

OPEN SOURCE REVELATION

# 开源启示录

第一季 创刊号

2015年6月

开源咨询

开源软件

开源  
实践

开源经验



InfoQ  
new

扫我，码上开启新世界



Geekbang, 有温度的技术社区。Geek是一种精神, 也是一种态度; Geekbang是一个圈子, 也是一种习惯。在这儿, 你要么是Geek, 要么走在成为Geek的路上。



International Architect Summit

# 全球架构师峰会 2015

中国 · 深圳 · 大梅沙京基海湾大酒店

[深圳站]

2015年7月17-18日



**李高峰**

华为商城首席架构师

**专题演讲:**  
垂直电商技术的艰辛之路



**余庆**

易到用车首席架构师

**专题演讲:**  
移动互联网下基于地理位置变化  
场景实时搜索引擎



**张跃**

批改网CEO

**专题演讲:**  
批改网: 基于语言大数据的英语作文  
自动批改服务系统建设和运营



**陈明**

微信高级工程师、朋友圈负责人

**专题演讲:**  
微信朋友圈技术之道



**方林**

华大基因深圳研发部副院长

**专题演讲:**  
生命科学中的大数据



**徐梦云**

饿了么数据技术部总监

**专题演讲:**  
数据仓库治理



**邓澍军**

猿题库研究部总监

**专题演讲:**  
猿题库: 大数据时代的在线教育



**曹彬彬**

中信证券首席数据应用专家

**专题演讲:**  
证券行业大数据应用探讨

联系方式:

购票热线: 010-89880682

在线咨询(QQ): 2332883546

会务咨询: arch@cn.infoq.com

在线交流(QQ群): 425975960

更多精彩内容, 敬请登陆: sz2015.archsummit.com

Brought by **InfoQ**

QCon上海站

2015年10月15-17日

上海光大会展中心

国际大酒店

[www.qconshanghai.com](http://www.qconshanghai.com)

Brought by **InfoQ**

# QCon

## 全球软件开发大会 International Software Development Conference



QCon是由InfoQ主办的全球顶级技术盛会，每年在伦敦、北京、东京、纽约、圣保罗、上海、旧金山，里约热内卢召开。自2007年3月份首次举办以来，已经有超万名高级技术人员参加过QCon大会。QCon内容源于实践并面向社区，演讲嘉宾依据热点话题，面向5年以上的技术团队负责人、架构师、工程总监、高级开发人员分享技术创新和最佳实践。



[www.qconferences.com](http://www.qconferences.com)

# 卷首语

很难准确说“开源”运动始于何时，自古以来人类就有分享和协作的优良传统，即使在计算机诞生之前亦是如此。1980 年，Usenet 新闻组的建立，为黑客们的分享和协作提供了巨大便利，从那时起，开源软件开始真正登上历史舞台。1983 年，一个长着大胡子的工程师发起了 GNU 计划，宣布要建立一个完全由自由软件组成的操作系统。他对软件自由几乎原教旨主义般的追求，给开源软件带来了巨大的影响（虽然他不认为自由软件就是开源软件）。1991 年，芬兰赫尔辛基大学的一名学生给 Usenet 新闻组发了一条消息：“我正在做一个免费的操作系统，只是个人爱好，不会像 GNU 那么庞大和专业”，从此改变了整个世界。几十年来，开源界留下了一串星光璀璨的名字：Linux、Red Hat、MySQL、Apache、Mozilla、Sourceforge、GitHub……如今，只要你还在使用互联网，这些名字就时刻影响着你。

最近几年随着互联网的再度兴起，开源软件迎来了爆发期。我们观察到近几年开源软件有一些重要趋势。

Docker 项目大获成功。Docker 最初只是一个很酷的想法，但是他们迅速成长为了业界热捧的虚拟化方案。这主要得益于他们良好的社区建设能力、清晰明确的项目组织架构、完善的文档，最重要的是，建立了自己的合作伙伴生态圈。Docker 和 Canonical、Red Hat、Google、Rackspace 都保持了良好的关系。如今，围绕 Docker 生态圈开发的周边项目都已

经数不胜数。Docker 作为近几年一个现象级的成功案例，值得我们深思。

开源数据库市场风头大增。2014 年，商业关系型数据库仅增长了 5.4 个百分点，而开源数据库市场增长了 31%，达到了 5.62 亿美元。Oracle 和 Microsoft SQL Server 的份额正在被蚕食。以前大家担心开源数据库的健壮性和性能，但随着开源数据库的逐渐成熟，选择它们的企业越来越多。

越来越多的企业开始使用开源软件。Black Duck 软件公司发布的 2014 年开源软件发展调查报告显示，越来越多的企业倾向于采取开源方案构建自己的系统。而这些企业选择开源软件的主要原因是可靠的质量、可以拿到源代码、丰富的功能、安全（因为曝光率高）以及易于部署。云计算/虚拟化、内容管理、移动、安全、协作、网络、社交媒体这些领域，开源软件技术已经占据了领导地位，比例从 63% 到 46% 不等。在 3D 打印和智能商务领域，开源软件的势头也非常迅猛，分别达到了 27% 和 26%，而无人机、游戏和 ERP 领域，则是开源软件的未来阵地。

GitHub 带来革命性影响。GitHub 把社交和代码托管完美地融合到了一起，把开源软件推向了一个新的高峰，所以他们成功了。为什么成功的不是 Sourceforge？不是 Google Code？因为开源的本质不是把代码扔那儿就行了，而是社区，是协作。GitHub 深刻认识到了这一点，他们的所有功能都是围绕协作，围绕社区建设来开发的。Google Code 在运营了多年之后宣布关闭，也是和 GitHub 的蓬勃发展有关。

当然，这些趋势只是整个行业良好发展势头的缩影。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。

——曹知渊

# 目录

## 开源资讯

- CockroachDB 母公司 Cockroach Labs 获 625 万美元投资
- Swift 2.0 发布：即将开源，支持 Linux
- Apple 使用 Apache Mesos 重建 Siri 后端服务
- 开源数据库的市场份额将进一步扩大
- Red Hat Linux 严重 Bug 将影响基于 Haswell 架构的服务器

## 开源软件

- Airbnb 发布开源的机器学习软件包 Aerosolve
- Twitter 推出新的流处理器系统 Heron
- Twitter 开源 MySQL 集群管理框架 Mysos
- 谷歌推出 Sky 框架：使用 Dart 编写 120fps 的 Android 应用
- Disque：Redis 之父新开源的分布式内存作业队列

## 开源实践

- 运营开源公司的三个经验教训
- 谷歌的容器之路：从 Borg 到 Kubernetes
- 开源项目运营经验谈
- Roslyn 开源第一年：试炼与凯旋
- GitHub 发布开源许可证使用情况

## 开源经验

- 开源经验：社区是如何管理 HBase 项目的？
- Apache 软件基金会总裁：Docker 是善意的独裁者
- 评价社区经理的绩效
- 为什么开源适合 LinkedIn
- 360 的开源软件使用以及开源文化构建经验



# 全 球 容 器 技 术 大 会

剖析容器企业实践 关注容器生态圈开源项目

2015年8月28日~29日  
北京·新云南皇冠假日酒店

# 开源资讯



# CockroachDB 母公司 Cockroach Labs 获 625 万美元投资

作者 谢丽

前谷歌工程师 Spencer Kimball 是 Photoshop 替代产品 [GIMP](#) 的创建者之一。据 [Wired 报道](#)，在离开谷歌以后，他需要一个与谷歌数据库系统 [Spanner](#) 类似的东西，因为他自己的项目用到了它。Spanner 设计用于在数以百万计的数据库服务器之间处理数据，即使多个数据库服务器或一整个数据中心离线，它也能使谷歌服务保持在线。虽然多数公司不会有谷歌那种规模的服务，但保持在线以及自动在服务器之间平衡资源的能力依然会非常有用。

但市面上并没有这样的产品。于是，Kimball 与同是前谷歌人的 Ben Darnell、Andy Bonventre 等人一起创建了开源数据库 [CockroachDB](#)。该项目在 2014 年 2 月启动后迅速吸引了数十名贡献者，但发展速度并未达到项目团队的预期。截至目前，该软件还未能在真实世界中应用。因此，Kimball 与其他 8 名开发人员辞掉工作，创建了 [Cockroach Labs](#)。近日，该公司从 Benchmark、Google Ventures、Sequoia 等风险投资者那里获得了 625 万美元的资金。其中，Benchmark 曾经投资过 [Hortonworks](#)，而 Google Ventures 投资过 [Cloudera](#)。

CockroachDB 是 Kimball 等人根据谷歌已发表的关于 Spanner 的论文而创建的。多年来，谷歌已经发表了多本白皮书，描述他们的关键创新。这催生了过去十年中最重要的软件。[Hadoop](#) 就是其中一例，它已经成为大数据革命的基础。而一篇关于谷歌 [BigTable](#) 数据存储系统的论文则拉开了新一波数据库设计创新的高潮，人们称之为 NoSQL。目前，苹果、Facebook、Twitter、Netflix 等许多公司都依赖于基于 BigTable 理念而设计的数据库，比如 [Cassandra](#)、[Hbase](#)。

但谷歌已经更进一步，他们已经在很大程度上转移到 BigTable 的继任者 Spanner 上了。Spanner 可以解决 NoSQL 牺牲掉一致性后所带来的问题。Cockroach Labs 公司认为，类似 Spanner 的数据库很快将成为同 Hadoop、NoSQL 一样重要的技术。Kimball 指出，“相比

Spanner 而言，CockroachDB 最大的创新是易于部署。”用户可以在笔记本上安装 CockroachDB 的单个实例，也可以随着业务增长扩展到成千上万的服务器上。Kimball 告诉 Wired：

新一轮的创新已经开始，而且速度会越来越快。

## Swift 2.0 发布：即将开源，支持 Linux

作者 郭蕾

在 6 月 9 日凌晨举行的 WWDC 2015 全球开发者大会上，苹果发布了 Swift 2.0，并宣布将于今年年底开源 Swift 语言。Swift 2.0 引入了很多的新特性以确保开发者可以更快、更简单的构建应用，这些新特性包括更好的性能、新的异常处理 API、可用性检查、支持 Linux 等。苹果将会在新发布的 iOS 9 中全面支持 Swift，iOS 9 beta 也会在今天对所有注册的苹果开发者开放。

Swift 2.0 包含了许多的新特性以及改进，在本周的 WWDC 大会上苹果将会深入介绍这些新特性。同时，在苹果的[开发者博客](#)上，官方也对其中的某几个新功能做了介绍：

**异常处理模型**：新的异常处理模型使用了开发者最为熟悉的 try、catch、throw 关键字，并且还将完美支持苹果的 SDK 以及 NSError。

**可用性**：通过使用新的 SDK，开发者可以操作平台的新功能，但某些老的操作系统可能并不支持这些新特性，所以开发者就需要额外的检查。在处理类似的兼容问题上，Swift 非常的得心应手。如果目标操作系统不支持某个 API，那在编译时 Swift 将会报错。同样，开发者也可以使用#available 来确保代码块可以运行于正确的操作系统版本上。

**协议可扩展**：协议（Protocol）用于统一方法和属性的名称，而不实现任何功能。Swift 2.0 增加了协议扩展，在标准包中可以使用它。当使用全局函数时，Swift 2.0 已经为方法添加了统一的类型，这样开发者就可以使用函数链，以提高代码的可读性。

除了这些新特性之外，另外一个重磅消息是苹果将在今年晚些时候开源 Swift。苹果的软件研发副总裁 Craig Federighi 在 WWDC 大会上表示 Swift 将会是未来的主流开发语言，它应该得到更为广泛的应用。但在大会上苹果并没有过多的解释关于 Swift 开源的更多信息，苹果曾在 2005 年开源过 [WebKit](#)，如果不出所料，Swift 项目的开源管理和运营模式应该和 WebKit

类似。目前可以确认的信息包括：

Swift 的源代码将会基于某个 OSI 组织批准的开源协议进行开源；

苹果将会允许并鼓励社区开发者贡献代码；

未来 Swift 将会重点支持 OS X、iOS 和 Linux 三个平台；

源代码将会包括 Swift 编译器以及标准库。

Swift 开源的消息在 [Hacker News](#) 上引起了激烈讨论。总体来看，开源可以更好的促进 Swift 的发展，一方面开发者可以直接向 Swift 贡献代码，另外开源可以更好的帮助苹果构建其生态系统。另外，关于 Swift 的详细内容读者可以参考阅读 InfoQ 的 [Swift 专栏](#)，更多关于 WWDC 2015 的内容读者可以阅读 “[WWDC 2015 大会十大看点总结：Swift 要开源了](#)”。

# Apple 使用 Apache Mesos 重建 Siri 后端服务

作者 Daniel Bryant 译者 韩陆

苹果公司宣布，将使用开源的集群管理软件 [Apache Mesos](#)，作为该公司广受欢迎的、基于 iOS 的智能个人助理软件 [Siri](#) 的后端服务。Mesosphere 的博客指出，苹果已经创建了一个命名为 [J.A.R.V.I.S.](#)，类似 PaaS 的专有调度 Framework，由此，开发者可以部署可伸缩和高可用的的 Siri 服务。

集群管理软件 [Apache Mesos](#) 将 CPU、内存、存储介质以及其它计算机资源从物理机或者虚拟机中抽象出来，构建支持容错和弹性的分布式系统，并提供高效的运行能力。Mesos 使用与 [Linux 内核](#)相同的系统构建原则，只是他们处在不同的抽象层次上。Mesos 内核运行在每台机器上，通过应用程序 Framework，提供跨整个数据中心和云环境进行资源管理和调度的 API。苹果已经创建了自己专有的调度 Framework 以运行 Siri 的后端服务，将其命名为 J.A.R.V.I.S.。

J.A.R.V.I.S.是“一种相当智能的调度器”（Just A Rather Very Intelligent Scheduler）的缩写，这个名字的灵感来自《钢铁侠》电影中的智能化[计算机助手](#)。苹果公司使用 J.A.R.V.I.S. 作为内部的平台即服务（PaaS）系统，使开发者编写的 Siri 后端应用程序可以部署为可伸缩性和弹性的服务，用于响应 iOS 用户通过个人助理应用程序请求的语音查询。

据 [Mesosphere 的博客](#)报道，在苹果公司总部加州库比蒂诺的聚会上，苹果的开发者表示，他们的 Mesos 集群有数千个节点。支持 Siri 应用程序的后台系统包括约 100 种不同类型的服务，应用程序的数据存储在 [Hadoop 分布式文件系统 \( HDFS \)](#) 中。从基础设施的角度来看，使用 Mesos 有助于使 Siri 具备可伸缩性和可用性，并且还改善了 iOS 应用程序自身的延迟。

Mesos 后端是第三代 Siri 平台 , 告别了之前部署在“传统的”基础设施的历史。 Mesosphere 博客认为 , 从概念上讲 , 苹果公司与 Mesos 的合作以及 J.A.R.V.I.S. 类似于 Google 的 Borg 项目 , 领先于其他支持长时间运行应用服务的类 PaaS Framework , 比如 Mesosphere 数据中心操作系统( DCOS )的相关组件 Mesosphere Marathon 和出自 Twitter 基础设施团队的 Apache Aurora 。

Mesosphere 高级研究分析师 Derrick Harris 在 Mesosphere 的博客中表示 , 关于 Siri 由 Apache Mesos 集群管理软件支撑的公告是对 Mesos 成熟度的证明 :

苹果公司能够信任使用 Mesos 支撑 Siri——这是一个复杂的应用程序 , 用以处理只有苹果知道每天会有多少数量的、来自数以亿计的 iPhone 和 iPad 用户的语音查询——这足以说明 Mesos 的成熟度 , Mesos 已经为各种类型的企业带来巨大影响做好了准备。

InfoQ 采访了 Mesosphere 高级副总裁 Matt Trifiro , 并询问了这项公告对正在考虑部署应用到 Mesos 的企业和软件开发者会有什么影响 :

#### InfoQ : 为什么苹果的这项公告对 Mesos 和 Mesosphere 很重要 ?

Trifiro : 苹果公司宣布 , 他们完全重建了 Siri , 以运行于 Mesos 之上。这再次表明 , Mesosphere DCOS 中的分布式内核 Mesos , 是编排大规模容器和构建新的分布式系统的黄金标准。

#### InfoQ : 不是每家企业都能达到苹果公司的规模 , 那么传统企业怎样应用 Mesos 呢 ?

Trifiro : 像苹果和 Twitter 这样的公司 , 几乎全部的基础设施都使用了这项技术。因为 Siri 和 Twitter 都依赖于 Mesos , 可想而知 , 它必须是可靠的。但是 , 开源的 Apache Mesos 是一项非常尖端的技术 , 通过开源工具手工装配 , 并将 Mesos 用于生产环境是非常困难的。这正是 Mesosphere 产生的原因。

任何公司都能使用这项久经考验的技术，构建完整的数据中心操作系统（DCOS），并具备和Twitter或者苹果公司同等的能力和自动化效果，而不必成为Twitter或者苹果那样大规模的公司。

### InfoQ：苹果公司从Mesos API直接实现了一套调度器(J.A.R.V.I.S.)，这意味着什么呢？

**Trifiro**：Mesos最强大的方面其一就是，它提供了用于构建新的分布式系统的基本功能。如果你去看其它的分布式系统，比如早于Mesos出现的Hadoop，它有几十万行代码，很多地方是在重复制造轮子。所有的失败处理、网络实现、消息传递和资源分配的代码，开发者不应重写这些功能。而为程序员提供了内置这些功能的Mesos内核的话，他们就可以快速构建新的高可用性和弹性分布式系统，而无需重复所有基本的功能。他们可以专注于业务逻辑的实现上。

### InfoQ：Mesos和Mesosphere DCOS之间是什么关系？

**Trifiro**：Mesosphere DCOS是一种新型的操作系统，跨越数据中心或云环境中的所有机器，将他们的资源放到一个资源池中，使他们的行为整体上像一个大的计算机。Apache的开源项目Mesos是这个操作系统的内核。我们将其和其他组件包装到一起，包括初始化系统（marathon）、文件系统（HDFS）、应用打包和部署系统、图形用户界面和命令行界面（CLI）。所有这些组件一起构成了DCOS。这就像苹果公司的Yosemite操作系统或者像Android，他们各有一个内核（分别是BSD和Linux），他们为内核添加了系统服务和工具，使内核成为值得笔记本电脑或者智能手机使用的产。我们为数据中心所做的工作也是相同的。

# 开源数据库的市场份额将进一步扩大

作者 谢丽

在过去三十年中，数据一直锁在专有数据库系统中，但据 InfoWorld 报道，情况正在快速地发生着变化。Gartner 分析师 Merv Adrian 的断言“92.1%的数据库收入来自排名前 5 的供应商”是正确的，他们全部都是专有数据库，第一位是 Oracle，IBM 和微软次之，SAP 与 Teradata 紧随其后。

但是，出于方便性考虑，越来越多的开发者开始在开源数据库中寻找新数据的存储方案。而且，根据 DB-Engines 的数据库月度流行度列表来看，数据库的排名显然不同。

Rank	May 2015	Apr 2015	May 2014	DBMS	Database Model	Score		
						May 2015	Apr 2015	May 2014
1.	1.	1.	1.	Oracle	Relational DBMS	1442.10	-4.03	-60.64
2.	2.	2.	2.	MySQL	Relational DBMS	1294.26	+9.68	-14.83
3.	3.	3.	3.	Microsoft SQL Server	Relational DBMS	1131.03	-18.09	-76.77
4.	4.	↑ 5.	5.	MongoDB +	Document store	277.32	-1.27	+52.70
5.	5.	↓ 4.	4.	PostgreSQL	Relational DBMS	273.52	+5.20	+32.88
6.	6.	6.	6.	DB2	Relational DBMS	201.04	+3.40	+14.57
7.	7.	7.	7.	Microsoft Access	Relational DBMS	145.58	+3.39	+0.22
8.	8.	↑ 9.	9.	Cassandra +	Wide column store	106.55	+1.66	+24.82
9.	9.	↓ 8.	8.	SQLite	Relational DBMS	105.16	+2.86	+15.87
10.	10.	↑ 13.	13.	Redis	Key-value store	94.73	+0.17	+32.69

也许有人会认为，流行度并不能等同于技术的实际应用情况，但 Gartner 分析师 Merv Adrian 和 Donald Feinberg 认为：

对于商业开源数据库而言，其应用远远超出了数据所显示的状况。

对于开源数据库市场份额的增长，Feinberg 和 Adrian 提到其中一个的原因，就是：

开源 RDBMS 产品日趋成熟，提供了 DBA 技能、DBA 工具以及与 RDBMS 几乎相同的功能。OSDBMS 已经成功应用在很大一部分组织的关键应用程序中。

而 Baron Schwartz 认为，开源数据库应用如此广泛以及快速增长的首要原因是社区：

在考虑一个数据库时，最重要的事情是……成功案例。世界已经不同于数十年前，那时，优秀的数据都是专有的，没人知道它们的工作原理，因此，概念可行性验证是一种主要的销售策略。但现在，大部分新数据库都是开源的，用户要么了解它们的工作原理，要么拥有足够的知识，可以在需要的时候把它弄清楚。

这样的社区使得软件购买更安全，对于数据库而言尤其如此，因为这减少了有关如何恰当应用数据库的不确定性。Schwartz 继续写道：

如果有一个号称可以完美解决所有问题的神奇数据库，以及一个应用广泛可以通过谷歌查询的数据库，那么我不会很认真地对前者进行研究。我想在线阅读关于应用场景、面临及已解决的扩展挑战、未完善之处、脚本、调优、技巧等方面的信息。我要大量的 Stack Exchange 讨论和博客文章。我想看到人们将数据库用于同我类似的工作负载，以及不同的工作负载，我想听到它的优点和不足。

据 Gartner 消息，在 2018 年年底，DevOps 团队将影响 30% 的数据中心基础设施购买。开发者正在用浏览器为开源数据库投票。另外，Gartner 还注意到，OSDBMS 已经占据了目标市场的 25%。这也就是为什么 DB-Engines 数据库月度流行排行榜上排名前十的数据库有一半是开源数据库，而前十之外还有许多开源数据库正在挑战前十的位置。

# Red Hat Linux 严重 Bug 将影响基于 Haswell 架构的服务器

作者 Jeff Martin 译者 魏星

最近，[Azul Systems](#) 公司的 CTO 与联合创始人 [Gil Tene](#) 在 Google Groups [报告](#)了一个十分重要，但鲜为人知的 [Linux 内核补丁](#)，采用英特尔 [Haswell](#) 架构的 Linux 系统用户和管理员尤其应该关注该问题。特别是基于 Red Hat 发行版的用户（包括 [CentOS 6.6](#) 以及 [Scientific Linux6.6](#)），应该立即更新这个补丁。即便是运行在虚拟机中的 Linux，如果这个虚拟机是在流行的云平台上（如 [Azure](#)、[Amazon](#) 等），它也可能跑在 Haswell 机器上，打补丁应该是有好处的。



Tene 是对该缺陷的描述如下：

这个内核漏洞的影响非常简单：在一些看似不可能的情况下，用户进程会死锁并被挂起。任何一个 [futex](#) 调用等待（即使被正确地唤醒）都有可能永远被阻止执行。就像 Java 里的 `Thread.park()` 可能会一直阻塞那样，等等。如果足够幸运，你会在 `dmesg` 日志中发现 `soft lockup` 消息；如果没那么幸运（比如跟我们这样），你将不得不花几个月的人工成本去排查代码中的问题，还有可能一无所获。

Tene 继续解释了这个缺陷代码是如何执行的（最终可以归结到一个遗漏了 `default` 情况的 `switch` 块）。现在最大的问题是，尽管问题代码已经在 2014 年 1 月修复，但是在 2014 年 10 月左右，该缺陷又被移回了 Red Hat 6.6 家族系统中。其他系统包括 SLES、Ubuntu、Debian 等有可能也被影响了。

这些系统的修复情况现在并不一致，并且有可能被忽略。Red Hat 用户应该采用 RHEL 6.6.z 或更新的版本。Tene 还指出另一个关键点在于，对于要将哪些东西放入内核，不同的发

行版会有不同的选择，这也导致问题的修复情况并不一致。

例如，对于 RHEL 7.1 而言，“其实上游的 3.10 内核是没有这个 bug 的，但 RHEL 7 的内核又不是纯粹的上游版本。不幸的是，RHEL 7.1（就像 RHEL 6.6 那样）在移植的时候把（基于 RHEL 7 版本）这个 bug 包含了进去……我认为其他发行版可能也是这么做的。”

对基于 RHEL 的发行版，Tene 提供了一个快速参考列表：

- RHEL 5（包括 CentOS 5 和 Scientific Linux 5）：所有版本（包括 5.11 版）都没有问题。
- RHEL 6（包括 CentOS 6 和 Scientific Linux 6）：从 6.0 ~ 6.5 版都没问题。但 6.6 版存在缺陷，而 6.6.z 版本没有问题。
- RHEL 7（包括 CentOS 7 和 Scientific Linux 7）：7.1 是有缺陷的。并且截至 2015 年 5 月 13 日也没有一个 7.x 的修复。

尽管在 [Hacker News](#) 上对受影响系统的数量存在一些争议，但它提供了一些环境来检查你的系统是否需要修复。

# 开源软件

OPEN SOURCE SOFTWARE

The word cloud is composed of several large, bold words: OPEN, SOURCE, and SOFTWARE. Interspersed among these are numerous smaller words that provide context and specific examples. These include: PRODUCT, PROCESS, BUSINESS, PRESS, PDF, BEER, MANY, CALLED, HARDWARE, CODE, MOVIE, LICENSES, CENTURY, RESEARCH, TECHNOLOGY, COMMUNITY, CULTURAL, PLATFORM, OPEN-SOURCE, COMPUTER, PROJECTS, MANUFACTURERS, VERSION, CREATIVE, INTERNET, PUBLIC, ALSO, INFORMATION, NETWORK, SHARING, SIMILAR, DIGITAL, EXAMPLES, PUBLISHING, PHARMACEUTICALS, MESSAGEBOARDS, BLOGS, RAYMOND, INCLUDING, WORK, COPYRIGHT, SYSTEM, INNOVATION, USE, ONLINE, USE, WORK, WORKS, richard, RESOURCES, CONCEPT, CREATED, NEW, USING, LICENSE, WIKIPEDIA, WITHOUT, WORLD, USERS, SCIENCE, VARIOUS, ANOTHER, GENERAL, INITIATIVE, ETHICS, DESCRIBE, COMMUNITIES, POTENTIAL, SYSTEMS, MADE, EVEN, USED, BEGAN, WEB, VOL, LIMITED, LIMITE, AUDIO, BASED, TIME, DATA, LAKE, OTHERS, KNOWLEDGE, PATENT, EDIT, COMMONS, ISBN, MODEL, FILM, IDEA, ERIC, INTERNATIONAL, LIST, COST, COMPANY, FORMATT, LAW, PRODUCTS, CASE, ECONOMIC, ONE, SCIENTIFIC, DEVELOPMENT, STANDARDS, FILE, AVAILABLE, MEDIA, PROPERTY, ORGANIZATIONS, CULTURE, INTERNATIONAL, LIST, COST, COMPANY, DESIGN, CHARGED, HISTORY, MAKING, PROJECT, NOW.

# Airbnb 发布开源的机器学习软件包 Aerosolve

作者 孙镜涛

Airbnb 是一个旅行房屋租赁网站，用户可以通过该网站发布、搜索度假房屋租赁信息并完成在线预定程序，它成立于 2008 年 8 月，总部设在美国加州旧金山市。Airbnb 的用户遍布 190 个国家的近 33000 个城市，发布的房屋租赁信息达到 50 多万条，被时代周刊称为“住房中的 EBay”。在 6 月 4 日举行的 OpenAir 开发者大会上，Airbnb 发布了一个为人而设计的机器学习软件包——[Aerosolve](#)。

与其他的机器学习库相比，Aerosolve 具有以下特点。

- 特征呈现基于 [thrift](#)，支持 Pairwise Ranking Loss 和单上下文的多条目呈现。在 Aerosolve 中，特征会按照逻辑分组，每一个组称为一个特征簇，我们可以一次性地对整个特征组进行转换，或者将两个不同的特征簇组合到一起创建新的特征簇。每一个特征向量（FeatureVector）有三种类型：stringFeatures、floatFeatures 和 denseFeatures。
- 支持一种[特征转换语言](#)，让用户能够对特征进行更多的控制。Aerosolve 将特征转换包含在一个独立的转换模块中，与模型解耦，用户既能够将转换操作拆散使用，又可以提前转换相关数据。例如，在一个应用程序中用户可以在运行时上下文确定之前对一个文集中的条目数据进行转换和存储，然后在运行时做上下文的转换，并将转换后的上下文与提前转换的条目逐一进行联合获取最终的特征向量。常用的转换操作包括：[列表转换](#)、[交叉转换](#)和[多尺度网格转换](#)。
- 人类友好的[调试模型](#)。模型目录中包含很多模型，但是其中最主要的两个是：线性模型和样条模型，其他的都是试验性的模型或者是为可推理模型创建转换的子模型。
- 独立的轻量级 [Java 推理代码](#)。

- 使用 Scala 代码进行[训练](#)。
- 简单的[图片内容分析代码](#)，适合于图片的排序或者排名。

需要注意的是，Aerosolve 适合于稀疏的、可推理的特征，例如搜索（搜索关键词、过滤词）或价格（房屋的数量、位置和价格）中通常会出现的特征；不适合非常密集的人类无法推理的特征，例如原始的像素集或者音频样本。

最后，Aerosolve 所需的制品托管在 bintray 上，如果你使用 Maven、SBT 或者 Gradle，那么需要将仓库指向 [bintray](#)。如果想了解更多信息，可以点击[这里](#)。

## Twitter 推出新的流处理器系统 Heron

作者 张天雷

2011 年，Twitter 发布了开源的分布式流计算系统 [Storm](#)。四年后，随着用户数量的急剧增加，Twitter 每天要处理的事件已经增加到十亿以上。Storm 系统应对如此庞大而复杂多样的流数据变得十分困难。为了解决该问题，Twitter 公司近期开发了一套全新的流处理系统——Heron。近日，[Twitter 公司在 SIGMOD 2015 会议上对 Heron 进行了介绍。](#)

据 Twitter 公司的技术经理 Karthik Ramasamy 表示，Twitter 公司之前对 Storm 所存在的问题以及新平台的功能需求进行了详细分析。首先，Storm 调试能力较弱的问题曾让 Twitter 员工比较困扰。[在 Storm 中，一个拓扑中的多个组件是捆绑在操作系统的一个进程中的。](#)这就使得在拓扑出现问题时很难迅速定位问题的根源。用户不得不借助 cleaner 映射来帮助实现逻辑计算单元到物理进程的映射，从而辅助调试。此外，Storm 还需要专门的集群资源来运行拓扑。这就使得它不能利用流行的集群调度软件进行计算资源的优化。而且，在使用 Storm 时，用户需要手动隔离机器才能部署一个新的产品拓扑。同样，在拓扑不再使用时还需要手动回收机器资源。所有 Storm 存在的这些问题严重限制了 Twitter 处理事件的能力，而且带来时间和计算资源的巨大浪费。因此，Twitter 认为新的系统需要具备能够每分钟处理上亿的事件、大规模处理数据的延迟为亚秒级、行为可预测、高数据精度、遇到局部流量过高或流水线拥堵时能够自行调整输入速率、调试方便以及能够在共享式框架中部署等功能。

据 Karthik 透露，Twitter 当初考虑了三种可能的方案——扩展现有的 Storm 系统、利用别的已经开源的系统和开发一套新的系统。然而，Storm 系统的核心部件并不能满足现有的需求，而对其进行修改又需要比较长的开发周期。同时，其他的开源流处理框架不能满足 Twitter 在规模、吞吐量以及延迟等方面的需求。更关键的是，它们不能与 Storm 的 API 相兼容，迁移工作会十分复杂。因此，Twitter 最终决定开发一套全新的实时流处理系统。

根据设计要求，Heron 保持了与 Storm 相同的数据模型和 API，运行的也是由 spout 和 bolt 构成的拓扑。其总体框架包含了一个调度器和若干个拓扑。该调度器只是一个抽象模型，可以具体化为 YARN、Mesos 或者 ECS 等，方便资源共享。用户利用 API 提交拓扑到调度器后，调度器把每个拓扑当作一个单独的任务，并根据集群中节点利用情况分派若干个容器来执行。在这些容器中，其中一个运行拓扑 master，负责管理拓扑。剩余的每一个容器都需要运行一个流管理器来负责数据路由、一个 metric 管理器负责收集和报告各种各样的 metric 以及若干个 Heron 实例进程来运行用户自定义的 spout/bolt 代码。最后，拓扑的元数据，如物理规划和执行细节等都被保存在 ZooKeeper 中。

因此，Heron 能够轻松部署在运行如 Mesos、YARN 或者自定义调度框架的共享架构中。而且，Heron 向后兼容 Storm，使得原来基于 Storm 的其它系统可以继续使用。在 Heron 中运行已有的 Storm 拓扑完全不需要任何改变，移除了复杂的迁移过程。容器中的 Heron 实例执行的是单独的任务，保证了用户利用 jstack 或者 heap dump 等工具即可进行实例的调试。Metric 收集器又使得拓扑中任何组件的失效变得十分明显，大大降低了调试的难度。此外，Heron 有一个背压机制能够在运行过程中动态调整拓扑中数据流的速度，而不影响数据精度。同时，与 2013 年 10 月发布的 Storm 相比，Heron 的吞吐量可以到达其 10-14 倍，而延迟时间却只是它的 1/15-1/5，资源消耗也更低。

目前，Heron 已经作为 Twitter 的主要流处理器系统在运行，其中包括了数百个拓扑。由于 Heron 的低资源消耗特性，迁移后的系统硬件减少了 2/3，大大提高了物理资源的利用率。Karthik 也表示，Twitter 公司非常愿意与 Storm 社区或者其他开源的实时流处理系统社区分享 Heron 的经验和教训。

## Twitter 开源 MySQL 集群管理框架 Mysos

作者 谢丽

Mysos 是一个用于运行 MySQL 实例的 Apache Mesos 框架。它极大地简化了 MySQL 集群的管理，具有高可靠性、高可用性及高可扩展性等特点。有关其具体功能，可以查看 [InfoQ 前期的报道](#)。

Mysos 需要 Python 2.7 及 Mesos Python 绑定。其中，后者包含两个 Python 包。mesos.interface 位于 PyPI 上，可以自动安装。但 mesos.native 是平台依赖的，用户需要在自己的机器上构建([相关命令](#))，或者下载相应平台的编译版本(Mesosphere 提供了部分 [Linux 平台的 egg 文件](#))。

Mysos 主要包含如下两个组件：

- mysos\_scheduler：用于连接 Mesos 主节点及管理 MySQL 集群；
- mysos\_executor：用于启动 Mesos 从节点(基于 mysos\_scheduler 请求)执行 MySQL 任务。

这两个组件可以单独构建和部署，也可以使用 [PEX](#) 将二者及其依赖包打包成一个可执行文件(具体过程参见[这里](#))。

Mysos 提供了一个 REST API，用于在 Mesos 上创建和管理 MySQL 集群。下面是集群创建的示例代码：

```
curl -X POST 192.168.33.7/clusters/test_cluster3 --form "cluster_user=mysos" \ --form "num_nodes=2" --form "backup_id=foo/bar:201503122000" \ --form 'size={"mem": "512mb", "disk": "3gb", "cpus": 1.0}'
```

其中，集群名称为 test\_cluster3，cluster\_user 指定了对集群中所有 MySQL 实例都拥有管理员权限的用



户 , num\_nodes 指定了集群节点数 , backup\_id 指定了 MySQL 实例启动时需要从哪个 MySQL 备份恢复 , size 指定了分配给实例的资源。该命令会返回用于访问 MySQL 实例的密码以及集群 URL。

Mysos 是 Twitter 和 Mesosphere 合作的产物。为了该项目的长远发展，在将其开源的同时，Twitter 也向 Apache 基金会提交了[孵化提案](#)，希望以这种方式确保该项目遵循 Apache 2.0 许可协议，促进 Mysos 社区的发展壮大。

# 谷歌推出 Sky 框架：使用 Dart 编写 120fps 的 Android 应用

作者 丛一

通常，非游戏类的安卓应用都是由 Java 语言编写的，不过[谷歌的一个团队正在尝试用内部的](#) Web 开发语言 Dart 以一种全新的方式编写安卓应用。这种方式的重点是完全去 Java 化，专注于速度并且与 Web 深度整合。

Dart 最初是由 Chrome 的 V8 Javascript 引擎团队成员所创建。最近该团队主办了 Dart 开发者峰会，并对外展示了名为 Sky 的 Dart on Android 项目。该项目目前已经基于 Apache 许可协议开源，源代码已经上传至 [GitHub](#)。

Sky 由两个组件组成：

- Sky 引擎。用 C++ 编写的引擎是整个系统的核心。引擎提供了许多用于构建高质量应用的基础元素，包括软实时调度程序和分层次的保留模式图形系统。
- Sky 框架。名为 Effen 的 Sky 框架通过提供用户熟知的交互部件，如按钮、无限列表和动画，让使用 Sky 构建应用变得更加容易。这些可扩展的组件所遵循的函数式程序设计风格的灵感来自于 [React](#)。

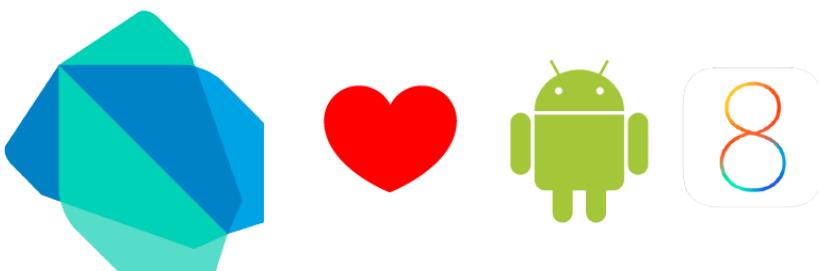
Sky 项目最大的目标是快速响应。目前，绝大多数的应用和开发人员在应用平滑度方面的标准仍然是 60FPS，而 Dart 团队计划将这一目标提升至 120FPS。这一目标乍听起来不太可能，因为目前主流的智能手机显示屏的刷新频率也只有 60Hz，根本无法有效显示 120FPS 的刷新频率。而对于安卓系统来说，绝大多数应用连 60FPS 的标准都达不到，更不要说 120FPS 了。

在 Dart 团队所展示的一个[示例应用](#)中，全部帧数渲染仅耗时 1.2 毫秒。尽管这个示例相当简单，一定程度上说明 Sky 仍有很大空间可以在更加复杂的应用中实现流畅的动画效果，这

也表明 120FPS 的目标（8 毫秒的渲染时间）并非痴人说梦。Dart 团队将 Sky 称为“不会卡的设计”，API 不会妨碍用户界面主进程的执行，也就是说即使应用的速度变慢，用户界面仍会保持流畅的响应速度。

Sky 将其 Web 后端也带到了移动开发领域。Sky 不依赖于平台，其代码可以运行在 Android、iOS，或是任何包含 Dart 虚拟机的平台上。这类应用的运行类似于网站。应用的很大一部分是基于 HTTP 的，通过 HTTP 提供服务的方式可以让开发变得更加简单。无需编辑代码、编译、安装新的应用，只需要在 HTTP 服务器端编辑代码，然后关闭并再次打开应用就可以用最新的代码“刷新”应用。就像 Web 浏览器一样。Sky 框架为安卓开发提供了一整套物化设计（Material Design）的组件，开发者可以很方便地添加工具栏、触摸效果、导航栏以及安卓应用中应有的所有其他组件。

与普通的应用类似，Sky 应用可以获取到完整的安卓特权和 API 的访问权限。但这种权限与 Web 服务器自动更新相结合，可能会带来比较大的安全隐患。不过，目前为止 Sky 仍然只是一个实验性的项目，这类问题在 Sky 成为正式的安卓应用解决方案之前，务必需要得到解决。虽然目前 Sky 团队仍然在持续不断地进行项目迭代，Sky 框架和底层引擎将来也可能会有多次不兼容的变动，不过 Sky 这种专注于速度和另辟蹊径的安卓开发方法仍值得我们密切关注其后续的发展动态。



# Disque : Redis 之父新开源的分布式内存作业队列

作者 谢丽

Disque 是 Redis 之父 Salvatore Sanfilippo 新开源的一个分布式内存消息代理。它适应于“Redis 作为作业队列”的场景，但采用了一种专用、独立、可扩展且具有容错功能的设计，兼具 Redis 的简洁和高性能，并且用 C 语言实现为一个非阻塞网络服务器。有一点需要提请读者注意，在 Disque 项目文档及本文中，“消息（Message）”和“作业（Job）”可互换。

Disque 是一个独立于 Redis 的新项目，但它重用了 Redis 网络源代码、节点消息总线、库和客户端协议的一大部分。由于 Disque 使用了与 Redis 相同的协议，所以可以直接使用 Redis 客户端连接 Disque 集群，只是需要注意，Disque 的默认端口是 7711，而不是 6379。

作为消息代理，Disque 充当了需要进行消息交换的进程之间的一个中间层，生产者向其中添加供消费者使用的消息。这种生产者-消费者队列模型非常常见，其主要不同体现在一些细节方面：

- Disque 是一个同步复制作业队列，在默认情况下，新增任务会复制到 W 个节点上，W-1 个节点发生故障也不会影响消息的传递。
- Disque 支持至少一次和至多一次传递语义，前者是设计和实现重点，而后者可以通过将重试时间设为 0 来实现。每个消息的传递语义都是单独设置的，因此，在同一个消息队列中，语义不同的消息可以共存。
- 按照设计，Disque 的至少一次传递是近似一次传递，它会尽力避免消息的多次传递。
- Disque 集群的所有节点都有同样的角色，也就是“多主节点”( multi-master )。生产者和消费者可以连接到不同的队列或节点，节点会根据负载和客户端请求自动交换消息。
- Disque 支持可选的异步命令。在这种模式下，生产者在向一个复制因子不为 1 的队列中添加一个作业后，可以不必等待复制完成就可以转而执行其它操作，节点会在后台完成复制。

- 在超过指定的消息重试时间后，Disque 会自动将未收到响应的消息重新放入队列。
- 在 Disque 中，消费者使用显式应答来标识消息已经传递完成。
- Disque 只提供尽力而为排序。队列根据消息创建时间对消息进行排序，而创建时间是通过本地节点的时钟获取的。因此，在同一个节点上创建的消息通常是按创建顺序传递的，但 Disque 并不提供严格的 FIFO 语义保证。比如，在消息重新排队或者因为负载均衡而移至其它节点时，消息的传递顺序就无法保证了。所以，Salvatore 指出，从技术上讲，Disque 严格来说并不是一个队列，而更应该称为消息代理。
- Disque 通过四个参数提供了细粒度的作业控制 分别是复制因子( 指定消息的副本数 )、延迟时间( 将消息放入队列前的最小等待时间 )、重试时间( 设置消息何时重新排队 )、过期时间 ( 设置何时删除消息 )。

需要注意的是，Disque 项目尚处于 Alpha 预览测试阶段，代码和算法未经充分测试，还不适合用于生产环境。在接下来的几个月里，其实现和 API 很可能会发生重大变化。此外，它还有如下限制：

- 其中还包含许多没有用到的 Redis 代码；
- 它并非源于 Salvatore 的项目需求，而是源于他看到人们将 Redis 用作队列，但他不是这方面的专家；
- 同 Redis 一样，它是单线程的，但鉴于它所操作的数据结构并不复杂，将来可以考虑改为多线程；
- Disque 进程中的作业数量受可用内存限制；
- Disque 没有进行性能优化。

总之，该项目尚处于研究测试阶段。感兴趣的读者可以查看该项目的 [GitHub 页面](#)，了解更多信息。

# 开源实践



# 运营开源公司的三个经验教训

作者 曹知渊



从表面上看起来，运营一家开源软件公司似乎很简单：把代码放到 GitHub 上，或者发起一个 Apache 软件基金会的项目，然后建立一个社区，把有相同想法的人聚拢来，接下来就是开公司，拉投资，最后可能上市，也可能不上市。一切看起来都水到渠成。而在

Grant Ingersoll 看来，运营一家开源软件公司意味着独有的机遇和挑战。Ingersoll 是 LucidWorks 的联合创始人兼 CTO，有丰富的创业经验，日前他在 [opensource.com](#) 上分享了他从自己的经历中学到的三个教训。

Lucidworks 公司的目标是利用搜索、机器学习、自然语言处理等技术，使信息访问变得更容易，他们的主要项目是围绕 Apache Lucene 和 Solr 来构建的。所以他们面临的第一个问题是：哪些功能应该贡献给社区，哪些功能应该当作商业卖点？

开源社区通常不是由一个人控制的，基于开源社区创立的公司会面临各种冲突。有时候对于同一个功能，社区的实现思路和公司不一样；有时候公司规划了一个商业功能，结果社区提前做出来了；最糟糕的情况可能是，公司想要把某个产品商业化，结果却连它的品牌所有权都没有。对此，Grant 给出的建议是，必须坦然面对，利用沟通解决问题。当公司想为社区贡献某个功能的时候，及早地公布计划，以免别人做重复的工作；如果已经有人在做类似的功能，更要跟他们好好沟通，争取让他们把公司的诉求也考虑进去；此外，可以经常搞点社区聚会，把社区团结起来。

除了沟通，Grant 认为更重要的一点就是，必须贡献对内容。公司应该寻求贡献底层的、

通用的组件，这些组件可以用来构建更高层的功能。而这些高层功能，很适合作为公司的商业卖点。Lucidworks 也是反复尝试才走对了路。他们给社区贡献了核心的分析功能，而他们产品中的推荐功能，就是基于此构建的。Grant 给出了 Lucidworks 处理此问题的三条原则：

1. 保持专职的工程师团队，他们不做其他事情，只为开源社区贡献代码。而这些开源项目，则会成为公司商业产品的基石。
2. 将中间件、第三方集成和 UI 作为商业化产品。
3. 提供开箱即用的各种常用数据分析技术的实现。

这三条原则使得 Lucidworks 在给社区作出贡献的同时，不对社区产生不利的影响。

Grant 遇到的第二个棘手的问题是，如何定位开源软件公司的商业模式。在 Lucidworks 初创时期，主要产品就是“知识”。虽然他们也有过商业化产品，但是最核心的价值还是那些熟知开源代码的工程师的知识。用户在部署开源软件遇到问题时，就会向他们求助，他们就依靠提供咨询服务来赚钱。Lucidworks 逐渐意识到这不是长久之计，因为问题一旦解决，用户就不再需要他们。而且那时用户和他们签的支持合同也只有一年，因为 Solr 通常拿来就能用，所以用户也不大需要强有力的技术保障。

Lucidworks 还是慢慢地从用户身上得到了一些启发。他们发现虽然 Solr 开箱即用，但是用户会不断地来问一些越来越深入的问题，比如，基本的搜索功能可以工作后，用户想知道如何把自然语言处理工具集成进去。用户经常会把他们最关注的一些问题反馈给 Lucidworks 的产品管理团队。据此，Lucidworks 逐渐把自己的模式演化为一种他们称之为“客户成功”的模式。这个模式最大的不同在于，以前工程师们坐在办公室里等电话响，做的都是一锤子买卖；而现在，对于提出问题的客户，他们会不断跟踪，帮助他们成功地部署、使用这些开源软件。另一方面收集这些用户的诉求，不断地反馈给产品团队，做出用户真正需要的、愿意为之付钱

的功能。这样，用户的距离拉近了，产品也越来越出色。

Grantz 最后谈到的问题是人的问题。这有几个方面，首先要找到合适的人，这对于一个基于开源项目开发收费产品的公司来说，并不是一件很容易的事。公司招募的工程师既要熟悉开源软件的运作模式，也要认同基于此来开发收费软件的做法。通常，这是难以两全的，做惯了闭源软件的人，很可能搞不清楚开源社区的模式、流程、文化等，当然反之也一样，而且认同开源软件的人还有个问题，他们经常不愿意为收费软件贡献力量。第二个难题是，Lucidworks 从开源社区招募的工程师，通常都分布在各个地方，他们只能远程办公。开发开源软件的工程师习惯于松散的沟通方式，这种习惯跟一个商业公司的流程是格格不入的。Grantz 认为解决这个问题的办法就是多交流以及规范文档。他认为现在有不少出色的工具可以帮助不同地点的人协同工作，减少摩擦，应该尽可能利用这些工具。另外，公司应该留出预算，定期把这些分布在各地人聚拢来，拉近距离，促进交流。

当然，不是什么人都适合远程工作的。在 Lucidworks，服务器软件团队的很多人是远程的，而产品管理和 UI 团队通常是在办公室一起工作的。有时候这些员工需要对同伴说，“赶紧来看看这个设计”，只有大家在一问房间里工作才能办到。而开发服务器的工程师，他们则希望不被人打扰，潜心钻研。

一个开源软件创业公司在找到可复制、可规模化的商业模式之前，总是要经过各种起起伏伏，Lucidworks 在成长过程中，也无时无刻不在与上面几个问题做斗争，所以 Grantz 希望把他们学到的经验教训分享出来，帮助更多的公司战胜困难。如果要让 Grantz 用一句话来总结他十多年的开源软件创业生涯学到的经验，那就是：“你永远不知道下一个创意在哪里，所以敞开心扉，随时准备拥抱它。”

## 谷歌的容器之路：从 Borg 到 Kubernetes

作者 张天雷

作为谷歌公司的开源容器集群管理系统，Kubernetes 在 Docker 技术之上，为容器化的应用提供了资源调度、部署运行、服务发现和扩容缩容等丰富多样的功能。在项目公开后不久，微软、IBM、VMware、Docker、CoreOS 以及 SaltStack 等多家公司便纷纷加入了 Kubernetes 社区，为该项目发展作出贡献。[谷歌高级副总裁 Urs Hözle 也曾表示](#)，通过多家公司及社区的共同合作，要确保 Kubernetes 在任何应用程序和任何环境（私有云、公共云以及混合云任何环境）都是一个强大并且开放容器的管理架构。

目前，Kubernetes 正处在快速发展的阶段，努力成长为容器管理领域的领导者。其迅速崛起吸引了大量开发人员的注意。除了对产品本身的兴趣，人们更感兴趣的是 Kubernetes 背后成功的原因和其发展过程中所经历的教训。但是，谷歌公司在之前对于内部管理系统 Borg 相关的信息都一直避而不谈，让外界很难了解 Borg 以及 Borg 与 Kubernetes 的关系。在 2015 年的 Eurosys 会议上，[谷歌终于公布了相关的细节](#)，让大家可以了解谷歌从 Borg 到 Kubernetes 的成功之路。接下来，本文就从全角度多方面分析，详细揭示 Kubernetes 与 Borg 的关系，从而探究谷歌领先全球技术的奥秘。

Borg 是谷歌公司的内部容器管理系统。早在十几年前，该公司就已经部署 Borg 系统对来自于几千个应用程序所提交的 job 进行接收、调试、启动、停止、重启和监控。该项目的目的是实现资源管理的自动化以及跨多个数据中心的资源利用率最大化。Kubernetes 项目的创始人 [Brendan Burns 曾表示](#)，针对 Borg 系统，谷歌进行了很多尝试，积累了大量经验。Kubernetes 项目的目的就是将 Borg 最精华的部分提取出来，使现在的开发者能够更简单、直接地应用。它以 Borg 为灵感，但又没那么复杂和功能全面，更强调了模块性和可理解性。因此，在 2013 年启动的 Kubernetes 项目只是谷歌公司顺应时代发展步伐，把 Borg 相关的技术

和经验予以公开和定制化的产物。接下来，我们首先从四个方面来分析 Kubernetes 从 Borg 项目所继承的内容，展示 Borg 所带来的经验。

Pod 是 Kubernetes 最基本的部署调度单元，用来定义一个或多个相关的容器。通过定义一个 Replication Controller，Kubernetes 可以将同一个模块部署到任意多个容器中，并自动管理这些容器，大大简化了系统管理的难度和工作量。其实，在 Borg 项目中已经有完成类似功能的模块——Alloc。在 Borg 中，Alloc 主要用于运行服务集群文件系统相关的日志以及数据传输工作的 web 服务器以及用户自定义的一些处理函数。Kubernetes 在提供这种一个容器运行一个应用的服务模式的基础上，又包含了一个虚拟机运行多个进程的功能。可以看出，谷歌在 Kubernetes 的开发过程中，既继承了 Alloc 的优势，又结合实际需求进行了改进，促进了 Pod 这一核心概念的成熟。

另外一方面，Kubernetes 继承了 Borg 项目中集群管理的理念。在 Borg 项目中，其所管理的对象是细粒度的任务或者机器。但是，Borg 中运行的应用程序用到了针对集群层次的重命名和负载平衡服务。正是这些服务令开发人员认识到了集群层次进行管理的高效之处。因此，Kubernetes 项目直接把 service 作为了基本操作单元。Service 是真实应用服务的抽象，对外表现为一个单一访问接口。这样，外部不需要了解后端运行情况就可以直接使用 service，方便了扩展和后端维护工作。

调试技术是 Kubernetes 从 Borg 项目中受益的另一个方面。在 Borg 项目中，由于使用人员都是谷歌公司内部员工，开发人员采用了把调试信息直接暴露给用户的方式。在遇到问题时，用户可以首先通过相互沟通来解决普遍存在的问题。此外，Borg 还提供了各种层次的 UI 和调试工具，让用户可以在面对大量数据时很好的针对自己遇到的情况进行详细分析。通过借鉴 Borg 中的成功经验，Kubernetes 提供了 cAdvisor 资源监控工具、基于 Elasticsearch 等日志聚合工具等。这些工具和机制为用户调试相关问题提供了很大的方便。

最后一方面是关于分布式系统的主节点 Master。Master 是一个控制器进程，在单元的级别上运行，并保存着所有 Borglet 上的状态数据。作为 Borg 生态系统中的核心，Master 包含了准入控制、周期性任务提交等服务。Kubernetes 在此基础上进一步提供了处理请求和管理下层状态对象的 API 服务器。类似节点控制器和复制控制器的集群管理逻辑都变为了 API 服务器的客户端。

通过以上四个方面，读者可以看到在 Borg 的设计中，谷歌公司已经采用了很多具有可扩展性的设计思路。这些想法为 Kubernetes 开发提供了成功的例子，使得谷歌可以在 Docker 崭露头角之时迅速启动 Kubernetes 项目。当然，Borg 项目也给出了一些深刻的教训，为 Kubernetes 设计提供了前车之鉴。

Borg 把 Job 作为任务 Task 的唯一成组机制。针对 Job 中的部分服务或者 Task 中的部分 Job，Borg 不能把他们进行局部成组尽心管理。针对该问题，Kubernetes 提出了 Label 的改变。Label 是用于区分 Pod、Service、Replication Controller 的 key/value 键值对，Pod、Service、Replication Controller 可以有多个 label，但是每个 label 的 key 只能对应一个 value。正是通过 Label，Service 和 Replication Controller 能够更好的与多个容器进行沟通。另外，Borg 还存在一个机器上的所有 Task 使用同一个 IP 以及配置过于复杂等问题。Kubernetes 针对这些问题都进行了优化。

可以看出，正是在过去十年间经验与教训的基础上，Kubernetes 项目才顺势崛起，迅速成为一个强大的容器管理架构。通过比较这些方面，广大开发者可以学习到谷歌公司从 Borg 走向 Kubernetes 的艰辛之路以及其在技术发展方面的前瞻性。



# 开源项目运营经验谈

作者 谢丽

从 GitHub 的用户基数来看，开源社区中有超过 850 万人在向开源软件做贡献。Brian Hyder 是 PencilBlue 的联合创始人兼首席技术官。近日，他根据自己运营开源项目的经验，探讨了如何吸引开源社区成员为项目做贡献以及如何确保项目的正常运转。

Brian 认为，开源项目运营可以归结为以下三个方面。

首先是明确项目愿景。通常，人们为项目做贡献，是因为该项目要解决的问题同他们要解决的问题一致。而且他们会希望，项目目标与他们的目标一致，而不仅仅是满足他们的即时需求。因此，非常有必要在 README 中明确描述项目愿景。但这还不够，为了使潜在的用户和贡献者对项目有信心，项目运营者还应该提供一个路线图。路线图上的时间不宜太具体，因为有时候用户反馈会导致项目运营者调整开发的优先级。比如，“多媒体服务将在 12 月份完成”要好过“多媒体服务将在 12 月份第二个周完成”。另外，贡献者/用户越多，项目愿景的要求就越高。项目运营者需要选择一个对项目而言最好的愿景，而不是对于个人而言最好的愿景。总之，项目必须满足用户的需求。

其次是制定项目贡献流程，确保贡献者总是在项目愿景范围内做贡献，保证代码的可维护性以及减少不必要的审核工作。以下是 Brian 提供的一些建议：

- 每个问题或特性创建一个分支；
- 每个分支的名称均要包含问题编号；
- 确保针对分支进行了所有的单元测试；
- 尽量不要合并自己的 pull 请求；
- 注明为什么采用那种方式，让 pull 请求成为一个学习过程。

最后是要带给人良好的感觉。网站、文档及 README 会给人第一感觉，它们可能会影响人们对相似项目的选择。代码质量确实可以体现出项目优劣，但人们在选择时通常不会首先研读代码。人们的贡献是项目的脉搏，脉搏稳定也有利于项目的长远发展。另外，可以利用一些工具增加项目的透明度。

- [Travis CI](#)：提供持续集成。
- [Coveralls](#)：提供代码覆盖。
- [Code Climate](#)：提供代码分析。
- [David](#)：提供依赖分析。

除了相应的功能外，这些工具还提供了徽章，标明了项目所处的阶段。这可以使其他人对项目运营者的代码能力产生信心。

# Roslyn 开源第一年：试炼与凯旋

作者 谢丽

[Roslyn](#) 是微软创建的一个.NET 编译器平台，提供了开源 C# 和 Visual Basic 编译器及丰富的代码分析 API，旨在使开发人员可以使用 Visual Studio 所使用的 API 构建代码分析工具。该项目于 2014 年 4 月 3 日开源。近日，在其一周岁生日来临之际，VB 项目团队经理 Kasey Uhlenhuth [撰文](#)回顾了 Roslyn 的开源之路。

据 Uhlenhuth 介绍，早在 2009 年重新设计 C# 和 VB 编译器的时候，他们就考虑到了开源。但直到 2014 年，在看到 F#、ASP.NET、TypeScript 开源取得成功后，他们才真正迈出了这一步。

开源之后，他们获得了许多来自社区的支持和帮助。社区反馈在 C# 6.0 的一些设计决策中发挥了重要的作用。比如，[主构造函数](#)、[null 条件操作符语法](#)和[字符串插值](#)均受到了社区反馈的影响。而且，社区还推动了许多与 Roslyn 相关的项目：

- [C# Pad](#) 是一个交互式 Shell，允许在浏览器中执行 C#；
- [CodeConnect.io](#) 可以实现设计时调用堆栈可视化，包括重构和搜索特性；
- [DuoCode](#) 可以将 C# 6.0 交叉编译为 JavaScript 代码；
- [LINQPad.CodeAnalysis](#) 是一个库，可以增强 LINQPad 的功能，使它更容易与 Roslyn 搭配使用；
- [Mono/Roslyn](#) 是.NET Framework 的跨平台开源实现；
- [OzCode v2.0](#) 使用 Roslyn 提供令人愉快的调试体验；
- [Scrawl](#) 是一个面向现代 Web 开发人员的轻量级编辑器；

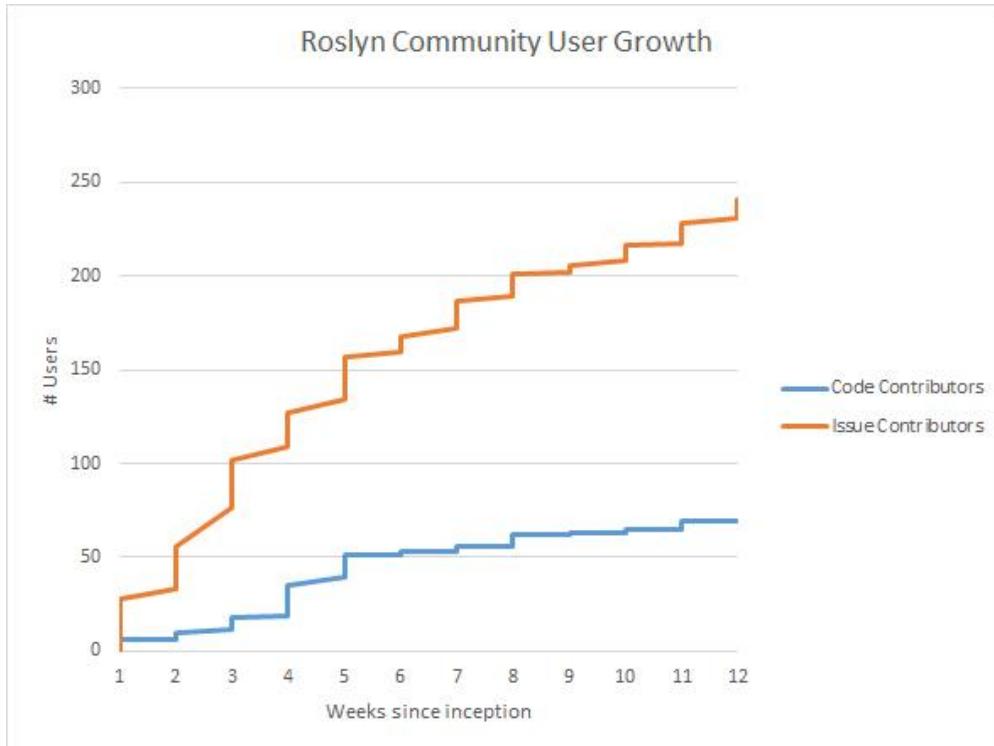
- [scriptcs](#) 是一个开源项目，允许开发人员将 C# 当作脚本语言使用，并提供一个命令行 C# REPL；
- [Try Roslyn](#) 演示了 Roslyn 的用法及如何重现一个编译 Bug；
- [WebEssentials Markdown 编辑器](#)也基于 Roslyn。

期间，他们还采取了一项重大举措，就是[将 Roslyn 源代码从 CodePlex 迁移到 GitHub](#)。

这不只是代码位置的变化，更重要的是开源模型和工作流程的变化。在迁移之前，他们采用的是一种“有限开源”模型。也就是说，问题跟踪和代码审核系统都是内部的，社区贡献的代码需要他们手动复制粘贴然后合并，而无法直接合并。在迁移之后，他们采用了“完全开源”模型，使用 GitHub 的问题跟踪和代码审核系统，通过 pull 请求提交代码，并制定了[代码贡献流程](#)。这一举措增加了社区的透明度，仅用三分之一的时间就几乎实现了社区参与度的成倍增长：

	Community PRs Accepted	Issues Filed By Community	Total Forks	Months
Source Open with Limited Contributions	12 out of 31	343 (70%)	188	9
Fully Open Source	21 out of 45	498 (28%)	363	3

Roslyn 的用户数也稳步增加，下图是 pull 请求数和问题记录数的增长趋势。



另外，Uhlenhuth 还提供了一些有关项目团队响应率的统计，旨在表明，与开发全新功能相比，他们优先接受代码，如下图所示。

PRs responded to within first hour of publishing	PRs closed in 3 days or less	Issues responded to within first hour of publishing	Issues closed in 3 days or less
72%	84%	33%	41%

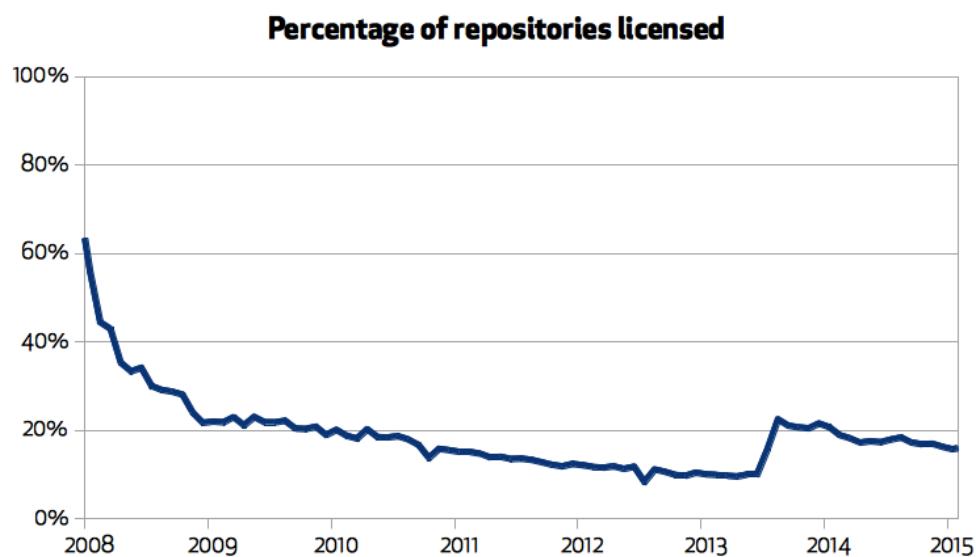
从 1 月份到现在，他们又开源了 [Scripting](#)、[“表达式求值器 Expression Evaluators”](#)、[Visual Studio 语言服务](#)等平台组件。现在，他们正在简化 [F5 构建](#)。将来，他们还有许多细节工作要做，比如，将一些历史问题从 CodePlex 迁移到 GitHub，找出一种在 GitHub 上标记问题的理想方法，将提交映射到不同产品版本的方法，等等。

## GitHub 发布开源许可证使用情况

作者 曹知渊

开源项目缺少了开源许可证，就不算完整的开源项目。GitHub 日前在其[博客上公布了](#) GitHub.com 上开源项目的许可证使用情况。

GitHub 给出了一张有许可证开源项目比例的变化图。



可以看到选择许可证的开源项目在逐步减少，但在 2013 年中出现了一个明显的反弹。因为在 2013 年中，GitHub 发布了 [choosealicense.com](#) 网站，以帮助开源软件开发者选择合适的许可证。

而用户对许可证的选择分布也不出所料。

排名	许可证	百分比
1	MIT	44.69%
2	其他	15.68%
3	GPLv2	12.96%
4	Apache	11.19%
5	GPLv3	8.88%

排名	许可证	百分比
6	BSD 3-clause	4.53%
7	Unlicense	1.87%
8	BSD 2-clause	1.70%
9	LGPLv3	1.30%
10	AGPLv3	1.05%

除去无法归类的“其他”，MIT、 GPLv2 和 Apache 占据了前三位。choosealicense.com 为许可证的选择给出了建议。MIT 是一个几乎可以“为所欲为”的许可证，如果你希望简单、宽松，它是你的不二选择。如果你关心软件的专利问题，但同样希望宽松，可以选择 Apache。如果你希望代码使用者同样能把他们的贡献分享出来，那就选择 GPL。MIT( 或类似许可证 ) 和 GPL 是开源许可证授权的两大方向，从这个数据可以看出，大部分开源软件的作者希望自己的项目能得到广泛的应用，不想给用户（尤其是企业）设置开放源码的门槛，而也有相当一部分作者推崇“copyleft”的价值观，要求修改项目人同样留下“一份拷贝”，所以他们选择了 GPL。感兴趣的读者可以从[这里](#)了解关于许可证选择的详细内容。

为了鼓励使用许可证，推动开源软件的繁荣，GitHub 发布了一套 License API，目前它提供了三种功能：

1. 列出所有的许可证；
2. 获取单个许可证；
3. 获取一个仓库所使用的许可证。

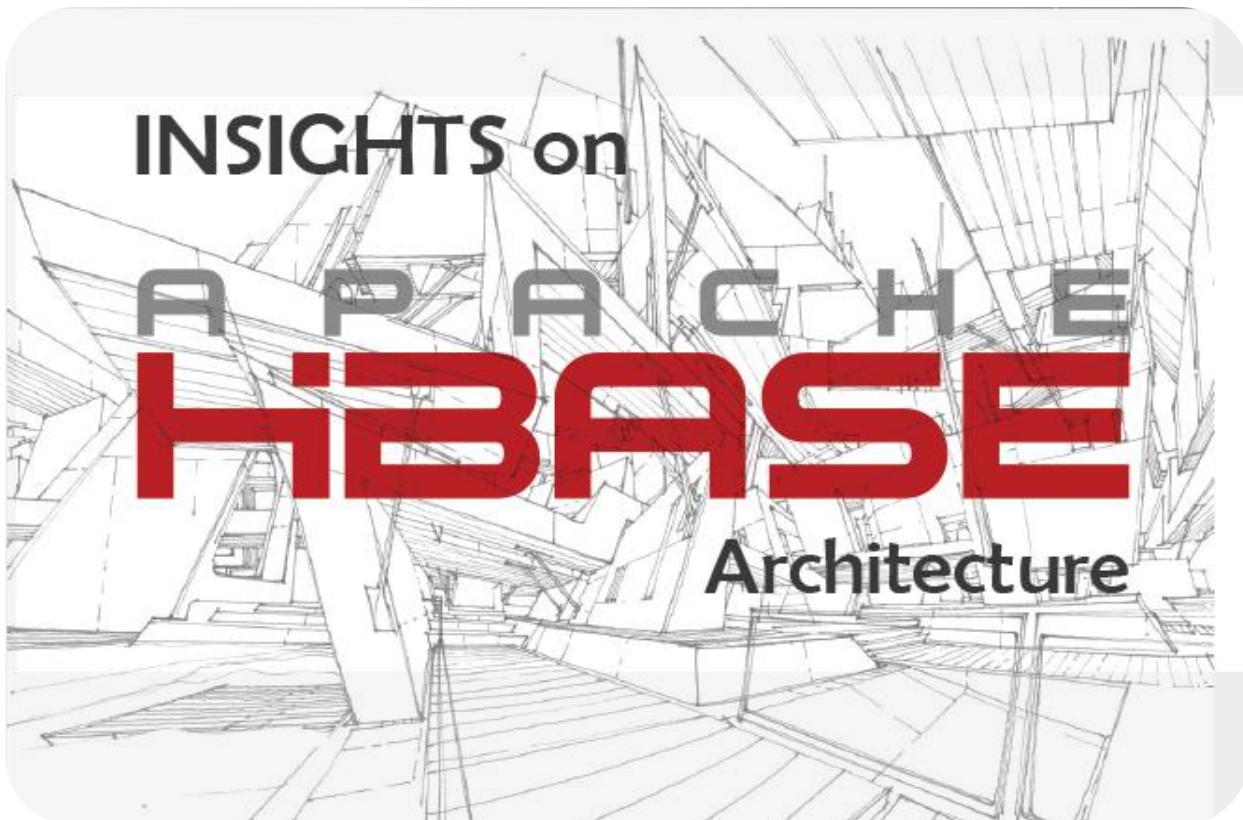
这套 API 目前还处于预览阶段，GitHub 有可能随时改变其接口。读者可以从[这里](#)查看其详细信息。



# 开源经验

# 开源经验：社区是如何管理 HBase 项目？

作者 郭蕾



3月10日，豌豆荚的张铎成为中国第5位HBase Committer。现在越来越多的公司和开发者都开始关注开源，开源正在经历前所未有的繁荣时期，但这只是开始。从整体比例来看，国内参与开源项目的人并不多，很多人都不知道如何参与开源项目。在这个背景下，InfoQ 采访了张铎，希望能深入挖掘 HBase 项目组织架构、运营、流程等方面的细节。

## 受访嘉宾

**张铎**，豌豆荚基础技术负责人，目前主要关注存储相关的技术。2010年研究生毕业，来豌豆荚之前一直在网易有道工作，从事的也是基础技术相关的工作。2014年底带领团队开发了名为 Codis 的分布式存储解决方案，并于 2014 年 11 月 7 在 GitHub 上开源。

## 第一次提交代码

作为一个历史悠久的开源项目，HBase 非常复杂，据统计，HBase 差不多有 34 万行代码，主要使用 Java 语言编写，部分模块可能会使用 C 语言。我在 2008 年底的时候就开始接触 HBase，但是提交第一个 Patch 是在去年的 9 月份，当时在工作过程中发现，HBase 有时候会丢失数据，确认自己的代码没问题后，我开始怀疑是 HBase 的 bug，定位问题后就立即对该问题进行了修复。而小米的冯宏华已经是 HBase 的 Committer，也正好是我的学长，在冯的鼓励下，我把自己发现的问题提交给了官方。

提交后不到 10 分钟的时间，就有一位 Committer 联系我让改代码的说明格式（有些地方不符合规范），简单修改后，这位 committer 立即@了负责这块代码的其他 Committer。整个提交过程非常快，HBase 方的反馈也很好，和我之前想的根本不一样。

如果要总结下第一次贡献代码的经验，我觉得应该胆子大点，不要害怕。不要怀疑自己的能力，发现问题就提交。不要担心自己的英文不好，只要发现问题就往上提，这对自己的成长也有好处。很多国外的程序员写的代码很烂，但是他们却敢于提交。而提交后又有很多牛人来帮你修改，这相当于免费请大牛当私人教练。

## 如何成为 HBase 的 Committer？

并不是第一次提交代码后就能成为 Committer，只要提交过代码的都是 Contributor 的角色。Contributor 要成为 Committer，需要持续不断的贡献代码，并且开发过新的 feature（猜测）。当代码贡献到一定程度时，项目管理者会主动与你沟通是否愿意成为 Committer，并不需要自己申请。

成为 Committer 之前，我一共提交过 30 次左右的代码，从官方的邀请信中可以看到，PMC（项目管理者）比较看重我为 HBase 1.1 提交的几个大的 feature，以及对 HBase 整个

项目的单元测试流程的改进贡献。

目前 HBase 共有 41 个 Committer，但真正活跃的不到一半，目前也没有相应的退出机制。Committer 之间主要是通过邮件沟通，没有固定的在线沟通时间。

## 代码提交后的流程

Contributor 不能直接 push 代码到仓库，他的代码需要经过 Committer 审核。如果是一个简单的改动，那一个 Committer 就可以直接做主。如果是一个比较大的改动，那就需要多个 Committer 一起讨论才能决定。如果是可能影响兼容性的改动，那需要与该版本的负责人讨论后才能确定。

HBase 的代码并没有使用 GitHub 进行管理，[GitHub 上的代码](#)只是一个镜像。Apache 基金会中一些比较老的项目使用的都是官方自建的 Git 仓库，缺陷管理工具用的是 JIRA，提交代码后，JIRA 中相关 issue 的状态会变为 Patch Available。同时，项目机器人会定时扫描是否有 Patch Available 状态的 issue，如果有，它会下载相应的附件，并通过脚本检查格式是否正确、单元测试能否通过，并把结果发回到 JIRA。当 Patch 要合并到某个版本之前，该版本的负责人会重点进行测试，包括功能和性能上的。

## HBase 的项目架构

HBase 的项目直接负责人称为 VP，VP 全职负责这个项目。VP 下面是几个 PMC，每个版本的 release manager 一般是某个 PMC。PMC 下面就是 Committer 了。一般情况下，一个 Committer 会对 HBase 的固定负责某个版本的某几个块功能模块特别熟悉，所以当 Contributor 提交代码时，项目组是有审核该代码的第一负责人。HBase 项目组会优先让对该模块比较熟悉的 Committer 来审核，这个 Committer 的意见也是最重要的。

HBase 主要的代码贡献者都是 Cloudera 和 Hortonworks 的员工，他们都是全职负责

HBase，甚至 HBase 中写文档的人都是 Cloudera 的员工。Cloudera 和 Hortonworks 都是基于 HBase 的商业公司，他们同时维护开源的 HBase，但同时也都有自己的商业版本。

## 社区分歧

每个开源项目都会遇到技术方案分歧的情况，同样 HBase 也有。HBase 在这方面并没有好的解决方案，每次讨论这样的问题时都会分为两派，大家都在说自己的解决方案以及优势，但是永远也没有结论。目前也没有相关的投票机制，比如谁的得票多就听谁的，因为投票很容易导致产生更大的分歧。

其它开源项目也没有好的解决方案。如果社区比较融合，大家都抱着解决问题的心态来看待这样的分歧，那还好办。但如果大家都比较偏执，一派人坚持要这样做，另外一派人坚持要那样做，那这样下去稍有不慎社区就可能分裂，这已经有先例了。再或者就像 HBase 一样先挂起这问题，但也不是长久之计。其实分歧问题和社区文化直接挂钩。

## 开源谈

如果只靠个人无私奉献，开源项目很难发展起来，更不用说建立生态圈。如同公司一样，开源社区需要有人来推动、管理和运营，开源不仅仅是代码本身。比如前面提到的开源文化，这完全是需要有人来引导的。

豌豆荚一直比较崇尚互联网「开放」「平等」的价值观，也很支持重视开源技术。公司决策层领导层也非常重视员工在技术方面的长期积累，所以也允许我有一些比较自由的时间来研究 HBase，做一些贡献，也不要求这个研究马上得在多少天内就贡献多少代码或者做出多少新 feature。我觉得豌豆荚对于基础技术的长期积累还是很看重的，而且很有耐心。纵观一些好的公司，一定会多给员工一些属于自己的时间探索新的东西。如果每天都很忙，不停的在加班，那什么时候去进步了？员工的成长甚至跟不上公司的成长，长期来看反而会拖累公司。

# Apache 软件基金会总裁：Docker 是善意的独裁者

作者 郭蕾

Ross Gardler 是 Apache 软件基金会现任总裁，在开源技术领域拥有丰富的经验，长期致力于开源技术的推广和开源社区的治理工作。Ross Gardler 是开源项目的积极贡献者，指导了大批开发者融入开源社区理解并遵从开源社区规则。在开源社组织的一次媒体见面会中，InfoQ 有幸采访到了 Ross Gardler 先生，听他阐释了 Apache 的社区治理之道。本文根据采访内容整理而成。

## 没有领导者，但有驱动者

Apache 曾经调研过 200 个大型开源项目，结果发现排名前九的开源项目不管是从参与人数还是活跃度上，都是十倍于其它一百多个开源项目的，并且这几个成功的项目都是由非盈利组织治理的，而非某个厂商。随着开源项目的发展以及参与人数的增加，开源项目的社区治理就变得非常重要，所以 Apache 社区的理念一直是社区重于代码。要治理好一个社区，需要两个方面的东西，一个是法律框架，一个是治理模型。在 Apache 中，法律框架其实是指开源许可证，许可证非常重要，它能够很好地保护代码贡献者的利益和权力。但是开源许可证只能提供一个合法的法律框架，并不能持续推动开源项目的发展。治理模型是由一些规章流程以及文化组成的，Apache 会制定一些最为常用且最优化的社区流程，以为社区未来长期的发展找到一个最优化的发展方法或道路。流程是必须的，但是 Apache 也明白社区应该是以人为本而不是以流程为核心，所以他们并没有所谓的领导者来对社区的参与者发号施令，却有驱动者、激励者，来帮助社区的每一个人找到自己所需要的东西，从而满足需求。

## 不通过投票的机制来决策

在 Apache 社区中，决策是通过统一共识来决定的，而非投票机制。虽然达成共识是一个

非常耗时和耗精力的过程（需要反复听取意见并讨论，也可能是争论），但 Apache 还是坚持这样做，因为他们认为投票会导致分裂。这样的做法称为“默认共识”，也就是默认大家都是为了社区共同的利益来做某一行代码的修改，而对于谁对谁错，就让代码去说话。

那么 Apache 是如何实现开源社区代码的沟通讨论呢？一个软件并不能满足所有用户的需求，所以社区可能会基于同一套开源代码，并根据自己的个性化需求开发不同的软件。但是随着衍生软件的不断更新，它们之间的共同代码可能就会越来越少，为了避免过于分化，Apache 会采用一定的措施让它们进行代码融合。假设在社区中有两种方案，第一种方案有更高的效率，但是第二种方案提供了更高的灵活性。就选择第一种方案还是第二种方案，无休止的争论可能只是浪费时间，这个时候让代码说话最合适。随着两种方案的演进，胜负自然会有，然后 Apache 再出力干涉，设法把它们融合起来。但也并不总是这样，有的时候两种方案可能永远无法再融合，这种情况也是可以接受的，也正体现了社区的多元化。

## Apache 项目的组织架构

Apache 软件基金会和项目的运作是分开的，基金会主要是为项目的运行提供保障，比如法律帮助、代码托管。而各个开源项目都是独立运转的，在技术细节上他们有完全的自主权。一个典型的 Apache 开源项目，通常都有四个角色，或者四类人：第一类人就是用户。用户表达自己的需求，想要什么功能，并提供一些反馈。第二类人就是贡献者，他们会根据实际情况来贡献代码，并解答用户的一些问题。当贡献者参与项目足够长的时间，并充分了解项目之后，他就会升级为 Committer，Committer 和贡献者是不一样的。贡献者所贡献的代码是补丁，但是要应用这个补丁，就必须经过 Committer 的同意。最后一个角色，叫项目管委会。项目每一个参与者都有表达意见的通道，但是当争论不休的时候，项目管委会就会出面做出最后的决定。

如果项目管委会被认为是违背了 Apache 之道，比如违背了大部分的项目参与方的意见，

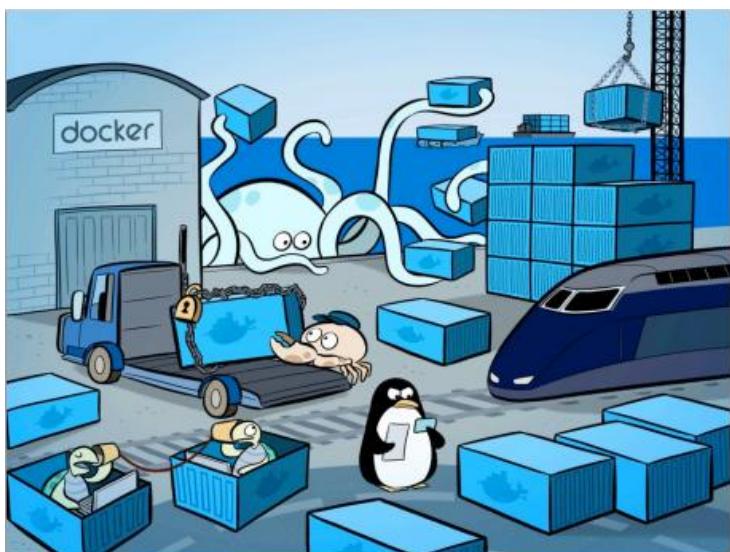
擅自作主，Apache 就可以把这个决策再升级到软件基金董事会，让董事会来做最后的裁决。

## 开源项目中的二八定律

纵观 Apache 下的开源项目，二八定律也是成立的：80%的代码都是来自 20%的贡献者。

反过来也是这样，20%的工作都是由 80%的人来做的。在西方的开源社区中，基本上是 20%的人做了 80%的工作，并且这 20%的人都是被雇佣来做这些事情的。而那些做了 20%工作的 80%的人，他们都是利用自己的周末、闲暇的时间来做的。

## Docker 社区有善意的“独裁者”



从治理方式上来说，Docker 和 Apache 之道是不一样的，如果是在 Apache 的开源项目社区中出现争议，最终可以由 Apache 软件基金董事会来统一做最后的裁决。但如果是 Docker，他会有一个叫做善意的独裁者的角色，也就意味着你提了很多的

建议，但是如果这个“独裁者”觉得不行，他一个人就可以做决定。当然他会听取社区的意见，所以应该把他叫善意的“独裁者”。这是二者最大的区别。

总体来看，在具体的执行过程中，Docker 和 Apache 的开源项目本质上是一样的，都是要促成共识。当然在极少见的情况下，如果发生了争议，Docker 可以独裁，但 Apache 不能。

## 如何促进开源在中国的发展

从 Apache 基金会的发展曲线中可以看到，西方的开源项目在一开始的很多年其实并没有大的突破，一直都是铺垫期，而中国现在正处在这一铺垫期。中国可以学习国外的经验从而缩

短铺垫期，争取尽快进入发展期。另外，在西方的开源项目发展初期，就有爱好者参与推动它的发展，随着时间的推移，这样的人就会越来越多，大家的意识也就会改变。目前中国就缺少这样的人或者组织。

在西方的开源历史上有三个大的里程碑，第一个里程碑是开源的 MySQL 数据库；第二个里程碑是第一个达到了数十亿美金身价的开源公司 Red Hat；第三个里程碑是最近的一家基于开源的成功的创业公司 MongoDB。而在中国，很少有这样的可以借鉴的成功案例，中国需要有这样的公司或者开源项目来教育市场和用户。

# 评价社区经理的绩效

作者 Jason Hibbets 译者 曹知渊

在一个开放型组织中，像社区经理这样的特殊角色，考核起来是有点难度的，尤其是和其他一些岗位比起来，因为其他岗位都有明确定义好的绩效指标、工作目标和输出结果，而社区经理没有。回顾我过去六年的工作经验，我跟我的上司紧密合作，就我的工作目标和内容保持共识，力求做好一名社区经理，让自己所扮演的角色发挥最大的影响力。

在《哈佛商业评论》的“[管理难以衡量的绩效](#)”一文中，Red Hat 的 CEO Jim Whitehurst 讲述了如何敏锐地发觉那些难以衡量的工作结果给组织带来的价值。这篇文章的主旨引起了我的共鸣，因为我正在思考我在 Opensource.com 担任的角色。我知道不是每个人都在 Red Hat 有和我一样的经历，但我想来分析一下 Jim 对于绩效评估的观点，以及我的做法。

## 衡量难以衡量之结果

首先，Jim 谈到了如何衡量难以预测的结果：

衡量某些人的工作结果，不是像数数他们写了几个小插件这么简单，而要看他们如何管理一个团队，如何影响他人或帮助他人更好地合作。这些工作应该如何评估？

在我担任的 Opensource.com 社区经理角色中，我没有直接管人。我们有内部团队，有[社区主编团队](#)，还有一般的[贡献者](#)团队，我的工作是在这些团队之间担任一个“影响者”角色。我没有直接的行政权力给某人下命令。但是我的领导风格能很好地融入这种角色，我经常会提一些对社区有利的任务和目标。我努力向大家解释长远利益在哪里，为什么我请求大家做的这些事情，可以帮我们获得这种长远利益。

举个例子，我们会给我们的社区主编们提供社交媒体方面的培训。我们召集一个一小时的视频会议，在会上分享一些我们正在使用的最佳实践和技术，教会我们这些关键的贡献者如何在社交媒体上把他们的影响力发挥到最大。那些与会的人们获得了重要的经验，学会了如何利

用社交媒体来履行网站的使命。我无法强求所有的社区主编都参加，但那些参加的人，着实从中获得了巨大的收获，因为在会上，我们分享的都是过去几年中，我们在运营 Opensource.com 的各种社交媒体帐号过程中所获得的知识和运用的策略。

像这样的培训会，你如何衡量它的影响力？我们没有时间每天去关注每个社区主编的社交媒体帐号，看看他们在干什么。实际上，我们观察参加培训的人数，然后持续监控那些社交媒体相关的数据（粉丝数、互动程度、转发数、访问流量）。经常使用社交媒体的人都知道，要衡量它带来的影响，取决于你想通过它达到什么目标。衡量一个社区经理影响社交媒体的能力，结果很难预测。

事实上，社区经理做了很多社交媒体以外的琐事。他们和社区的成员有很多互动，而这些成员一般很难去跟踪他们在干什么——这就是为什么我们一定要找到衡量社区经理影响力的方法。我的上级和我都认识到，很多一对一的互动——电子邮件、Twitter 帖子、私人短信——是很难在日常工作中记录下来的。所以我们把精力放在更宏观的目标上，比如招募新的社区主编，或把新的作家介绍到我们社区。我的责任就是把那些日常琐事，体现到我们更宏观的目标上。

## 和上司达成共识

Jim 在文章中也谈到了如何和自己的上司达成共识：

我们认为有一件事很重要，就是必须确保下属和上级之间，就社区经理的职责和工作结果预期达成共识。

我和我的上司用两种机制来完成这个目标。首先，我们俩每周会进行一次面对面的会谈。在这个会上，我可以随意地提出问题、关切、目标，或亮出一些新的创意。我的上级也会提出一系列他希望核实的事情来讨论。更重要的是，我们会利用这个机会来讨论那些有挑战性，或者需要调整的目标——这类目标比你想象的多。有时候，这就好比我们俩在读一本书，我们需

要确认我们读的是同一章节，然后共同翻到同一页上。

第二种机制是，我们每周会跟整个团队开一个会，讨论未来 30 天、60 天、90 天的计划。

我们用这个机会来讨论中长期的目标——这些事情经常由于我们平时工作繁忙而无法完成。我们为我们想完成的事情，比如编写一个新的资源网页（想想 “Linux 是什么”），设定合理的目标和期限，我们利用周会来核实进度，共享我们的成果，并且做出调整。

给大家一个“达成共识”的例子，我想记录下我的工作时间都花在哪里。我经常会在各种开源大会上演讲，或者参加 Raleigh-Durham 区的本地会议。我上司和我开玩笑地谈起——要给我配备一个 GPS，倒不是为了记录我每时每刻的行踪，而是想从中看出我作为一个社区经理，一共做了哪些事，去了哪些地方。你很难用常规手段从社区经理身上获取这些信息，比如绩效评价。

整年内我们记录了前面所说的 30 天、60 天、90 天的目标的执行情况，用于评价绩效。这样做可以更容易地回顾那些中长期目标是否达成，从而更宏观地来评价绩效。

## 抓住机会

我想讨论的最后一个问题是，也是我为什么如此尊敬我的上司的原因。Jim 写下了他最后一条观点：管理者关注机会，而不是整天在那儿算分数。

对于我这个角色来说，这个观点非常正确。但是，虽然我和我的上司相处有几年了，我直到最近才意识到这一点。

在庆祝完 Opensource.com 成立五周年后，我有机会仔细回顾过去几年中的成就和我担任过的各种角色。我意识到我的上司总是在寻找下一个机会，用一种鼓励的方式，推动我去探索，去执行。

就在最近有个例子，是关于一个我跟了将近一年的项目。我们曾经讨论了很多关于更新 Opensource.com 网站的导航方式的事情。整个团队觉得旧菜单已经完成了这个网站的最初使

命，那就是组织好有限的几个领域的话题（商业、教育、政府、健康、法律和生活）。随着 Opensource.com 网站的启动，我们进入了很多其他领域，比如开放硬件、人道主义自由和开源软件( HFOSS )、DevOps 以及其他很多——修改旧菜单，塞些新代码进去不是那么简单的。要改变很难，不是吗？

几个月来我们一直致力于更新网站架构和导航，思考着我们要如何去适应当前的内容和未来的需求。我和我的上司讨论过很多次，最后，这件事情的重要性就成了讨论焦点。我成功地向我的上司展现了一个机会，而他也一直鼓励我追求必要的变革，并且和自己的团队一起做出正确的决策。

Jim 谈到了管理者应关注机会——在我身上，我觉得这非常正确。多年以来，我也学到了，作为下属也要有勇气去呈现他们看到的机会，这样他们的上司才能进一步去评估这些机会。如果你有持续改进的想法和创意，别害怕，大胆说出来。

## 如何衡量社区经理的影响力？

衡量社区经理的影响力，没有放之四海皆准的法则。Jim 就是想表达这个观点，他写道，“某些人能给我们的组织和我们参与的社区带来影响力，而传统的绩效评价打分则漠视了这种影响力。”

不变的只有变化本身。如果下属能意识到这一点，和上司步调一致，那么到年终评价绩效的时候，得到出色的成绩也不要吃惊。为什么？因为持续的沟通是至关重要的，给像我这样的人自由度，让我决策一线的事情，可以使一个开放型组织茁壮成长。那些一对一的会议真是好时光啊，我从中获得了方向、解释，甚至信心。

在设定目标的时候，不要好高骛远。我们那些定下 30 天、60 天、90 天目标的会议，是非常灵活的，当我们有了更宏大的目标，我们一边走一边调整。在做出这些调整时，积极地开展讨论，定下合理的团队目标和绩效评价手段。灵活性、沟通和协作是公司能蓬勃发展的关键。

# 为什么开源适合 LinkedIn

作者 曹知渊

开源软件不再局限于解决小问题、底层问题，否则也不会有公司花代价去创建开源软件。如今，创建、部署开源软件的社区日渐成熟，他们代表着世界上最大的几个技术公司。为什么有此变化？为什么公司应该把一些精力花在开源软件上？为何要开源？

初衷就是开源软件能使你的工程师变得更强。随着他们的作品呈献给整个社区，他们的技艺也在精进。

## 他们能写出更好的软件

为别人工作，比起为自己公司工作，更能写出好软件，这听起来有点矛盾，但事实上却很有道理。当开发人员写一个“内部”软件的时候，他们倾向于在一些环节上偷工减料——我也是如此——尤其是在文档、代码的可读性和可重用性，以及编写完善的测试用例方面。但开源社区有多种选择，如果你的代码太晦涩，他们不会有耐心去搞明白它究竟在干什么。而公司内部则没得选。

在开源项目中，开发人员的名字会和他们创建的软件绑在一起，整个社区都在审视他们。这就好像给网上的代码和声誉配上一张面孔，每个人都能看到他们的设计取向和 bug。这大大激发了他们去完善自己的软件。开发人员都希望自己的名字和设计精良的美好事物联系在一起。

每个优秀的工程师永远在不停地学习，他们想跟上自己专业领域的发展步伐。直面那些自己公司之外的开发社区，能帮助他们看到最新的趋势，以及帮助他们学会如何利用社区带来的价值。而且，如果他们自己是一个开源项目的所有人，他们的“技术领导”技能也能得到长足的进步，因为他们必须要选择接受或拒绝来自社区的补丁。一个成功的项目，应该是贡献者齐聚，各种观点争鸣的社区。但不是所有的问题都有更简洁更好的技术解决方案的。他们必须决定选择什么，放弃什么，这很难，但在这种困难中，他们锻炼了新的技能。很多开源项目的留言板

上充斥着偏激言论，口水战经常处于一触即发的边缘。

## 从公司的角度看，它能帮助打造“研发品牌”

研发品牌意味着一种公司特有的开发能力的烙印，就好像给产品贴个标签，上面写着“这是我们制造的”。打造这种品牌有很多方法。开源是一种很好的和其他开发者共享代码的方式。它可能无法完全揭示一个公司的内部运作机制，但肯定能展现出跟这个项目相关的各个方面。这也是资深员工给应聘者的一个展示窗口，一个企业向未来雇员发出的明确信号，即，如果他们来这儿工作，不仅能获得个人技能的提升，并且能充分利用那些他们喜欢的软件系统，而不用重新发明它们。

当然，在那么多支持的观点之外，也有一个反对的声音，那就是，开源并不适用于所有公司，也无法解决所有问题。

## 不要因为缺少人力而开源

在早期这是个常见的错误观点。这个观点的逻辑是，如果我有一个优秀的项目需要额外的帮助，那么开源它会吸引全世界的开发者为它做贡献。这是错误的，原因有二：如果项目成功了，它会从外部社区收到无数的需求，需要耗费额外的精力和时间把代码写出来，而这些需求很有可能跟公司创立这个项目的初衷相悖。如果项目没有成功，前期花了大量的资源去包装它就全部白费了。而且，要判断为项目写代码和提交补丁的个人或社区的能力好坏也很困难。毕竟在外部世界什么都有可能。要做出成功的开源项目，需要投入大量时间来开发、监管和培育。



## 不要为了开源而开源

这个问题我通常称之为：“做个好公民”，多做做功课。在创立任何项目前，做个好公民，调查一下是否已经有社区在致力于开发类似的解决方案了。然后花点精力去评估一下，是否给已有的项目贡献代码会更好？通常，使用现有的项目并做出自己的贡献，价值远胜于重复发明轮子。

把糟糕的项目拿出去开源，会产生负面影响。人们审视这个项目后会疑惑，既然已经有了一个很棒的开源解决方案，为什么这个工程师或团队还要搞出这么一个水平低劣的东西来。这对于上面所说各种开源的好处，都是一种伤害。

## 如何创建成功的开源项目

这很不容易。老实说，LinkedIn 早期在这上面也是磕磕绊绊。我们早期学到的一个教训就是，开发团队不能为了开源，简单把代码丢给社区，然后就停止创新了。很多人认为只要一开源，社区会帮你带领这个项目继续前进，这种情况偶尔也会发生。但大多数情况下，你作为项目的缔造者，有责任保障它在开源的模式下继续发展好，并且你要尽可能多花时间在上面，就把它当作还是你公司内部的项目。

我们学到的另外一个教训是，多半的小型项目，应该集成到大型项目中去。LinkedIn 的第一批开源项目是一些小型的搜索组件，我们从自己基于 “[Lucene](#)” 项目的主搜索引擎之上创建了它们，但我们却没有和 Lucene 项目的开发者社区保持同步，大多数组件都没有集成到 Lucene 中，最终被丢弃——其实它们中的很多项目都很出色。Apache 社区则是一个极好的正面例子。单独的开源项目是不错，但是如果能让这个项目抱上 Apache 这条大腿，那么请不要犹豫。一个成熟社区中的开发人员如果有激情去开创新的项目，成功率要高得多。

后来我们创建 Kafka 项目时，从过去所犯的错误中吸取了教训，明智地选择了投入资源，

最终它成了我们的最佳开源项目。

最后，如果一个公司决定开源某个软件，他们必须要有心理准备，最后不得不跟这个项目分道扬镳。我第一次不得不给我的一个项目单独拉出分支的时候，感觉就像是杀死了自己的孩子。做出这个决定后，我整个周末都没睡好，感觉很惭愧……感觉自己像个叛徒。但是后来我明白了：一个项目一旦开源，它就有了自己的生命力。它很有可能演进成为一个你根本不想要的东西，并且不再和你的软件系统兼容。这一刻到来时，你别无选择，只能拉出分支单干了。这是一个自然现象，时不时会发生。你的孩子长大成人了，让大人重新过回他们自己的生活吧。

但这不应该把谁吓跑。开源是一件非常值得做的事情，创建、管理、参与开源项目，其乐无穷。

# 360 的开源软件使用以及开源文化构建经验

作者 郭蕾

奇虎 360 企业安全从 2011 年到 2015 年，经历了几次服务端架构的变迁。从一开始的软件自研到最后的全部使用开源软件，360 企业安全部真切体会到了开源软件带来的好处。另外，360 也通过各种各样的方式来构建自己开放、透明、平等的企业文化。InfoQ 编辑采访了奇虎企业安全高级工程师温铭，听他分享了 360 内部的开源软件实践经验以及开源文化构建经验。同时，温铭还将在 ArchSummit 全球架构师峰会上分享题为“[开源文化对 360 天擎架构演进的影响](#)”的演讲，敬请关注。

**InfoQ：能分阶段介绍下 360 企业安全的服务端架构的演进情况吗？**

**温铭：**360 企业安全服务端的架构演进主要分为三个阶段。

第一个阶段是以自己开发的组件为主。我们用 C++ 自己写了一个简易的 Web Server，并在此基础上面衍生的一系列工具。页面的开发，我们没有使用成熟的 PHP 框架，而是自己写了个路由。这一阶段产生的问题就是基础组件不稳定，跟不上产品功能的快速迭代，而不稳定的产品，企业用户也不能接受。开发付出了很多努力，但一直没有找到适当的节奏。

后面有几个开发的同事觉得要跳出这个糟糕的循环，于是就在新模块的开发中引入了 OpenResty 和 Yii，并和自研的组件并存。这样带来的明显好处是效率的提升，开发有精力来完成了前后端分离，QA 也从完全的黑盒测试中脱身，搭建了服务端的自动化测试和性能测试。这些改变为后面重构打下了坚实的基础。

在第三个阶段，我们重构了产品，并统一了服务端的技术架构，同时相关的功能组件全部换用成熟的开源软件搭建。比如我们使用 OpenResty 来搭建整个服务端，周边工

具用 Python 和 Go 来完成，报表和数据分析采用 ElasticSearch。这样开发就可以更专注于产品功能实现和开源软件的深入学习，而不用担心基础组件的稳定性。

**InfoQ：你提到，一开始的时候你们的开发人员都在自己『造轮子』，那这样的方式有什么问题？在自研和开源软件之间，你认为应该如何选择？**

**温铭：**很多时候，『造轮子』的原因是开发人员没有找到合适的开源软件，或者在技术方面自视过高。盲目的自己从头开发组件，时间成本、稳定性以及后续的维护都是问题。我认为『造轮子』的前提是，现有的成熟开源软件不满足你的需求。比如 360 云查杀对性能要求非常高，而当时没有开源软件符合需求，所以我们就在 LevelDB 的基础上面开发自己的 Key-Value 数据库。而对于大部分的服务端开发来说，开源软件足以应付相关需求。

**InfoQ：开源软件的选型上，你有什么好的经验吗？**

**温铭：**现在开源软件发展非常快，新技术层出不穷，在技术选型的时候，你会面对很多的选择。不管使用什么开源软件，在服务端的架构层面，你都需要做到各个组件像乐高积木一样，没有耦合并且可以方便的进行更换。在你产品的第一个的版本里面，最好选择成熟稳定、自己团队熟悉的开源软件，而不是性能最好的，因为这时候你需要快速的出产品和快速迭代，性能并不是最重要的；在随后的版本中，你可以通过性能测试的各项数据，来决定使用哪个开源软件。

除了性能的考虑之外，开源软件自身的发展也需要考虑：对开发者是否友好、修复 bug 的速度、版本迭代的周期等。

**InfoQ：开源软件的使用过程中，经常会涉及到需要修改开源软件的代码。那在内部版本和社区版本之间，你们是如何同步的，有什么好的经验吗？**

**温铭**：360 企业安全服务端有一个特殊的产品需求，是其它产品的服务端开发不会遇到的，就是需要支持 Windows 平台。因为很多企业还是希望软件运行在 Windows 上面。所以我们会对一些开源软件进行修改，以增加对 Windows 平台的支持。我们刚开始的做法是下载源代码，然后修改，但造成的问题是没法快速跟进开源软件版本的更新，也没法把修改的代码回馈到社区。现在我们会 fork，然后把改动 pull request 向社区来回馈代码，同时要有完善的测试案例和文档，代码要符合软件本身的编码风格。

**InfoQ：你怎么看开源文化？360 是如何构建开源文化的？**

**温铭**：360 在 GitHub 上面开源了多个自己开发的软件，公司内部有技术评级以及定期的技术嘉年华，鼓励工程师主动分享技术并参与到开源软件中去。在技术团队中，透明和平等的文化，最适合各种技术的成长。我们团队有一个不成文的规定：AKA ( all know all )。比如在技术选型上，我们会出一个大概的架构，然后发给所有开发讨论，由于我们是一个大杂烩的技术团队，有人说“PHP 是最好的语言”，有人是 Python 的粉丝，还有人推崇 Go，还有人坚守 Windows 平台，所以各种讨 ( chao ) 论 ( jia ) 后才确定最后的选型。

**InfoQ：要使用开源软件，是不是需要团队成员对这些开源软件有足够的了解？**

**温铭**：即使不去修改开源软件的代码，团队成员也需要对这些软件的内部实现有深入的了解，才能用到适合的场合以及做参数的调整。我们每周会有定期的技术分享和 code review，来保证团队每个人都知道团队使用了哪些技术，如何更好使用这些技术。

### InfoQ：团队在参与开源软件方面，你有什么好的经验可以分享？

**温铭**：最重要的是要团队成员有主动性和热情参与其中。并不是贡献了 patch 才算参与到开源软件中，在实际的开发中解决了问题，总结分享出来，也是一种方式。我们团队正在 GitBook 上面写关于 OpenResty 和 ElasticSearch 的书，分享我们的一些实践。这也是一种参与开源的方式。

## 受访嘉宾介绍

**温铭**，一直在互联网安全公司从事高性能服务端的开发和架构，用各种技术手段打击木马传播和互联网欺诈。目前在奇虎用互联网技术帮助企业提高安全防护。



AWS 引领云服务

没错，

您想找的关于 AWS  
最全最权威的中文资料都在这里！

客户案例

学习资料

白皮书下载

演讲访谈

活动推荐

技术资讯



《开源启示录》第一季

出品人：霍泰稳

总策划：崔康

本期主编：郭蕾

流程编辑：丁晓昀

封面设计：王硕映

读者反馈/投稿：editors@cn.infoq.com

商务合作：sales@cn.infoq.com