OPEN SOURCE REVELATION

开源启示录

第二季 季刊 2015年9月

InfoQ

InfoQ

开源资讯

Amazon EC2容器服务背后的技术
Readme.io创始人谈API文档的未来

开源软件

Log4j版本1生命周期终结
ACAT：来自英特尔的霍金专用语音系统

特别专题

内容为王，文档写作新理念
DevOps在撰写文档中的实践
文档，重中之重

开源经验

开放源码在大型企业中的使用情况
开源领导者应该入乡随俗吗？

开源实践

Esty的开源项目运营经验
Facebook开源的真正原因

开源启示录 第二季

本期主编 郭蕾

流程编辑 丁晓昀

发行人 霍泰稳

联系我们

提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com

卷首语

我之所以能够看得更远，是因为我站在巨人的肩膀上。

——艾萨克 牛顿

这次我们聊聊开源的未来。

谈到开源，多数人的脑海中闪现的一定是开源软件，一旦说起开源软件，那么就可以如数家珍地列出一些软件，如Linux、Firefox、OpenStack、Docker、Hadoop、Spark；列出一些代表人物，如Linus Torvalds，Eric S. Raymond，列出一些代表性的公司，如红帽、Google、Facebook、GitHub、Black Duck；列出一些组织，如Apache基金会、Linux基金会、OSI、Eclipse基金会、Python基金会等。总而言之，头脑中对于开源已经锁定在开源软件，如果是技术爱好者的话，则更进一步去看架构、静态分析代码、运行起来然后再计较其它。

那么其实在这里有一个大大的误区，也就是说开源已经指的不是软件这么一个领域了。尽管开源这个词汇来源于软件领域，标志性的事件即是Eric S. Raymond发起的“开源软件运动”，经过这么多年的发展，其范畴已经远远超越软件本身，一如著名的开源文化、思想的栖息地opensource.com所列出的：开源企业、教育、政府、法律、健康。开源不仅仅是指软件，而是指的一种文化现象：愿意与他人分享，透明的合作方式，为了改进而鼓励失败，期望，甚至鼓励他人也这样做。

未来是难以预测的，也就是说我们无法看到未来的开源是如何的，但是，从现状来讲我们知道，开源已经占有一席之地，是这个世界不可或缺的一个部分，无论是已经融入我们生活各个方面的开源软件，还是其思想所影响的协作、生产方式。我们的世界，既充满合作，又充满冲突；同时也遍布着协作与竞争。难以置信封闭的、“大教堂“式的组织、开发、经营模式亦有不凡的成绩，Apple的市值和流行程度是这方面的铁证；但是开放的、“集市”式的组织、开发流程也不甘落后，Android对于智能移动设备的贡献、Linux成为云计算的基石、Hadoop成为大数据最为流

行的工具就是明证。世界的多元，才是美好的前提。而我们要做的就是让世界变得更加美好，不是吗？当然，能够选择开源作为生产、协作模式，乃至代码、产品的开源，让之传播更加的广泛，让更多的人参与，更加的平等和民主，不是更好吗？

前不久，红帽的现任 CEO 出版了一本书，叫 做《The Open Organization: Igniting Passion and Performance》，书中探讨了开源的文化，如何在一家商业公司中发挥力量，一种扁平、自下而上的、开放的组织是如何运转的。而在 2012 年出版的另外一本书《一的力量》，探讨了各种组织形式的团队及文化。这两本书所列举的例子中都提到了 w.l Gore，这家公司的文化无疑是开源文化的典型：完全的扁平化，自由的组合，没有预先定结论的沟通等，这是家成功的公司，已经经营了半个多世纪，其创新的产品也是我们日常生活中不可或缺的。

将我们的视野收拢一下，抛开世界、未来的宏大叙事，作为一个具体职业的个体，选择开源阵营有未来吗？自由是有代价的，而且通常都很昂贵！开源在未来是否能成为主流，取决于一个个的个体。当然，个体也应先去掂量下自己，问自己几个问题：你能做到独立思考吗？你渴望并愿意追求自由吗？你相信平等吗？你能够做到妥协吗？乐于分享吗？如果上述几个问题大多数的答案为“是”的话，那么就不要犹豫，参与进来，你就是改变世界的一份子。

创新，是未来的唯一动力，而创新的基石就是自由，不限制任何的想法，而这恰是开源的初衷。无论是 Hacker、Geek 精神，创客、工匠精神，都是开源所倡导文化的具体体现。

最后，大家分享一组幻灯片，是著名的开源公司 Black Duck 新发布的 2015 The Futrue of Open Source，用详实的数据回顾了过去几年开源软件的发展情况，并预测 2015 年的开源软件状态。相信可让读者受到启发。

适兕
2015 年 9 月

Amazon EC2容器服务背后的技术

作者 谢丽

Amazon EC2 Container Service (ECS) 是一个高度可扩展的高性能软件容器管理服务，它支持 Docker，使用户可以轻松地在 Amazon EC2 实例集群上运行应用程序。近日，Amazon 首席技术官 Werner Vogels 撰文介绍了 Amazon ECS 的架构。图 1 是 Amazon ECS 包含的基本组件。

Amazon ECS 的核心是集群管理器，这是一个处理集群协调和状态管理任务的后台服务，它的上面是不同的调度器。集群管理和容器调度相互分离，用户可以构建自己的调度器。集群是一个供用户应用程序使用的计算资源池，而所谓的资源是指由容器划分的 Amazon EC2 实例的 CPU、内存和网络资源。Amazon ECS 通过运行在每个实例上的 Amazon ECS 容器代理协调集群。该代理允许 Amazon ECS 与 EC2 实例通信，并在用户或调度器请求时启动、停止和监控容器。它是用 Go 编写的，在 GitHub 上遵循 Apache 许可协议开源。

为了协调集群，需要一个有关集群状态的唯一信息源，提供诸如集群包含的 EC2 实例、运行在实例上的任务、组成任务的容器以及可用资源或已占用的资源这样的信息。这样，才能成功地启停容器。为此，他们将状态存储在一个键 / 值存储中。在任何现代集群管理中，键 / 值存储都是一个核心。而且，为了实现持久性和高可用性，预防网络分区或硬件故障，该键 / 值存储需要采用分布式部署。但这又带来一个问题，就是数据一致性很难保证，并发修改也很难处理。这就需要有一种并发控制机制来确保多个状态修改不会冲突。

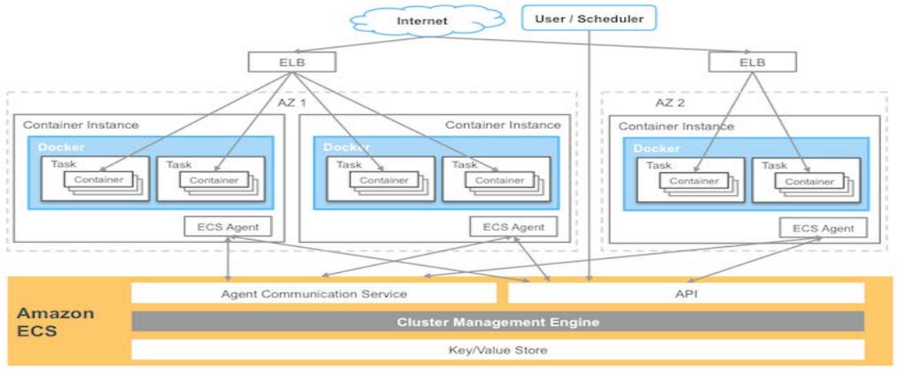


图 1

为了实现并发控制，他们在实现 Amazon ECS 时使用了 Amazon 的其中一个核心分布式系统组件：一个基于 Paxos 算法以事务日志为基础的数据存储。该组件记录了每个数据条目的每次修改。每次写入操作都会作为日志中的一个事务提交，并且有一个特定的有顺序的 ID。数据存储中的当前值是根据日志记录所做的所有事务操作的总和。它允许 Amazon ECS 采用乐观并发的方式存储集群状态信息，在一个共享数据不断变化的环境中，这是非常合适的。

有了键 / 值存储，就可以协调集群了。而为了使用户能够利用 Amazon ECS 的状态管理功能，他们通过一组 API 开放了 Amazon ECS 集群管理器。用户可以通过它们以一种结构化的方式访问存储在键 / 值存储中的所有集群的状态信息。这组 API 成为用户在 Amazon ECS 上构建自己的解决方案的基础。Vogels 举了两个例子。

一个是自创建第一天起就托管在 AWS 上的免费叫车应用 Hailo。在过去的几年里，该应用从一个运行在单个 AWS 区域中的单体应用程序演化成为一个运行在多个区域中的基于微服务的架构。起初，每个微服务运行在一个实例集群上。但实例为静态分区，导致每个分区的资源利用率都不高。为此，他们决定基于服务优先级和其它指标在一个弹性资源池上调度容器。他们选择了 Amazon ECS，因为后者通过 API 完全暴露了集群状态，使他们可以使用满足特定应用需求的逻辑构建一个自定义的调度器。

另一个是教育类通讯软件 Remind。它起初是一个运行在 Heroku 上的大型单体应用。但随着用户数的增长，他们希望具备水平扩展的能力。因此，大约在 2014 年底，其工程团队开始探索使用容器迁移到微服务架构。他们希望在 AWS 上构建一个兼容 Heroku API 的 PaaS（平台即服务）。为了管理集群和容器编排，他们首先考察了一些开源解决方案，如 CoreOS 和 Kubernetes。但考虑到团队规模较小，他们没有时间管理集群基础设施及保持集群高可用。经过简单的评估之后，他们决定在 Amazon ECS 上构建他们的 PaaS。这样，工程团队就可以专注于应用开发和部署。在 6 月份的时候，Remind 开源了他们的 PaaS 解决方案“Empire”。在接下来的几个月中，他们将把核心基础设施的 90% 迁移到 Empire 上。

总之，Amazon ECS 的架构提供了一种高可扩展、高可用、低延迟的容器管理服务。它允许以乐观并发的方式访问共享的集群状态信息，并通过 API 赋予用户创建自定义容器管理解决方案的能力。另外，Vogels 还提到，集群中实例的数量并不会对 Amazon ECS 的延迟产生明显的影响。

感兴趣的读者可以点击这里查看过去一年来 Amazon ECS 增加的特性。

Readme.io创始人谈API文档的未来

作者 Jerome Louvel 译者 陆志伟

那些包含着相对较新工具的文档，比如 @Asciidoctor 和 Cyrille Martraire 的领域驱动设计启示书《动态文档》，是软件开发过程中被忽略的最大领域之一，如今终于得到了大家的些许关注。对于一个 API 文档来说，其被认为是至关重要的。Gregory Koberger 正在开发一套系统，可以让开发者文档与 API 和 API 仪表盘更直接地对接。

最近 InfoQ 采访了 Readme.io 的首席执行官 Gregory，为了实现他对于 API 文档的愿景，其创建了这家公司。

InfoQ：您能描述一下创建 Readme.io 背后的故事吗，以及它是如何演化的？

GK: 当然可以，首先我是一名程序员及设计师，并且对开发工具感兴趣，因为在那片领域有着一套独特的设计挑战魅力。

我曾经花费了大量的时间用来写作和阅读文档，但是我认为应该有更好的方式实现它。大约五年前，我下定决心我要潜心专研这片领域，但是刚开始的时候我度过了一段艰难的时期。谢天谢地的是，像 Stripe 和 Twilio 这样的公司已经向人们展示了文档的重要性。大约一年前，我们成立了这家公司。

公司运行的非常好。我们也确实发现，一份更优秀的文档能够在某些产品和问题上发挥好的作用。我们正走向希望人们开始改变对文档的原有看法这一步。它不应该是一成不变的。它应该随着阅读它的群体和阅读群体的知识量而改变。

InfoQ：您对 Readme.io 的愿景是什么？

GK： API 由三部分组成：文档、仪表板（用来生成开发者密匙等）和 API 本身。

我想着手把它们整合到一起。API 了解代码和数据。仪表板了解用户。文档习惯上仍是一成不变，对它们一无所知。

比如，想象一下，如果文档知道用户语言，并且可以直接显示代码片段，或者想象如果文档知道用户犯了某个具体的错误，并能够帮他解决这个错误。

InfoQ：当需要记录 API 时，您的 API 团队主要面临的挑战是什么？

GK： 一个大问题是，记录 API 文档不是一个优先事项。人们包括我自己都非常不善于记录某个 API。很难回到从前，猜测新人在使用你的 API 时需要了解哪些东西。并且很难永远保持文档是最新可用的。

InfoQ：Readme.io 是如何帮助人们的？它的主要特点有哪些？

GK： 目前，我们主要专注于文档。就像 WordPress 专注于代码和 API 一样。我们让您在为客户提供开发体验时变得更加容易。

Readme 已经能够支持表单、登录页面、教程等等。我们让人们在在网页上正确地使用 API，进行变更，允许他们看看会发生什么。我们同样能够从 GitHub 上自动同步 API 文档并友好的展示出来。

InfoQ：Readme.io 与其它开源替代方案相比如何，比如 Swagger UI 或 Slate ？

GK： Readme.io 有一个优势是公司的所有人都可以参与其中，不仅仅是开发者。很快我们将支持 Swagger，这样我们就可以替换 Swagger UI。同样我们认为，社区也应该参与到文档的记录过程中。因此，我们有一些建议性的编辑器：公司内部或者外部人员可以通过一个友好的拖放编辑器提交更改意见。

InfoQ：Readme.io 创建的文档可以部署到某个现有的开发者门户网站吗？

GK: 我们的目标是成为一个开发者中心。目前，你可以在一个现有的中心使用它。在未来，我们希望将文档，仪表板和支持整合到一块。当它们都在同一个地方时，它们真的会一起运行的很好。

InfoQ：你们支持 API 定义语言吗？比如 Swagger，RAML 和 API Blueprint。

GK： 是的。目前我们用一种叫做 APIdoc 的东西。不久我们将会发布对 Swagger，RAML 和 API Blueprint 的支持。

与编写段落文本相比，通过语义上记录你的 API，你可以创造更多的价值。因为你可以着手给用户定制生成 SDKs 或者代码示例。

现在，我们仅限于导入这些语言，但是，我们认为让人们导出语言同样的重要。Readme.io 很幸运能够站在开发体验中心的舞台上，我们真的想利用这次机会成为不同 API 公司的交流中心。

InfoQ：你们是如何支持各种 API 开发工作流的，是代码优先还是 API 优先？

GK： 通常，API 仅仅是以复制主网站构建的方式编写。那是一个很糟糕的事情，因为很多时候，你用 API 做一些网站不能做的事情。

所以，我认为将 API 看做网站的一个独立的实体是很重要的，因为你希望它足够的灵活，能够做一些网站不能做的事情。

InfoQ：您认为哪种方式能最好的描述 API，并且能够与其实现的演化相同步？

GK：现在有很多种方法可以描述 API，比如 Swagger，RAML 和 API Blueprint。我们选择 APIdoc，是因为这种文档非常接近代码本身。它类似于 Javadoc，这种文档是对代码的一种注释，而不是一个单独的文件。我们可以从 GitHub 上自动同步。

InfoQ：在一个典型的 API 团队中，谁应该负责记录 API？

GK：每一个人都应该。习惯上来讲，只有开发者做记录。但是，API 对业务、市场营销和产品管理团队同样非常重要。我们希望实现每个人从开发者到 CEO 都能够更新与他们相关联的文档。

我们认为社区对记录文档拥有难以置信的重要意义，因为他们有其他人所没有的疑问和用例。

InfoQ：从您的观点来看，API 文档和操作的 API 管理是否应该分开处理？

GK：我真的很喜欢这个 API 的生态系统。现在有很多公司在做具体事情，并且做的很好。而对于较大的 API 管理公司，什么事都想自己做 但大部分做的很差。

我认为 API 管理和文档应该分开，但是，它们应该被更紧密的集成起来。

InfoQ：请问 Readme.io 下一版本的推出时间和新功能有哪些？

GK：我们正在做一个重大的重新设计工作，大约需要一个月左右的时间。除了看上去更加美观，我们将为参考指南和示例设置单独的版块。

对于文档编写人员，事情的发展和文档中应该包含哪些内容变的更加明显。对于终端用户，将会更容易知道从哪里获取他们想要的信息。

因为我们想着手自动化记录您 API 更多的过程，所以像 Swagger 导入之类的事变的非常重要。

大部分开发人员都要花费时间阅读或者编写文档。它是我们与代码库或 API 进行交互的界面，而且我真的很高兴 Readme 正变的有能力改变人们以往的方式。

你可以尝试免费版的 Readme.io，并且它保留了免费的开源项目。对于专属软件，企业在 Readme.io 子领域提供了一个基本包，包括 3 个文档版本和一个管理员用户，花费 $14/ 月。开发者中心版本允许您使用自己的域名，并且支持 10 个管理员用户，花费 $59 / 月。

查看英文原文：Interview with Readme.io Founder on the Future of API Documentation

Log4j版本1生命周期终结

作者 Abraham 译者 金灵杰

Apache 宣布 Log4j 版本 1 生命周期终结。虽然 Log4j 版本 2 在 2014 年 7 月已经发布，版本 1 仍然维护到 2015 年 8 月初。新版本是一个完全重写的日志库，解决了许多版本 1 的问题，达到了前所未有的性能。Apache 已经为简化升级做出了努力，但是高级用户可能需要做一定的迁移工作。

根据 Apache 的报告，最早发布于 1999 年的日志框架 Log4j 版本 1 有许多架构上的问题和发布过程中的不足，这些问题导致开发起来相当困难。这促使一些维护 Log4j 的社区开发者放弃了这个框架，转投其他类似的项目，如 Logback，这些项目也鼓动开发者这样做。为此，Apache 决定从头开始编写 Log4j 版本 2，克服第一个版本的不足之处，恢复部分社区基础。

尽管 Log4j 版本 2 有如此多优势，到目前为止它的使用率增长很慢。根据 maven 中央仓库统计，在写这篇文章的时候，使用 Log4j 版本 2 的构件有大约 350 个，而使用版本 1 的有将近 6000 个。相比之下，使用 Logback 的构件有超过 5000 个。

为了克服这个问题，Apache 试图让升级 Log4j 版本 2 过程尽可能的简单。对于通过类似 SLF4J 等日志门面使用 Log4j 的场景，升级只需要将绑定的 jar 文件从 slf4j-log4j12 替换成 log4j-slf4j-impl-2.0，移除所有 Log4j 版本 1 的引用，添加版本 2 的实现 jar 文件。对于直接使用 Log4j 的场景，用户需要参考 Apache 的迁移指南，迁移指南提供了两种方案：所有调用都转换成新的 API，或者使用桥

接 jar 文件，它将会捕捉所有使用 Log4j 版本 1 基础设施的调用，将它们转发到 Log4j 版本 2。

不幸的是，这个桥接文件针对用户代码中有自定义追加器（Appender）、过滤器（Filter）等内容将无法工作，因为这些在 Log4j 版本 2 中的工作方式有所不同。例如在 Log4j 版本 1 中，自定义一个追加器必须要扩展 AppenderSkeleton 类，实现和继承其中的方法。而在 Log4j 版本 2 中，自定义追加器以插件形式创建，并需要注册。考虑到不得不进行转换工作，在这种场景下完全迁移到新版本可能是一个最好的选择。

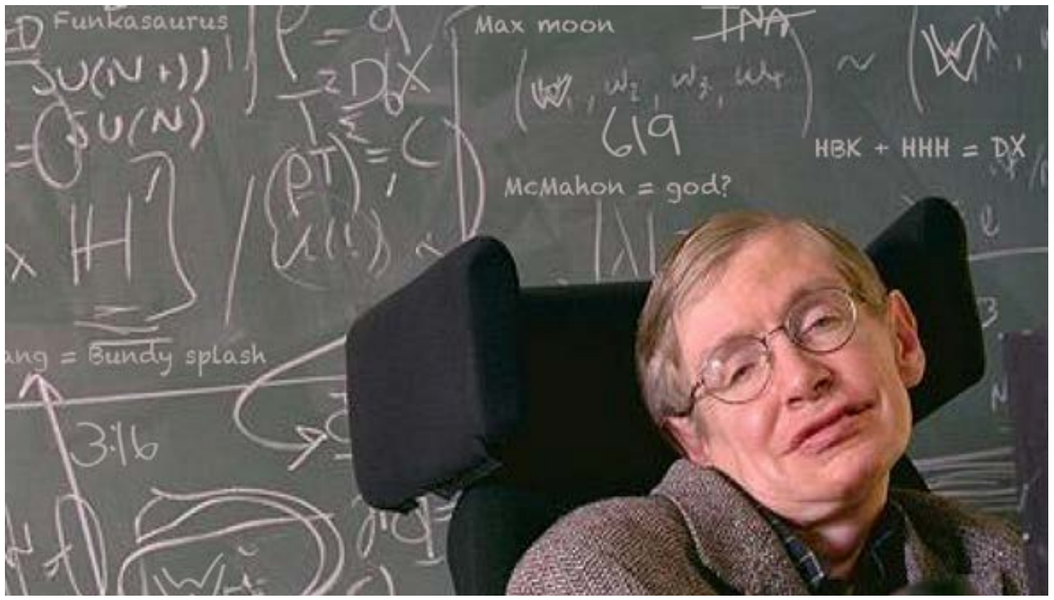
另一方面，警惕传递依赖的用户在添加 Log4j 版本 2 依赖的时候可能会惊讶。一些用户报告说，新版本的日志框架可能会引入超过 30 个直接依赖。然而进一步检查发现，大部分依赖不是测试范围的，就是可选的。这意味着实际在生产环境添加的直接依赖不会超过两个。

更简单的替代品

只有非常简单日志记录需求的用户不必为选择不同的日志框架或者处理升级犯愁：从 Java 4（1.4）开始，Java 包含一个日志记录工具，作为 java.util.logging 包中的一部分。该工具的效率没有像 Log4j 或 Logback 这些库那样高，在功能方面也比较落后。但是由于该工具被包含在标准 OpenJDK 中，使得它在许多场景下成为一个合适的选择。

查看英文原文： Log4j Version 1 Reaches End of Life

ACAT：来自英特尔的霍金专用语音系统



作者 张天雷

据报道，过去 20 年来，英特尔一直在为著名物理学家斯蒂芬·霍金提供计算机方面的帮助，并专门为其研发了辅助语音系统，帮助其与外界交流。为了帮助更多的残障人士，英特尔近日开源了该系统——ACAT。接下来，本文对该 ACAT 系统进行简要介绍。

近年来，英特尔实验室一直在进行面向残障人士的工具研发。这些工具所组成的集合就是 ACAT（Assistive Context-Aware Toolkit，辅助语境觉知工具包）。ACAT 主要利用摄像头追踪用户的面部动作，将其转化为计算机指令。以霍金为例，ACAT 捕捉的就是其颊肌的抖动。根据需要，ACAT 还可以支持其他输入设备。目前，该系统包括上下文菜单和多种快捷键，可以方便用户说话、搜索和电子邮件。而且，ACAT 的用户界面还提供了静音按钮，可以让用户适时的关闭语音合成器。

由于该平台为英特尔研发，其运行环境为 Windows 平台的 PC，并不支持苹果等操作系统。系统版本需要为 XP 或者更高级的 Vista、Win7、Win10 等。此外，ACAT 系统还需要安装简单的摄像头。英特尔的首席工程师 Lama Nachman 表示，公司一直在努力尝试不同的传感器，并与病人合作进行测试。之前，Intel 已经利用近程传感器、加速度传感器和 Intel 的实感 3D 摄像头进行了测试。

Nachman 透露，英特尔开源 ACAT 的目的在于使得广大相关领域的开发人员可以在其基础上方便进行开发，从而把根专业的东西做的更好。英特尔的设想是把从事传感器、用户接口、上下文识别等方面的开发人员的精力更多的放置到专业领域的完善，而不是底层平台的设计。这样，通过将 ACAT 软件开源，英特尔希望相关的工程师、开发者和研究人员等能够将以该技术为基础，开发新的技术和功能，帮助运动神经元疾病患者和无法使用正常计算机界面的人群，提高沟通能力。

目前，英特尔已经把 ACAT 发送给一些用户，并向其讲解了安装和使用 ACAT 的步骤。用户也可以通过 GitHub 下载源代码，参考用户指南即可轻松上手。此外，英特尔也与一些病人合作进行相关测试。未来，英特尔将会与大学合作进行 ACAT 的测试。

Esty的开源项目运营经验

作者 曹知渊

在 Etsy（译者注：一家以交易手工制品和旧物品为主的 P2P 电商），我们是开源的忠实粉丝。在互联网上，无数的人解决了无数个问题，并使用开源许可证发布了他们的代码，没有这些人，就没有 Etsy。我们在 Linux 上运行 Apache 网页服务器，来保持 etsy.com 的运作，我们服务端的代码多数是用 PHP 写的，我们使用 MySQL 来保存数据。为了保持与时俱进，我们使用 Ganglia 和 Graphite 生成的可视化数据来度量我们的系统状态，同时使用 Nagios 来监视系统的稳定性。当然这只是一些主要的例子，我们技术栈里面的每个边边角角，你都能发现开源代码的身影。

在 Etsy，“慷慨精神（Generosity of Spirit）”是每个人工作的一部分，它的意思是要给行业以回馈。对工程师来说，这意味着至少每年一次，或努力在某个会议上做一次演讲，或在本博客上写一篇文章，或给开源项目做点贡献。当我们解决了一个问题时，如果我们觉得解决方案或许对他人有用，我们很乐意把它回馈给开源社区。

维护与分化

这个问题一直困扰着我们公司的很多开源项目，因为源源不断的贡献同时来自于我们的工程师和开源社区。我们从不羞于开源我们技术栈的核心部分。我们一直在公开开发我们的部署系统、统计数据收集器、团队即时管理工具以及代码搜索工具。我们甚至开源了我们的原子部署系统的关键部分。开源确实很有好处，我们广泛地收到来自社区的新功能和 bug 修复，我们的软件变得越来越成熟和稳定。

我们开源的项目越来越多，当我们想快速搞定某些新功能的时候，显然很有理由为项目创建一个内部的分支。这些建立内部分支的项目，很快就和它们的开源版本发生了分化。这意味着，维护好这些项目的工作量会翻倍。内部的任何修复或者新功能，都要在开源版本中重新做一遍，反之亦然。在一个商业公司中，内部版本的优先级往往高于开源版本。看看我们的 GitHub 主页，很难搞清楚哪些项目还在频繁维护中的，甚至我们自己的工程师也搞不清楚。

所以我们的结局就是，手上捂着一大堆“年久失修”的项目。开源社区的人花时间报告了一个 bug，却等不到任何答复，有时候甚至几年都等不到，这使潜在的开源用户心寒。没人能说得清楚，某个开源项目是一个持续维护的软件，还是仅仅是验证完某个想法后就被丢弃了。

向前走

我们想把开源社区建设得更好，因为我们从现有的开源软件中受益良多。为了使我们的开源项目的状态更清晰，我们对此来了一次“春季大扫除”。我们的开源项目将会被清晰地标注为：维护中、停止维护、已归档。

维护中

维护中是默认的状态，所以不会特别标注。维护中，说明我们要么内部就在使用开源版本，要么正在把内部版本的修改同步到开源版本中。我们已经对我们的部署工具完成了这项工作。我们正在处理所有维护中的项目：对 pull request 进行合并或给出我们的看法，答复问题报告，以及添加新功能。

停止维护

我们也有一些项目已经几年没有公开更新过了。通常这是因为虽然我们想在内部也使用开源版本，但我们不能拖慢开发周期，我们难以找到两全的办法。但是，代码还是在那儿，依然可以视作是对某种创意的展示，诠释我们如何解决某个问题。或者，这原本是一个研究项目，但我们放弃了它，因为长久以来它都无法解决我们的问题，但是我们仍然希望展示我们曾经做过的尝试。那些项目就保持原样放在那儿，很有可能不会再有更新。我们会关闭问题报告，不再接受 pull request，并且会在 README 中清楚地说明此项目仅作展示之用。

已归档

我们开源的项目中也有一批是我们在特定时期用过的，但是目前已经不再使用。很可能我们对某个问题找到了更好的解决方案，或者是老方案从长期来看并不能真正解决问题。在此类情况下，我们会往主分支上提交一个 commit 来删除所有的代码，仅留下 README 用于描述项目及其状态。README 会指向源代码删除前的最后一个 commit。这样代码不会真正消失，但项目的状态却明白无误。此类项目同样会关闭问题报告和 pull request。

除了这些归档的项目外，我们也在某些时间

fork 了别人项目，里面有些我们也不再维护，我们也在着手清理这些项目。

结束语

在过去几年中，我们在维护开源项目方面学到了很多。我们想分享的最大教训就是，一定要在内部直接使用软件的开源版本，这很重要，有可能其他开源开发者也想使用此软件，这可以给他们带来更好的体验。我们一直在努力学习，力求把每件事情做得更好。如果你在等我们对某个问题做出回应，或合入某个 pull request，希望本文能帮助你深入理解当下在发生什么，为什么等个回复这么久，并且我们希望新的项目分类方式能帮你梳理清楚我们开源项目的状态。我们希望成为开源世界的典范，我们希望以一种众人受益的方式来给予回馈。

给项目做一次春季大扫除总是一件好事，即便当下已是夏季。

你可以在 Twitter 上关注 Jared 和 Daniel。

https://codeascraft.com/2015/07/09/open-source-spring-cleaning/

本文由作者 Daniel Schauenberg 发表在 codeascraft.com 上：Open Source Spring Cleaning。经授权，在 InfoQ 中文站翻译共享。本文在 Creative Commons Attribution-NonCommercial-NoDerivs 3.0 United States License 许可证下发布。

作者 曹知渊

在 OSCON 的第三天，来自 Facebook 的 James Pearce 带来了一场主题演讲，我也有幸在场。

Pearce 解释了 Facebook 为何大规模开源其软件。他告诉我们，Facebook 每个月都会发布数个开源项目，并且有数百个工程师会持续地支持这些项目——他们参与全世界的各个开源社区，改进软件的体验。

但是，Facebook 究竟为何要使用、支持和发布开源项目？这个问题比探究 Faceboot 如何做开源更有意思。

Pearce 大可以带我们回顾一下历史性的那一天，Mark Zuckerberg 坐在宿舍房间里，选择了 LAMP 作为 Facebook 的基础；他大可以跟我们说说 Facebook 的黑客社区；他也大可以侃侃 Facebook 所感受到的社会责任，但他没有这么做。这些都是真的，但都不是这家公司走上开源之路的真正原因。

真正原因到底是什么？对公司有利，仅此而已，Pearce 说道。

他的这个论断，一般人可能难以得其要领——我整个教学生涯也一直为这个问题所困扰——但 Pearce 提出了以下几个很棒的理由来解释从事开源的原因：

- 共享 Facebook 的代码（通常是软件“栈”，偶尔也包括硬件设计）促进了这个世界的创新。这些代码帮助他人更快地开发软件。因为 Facebook 不是一家软件公司，所以它在开源过程中没有面临竞争对手的威胁，相反，开源带来的价值在逐渐显现。用户使用 Facebook 的开源代码可以更快地构建应用，而他们也乐于回馈代码，使 Facebook 从中受益。

- 拥抱开源，意味着 Facebook 必须一开始就写出更优秀的软件。如果他们知道某个

软件从诞生起就要公开，那就必须要好好做，提高可用性和可靠性，因为将来外面的人都会用它。这种压力也会给公司内部带来更多的价值。

- 开源带来了共享挑战的机会。开源项目面临的难题会吸引一些外部的优秀人员，而结果是，他们也带动了公司内部人员的能力提升。每天 Facebook 都承载了超过一亿人的沟通互联，何以能做到？唯有开源的力量。

Facebook 的 GitHub 账户有 274 个代码库，收到了 39000 次 fork、提交了 79000 个 commit，并且拥有 242000 个关注者。他们开源的这些项目可不是黑客聚会上随便想出来的点子，或者实习生练手的代码。这些工具都是 Facebook 在产品中使用的。Facebook 只会拿他们真正在用的代码来开源——这样，人们才会相信 Facebook 提供的代码有价值，并且会得到持续的维护。仔细审视这些产品，Facebook 的工作方式就会在你眼前呈现。

Facebook 的开源办公室只有两个员工，所以开源项目团队的工作必须高效，他们需要自己去收集数据来判断项目的当前状态。他们大量使用 GitHub 的 API 去获取尽可能多的数据，每分钟都在获取。然后他们把这些数据收集起来实时地共享，并且每个月会做个月报。这给工作带来了些许刺激的趣味，开发人员们可以互相竞赛，看谁的项目表现更好。尽管被成功的光环笼罩，Facebook 总是力求做得更好。这就是为什么 Facebook 要加入 ToDo，这就是为什么 Facebook 要参加今年的 OSCON。

本文由作者 Nicole C. Engard 发表在 Opensource.com 上：The real reason Facebook does open source。经授权，在 InfoQ 中文站翻译共享。本文在 Creative Commons BY-SA 4.0 许可证下发布。

开放源码在大型企业中的使用情况

作者 Abel Avram 译者 陆志伟

很明显，现如今开放源码在许多组织中被大量使用，并且扮演着一个重要的角色，但是开放源码在大型企业中的使用情况如何呢？这个问题在最近一项被称作“开源时代”的研究中被提出，这项研究由 Oxford Economics 和 WIPRO 发起。Oxford Economics 是一家与牛津大学合作，致力于经济预测与定量分析的企业，WIPRO 则是一家提供 IT 咨询与全外包服务的公司。

这项研究调查了 100 名来自美国、欧洲和亚太地区的 C 级别的高管人员，这些高管人员分别在金融服务、零售、消费品、医疗、生命科学和政府部门工作，直接向执行委员会汇报。在过去一年的财政年度，这些公司分别至少有 $1B 的税收（revenue），其中 5% 有超过 $20B 的税收。

当谈到开源软件（OSS）在大型企业的实际使用率时，该报告指出，只有 21% 的大型企业在整个企业使用，25% 的大型企业已经在业务单元部署。其它 54% 的大型企业仍在计划阶段（21%），或者在互联网相关程序中使用（13%），又或者开展一个试点项目来评估它（20%），如图 1 所描述。

关于开源软件对他们各自行业的感观影响，报告指出，55% 认为 OSS 是未来竞争优势的关键，但是只有 11% 的认为，现如今 OSS 已经在他们行业产生了积极的影响。但是，当评估三年后 OSS 的作用时，数字发生了翻天覆地的变化：61% 认为 OSS 将会带来竞争优势，62% 认为 OSS 将会对他们行业产生积极的影响（见图 2）。

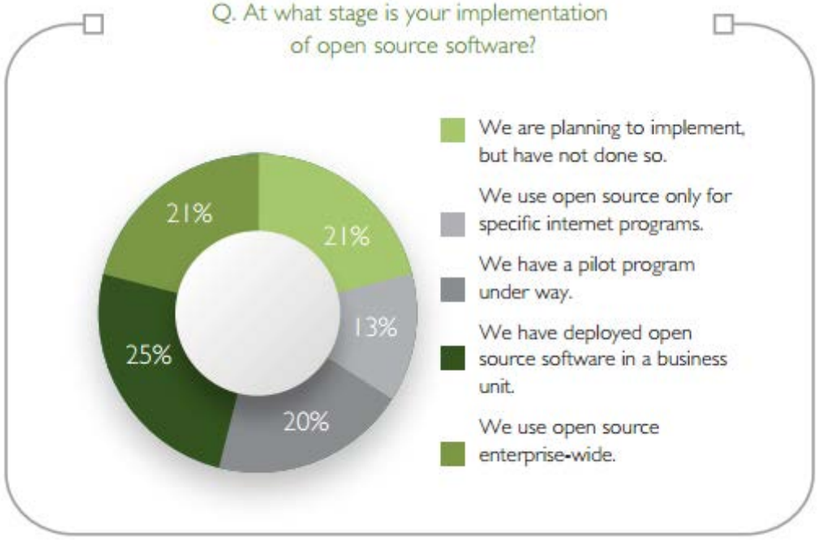


图 1

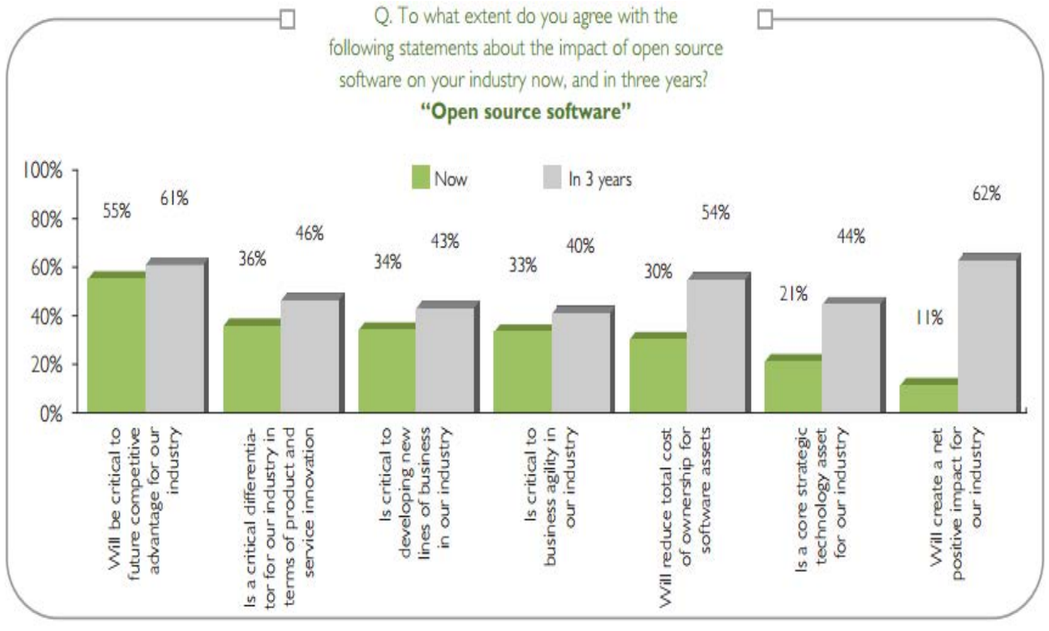


图 2

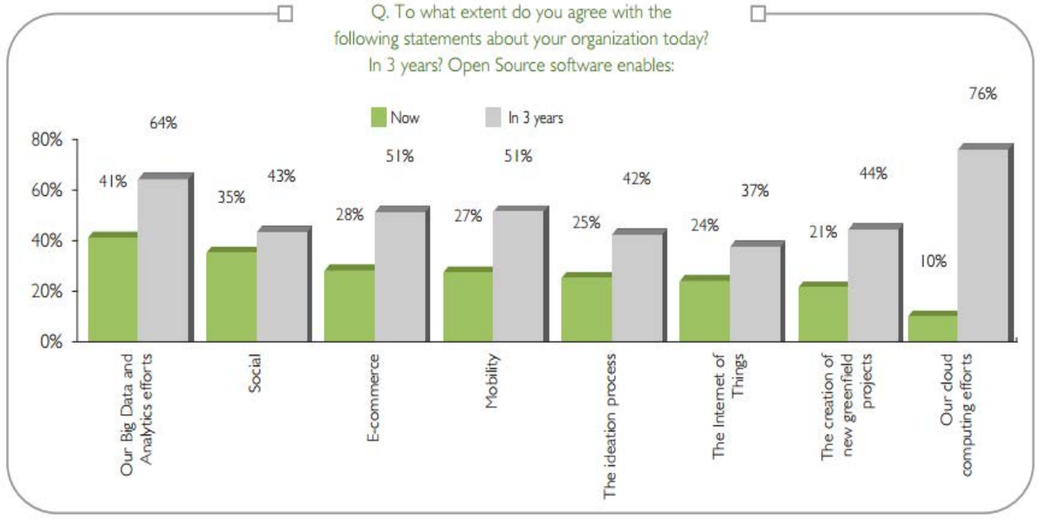


图 3

今天，大多数企业使用 OSS 进行大数据处理和分析项目（41%），而只有 10% 的企业用于云计算。但是，他们认为三年后现状将会发生改变。64% 希望用 OSS 来处理大数据和分析，76% 预测将用于跟云有关的活动。

图 3 中的数据表明，OSS 在大型企业还不是那么受欢迎，但预计在未来几年内将会有所改变。大型企业发觉难以使用 OSS 最主要的原因之一是：OSS 与他们现有的系统很难集成（75%）。其它原因包括：缺乏 OSS 管理技能（56%），用 OSS 解决问题需要一定的时间（35%），以及关注它的品质（43%）。

当谈到如何面对发展 OSS 所带来的挑战时，（63%）受访者认为发展 OSS 需要重新评估整个生产过程，（61%）员工需要扮演一个新的角色，（47%）他们需要雇佣新员工，（44%）需要获取新技能，并且（44%）将会引起开发文化的变革。（见图 4）

更多报告，比如用“OSS 实现 IT 和商业目标”，“用 OSS 后的最大收益”，“公司是如何采购和实施 OSS 的”等等。我们建议通过图表或者下载一份该报告的副本。

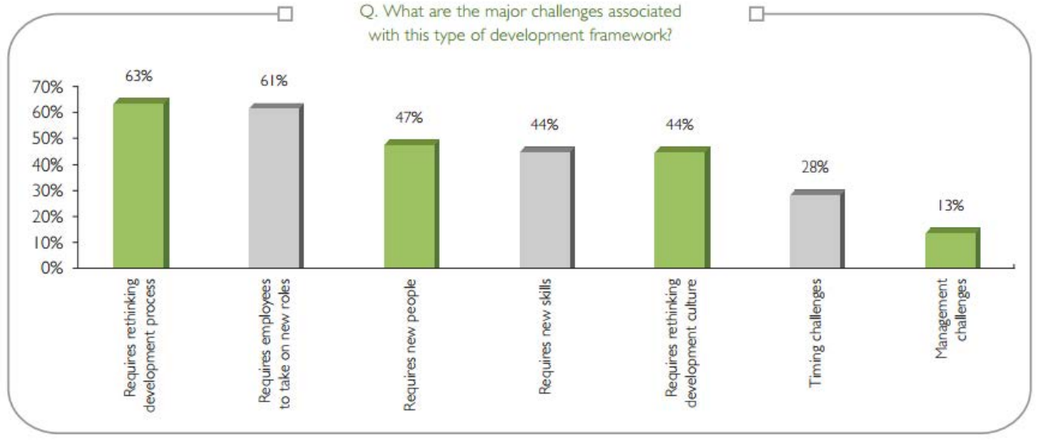


图 4

开源领导者应该入乡随俗吗

作者 曹知渊

人类学家们经常半开玩笑地争论是否要入乡随俗，因为他们要深入丛林去研究各种各样的部落——这种争论意味着，你想去了解一群人，你就要考虑到底是要融入他们的生活方式，还是要保持距离（为了保证研究的科学客观性）。怎么选择取决于你如何设计这项研究工作，但这种选择也决定了你的身份，除非你只想做一个好奇的旁观者。

新型的人际关系网和虚拟社区带来了一个新世界，置身其中的领导者们面临着同样的身份选择。领导者们越来越依赖那些跟他们没有直接关系的人们，这就是所谓“群众的智慧”，而开源运动正给世界带来变革。在此大势下，他们应该融入社区到什么程度呢？是让自己成为关系网中的一员？还是跳出圈子，从外部去管理他们？

关系网、社区和乔伊法则

一开始，这看起来像个伪命题。通常的观念告诉我们，社区关系网很难去管理，因为他们不喜欢被管理。但如今，很多领导者都明确地借助那些非正式的社区关系网来达成战略目标。有些情况下，他们放下身段，入乡随俗，成为社区的一员，用他们的个人影响力和领导力来动员社区的天才“同事”们追随他们一起做项目。在其他一些情况下，他们会站在一边，但他们会直接或间接地激励社区的志愿者们跟他们一起追求共同的事业。不管是哪种情况，都可以说他们（在一定程度上）领导着社区。越来越多的领导者在面对这样的挑战，他们发现十年前比尔乔伊早已极富远见地断言：人与人之间的联系无处不在，天才很多，但组织很松散，在这样一个世界中，“大多数绝顶聪明的人都在为别人工作”。但是，作为一个领导者，你还是要想尽办法把这些天才招募到你麾下。

Red Hat 就是乔伊法则的最好例证。很多 Linux 软件运动中最聪明之人，都没有在为 Red Hat 的 CEO 效力，但公司依靠志愿者组成的社区来达成它的策略——提供极具附加值的产品。这大概就能解释，为什么 Red Hat 的文化能和 Linux 开源运动本身提倡的热爱自由、自我管理的价值观产生共鸣。Linux 运动和这家企业都由社区关系网组成，关系网中的人们极富知识和经验，并且都为 Linux 的成功做出过至关重要的贡献。他们分享工作方法及价值观，按照旧时代的人类学家的说法，他们极具团体意识。

在 Red Hat 是否要入乡随俗

Red Hat 的 CEO Jim Whitehurst 新出了一本书叫《开放型组织（Open Organization）》，这是一本讨论新型领导力的沉思录，不要惊讶，他早早地把是否要“入乡随俗”的问题列为此书的必谈话题。这本书对如何在以社区关系网为中心的世界中发挥领导力做了引人入胜的研究。从 Whitehurst 面试 Red Hat CEO 这份工作那一刻起，他就意识到，想要被录用（并且有所作为），他必须要融入开源文化——这种文化跟他以前在达美航空担任 COO 时的那种“命令加控制”式的文化截然不同。达美航空这种传统行业公司有领导专用车位，有独辟一隅的专用办公室，有根深蒂固的员工等级差异，这一切在 Red Hat 都没有。欢迎来到一个敢为天下先的社区，这里鼓励随时随地讨论，这里不看职位任人唯贤，这里提倡有强烈社区风格的价值观。

Whitehurst 带着狂热的使命感拥抱了这一切。这本书为我们呈现了一则又一则关于入乡随俗的轶闻。书中赞扬了 Red Hat 员工们是如何教会他逆向思维可以产生更好的结果，回顾了一位资深工程师如何说服他收回一个关于软件的重要决定，也反思了那些众人参与的设计项目，是如何给每个参与者带来更多价值的。Jim Whitehurst 的眼镜闪过一丝智慧的光芒，以人为本的社区网络和开源思维是他坚定的信仰：在一个崭新的技术领域，一是要找专家，二是要找与此行业休戚与共者，得到这两类人的帮助，推动变革就会行云流水。

关于“管理授权”的好处，我们已经有无数经验了。但更微妙的问题来了，当一个人作为 CEO 时，你就必须履行“控制”他人的职责，那些人最终也是要向你汇报工作的，在此情况下，如何去掌握授权的度呢？换种说法，一个社区领导者如何避免“完全让动物们自己管理动物园”（借用 Whitehurst 自己的话）？在什么情况下，你应该放弃“入乡随俗”，而以领导的身份介入？如何去做？

尝试不同角色和身份

在这个问题上，Whitehurst 的书要比想象中涉及得少。但他在领导力方面的成长历程，必然经过了频繁的角色切换，有时候他的身份需要他时不时去拍板，而有时候他又是一个几乎不需要做决定的角色。他讲的故事，就是一部两种角色的碰撞史，一会儿讲到公司等级制度中的一员（这个或那个职能的领导，“高级领导小组”之类的），一会儿又讲到能者居之的社区领导力。他有时候称自己为 CEO，有时候自称“一名领导”（注意，只是一名，说明 Red Hat 有很多这样的人），还有些时候他看起来仅是一个不起眼的普通员工。

既想培育社区力量，同时又想领导一家上市公司，不可避免会遇到是否要融入社区的冲突，

《开放型组织》一书对此提出了极有价值的洞见。有时候，为了履行外部的义务（比如对股东或监管机构的义务），Whitehurst 不得不放下社区角色，回到 CEO 的位子上来。这位 CEO 写道，社区“领导人”要建立公司和社区的沟通平台，确保公司对社区的巨大投入能带来规模的增长。社区领导人（而不是其他成员）也要保证工作方面的激烈争论不要演变成人身攻击，或者乱作一团失去控制。他或她必须把开源文化带来的激情和公司的目标有效结合到一起，真正推动公司的成功。CEO（而不是社区成员）必须限制好花在“异想天开”的项目上的“创意时间”，因为它们可能不会产生真正的利润。

领导和界限

Whitehurst 作为一名社区领导（他谦虚地自称“还在学习中”），神经是非常紧张的，给他带来紧张的源头也很明确：虽然这位 CEO 把“开放型组织”定义为内外部皆参与协作性社区的组织，但他也提到内外部社区还是有界限的。即使 Whitehurst 信奉开源文化，即使 Red Hat 只是更广泛的社区关系网（比如 Linux 运动，Whitehurst 对其几乎无影响力）中的一员，他还是要把 Red Hat 这个社区监管起来。内外部社区都要参与——Red Hat 的服务依赖于更广泛的 Linux 运动——但 CEO 最终是要对公司的直接利益相关者负责的：Red Hat 的员工、股东和正式合作伙伴。Red Hat（作为一家公司）最终确实需要更多的传统式监管，而这种监管在开源运动中很难站得住脚。

高空走钢丝

为了公司业绩考虑，Whitehurst 必须处理好公司和社区的两种不同身份。一方面要融入社区，另一方面又要使互相之间的协作朝着公司需要的方向发展，从一个局外人的角度看，Whitehurst 正在寻找两者间的平衡。

这是一种高空走钢丝般的高风险领导行为，需要更多知识和经验。在一个互联网无处不在、开源大行其道的新世界中，协作型社区产品同时会和多个圈子有交集，领导者要学会管理这多个圈子——在“自由”和“责任”间游走。乔伊法则会越来越显现其正确性。领导人们很有必要重新审视自己的思维模式和行为。全世界的社区领导者都希望看到《开放型组织》的第二卷问世。

本文由作者 Brook Manville 发表在 Opensource.com 上：Should open source leaders go native?。经授权，在 InfoQ 中文站翻译共享。本文在 Creative Commons BY-SA 4.0 许可证下发布。

内容为王，文档写作新理念

作者 Mikey Ariel 译者 适兕

大家都认可文档的重要性吧？是不是都希望拥有更好的文档？本文不会再赘述文档为什么重要，因为我相信你最后一定会同意我的观点。文档确实很重要。

言归正传！作为一名在企业级软件界混了差不多快 7 年的技术作者， 我看到软件使用者，软件创作者，和软件组织对技术文档的态度的一个巨大改变。

如果说我早期的职业生涯是被教导尽可能多的去写文档，但是现在人们更加注重文档的质量而不是数量。

关于文档的一点题外话

在我加入红帽之后，我们很快经历了一场巨大的变革，公司将内容服务团队的角色定位变更为面向客户，诸如技术支持、客户经理、客户体验经理等。

这意味着我们需要开一些重要的会议，检查我们所做的一切，因为我们现在是产品的客户体验环节的重要角色。不再是躲在我们工程师背后的角色了，技术作者拥有了正式的地位和声音，同样也意味着更多的责任。

我们开始了一轮密集的调查，如何才能够写出更好的文档。我们得出了结论－虽然很困难－在我们进行内容处理时最重要的问题是：

- 我们如何才能够创作出用户真正去读的有用的文档？
- 我们如何才能够将内容的创作集成到我们已经存在的软件交付流程中？
- 我们如何才能激励社区成员像贡献代码那样有激情的贡献文档？

当然，没有人可以做到与世隔绝，于是，我开始参与一些开源社区当中，也开始参加一些技术活动，我遇到了一些公司和项目也有这种或者类似的问题，这说明了这是一个普遍存在的现象，这也证明了我作为 docs lady 的价值，多么的让人心满意足！

在此系列的三篇文章中，我试图巧妙的使用一些成功的文档解决方案的例子来回答上述这些问题，这些均来自开源社区，以及红帽文档团队。

在第一篇当中，我聚焦在回答第一个问题，讨论下文档的一些理念。然后，在剩下的两篇文章中，我将继续回答关于文档演化的解决技术的问题和社区因素的问题。

内容为王，技术文档新理念

让我们来看下第一个问题：我们如何才能够创造出正确的内容？给那些有真正需要的用户？然后在正确的时间和地点将之交付给他们？

如果你熟悉内容为王的概念的话，你可能非常想知道技术文档如何贯彻这一理论。不是诸如网站设计和文案策划吗？ 再也不要这么狭窄的理解了。如果你将文档视为单独的产品或者是作为软件的交付需求的话，我们必须规划、创建、交付和管理我们的内容，和其它可交付的内容同等的对待，而这意味这必须深入思考和规划。

在这点，我为大家推荐 Rich Bowen，他发表在这里 opensource.com 的文章 RTFM？ How to write a manual worth readming? ，其开启了文档菜单栏目。他的文章涉及到很多关于文档的关键因素的深入思考，我根据他的理论，扩展了几点。

先问，然后再写下来

内容为王的核心是完全聚焦在用户关心的内容上，即读者。当我们去为特性、组件或用例去写文档的时候，我们必须考虑到我们的读者。无所谓原创或者重复别人的劳动，下面我使用著名的 5W 来解释这一思考过程。

谁是我的读者

正如 Bowen 所指出的，创建用户的内容的第一步就是找出我为谁而写？我是写给最终用户的？还是开发者？还是业务人员？还是系统管理员？我的读者是初学入门者还是传说中的高手？

如果能够确定我的读者的角色（gnome 帮助页面是一个很好的例子）将对构建对用户有利的内容奠定基础。现在你可以基于确定的用户已知道一些什么，如何思考，怎么样的内容才能更加的吸引他们的假设的前提下，问一些其他问题了。

我的读者们想要知道什么

此问题所揭示的不仅是读者要什么类型的信息，还有我交付给用户的格式。举例来说，当我面临为 OpenShift 企业版写 Jboss Fuse 的部署文档时，我做的就是以最少的内容为新用户提供入门手册，而不是以庞杂而大量的信息或者是很 cool 的内容来轰炸他们。

相反的情况是，如果你为开发者提供命令行工具或某个库，一个所有命令的完整参考和工具提供者的所有属性则是产品使用的自然延伸。哦，或许一个 man 页是提供此类内容的最好办法；没有什么比运行 man [COMMAND] 更加的令人愉悦，且可以找到非常好的属性描述和实例。

我的读者做什么的时候需要这些内容

此问题非常的接近接下来的“何地”的问题，而且有时候答案会覆盖它们两者，但是“何时”还是单独的拿出来表述了，答案可以表明用户在什么时间使用该软件，或者说明读者需要什么样的内容。

这意味着如果你是为开源项目如 Arch Linux 的骨灰级玩家创建内容的话，你可以猜的到这些读者已经在尝试解决它们的问题了，又或者是独立的实现它们的目的。所以他们看到大篇的文档一定很烦，他们有明确的要搜索的内容。这样的话，维基百科是最好的实现途径，wiki 强大的搜索功能使他们事半功倍，而不是浏览全部的内容，不是谁都有时间去浏览大量的内容的。

加分项：在此你可以思考多长时间 release 和还有，错误信息。当发生问题时，清晰信息可以节省用户去查看文档库的时间。所以没有理由不为用户提供在发生问题时交付清晰的信息的内容。

我的读者为什么需要这些内容

此问题是所有问题中最为棘手而难以回答的。它将你的文档内容置于 WIIFM（对我来说它内在是什么？）测试的审查之下：你为什么要写这些文档？你解决了读者的那些痛点？为什么他们要在乎你写了什么？我知道，太多刁钻的问题需要回答了。

让我们回到本文的开始，我问读者的一个问题——首先你得同意文档非常的重要——否则，我就不会花费时间和精力来写这篇文章。记住，你阅读本文是因为你打算学习如何改进你的文档，如何在你的组织中进行理念的变革，请放心不是你一个人会这样思考，我会试图构清理你的内容。什么信息是在版本发布后读者需要立即就想知道的？什么时候结束早期版本的内容？等等。

我的读者在哪里消化这些内容

还记得我在回答“什么”问题时提到的 man 页吗？这是一个非常典型而现实的例子，这种情况下一定是在终端下。同样的，当你交付一个 IDE 或桌面应用时，嵌入的或基于上下文的帮助会决定用户的体验。对托管在 github 上的项目有一个友好的 README 文件又是多么的重要。

API 参考可以使用代码自动生成工具，给出几个例子，如 Sphinx auto-doc 或 JavaDoc。花时间在可读的代码注释不仅对用户有用，对于 6 个月之后的开发者们也非常多有用。（包括开发者本人，请面对现实吧，少年！）

建本文中可能最有效的方式，并希望我能鼓励你进行到底。

好理论，但是如果……

也许事实上是我上述所描述的方法论对于你的组织或社区掌控文档太过于遥远。我不可能知道所有问题的答案，我能给出的最好的建议就是从小做起，然后观察它如何发展。

所以如果你的公司拥有多个技术写作团队的话，让其中一个团队运行 POC（概念验证）是其中之一，请保持你的同事能够紧跟流程，且记录你所有的发现，以及其他同事的流程，这就是我所建议的最佳实践。

如果你的开源项目缺乏技术写作能手，请到其他团体寻求帮助。协作是开源软件的支柱之一，没有理由不集中我们的资源，创造更好的文档，每个人都可以从中受益。

自从在红帽我们开启了文档的旅程，我们花了很长的时间去理解用户，方才取得了一点实质性的进展，我们甚至还有文档内容战略家，对于文档扮演产品经理的灵魂角色，我们做非常关键的分析和归类内容，这样我们才可以在用户真正去读的文档上下足够的功夫。

当然，理念的变化仅仅只是一个开始，在下一篇文章中，我会试图解决成功的文档的技术层面的内容。

DevOps在撰写文档中的实践

作者 Mikey Ariel 译者 适兕

在上一篇《内容为王，技术文档新理念》中我们围绕技术文档探讨了理念的转变，以及新的、令人振奋的针对用户关心的方法。

我们已经拥有了关于 who、what、when、where，以及 why 的伟大的启示，接下来，我们将要介绍如何才能撰写好的文档。对于我来说，那就是文档的 DevOps。

没错，就是 DevOps。此术语自从诞生就论战不断，就我个人而言，我非常的喜欢论战，我作为 scrum master（已退休）学会了很多，如何实现 DevOps 完全取决于你个人：拿到自己需要的，不要去担心你不会的，直到一切都回到自身的位置。

你个人的 DevOps 宣言

感谢灵活的 DevOps 的定义，有大量的方法来分解和组合这些实践，幸运的是，docsphere 经受的起多处的分解，我们主要的目的是文章而不是代码。

再次，让我们来看下在这个领域里开源社区和红帽文档团队做什么。现在我们来尝试下分解文档的 DevOps 为以下几类：

- 统一的工具链；
- 持续交付；
- 合作。

其中一些实践以及我即将列出的工具都是多年积累下来的，另外一些是围绕 docsphere 不久前才学习到的，让我兴奋的是，作者可以利用所有这些现有的知识，技能和工具，与我们的工程师并行共创快捷和流畅，然后将内容交付我们热心的用户。

同时也意味着为内容做一些非常酷炫的事情！

统一的工具链

从源代码控制，标记语言，发布解析，到持续集成，自动化测试，以及从文字处理到印刷成书之间漫长的等待，还记得文档持续交付吗？它并不完美。

开放源代码技术已经在推动这方面的努力中发挥了关键作用。作为内容创作整体家族成员的创建文档，它需要快速、简明和集成，且我们文档代码使用的发布工具扮演了非常重要的角色。

现在多数的核心文档都可以找到源代码文件，这些源代码文件使用一种或多种我们早已经掌握的标记语言，比如 DocBook XML、Markdown、AsciiDoc 和 reStructuredText。这些文本文件通常使用 Git 来做源代码控制（无论有没有 GitHub 或 GitLab 的帮忙）。大多数的开源社区即使是红帽也使用 Git 来管理他们的源代码。

很多开源项目管理文档都接近于他们管理代码，或者干脆说和代码是一样的管理。其中一些成员热衷于基于项目构建来优化出版流程以及简化内容。举例来说，KDE 社区使用一些灵巧的脚本来将 Wiki 转换为 Docbook 的指南，哪怕你是 Markdown 的忠实拥泵，照样可以使用强大的 Docbook 出版能力。NixOS 使用 Nix 包管理器来给 Nix 文档打包。GitHub 使用 GitHub 来写 GitHub 文档。 都表现都异常强大！

在红帽以及其他大型的企业，众多产品的文档以及写作团队之间的缠斗意味着我们需要一些额外的制衡措施，以确保在所有的开发、写作过程中，我们的文档是健全的。非常感谢，我们有激情四射的、非常有天份的同事，使用很多开源的项目创造性的改进了我们的内部流程，还帮我们调整了很多的工具链，当然，也会尽可能使用我们所支持的开源项目。

下面是我们如何成为部署基于 Jenkins 持续集成的终极者的（我们也持续集成文档！）。每一个上游的提交不仅会检查整个产品文档库，诸如损坏的连接和一些错误，而且可生成 HTML 和 PDF 格式的文件，从而使我们可以在 Jenkinscat 用户界面中直接查看。基于 Publican 的解析器处理 DockBook 和 AsciiDoc 源文件，然后输出统一风格，自动发布到用户门户，基于 GitLab 的合并请求和内容审核。而且你完全拥有了属于自己的秘密武器（工具链）。

在自动化方面，关于 API 文档，对于 JavaDoc、Sphinx auto-doc， 以及 AsciiDoctor 等工具等感谢是无须多讲的。近来，对于文档的单元测试也在风行一时，而不再仅限于代码了。崭露头角的工具如 hemingway 和 emender 等不是为你纠正语法错误的，而是分析你的内容，并且会报告常见的语法错误，如复句、被动语态、动词的错误等。

希望我们将来会看到更多的对于这些工具的贡献！比如它们可以自动完成一些比较繁琐的同行评审任务，比如可以帮助我们维护文档的一致性和简单性。

持续交付

不同于它的工程部分，docsphere 的实现更加注重内容的异步发布，它就意味着脱离软件交付，从而交付文档成为一个独立的流程。
Freedom！ 无论任何时候只要我们有需要就去自由的发布我们的文档，尽我们最大的努力去敏捷，自由度更高的敏捷（有时，要比我们的工程师同事更加的敏捷）。自由的去为产品提交补丁，还可以这么说，为原先发布的内容做进一步的完善。

那些开源项目的贡献者们也许会对我如此的激动窃笑不已，针对文档发布一个补丁和编辑 wiki 一样的简单，但是这在企业的世界里绝对是一个大的改变，当然，这是爱上开源公司的另外一个原因，来自社区的影响非常容易接受且非常的受欢迎。

在红帽，我们所有的文档都在红帽客户门户中作为官方产品发布的对外开放，这就意味着不论什么时候我们修改内容，我们都需要重新构建图书并上传，内容的变化已经是我们工作中很自然大一部分了，当我们计划任务的时候，我们会尝试平衡发布和已发布的内容。当一些人去抢着修复 Heartbleed 或 Poodle 一 induced 会应用到很多的产品以及图书时，此种做法就非常的方便。

我们是如何管理所有这些发布周期同步的了？它取决于具体的产品和团队。我们其中一个工程师团队使用类似敏捷的方法论，所以这样文档团队就相应和他们一起可以敏捷的迭代。其中还有文档团队独立于工程师团队来实施敏捷。其他一些团队甚至都不使用敏捷，而是选择早于发布周期开展内容的任务，当一个特性准备好可以写文档的时候才集中去写。

合作

“没有人能做到与世隔绝”，这是我经常重复的一个句子。这是有充分的理由的：没有合作，就不会有开放源代码项目。除了内容策略须做到以用户为中心之外，当我们创建文档时，我们必须听取相关合作者的声音。
那么，谁会是我们的合作对象了呢？对于一个刚入门的人来说，每个人都是软件交付的参与者。开发者、测试工程师、产品经理、设计师、支持工程师、翻译者，哇塞，我们为了软件的发布交付而走在了一起，还有很多是作为一个产品的团队才合作的，或者说为了提供整个团队的沟通。还有，我们不应该忘记的有用户本身，（这才是意义本身）以及开源社区成员，企业级的客户，以及面对客户的工程师们。

多数的开源社区已经把代码贡献的流程，不同深度的代码审核和社区治理，应用于文档管理中。诸如 Mozilla 开发者文档项目展示了开源软件合作的原则以及内容社区的社区实现，以及 OpenStack 社区文档的重大启示：“文档应该和代码一样对待，都由社区驱动。”

对于开源社区项目的文档合作还有很长的一段路要走（正在进行），关于此方面将会在下一篇文章重点讨论！

在红帽，文档战略现在能够占有一常驻席位；在所有的发布计划中拥有投票权；在产品管理会议中，能够和在发布过程所有其他角色的代表一起共同决策。除了内部的地位稳固之外，我们也将我们对文档的反馈形成了一个闭环，无论是直接来自客户的反馈，还是来自技术客户经理，支持交付经理，和其他面对客户的团队。

此种类型的合作对于我们来说是全新的，但是我非常的有信心，它会帮助我们优化我们的内容战略规划，一旦和我们的持续交付原则所匹配时，将允许我们快速的交付内容给用户，这样可进一步的发展用户成为贡献者，从而进一步增强和修复我们的内容。

接下来做什么

现在我们开始有了好几处的困惑：我们深信我们的经理或项目负责人能够认识到有价值的文档要好过于大而全的文档，而且我们尽可能的利用技术来实现平滑的内容交付，剩下的就是让人们去写一些内容，对不对？

好吧，还有一些依赖因素。当你的工作岗位里以这样或那样的方式包含有“作者”这个词汇时，那么就说的是你。但是你如何驾驭哪些非常不稳定的开源贡献者了呢？他们基本上是以志愿者的身份花时间为项目写代码的，而无需花时间，激励，相信他们的语言能力能写出你想要的所有这些文档？

下一篇文章里，我总结下围绕 docsphere 以及关于上述内容分享我自己的“狂热之旅”。同时，我非常的渴望能够听到读者——你的关于 DevOps 写作的实践的想法和故事。

文档，重中之重

作者 Mikey Ariel 译者 适兕

关于描述社区的建设、激励和教训的文档大家已经着墨了很多内容。到处的文章和博客都是围绕着社区管理的经营和启动作讨论的，如何让社区增长，如何支持社区，甚至有的是告诉你如何不要搞砸一个社区。

正如我在前面两篇文章内容为王和 DevOps 能为你的文档做些什么中所提到的，能够致力于追求质量上乘的文档，不仅仅对于用户有好处，对于贡献者也大有益处。如果好的文档能够帮到用户学习一个软件，那么它也一定能够帮助新的贡献者参与到项目中来，对不对？

哦，对了，当时间非常紧迫的时候有例外。如果项目的参与者很少的话，什么内容为王，什么 DevOps，都帮不了你去写什么文档。当只有少数的开发者已经被到处都是不能正常工作点软件功所累而疲于奔命时，或者是为某些新的特性而兴奋不已，已经进入状态，撸起袖子准备大干一场时。但不知道什么原因，如何进入文档写作状态的教程却无从谈起。

写文档还是不写文档

当面临下面这些问题的时候，有多少开发者是作出类似回答的？

“我没有时间去为它写文档。”
“它的文档是自动生成的。”
“喝多了，稍后修复。”

快进到几个星期或几个月之后。

“这段代码没有文档！”
“这段代码干了些什么事？”
“我应该给它写点文档的……”

不是只有你一个人在战斗。文字仍然能起到它的作用。有时候它可能比写代码要麻烦些，因为它们也不会整洁的编译为二进制，但是它真的没有你想象中的那么的麻烦。

帮助你的社区就是帮助你自己

有很多种方法改进社区文档，而且在项目中大家各司其职，一定有相应的事情去做，而这根本无关乎他们的角色或是否有经验。无论你的项目是刚刚稍具雏形，还是已经运行多年，一定要花时间去改进写作和发布文档的协作方法，这意味着能够让你的贡献者们懂得如何并且轻松的融入到流程中。

记住，最为重要的事情，就是在协作、可扩展和保持简单上下功夫。现在我尝试描述下我曾经在开源社区看到过的，或者说亲身经历过的一些例子，即在开源社区中旨在增加开源项目好文档的参与度，以及在作者和工程师之间架设一座桥梁，从而填补他们之间的知识差异。

文档准则

如果说代码贡献准则是帮助开发者入门和一些附加的标准，那么文档贡献的准则也是相同的目的。文档准则可以包括样式的约定，如何提交文档补丁的流程，以及那些代码贡献需要文档等。

对于文档准则的唯一前提条件就是成功的贡献一些内容。下一次在你的项目中追踪文档的任务的时候，为什么不写一些注意事项或建议并且将它们分享给社区？在一开始就讨论以最佳的方式将文档整合进项目，这能够提醒参与者们这是一个值得讨论的议题。

需要一些例子从而带来一些灵感？可以参考诸如 Django，OpenStack，KDE 以 及 ArchLinux 等项目的文档准则。

还有请记住，文档的提交通常比代码的提交风险要小的多。你必须尽可能的简化你的流程，且要保持你的说明清楚而简洁。如果你需要创建三个不同的账户，为了获取四个荣誉点而仅仅是修复了一个拼写错误，那么请你尽快的远离我，甚至都不需通过代码审核工具。

模版

模版是可以将文档很好的标准化和模块化的，举个例子，看下 README 文件，每个开源项目都会以一个 README 文件作为入口，它放在显眼的位置，这对于让人们留下对文档的第一印象非常的重要。

其中的 Python 行者界，阅读文档的 REDAME 模版在很长一段时间成为了事实上的标准，Drupal 项目也有基于章节描述的 README 模版，以及 Fedora 文档团队是基于文档“菜谱”的指导下工作的，它们不仅描述了如何贡献文档和撰写文档的准则，还用于模版本身。

没有时间制作模版？那就找一些例子来临时的替代！你可以选择一些你自己认为能够代表优秀的结构和格式的议题或内容的文件，将之囊括到你的新的文档中从而作为样例。任何想要作贡献的人，只要看到这些源文件，就至少能够对如何来组织标题和包含哪些章节有了一个基本的认识。

文档写作集中冲刺，黑客明星汇，以及各种研讨聚会

作为一名文档作者，这些都是我喜欢参与的活动，因为它们都涉及到直接参与到开源项目中，且能够立即得到和可重复性的结果。结合开发

者深度的知识和软件的整体方法，对于技术作者来说，不仅能够获得对于文档深刻的洞见，对于整体项目的了解也非常的有益处。

我曾经很荣幸的参与过诸如 fedora 和 NixOS 的文档集中冲刺，以及组织 KDE（过去）和 Django（即将）的研讨聚会，这些集中冲刺和研讨聚会让我学习到了开源文档的内部运作的情况，举个例子，关于 NixOS 的情况，冲刺的结果不仅重构了 3/4 的文档，甚至还撰写了新的文档贡献步骤。

文档写作集中冲刺也是一种将新成员介绍给大家不错的方法，而且一般情况都会有更多的新人加入。在每次的 Django girl 的研讨聚会中，我都会在后续的进程中设立并参与为教程，组织手册，以及项目其它资源的文档写作集中冲刺。这是一个为新人介绍日常的 Github 工作流程很棒的方式，且同时还改进了文档。

咨询台和展位

近来，我很荣幸的成为第一个倡导在开发者的聚会中搞一个文档咨询台，这个聚会是著名的 EuroPython 2015，我和 Paul Roeland 以及 Maciej Szlosarczyk 组成一个团队，我们穿上医生的帽子和长袍，开了临时的文档“诊所，为所有境况不佳的开源文档提供咨询和“诊断”服务。

在预计三小时，最后变为四小时的交流中，我们为来自 15-20 个项目的人们提供了咨询，就他们较为关心的问题，诸如如何重构文档结构，如何制作从原来的文档处理软件转为标记语言和源代码控制的计划，甚至是如何重构站点以让访问者更加容易的访问文档等等。

作为额外的收获，我们获得了进一步的整合和加强在开发者社区的文档讨论。在接下来的研讨会日子里，我们陆续听到那些错过的我们咨询的参与者，或者是不知道我们的参与者，称非常遗憾，因为他们曾经得到过我们的帮助。或许文档的咨询台和展位会是在开发者的会议中成为一种趋势？

在开发者的会议中分享文档创作

技术作者要在开发者会议上分享关于文档的演讲。这绝对不是在开玩笑，我很欣慰的已经看到有很多人开始这么做了。在开发者大会中开发者分享关于文档的演讲也同样是被接受和鼓励的。当然，重要的是要作出能够吸引开发者们的眼球的主题。

在研讨会上关于文档的分享在第一次可能很难吸引那些开发者们，但是未尝不是一个收集信息和想法的好办法，至少，让开发者们对于文档有一个不同的认知。目前我最为喜欢的一个主题是 FOSS DOC 101，因为它覆盖了更高层次的概念，目的是鼓励开源的开发者们认证的对待文档。

文档会议

离 TechComm 会议已经有一段日子了，至今仍对会上的 12 个关于单一来源和没有单独的开发者可以掌控全局的讨论有深刻的印象。新一代的文档社区正在被关注，这要感谢开源的哲学，草根的合作以及各种大胆的实验。

我很自豪的参与到了文档的写作的分享。从两年前在俄勒冈的波特兰的一个一次小型会议开始，我就开始传播之旅，足迹遍布美国的各种聚会，包括两次大型的国际会议。它的使命：整合，而不是区别对待，通过创造给所有技术人员参与到关于内容的讨论中来的空间，来让文档在技术社区中占有一席之地。事实上，参加这些会议的有 50% 以上并非是作者，而他们代表的是各个开源技术和社区。

另外一个值得我们提及的是一个年度的会议：Open Help，它更加的聚焦于文档集中冲刺和开放的讨论。Open Help 邀请所有的开源项目聚在一起，搞得活动由：头脑风暴，黑客，基于本身内容的写作，通过预先选择的演示来传播讨论和激发人们。还有，你不一定非得是一名作者才能参加会议，会议的唯一条件就是你关心内容，关心开源！

传播变革，鼓励合作

我所谈论到的所有的这些想法，是离不开人们投入到实践中来的。这也是我个人能做到的极致了，所以我呼吁大家，无论是开发者还是作者，伸出你们的双手，共同开创一个协作的环境。

如果你是作者，不要犹豫，向开发者会议分享你的文档演示、集中冲刺或者工作室，你的专业的分享，开发者们就能够学习到你的经验，从我第一次参与到开源的世界，我在开发者会议，聚会以及文档冲刺等场合亲历了大家对于文档作者态度的改变，这要感谢这些公开的讨论，我们技术作者作为搭建知识的桥梁，填补工程与用户之间的缝隙，做我们应该去做的！如果你是开发者，请向文档作者们伸出友好之手，邀请他们到你们到研讨会，聚会，和集中冲刺等，参加文档的研讨会，看看作者们是如何思考的，如何工作，以及在社区中是如何做的，每个人都能从质量上乘的内容中受益，而且作者对于分享自己所积累的知识的热情，一定能让你刮目相看，当然你也要助阵开源文档。你我同行，需要你做的就是伸出友好之手。

作者简介

Mikey Ariel，著名开源项目 oVirt 社区负责人，目前是红帽的 OpenStack 平台的高级技术作者，工作地点在 Brno，CZ，空闲时间，她是欧洲文档写作的领导者，Django girl 的聚会组织者，开源项目的文档教练，在开源会议上是热情的演讲者，会谈及关于文档，敏捷和社区。请关注她的 Twitter：thatdocslady

InfoQ 中文站　2015迷你书

开源启示录
第一季

开源软件的未来在于建立一个良性循环，以参与促进繁荣，以繁荣促进参与。在这里，我们为大家呈现本期迷你书，在揭示些许开源软件规律的之外，更希望看到有更多人和企业参与到开源软件中来。

顶尖技术团队访谈录
第三季

《中国顶尖技术团队访谈录》·第三季挑选的6个团队虽然都来自互联网企业，却是风格各异。希望通过这样的记录，能够让一家家品牌背后的技术人员形象更加鲜活，让更多人感受到他们的可爱与坚持。

云生态专刊
2015年04期

《云生态专刊》是InfoQ为大家推出的一个新产品，目标是“打造中国最优质的云生态媒体”。

架构师 月刊

《架构师》月刊是由InfoQ中文站针对高级技术开发和管理人员所推出的电子刊物。