的目标是建造一个结构，让棉花糖距离地面尽可能远。

Skillman 实验涵盖了多种类型的参与者：从商学院学生到工程师，再到幼儿园的孩子。在这些人中，商学院的学生表现最差。据 Skillman 说，他们花了太多时间争论谁将成为“意大利面公司”的首席执行官。工程师类型参与者做得很好，但也没有名列前茅。作为一个拥有工程学位并在聚会中参加过类似活动的人，笔者怀疑他们花了太多的时间讨论使用前置“瀑布式软件开发方法”的最佳结构设计方法。

相比之下，幼儿园的孩子们并没有坐在一起谈论这个问题。他们选择了直接开始建造，以确定哪些可行，哪些不可行。他们做得最好。

建立一个允许并鼓励此类实验的系统和环境，将使得敏捷软件开发中的“成功地失败”成为可能。当然，这并不意味着没有人要为失败负责，相反，它使问责变得更容易，因为“问责”并不等同于“造成某种灾难”。在这方面，它改变了“责任”本身的性质。

当我们考虑这样一个系统时，我们应该考虑 4 个原则：正确的范围、正确的方法、正确的工作流和正确的激励。

2. 正确的范围

“正确的范围”原则专注于“将故障的影响限制在小范围，同时有效避免连带失败反应”这一核心。这是鼓励实验的关键，因为故障的影响能被降到最低。（而且，如果你没有失败过，你就不是在试验。）通常，你希望将“活动”和“决策”拆解开。从 DevOps 的角度来看，这意味着通过部署小型的、自治的和有边界的上下文服务（如微服务或类似的模式）来部署渐进的、频繁的和例行的事件。

3. 正确的方法

“正确的方法”原则是指需要不断地试验、迭代和改进。这又回到丰田生产系统的改善（持续改进）和其他制造先例。最有效的流程是保持沟通（比如通过 Scrum 和看板）并提倡协作，从而在可能的失败发生之前未雨绸缪。同时，如果故障确实发生了，该机制会通过反馈来不断进行改进并提倡持续学习以取得进步。

4. 正确的工作流

“正确的工作流”这一原则通过反复自动化来实现一致性，从而减少由于不可避免的偶然错误（如命令输入错误）而导致的故障数量。这使得我们能够更加关注设计错误和其他系统性故障的原因。DevOps 大部分采用 CI/CD 工作流的形式，该工作流使用监控、反馈循环和自动化测试套件来尽可能早地捕获过程中的故障。

5. 正确的激励

“正确的激励”原则使奖励和行为与理想的结果保持一致。激励计划（如晋升、金钱、认可）需要奖励信任、合作和创新。这里的关键词是：

每个人都需要能控制自己的成功。这里值得说明的是：失败并不总是一个正向结果。尤其是，如果这些失败是由于“反复不遵循既定的流程和设计规则”而导致的，需要明确错误的行动仍然应该有惩罚后果。

6. 文化

除了前面提到的 4 个原则以外，还有更重要的事情——文化。健康的文化是成功的 DevOps 项目和成功的开源项目与社区的先决条件。除了作为创新工具的来源以外，开源还是 DevOps 成功所需的迭代开发、开放协作和透明社区的伟大模式。

建立正确的文化，我们至少需要在一定程度上建立“允许优雅故障和失败”的组织和系统，从而使框架内的问责成为一种积极的属性，而不是指责游戏的一部分。这需要透明度。正确的文化同时还需要理解“好的决定也可能产生坏的结果”这件事。例如一项技术没有像预期的那样发展，市场发生了变化，一种架构方法被证明是不能伸缩的这些事情都可能会出现，从而影响结果。创新本身就是有风险的。我们应该减少损失，继续前进，避免陷入沉没成本谬误。

其中一个关键的转型要素是通过开放性和问责制在开发人员、运营人员、IT 管理人员和业务所有者之间建立信任。

最终，当 DevOps 的原则渗透到组织中而不是局限于开发人员和 IT 运营角色时，DevOps 将变得最有效。这包括制定激励措施以鼓励试验和（快速）试错、提高决策过程透明度以及鼓励信任与合作的奖励制度。开源模式的分布式丰富通信流对 DevOps 项目和更广泛的现代组织来说同样重要。

7. 改变文化

文化的转变总是具有挑战性的，而且往往需要以进化的方式演

进。例如，Target 公司首席信息官 Mike McNamara 在 2017 年的一次采访中指出：

你可能会面对的是传统思维——“我的领域不能是敏捷的，因为……”。这是一种对变革的自然阻力。在一些关键任务领域，这种担心是有道理的。因此，在这些领域，我们开始以敏捷的方式开发发行版，但仍在可控的环境中发行。随着团队对支持 CI/CD 的流程和工具越来越熟悉，他们自然开始变得越来越敏捷。

人们会倾向于说：在开源项目和 DevOps 中，正确处理文化层面的问题是非常重要的。这种思维有点狭隘。在 IT 和其他领域，文化是一个更广泛的故事。尽管我们都在谈论技术，但从某种意义上说，技术是最容易的部分，关于“人”的部分才是最难处理的。正如麻省理工学院斯隆管理学院的 George Westerman 在主持 2020 年麻省理工学院斯隆管理学院的一次座谈时所说，“数字创新的第一定律：技术变化很快，相比之下组织变化慢得多，而最遗憾的是，组织文化的变化则更慢”。然而，推进组织文化变更是必须的，因为正如 CarMax 公司的首席信息和技术官 Shamim Mohammad 在同一次活动上所说，“‘文化’是运营公司的操作系统”。

在 *The Open Organization Guide to IT Culture Change* (Red Hat, 2017) 中，Red Hat 公司首席信息官 Mike Kelley 观察到：

这种向开放原则和实践的转变为 IT 领导者带来了前所未有的挑战。随着团队变得更具包容性和协作性，领导者必须改变他们的战略和战术，以利用这种新的工作方式产生的能量。他们需要完善自己的方法，吸引多方人员参与对话，同时确保每个人都能感受到被倾听。他们需要磨炼自己的能力，将团队正在做的工作与组织的价值观和目标联系起来，以确保部门中的每个人都明白他们是比自己（和个人自我）更重要的事情的一部分。

6.8 无处不在的开源

弄清楚并正确使用与开源软件产品相应的商业模式，是非常重要的一件事。可行的商业模式不仅包括项目使用，还包括项目回馈，以维持健康的开源社区等多方面。虽然许多个人都有动力在自己的时间内为开源项目做出贡献，但当今世界大量的开源软件依赖于企业的贡献，而这种贡献是企业盈利商业计划的一部分。

不过，商业模式并不是孤立存在的。对软件企业来说，它们与开源一起成长起来的开发模式和实践（如 DevOps）有着关键的共生关系。所有这些都关联于更广泛的“组织文化”背景下。

我们要肯定可行的商业模式在当前是存在的，尽管在一些“想要搭便车而不喜欢贡献”的公司影响下，要正确使用它们是一种挑战。第 7 章将更深入地讨论这些主题，特别是在云计算的背景下。事实上，许多组织已经发现参与开源软件开发，甚至在其业务的其他方面采用开源实践，具有更广泛的好处。