第 2 章　可扩展性技术组织的角色

孙子说：将弱不严，教道不明，吏卒无常，陈兵纵横，曰乱。

可扩展性和可用性失败的共同原因是责任不清。本章通过回顾角色不清，责任不明的两个真实案例，讨论执行人员的角色、组织的责任以及各种独立贡献者的角色。最后，我们通过引入可以协助定义责任和有助于减少情感冲突的工具来结束本章。

本章适用于任何规模的公司。对大公司，可以作为一个清单来确保厘清与扩展性相关的技术和执行人员的角色和责任。对小公司，帮助开启定义与可扩展性相关角色的过程。对技术新手，可以作为了解技术组织如何工作的入门读物。对于经验老到的技术专家，可以帮助大家回顾组织的结构，以确认可以满足扩展性的需求。对所有的公司，本章清楚地解释了公司里的每个人对扩展性都有自己特定的作用。

2.1　失败的影响

有时，角色和责任不清意味着有些事情没人去做。比如，公司

没有安排任何人或团队去负责系统的容量规划。在这种情况下，系统的容量规划是通过比较系统未来的需要和现有的容量，拟定计划来确保系统有足够的容量来达到甚至超过需要，这包括申请购买服务器、软件调优或者增加持续层的数据存储，比如，数据库或NoSQL 服务器。

在这种情况下，缺乏团队或者个人来负责系统的容量规划，对一个快速增长的公司是灾难性的。然而这种情况引起的失败却经常发生，特别是在那些新公司里面。即使公司指定了负责系统容量规划的人或者组织，往往因为系统过去没有数据积累而无法规划。

“无人负责”问题的另一个极端是有多个组织或人被赋予了同样的目标。请注意，这和“共享一个目标”并不是同一件事情。共享目标是好的，因为“共享”内含合作之意。我们这里描述的是目标有几个主人，而且彼此之间没有清楚地说明谁该去做什么，以达成什么目标，有时候，他们并不知道其他的人或团队在做着相同的事情。在小公司里，这种问题看起来好像有些好笑，因为每个人都知道其他人在做什么。不幸的是，这种问题存在于很多大公司中，而且当它发生时，不仅浪费金钱，而且还破坏股东的价值，同时在组织之间产生长期的怨恨，降低员工的士气。造成团队士气低落的最大的原因，莫过于某个团队对某项任务负全责，因为团队成员以为其他人在负责某个部分而没有去做，结果造成整个任务的失败。多人负责的问题是本书第 1 章所描述的基于情感冲突的核心问题所在。

作为一个案例，我们假设一个组织分成工程和运维两个部门，工程部门负责研发软件，运维部门负责搭建、配置和运行数据库、系统及网络。进一步，我们假设经验不足的 CTO 最近读了一本讨论

共享目标的书，所以决定把平台可扩展性的责任交给两个部门共同承担。公司的容量规划人员确定，要满足明年业务发展的需要，团队必须把客户合同管理子系统的处理能力提高一倍。

工程和运维部门的架构师们读过本书的技术部分，他们决定分割客户合同管理子系统的数据库。这两个部门的架构师们都相信他们已经得到足够的授权进行独立决策，他们没有意识到同样的责任已经被赋予了几个人，并且没有人通知他们去一起工作。工程部门的架构师认为以交易为功能进行分割最有效（这是网站的功能，比如在电子商务平台上买一个产品和看一个产品是两个不同的功能），相反，运维的架构师认为按照客户的边界来分割数据库效果是最好的，这样把不同组别的客户分在不同的数据库里。两组架构师都做好了分割的初步计划，组建好了团队，然后分别请求对方团队来做一些协助工作。

听上去这个例子可能有点儿荒谬可笑，但是却经常发生。在最好的情况下，两个团队会停下来解决纠纷，损失的只是两组架构师的宝贵时间。不幸的是，通常情况下，团队会变得极端化，甚至会在政治斗争上浪费更多的时间。如果把这个项目在一开始就交给单个人或团队来承担，同时听取另一个团队的建议，解决方案的效果可能会更好。

2.2　定义角色

这个部分将以实例来讲解如何通过清楚地定义角色来解决基于责任的冲突。案例讲述了在典型的技术组织结构内，如何定义公司

高管团队和独立贡献者的角色。

实例中提到的高管和独立贡献者，他们的责任并不局限于某个特定的职位或组织。相反其目的是要帮助厘清公司里必要的角色，清楚地定义某些责任或聚焦的专业领域。为了让大多数的读者能够更容易地理解，我们选择定义那些公司里普遍存在的传统角色。例如，运维、基础设施、工程和质量保证（QA）在同一个团队里，每个团队专门负责一个产品线（事实上，我们强烈地推荐这种组织结构，在下一章你将会看到）。

你可能还记得，我们在前面的引言中提到过组织的设计并没有对错之分，任何的组织结构决策都有利有弊。重要的是在组织的设计中要包括全部的责任，不仅要清楚地定义谁是决策者，还要搞清楚谁负责为决策提供信息，决策和行动方案都应该通知谁，谁来负责执行决策。做出最佳决策最为关键的要素是有一个最佳的决策过程，来确保合适的人收集合适的信息，并把它提供给最终的决策者。这适用于公司里任何要做决策的人。我们会在本章的概要部分，用一套有价值的工具对最后这一点进行讨论。尽管我们无法彻底消除冲突，但是合理的组织结构可以帮助我们限制一些因为缺乏清晰度而造成的冲突。作为一个领导者你至少应该清楚在自己的组织内，每个团队的责任和期望的产出。

有关权力下放的注解

在讲组织内责任划分问题之前，我们认为讨论一下权力下放是很重要的。当定义一个组织的角色和责任的时候，你其实就是在设计一幅权力下放的蓝图。广义地说，权力下放就是授权别人

做你该做的事情。比如，由架构师或架构师团队来设计系统，就是你把架构设计工作的权力下放给那个团队。根据公司大小和团队的能力，你或许也会把做决策的权力下放给某个团队。

这里有一点非常重要，那就是你可以下放任何权力，但是必须对其结果承担所有的责任。接受你权力下放的个人或团队最多承担连带的责任，虽然你可以解雇、提升、奖励或惩罚团队，但是必须清楚自己要对最终的结果负全责。好的领导本能地明白这个道理，他们总是把赞扬留给团队，承认失败并公开地承担责任。相反，差的领导在失败时找替罪羊，在成功时抢功。

为了帮助你理解这一点，我们来做个“股东利益”的测试。假设你是一个公司的 CEO，你决定把某个业务单元交给一位总经理负责。当你告诉董事会或者股东，这个业务运营的结果和你无关时，请你想象一下股东们会有什么反应。更进一步，如果业务表现达不到预期，你觉得董事们会不会追究你全部或至少部分的责任？

这并不是说要你自己去做所有的决策。随着公司的成长和团队规模的扩大，你根本就不可能对所有的事情都去做决策。在很多情况下，事实上，你可能没有资格去做决定。例如，一个不懂技术的 CEO 或许不应该去做架构的决策，一个 200 人的工程机构的 CTO 不应该去写最重要的代码；这些高管们的工作经常需要由经理们去完成。现实告诉我们，你必须找到最好的人，然后才有可能把权力下放给他，并对这些人以最高的标准来严格要求。这也意味着你应该问最合适的问题，比如那些最为关键的项目和系统的决策是怎么做出来的？

2.3　执行人员的责任

公司的高管，对在公司中营造一种在第 1 章中提到的可扩展性的文化氛围负有最大的责任。

2.3.1　首席执行官

CEO 是公司里负责扩展性的最高官员。和公司里的其他事务一样，对于扩展性，CEO 是最终的决策者和扩展性的仲裁者。一个技术公司，好的 CEO 需要精通技术，但是这并不意味着他必须是一个技术专家或者主要的技术决策者。

很难想象一个升到 CEO 位置的人，读不懂资产负债表、损益表、现金流表。同样，除非有财务背景或者当过 CFO，否则很难理解财务制度里面的复杂逻辑。CEO 的工作就是提出合适的问题，让合适的人参与，并且协调外部的支持或建议来寻求正确的答案。

同理，在技术世界里，CEO 的工作是了解一些基础知识（相当于前面提到的财务报表），知道该问什么问题，知道去哪里和在什么时间可以得到帮助。在技术的组织机构中，我对没有做过 CTO 或 CIO，没有技术学历或工程师经历的 CEO 及其他的管理者有下面一些建议。

1. 提出问题并从答案中寻找一致性

你的部分工作就是寻找真相，只有真相才能使你做出及时和明智的决策。尽管我们不认为团队会说谎，但是对事实经常会有不同的解读，特别是那些涉及可扩展性的问题。当你对一些事情不理解，或者发现事情看起来不太对劲的时候，就要提出质疑。如果你没有

办法从不同的理解中甄别出事实，那就在答案的一致性上努力。如果你能够战胜自我或骄傲，提出一些看起来容易被忽略掉的问题，那么就会发现不仅学到了很多东西，而且也磨炼出了发现真相的重要技能。

很多成功的领导都具备“高管盘问”（executive interrogation）这个关键的能力。比尔·盖茨版本的盘问技能叫“比尔·盖茨审查”[⊖]。懂得在什么时候去调查、去哪里调查，直至找出满意的答案，这种技能不仅限于 CEO。事实上，经理和独立贡献者也应该尽早磨炼出这个技能。

2. 寻找外部的帮助

在可扩展性方面，要从朋友或专家那里寻求帮助。别指望着请他们进来，然后让他们帮你打理清楚，如果这么做，可能会产生很大的破坏性。相反，我们建议你与技术公司建立专业或个人的关系，依靠这些关系来帮助你提出正确的问题，并且当你要深究的时候，协助你评估答案。

3. 加强对可扩展性的理解

列个清单，把你自己在技术方面的弱点都罗列出来，特别是那些有问题的，然后寻求帮助使自己变得更聪明。可以向你的团队或者外部的专家提问，阅读与公司或产品的扩展性相关的互联网上的文章，参加那些为没有技术背景的人专门举办的研讨会。你可能已经通过阅读专业书籍，特别是那些普通技术类的书籍来解决这个问题。你没必要去学习程序语言、理解操作系统、搞清楚数据库是怎

⊖ 参见 Joel Spolsky. “My First BillG Review.” http://www.joelonsoftware.com/items/2006/ 06/16.html.

么工作的或者理解如何实现“并发的碰撞检测”。相反，你需要的是能够更好地提出和评估问题。扩展性是一个业务问题，但是要解决就必须熟悉相关的技术情况。很有可能，CEO 会把权力下放给团队里的几个人，包括 CFO、独立业务部门的负责人（总经理）和负责工程技术的高管（本书指 CTO 或者 CIO）。

2.3.2　首席财务官

一般来说，CEO 会把预算权下放给 CFO。系统容量规划是成功的可扩展系统中非常大的部分，前面的例子已经提过系统容量规划不力所带来的影响，系统容量规划的结果一般都会通知预算部门。预算办公室责任中的一个关键任务，就是要确保团队和公司能有足够的资金来扩展平台、产品和系统。要有足够的预算，公司才能通过增加或者减少服务器，雇佣更多称职的工程师和运维人员来扩展系统达到预期的需求。在公司的业务还没有那么大规模的情况下，不应该过度预算盲目地扩大规模，那样会稀释公司短期的净收入，收效甚微。适时地采购和部署系统及解决方案会优化公司的净收入和现金流。

CFO 也不太可能有技术背景，但是可以像 CEO 一样，通过构建适合的网络，提出正确合理的问题。CFO 可能需要问清楚，在制订预算的过程中是否考虑过其他的可扩展性方案，在确定预算方案的时候，做过什么样的权衡安排？其目的是要确保团队考虑得多一些。不太好的答案是“这是唯一可能的预算”，其实这不太可能，只有一种可能的事情极少。例如，如果公司面临降低成本的挑战，CFO 就要决定旧服务器和网络设备是否能继续使用，因为折旧会对

账面有较大的影响。然而事实上，旧设备通常效率较低而且容易出故障，会造成整体持有成本的大幅上升。比较好的回答是“我们评估了所有的可能性，这个方案用相对较低的成本来水平扩展，为未来的扩展搭好一个框架，使我们可以持续扩展。”

2.3.3　业务部门的负责人、总经理、产品线负责人

负责公司或部门盈亏的人，如业务部门的负责人等，要对平台、产品和系统的业务增长做好预测。在中小公司里，业务部门的负责人很有可能是 CEO，其权力和责任会下放给一些员工。然而，需求预测对确定扩展计划至关重要，这可以避免在公司的实际业务没有发生之前，安排过大的预算。

需求指的是系统承受用户并发请求的数量，我们经常会碰到业务负责人无法对需求进行预测的情况，这是绝对不能容忍的不负责任的行为。如果没有专人根据业务的发展进行需求预测，那么对可扩展性预测的责任就会被转移到技术部门，它们对业务需求的预测能力远没有业务部门的强。预测或许不准，特别是在公司发展的初期，但是开启这个过程至关重要，预测的准确度会随着时间的推移而逐渐成熟。最后，和公司里的其他高管一样，业务负责人有义务帮助营造可扩展性的文化氛围。高管要在技术部门里向同级或者下级提出合适的问题，确保那些技术伙伴们能够得到足够的资金，有效地协助有扩展性问题的业务部门。

2.3.4　首席技术官／首席信息官

CEO 是公司负责可扩展的最高官员。首席技术官主要负责技

术、流程和组织的可扩展性。对以技术为主要产品的公司，特别是互联网公司，常常把这位高管称为 CTO。在对产品的研发和交付中只负责技术支持作用的公司里，常常称其为首席信息官 CIO。如果公司里只有一位 CTO，那么这个人一般就聚焦在产品的科技上。本书混用 CTO 和 CIO 这两个词来指公司的技术高管。这位高管很可能有最强的背景和能力，在业务发展需要的时候，可以确保平台和系统有效地扩展。

对平台可扩展性的成功，CEO 责无旁贷，CTO 从 CEO 那里接过并与 CEO 共同分担这份责任。如果平台无法扩展将会导致 CTO 和部分组织，甚至 CEO 被解职。

CTO 或 CIO 必须要有公司的整体技术愿景，而且这个愿景要包括可扩展性。CTO 除了负责设定愿景中积极的、可度量的、可达成的目标外，同时还负责为团队配置合适的人员以确保其能完成可扩展性的相关使命。CTO/CIO 的责任还包括发展可扩展性的企业文化和过程，以确保公司走在用户需求的前面。

当公司的规模扩大的时候，CTO 或者 CIO 需要把某些有关可扩展性决策的责任下放。当然，如前所述，权力下放也绝不会解除高管确保按时、按预算做好可扩展性工作的责任。另外，在高速发展的公司里，可扩展性对公司的生死存亡至关重要，CTO 绝不应该把制订可扩展性发展愿景的权力下放。在这件事上，没有什么比“身先士卒”更为重要，而且制订愿景并不需要专深的技术。尽管我们所见到的最好的 CTO 有从独立贡献者到系统分析员或项目经理等不同的技术背景，但是我们也曾经见到过没有技术背景的成功 CTO。对没有技术背景的 CTO 或者 CIO 而言，他们必须要有技术的敏锐

性，同时具备较好的语言能力，了解技术领域内诸如时间、成本和质量之间的重要关系平衡。让一个初出茅庐的 CTO 来领导技术团队，就像让一个不会游泳的人从船上跳到湖里一样；假如那个人会游泳，你可能很欣慰，但更大的可能是，你要给自己再找个新的伙伴一起划船了。对初来乍到的 CTO 同等重要的是赢得技术人员的信任，没有手下的信任，CTO 无法有效地领导。

同样重要的是，CTO 要有一些业务感觉，但不幸的是，这和找一个有电气工程博士学位的首席市场官一样难，这类人存在，但是很难找到。大多数的科技工作者，他们在本科和研究生课程中，不学习商业、财务或者市场方面的知识。尽管 CTO 不必是一位资本市场的专家（那是 CFO 的工作），最好还是需要理解公司业务运作的基本情况。例如，CTO 应该能分析和理解损益表、资产负债表和现金流表之间的关系。在一个像互联网公司这样的以技术为中心的公司，CTO 通常拥有全公司最大的预算。这样一位高管经常负担着非常大的投资任务，购买电信运营商的服务、数据中心的租约、软件的许可权，所以缺乏对财务计划的了解会给公司带来很大的风险，如挪用资金、多付给供应商、无计划、没有预见的支出（定期软件许可费用的调整）。CTO 应当有市场营销的基础知识，至少读过社区学院或公司资助的专门课程。这并不是要求 CTO 是这些领域的专家，而是希望 CTO 对这些方面要有基本的了解，从而为可扩展性做好商业安排，同时可以有效地在商业界沟通。本书稍后会讨论这些方面的内容。

2.4　独立贡献者的责任

这里会对在大多数公司成长和扩展时需要独立贡献者所扮演的角色进行描述。这些角色包括总体架构、软件工程、系统运维、基础工程和质量管理。

我们的目标不是要严格定义组织或功能的边界。如前所述，通过功能组织是设计组织的一种方法。当应用瀑布方法开发的时候，尤为有效。我们将会在第 3 章中讨论，当公司聚焦在产品开发的时候，我们希望组织与公司的产品生产与开发的过程相匹配。

2.4.1　架构师的责任

架构师的责任是确保系统的设计和架构可以随着业务的发展而扩展。这里我们清楚地指出设计和实施之间是有差别的。架构师需要在业务需要发生之前就想好，远在业务部门的预测超过平台的容量之前，就已经对如何扩展系统深思熟虑了。例如，架构师可能已经开发了一个可扩展的数据存取层（DAL）或数据存取目标（DAO），当用户需求在任何一个方面增加的时候，允许通过不同的数据结构，从多个物理数据库存取数据。真正的实施或许只是一个数据库，通过对 DAL 或者 DAO 做的一些修改及迁移脚本的开发来提高效率。当需求发生时，用几周而不是几个月的时间，更多的数据库就可以被部署到生产环境。架构师也负责制定代码设计和系统实施的技术标准。

架构师负责设计系统并确保其设计能解决任何的扩展问题。在第二部分中我们会介绍一个架构团队应该采纳的关键过程，它可以

帮助架构师定位跨越所有技术领域的扩展性问题。

架构师也可以负责信息技术的管制、标准和过程，通过第 13 章讨论的架构审查委员会（ARB）和联合架构设计（JAD）执行这些标准。首席技术官会要求架构师履行他们的这些职责。有一些大的公司可能会组织流程 – 工程联合团队，负责对流程的定义和标准的执行。

公司给予那些聚焦在技术架构设计的独立贡献者不同的职衔，包括软件架构师、系统架构师、扩展性架构师、企业架构师和敏捷架构师。不管什么职衔，这些角色基本上都会聚焦在一两个领域。首先是软件架构师，主要关注如何设计和架构软件。这些架构师聚焦在不同的领域，例如，面向服务的架构框架或者代码模板为研发人员提供指导。其次是系统架构师，主要负责解决软件在硬件配置和支持方面的问题。这些架构师聚焦在去除单一的失败点，发现出错点和做好容量规划方面。从客户的角度看，系统的可扩展性是和系统架构最为相关的。本节的后半部分，会更加深入地讨论有关独立贡献者这一重要的角色。

2.4.2　工程师的责任

工程师是战斗在第一线的，真枪实弹的基层人员。工程师是可扩展性使命的首席实施官和系统的首席调优官。工程师遵循公司的架构标准，根据架构进行具体设计，并且最后完成代码的实现。工程师团队是最有可能真正了解系统局限性的仅有的几个团队，他们也是发现未来可用性问题的关键人员。

2.4.3　DevOps 的责任

DevOps 是“Development”（开发）和“Operations”（运维）两个词的合并和缩写，近来常用这个词描述软件研发团队和技术运维团队之间交互和合作的现象。这个词的出现代表着多年来大家所熟知的事实得到了认可，就是系统和服务既需要软件也需要硬件。由此软件研发和系统管理混合成 DevOps。

在 SaaS 和 Web 2.0 时代，DevOps 通常负责配置、运行和监控生产系统。这个团队也应该是本书后续将要讨论的敏捷开发的一部分。即使不想采用敏捷的组织结构或开发策略，至少也应该考虑让 DevOps 的人存在于工程团队中，以帮助研发人员更好地理解如何把研发好的应用配置到生产系统。DevOps 的人员既知道系统日常运行的情况，也了解系统资源的使用情况。只有他们有足够资格发现系统的瓶颈，并且在系统设计的时候时刻想着系统配置。

通常，DevOps 的人员负责研发和测试环境，包括研发和配置脚本、监控、日志和其他系统工具。DevOps 的人员负责输出报表展示一个阶段可用性的发展趋势，分析出问题的根源并给予纠正，确定各类问题的平均解决时间和平均恢复时间。

不管团队的构成情况怎么样，都是 DevOps 负责监控、报告应用与系统的健康情况和服务质量，在解决可用性问题的时候起着关键性的作用。这个部门所采用的管理问题和解决问题的流程会成为其他流程的入口信息，以帮助在系统发生大规模灾难前，确定扩展性问题之所在。运维所收集的数据，对进行系统容量规划、解决系统性及反复发生的可扩展性的问题，有令人难以置信的宝贵价值。架构和工程团队严重依赖 DevOps 来帮他们确定在什么时间解决什

么问题。在第二部分第 8 章和第 13 章我们会详细讨论这些流程。

2.4.4　基础设施工程师的责任

负责基础设施的工程师技能独特，在敏捷团队中甚至不必每天露面，但是却横跨很多个敏捷团队。有些大的企业，为了研发端对端的解决方案，把这部分事情放在敏捷团队中。但是在大多数的中小企业中，这部分人集中在基础设施部门中来支撑多个工程团队。这样的团队包括 DBA 数据库工程师、网络工程师、系统工程师和存储工程师。常常由他们来确定使用什么系统，什么时候购买，什么时候报废。不论基础设施人员团队的大小，其主要责任都包括设计共性资源的架构（像网络和传送系统），定义全局的存储架构，确定关系型或非关系型数据库的解决方案。

对使用云的公司来说，基础设施团队经常负责管理虚拟服务器、网络和信息安全。基础设施工程师也有助于寻找系统、网络和数据库的容量限制点，支持和帮助其他团队确定解决可扩展性相关问题的合适方法。

2.4.5　质量保证工程师的责任

在理想情况下，质量保证工程师主要是指那些负责应用测试，确保测试结果和公司期望的结果一致的工程师，在可扩展性测试过程中也同样起着重要的作用。新产品和功能会改变系统、平台或者产品的需求特性，最常见的是增加新功能，从而产生对系统资源的额外需求。最为理想的做法是我们掌握新产品的需求的特点，确保新功能和新应用不会给生产环境带来大的冲击。质量保证的专业人

员需要了解和掌握其他的变更情况，以确保应用能够及时通过可扩展性相关的测试。质量保证一定要聚焦自动化的测试，而不是手工测试。不仅仅生产系统要可以扩展，而且当添加新功能的时候，测试的过程也要可以高效扩展。

2.4.6　系统容量规划师的责任

有系统容量规划责任的人可以在任何团队工作，但是他们需要能够取得最新的系统、产品和平台的性能数据，系统容量规划是高效扩展和成本控制的关键。如果做得好，对那些很容易做水平扩展的系统可以适时地买进设备，对无法进行水平扩展的系统可以安排紧急购买大型设备，清查系统的可扩展性问题，分优先级逐步予以解决。

你或许注意到了，我们在描述采购大型系统的时候用了紧急两个字，很多的公司把这种按比例放大的方法当成有效的策略。正如我们将从第 19 章到第 23 章要讨论的那样，如果你的扩展策略是依靠更快和更大的硬件设备，那么该解决方案将不具备可扩展性，你的可扩展性取决于供应商。如果说换更大更快的硬件能够扩展系统，那么购买更大更快的车是否可以使你跑得更快？在还没有赚那么多钱之前，你只能和收入水平类似的人开得一样快。可扩展能力与更大更快的系统以及应用服务器的新版本没有关系。

缺乏角色和技能会怎么样

eBay 的故事

你是否曾想过，如果请一位木匠来家里修理水管，会是怎么

样的一种情况？马斯洛的锤子理论预见了这个假设的结局，很有可能，木匠会拎着锤子来完成工作。你很可能以前就听说过马斯洛的锤子理论，通常和另一句一起出现，“如果你手里只有一把锤子，那么所有你看到的都是钉子”。

故事得从 2001 年我们的合伙人汤姆·科文加入 eBay 说起。第一天，汤姆作为负责基础设施的副总裁，走进了运维中心，随便指着挂在墙上显示数据的一些大屏幕问道：“这些是做什么用的？”

一位运维操作员转过身来，看了一下汤姆，然后转回身继续做他的事情。“那是我们的控制器”，运维操作员回答道。

汤姆又问道：“什么是控制器？”

“接受来自于用户的请求，然后分配到网站的服务器上”，运维操作员回答道。

“哦，是负载均衡设备”，汤姆说。

“我不知道负载均衡是什么玩意”，运维操作员说。

这件事看起来有点儿怪，但却是对 2001 年发生的事件的准确描述。对话的细节或许无意中有些许改变，但是对话的要点确实相差不多。在那个时候，网上交易才只有几年的历史，并不是每个人都能够理解系统基础设施方面的基本概念，当然这在今天已经是司空见惯的事情了。

汤姆所指的以硬件为基础的负载均衡器，当时（2001 年）已经在市场上出现了好几年的时间。这些设备不仅仅用在网站上，也用在各类 CS 架构的系统上，服务于公司的内部员工。我们来讨论一下这个故事说明的问题。

汤姆决定寻找 eBay 当时使用的更多的“控制器”，结果发现这些设备已经使用了好几年，在负载均衡之前，eBay 使用 DNS（域名服务）轮询的技术来控制网络的流量。（DNS 轮询技术我们至今仍然还能偶尔在一些客户那里看到，这可不是开玩笑。）此外，原来负责选择解决方案的人，是没有多少基础设施经验的软件工程师。

汤姆对软件和硬件负载均衡都有所了解。例如，他知道硬件负载均衡每秒能处理更多的请求，而且比它的软件表弟（“控制器”）的故障率更低。因为软件负载均衡是按每台设备或每张许可证收取的，所以成本更高，不过如果把吞吐量也考虑在内，每笔交易的费用一般会比硬件的低。如果再把整体可用性（较低的设备故障率）考虑进去，那么在选择负载均衡解决方案的时候，硬件负载均衡器明显胜出。

汤姆经常表现出他对基础设施架构的奇妙偏好，其实他的洞察力并没有什么神奇魔法。这些洞察力来源于 20 年来在基础设施、运维和解决方案架构的生产第一线。积累起来的知识实属弥足珍贵，来之不易。汤姆重新审查了与软件负载均衡相关的事件以及许可证的费用，结果很快发现，如果把负载均衡换成硬件，不仅可以提高可用性，而且还能降低成本。

这里所学到的是 eBay 缺乏关键的技能，即深而广的基础设施经验和知识。这并不是说 eBay 在汤姆加入前没有特别好的基础设施人才，实际上，eBay 确实有这类人才。问题是在做关键的负载均衡决策的时候考虑的范围不够广泛，没让这类人才有机会参与决策，以避免错误。对 eBay 平台系统容量的管理乏人问津，显

而易见，这是个可扩展性相关的错误。

汤姆的决策最后稍微降低了 eBay 每笔交易的成本，提高了系统可用性。尽管结果并不是惊天动地，但是股东（成本降低）和客户（可用性提高）最终都受益了。这正如我们说的，积少成多，集腋成裘。

2.5　RASCI 工具

我们有许多客户在使用一种简单的工具，来清楚地定义项目中的角色。每当我们进入一间需要做可扩展性项目的公司，我们就用该工具来定义谁该干什么，去除浪费的资源，确保全面覆盖扩展性相关的所有需求。尽管这只是一个过程，但因为本章是论述角色和责任的，我们觉得很有必要介绍这个工具。

我们用的这个工具叫 RASCI，是一套用来确定责任的表格，RASCI 是指负责、批准、支持、咨询和知情。

R：负责（Responsible）对项目或者任务的完成负责的人。

A：批准（Accountable）项目关键决策的批准人。

S：支持（Supportive）为项目完成提供资源的人。

C：咨询（Consulted）为项目提供数据或者信息的人。

I：知情（Informed）需要了解项目相关情况的人。

RASCI 可以用在矩阵当中，每个活动或者任务标在 Y 轴上，每个独立贡献者或者组织的名字标在 X 轴上。活动（Y 轴）和组织（X 轴）的交叉将会有（R、A、S、C、I）中的一个字母，如果交叉

处什么都没有，那么相关的独立贡献者或者组织就不是这个任务的一部分。

在理想情况下，一个任务会有一个 R 和一个 A。这个工具可以帮我们解决本章前面提到的那个问题，多个组织或个人认为他们对任何工作负责。这样可以明确只有一个人或者组织对任务的成败负责，坚守“职责清楚、奖惩分明”的原则。一种更为温和的说法是分给几个人负责的项目等于没有人负责。

这并不是说其他的人不允许为项目或者任务提供建议。RASCI 模型明确允许并执行对顾问、公司内部或外部可以为任务增加价值的人的使用。A 不应该批准 R 的方案，直至 R 已经就方案的正确性咨询了所有相关的人。当然，如果公司的文化好，R 不仅可以寻求人们的帮助，而且会让那些被咨询的人感觉到自己有价值，其价值已经被考虑到决策支持的过程中。

只要你愿意加，觉得有价值，或者对完成项目来说是必需的，你可以增加多个 C、S 和 I，但是，同时要注意不要过度知会。记住本书前面提到的关于陷入邮件和沟通泥沼中的事例。新的公司往往假设决策应该让每个人都参与或者知会，这种信息的发布机制是没有可扩展性的，结果是大家花时间读邮件，而不是去做他们应该做的，能为股东产生价值的事情。

表 2-1 是一个部分完成了的 RASCI 矩阵样例。根据我们关于不同角色的讨论，让我们来看看如何完成这个 RASCI 矩阵。

早些时候，我们曾指出，CEO 必须对可扩展的文化氛围负责，这是公司可扩展的 DNA。从理论上说，尽管让 CEO 把责任下放给公司的其他人，在客观上是可行的，就像你在关于领导的一章中读

到的，这个高管必须践行公司和平台可扩展文化的价值。因为我们在讨论公司如何行动才能实现可扩展性，即使 CEO 授权，也还是必须要对可扩展性负责。因此，我们把 R 放在 CEO 列和可扩展性任务行。很明显，CEO 对董事会负责，因为营造可扩展性的文化氛围和整体的文化氛围有关，所以我们把董事会标成 A。

表 2-1　RASCI 矩阵

	CEO	业务经理	CTO	CFO	架构师	工程师	运维工程师	信息安全	质量保证工程师	董事
扩展性文化	R									A
扩展性的技术愿景	A	C	R	C	S	S	S	S	S	I
产品扩展设计			A		R					
软件扩展实施			A			R	S			
硬件扩展实施			A				S	R		
数据库扩展实施			A				S	R		
扩展性验证			A						R	

对可扩展文化任务而言，谁是 S？应该知会谁？应该咨询谁？在寻找答案的过程中，任何情况都允许有支持者 S 存在，也可以在寻找解决方案的时候有顾问 C 参与。因为 C 和 S 的参与所以才会有结果，因此一般上没有必要包括 I，那个需要沟通决策和执行结果的人。

我们填了可扩展性的技术愿景一行。如前所指，CTO 负责制订产品、平台和系统可扩展性的愿景。CTO 的老板是 CEO，所以 CEO 要负责审批决策和路线。请注意，在决策中 R 的老板不一定非得是 A。R 完全可能代替某个和他不相干的人去完成某个任务。在这种情况下，假如 CTO 为 CEO 工作，那么由 CEO 以外的人去批准可扩展性愿景或者计划的可能性极小。

CTO 可扩展性愿景的顾问包括那些需要依赖 CTO 来实现生产系统可用性或公司运营后台系统的人。这些人需要被咨询，原因是 CTO 搭建和运维的系统是业务部门的生命线，后台系统是 CFO 工作必须依赖的心脏。

我们曾经指出 CTO 的组织（架构、工程、运维、基础服务和质量保证团队）全部都是愿景的支持者，其中一个或几个部门也可能是咨询者。CTO 的技术背景越差，就越需要依赖其团队来制订可扩展性的愿景。在表 2-1 中，我们假设 CTO 有很丰富的技术经验，但是实际情况并不总是这样。CTO 可能也需要从外部引入资源协助，以确定可扩展性的愿景和计划。这种外部的帮助可以是顾问服务公司，或者技术顾问和管制委员会，他们提供与公司董事会一样的技术管制和监督建议。

最后指出需要把可扩展性愿景知会董事会。这或许是董事会的一个会议纪要，或者是围绕着现有平台能支撑什么规模的业务所做的讨论，公司需要什么样的投入来满足来年可扩展性的目标。

矩阵的部分空白已经填好了。和矩阵相关的一个要点是我们已经把任务分割以避免 R 部分的重叠。例如，基础设施团队的责任已经从软件开发和架构设计团队中分割出来。这样保证责任清楚，符

合奖罚分明、职责清晰的思路。然而，这样做的结果是组织向纵向条块化发展，与我们长期的发展方向不符。你要把组织调整成这样吗？要设计出最好的解决方案，就需要不同的团队在一起工作，这一点很重要。矩阵型组织的团队，不但可以避免团队内部存在的围绕功能或组织的责任而产生的独立心态，而且可以从 RASCI 中获益。既要有单一责任的组织，又要确保合作。通过 C 特性的使用来落实 RASCI。

建议你多花点儿时间来完成表 2-1 中的余下部分，直到掌握 RASCI 模型为止。这是一个非常有效的工具，它可以清楚地定义角色和责任，帮助我们去掉重复的工作，减少会引起士气低落的不幸的争斗和发现失踪的任务。

有关角色和责任的最后一个注解：不应该有任何一个团队，把角色和责任作为无法完成工作的障碍。公司里的每个人的主要责任都是为客户和股东创造价值。当然会有这种可能，当员工履行职责时，完成的任务超过了定义好的责任范围。如果能够帮助公司完成使命，那么员工可以，也应该自愿跨越边界去完成工作。重要的是，当这种情况发生时，他们应当和领导一起去找出到底无人负责的地带在哪里，领导应该承诺在未来纠正这些问题。

2.6　结论

定义清晰的角色是领导和经理的责任，不管是独立贡献者还是组织都需要角色的清晰性。本章通过一些案例讲解了怎样清楚地定义角色从而帮助组织实现高可用性的使命。这些例子中提到的组织

是众多组织结构中的一部分，根据个人、组织和他们的角色可以产生很多不同的结构。你的组织现实的解决方案或许和这些结构相比差别很大，角色的定义要和公司的文化和需要一致。当你加强角色清晰度的时候，要注意避免责任的重叠，这会造成无效的努力和价值损毁的冲突。在公司内部，我们通过清晰的角色来确保可扩展性，当然也可以应用在公司业务的其他方面。

我们引进了一套叫做 RASCI 的工具，来协助公司内部的组织定义清晰的角色和责任。你也可以尽情地使用 RASCI 工具，定义自己组织的角色和项目内部的角色。RASCI 的应用可以去除重复的工作，使你的组织更加有效、更有力、可扩展性更好。

关键点

- ▼ 角色清晰对可扩展任务的成功极为关键。
- ▼ 责任重叠会造成资源浪费和价值损毁的冲突。
- ▼ 无人负责形成真空地带无法完成扩展的任务。
- ▼ CEO 是公司里负责扩展性的最高长官。
- ▼ CTO 或者 CIO 是公司里负责技术扩展的最高长官。
- ▼ RASCI 是一个可以帮助减少责任重叠，产生清晰角色的工具，RASCI 以矩阵方式出现。
  - R 代表负责，指决定要做什么事的人，而且负责任务的实施。
  - A 代表批准，指在决策过程中批准任务并验收任务结果的人。
  - S 代表支持，指为完成任务而提供服务的任何人。
  - C 代表咨询，指在决策前和关于任务完成情况接受咨询的人。
  - I 代表知情，指需要通知决策和任务执行结果的人。