

架构师

3月 ARCHITECT



特别专题

探索算法和推荐系统

一次推荐算法的普及性讨论

百分点推荐引擎——从需求到架构

从赌钱游戏看PageRank算法

QCon北京 2013

参加QCon北京2013的十大理由

云端之道：网易有道蒋炜航专访

NoSQL的现状

深入理解Java内存模型（四）

Heroku危机带来的启示

Flash MMORPG开发中基本原则

Hadoop的现在和未来



促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 |

本期主编

霍泰稳

总编辑

贾国清

美术/流程编辑

水羽哲

发行人

霍泰稳

读者反馈

editors@cn.infoq.com

商务合作

Sales@cn.infoq.com

15810407783

卷首语

从客户驱动到技术驱动

又是一个企业家的苦恼，“现在是苦于奔命，努力迎合客户，可是最终发现还是被客户甩的远远的”，从年前到年后，我想我至少听到有3个公司的负责人，而且是有代表性的大中型公司，这样描述他们所遇到的困境了。初听此言，觉得不可思议，“顾客是上帝”没有错啊，再仔细思考一下，越发觉得互联网时代的软件公司要保持基业长青，确实不易。从客户驱动，到技术驱动，迈过的不仅是一个门槛，可能还是一个时代。

在 MIS (管理信息系统) 时期，不论你是用 Visula ++，还是 Delphi，还是 Visual Basic，甚至 PowerBuilder，做出一套工作流通畅，访问速度还不错的系统，总是可以的，技术上的差异性并不大。究其原因，大多数的 MIS 系统是企业内使用，数据量较小，使用稍微高档的服务器基本就可以解决问题，对技术人才的要求也不是那么高。步入大数据时代，云计算时代，显然已经不再是开发工具或者语言的较量。在此前和推荐系统领域的一个朋友聊天时，他讲到一个段子，说有个大型电商公司的算法专家，将原来广告系统中的参数做了微调，结果网页上广告的点击量就上升了近 40%。而再看国外现在冉冉升起的创业公司如 Pinterest、Asana 等，也是从 Facebook 技术团队走出来的少数技术大拿在短时间内创建的。技术或者技术人才的价值在这些案例上表现的淋漓尽致，如果按照市场估值计算，他们在短短一两年所创造出来的价值可能比很多传统软件公司辛辛苦苦数十年的的估值还要高。

相对于客户驱动，技术驱动的优势更加明显。如果简单来做个比喻的话，客户驱动相当于被动地改变，而技术驱动则是在主动地创造。在技术驱动的今天，很多老牌的企业，因为跟不上变化，渐渐被时代所甩掉，或者破产或者被并购。比如叱咤一时的柯达，虽然最终它们认识到数码相机的重要性，也有了好的产品，但是为时已晚，冠名多年的柯达剧院也被迫易名；已经有上百年历史的摩托罗拉，那个曾经制造出 V 系列多款经典手机的无线通信厂商，只因为未能跟上智能手机的步伐，不得不被 Google 所收购；诺基亚更是，虽然赶上了智能手机的浪潮，但选错了操作系统阵营，结果现在处处挨打，被苹果、三星等后来者居上。

不论是客户驱动还是技术驱动，都没有错，其背后所暴露的问题实质都是对人才，尤其是技术人才的重视。我想这也是现在为什么很多公司每年在做预算时都特别预留出大笔培训预算，不论是请进来，还是走出去，都越发重视人才培养的主要原因之一。一步错，步步错，一步跟不上，步步跟不上，深厚的技术储备、人才储备，可以有效避免错误的发生。别说了，³从现在起，重视你的技术团队吧，尽力将自己变成一个技术驱动的公司吧，短时的阵痛可以换来长久的利益，还犹豫什么？

目录

[人物专访]

虚拟座谈会：PaaS 的路由延时问题与架构设计思路 7

[热点新闻]

如何成为强大的程序员？ 13

Heroku 的教训：糟糕的负载均衡 + RoR 单线程 = 糟糕的性能 17

jQuery 作者 John Resig：WebKit 就是浏览器引擎中的 jQuery 21

Facebook 如何实现 PB 级别数据库自动化备份 25

[特别专题]

一次推荐算法的普及性讨论 29

百分点推荐引擎——从需求到架构 34

从赌钱游戏看 PageRank 算法 41

[本期专栏]

参加 QCon 全球软件开发大会（北京站）2013 的十大理由 52

云端之道——QCon 北京 2013 云计算专题出品人网易有道蒋炜航专访 55

[推荐文章]

NoSQL 的现状 60

深入理解 Java 内存模型（四）——volatile 70

Heroku 危机带来的启示 82

访“魔道”团队：Flash MMORPG 开发中的“五个基本原则”	86
Hadoop 的现在和未来	93

[新品推荐]

JavaScript 成为 GNOME 的首选语言	105
Facebook 引入 Chef 来管理其 web 层	105
Xamarin 2.0 带来新的 IDE、支持 iOS 的 Visual Studio 插件和组件商店	105
Miguel de Icaza 对 InfoQ 详细介绍了组件商店。Cloud Foundry Core——保持云应用的可移植.....	105
微软为 Windows Azure 创建基于 Linux 的虚拟机的目录	106
Android 运行 Windows 应用，黑莓运行 Android 应用，以及 Ubuntu 手机的消息.....	106
MyGet 为 CodePlex、GitHub 和 BitBucket 提供免费构建服务	106
Twitter 发布基于组件的轻量级 JavaScript 框架——Flight	107
Visual Studio 拥抱 Git	107
Meteor 0.5.3 发布：改进的性能与实时的反应式更新	107

人物专访

虚拟座谈会：PaaS 的路由延时问题与架构设计思路

作者 [杨赛](#)

最近的 [Heroku 大客户暴走风波](#)引发了云计算领域的又一波争论。在云平台和云平台上的用户规模不断增加时，困扰了各种企业架构、互联网架构的伸缩性问题也被摆在了云服务提供商和用户的面前。

我们能从 Heroku 的事件中学习到什么？大规模的 PaaS 架构应该如何设计？如何确保及时的发现问题、响应问题？开发者如何保证自己在使用云服务的时候不被忽悠？云计算的架构设计者能否借鉴传统企业级架构和互联网架构的一些经验？

为了寻求这些问题的答案，InfoQ 从今天开始会组织一系列主题为《[云服务如何面临业务增长的挑战](#)》的虚拟座谈会。本期为第一期，主要讨论如下两个话题：

1. Heroku 本次反映出来的路由系统延时问题，背后究竟有哪些方面的原因？
2. 对于路由系统的架构设计与集群扩展，在座的嘉宾们有哪些思路和经验可以分享？

下面先简单介绍一下本期参与虚拟座谈会的几位嘉宾：

- 丛磊（[@kobe](#)），新浪云平台首席架构师，骨灰级码农+深度技术控，负责 Sina App Engine 整体架构和技术研发。
- CodeBox（[@CodeBox-腾讯](#)），效力于腾讯云平台。
- 杜熙（[@raregogogo](#)），百度云平台架构师。
- 郭理靖（[@郭理靖](#)），京东商城云计算研发部资深工程师。
- 陆林青（[@openshift](#)），效力于红帽，专注于 OpenShift 布道。

另外，本次我们还邀请了程辉（[@程辉](#)）和邓侃（[@邓侃](#)）的参与，两位在仔细研究了这个问题之后，认为这个问题看起来虽然简单，但背后的信息量却很大，需要掌握更多的信息后再进行评论。因此，我们会在本次虚拟座谈会的未来几期，或者其他更加合适的时机，再来邀请他们过来分享。

另一方面，通过一次虚拟座谈会，对上面两个问题也并不一定能产生完美的解答。根据目前跟一些嘉宾们的沟通，后续仍有更多的反馈和内容补充，因此我们决定将本次的话题延续到下周继续讨论，让真理越辩越明。

以下是本期嘉宾们的观点。

丛磊：

 其实随机不是大问题，关键是要有一套良好的健康检查机制，另外对于等级用户需要做隔离。

为什么这么说呢，其实路由规则里面随机是个不错的选择，不一定非要用优先队列什么的，但问题的关键是一定要有健康检查，后端出现故障要能避开，后端出现延迟要能绕开，否则，这种随机只能被定义为不合格了。

SAE 目前日 PV 达到 10 亿，SAE 的路由实现是经过长时间高访问并发考验的，非常有效。其实路由分成两种，一类是从外到内（如 heroku 文中所指），一类是从内到外（SAE 内部叫 SocketProxy）。

从外到内：SAE 拥有一套自己开发的健康检查和 redispatch 机制，能够当后端出现故障时绕开，并在出现临时情况时通过适当的 redispatch 解决。其实健康检查是一个很大的话题，如何判断“挂不挂”，如何最快速的判断，都是一个可以深入讨论的话题。此外，SAE 目前的三个语言平台，PHP、Python、Java 在路由细节上还有所不同，SAE Python 平台的路由规则非常像 Heroku 文中描述的智能路由 经过我们 Python 平台长时间运行检验，其实现是可靠的。

从内到外：SAE 内部所有 3 层以上的对外网络请求都是经过我们的代理（SocketProxy 服务）路由的，我们这套网络路由有以下几个特点：

完全做在系统内核层，无需语言 runtime 关心，支持 TCP/SSL/HTTP/HTTPS
可以针对目的地址的不同选择代理加速

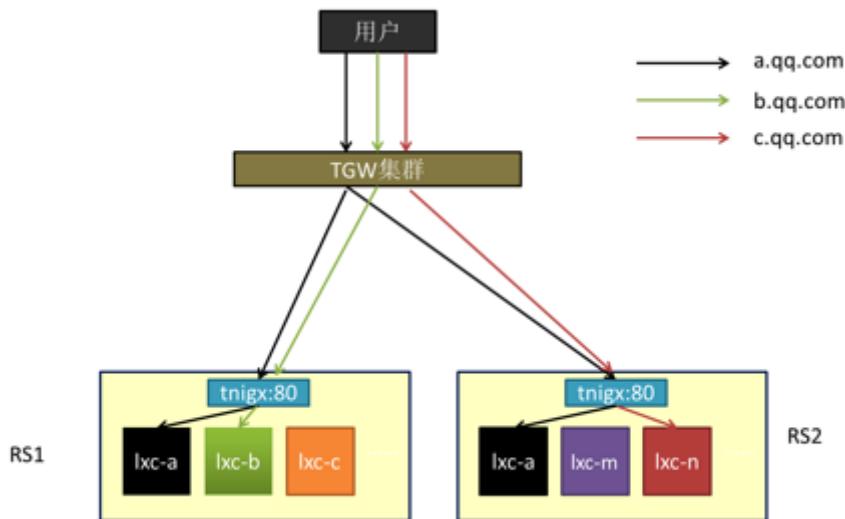
拥有健康检查机制，防止阻塞

拥有针对来源地址和目的地址的统计、配额、限速功能，防止从内到外的网络攻击（如恶意抓站等）

最后就是，针对企业用户，需要做隔离，避免 runtime 的互相干扰，也要避免路由代理之间的互相干扰，在这方面，SAE 通过在其上运行的 2000 家付费企业积累了大量的经验。这方面以后有时间我会再详细讨论。

CodeBox :

说说我的个人观点吧，不代表公司意见。我先介绍一下我们的 PaaS 平台的类似系统的设计。



腾讯 PaaS 平台 CEE 处理用户请求时，是通过 TGW 和 tnginx 共同来完成 routing 和 load balance。

用户请求发到 TGW 的 vip 上 , TGW 根据域名和负载均衡策略 , 选择一台合适的 RS 机器 , 将请求转过去。 TGW 是工作在内核里面 , 本质上是个 Syn Proxy , 因为要 hook 域名 , 所以必须自己跟客户端完成握手并处理一部分应用层数据。而不能像 NAT 代理一样透传 syn 包。从工作原理上来讲 , TGW 扫在前面可以处理一部分安全威胁 , 如半连接攻击等。

RS 上的 nginx 后再根据域名做二次转发，到应用容器里面去。 nginx 工作在 RS 中，也就是 vm host 与 guest 中的 fastcgi 框架的通信是通过 unix socket 实现的。

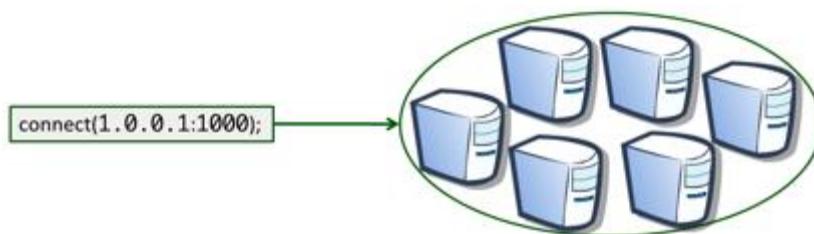
整个过程中没有排队，硬要说有的话也是系统层面的 backlog。如果有突然的流量增加，由于监控和调试系统来扩容。

我们解决问题的思路跟 Heroku 的思路不太一样，我们一般从底层（系统层面）去解决问题，而不针对特定的语言。如我们的内外网负载均衡都是在内

核层面做的，跟语言和组件无关。看讨论，似乎是 Heroku 的解决特定语言和特定架构引入了路由+排队系统，而排队系统的监控又不到位，请求停留在里面时间过长也没能发现。

我们的内网负载均衡也是在底层实现的，也跟语言和组件无关，提供最基础的 ip:port 来抽象服务。

没有私有API，全透明使用，通过一个ip访问一个集群



我们还是比较喜欢从底层解决问题。例如，新浪 SAE 是在 Apache 和 PHP 上做不同用户应用层面的隔离（2011 年 QCon 杭州了解到的，不知现在如何），CEE 是在系统层面用 container 做的，这也体现了我们的一贯思路。两种方式各有优缺点，没有绝对的好坏，例如在底层做会有抽象程度不高等问题。方案的选择主要看你侧重什么、擅长什么。

另，这个论文 [The Power of Two Choices in Randomized Load Balancing](#)

格调有点高雅，一堆公式，看起来很上流，虽不明，但觉厉。只是隐隐觉得有必要弄得这么复杂吗，我还是习惯走“简单粗暴，行之有效”的屌丝之路，哈哈。

杜熙：

“我认为这次事件的直接原因是 Heroku 默认的 Thin 服务器是单线程的（而且是非异步）。每个 dyno 只启动一个 Thin 服务器进程，应用的并发数主要受限于 dyno 数，在高并发请求下，不管智能还是随机路由效果都有限，智能路由只是在一定程度缓解而已。此时，应用开发者有两个选择：一是接受 Heroku 的排队机制，其结果是部分请求延迟会增大（本次 case）；二是花更多的钱，增加 dyno 数。不过以通常业务系统来看，并发消耗的 dyno 大多在等待 io，资源并未真正被有效利用，因此花费巨额的金钱增加 dyno 不是一个性价比高的办法。所以，Heroku 提供了切换到支持并发的 Web 服务器。看起来一直把 Thin 设置成默认服务器确实是一个错误的选择。”

本质上，问题的根本原因是 PaaS 的提供者是否有能力为承载的应用提供保障。Heroku 作为一个 PaaS 的提供者，应该为开发者提供一套分布式架构的 Web App 运行环境。将很多底层细节屏蔽，从而让开发者从这些问题中解脱出来，专注于自身业务。而 Heroku 显然犯了一些错误：如给了开发者太多的选择（提供多种服务器支持，而非定制一个靠谱的），给了一个错误的推荐（Thin），没有及时的通知路由策略的变化，等等。因此，开发者需要擦亮眼睛，谨慎的选择有能力、有经验的 PaaS 提供商，而不能仅凭费用低廉。另外，必要的 SLA 独立监控也是必要的，不能仅凭提供商的一面之词。

最后说一下百度应用引擎（BAE）经验。BAE 脱胎于百度十二年来积累的大规模数据和服务处理的云计算能力，目前每天为数万个应用提供运行支撑，请求量超过 100 亿次/天。在此如此高并发的状态下，为了提高请求的处理效率，BAE 对静态请求采用异步机制处理，动态请求采用多进程池处理。路由算法也是带智能的随机路由，会跳过繁忙或故障的后端节点。而且如果某个应用遇到瓶颈，BAE 能够自动在短时间内增加处理进程或服务器节点，这种“弹性伸缩”才是云计算真正的优势。BAE 也按照百度提供大规模搜索和在线服务的经验，对很多的底层策略和机制进行了调优，确保给应用的默认设置就是最好配置。因此，BAE 能够非常好的解决类似问题。

郭理靖：



对于 PaaS 平台，上层路由是必需存在的，每个 APP 的 Instance 是分布在不同机器上，不同 APP 的 Instance 又共享一台机器，需要路由来把发给 APP 的请求转向给具体的 Instance。Heroku 的核心问题是在于路由有问题。

我们在使用 CloudFoundry 的过程中也发现有类似的问题，目前来看 CloudFoundry 的路由做的比较简单，我们目前在做的一个工作就是把 DEA 的负载状态以及 DEA 里执行的 Instance 的状态都上报给 router，这样路由层有更多信息做更好的负载均衡。

陆林青：



对于 Heroku 的问题，作为另一个 PaaS 的代言人恐怕不便评论。不过从一个用户的角度出发，我希望 Heroku 能够开源，这样对解决问题、对项目发展都有利。

OpenShift 方面，对于来自 App 的更新、访问，利用 DNS+virtual host，实现了直接联络 node，而且对于 scaling app，用户访问的甚至不是同一个 node，大大减少了在某一个部分产生排队现象的几率。对于来自客户端（rhc, web console 等等）的管理请求，OpenShift 自带有 broker 和 proxy 来保证 HA；队列处理方面则有 ActiveMQ。RedHat 为了更加完善 ActiveMQ，已经收购了 FuseSource，所以这项核心技术现在也是 RedHat 主导。

OpenShift Online 目前已经有百万级的大型应用在跑，比如 cloud9.io——一个基于 Node.js 后端和 HTML5/JS 前端的在线 IDE。

其实 OpenShift 的技术和组件都是开源的，用户可以随时从 upstream 获得代码自行部署测试。这是 OpenShift Origin 的搭建文档：

<https://openshift.redhat.com/community/wiki/build-your-own>

这里跟 OpenShift online 用的代码有 99% 都是一致的，唯一的不同是少了一些商业的 Cartridge，比如 JBoss EAP。

还有 OpenShift Enterprise 版的部署文档，目前仍然在社区中更新，已经发布了两部分：

<https://openshift.redhat.com/community/blogs/installing-enterprise-paas-part-2>

其实 Enterprise 版就是很多客户期待已久的 on-premises 版。Origin 是上游代码，Online 和 Enterprise 是公共云和私有云两个不同实现。对 OpenShift 的实现感兴趣的同学，欢迎多去了解我们的文档和代码。

原文链接：<http://www.infoq.com/cn/articles/vpanel-paas-routing-issue>

相关内容：

- [SAP 基于 OSGi 的 Java PaaS 实现了对 JavaEE6 的 Web Profile 兼容性](#)

热点新闻

如何成为强大的程序员？

作者 侯伯薇

Aaron Stannard 是新创公司 [MarkedUp](#) 的 CEO，他最近花费大量时间雇佣、评估很多不同的程序员，并和他们一起协作。在这个过程中他发现并总结了[十种程序员无法意识到自己潜力的原因](#)，意在让更多程序员发掘出自己的潜力，从而成为强大的程序员。

Aaron 提到，他的公司中所使用的技术非常复杂，某些大型企业都很难掌握，所以对于想要加入团队的程序员来说，入门门槛非常高。因此，尽管他们非常仔细地雇佣新人，但还是很难找到足够天才的程序员。于是，他总结出十种阻碍程序员职业生涯发展的行为，并据此来帮助想要提升自身的平凡的程序员们。

1. 太害怕学不会新的工具、语言和框架

一般的程序员会墨守他们最喜欢的工具，而不希望学习新的，因为他们认为，离开了那些语言和工具，多年的经验就会付诸东流。而强大的程序员会拥抱那些挑战和机会，积极地学习新的工作方式。

2. 直到特性“完成”的时候才会提交。（但永远都不会完成！）

他在 [MarkedUp](#) 公司中把这种行为叫做“囤积提交（commit hoarding）”。有些程序员没有足够的信心来承受团队中其他成员的批评和审查，因此会把自己的工作藏起来，直到“完成”状态才提交。

这种开发者会损害团队中其他人员的生产力，因为团队看不到他每天的成果，而且他也不会在正常开发的过程中寻求帮助，这样就会造成很多“最后一分钟”的缺陷，从而让交付延迟。而强大的程序员会知道，代码并不是他们自己，因此会把代码经常自信地呈现在其他团队成员的眼前，获得批评和建议。

3. 只是“知其然”会很危险

在这里 Aaron 举了微软最近在 [C# 5.0 中引入的 async 和 await 关键字](#)为例，这两个关键字会让创建和管理异步调用变得很容易，但是也会造成上下文切换、对共享资源进行多线程访问的成本，仅仅对此有基本了解的程序员会盲目地使用这

些特性，把所有 I/O 调用都封装成 [C#中的 Task 对象](#)，这会创建出危险的、不可预测的而且非常难以测试的代码。

好的开发者不仅“知其然”，而且会了解为什么这么做以及应该在什么样的条件下使用。

4. 分析瘫痪 (Analysis paralysis)

分析瘫痪是指在程序开发初期进行系统分析，常因为太过执着于控制所有可能的变化和意外，而造成大量时间的浪费，裹足不前。这是一种很经典的问题，会影响很多一般的程序员。它通常是由过度分析造成的，但是 Aaron 认为其根本原因在于不敢做出坏的决定。一般的程序员会担心犯错，只想一次成功。

而强大的程序员不会害怕，他们会编写很烂的代码，对其进行单元测试，如果认为无法达到目的，就会在 45 分钟之内把它抛弃。强大的程序员会积极地限制用来研究的时间，因为他们知道那是个陷阱——看起来是有效的，但经常都无效。

5. 没有对工具和开发过程投入

如果你想要成为天才程序员，那么就需要投入时间提升技能和知识，而将你和普通的代码工人区分开来的是快速编写出生产级别代码的能力。你可以同时拥有好的代码和速度，但是你需要先对你用于构建的过程投入。

一般的程序员不会对工具、过程和环境投入，只会使用大量的时间学习新的语言特性和 API 如何工作，但那并不会改变什么。

通常，你作为程序员所能够做出的最大改进并不是专注于你所编写的代码，而是优化你编写代码的过程。

6. 羞于请求帮助

一般的程序员羞于或者不想让人知道自己不懂，所以他们装作什么都知道，但这样就有可能提交某种非常可怕的代码到库中。说“我不知道怎么做。”没什么错，强大的程序员知道这一点，所以当被问题难住的时候就会请求帮助。

7. 不知道如何让其他程序员更容易使用你的代码

在所有技术团队中，工作很重要的一部分就是人员的并行(human parallelism)，也就是多个人能够同时对同一代码库工作的能力。但是对于团队来说，能够异步工作也很重要，当你不在的时候我可以修改你的代码，反之亦然。

一般的开发者并不这么认为，他们会开始对一项任务编写代码，认为他们会永远拥有这段代码。而强大的开发者会知道技术债务的说法，从而试图通过设计代码来对其限制，让它尽可能可维护和自解释。

编写可读的代码需要程序员改变他们的看法——你的代码要比你在组织中存在的时间长。

8. 不知道如何阅读其他人的代码（或者不想读）

当一位一般程序员看到用他所不熟悉的语言或框架编写的代码库时，就想立刻重写，而不考虑业务价值或者推向市场的时间。而强大的程序员会接受这样的观点，重写所导致的业务成本通常是不可接受的，所以应该避免这种行为。他们会试图坐在计算机前，理解、学习然后修改现有的代码。

阅读代码要比编写代码还难，但是强大的程序员会投入时间来学习如何超越。

9. 不能从最终用户的角度编码（你考虑的范围太狭窄）

有句话说得好：作为程序员，你的工作不是解决技术问题，你之所以解决技术问题，是为了解决业务问题。

一般的程序员只会陷在技术问题之中，而不知道最初为什么要解决这个问题。更严重的是，一般程序员无法从头开始创建出具有业务价值的东西。当被要求基于简单的用户设计新特性的时候，他们会死板地、照着字面对故事或者说明书做出解释，这样交付的产品用户根本无法使用。因为他们不会考虑相关的用例；不会考虑最终用户的体验；并且在做面向用户的内容时，设计都会很笨重。这导致他们无法编写业务应用，只能做产品。

好的程序员会从最终用户的角度来看他们的代码。我怎样才能让它更轻松地解决用户的问题呢？故事的文字内容之外有哪些方面会让这个特性给用户带来更多收益呢？

10. 无法判断任何编程任务的业务价值

这个问题和上一个是相关的，很多技术上很强的程序员之所以无法意识到自己的潜力，是因为他们不会停下来，从业务或者组织本身的角度去看一下他们的工作。

强大的程序员能够自我管理，对选择如何投入时间做出很好的业务决定，他们会问这样的问题：这是我现在应该做的最有价值的事情吗？我应该为之投入多少时间？离交付日期有两个星期，我现在能做什么，从而更容易满足那个日期呢？

一般的程序员不会，他们只会拿着说明书，然后盲目地实现，直到结束，不关心他们的工作和公司的业务目标有什么关系，以及对其他团队和业务组会产生什么样的影响。这样，他们就会在业务价值很低的技术任务上浪费大量开发时间。

Aaron 在最后做出总结：如果你想要成为更好的程序员，那么就要从改变你看待代码以及编码的方式开始。你需要理解所编写的每行代码背后的业务成本；你需要从客户或者最终用户的角度来看待工作；你需要接受代码会比你在组织中存在的时间更长，所以要以其他开发者能够继承的方式来设计；最重要的，永远都不要害怕新的挑战，也不要害怕请求帮助，你无法独居一隅来提升工作效果，软件开发也是社会化的工作。

原文链接：

<http://www.infoq.com/cn/news/2013/02/howto-strong-developer>

相关内容：

- [产品经理如何分析互联网产品？](#)
- [Team Foundation Server 11 中的应用生命周期管理](#)
- [实战：持续交付中的业务分析](#)

热点新闻

Heroku 的教训：糟糕的负载均衡 + RoR 单线程 = 糟糕的性能

作者 [杨赛](#)

两天前，Heroku 最大的用户之一 Rap Genius（这是一家面向嘻哈音乐迷的网站）在其官方网站上发布了[一篇博客](#)，表达了对 Heroku 极大的不满，因为 Heroku 的路由系统（Heroku routing mesh）性能太差。该文章很快被推送到 [Reddit](#) 和 [Hacker News](#) 上，引起了极大反响。

之后，Heroku 的 COO Oren Teich 很快在官方网站上[更新了一篇声明](#)，表示这是一个累积了三年的性能问题，所有运行在 Bamboo 上的 RoR 应用在 Heroku 经历扩展的同时，性能一直在下降。Teich 表达了歉意，并承诺会在第二天发布一篇深入的技术报告，说明问题的原因。同时，Teich 还表示 Heroku 将会提高用户应用 Web 请求队列的透明度，提供更好的文档、工具和沟通机制。

今天，Heroku 的产品经理 Jesper 如约在网站上发布了相关的[技术细节](#)。

Rap Genius 方面的问题描述

Rap Genius 目前用户数量超过 1500 万，每月投入到 Heroku 上的成本约 2 万美元。十天前，Rap Genius 的开发人员在一系列 AB 基准测试中发现，自己的版权页面——一个纯静态页面——的平均响应时间居然达到了 6330ms。更大的问题在于，Heroku 的后台和 Heroku 的监控合作伙伴 New Relic 报告的数据是 40ms，相差了两个数量级。

于是他们去问 Heroku。Heroku 的工程师说，两个数值不同的原因在于“应用的请求需要在 dyno 这一层排队”。请求被处理的速度很快，时间都花在排队上了。（注：dyno 是 AWS 上的虚拟机。目前 Heroku 有两种活跃的堆栈，Bamboo 是老堆栈，运行 Debian 5.0；Cedar 是新堆栈，运行 Ubuntu 10.04。）

但是，在 Heroku 后台提供的日志功能当中，完全没有“dyno 这一层的排队等待时间”这一数值。

都是 routing mesh 的错

Rap Genius 的开发人员们四处翻查文档，终于发现了一个情况：Heroku 在 2010 年年中重新设计了 routing mesh，新系统的路由规则是：通过一个随机算法将请求分发给 dyno——无论 dyno 是否空闲。

Heroku 在 2009 年的文档表示，在智能路由规则下，请求只有在 dyno 可用的时候才会被路由到该 dyno，如果一个 dyno 上运行了一个长任务，路由会把请求提交至另一个 dyno，而不是继续排队。

于是，Rap Genius 方面认为，现在的延时问题都是 Heroku 偷偷修改了路由规则所导致的。

文章里按照两个路由规则做了一个模拟，结论是：在智能路由模式下用 80 台 dyno 可以搞定的并发请求数，在随机路由模式下需要 4000 台 dyno！

“天真的”随机路由模式，老旧且不一致的文档，监控度量的缺失，造成了 Rap Genius 方面的极大不满。他们希望 Heroku 能够将路由模式切换回智能模式。

Heroku 方面的说明

Heroku 的回应文章承认了延时问题的存在，对自己没能早些发现这一问题表示道歉。但是，Heroku 表示延时问题跟 2010 年的新路由规则无关，并简单介绍了 Bamboo 和 Cedar 上的路由机制。

2009 年开始启动的 Bamboo 堆栈仅支持 Ruby 语言，Rails 框架和 Thin 服务器。Bamboo 本身不支持并发，一个进程一次只能处理一个请求。所以在请求分发上，为支持 Bamboo 的架构，Heroku 设计了 HTTP Router，在路由层为请求消息进行排队，然后再分发给 dyno；每个路由自己建立自己的应用队列，并不存在全局的应用队列。

也就是说，2009 年文档上描述的“请求仅会被路由到空闲 dyno 上”的智能路由模式，其实仅针对单一路由，并非针对整个路由系统。在整个路由系统中，一直存在“dyno 层的排队”这一事件。在路由集群较小的情况下，这一事件发生的几率较低，所以整个集群看上去还比较智能。

新的路由机制针对在 2011 年开始运作的 Cedar 堆栈。由于 Cedar 希望支持 HTTP 轮询和区块响应（chunked response），支持 JVM、Node.js、Unicorn、

Puma 等可以多线程、多进程的运行时，支持无状态架构，满足伸缩性需求，所以 Heroku 的工程团队为 Cedar 选择了随机路由模式。这一模式仅针对 Cedar，而不针对 Bamboo 上的老用户。

按照 Heroku 的说法，他们最初设想的是 Bamboo 用户会逐渐将自己的应用迁移到 Cedar 上面去，这样就可以保持一个较小的路由集群。然而现实情况是，Bamboo 上的应用一直在不断扩展，导致 Heroku 团队不得不往 Bamboo 的路由集群中不停地增加节点，这导致智能路由的效率越来越差。用 Jesper 的话来说，当智能路由集群的节点过多时，这个集群的工作方式基本上等同于一个随机的路由集群。

而 Cedar 用户感受到的延迟，则源于 Rails 无法处理并发请求，而大量 RoR 应用被按照官方文档的指导部署在了 Thin 服务器上——这是个单线程的服务器。Heroku 表示将应用迁移至 Puma 或 Unicorn 这样的并发 Web 服务器上能够缓解这个问题，他们会提供更多的文档和支持帮助用户完成此类迁移。

相关评论

Heroku 承认错误的态度得到了不少旁观者的称赞，但是 Heroku 针对延时问题提出的行动方案，却完全没能让 Rap Genius 方面满意。Rap Genius 方面针对 Heroku 的技术总结文章又[展开了回应](#)，做出如下评价：

1. Heroku 真的刚刚知道这个问题么？[2011 年 2 月他们就知道了吧](#)。
2. Heroku 针对 Rails 的默认 Web 服务器是 Thin，无论在 Bamboo 还是 Cedar 上都是如此。为什么不把 Unicorn 设置为默认服务器？害怕占用太多内存么？
3. Heroku 提出了行动方案，但完全不是解决方案。

Hacker News 上也普遍存在两种质疑：

1. 即使可以并发，也只是暂时缓解延时的问题；当规模继续扩大时，延时的问题又会变糟糕。
2. 多花钱用 PaaS 就是为了不折腾。如果多花钱还免不了折腾，还不如直接用 AWS。

对于 Heroku 遇到的路由延时问题，你有什么看法？大规模的应用适合放在 PaaS 上面跑么？欢迎讨论！

原文链接 : <http://www.infoq.com/cn/news/2013/02/heroku-routing-mesh>

相关内容：

- [Heroku 危机带来的启示](#)
- [VMware 发布 Cloud Foundry 的免费版本](#)
- [Cloud Foundry 遭遇存储故障](#)

热点新闻

jQuery 作者 John Resig :WebKit 就是浏览器引擎中的 jQuery

作者 [彭超](#)

JavaScript 领域的神级人物 jQuery、《Pro JavaScript》与《JavaScript Secrets》的作者， Khan Academy 计算机科学学院的院长 John Resig，在看到 Opera 浏览器切换到 WebKit 引擎的新闻，以及 Twitter 上许多咬牙切齿的争论后，忍不住在自己的博客发表了一番见解，[英文原文在此](#)。他觉得大家对这件事的反应让他感觉回到了 2008-2009 年，而现在已经 2013，Chrome/Chromium 团队已经通过自己的成绩向大家证明了当使用 WebKit 的时候，可以不用担心发生停滞的步伐或者创新的缺失，反而还能在实现通用 Web 标准的时候少花很多时间。他还认为，WebKit 在这一点上发挥的作用，就像 jQuery 对 JavaScript 一样。

在 Opera 切换浏览器引擎这件事上，John 总结了他见到的几种典型言论，并一一加以点评：

浏览器切换到 WebKit 会导致浏览器引擎发展停滞



很明显这不是真的。KDE 创立了 KHTML，在 KHTML 的基础上 Apple 创立了 WebKit，而 Google 在它的基础上创立了 WebKit/Chromium。我相信任何人都能指出，相比 Safari 来说，Chrome/Chromium 比 Safari 更好，而 Safari 比 Konqueror 更好。Chrome 团队已经证明了在选择使用 WebKit 时，作为 WebKit 的项目贡献者，他完全有能力将 WebKit 驶向你想要去的方向(一般来说是去更好的方向)。我完全相信拥有高素质开发团队的 Opera 也会做类似的事情。他们完全可以在 WebKit 上实现相当数量的 Opera 独有特性，这些特性之后又非常可能会流向其他也使用 WebKit 作为引擎的浏览器。

这会帮助 WebKit 成为一个事实上的标准



我没看到比这更中肯的观点了——WebKit 已经是事实上的标准。比如，每个人都清楚浏览器早就实现了带有诸如 '-webkit' 这样的特性。很显然，

WebKit 成为事实标准这件事儿早已经深入人心。当然，Bugs 也是。WebKit 是一个由许多浏览器厂商贡献代码的公共代码库，然而，每个浏览器厂商都可以改变他们自己的代码分支。我认为已经拥有事实标准的 WebKit 的 bug 们肯定会被浏览器厂商修复，但引擎中有 bug 并不代表浏览器厂家没有修复能力——他们可能故意不去修复这些 bug(就像浏览器厂家们目前正故意克隆那些-webkit 前缀一样)。

就像 JavaScript 库一样，事实上大家都已经把 jQuery 当作了标准。但是这并没有导致发展的停滞。这还导致出现了很多构建在 jQuery 之上的，高层流行框架的产生，比如 [Twitter Bootstrap](#)，[HTML5 Boilerpalte](#)，和 [Backbone.js](#)。

这会妨碍 Opera 影响 Web 标准的能力

“我不认为 Opera 切换到 WebKit 会导致这个情况发生。但我确实看到 [Anne van Kesteren](#) 从 Opera 跳槽到 Mozilla 确实是 Opera 推进 Web 标准能力的巨大打击。但我对内幕一无所知，但是如果他的跳槽是源自这次浏览器引擎的切换，那么 Opera 影响 Web 标准的能力确实遭受到了损失。

Opera 切换到 WebKit 是在走下坡路/Opera 份额太小，如果 Firefox 或者 IE 切换到 WebKit 那会是一个大问题

“我认为有一点已经非常清楚了：目前 WebKit 在移动端已经非常明确的大获全胜。包括即将切换到 WebKit 上的 Opera Mini/Mobile，WebKit 几乎是市面上绝大多数移动浏览器所选用的唯一渲染引擎。如果任何其他浏览器想要在移动世界占据一席之地，那它必须和 WebKit 在功能上保持一致。让我们在此得到一个逻辑上的结论：在移动世界已经被 WebKit 统治的情况下，为了保持同步，Mozilla 和 Microsoft 将会感受到巨大的压力来迫使他们将自己的浏览器也切换为 WebKit。Google 已经通过 Chrome 证实了使用 WebKit 并不会导致发展停滞，其他公司更没有理由不来继续打造 WebKit（有可能会创造 WebKit 的混合体，比如 WebKit+IonMonkey）。

这些问题都归结到这个大问题：他们(Mozilla , Microsoft)应该切换引擎吗？

“老实说，在这一点上，对 Mozilla 和 Microsoft 来讲，这成了一个商业问题，或者是工程问题。如果你的一些开发者要花他们所有精力去实现别人正在实现的相同标准，那么切换到一个公共的代码库（指 WebKit，编者注）将会

解放你的劳动力，让你可以做点儿别的事情。你可以看到 Chrome 的例子：他们解放出的生产力全面投入到了性能的竞赛上。这个打造最快浏览器的比赛促进了大家共赢的结局。

最后，我们一定要了解，WebKit 并不是一个完整的实体。他是一个有多家公司贡献代码的共享代码库。（从这方面来说，这和 jQuery 并不相同：几乎所有的 jQuery 代码贡献最终都回到它的主代码库，而 WebKit 的有些更新则只保留在分支之中）。使用一个共同的代码库并不意味着这是浏览器开发的一切，更不是浏览器开发的终结。在一个共享的代码库中仍然可以有持续不断的创新，它的性能也当然会一直提升下去。

在原文评论中我们看到，即便是这种重量级人物的发言，也会引来大量吐槽。大家赞同 John 对 WebKit 的认可，但是不少人对于 jQuery 的部分却不那么满意。用我们熟悉的词来形容，就是“楼盖偏了”：

 **mitch**：我很了解 John 和 jQuery，所以我不得不承认这是一个负面评论：我认为 jQuery 只是 JS 框架中的 IE7 而已。所有我见到在使用 jQuery 的人都把代码搞的一团糟，毫无结构可言。毫无疑问，jQuery 非常强大，但是它被自己局限住了。我可不认为 WebKit 是浏览器中的 jQuery。

michael camden：@mitch 我可不太赞同你的观点。jQuery 把代码结构的问题完全留给了程序员。jQuery 非常灵活，它只是并不像那些庞大的框架一样去强制要求你按照固定格式书写代码而已。

Mark V：完全同意 Mitch 对 jQuery 的观点。

jQuery 只是在 JS 的顶层做了一些事情，而为了这一点，它向很多短视的利益屈服了。而这是很多大的 JS 项目根本不看中的。如果所有的浏览器都向 WebKit 迁移，这可能还不错（比如没有 IE7-9 的烦恼了，真棒），但是长期来讲这伤害到了整个 Web 开发产业。

还记得 IE6 吗？当它刚发布的时候吗？当时它还是一个相当不错的浏览器。最起码比 IE5 强多了。所以当时所有人都在 IE6 下写代码，和 bugs 为伍，适配网页布局和引擎的怪异模式。一片祥和，不是吗？而 10 年后，它却成了 Web 开发行业里最沉重的负担。

当人们开始在单一环境下编写代码，并开始利用它的 bug 来实现各种特性的时候，标准就不再重要了。各种实现成了新的标准。

特别说明：Backbone 是可以脱离 jQuery 运行的。（编者注：这位仁兄最后还不忘吐槽。）

Rob :一个更恰当的比喻应该是 Linux 内核。Chrome 就像 Ubuntu ,Safari 是 Fedora , Opera , 我估计应该算 Mint 或者 Arch。

更多有趣的留言，大家可以[点击此处查看英文原文](#)。

对于此事现在业界众说纷纭，JavaScript 的另一位大神 Douglas Crockford 怎么看？敬请继续关注 InfoQ 的后续报道。也欢迎各位留言讨论。

原文链接：

<http://www.infoq.com/cn/news/2013/02/webkit-is-like-jquery>

相关内容：

- [Web 趋向统一？Opera 宣布浏览器引擎将切换至 WebKit](#)
- [《JavaScript 语言精粹》作者 Douglas 谈 Web 开发、jQuery 和 WebKit](#)
- [轻量级 jQuery 网格插件——ParamQuery](#)

热点新闻

Facebook 如何实现 PB 级别数据库自动化备份

作者 [郑柯](#)

Facebook 的 MySQL 数据库，是世界上最庞大的 MySQL 数据库之一，在不同地区有数千个数据库服务器。因此，备份对他们来说是个巨大的挑战。为了解决这个问题，他们构建了一个高度自动化、非常有效的备份系统，每周移动多个 PB 的数据。Facebook 数据团队的 [Eric Barrett](#) 通过[一篇文章](#)分享了他们的做法。

他们没有采用大量前载 (front-loaded) 测试，而是强调快速检测失败，并且进行快速、自动化纠正。部署几百个数据库服务器，只需很少人力干预。使用下面的三个措施，他们做到了有节奏的增长，同时具备支持上十亿用户的灵活性。

措施 1：二进制日志和 mysqldump

第一道防线称为“措施 1”，或“机架”备份 (rack backup)，简称 RBU。在每个数据库机架上，不论其类型为何，都有两个 RBU 存储服务器。以 RBU 作为数据库服务器放在同一个机架中，这可以保证最大的带宽和最小的延迟，它们同时可以作为缓存，在备份的下个措施使用。

收集[二进制日志](#)，是这些服务器的工作之一。二进制日志会不断以流形式，通过模拟从进程 (simulated slave process) 输送到 RBU 主机中。这样一来，不需要运行 mysqld，RBU 就可以接收到同样的更新作为复制版本。

在 RBU 上保存同步的二进制日志很重要：如果一个主数据库服务器离线，该服务器上的用户将无法更新状态或是上传照片。出现问题后，他们需要保证修复时间越短越好。有可用的二进制日志，就能让他们在数秒内启动另一个数据库作为主数据库。由于 RBU 中有秒级的二进制日志，即使某个旧主数据库完全不可用，也没有关系，只要利用将记录下的事务恢复到上一个备份中即可完成立即恢复。

RBU 服务器的第二个工作是执行传统备份。MySQL 备份有两种方式：二进制和逻辑 (mysqldump)。Facebook 使用逻辑备份，因为它与版本无关，提供更好的数据完整性，更紧凑，恢复起来更省事。不过，当为某个数据库构建全新复制时，他们仍然使用二进制拷贝。

mysqldump 的一个主要好处是：磁盘上的数据损坏不会影响到备份中。如果磁盘某个扇区出现问题，或是写入错误，InnoDB 页面校验和就会出错。在组合备份流时，MySQL 会从内存中读取正确的内容，或是去磁盘读取，然后遇到错误的校验和，停止备份（以及数据库进程）。mysqldump 的问题是：污染用来缓存 InnoDB 块的 LRU 缓存。不过，新版本的 MySQL 中，会将 LRU 插入操作从扫描时放到缓存结束。

对在自己权限范围内的所有数据库，每个 RBU 都有一个夜间备份。尽管有着天量级别的数据，Facebook 的团队还是可以在几个小时内完成对所有数据的备份。

如果 RBU 失败，自动化软件会将其职责分配给同一集群中其他系统。当它恢复上线后，职责会自动返回到最初的 RBU 主机。

Facebook 团队不会过分担心单个系统的数据保留问题，因为他们有措施 2。

措施 2：Hadoop DFS

在每个备份和二进制日志收集完成后，他们会马上将其复制到他们的大型定制化 Hadoop 集群中。这些集群是非常稳定的复制数据集，并有固定的保留时间。因为磁盘大小增长很快，较老的 RBU 可能不足以保存一到两天的备份。不过他们会按需要增长 Hadoop 集群，同时不需要担心底层硬件情况。Hadoop 的分布式特性让他们有足够的带宽，完成快速数据恢复。

不久，他们会把非实时数据分析放到这些 Hadoop 集群中。这可以降低数据库中非关键读的次数，让 Facebook 网站的响应速度更快。

措施 3：长期存储

每周，他们会从 Hadoop 备份移动到另一个地区的分散存储中。这些系统是最新而且安全的存储系统，在他们的[日常数据管理工具](#)流程之外。

监控

除常用的系统监控外，他们还会捕捉很多特定的统计数据，比如 binlog 集合延迟、系统容量等等。

为备份失败打分，是他们最有价值的工具。因为 Facebook 的数据库和同时运行的维护任务量级，错过某些备份也不奇怪。广泛的失败和多日没有成功的单个备份，这都是他们要注意的重点。因此，某个错过备份的得分会随着时间呈指数级

增长，这些得分的不同聚合，让团队能对备份的整体健康度有一个有效而快速的了解。

比如，在一天内，某个数据错失一次备份，得 1 分，一天错失 50 次备份，就是 50 分。但在三天内的一次数据库错失，就是 27 分（3 的 3 次幂），三天内 50 次，这是很严重的问题，得分就是 1350（50 乘以 3 的 3 次幂）。这会在他们的监控图上出现一个巨大的波峰，团队会马上对其采取行动。

恢复

在系统管理员中有句老话：“如果你没有测试过你的备份，就等于没有备份。”

因此，Facebook 团队构建了一个测试系统，会持续地从措施 2 开始，将数据恢复到测试服务器上。恢复完成后，他们会执行多次数据完整性检查。如果有任何反复出现的问题，系统就会报警，提醒相关人员关注、审核。该系统可以发现所有问题，包括 MySQL 的 bug，到备份过程中的纰漏，并可以让他们更灵活地应对备份环境中的变化。

他们构建了一个名为 ORC（ORC 恢复协调器的递归缩写）的系统，工程师如何需要恢复他们所用工具的数据库的过去版本，就可以以自服务方式使用该系统恢复数据。对于快速开发来说还是挺方便的。

在结尾，Eric Barrett 说道：

“备份不是最迷人的工程工作。它们即是技术活，又是重复性的，如果一切正常，没人会注意。它们也是跨学科和团队的，需要懂得系统、网络和软件等多方面的专业知识。但是，确保你的记忆和联系安全无误，这是无比重要的事情，而且到最后，也是充满回报的事情。

有网友问到：

在不运行 mysqld 的 RBU 上，你们如何完成二进制日志的流传送？什么是模拟从进程？

Facebook 的 MySQL 性能工程师 [Harrison Fisk](#) 给出了答案：

“我们使用 mysqlbinlog 的--never--选项，并有一个用 python 开发的小包装程序，会监控并保证 mysqlbinlog 运行成功。

特别专题

“探索算法和推荐系统”专题前言

算法是否重要，看看目前互联网领域有多少技术或者产品基于它们而设计就晓得。尤其是现在的电商网站，没有算法和推荐在里面，很难想象他们如何从买家手中赚到更多的钱。笔者从前在和某大型电商公司的朋友沟通时，谈到一个很有意思的传言，技术团队一开始对技术架构等基础设施非常感兴趣，投入了很多人力和物力，但是公司发展到一定规模，发展首页的广告点击始终保持在一定的规模，上不下下不来，很是愁人。后来技术团队进入了一个算法方面的人才，将其广告系统的参数做了微调，结果却显示广告效果较原来提升了近 40%。而百度知道在一次调整推荐算法后，该产品上问题的回答量也比从前有了数量级的增长。

本专题试图通过和百度、百分点专家的沟通，分享他们在算法和推荐系统方面的经验，抛砖引玉，进而希望能有更多的朋友对该技术领域感兴趣，一起推动算法和推荐系统技术的普及。

特别专题

一次推荐算法的普及性讨论

作者 霍泰稳

但是在笔者参加了一些和算法以及推荐系统相关的活动之后，发现这一高深的学问已经被从事软件开发的朋友应用的非常广泛。特别是在电商火爆的今天，各种和推荐相关的网站风起云涌，算法进入平常百姓家也就是水到渠成了。这也是为什么 InfoQ 在 2013 年，将算法和推荐系统作为一个和移动开发、云计算等相并列的重点专题的原因之一。

作为一个编辑，要想为读者提供说得过去的内容，前提肯定是要对这一技术领域能有个七七八八的理解。也基于这个原因，笔者和业界的一些专家做了相关的沟通，包括 ResysChina 论坛的发起人谷文栋，百度的前技术委员会主席廖若雪等。本文旨在将笔者从这些专家身上所学习到的内容分享给大家，希望对想要算法有所了解的同学有所帮助。

算法和推荐引起关注的原因

在 2012 年年底，InfoQ 曾参与举办过一次以算法为主题的百度技术沙龙，邀请了百度的研发工程师赵岷以及百分点 COO 兼技术副总裁张韶峰参加，能容纳 160 人的场地来了差不多 260 人，反响非常强烈。而在 ResysChina 举办的推荐系统大会上，更是火爆，来自 Lulu、Facebook、百度、淘宝、腾讯等公司的算法专家分享自己的经验，在微博上也激起热烈的讨论。至于算法和推荐之所以受关注的几个原因，大体可以归纳为以下几点：

- 从行业趋势来看。信息大爆炸使得信息极大丰富，传统获取信息的手段已经不能很好地解决这种环境下的信息获取需求，推荐和个性化技术，作为解决信息爆炸问题的一个方法，取得了不错的效果。在业内的各个公司的应用也越来越多，这反过来也促进了大家对推荐的热情
- 从技术本身来看。推荐涉及到的技术深度、复杂度往往也是计算机科学最前沿的。例如：推荐系统要处理的数据规模往往高达 PB (Petabytes , $1PB=1024TB$)，而实时性要求则要求到秒级别，这对于架构和算法都是非常高的挑战。

- 推荐是系统和人的互动，推荐首先需要更好地理解人，理解用户。就百度前技术委员会主席廖若雪看来，这是一个互联网更加智能化的发展方向，系统的智慧会越来越高级，这本身是非常有吸引力的方向。

算法并不深奥，案例为证

如笔者在本文开始时所谈到的，考虑到算法和数学等结合很紧密，而且在日常的生活中应用的也较少，其实很多人对算法还是多少有些恐惧。但算法真的有那么深奥吗？广义来说，算法是人们归纳总结出来的解决一类问题的方法。一般而言，我们所说的计算机算法，为了描述的精确严密，通常会采用数学的形式语言去描述问题和给出对应的解决方案。抽象的形式语言对大众而言是不容易理解的。但算法本身只是一个解决问题的途径，实际的应用中我们会使用很多的算法，有的简单有的复杂。

为了让普通技术人员对算法有更好的了解，廖若雪列举了一个电商网站经常有的一个功能——“热销榜”。其实“热销榜”就是一个简单有效的算法引用，利用销售量直接进行排序。同时，在这个基础上也可以进行进一步的深入分析，针对实际问题，做出更有针对性的策略升级。但直接按照销售量排序，有时我们会遇到大量热门商品长期占据榜单，存在“过期”的情况。比如，适合冬季销售的产品大量卖出，到夏季时已经无人再购买，但由于历史数据，使得这个商品仍然占据着榜单的位置，显得不合时宜。这时我们就要有针对性地改进算法，加入“时间”维度的信息。

我们既可以简单地按照离散的一刀切的方式，如只计算1个月以内的销售量；也可以做一个连续化的函数，使得历史的销售量会随着时间进行衰减，如使用“牛顿冷却定理”（注：温度高于周围环境的物体向周围媒质传递热量逐渐冷却时所遵循的规律。当物体表面与周围存在温度差时，单位时间从单位面积散失的热量与温度差成正比，比例系数称为热传递系数）。在算法的指定和改进中，我们既可以针对实际问题，使用独特的有针对性的方案；同时也可对问题进行抽象，将实际问题转变为一个已知问题的变种，利用已有的知识和成熟的算法，来解决新问题。

其实很多算法，其基本原理都是来源于我们的常识，例如：你想购买一部手机，但是满足你基本要求的手机是在太多了，你开始纠结。于是你关注周围的同事和朋友们都在使用哪款手机，并咨询他们对这款手机的评价，从而来帮助自己做决定。

策，因为你觉得他们在对手机的使用习惯和需求，比其他人（比方说你的长辈或者晚辈们）的更相似，所以他们的评价更有参考价值。这就是大名鼎鼎的协同过滤算法，是推荐领域里最为经典的算法。它的核心思想就是：根据与你最相似的一些用户对某个事物的喜爱程度的投票，来预估你对这个事物的喜爱程度。

算法、架构、策略、机器学习之间的关系

在过往和技术人员交流时，很多人对算法和架构之间的关系感到不可理解，算法是软的，架构是硬的，难道算法和架构还有什么关系不成？其实不然，算法和架构的关系非常紧密。在互联网时代，我们需要用算法处理的数据规模越来越大，要求的处理时间越来越短，单一计算机的处理能力是不可能满足需求的。而架构技术的发展，带来了很多不同特点的分布式计算平台。算法为了能够应用到这些分布式计算平台上，往往需要进化，例如：并行计算要求算法可以拆分为可并行计算的几个独立单位，但很多算法不具备这种可拆分特性，使得不能简单通过分布式计算来提高效率。这时候，为了实现分布式化的计算效果，需要将算法进行等效改写，使得其具有独立拆分性。另一方面，算法的发展，也反过来会对计算架构提出新的要求。

对算法和策略的关系亦是，不过这两个概念并不像算法和架构那样好解释，廖若雪给了一个自认为偏个人的看法。策略是解决具体问题的手段，而算法是解决一类问题的方法。解决一个具体问题，可能需要将问题分解为一个或者多个算法，一起作用来解决，也可能不需要算法。例如，对于个性化新闻，我们可能有一个策略是：重大新闻需要及时展现给用户；而实现的具体算法可能只包括“重大新闻挖掘算法”等。

机器学习是一类算法的统称，在一定的数据集合上，利用机器学习的算法，自动得到规律，来进行预测，机器学习领域常见的问题包括分类问题、回归问题等。而预测，尤其是对用户的偏好进行预测是推荐领域的核心问题之一，机器学习算法在解决此类问题上会发生很大的作用。

学习推荐算法的几点经验

记得从前在和某个大型电商公司的朋友沟通时，谈到一开始大家只对技术架构什么的兴趣，对于算法并不是非常的上心，只是用一些现成的方案去做，认为可优化的区间不大，投入产出比少。但是后来公司引进了一个在算法方面非常有经

验的同学加入，这哥们来了之后只是将其广告系统的参数做了微调，就发现广告效果较原来提升了 40%。这就是算法的神奇和实用之处。而据廖若雪介绍，在其自有产品百度知道上，他们通过优化推荐算法，在很大程度上提高了知道产品上的问题回答量。在前段时间百度所发布的一个公开报告中，详细地解释了该算法的使用。

在和其他一些推荐算法领域的朋友沟通过程中，他们也提到这些情况是非常有可能存在的，在合适的场景，使用最合适的算法，能够带来意想不到的效果。尤其是对于一些复杂的数据应用场景，比如计算广告学、搜索排序和个性化推荐等。当然，改进算法可获取的收益空间，也依赖于实际场景下的效果。比如对于一个准确率已经达到 90% 的问题而言，准确率自身已经没有太大的提升空间。其实问题还是在于是否抓住了问题的关键和主要矛盾。也许对于某些应用，调整一下页面布局、字体、颜色等，有时也能实现很好的效果。

对于想深入研究推荐算法的朋友，廖若雪也给出了自己的一些经验，虽然没有涉及到非常细节的地方，但相信搞算法的同学都是绝顶聪明之人，从这些点滴经验中就能体味到“顿悟”的感受吧：

- 没有最好的算法，只有合适的算法。推荐算法和产品需求、应用场景、数据密切相关，不要相信有什么包打天下的算法；
- 数据是基础：数据充足而且质量高，简单算法也可以有不错的效果；反之，则多好的算法也不可能有好的效果；
- 木桶效应：算法策略要和用户需求、功能展现密切配合；（注：木桶原理又称短板理论，其核心内容为“一只木桶盛水的多少，并不取决于桶壁上最高的那块木块，而恰恰取决于桶壁上最短的那块。”）
- 一般而言，推荐算法都需要考虑是否能处理大数据，是否能大规模并行化。

再次提醒各位读者，本文只是 InfoQ 推荐算法专题的开篇，接下来我们计划走进技术社区的各个技术专家，和他们进行更加深入的交流，了解他们在算法这一技术领域的经验和教训，并分享给大家。如果各位读者有任何线索，或者想了解某公司某人的观点，大可以直接邮件（editors@cn.infoq.com）或者微博私信[@InfoQ](#)，我们的编辑会扮演好中间人的角色，套出这些专家的经验给大家。也希望“推荐算法”这个专题能够更加引起大家对这一技术领域的重视和了解。

百分点推荐引擎是国内领先的推荐技术平台，专注于为电子商务和资讯网站提供 SaaS 模式的个性化推荐服务，提高网站的整站转化率和用户黏度。本文将从电子商务网站的实际需求出发，介绍百分点推荐引擎架构设计和搭建。

需求

当下，个性化时代的潮流势不可挡，业界普遍意识到了推荐是网站的一项基本服务。但是，人们对推荐该如何来做，也就是推荐技术本身，还不甚了解。我们经常会遇到这样的疑问：“购买过该商品的用户还购买过哪些商品这种推荐，不是一个 SQL 语句就搞定了吗？”其实不然，推荐技术远远不是这么简单。广义上讲，推荐技术属于数据挖掘和机器学习范畴，这也意味着好的推荐服务依赖于科学的推荐算法和大量的学习数据。对于电子商务和资讯网站来讲，想在推荐技术领域精耕细作，研发高端的推荐算法并应用到海量数据上是非常困难的。正是在这样的背景下，百分点推荐引擎应运而生。在百分点推荐引擎产品的开发过程中，我们与麦包包、红孩子、走秀网、耀点 100 等知名电子商务网站，以及天极网、亿邦动力等知名媒体资讯类网站的技术部门进行了深入探讨，从他们那里得到了很多帮助与启发。在与这些行业先锋的交流中我们发现，有一些需求是行业共有的，比如推荐的实时性、高可用性。另外一些需求是行业性的，比如婴幼儿用品的单品重复购买率比较高，但相同的包包的重复购买率就不算高。对于一位正在育儿的母亲，我们可以给她重复推荐符合她们偏好的、相同的奶粉和尿片，但对于一位时尚的女孩，我们向她重复推荐相同的包包可能就不合适了。

经过广泛的市场需求和交流，我们要求百分点推荐引擎能够从方方面面支持客户的市场营销策略，概括的讲主要包括：

- 科学高效的推荐算法，并且根据网站特点选择最佳的推荐算法和推荐策略；
- 根据用户的全网行为分析他们的潜在偏好，帮助网站实现站内站外精准营销；
- 根据全网的商品和资讯信息分析各种内容之间的相关度，帮助网站优化站外流量导入工作。

特别专题

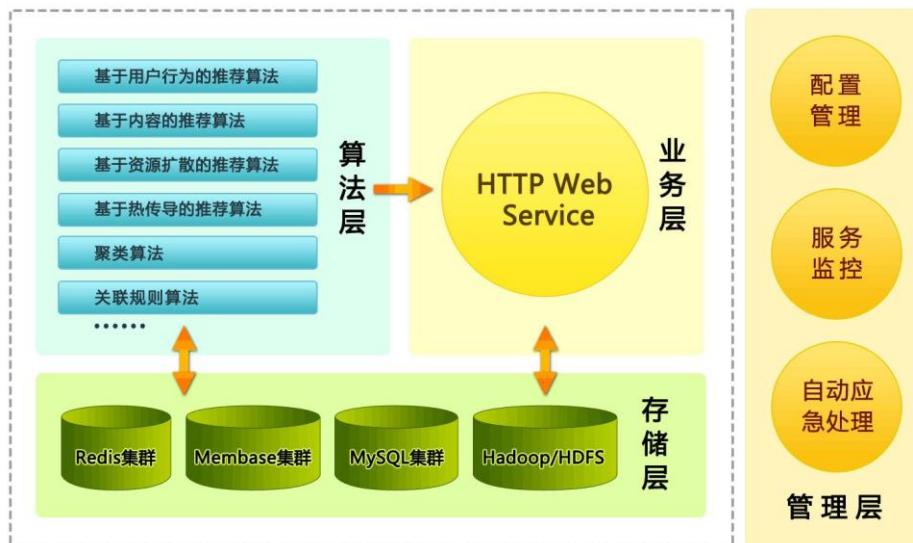
百分点推荐引擎——从需求到架构

作者 [百分点科技推荐引擎研发部](#)

百分点推荐引擎面对的是全网的商品资讯信息以及用户行为，如何科学有效的利用这些数据为电子商务和资讯网站提供丰富的推荐服务，满足其推广营销目标，成为了我们最大的技术挑战。为此我们对百分点推荐引擎提出了以下技术要求：

- 支持各种推荐算法和科学衡量指标。研究人员们已经提出了数百种推荐算法以及相应的标准数据集和推荐效果衡量指标，百分点推荐引擎必须足够灵活以便能够支持这些算法。而且我们要明确每种算法在各个数据集上的性能指标，以便为具体需求选择合适的推荐算法。
- 大数据处理。面对全网资源和用户行为，如何安全可靠的存储和分析这些数据是非常关键的。我们的最低要求是每天能够处理 1 亿级别的数据输入和推荐请求，并且保证数据绝对安全。显然，分布式和云服务是我们唯一的选择。
- 高可用性和实时性。作为一个 Web Service 提供商，提供稳定可靠低延时的服务是基本要求，我们从用户体验角度出发，要求每个推荐请求都能在 2ms 内处理完成。
- 可扩展性。这是所有计算机系统的普遍需求，我们要求百分点推荐引擎可以很方便的添加各种新的推荐逻辑，提供新的推荐服务。并且当整个系统需要升级扩容的时候，人力和硬件成本是线性可控的。
- 便于管理。运维是 Web Service 的重头戏，我们要求百分点推荐引擎中的各个部件（或逻辑单元）都是独立可拆卸可替换的，每个部件都要有完善的容灾备份恢复机制，这样整个系统的管理工作逐步细分，有利于分工协作。

架构设计



根据上节提出的需求，我们将百分点推荐引擎设计为一组云服务的有机组合，如上图，百分点推荐引擎可以分为存储层、业务层、算法层和管理层四大功能组件。每个组件内部又可以细分为更小的单元，或者服务模块，提供基本的存储或运算服务。单元与单元之间尽量解耦合，仅通过 API 协议进行协作，这样一个单元的升级变动带来的影响是可控的。每个单元都要做到可靠可用。下面，我们全面介绍百分点推荐引擎四大功能组件。

存储层

存储层提供基本的数据存取服务，并做好备份和灾难恢复工作，以保证数据的安全可靠。根据不同的应用需求，存储层细分为 Redis 集群，Membase 集群，MySQL 集群和 Hadoop/HDFS 四类。

- **Redis 集群。** 百分点推荐引擎采用了 Redis 作为缓存，用于存储热门数据，包括资源（商品或者咨询）ID，名称，链接，图片，分类，品牌等。这些信息数量不算非常多，但是使用频率非常高，基本上我们的每次推荐都要用到数十甚至数百个商品信息。之所以选用 Redis，我们看重的是它的速度，持久化和以及主从机制。目前，我们使用 Redis 的方式是一个 Master 带若干个 Slaves 以便实现读写分离，Master 只负责写，Slaves 只负责读，其中两个 Slave 有序列化机制，并且必定有两个 Slave 在不同的机器上以消除单点故障隐患。
- **Membase 集群。** Membase 在百分点推荐引擎中扮演了主存的角色，主要用于支持百分点推荐引擎的计算。目前，百分点推荐引擎包含了

大大小小十多个在线和离线计算模块，这些模块计算过程中需要用到很多数据，并产生以及大量的中间结果，包括用户在各个网站的行为历史，资源之间的关系等等。这些数据的特点是不需要 Schema，数量多，但绝大多数的使用频率很低。之所以选用 Membase，主要原因是因为它可以很方便的进行横向扩展以及有丰富的 Client API 支持。

- MySQL 集群。在百分点推荐引擎的最初阶段，我们赋予 MySQL 的主要任务是存储所有客户的原始数据（包括用户行为，推荐请求及推荐结果等）以作备份之用，并在后期统计推荐效果。但很快我们就发现 MySQL 数据库变得极其庞大，以至于每周都需要对其进行压缩备份和切割，运维工作量太大。现在，我们已经将数据备份和后期统计工作转移到了 Hadoop/HDFS 平台，只在 MySQL 中存储最终的统计数据以及其他客户配置信息等小规模的数据。由于 MySQL 的任务量不重，我们仅对其做了双机热备以避免单机崩溃造成无法继续服务。
- Hadoop/HDFS。正如前面所说，目前我们使用 Hadoop/HDFS 来存储客户的原始数据，并在其上做一些统计处理。另外，我们也在计划将一些离线算法和数据转移到 Hadoop 平台上以便发挥 Hadoop 的潜力。Hadoop 的 NameNode 存在单点故障隐患，为此我们为建立了一个备份的 NameNode，并在主服务器出现问题时将服务切换至备份服务器上。

算法层

这是百分点推荐引擎最核心也是最具挑战性的部分，我们将这一层设计为一系列抽象算法的集合。我们深入研究了学术界在基于用户行为的推荐算法，基于内容的推荐算法和关联规则等多方面的理论知识，在此之上自主研发了十多种适用于大数据处理的在线和离线推荐算法。目前，我们的在线算法包括协同过滤（UserBased/ItemBased CF），基于内容的推荐（Content Based），热扩散（Heat Diffusion），用户行为模式分析（Behavior PatterAnalysis）等等。离线算法包括 KNN 聚类，基于 FP Tree 的关联规则挖掘，基于上下文统计的关联规则挖掘，序列模式算法，文档建模算法等等。

算法层并不关心具体的业务逻辑，而只负责数据处理和结果返回。以热扩散算法为例，一方面它接受（用户，资源，偏好指数）的三元组作为计算输入，实时计

算用户与用户/资源之间的关系；另外，我们也可以向它请求某个用户对哪些资源最感兴趣，或者某个资源与哪些资源最相关。

将业务逻辑和推荐算法本身剥离这样的设计方式使得推荐算法具有了最大的通用性，也保证了前端的推荐功能模块可以根据逻辑需求综合多个算法。以百分点推荐引擎的“基于浏览历史的个性化推荐”为例，它就使用到了热扩散和基于内容的推荐两种算法。

得益于存储和算法的分离，算法层并不需要考虑数据的备份容灾等问题。这样，如果某个算法模块由于服务器故障出现异常，我们可以很快在另外的服务器上启动同一个它的一个备份来替换它，而不涉及任何数据迁移问题，最大限度满足了可用性。

业务层

这是百分点推荐引擎中直接面对客户的部分，也就是我们的 HTTP Web Service，它主要负责两件事：收集客户提交的数据，并将其转换为各个推荐算法需要的输入数据，交由推荐算法计算；根据客户提交的推荐请求，向一个或几个推荐算法请求数据，并转换为客户需要的数据格式。可以看出，业务层起到了连接具体需求与推荐算法，真实世界与计算机世界之间的作用。

以“购买过该商品的用户还购买过哪些商品”为例，我们来简介这个推荐功能模块是如何沟通客户需求和推荐算法。目前我们主要采用热扩散算法来实现这个推荐功能模块。首先，客户提交购买数据时，百分点推荐引擎会根据一定的业务逻辑将这个事件处理为算法可以接受的三元组。例如用户 U 购买了商品 K，我们可能会向算法发送一个输入数据(U, K, 1.0)。其次，当客户请求买过 K 的用户还买过哪些商品时，我们一方面以 K 作为参数向算法请求与 K 最相近的资源；另一方面如果客户提交了用户 ID，我们还会向算法请求该用户可能感兴趣的商品；最后我们将两个结果加权整合，挑选其中权重最大同时满足客户额外需求（例如过滤用户的购买历史，按照商品类别/价格过滤等）的几个商品作为最终推荐结果。

可见，业务层完全将推荐算法作为黑盒子来使用，这样业务层就可以集中注意力满足客户多种多样的需求。另外，同算法层一样，业务层也无须关心数据的存储备份和容灾。

管理层

在百分点推荐引擎中，管理层负责内部 DNS，配置管理，服务部署，服务监控和自动应急处理。

- 内部 DNS 是实现高可用性的重要环节。百分点推荐引擎的各个组件都是通过内部域名访问其他服务的，所有服务器的主次 DNS 也都设置成了内部 DNS。这样，当有关键的服务器，例如 Hadoop 的 NameNode，出现故障时，我们可以通过修改域名对应的 IP 保证服务不间断。
- 配置管理。这个模块的主要功能是实现配置的自动化更新和通知。我们曾经考虑过用 Zookeeper 来实现这一功能，但后来觉得 Zookeeper 太过重型，于是自己根据自己的需求开发了一个配置管理服务。百分点推荐引擎的内部服务可以将自己注册在配置管理的某个项目下，在改配置项变动时，配置管理模块会通知该服务以便其获得最新的配置信息。
- 服务监控。这个模块主要用于监控服务器的健康状况，各个进程是否能够正常提供服务，并在出现异常情况时执行短信报警和触发自动应急处理。我们的方法包括：
 - 通过 top, ps, free 等一些基本工具来查看系统负载以及各个进程是否存活，CPU, Memroy 等资源占用情况。利用 redis-cli, memstats 等特定工具来查看 Redis, Membase 的运行状况。
 - 对于自主开发的程序，我们都要求提供一个可供测试的调用，这个调用可以走完主要的服务流程，并返回执行流程中是否出现异常，例如配置项设置错误，执行流程超时等等。
 - 我们会对各个服务输出的 LOG 进行分析，找出异常状况。例如短期内出现大量 EXCEPTION 或者 ERROR，请求处理时间超长，大量推荐请求得不到结果等等。
 - 监控模块一旦检测到异常情况，会立即短信通知我们的运维人员，并通知自动应急处理模块尝试修复异常。
 - 自动应急处理。我们在自动应急模块中实现了修改 DNS 配置，启动/停止业务层服务程序和推荐算法的功能。举个例子，当 MySQL 主服务器宕机时，自动应急模块会收到来自监控模块的通知，而后它会尝试修改将主从 DNS 中的 MySQL 服务器域名修改为 MySQL

从服务器的 IP ; 又或者如果自动应急模块收到监控模块的通知说业务层某个服务进程在连续的 1 分钟内一直占用了 100% 的 CPU , 应急模块会将它 kill 掉并重新启动 , 因为很可能是该进程出现异常了。

小结

本文较为细致的介绍了百分点推荐引擎的总体架构和功能划分 , 不难看出 , 在整个架构设计中 , 我们一直坚持模块化 , 低耦合 , 消除单点等原则 , 力求将百分点推荐引擎打造成扩展性和可靠性极佳的推荐技术平台。经过近两年的多个大中型电子商务合作网站的实践检验 , 这套架构完全满足了我们一开始提出的各项需求 , 而且在可见的未来内 , 它也足以胜任百分点推荐引擎的战略规划。这套架构在稳定性和灵活性等多方面体现了出了百分点推荐引擎团队在推荐技术和服务上积极的努力耕耘和领先的技术。

关于作者

百分点科技推荐引擎研发部由 40 余名技术精英组成 , 绝大多数具有国内外知名院校硕士及以上学位 , 拥有丰富的国内外互联网企业的研发经验 , 是国内领先的推荐引擎架构设计研发团队。该团队在推荐引擎算法 , 大数据平台和推荐技术应用等领域所做出的一系列创新成果 , 已成功服务于国内多家著名电子商务企业 , 显著提升了电商企业的运营绩效 , 也奠定了百分点科技在推荐引擎技术领域的领先地位。

原文链接 :

<http://www.infoq.com/cn/articles/baifendian-recommendation-engine>

相关内容 :

- [点点网如何出产高品质内容](#)
- [社会化推荐在人人网的应用](#)

QClub

我们影响有影响力的人

北京 上海 广州 大连 西安 太原 成都 杭州 武汉 南京 深圳...

QClub

邀请
业内知名专家

自由开放的
讨论氛围

定期举办的线下活动

结识
圈内技术好友

InfoQ



中文 | 英文 | 日文 | 葡文 |

特别专题

从赌钱游戏看 PageRank 算法

作者 [于峰](#)

谈到并行计算应用，会有人想到 PageRank 算法，我们有成千上万的网页分析链接关系确定排名先后，借助并行计算完成是一个很好的场景。长期以来，Google 的创始发明 PageRank 算法吸引了很多人学习研究，据说当年 Google 创始者兴奋的找到 Yahoo!公司，说他们找到一种更好的搜索引擎算法，但是被 Yahoo!公司技术人员泼了冷水，说他们关心的不是更好的技术，而是搜索的盈利。后来 Google 包装成了“更先进技术的新一代搜索引擎”的身份，逐渐取代了市场，并实现了盈利。

由于 PageRank 算法有非常高的知名度和普及度，我们接下来以 PageRank 算法为例讲述“并行计算+数据算法”的经典搭配，并且这种“海量数据并行处理、迭代多轮后收敛”的分析过程也跟其他的数据挖掘或者机器学习算法应用类似，能起到很好的参考作用。

下面是 PageRank 算法的公式：

$$PR(A) = \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \dots \right) q + 1 - q$$

我们其实可以直接阐述该公式本身，并介绍如何使用并行计算套用上面公式得到各网页的 PageRank 值，这样虽然通过并行计算方式完成了 PageRank 计算，但是大家仍然不明白上面的 PageRank 公式是怎么来的。

我们把这个 PageRank 算法公式先放在一边，看看一个赌钱的游戏：

有甲、乙、丙三个人赌钱，他们的输赢关系如下：

甲的钱输给乙和丙

乙的钱输给丙

丙的钱输给甲

例如，甲、乙、丙各有本钱 100 元，按照以上输赢关系，玩一把下来：

甲输给乙 50 元、输给丙 50 元

乙输给丙 100 元

丙输给甲 100 元

如果仅是玩一把的话很容易算出谁输谁赢

但如果他们几个维持这样的输赢关系，赢的钱又投进去继续赌，这样一轮一轮赌下去的话，最后会是什么样子呢？

我们可以写个单机程序看看，为了方便计算，初始本钱都设为 1 块钱，用 x_1 ， x_2 ， x_3 代表甲、乙、丙：

`double x1=1.0,x2=1.0,x3=1.0;`

用 x_{1_income} ， x_{2_income} ， x_{3_income} 代表每赌一把后各人赢的钱，根据输赢关系：

`double x2_income =x1/2.0;`

`double x3 income =x1/2.0+x2;`

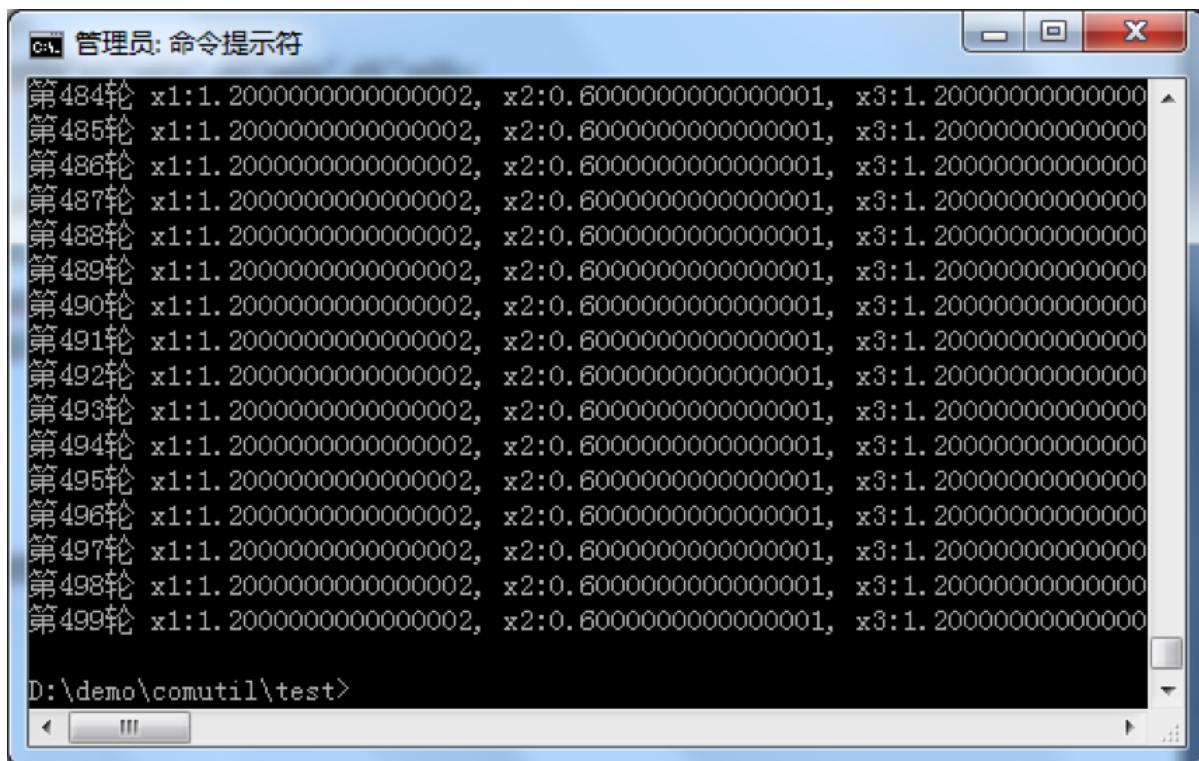
`double x1_ income =x3;`

最后再把各人赢的钱覆盖掉本钱，继续往下算。完整程序如下：

```
// Gamble单机程序
public class Gamble
{
    public static double x1=1.0,x2=1.0,x3=1.0;

    public static void playgame(){
        double x2_income=x1/2.0;
        double x3_income=x1/2.0+x2;
        double x1_income=x3;
        x1=x1_income;
        x2=x2_income;
        x3=x3_income;
        System.out.println("x1:"+x1+", x2:"+x2+", x3:"+x3);
    }
    public static void main(String[] args){
        for(int i=0;i<500;i++){
            System.out.print("第"+i+"轮 ");
            playgame();
        }
    }
}
```

我们运行 500 轮后，看到结果如下：



```

第484轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第485轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第486轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第487轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第488轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第489轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第490轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第491轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第492轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第493轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第494轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第495轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第496轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第497轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第498轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
第499轮 x1:1.2000000000000002, x2:0.6000000000000001, x3:1.2000000000000000
D:\demo\comutil\test>

```

我们发现，从 107 轮后，各人的输赢结果就一直是
 $x_1:1.2000000000000002, x_2:0.6000000000000001,$
 $x_3:1.2000000000000002$

.....

可能你都没想到会有这么个规律，这样一直赌下去，虽然各人每轮有输有赢，但是多轮后的输赢结果居然保持平衡，维持不变了。用技术术语来说就是多轮迭代后产生了收敛，用俗话来讲，就是玩下去甲和丙是不亏的，乙不服输再继续赌下去，也不会有扳本的机会的。

我们再把输赢关系稍微改一下：丙的钱输给甲和乙

```

double x2_income=x1/2.0+x3/2.0;
double x3_income=x1/2.0+x2;
double x1_income=x3/2.0;

```

运行 10000 轮后，发现又收敛了：

$x_1:0.6666666666666667, x_2:1.0, x_3:1.333333333333333$

...

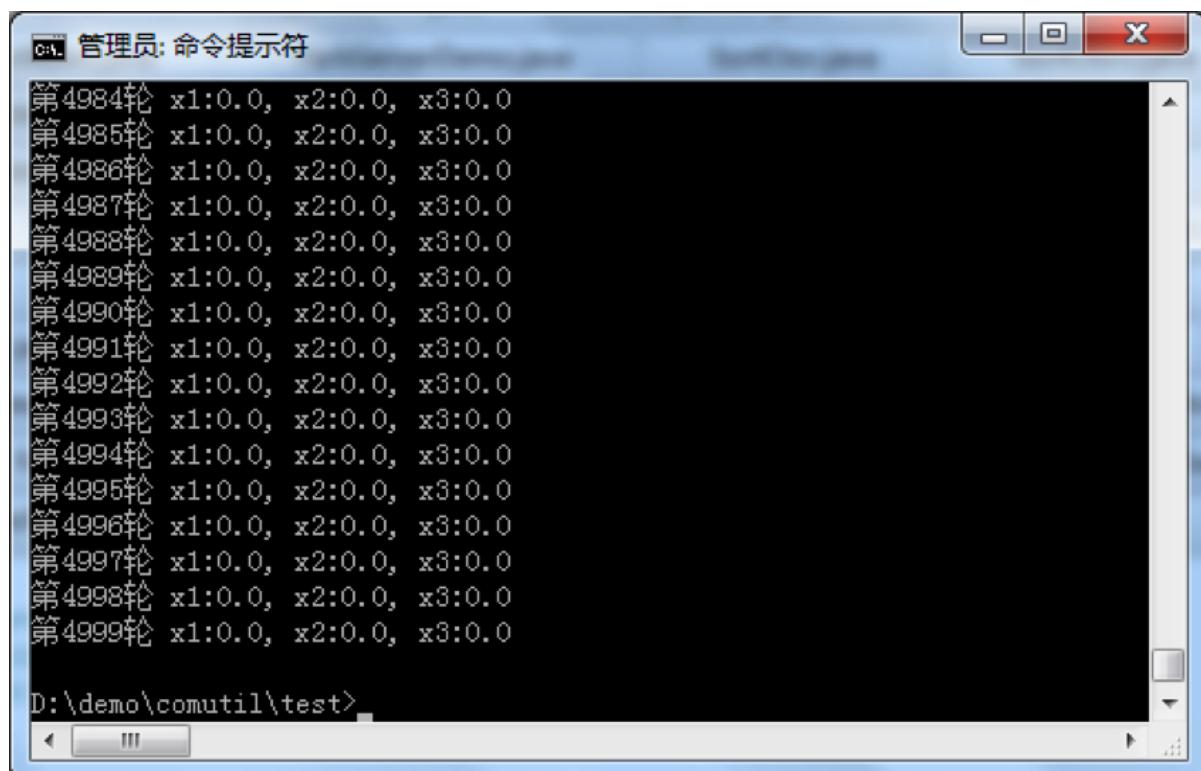
不过这次就变成了“甲是输的，乙保本，丙是赢的”，我们发现收敛的结果可用于排名，如果给他们做一个赌王排名的话，很显然：“丙排第一，乙第二、甲第三”。

那么这样的收敛是在所有情况下都会发生吗，什么情况不会收敛呢？

我们回过头观察上面的输赢关系，甲、乙、丙三人互相各有输赢，导致钱没有流走，所以他们三人才一直可以赌下去，如果把输赢关系改一下，让甲只输钱，不赢钱，如下：

```
double x2_income=x1/2.0+x3/2.0;
double x3_income=x1/2.0+x2;
double x1_income=0;
```

那么运行下来会是什么结果呢？



```
管理员: 命令提示符
第4984轮 x1:0.0, x2:0.0, x3:0.0
第4985轮 x1:0.0, x2:0.0, x3:0.0
第4986轮 x1:0.0, x2:0.0, x3:0.0
第4987轮 x1:0.0, x2:0.0, x3:0.0
第4988轮 x1:0.0, x2:0.0, x3:0.0
第4989轮 x1:0.0, x2:0.0, x3:0.0
第4990轮 x1:0.0, x2:0.0, x3:0.0
第4991轮 x1:0.0, x2:0.0, x3:0.0
第4992轮 x1:0.0, x2:0.0, x3:0.0
第4993轮 x1:0.0, x2:0.0, x3:0.0
第4994轮 x1:0.0, x2:0.0, x3:0.0
第4995轮 x1:0.0, x2:0.0, x3:0.0
第4996轮 x1:0.0, x2:0.0, x3:0.0
第4997轮 x1:0.0, x2:0.0, x3:0.0
第4998轮 x1:0.0, x2:0.0, x3:0.0
第4999轮 x1:0.0, x2:0.0, x3:0.0

D:\demo\comutil\test>
```

我们发现很多轮后，全部为 0 了。我们分析一下过程，第一轮后，甲的钱就输光了，没有赢得一分钱。但是乙和丙各有输赢，他们一直赌到 2000 多轮时，乙的钱全部输光了，甲乙都没钱投进来赌了，导致丙再也赢不到钱了，最后所有人结果都变为 0 了。

我们再分析一下输赢关系，甲的钱全部输给丙和乙后，丙跟乙赌，赢的多输的少，于是所有的钱慢慢都被丙赢走了，导致最后无法维持一个平衡的输赢结果。因此，

如果我们要维持平衡和收敛，必须保证赢了钱的人不准走，必须又输给别人才行，让钱一直在三人圈里转不流失。换句话说，如果存在某人只输不赢，那么这个游戏就玩不下去。

赌钱游戏讲完了，我们再看看 PageRank 算法的公式：

$$PR(A) = \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \dots \right) q + 1 - q$$

上面的 $L(B)$ 代表页面 B 指向其他页面的连接数，我们举个例子：

假设有 A、B、C 三张网页，他们的链接关系如下：

A 包含 B 和 C 的链接

B 包含 C 的链接

C 包含 A 的链接

根据上面的公式，得到各网页 PR 值如下：

$$PR(B)=PR(A)/2;$$

$$PR(B)=PR(A)/2+PR(C);$$

$$PR(A)=PR(C);$$

可以回过头对照一下，把 A、B、C 改成甲、乙、丙就是上面举的赌钱游戏例子。

那么 q 是干吗的？公式里的 q 叫做逃脱因子，名字很抽象，目的就是用于解决上面赌钱游戏中“只输不赢”不收敛的问题， $1-q$ 会保证其中一个 PR 值为 0 时计算下来不会全部为 0，那么加了这么一个 $(...)*q+1-q$ 的关系后，整体的 PR 值会变化吗？

当每个页面的初始 PR 值为 1 时， $0 <= q <= 1$ （计算时通常取值 0.8），我们把所有页面的 PR 值相加看看，假设有 n 张网页：

$$\begin{aligned} & PR(x_1) + PR(x_2) + \dots + PR(x_n) \\ &= ((PR(x_2)/L(x_2) + \dots)q + 1 - q) + \dots + ((PR(x_1)/L(x_1) + \dots)q + 1 - q) \\ &= (PR(x_1)L(x_1)/L(x_1) + PR(x_2)L(x_2)/L(x_2) + \dots + PR(x_n)L(x_n)/L(x_n))q \\ &\quad + n(1 - q) \\ &= (PR(x_1) + PR(x_2) + \dots + PR(x_n))q + n - nq \end{aligned}$$

$$= nq + n - n^*q$$

$$= n$$

由于初始 PR 值为 1 ,所以最后所有页面的 PR 值相加结果还是为 n ,保持不变 ,但是加上(...)*q+1-q 的关系后 ,就避免了 PR 值为 0 可以寻求收敛进行排序。

当然实际应用中 ,这个公式还可以设计的更复杂 ,并通过高等代数矩阵旋转求解 ,我们这里只是为了理解原理 ,并不是为了做搜索算法 ,所以就不再深入下去了。

总结 :世界的很多东西都是零和游戏 ,就像炒股 ,股民赚的钱也就是机构亏的钱 ,机构赚的钱也就是股民亏的钱 ,也许股民们应该研究一下 PageRank 算法 ,看看股市起起落落的背后是不是收敛了 ,收敛了说明炒下去永远别想解套 ,而且机构永远不会亏。

如何使用并行计算方式求 PR 值 :

我们这里通过 fourinone 提供的各种并行计算模式去设计 ,思路方法可以有很多种。

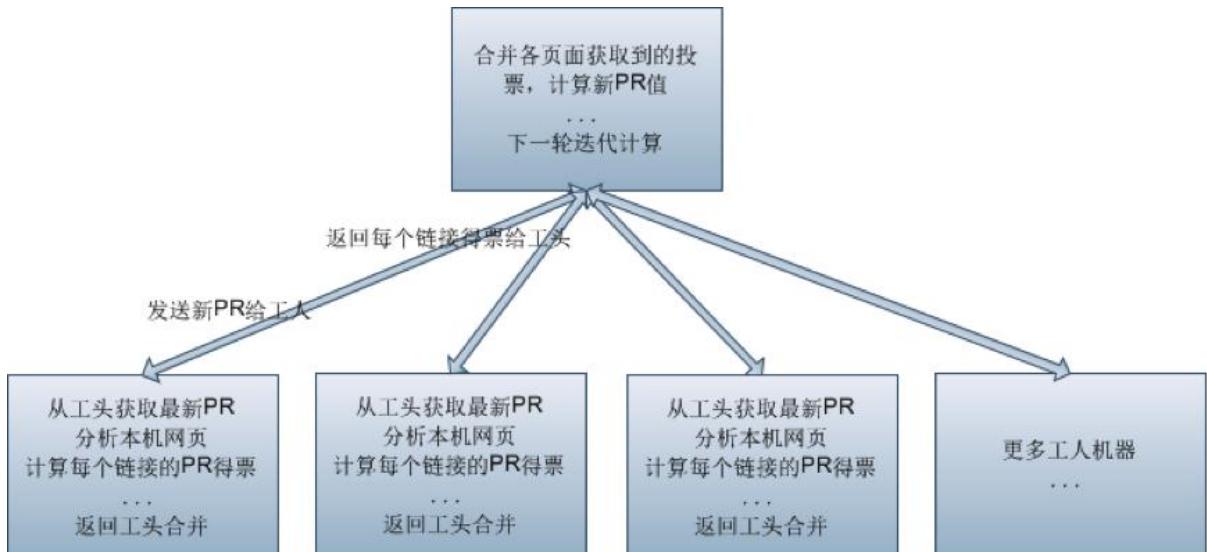
第一次使用可以参考[分布式计算上手 demo 指南](#) ,开发包下载地址 :

<http://www.skycn.com/soft/68321.html>

思路一 :可以采取工人互相合并的机制 (工人互相合并及 receive 使用可参见[sayhello demo](#)) ,每个工人分析当前网页链接 ,对每个链接进行一次 PR 值投票 ,通过 receive 直接投票到该链接对于网页所在的工人机器上 ,这样经过一轮工人的互相投票 然后再统计一下本机器各网页所得的投票数得到新的 PR 值。但是这种方式 ,对于每个链接投票 ,都要调用一次 receive 到其他工人机器 ,比较耗用带宽 ,网页数量庞大链接众多时要调用很多次 receive ,导致性能不高。

思路二 :由于求 PR 值的特点是输入数据大 ,输出数据小 ,也就是网页成千上万占空间多 ,但是算出来的 PR 值占空间小 ,我们姑且用内存可以装下。因此我们优先考虑每个工人统计各自机器上的网页 ,计算各链接对应网页的所得投票 ,然后返回工头统一合并得到各网页的 PR 值。可以采用最基本的 “总一分一总” 并行计算模式实现 (请参考分布式计算上手 demo 指南) 。

并行计算的拆分和合并设计如下 :



可以看到：

工人负责统计各自机器上网页的各个链接的 PR 得票。

工头负责合并累加得到各链接对应网页的新 PR 值，并迭代计算。

程序实现：

PageRankWorker：是一个 PageRank 工人实现，为了方便演示，它通过一个字符串数组代表包括的链接（实际上应该从本地网页文件里获取）

```
links = new String[]{"B", "C"};
```

然后对链接集合中的每个链接进行 PR 投票

PageRankCtor：是一个 PageRank 包工头实现，它将 A、B、C 三个网页的 PageRank 初始值设置为 1.00，然后通过 doTaskBatch 进行阶段计算，doTaskBatch 提供一个栅栏机制，等待每个工人计算完成才返回，工头将各工人返回的链接投票结果合并累加：

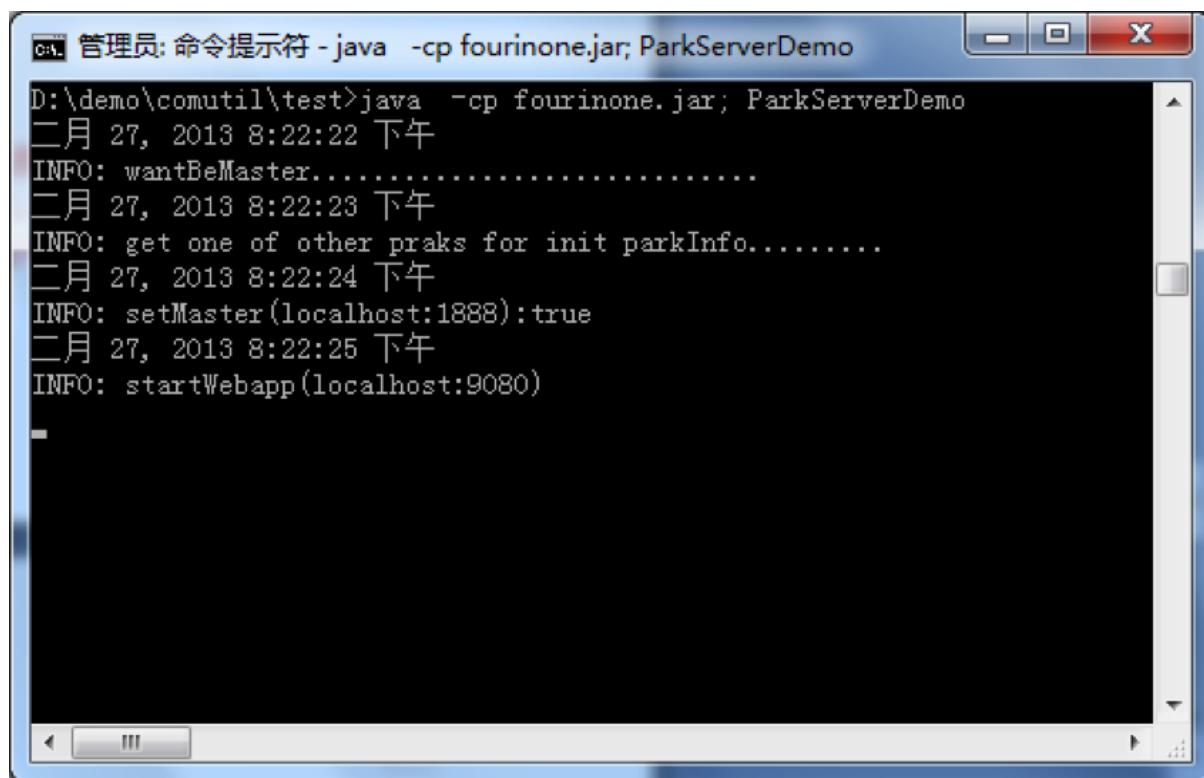
```
pagepr = pagepr+(Double)prwh.getObj(page);
```

得到各网页新的 PR 值（这里取 q 值为 1 进行计算），然后连续迭代 500 轮计算。

运行步骤：

- 启动 ParkServerDemo（它的 IP 端口已经在配置文件指定）

```
java -cp fourinone.jar; ParkServerDemo
```



The screenshot shows a Windows Command Prompt window titled "管理员: 命令提示符 - java -cp fourinone.jar; ParkServerDemo". The window displays the following log output:

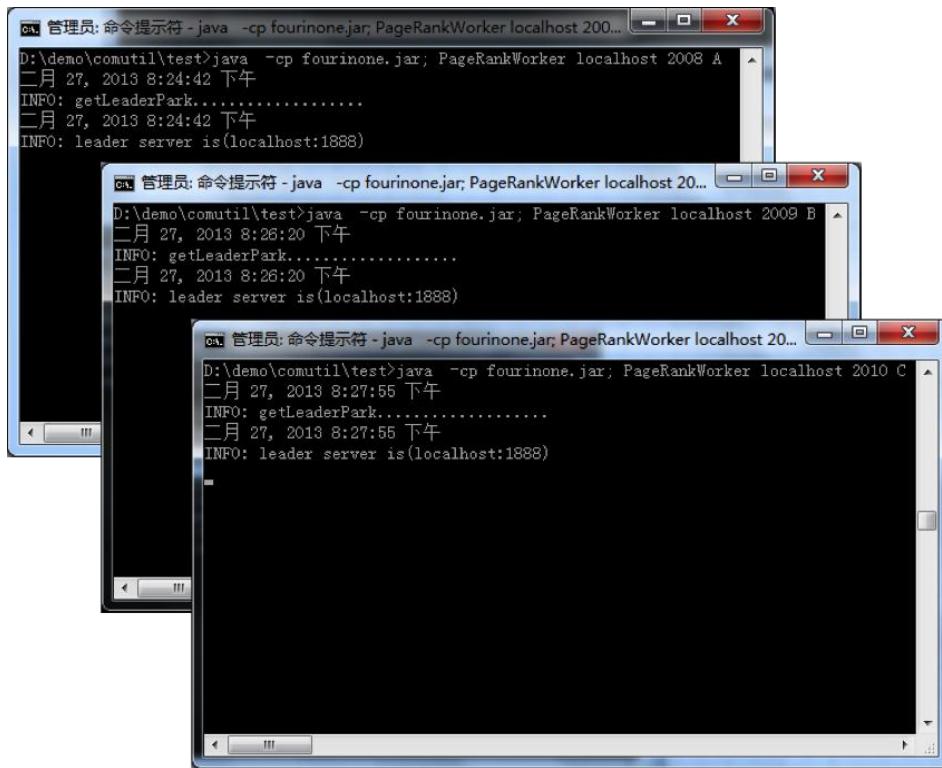
```
D:\demo\comutil\test>java -cp fourinone.jar; ParkServerDemo
二月 27, 2013 8:22:22 下午
INFO: wantBeMaster.....
二月 27, 2013 8:22:23 下午
INFO: get one of other praks for init parkInfo.....
二月 27, 2013 8:22:24 下午
INFO: setMaster(localhost:1888):true
二月 27, 2013 8:22:25 下午
INFO: startWebapp(localhost:9080)
```

2、运行 A、B、C 三个 PageRankWorker，传入不同的 IP 和端口号

```
java -cp fourinone.jar; PageRankWorker localhost 2008 A
```

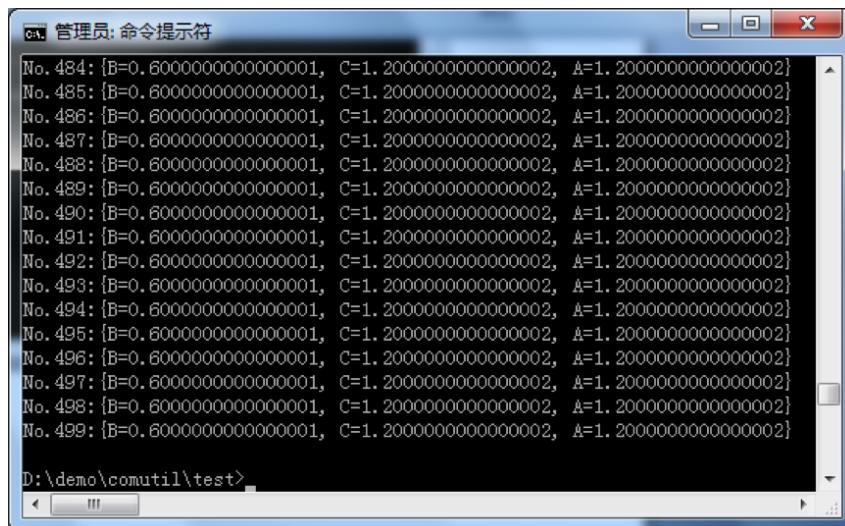
```
java -cp fourinone.jar; PageRankWorker localhost 2009 B
```

```
java -cp fourinone.jar; PageRankWorker localhost 2010 C
```

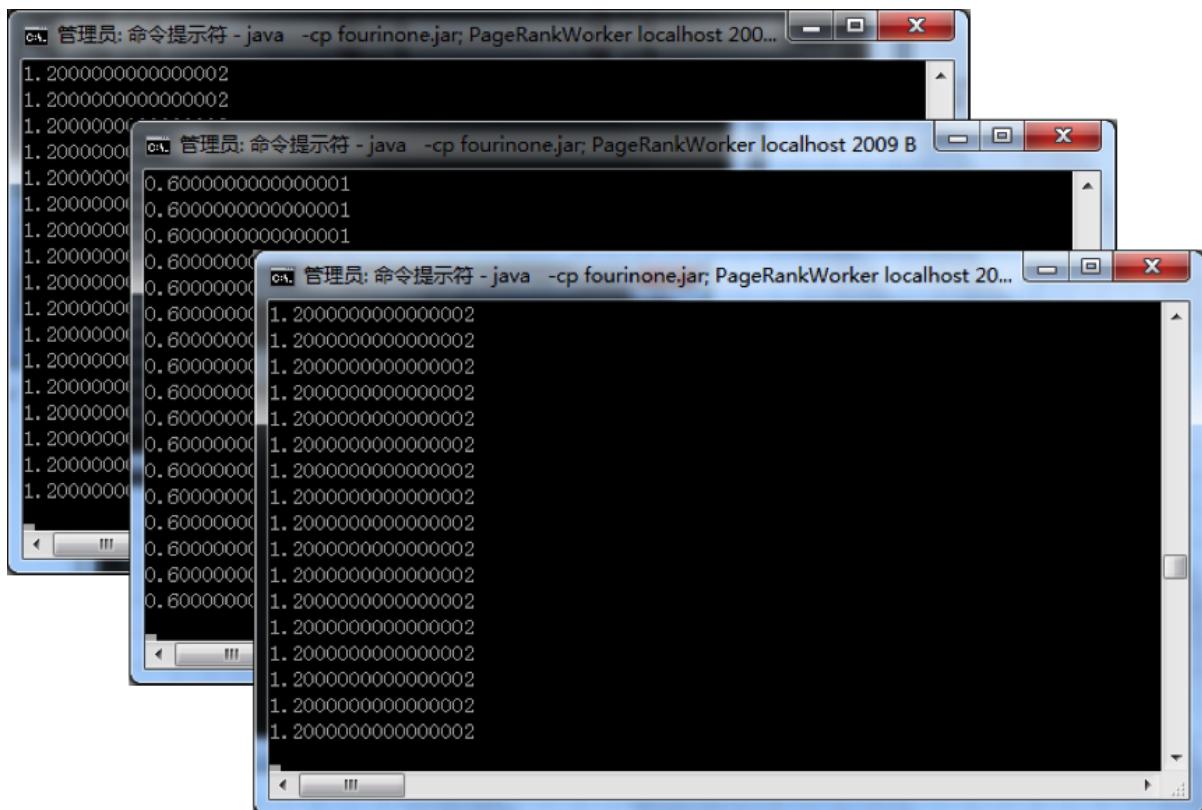


3、运行 PageRankCtor

```
java -cp fourinone.jar; PageRankCtor
```



我们可以看到跟开始的单机程序的结果是一样的 ,同时各工人窗口依次输出了各自的 PR 值 :



完整 demo 源码如下：

```

// ParkServerDemo
import com.fourinone.BeanContext;

public class ParkServerDemo{
    public static void main(String[] args){
        BeanContext.startPark();
    }
}

// PageRankWorker import com.fourinone.MigrantWorker;
import com.fourinone.WareHouse;
import com.fourinone.Workman;

public class PageRankWorker extends MigrantWorker{
    public String page = null;
    public String[] links = null;
    public PageRankWorker(String page, String[] links){
        this.page = page;
        this.links = links;
    }

    public WareHouse doTask(WareHouse inhouse) {
        Double pr = (Double)inhouse.getObj(page);
        System.out.println(pr);
        WareHouse outhouse = new WareHouse();
        for(String p:links)
            outhouse.setObj(p, pr/links.length); //对包括的链接PR投票
        return outhouse;
    }

    public static void main(String[] args){
        String[] links = null;
        if(args[2].equals("A"))
            links = new String[]{"B","C"}; //A页面包括的链接
        else if(args[2].equals("B"))
            links = new String[]{"C"};
        else if(args[2].equals("C"))
            links = new String[]{"A"};
        PageRankWorker mw = new PageRankWorker(args[2],links);
        mw.waitWorking(args[0],Integer.parseInt(args[1]),"pagerankworker");
    }
}

// PageRankCtor
import com.fourinone.Contractor;
import com.fourinone.WareHouse;
import com.fourinone.WorkerLocal;
import java.util.Iterator;

public class PageRankCtor extends Contractor{
    public WareHouse giveTask(WareHouse inhouse){
        WorkerLocal[] wks = getWaitingWorkers("pagerankworker");
        System.out.println("wks.length:"+wks.length);
        for(int i=0;i<500;i++){ //500轮
            WareHouse[] hmarr = doTaskBatch(wks, inhouse);
            WareHouse prwh = new WareHouse();
            for(WareHouse result:hmarr){
                for(Iterator iter=result.keySet().iterator();iter.hasNext()){
                    String page = (String)iter.next();
                    Double pagepr = (Double)result.getObj(page);
                    if(prwh.containsKey(page))
                        pagepr = pagepr+(Double)prwh.getObj(page);
                    prwh.setObj(page,pagepr);
                }
            }
            inhouse = prwh; //迭代
            System.out.println("No."+i+": "+inhouse);
        }
        return inhouse;
    }

    public static void main(String[] args){
        PageRankCtor a = new PageRankCtor();
        WareHouse inhouse = new WareHouse();
        inhouse.setObj("A",1.00d); //A的pr初始值
        inhouse.setObj("B",1.00d); //B的pr初始值
        inhouse.setObj("C",1.00d); //C的pr初始值
        a.giveTask(inhouse);
        a.exit();
    }
}

```

本期专栏

参加 QCon 全球软件开发大会 (北京站) 2013 的十大理由

作者 [彭超](#)

大家都知道 QCon 是由 InfoQ 主办的全球顶级技术盛会，除每年在中国举办之外，东京、纽约、圣保罗、杭州、旧金山都有 QCon 的旗帜飘扬。现在，QCon 已经成为国内技术从业人员心目中最具内容含金量的技术分享圣地。

QCon 北京 2013 将是 QCon 在北京的第五个年头。在每届成功举办的基础上，本届 QCon 史无前例的提前整整半年开始筹备。有必要吗？这这么长的时间里，QCon 项目组做了哪些工作？以下十点，可以帮助您更加了解此届 QCon，同时更加清楚，这将是一次不容错过的技术盛典。

1. 场地更完善：

由于去年 QCon 在京仪大酒店参会人数爆棚，为保证参会体验，我们只能提前停止售票。为给大家提供更大更好的参会场所，今年 QCon 大会的地址挪至了位于鸟巢东南侧的国际会议中心。该会址紧邻四环，靠近地铁口，同时内部设施齐备。不但北京当地的参会人员可以方便到场，外地参会者可方便往返机场的同时，还可以在闲暇时间在奥林匹克公园散步，欣赏北京的特有景观。

2. 主题更全面：

如本文开头所说，此次大会提前半年开始策划准备，最终确认了 16 个专题，不但涵盖了软件开发领域的既有经典，如企业开发、架构探秘、运维、测试，也没有漏掉诸如大数据、云计算这样的热点。这还不是结束，本届大会我们还前瞻性的设置了推荐系统、新锐编程语言、Node.js 专题，这些全面又创新的设置，源于当今变化的技术趋势，和新技术的不断涌现。我们希望参会者在对既有技术深入了解、重新学习的基础上，亦可了解当下技术走向，得到全面的收获。

3. 题材更新颖：

大会就全是讲座？错！[敏捷嘉年华专题](#)，不要听讲！参与者将在敏捷教练和游戏引导师的引导下，体验各种精心策划的游戏和原型实践，从中获得新知与思考。

精彩游戏将从午后一直持续到黄昏 : 极客爱乐高、极客爱折纸、打造 Pizza 团队、挑战最大规模的团队抛球游戏……敏捷社区组织者和志愿者团队，将为 5 周岁的 QCon 北京，奉上与众不同的会议形式。

此次大会还首次策划了[用户体验与产品设计](#)的内容。此专题源自于 QCon 杭州 2012 彦风老师的交互设计演讲。那次演讲的火爆让我们意识到，作为产品的最终实现者和用户，工程师有权，有必要，同时也有热情去了解产品需求和进展以及参与产品讨论。除此之外，工程师还应该增进与设计师的协作，以自身的专业知识弥补设计师对于实现方式了解的不足，从而完善设计。希望这次的内容会再次给大家带来美好的体验。

4. 讲师更重磅：

过去的四个月，除了确认专题设置，我们还做了什么？当然是仔细斟酌邀约业内最佳讲师。除了已确认 JavaScript 泰斗 JSON 发明人 [Douglas Crockford](#)，丁香园 CTO、“小道消息”作者 [冯大辉](#)，CoolShell 博主@左耳朵耗子 [陈皓](#)，持续集成与持续交付专家 [乔梁](#)等若干国内外一流专家以外，此届 QCon 还有尚未揭晓的重磅神秘嘉宾。在 QCon 现场，您将获得与这些重磅嘉宾面对面的机会。

5. 内容更深入：

GitHub 作为业内闪耀的明星，技术架构如何承载膨胀扩张的业务？跨终端的 WebKit 有怎样的渲染机制？百亿规模下如何通过性能测试达到提速效果？双十一购物节如何承载全球最强购物狂潮？推荐系统是否高深，如何成功部署？这些深入话题将在此届 QCon ——为参会者展现。少量空白的议程不是没有牛人，只希望为大家带来最深入的话题，与最合适的人选。

6. 视频更迅速：

毫无保留的发布每一个同步过 slides 与视频时间轴的演讲是 QCon 的特色。然而限于人力物力，我们只能在相对较长的时间内排期发布所有的演讲场次。今年 InfoQ 英文站成功开发出更为快速的视频发布程序，并准备在 QCon 北京预先尝试。借此便利，所有的参会者将会收到唯一的验证码，在大会结束后的 10 天内就可以浏览此届大会的所有视频演讲。

7. 活动更社交：

即便可以看到所有场次的演讲视频，QCon 现场仍对技术人员有着无尽的魅力。这次大会迅速的视频发布机制可以降低参会者错失演讲的担忧。带足名片，磨薄脸皮，在午后活动、在演讲间歇、在晚场活动，大胆的去和同行沟通交流吧。交流机会才是参与现场活动的魅力与精华。

更值得一提的是，GitHub 著名的线下啤酒聚会让无数 GitHuber 望洋兴叹，而借本次 QCon 的机会，GitHub DrinkUp 将首次来到中国，在春意盎然的夜晚，为大家奉上无限量的啤酒。

大会第二天的晚场活动“社区之夜”邀请了知名技术社区，以小团体集会的形式，更为参会者提供目睹社区明星开发者的机会，和更为开阔的社交场所。

8. 晚场更丰富：

除社交化的“社区之夜”之外，此届 QCon 正在悉心策划一个全新的晚场活动——“‘硅谷’传奇”。此活动将邀请著名技术创业领袖授业解惑，满足技术人员的创业好奇心，并将会在现场点评技术团队/创业产品，为创业之路点一盏明灯。

9. 落幕更精彩：

三天的大会内容充实，参会者确实辛苦。往届大会第三天下午纵使内容再精彩，上座率、清醒率均为三日最低。我们体会参会者的感受，所以本届 QCon 我们重新设置了议程，除了在最后一天下午放置干湿结合的主题演讲外，还会引入数位跨界人士，做紧凑又震撼的短篇演讲，充分刺激参会者的神经。最后的最后，我们也无法免俗，将在大会落幕前由嘉宾现场抽取幸运观众，赠送由 InfoQ 诚意准备的各种大小礼品。

10. 地域更广阔：

在说完落幕的事儿之后，必须再给大家做最新预报：QCon 成都会以 QCon 北京姊妹篇的形式亮相大众，西南地区的朋友敬请期待。

这么多理由，是否有打动你的心？欲想了解详细议程，[请点击此处](#)。

本期专栏

云端之道——QCon 北京 2013 云计算专题出品人 网易有道蒋炜航专访

作者 [彭超](#)

蒋炜航目前担任网易技术总监，负责有道云笔记团队。在此之前，他曾就职于惠普实验室和 NetApp 高级技术部，并参与创办 Pattern Insight 公司的 LogInsight 业务(大规模 IT 数据分析产品，于 2012 年 7 月被 VMware 公司收购)。他自 80 年代末起接触编程，02 年从浙江大学获得计算机学士学位，08 年从伊利诺伊斯大学香槟分校获得计算机博士学位。他在大数据处理、云计算、以及分布式系统等方面有多年的经验。

作为 QCon 北京 2013 大会云计算专题出品人，蒋博士接受了 InfoQ 的采访，以下是采访内容。

InfoQ：能否先简单谈谈您在云计算这个领域的从业经验，和您对云计算的理解。

蒋博士 我从 2010 年 6 月加入网易有道伊始开始负责组建有道云存储团队，负责个人云计算产品 8 有道云笔记。

在此之前在硅谷创业参与创办 Pattern Insight 公司的 LogInsight 业务，此业务于 2012 年 7 月被 VMware 公司收购成为其 Big Data Analytics 部门。虽然 LogInsight 不属于云计算，但是其中的技术和云计算所涉及的技术相关。

InfoQ：您认为云计算这个概念是炒作还是大势所趋？

蒋博士：我目前专注与个人云方向，因此主要围绕个人云来阐述我的理解。

认为是炒作的往往由于将云计算和某个技术（比如分布式系统、虚拟化、数据中心等）对等起来。我认为云计算不等同于这些技术，而是一种服务模式。

这种服务模式把复杂的技术、存储、服务包装在云端，让用户很简单的使用这些服务和功能。要实现这个目标要用到很多技术。

个人云计算是大的趋势，因为：

大规模、集中化计算的构想成为现实。 -》 海量存储、大规模数据处理、高性能、低能耗。 -》 千万用户的数据量

终端越来越多样、计算量越来越大。 -》 传统 PC、平板电脑、智能手机、（智能电视、智能汽车、移动健康设备、智能建筑。。。） -》 同步多设备之间的数据，为用户提供一个新的数据中心（代替 PC）

云端能力使得软件“服务化”，不只是封装好的商品。 -》 一些传统软件不易实现的功能 —— 需要大量计算资源、需要大量存储空间、需要大数据分析、需要弹性分配等等。

InfoQ：您如何理解个人云计算服务，什么才是用户对个人云计算服务的核心需求？

“ 蒋博士：个人云计算服务会在 2-3 年内流行起来，改变普通网民工作、学习、娱乐的方式。

用户的核心需求是好用。打个比方，个人云计算就是用一个黑盒子把复杂的技术、存储、服务包装起来，让用户很方便的使用。要做到这点，在三个层面上我们都要做好：

黑盒子要够大、够坚固 —— 也就是云端的基础架构要够稳定、可靠，能够有效的提供大规模的存储和计算能力。

黑盒子要够聪明 —— 也就是云端的云端服务能力要强大灵活，云和端之间的协议要高效，要能够让用户按需租用，等等。

用户端要能和黑盒子一起工作 —— 也就是 Mobile APP 和 PC 客户端都要能够无缝地和云端一起工作。

InfoQ：一个成功云计算平台会有哪些成功要素？达成这些要素的难点在哪里？

“ 蒋博士：成功要素在于云计算相关技术的积累，大众市场的接受程度。而难点在于只有围绕具体的云计算服务，才能促进这两者的发展。

InfoQ：云计算在国内的发展，与国际相比，有哪些差别与差距？

蒋博士：暂时还没有高穿透使用率（high penetration）的个人云应用，有道云笔记希望成为这样的应用。美国市场已经有多个穿透率超过10%的个人云应用。

InfoQ：“高穿透使用率（high penetration）”在国内并不是一个广为人知的概念，能否给大家介绍一下此概念，并讲讲有道云笔记在提高穿透使用率上做的尝试

“蒋博士：穿透使用率就是指在所有的互联网用户中有多少人开始使用个人云计算服务。以存储类云服务来看，艾瑞2012年6月的数据体现中国来看，中国5亿多的互联网用户当中只有小几千万在使用这类个人云计算服务。”

这是符合市场发展规律的——一个新生类型的产品要跨过“从早期尝鲜者到主流用户”这样一个鸿沟。有道云笔记目前已经有近千万用户，就处在一个这样的鸿沟面前。我们做了几件事情来跨过这个鸿沟：

提供符合中国人习惯的产品体验：原笔迹手写、与微博、微信打通。

从技术上降低使用门槛：免登录使用，用qq、微博账号登录等。

通过新媒体营销来教育市场：拍视频（情人节视频、韩寒方舟子视频、末日三行情书视频）；通过SNS来与用户互动，教育使用场景，分享用户故事。

InfoQ：云计算在2012年有哪些值得记录的进展？在未来又有怎样的挑战与展望

“蒋博士：在中国，2012年涌现了一批个人云产品，达到了千万用户量级。相信在2-3年之内，个人云会取代个人电脑，成为用户的数据中心。”

InfoQ：能否评价几款让您印象深刻的个人云计算产品，满足了您哪些需求，还可以如何改进

“蒋博士：

- 有道云笔记：从2011年4月内部alpha版本开始到现在，我总共记了1622篇笔记，大约是平均每天记录2-3篇笔记。这里有我工作的记录、积累下来的资料、平时日常生活中的感悟、旅游计划等等。
- 网易云阅读

InfoQ：未来个人云计算服务会有哪些形态？能否谈谈您的产品构想？

蒋博士：

- “
- a) 个人云计算服务会连接更多种多样的设备：从 PC、智能手机到智能电视、可穿戴智能设备（比如 Nike+ Fuelband）、车载智能设备等等。
 - b) 个人云计算服务会更聪明：从提供基础的存储服务到提供更加智能的云服务。

InfoQ：做为 QCon 中云计算专题的出品人，您希望通过此专题为大家带来哪些实践经验，解决哪些问题？

“

蒋博士：从实际的例子出发了解云计算，了解背后具体的技术，了解这个产业。

InfoQ：非常感谢蒋博士。最后一个问題，如果不从事 IT 行业，您最希望能从事什么工作？

“

蒋博士：不会的，从 80 年代末第一次接触电脑开始就决定一直从事这个行业。



全球软件开发大会

International Software Development Conference

[北京站] 2013

• 大会部分精彩内容呈现 •

JavaScript大师Douglas Crockford:
编程风格和思维观念

GitHub资深运维工程师Jesse Newland:
GitHub运维机器人

《POJO in Action》作者Chris:
分解应用程序以实现可部署性和可扩展性

腾讯资深前端开发专家黄悦:
响应式 Web 设计在跨终端广告创意中的应用

ThoughtWorks韩楷 唱鑫:
大型金融企业项目部署流水线的演化

天猫性能测试专家王德山:
性能提速—百亿级性能测试

Chromium 朱永盛:
跨终端的WebKit渲染机制

全国最大的Python应用:
Sohu邮箱经验分享

eTao高级技术专家袁全:
大规模电商推荐系统应用经验分享

iCosta、AirSlides作者李亮:
iOS Android双版本开发经验谈

腾讯互联网产品运维副总监赵建春:
海量SNS社区网站高效运维探索

人人网技术副总监周欣:
高效能团队的进化之路

阿里巴巴数据平台技术专家罗李:
阿里云梯分布式平台

有道云笔记云端负责人王奉坤:
云笔记的特色云架构

新浪高级开发工程师王达心:
分布式中间层的Node.js实现

百度交互团队负责人香超:
小屏幕，大设计—移动体验设计

北京·国际会议中心

2013年4月23日-24日(会前培训) / 2013年4月25日-27日(大会)

主办方:



www.infoq.com/cn

战略合作伙伴:



www.qconbeijing.com

9折优惠抢购中
5人以上团购更享超值优惠



另有机
会前培训
最低折扣报名中!!

推荐文章

NoSQL 的现状

作者 Stefan Edlich 译者 张卫滨

经过了至少 4 年的激烈争论，现在是对 NoSQL 的现状做一个阶段性结论的时候了。围绕着 NoSQL 发生了如此之多的事情，以至于很难对其作出一个简单概括，也很难判断它达到了什么目标以及在什么方面没有达到预期。

在很多领域，NoSQL 不仅在行业内也在学术领域中取得了成功。大学开始认识到 NoSQL 必须要加入到课程中。只是反复讲解标准数据库已经不够了。当然，这不意味着深入学习关系型数据库是错误的。相反，NoSQL 是很好的很重要的补充。

发生了什么？

NoSQL 领域在短短的 4 到 5 年的时间里，爆炸性地产生了 50 到 150 个新的数据库。nosql-database.org 列出了 150 个这样的数据库，包括一些像对象数据库这样很古老但很强大的。当然，一些有意思的合并正在发生，如 CouchDB 和 Membase 交易产生的 CouchBase。但是我们稍后会在本文中讨论每一个主要的系统。

很多人都曾经假设在 NoSQL 领域会有一个巨大地整合。但是这并没有发生。NoSQL 过去是爆炸性地增长，现在依旧如此。就像计算机科学中的所有领域一样——如编程语言——现在有越来越多的空白领域需要大量的数据库。这是与互联网、大数据、传感器以及将来很多技术的爆炸性增长同步的，这导致了更多的数据以及对它们进行处理的不同需求。在过去的四年中，我们只看到了一个重要的系统离开了舞台：德国的 Graph 数据库 Sones。为数众多的 NoSQL 依然快乐地生存着，要么在开源社区，不用考虑任何的金钱回报，要么在商业领域。

可见性与金钱？

另外一个重要的方面就是可见性与行业采用的情况。在这个方面，我们可以看到在传统的行业中——要保护投资——与新兴的行业（主要是初创公司）之间有

很大的差别。几乎所有热门的基于 Web 的创业公司如 Pinterest 和 Instagram 都在使用混合式 (SQL + NoSQL) 的架构，而传统的行业依然纠结于是否采用 NoSQL。但是观察显示，越来越多这样的公司正在试图将它们的一部分数据流用 NoSQL 方案进行处理并在以后进行分析，这样的方案包括 Hadoop、MongoDB 以及 Cassandra 等。

这同时导致了对具备 NoSQL 知识的架构师和开发人员的需求持续增长。[最近的调查](#)显示行业中最需要的开发人员技能如下：

- HTML5
- MongoDB
- iOS
- Android
- Mobile Apps
- Puppet
- Hadoop
- jQuery
- PaaS
- Social Media

在前十名的技术需求中，有两个 NoSQL 数据库。有一个甚至排在了 iOS 前面。如果这不是对它的赞扬，那是什么呢？！

但是，跟最初预计相比，对 NoSQL 的采用变得越来越快，越来越深入。在 2011 年夏天，Oracle 曾经发布过一个著名白皮书，它提到 NoSQL 数据库感觉就像是冰淇淋的风味，但是你不应该过于依附它，因为它不会持续太长时间。但是仅仅在几个月之后，Oracle 就展现了它们将 Hadoop 集成到大数据设备的方案。甚至，他们建立了自己的 NoSQL 数据库，那是对 BerkeleyDB 的修改。从此之后，所有的厂商在集成 Hadoop 方面展开了竞赛。Microsoft、Sybase、IBM、Greenplum、Pervasive 以及很多的公司都已经对它有了紧密的集成。有一个模式随处可见：不能击败它，就拥抱它。

但是，关于 NoSQL 被广泛采用的另一个很重要但不被大家关注的重要信号就是 NoSQL 成为了一个 PaaS 标准。借助于众多 NoSQL 数据库的易安装和管理，像 Redis 和 MongoDB 这样的数据库可以在很多的 PaaS 服务中看到 如 Cloud

Foundry、OPENSHIFT、dotCloud、Jelastic 等。随着所有的事情都在往云上迁移，NoSQL 会对传统的关系型数据库产生很大的压力。例如当面临选择 MySQL/PostGres 或 MongoDB/Redis 时 将会强制人们再三考虑他们的模型、需求以及随之而来的其他重要问题。

另外一个很有意思的技术指示器就是 ThoughtWorks 的技术雷达，即便你可能不完全同意它所包含的所有事情，但它总会包含一些有意思的事情。让我们看一下他们 2012 年 10 月份的技术雷达，如图 1：

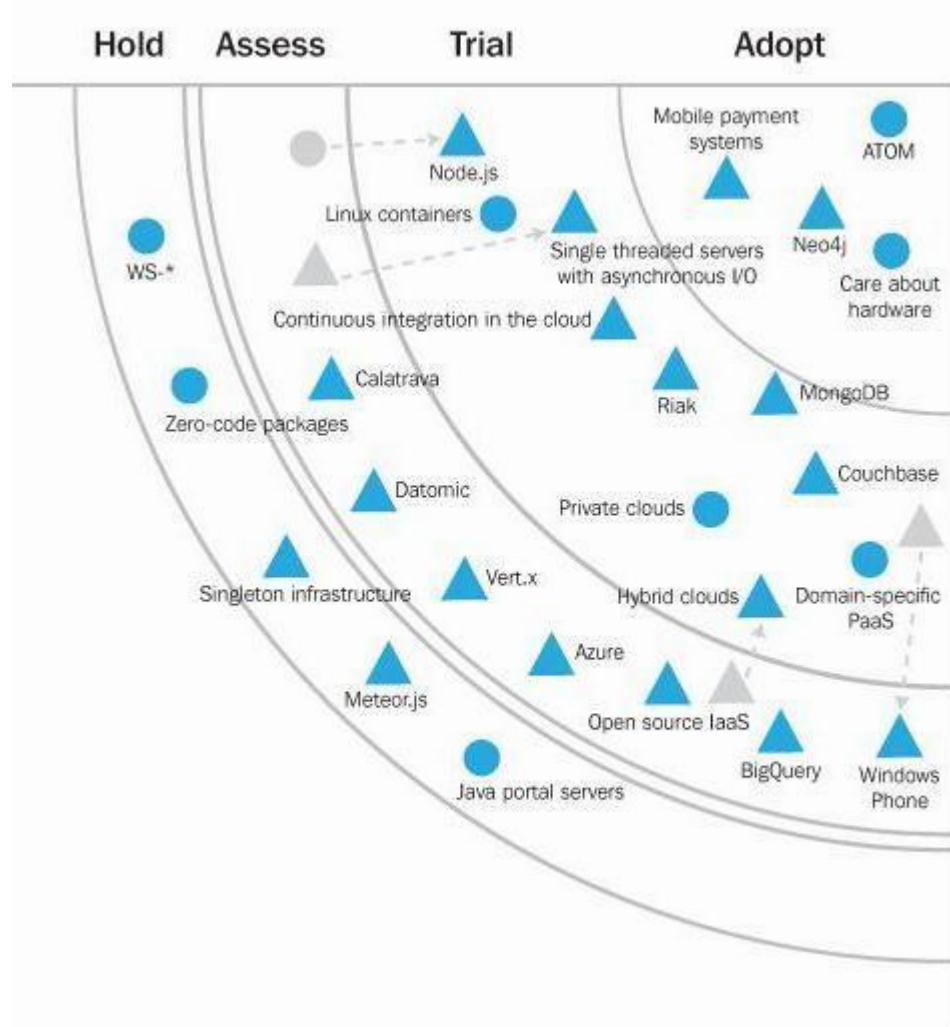


图 1：ThoughtWorks 技术雷达，2012 年 10 月——平台

在他们的平台象限中，列出了 5 个数据库：

- 1、 Neo4j (采用)
- 2、 MongoDB (试用阶段但是采用)
- 3、 Riak (试用)

-
- 4、 CouchBase (试用)
 - 5、 Datomic (评估)

你会发现它们中至少有四个获得了很多的风险投资。如果你将 NoSQL 领域的所有风险投资加起来，结果肯定是在一亿和十亿美元之间！Neo4j 就是一个例子，它在一系列的 B 类资助中得到了一千一百万美元。其他得到一千万到三千万之间资助的公司是 Aerospike、Cloudera、DataStax、MongoDB 以及 CouchBase 等。但是，让我们再看一下这个列表：Neo4j、MongoDB、Riak 以及 CouchBase 已经在这个领域超过四年了并且在不断地证明它们是特定需求的市场领导者。第五名的数据库——Datomic——是一个令人惊讶的全新数据库，它是由一个小团队按照全新的范式编写的。这一定是很热门的东西，在后面简要讨论所有数据库的时候，我们更深入地了解它们。

标准

已经有很多人要求 NoSQL 标准了，但他们没有看到 NoSQL 涵盖了一个范围如此之大的模型和需求。所以，适用于所有主要领域的统一语言如 Wide Column、Key/Value、Document 和 Graph 数据库肯定不会持续很长时间，因为它不可能涵盖所有的领域。有一些方式，如 Spring Data，试图建立一个统一层，但这取决于读者来测试这一层在构建多持久化环境时是不是一个飞跃。

大多数的 Graph 和 Document 数据库在它们的领域中已经提出了标准。在 Graph 数据库世界，因为它的 tinkerpop blueprints、Gremlin、Sparql 以及 Cypher 使得它更为成功一些。在 Document 数据库领域，UnQL 和 jaql 填补了一些位置，尽管前者缺少现实世界 NoSQL 数据库的支持。但是借助 Hadoop 的力量，很多项目正在将著名的 ETL 语言如 Pig 和 Hive 使用到其他 NoSQL 数据库中。所以标准世界是高度分裂的，但这只是因为 NoSQL 是一个范围很广的领域。

格局

作为最好的数据库格局图之一，是由 [451 Group 的 Matt Aslett 在一个报告中](#) 给出的。最近，他更新了该图片从而能够让我们可以更好地深入理解他所提到的分类。你可以在下面的图片中看到，这个格局是高度碎片化和重叠的：

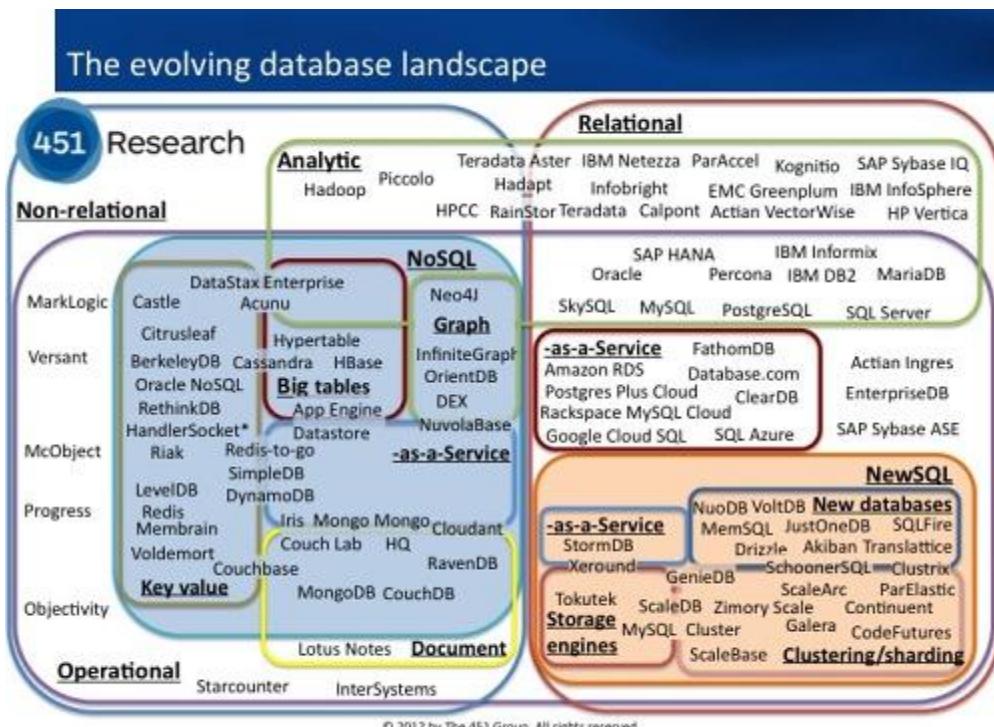


图 2 : Matt Aslett (451 Group) 给出的数据库格局

你可以看到在这个图片中有多个维度。关系型的以及非关系型的、分析型的以及操作型的、NoSQL 类型的以及 NewSQL 类型的。最后的两个分类中，对于 NoSQL 有著名的子分类 Key-Value、Document、Graph 以及 Big Tables，而对于 NewSQL 有子分类 Storage-Engine、Clustering-Sharding、New Database、Cloud Service Solution。这个图有趣的地方在于，将一个数据放在一个精确的位置变得越来越难。每一个都在拼命地集成其他范围数据库中的特性。NewSQL 系统实现 NoSQL 的核心特性，而 NoSQL 越来越多地试图实现“传统”数据库的特性如支持 SQL 或 ACID，至少是可配置的持久化机制。

这一切都始于众多的数据库都提供与 Hadoop 进行集成。但是，也有很多其他的例子，如 MarkLogic 开始参与 JSON 浪潮，所以也很难对其进行定位。另外，更多的多模型数据库开始出现，如 ArangoDB、OrientDB 和 AlechemyDB（现在它是很有前途的 Aerospike DB 的一部分）。它们允许在起始的时候只有一个数据库模型（如 document/JSON 模型）并在新需求出现的时候添加新的模型（Graph 或 key-value）。

另外一个证明它开始变得成熟的标志就是图书市场。在 2010 年和 2011 年两本德语书出版之后，我们看到 Wiley 出版了 Shashank Tiwari 的书。它的结构很棒并且饱含了深刻伟大的见解。在 2012 年，这个竞赛围绕着两本书展开。“七周七数据库”（Seven Databases in Seven Weeks）当然是一本杰作。它的特点在于新颖的编写以及实用的基于亲身体验的见解：它选取了 6 种著名的 NoSQL 数据库以及 PostGreSQL。这些都使得它成为一本高度推荐的图书。另一方面，P.J. Sandalage 以及 Martin Fowler 采取了一种更为全面的方法，涵盖了所有的特征并帮助你评估采用 NoSQL 的路径和决策。

但是，会有更多的书出现。Manning 的书出现在市场上只是个时间问题：Dan McCreary 和 Ann Kelly 正在编写一本名为“[Making Sense of NoSQL](#)”的书，首期的 MEAP（指的是 Manning Early Access Program——译者注）章节已经可以看到。

在介绍完理念和模式后，他们的第三章看起来保证很有吸引力：

- 构建 NoSQL 大数据解决方案
- 构建 NoSQL 搜索解决方案
- 构建 NoSQL 高可用性解决方案
- 使用 NoSQL 来提高敏捷性

只是一个全新的方式，绝对值得一读。

领导者的现状

让我们快速了解一下各个 NoSQL 的领导者。作为市场上很明显的领导者之一，Hadoop 是一个很奇怪的动物（作者使用这个词，可能是因为 Hadoop 的标识是一只大象——译者注）。一方面，它拥有巨大的发展势头。正如前面所说，每个传统的数据库提供商都急切地声明支持 Hadoop。像 Cloudera 和 MapR 这样的公司会持续增长并且新的 Hadoop 扩展和继承者每周都在出现。

即便是 Hive 和 Pig 也在更好地得到接受。不过，有一个美中不足之处：公司们依然在抱怨非结构化的混乱（读取和解析文件本应该更快一些），MapReduce 在批处理上做的还不够（甚至 Google 已经舍弃了它），管理依旧很困难，稳定性问题以及在本地很难找到培训/咨询。即便你可以解决一些上面的问题，如果 Hadoop 继续像现在这样发展或发生重大变化的话，它依然会是热点问题。

第二位领导者，MongoDB，同样面临激烈的争论。处于领导地位的数据库会获得更多的批评，这可能是很自然的事情。不过，MongoDB 经历了快速的增长，它受到的批评主要如下：

- a)就老版本而言或者
- b)缺少怎样正确使用它的知识。尽管 MongoDB 在下载区域清楚地表明 32 位版本不能处理 2GB 的数据并建议使用 64 位版本，但这依然受到了很多近乎荒谬的抱怨。

不管怎样，MongoDB 合作者和资助者推动了雄心勃勃的发展路线，包含了很多热门的东西：

- 行业需要的一些安全性/LDAP 特性，目前正在开发
- 全文本搜索很快会推出
- 针对 MapReduce 的 V8 将会推出
- 将会出现比集合级别更好的锁级别
- Hash 分片键正在开发中

尤其是最后一点吸引了很多架构师的兴趣。MongoDB 经常被抱怨（同时也被竞争对手）没有实现简洁一致的哈希，因为 key 很容易定义所以不能保证完全正确。但在将来，将会有个对 hash 分片键的配置。这意味着用户可以决定使用 hash key 来分片 还是需要使用自己选择分片 key 所带来的优势（可能很少）。

Cassandra 是这个领域中的另一个产品，它做的很好并且添加了更多更好的特性，如更好的查询。但是不断有传言说运行 Cassandra 集群不容易，需要一些很艰难的工作。但这里最吸引人的肯定是 DataStax。Cassandra 的新公司——获得了两千五百万美元的 C 类资助——很可能要处理分析和一些操作方面的问题。尤其是分析能力使得很多人感到惊讶，因为早期的 Cassandra 并没有被视为强大的查询机器。但是这种现状在最近的几个版本中发生了变化，查询功能对一些现代分析来讲已经足够了。

Redis 的开发进度也值得关注。尽管 Salvatore 声明如果没有社区和 Pieter Noordhuis 的帮助，他做不成任何的事情，但是它依旧是相当棒的一个产品。对故障恢复的良好支持以及使用 Lua 的服务器端脚本语言是其最近的成就。使用 Lua 的决策对社区带来了一些震动，因为每个人都在集成 JavaScript 作为服

务器端的语言。但是，Lua 是一个整洁的语言并为 Redis 开启新的潘多拉盒子带来了可能性。

CouchBase 在可扩展性和其他潜在因素方面看起来也是一个很好的选择，尽管 Facebook 以及 Zynga 面临着巨大的风波。它确实不是很热门的查询机器，但如果他们能够在将来提高查询能力，那它的功能就会相当完整了。与 CouchDB 创立者的合并毫无疑问是很重要的一个步骤，CouchDB 在 CouchBase 里面的影响值得关注。在每个关于数据库的会议上，听到这样的讨论也是很有意思的，那就是在 Damien、Chris 和 Jan 离开后，CouchDB 会变得更好呢还是更坏呢？大家在这里只能听到极端的观点。但是，只要数据库做得好谁关心这个呢。现在看起来，它确实做的很好。

最后一个需要提及的 NoSQL 数据库当然是 Riak，在功能性和监控方面它也有了巨大的提升。在稳定性方面，它继续得到巨大的声誉：“像巨石一般稳定可靠且不显眼，并对你的睡眠有好处”。Riak CS fork 在这种技术的模块化方面看起来也很有趣。

有意思的新加入者

除了市场领导者，评估新的加入者通常是很有趣的。让我们深入了解它们中的一部分。

毫无疑问，Elastic Search 是最热门的新 NoSQL 产品，在一系列的 A 轮资助中它刚刚获得了一千万美元，这是它热门的一个明证。作为构建在 Lucene 之上的高扩展性搜索引擎，它有很多的优势：a) 它有一个公司提供服务并且 b) 利用了 Lucene 在过去的多年中已被充分证明的成就。它肯定会比以往更加深入得渗透到整个行业中，并在半结构化信息领域给重要的参与者带来冲击。

Google 在这个领域也推出了小巧但是迅速的 LevelDB。在很多特殊的需求下，如压缩集成方面，它作为基础得到了很多的应用。即使是 Riak 都集成了 LevelDB。考虑到 Google 的新数据库如 Dremel 和 Spanner 都有了对应的开源项目（如 Apache Drill 或 Cloudera Impala），它依然被视为会继续存在的。

另外一个技术变化当然就是在 2012 年初的 DynamoDB。自从部署在 Amazon 中，他们将其视为增长最快的服务。它的可扩展性很强。新特性开发地比较慢但它关注于 SSD，其潜力是很令人振奋的。



easou+

宜搜+ 给你惊喜

全新UI设计，赏心悦目从新开始，APP与浏览器完美结合，打造一站式阅读体验



搜索

简单、便捷、可依靠的手机搜索引擎

订阅

海量内容，随心订制，立刻拥有个人频道

离线下载

WIFI联网时缓存内容，无网络时轻松阅读

个性化+分享

精彩不仅这些，更多乐趣等你发现

多模块数据库也是值得关注的一个领域。最著名的代表者是 OrientDB，它现在并不是新的加入者但它在很迅速地提高功能。可能它变化得太快了，很多使用者也许会很开心地看到 OrientDB 已经到达了 1.0 版本，希望它能更稳定一些。对 Graph、Document、Key-Value 的支持以及对事务和 SQL 的支持，使得我们有理由给它第二次表现的机会。尤其是对 SQL 的良好支持使得它对诸如 Pentaho 这样的分析解决方案方面很有吸引力。这个领域另一个新的加入者是 ArangoDB，它的进展很快，并不畏惧将自己与已确定地位的参与者进行比较。但是，如果有新的需求必须要实现并且具有不同类型的新数据模型要进行持久化的话，对原生 JSON 和 Graph 的支持会省去很多的努力。

到目前位置，2012 年的最大惊喜来自于 Datomic。它由一些摇滚明星采用 Clojure 语言以难以令人置信的速度开发的，它发布了一些新的范式。另外，它还进入了 ThoughtWorks 的技术雷达，占据了推荐关注的位置。尽管它“只是”已有数据库中一个参与者，但是它有很多的优势，如：

- 事务
- 时间机器
- 新颖且强大的查询方式
- 新的模式方式
- 缓存以及可扩展性的特性

目前，支持将 DynamoDB、Riak、CouchBase、Infinispan 以及 SQL 作为底层的存储引擎。它甚至允许你同时混合和查询不同的数据库。很多有经验的人都很惊讶于这种颠覆性的范式转变是如何可能实现的。但幸运的是它就是这样。

总结

作为总结，我们做出三点结论：

- 1、关于 CAP 理论，Eric Brewer 的一些新文章应该几年前就发表。在[这篇 文章中](#)（[这篇佳文的中文版地址](#)——译者注），他指出“三选二”具有误导性，并指出了它的原因，世界为何远比简单的 CP/AP 更为复杂，如在 ACID/BASE 之间做出选择。虽然如此，近些年来有成千上万的对话和文章继续赞扬 CAP 理论而没有任何批评性的反思。Michael Stonebraker 是 NoSQL 最强有力的支持者之一（NoSQL 领域也对他颇

多感激），他在多年前就指出了这些问题！遗憾的是，没有多少人在听。但是，既然 Eric Brewer 更新了他的理论，简单的 CAP 叙述时代肯定要结束了。在指出 CAP 理论的真实和多样性的观点上，请站在时代的前列。

- 2、正如我们所了解的那样，传统关系型数据库的不足导致了 NoSQL 领域的产生。但这也是传统帝国发起回击的时刻。在“NewSQL”这个术语之下，我们可以看到许多新的引擎（如 database.com、VoltDB、GenieDB 等，见图 2），它们提高了传统的解决方案、分片以及云计算方案的能力。这要感谢 NoSQL 运动。

但是随着众多的数据库尝试实现所有的特性，明确的边界消失了确定使用哪种数据库比以前更为复杂了。

你必须要知道 50 个用例、50 个数据库并要回答至少 50 个问题。关于后者，笔者在过去两年多的 NoSQL 咨询中进行了收集，可以在以下地址找到：[选择正确的数据库，在 NoSQL 和 NewSQL 间进行选择](#)。

- 3、一个通用的真理就是，每一项技术的变化——从客户端-服务端技术开始甚至更早——需要十倍的成本才能进行转移。例如，从大型机到客户端-服务端、客户端-服务端到 SOA、SOA 到 WEB、RDBMS 到混合型持久化之间的转换都是如此。所以可以推断出，在将 NoSQL 加入到他们的产品决策上，很多的公司在迟疑和纠结。但是，大家也都知道，最先采用的公司会从这个两个领域获益并且能够快速集成 NoSQL，所以在将来会占据更有利的位置。就这一点而言，NoSQL 解决方案会一直存在并且评估起来会是有利可图的领域。

关于作者



Prof. Dr. Stefan Edlich 是德国柏林 Beuth HS 技术 (University of App. Sc.) 的高级讲师。他为诸多出版社如 Apress、O'Reilly、Spektrum/Elsevier 等编写了超过 10 本 IT 图书。他维护着 [NoSQL Archive](#) 网站，从事 NoSQL 咨询并组织 NoSQL 技术会议，编写了世界上最早的两本 NoSQL 图书，现在他热衷于 Clojure 编程语言。

原文链接：<http://www.infoq.com/cn/articles/State-of-NoSQL>

推荐文章

深入理解 Java 内存模型（四）——volatile

作者 程晓明

volatile 的特性

当我们声明共享变量为 volatile 后，对这个变量的读/写将会很特别。

理解 volatile 特性的一个好方法是：把对 volatile 变量的单个读/写，看成是使用同一个监视器锁对这些单个读/写操作做了同步。下面我们通过具体的示例来说明，请看下面的示例代码：

```
class VolatileFeaturesExample {
    volatile long v1 = 0L; // 使用volatile声明64位的long型变量

    public void set(long l) {
        v1 = l; // 单个volatile变量的写
    }

    public void getAndIncrement () {
        v1++; // 复合(多个) volatile变量的读/写
    }

    public long get() {
        return v1; // 单个volatile变量的读
    }
}
```

假设有多个线程分别调用上面程序的三个方法，这个程序在语意上和下面程序等价：

```

class VolatileFeaturesExample {
    long vl = 0L;           // 64位的long型普通变量

    public synchronized void set(long l) {      //对单个的普通 变量的写用同一个监视器同步
        vl = l;
    }

    public void getAndIncrement () { //普通方法调用
        long temp = get();          //调用已同步的读方法
        temp += 1L;                //普通写操作
        set(temp);                 //调用已同步的写方法
    }

    public synchronized long get() { //对单个的普通变量的读用同一个监视器同步
        return vl;
    }
}
    
```

如上面示例程序所示，对一个 volatile 变量的单个读/写操作，与对一个普通变量的读/写操作使用同一个监视器锁来同步，它们之间的执行效果相同。

监视器锁的 happens-before 规则保证释放监视器和获取监视器的两个线程之间的内存可见性，这意味着对一个 volatile 变量的读，总是能看到（任意线程）对这个 volatile 变量最后的写入。

监视器锁的语义决定了临界区代码的执行具有原子性。这意味着即使是 64 位的 long 型和 double 型变量，只要它是 volatile 变量，对该变量的读写就将具有原子性。如果是多个 volatile 操作或类似于 volatile++ 这种复合操作，这些操作整体上不具有原子性。

简而言之，volatile 变量自身具有下列特性：

- 可见性。对一个 volatile 变量的读，总是能看到（任意线程）对这个 volatile 变量最后的写入。
- 原子性：对任意单个 volatile 变量的读/写具有原子性，但类似于 volatile++ 这种复合操作不具有原子性。

volatile 写-读建立的 happens before 关系

上面讲的是 volatile 变量自身的特性，对程序员来说，volatile 对线程的内存可见性的影响比 volatile 自身的特性更为重要，也更需要我们去关注。

从 JSR-133 开始，volatile 变量的写-读可以实现线程之间的通信。

从内存语义的角度来说，volatile 与监视器锁有相同的效果：volatile 写和监视器的释放有相同的内存语义；volatile 读与监视器的获取有相同的内存语义。

请看下面使用 volatile 变量的示例代码：

```
class VolatileExample {
    int a = 0;
    volatile boolean flag = false;

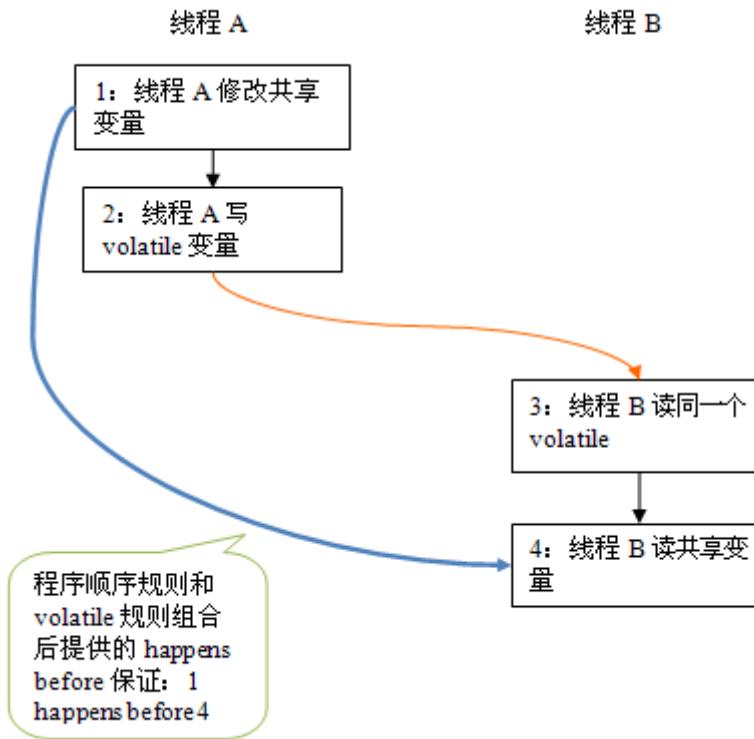
    public void writer() {
        a = 1;                                //1
        flag = true;                            //2
    }

    public void reader() {
        if (flag) {                           //3
            int i = a;                      //4
            ....
        }
    }
}
```

假设线程 A 执行 writer()方法之后，线程 B 执行 reader()方法。根据 happens before 规则，这个过程建立的 happens before 关系可以分为两类：

- 1、根据程序次序规则，1 happens before 2; 3 happens before 4。
- 2、根据 volatile 规则，2 happens before 3。
- 3、根据 happens before 的传递性规则，1 happens before 4。

上述 happens before 关系的图形化表现形式如下：



在上图中，每一个箭头链接的两个节点，代表了一个 happens before 关系。
 黑色箭头表示程序顺序规则；橙色箭头表示 volatile 规则；蓝色箭头表示组合这些规则后提供的 happens before 保证。

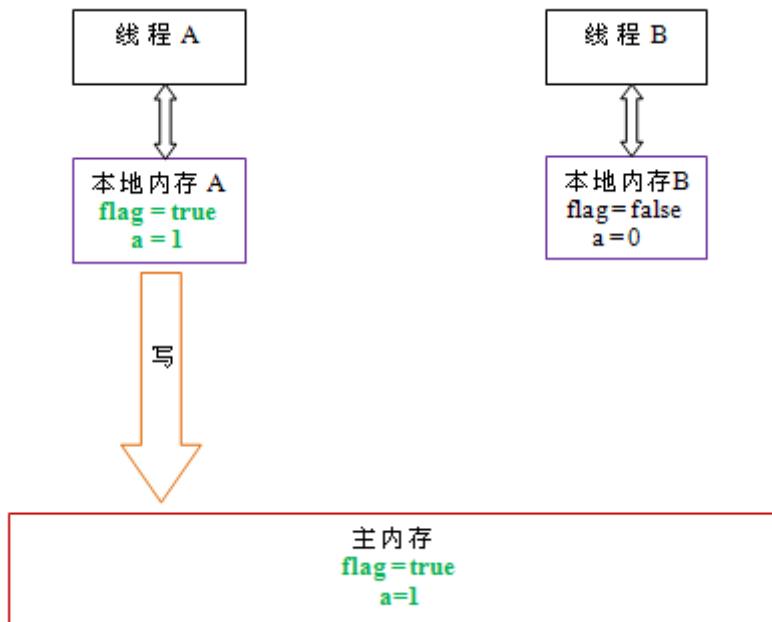
这里 A 线程写一个 volatile 变量后，B 线程读同一个 volatile 变量。A 线程在写 volatile 变量之前所有可见的共享变量，在 B 线程读同一个 volatile 变量后，将立即变得对 B 线程可见。

volatile 写-读的内存语义

volatile 写的内存语义如下：

当写一个 volatile 变量时，JMM 会把该线程对应的本地内存中的共享变量刷新到主内存。

以上面示例程序 VolatileExample 为例，假设线程 A 首先执行 writer()方法，随后线程 B 执行 reader()方法，初始时两个线程的本地内存中的 flag 和 a 都是初始状态。下图是线程 A 执行 volatile 写后，共享变量的状态示意图：

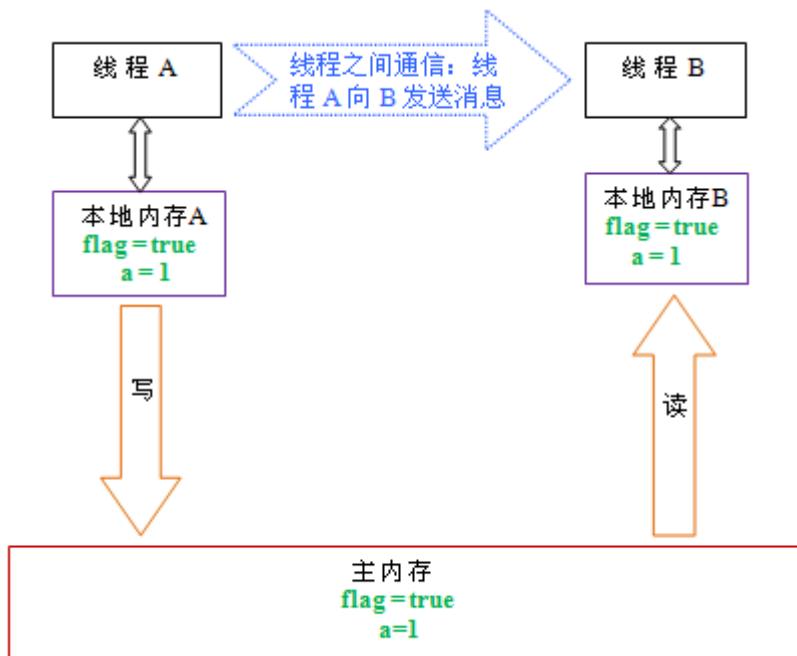


如上图所示，线程 A 在写 flag 变量后，本地内存 A 中被线程 A 更新过的两个共享变量的值被刷新到主内存中。此时，本地内存 A 和主内存中的共享变量的值是一致的。

volatile 读的内存语义如下：

当读一个 volatile 变量时，JMM 会把该线程对应的本地内存置为无效。线程接下来将从主内存中读取共享变量。

下面是线程 B 读同一个 volatile 变量后，共享变量的状态示意图：



如上图所示，在读 flag 变量后，本地内存 B 已经被置为无效。此时，线程 B 必须从主内存中读取共享变量。线程 B 的读取操作将导致本地内存 B 与主内存中的共享变量的值也变成一致的了。

如果我们把 volatile 写和 volatile 读这两个步骤综合起来看的话，在读线程 B 读一个 volatile 变量后 写线程 A 在写这个 volatile 变量之前所有可见的共享变量的值都将立即变得对读线程 B 可见。

下面对 volatile 写和 volatile 读的内存语义做个总结：

- 线程 A 写一个 volatile 变量，实质上是线程 A 向接下来将要读这个 volatile 变量的某个线程发出了（其对共享变量所在修改的）消息。
- 线程 B 读一个 volatile 变量，实质上是线程 B 接收了之前某个线程发出的（在写这个 volatile 变量之前对共享变量所做修改的）消息。
- 线程 A 写一个 volatile 变量，随后线程 B 读这个 volatile 变量，这个过程实质上是线程 A 通过主内存向线程 B 发送消息。

volatile 内存语义的实现

下面，让我们来看看 JMM 如何实现 volatile 写/读的内存语义。

前文我们提到过重排序分为编译器重排序和处理器重排序。为了实现 volatile 内存语义，JMM 会分别限制这两种类型的重排序类型。下面是 JMM 针对编译器制定的 volatile 重排序规则表：

是否能重排序	第二个操作		
第一个操作	普通读/写	volatile 读	volatile 写
普通读/写			NO
volatile 读	NO	NO	NO
volatile 写		NO	NO

举例来说，第三行最后一个单元格的意思是：在程序顺序中，当第一个操作为普通变量的读或写时，如果第二个操作为 volatile 写，则编译器不能重排序这两个操作。

从上表我们可以看出：

- 当第二个操作是 volatile 写时，不管第一个操作是什么，都不能重排序。这个规则确保 volatile 写之前的操作不会被编译器重排序到 volatile 写之后。
- 当第一个操作是 volatile 读时，不管第二个操作是什么，都不能重排序。这个规则确保 volatile 读之后的操作不会被编译器重排序到 volatile 读之前。
- 当第一个操作是 volatile 写，第二个操作是 volatile 读时，不能重排序。

为了实现 volatile 的内存语义，编译器在生成字节码时，会在指令序列中插入内存屏障来禁止特定类型的处理器重排序。对于编译器来说，发现一个最优布置来最小化插入屏障的总数几乎不可能，为此，JMM 采取保守策略。下面是基于保守策略的 JMM 内存屏障插入策略：

- 在每个 volatile 写操作的前面插入一个 StoreStore 屏障。
- 在每个 volatile 写操作的后面插入一个 StoreLoad 屏障。
- 在每个 volatile 读操作的后面插入一个 LoadLoad 屏障。
- 在每个 volatile 读操作的前面插入一个 LoadStore 屏障。

上述内存屏障插入策略非常保守，但它可以保证在任意处理器平台，任意的程序中都能得到正确的 volatile 内存语义。

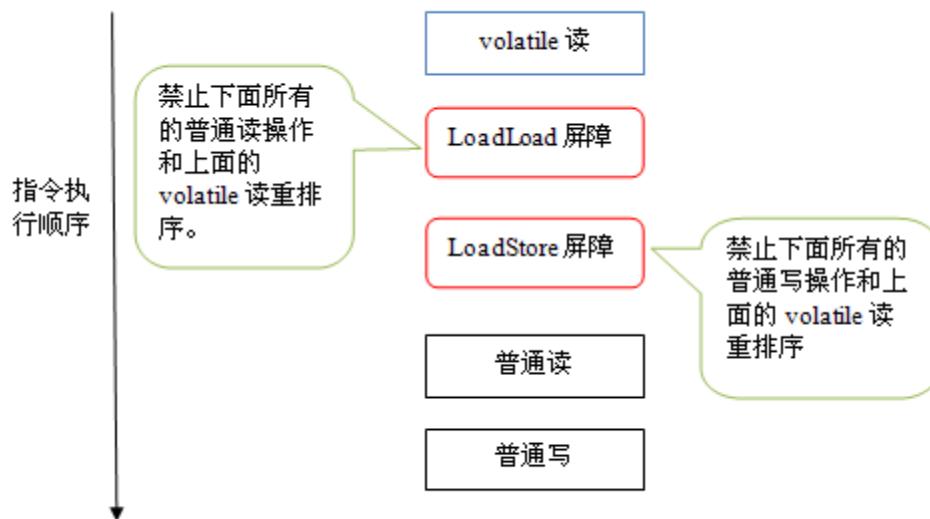
下面是保守策略下，volatile 写插入内存屏障后生成的指令序列示意图：



上图中的 StoreStore 屏障可以保证在 volatile 写之前，其前面的所有普通写操作已经对任意处理器可见了。这是因为 StoreStore 屏障将保障上面所有的普通写在 volatile 写之前刷新到主内存。

这里比较有意思的是 volatile 写后面的 StoreLoad 屏障。这个屏障的作用是避免 volatile 写与后面可能有的 volatile 读/写操作重排序。因为编译器常常无法准确判断在一个 volatile 写的后面，是否需要插入一个 StoreLoad 屏障(比如，一个 volatile 写之后方法立即 return)。为了保证能正确实现 volatile 的内存语义，JMM 在这里采取了保守策略：在每个 volatile 写的后面或在每个 volatile 读的前面插入一个 StoreLoad 屏障。从整体执行效率的角度考虑，JMM 选择了在每个 volatile 写的后面插入一个 StoreLoad 屏障。因为 volatile 写-读内存语义的常见使用模式是：一个写线程写 volatile 变量，多个读线程读同一个 volatile 变量。当读线程的数量大大超过写线程时，选择在 volatile 写之后插入 StoreLoad 屏障将带来可观的执行效率的提升。从这里我们可以看到 JMM 在实现上的一个特点：首先确保正确性，然后再去追求执行效率。

下面是在保守策略下，volatile 读插入内存屏障后生成的指令序列示意图：



上图中的 LoadLoad 屏障用来禁止处理器把上面的 volatile 读与下面的普通读重排序。LoadStore 屏障用来禁止处理器把上面的 volatile 读与下面的普通写重排序。

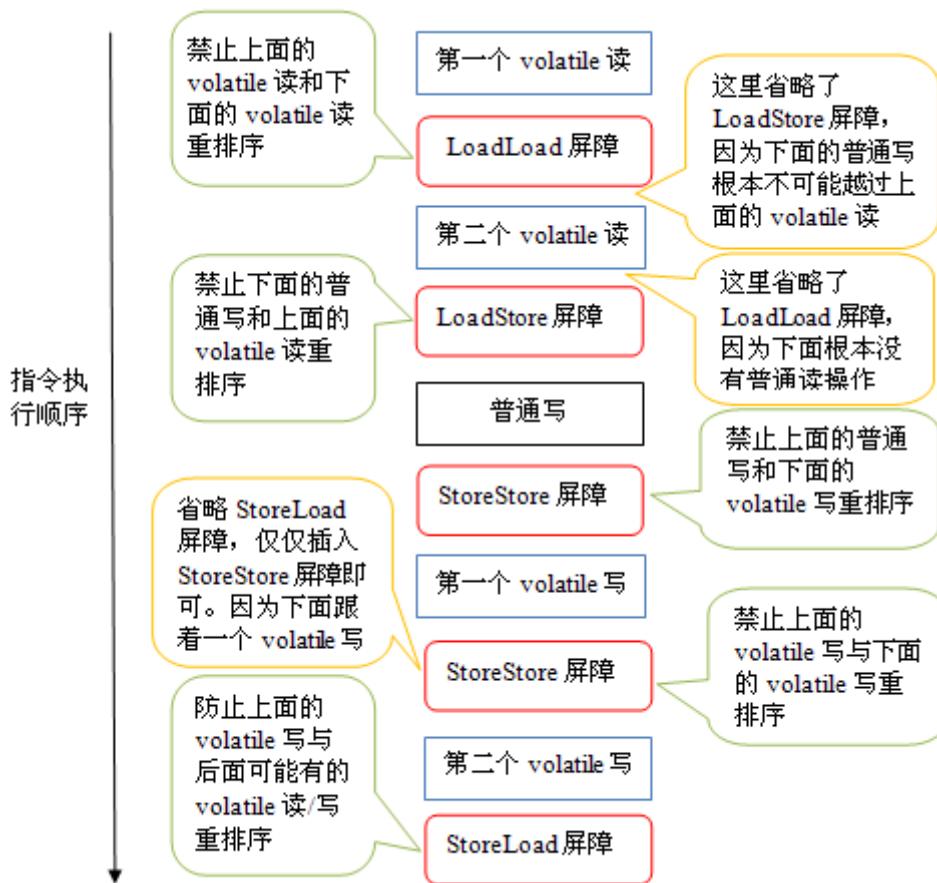
上述 volatile 写和 volatile 读的内存屏障插入策略非常保守。在实际执行时，只要不改变 volatile 写-读的内存语义，编译器可以根据具体情况省略不必要的屏障。下面我们通过具体的示例代码来说明：

```

class VolatileBarrierExample {
    int a;
    volatile int v1 = 1;
    volatile int v2 = 2;

    void readAndWrite() {
        int i = v1;                      // 第一个 volatile 读
        int j = v2;                      // 第二个 volatile 读
        a = i + j;                      // 普通写
        v1 = i + 1;                     // 第一个 volatile 写
        v2 = j * 2;                     // 第二个 volatile 写
    }
    ...
}
    
```

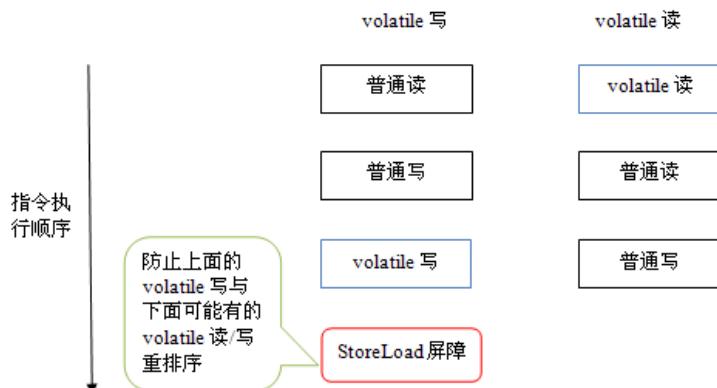
针对 readAndWrite()方法，编译器在生成字节码时可以做如下的优化：



注意，最后的 StoreLoad 屏障不能省略。因为第二个 volatile 写之后，方法立即 return。此时编译器可能无法准确断定后面是否会有 volatile 读或写，为了安全起见，编译器常常会在这里插入一个 StoreLoad 屏障。

上面的优化是针对任意处理器平台，由于不同的处理器有不同“松紧度”的处理器内存模型，内存屏障的插入还可以根据具体的处理器内存模型继续优化。以 x86 处理器为例，上图中除最后的 StoreLoad 屏障外，其它的屏障都会被省略。

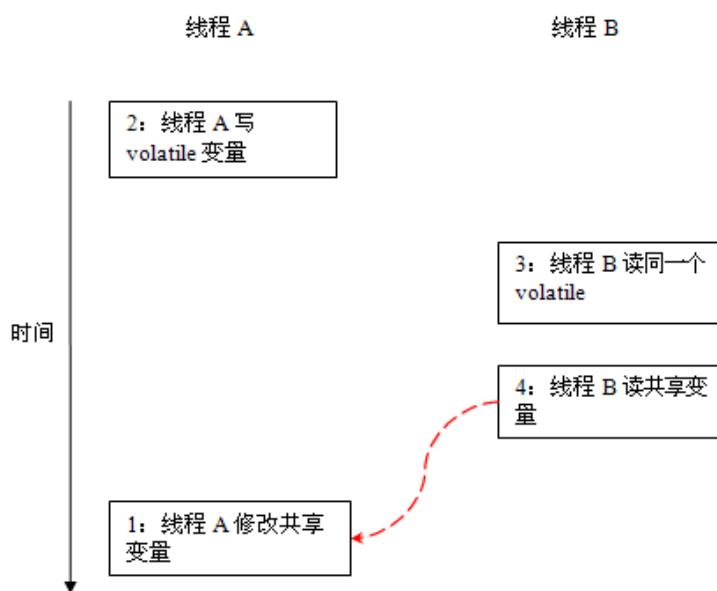
前面保守策略下的 volatile 读和写，在 x86 处理器平台可以优化成：



前文提到过，x86 处理器仅会对写-读操作做重排序。X86 不会对读-读，读-写和写-写操作做重排序，因此在 x86 处理器中会省略掉这三种操作类型对应的内存屏障。在 x86 中，JMM 仅需在 volatile 写后面插入一个 StoreLoad 屏障即可正确实现 volatile 写-读的内存语义。这意味着在 x86 处理器中，volatile 写的开销比 volatile 读的开销会大很多(因为执行 StoreLoad 屏障开销会比较大)。

JSR-133 为什么要增强 volatile 的内存语义

在 JSR-133 之前的旧 Java 内存模型中，虽然不允许 volatile 变量之间重排序，但旧的 Java 内存模型允许 volatile 变量与普通变量之间重排序。在旧的内存模型中，VolatileExample 示例程序可能被重排序成下列时序来执行：



在旧的内存模型中，当 1 和 2 之间没有数据依赖关系时，1 和 2 之间就可能被重排序 (3 和 4 类似)。其结果就是：读线程 B 执行 4 时，不一定能看到写线程 A 在执行 1 时对共享变量的修改。

因此在旧的内存模型中，volatile 的写-读没有监视器的释放-获所具有的内存语义。为了提供一种比监视器锁更轻量级的线程之间通信的机制，JSR-133 专家组决定增强 volatile 的内存语义：严格限制编译器和处理器对 volatile 变量与普通变量的重排序，确保 volatile 的写-读和监视器的释放-获取一样，具有相同的内存语义。从编译器重排序规则和处理器内存屏障插入策略来看，只要 volatile 变量与普通变量之间的重排序可能会破坏 volatile 的内存语意，这种重排序就会被编译器重排序规则和处理器内存屏障插入策略禁止。

由于 volatile 仅仅保证对单个 volatile 变量的读/写具有原子性，而监视器锁的互斥执行的特性可以确保对整个临界区代码的执行具有原子性。在功能上，监视器锁比 volatile 更强大；在可伸缩性和执行性能上，volatile 更有优势。如果读者想在程序中用 volatile 代替监视器锁，请一定谨慎。

参考文献

[Concurrent Programming in Java™: Design Principles and Patterns](#)

[JSR 133 \(Java Memory Model\) FAQ](#)

[JSR-133: Java Memory Model and Thread Specification](#)

[The JSR-133 Cookbook for Compiler Writers](#)

[Java 理论与实践：正确使用 Volatile 变量](#)

[Java theory and practice: Fixing the Java Memory Model, Part 2](#)

作者简介

程晓明，Java 软件工程师，国家认证的系统分析师、信息项目管理师。专注于并发编程，就职于富士通南大。个人邮箱：asst2003@163.com。

原文链接：<http://www.infoq.com/cn/articles/java-memory-model-4>

相关内容：

- [深入理解 Java 内存模型（五）——锁](#)
- [深入理解 Java 内存模型（三）——顺序一致性](#)
- [深入理解 Java 内存模型（二）——重排序](#)

推荐文章

Heroku 危机带来的启示

作者 丁雪丰

这些日子，说 Heroku 处在风口浪尖一点都不为过，先前 InfoQ 就 Heroku 大用户 Rap Genius 的不满做了报道，虽然 Heroku 在官方博客上做了说明，但 Rap Genius 和众多网友并不买账，问题仍在延续，不妨让我们对整件事件做个回顾，看看能够从中得到什么启示。

事件回顾

2月13日，Rap Genius 在其网站上发布了一篇文章，讲述了他们在 Heroku 上遇到的问题，并做了分析。问题的导火线是之前为了解决一些 JavaScript 的小问题，Rap Genius 在线上做了些 ab 基准测试，结果却发现有些页面的响应时间远比 Heroku 和 New Relic 所报告的时间长。以静态的版权页面为例，Heroku 提供的平均响应时间是 40ms，而 Rap Genius 测试所得的时间却是 6330ms。

Rap Genius 的同学在自己分析的同时，也和 Heroku 的工程师在进行沟通，最后确定问题的原因是处在 Heroku 的路由策略上，关键是 Heroku 的文档并未正确地描述其路由策略（在各个时期的不同文档中，对“智能路由”的描述都是只有在 Dyno 可用时才会将请求路由过去），但实际情况是路由采用了随机分配的策略，不考虑 Dyno 当时的情况，导致大量请求都在 Dyno 上排队，没有得到及时处理。更不幸的是 Heroku 的日志和监控并没有很好地反应实际情况。

这篇文章在 Hacker News 和 Reddit 上一经转发，也获得了不少关注，有网友就在评论中讨论了各种路由算法。在 Rap Genius 的文章中，他们用 R 做了个模拟，原本只需 75 个 Dyno 的一个应用 使用随机路由则需要 4000 个以上的 Dyno。

面对这一情况，Heroku 的危机公关并不成功，虽然 Oren Teich (Heroku 的 GM) 及时在 Hacker News 和官方博客上发表声明，承认了他们在 Ruby on Rails 应用上的性能问题，并做了一些承诺；第二天，官方博客上如约发表了具体的技术分析及后续改进的计划。但这些举措都没能让用户满意，暂且不论 3 年来为此多收的费用，至少 Heroku 并没有写出真实的情况：

模糊了本次事件的时间——Rap Genius 的共同创始人 Tom Lehman 在 [2月5日](#) 就已经和 Heroku 的 CTO Adam Wiggins 讨论这个问题了，而且在 2 月 8 日发给他了模拟结果，因此 Heroku 并不是在 2 月 13 日才知道这件事情的。更有人早在 2011 年 2 月 17 日就写过[博文](#)反应这个问题，还在 Google Group 里进行了讨论，Oren Teich 还亲自答复过，所以 Heroku 早就知道文档和系统实现不符合，存在性能问题。

模糊了本次事件的影响范围——Heroku 的文中只说了 Bamboo 上的 Rails 应用的性能受到影响，Tom Lehman 调侃到为什么 Cedar 的性能没有受到影响，是因为它用 Thin (Cedar 上默认的 Web 服务器) 跑 Rails 的性能就没好过。

问题剖析

在 Heroku 的[技术分析](#)里详细描述了造成问题的原因。

[Bamboo](#) 是早期的 Heroku 运行时技术栈，运行于 Debian 5.0 之上，提供了两套 Ruby VM——Ruby Enterprise Edition 2011.03 1.8.7-p334 和 MRI 1.9.2-p180，只支持 Ruby on Rails 2.x，使用 Thin 作为 Web 服务器。后来，随着技术的发展，Heroku 将默认的 Bamboo 换成了 [Cedar](#)，从仅支持 Rails 演化为了支持多平台多语言 (Ruby on Rails 的默认服务器同样为 Thin)。而问题也正是从这次大的系统架构升级开始的。

[Thin](#) 本质上是单线程的 Web 服务器，基于 EventMachine 来做多路复用，通过回调实现请求的并发处理。但要让 Rails 应用充分发挥 Thin 的优势，在编写应用时也要按照 EventMachine 的写法加以修改，对所用的库也有要求。所以，不管是 Bamboo 还是 Cedar，默认情况下还是单线程的，不能很好地支持并发请求处理。如果在 Cedar 中选择 [Unicorn](#)，那么情况则会有所不同。

因为使用 Thin 的 Dyno 无法并发处理请求，所以最初 Routing Mesh 的路由策略是选择只有当 Dyno 可用时才会将请求分配给它。但是由于 Cedar 支持多语言多平台，因此 Routing Mesh 的策略也做了调整，改为了更加通用的随机分配，暂时无法处理的请求在 Dyno 对应的队列中排队，不考虑 Dyno 的负载。对于 Java 应用，这个策略简单有效，但是对于单线程服务器中运行的未经优化的 Rails 应用，这就很成问题了。

更致命的是 Heroku 的文档并没有反映出这一变化，而监控也没能体现这一排队的耗时。原本 Heroku 打算让 Bamboo 的用户继续使用原来的 Bamboo 路由，

直到他们做好准备迁移 Cedar。可惜的是随着流量的增加，Heroku 往路由集群里加入越来越多的节点，慢慢的使用新策略的节点占比越来越高，于是 Bamboo 就渐渐地出现了性能的下降。至于 Cedar 上用 Thin 运行的 Rails 应用，自打其上线起 就因随机路由策略而性能不佳，因此 Tom Lehman 的调侃也不无道理。

经验教训

本次事件，作为云服务提供商的 Heroku 有不可推卸的责任，其 CTO 和 COO 在很久前就知情的情况下，没有有效修正该问题，对用户造成了巨大的影响。对于一家云服务提供商而言，在进行大的架构升级前应该进行充分的评估及后续的追踪；在发现问题后，应该及时修正，减少对用户的影响。除了关键的技术实力，还有很多其他细节需要考虑，比如相关的文档和客户支持，本次 Heroku 在这些方面也都做的有欠妥当——文档没有正确反映实际情况，虽然 CTO 和 COO 亲自应答这一问题，但没有实际的后续动作。

在文档方面，同为 PaaS 的 Cloudfoundry 就比较透明，一方面因其开源特性，使用者可以完全了解其运行机制，另一方面有大量的材料能帮助大家更好地使用它。同样以路由机制为例，EMC 中国研究院的颜开撰写的[《新版 CloudFoundry 揭秘》](#)中详细说明了其路由的实现：nginx 结合 Lua 脚本，由 Lua 分析请求，直接转发给对应的 Droplet，转发是随机的，但是请求中增加了 Cookie 信息，由此实现粘性会话，尽量将请求转发到同一个 Droplet 上。正是由于这些材料，Cloudfoundry 的使用者可以更好地正确使用 Cloudfoundry 的公有云。

在客服方面，好的客服能及时帮助客户解决问题，更能带给客户好的使用体验，而糟糕的客服，则有可能造成客户流失。[之前，国内的盛大云因客服的问题遭到投诉，最终用户放弃盛大云，转用其他产品。](#)可见，要做好一个云服务平台，光有技术是远远不够的。

虽说使用 PaaS 可以降低使用者的运维成本，无需再维护各种基础设施，同时提供适当的冗余，保证可用性。但是，这并不意味着使用者自己不需要下功夫，正确的架构必不可少，还需要一定的投入。

例如，本次 Rap Genius 在路由的问题上吃亏，一定程度上也是由于其 Web 请求的处理时间过长，才造成了队列中请求积压。对于耗时较长、占用资源的任务应该异步化，而非置于同步的 Web 请求中，交由后台专门的 Worker 处理更为合适（比如 Heroku 上的 Worker Dyno 就比 Web Dyno 更适合这种任务）。

eBay 在其分享中也多次提到了任务的异步化，可见异步化对于一个高负载高可用系统是何其重要。

要在故障之初就发现端倪，良好的监控是必不可少的，虽然各家云服务提供商在其产品中都有监控功能，但用户还是应该自己定制适当的监控。除了对服务端的监控，也不能忽视了用户端的监控，如果平时就能察觉用户访问的时间存在异常，那么早就可以发现问题。[百姓网在 Velocity 大会上就分享过他们的经验](#)，从用户的点击开始获取用户的真实体验。此外，还可以使用 [Google Analytics](#)，国内的[监控宝](#)也能对站点进行有效的监控。

随着网站的不断发展，集群也在不断壮大，此时应该将故障视为常态，即使在云端，故障也是不可避免的，AWS 在过去的一年里发生的重大故障就不下三次，[2011 年那次著名的 US-EAST-1 故障甚至还让“天网”的攻击没有发生](#)。在这方面 Netflix 就做的非常领先，他们的[“猴子军团”](#)在业内非常有名，通过日常自己制造故障来验证系统能否在真正的故障到来时自行恢复或者降级，降低对用户的影响。

对于云服务的使用者，如果要达到很高的可用度，必须在架构上就将容错和容灾纳入考虑之中，比如节点失效、网络中断、机房断电，甚至是 AWS 可用区整个故障。公有 PaaS 在这方面为用户提供了一些支持，但是在 AWS 可用区故障容灾方面应该还没有很好的支持，比如跨可用区的数据冗余，主从如何选择都是需要考虑的问题，需要用户自己来继续完善。

要做好一个云服务平台需要投入很多的精力与成本，而要用好这个平台，在其中上搭建站点为最终用户创造更多价值也并绝非易事，希望这次 Heroku 的事件能给大家带来一些启示，不知读者朋友您又有何经验可以分享？

原文链接：<http://www.infoq.com/cn/articles/heroku-inspiration>

相关内容：

[VMware 发布 Cloud Foundry 的免费版本](#)

[从简单到复杂：大型 Rails 与 VoIP 系统架构与部署实践](#)

[Node.js 获得企业开发者青睐](#)

特别专题

访“魔道”团队：Flash MMORPG 开发中的“五个基本原则”

作者 贾国清

本文是针对游戏开发领域系列采访中的第四篇，我们会采访到目前游戏开发比较热门的技术和公司来对问题进行解答，问题主要涉及游戏简介、开发中 5 件对的事情、5 件可以改进的事情、美工设计以及开发经验分享等。话题主要涵盖开发、发布、平台、开发工具和创新工具等展开。本文是针对 SGF Games，魔道六宗游戏制作人谈熠的采访。

采访主要从一款 Flash MMORPG 游戏出发，分别阐述了“魔道”开发团队在选择代码库、提升用户体验、进行测试覆盖和快速迭代上的思考，以及在团队协作、自动化测试和开源代码库选择上所遇到的挑战。以下是采访的具体内容：

InfoQ：听闻谈总在负责一个跨平台的 Flash MMORPG 网页版本叫“魔道六宗”，iOS/Android 版本叫“魔道”），能否对产品理念做一下简单的介绍？

“魔道”：我叫谈熠，来自 SGF Games，目前负责《魔道六宗》这个产品的研发工作。“魔道”是我们花了不少心思打磨的一款即时战斗的角色扮演类的网络游戏（MMO ARPG）产品。它同时也是一款泛终端的游戏产品，采用 Adobe 公司提供的 AIR 技术开发。其网页游戏由北京易橙天下公司运营，手机游戏在苹果应用商店中查找“魔道”即可下载安装。安卓版正在进行接入测试，不日将登录各大应用市场。

InfoQ：在游戏的设计和开发过程中，请列举 5 件认为做的对的事情（5 “Rights”），请配以具体事例，分别说明）？

“魔道”：开发一款游戏的过程无疑是一个充斥着大量细节和复杂矛盾的系统工程。而我们的团队情况比较特殊，没有显赫的从业背景，大家都是从玩家转型而来的开发者，凭着对游戏的热爱，用心投入地来做这个产品。

我始终记得在“魔道”立项之初，一位备受尊敬的业界前辈就曾告诫过我们：网游的本质是人与人的交互，打磨一款游戏产品需要沉淀大量的心法（know-how）。因此为了避免项目走得太远而迷失，我们在研发过程中定

义了一些规则。至于这些规则是对或错，还有待玩家和时间的检验。不过我们的团队已经从中收益良多。借这个机会，我很高兴能和大家一起分享一下我们所得的经验和教训。也希望能够通过 InfoQ 平台得到同行们的建议和指点，我的联系邮箱是 tanyi@smartgf.com。

规则一：一个代码库

开发魔道的客户端时，工具链的选择曾一度让我们伤透了脑筋。因为作为一个开发团队，我们的时间和精力十分有限。我们要确保自己有能力快速的根据玩家和运营的需求对产品做出调整。所以我们很害怕工具选择的不慎使我们陷入到无穷尽的开发折腾里去。

为了避免出现这样的情况，我们对当时市场上流行的各类开发工具和环境都做了一些调研，不过结果都不尽人意。

具体的来说吧，HTML5 环境下的渲染性能和 API 成熟度远远无法满足 MMO 产品的开发需求；UNITY 3D 技术在移动市场上相对比较成熟，并且提供整合的开发环境，但是在浏览器市场上的渗透率过低，此外所提供的联机环境下的开发能力十分有限。Cocos2d-X 在 iOS 平台上有很多成功案例，而且基于 C++ 的移植有广泛的代码库支持。可惜基于 JavaScript 的移植无法满足浏览器中的 MMO 开发需求，而且作为一个开源产品，在 Android 的移植上，对系统的依赖度很高，我们非常担心如果采用 Cocos2d-X 作为开发基础的话，最后将不得不陷入多个平台版本多个代码库的痛苦分裂。

总结下来，我想我们所需要的工具链至少需要具备三个基本特性。

首先必须是一个成熟的虚拟机环境，能够有效的将不同平台系统中的差异进行抽象，而不是把这些折腾成本转嫁给开发者。然后必须能够使我们充分利用到物理设备上的 CPU 和 GPU 资源，特别是在移动设备上，MMO 游戏中的大量图形渲染需求需要充分地利用 GPU 的并行计算能力。最后必须提供完整的底层接口，使我们能够针对不同的设备系统进行差异化的扩展，而不影响主代码库中的实现。

2011 年末，在 Adobe 的 Peter 和 7yue 的邀请下，我们有幸加入了当时正在研制阶段的 AIR SDK 的专家预览小组。熟悉了 AIR 技术之后，我们在选择工具平台上的纠结一下子都释然了。虽然 AIR 技术在跨平台开发工具市场中成熟得相对比较晚，但是无论在深度还是广度上，AIR 都给出了近乎完

美的解决方案。当我们的开发需要触及底层渲染时，通过 Adobe 的 AGAL 语言可以能够直接对抽象后的显卡设备编写汇编指令。而同时兼容 Stage3D (GPU 渲染) 和显示列表 (CPU 渲染) 的 AIR 技术，给我们的开发提供了极大灵活度，大幅缩短了 UI 移植和终端适配的时间成本。

规则二：体验至上

我们刚开始制作魔道的时候，正值国内网页游戏的市场的如火如荼的高峰。而今移动平台在一个极短的时间内走完了 PC 平台多年的积累，迅速崛起。现在市面上流行的网页游戏都同时运营网页和手机版本。

我们着手开发手机游戏版的最初想法是将网页游戏进行简单的移植。为了验证这个想法的可行性，我们花费了数百个小时玩了各类流行的手机端 MMO 游戏。然后发现一个令人失望的现实——手机和平板电脑的人机交互属性和 PC 有很大的差异，简单的移植只会让玩家体验大打折扣。

就拿客户端断线重连功能来举例。PC 设备上的大量用户处于稳定的网络连接情况，并且键盘和鼠标给玩家提供了高效率的输入途径。但是手机用户往往处于不稳定的网络环境下，而且游戏进程被切换到后台进而中断的情况十分常见。当然我们可以简单移植网页游戏的处理方法，每次断线之后让玩家重新输入用户名和密码进行登录。但在手机上进行文字和符号的输入，对很多玩家而言无疑是一个痛苦的障碍。所幸 FLASH/AIR 中 TCP 连接的 Socket 抽象对象提供了十分细腻的 API ,从而使得我们能够在手机游戏中实现自动的断线重新连接功能。



图一：兼容显示列表的 FLASH/AIR 技术大幅降低 UI 重构成本

再举一个例子，通过统计数据，我们发现大量手机版玩家临睡前上线。因此我们把手机版的界面尽量设计为单手方便操作。然而网页游戏中，玩家都是直视电脑，主要的输入操作通过键盘和鼠标完成。因此网页游戏和手机游戏采用了两套不同的 UI 实现。为了能够快速完成两套 UI 的实现，我们采用 Starling UI 框架。Starling 在 GPU 模式下还原了 FLASH 的显示列表结构。从而让 UI 的移植和修改都简单很多。

规则三：保持简单



图二：交互设计上尽量保持简单

移动设备在我们的理解中和 PC 设备有着本质性的区别。终端用户在移动设备上的交互体验集中在娱乐和消费上，而在 PC 上进行更多的生产型工作。因此设计手机版本的交互操作时，我们秉承保持简单的原则，尽可能的做减法。在魔道的手机版中，我们抛弃了强制性的步骤引导，而改用根据玩家在游戏中的发展而逐步开启玩法模块的设计。同时我们也将 PC 上的拖拽操作改为触屏操作。

规则四：没有测试覆盖绝不能发布

在整个开发过程中，随着项目复杂的增加，我们越来越发现测试覆盖的重要性。在魔道运营之初，由于测试覆盖的不完整，当数万个玩家涌入时，服务器发生一系列的异常。我们不得不进行的反复维护而使玩家的游戏体验大打折扣。

经历了这次挫折之后，我们决定整体重构游戏服务器。即时战斗的 MMO 游戏的服务器端程序是一个高并发处理程序结构。在这样的结构下，对系统资

源调度的不谨慎将必然使程序陷入竞争危害。所以服务器的重构采取开放和细分的模块化设计，然后针对每个模块进行测试覆盖。

但是即便对每个模块进行了完整的测试覆盖之后，系统整合时我们依旧遇到很多异常。这主要是因为游戏服务器本身是一个大量的异步请求的过程性操作。为了尽可能模拟终端玩家的环境，我们采用 Node.js 编写了一系列的用例测试来检查系统整合的功能。通过这两个测试环节的控制，魔道的服务器变得非常稳定，在生产运营中，除了版本迭代外，没有再发生额外的异常维护。

规则五：保持快速迭代

计划总是赶不上变化。在开发魔道的过程中，我们已经习惯了在测试和运营后返工修改预期的功能设计。为了能够让魔道在调整中不断的快速进化，我们将产品版本的迭代周期严格的控制在 3 周以内。我想我们能够尽量缩短迭代周期，主要还是归功于 AIR 技术所提供的成熟的工具链和我们质保小组实现的大量的用例测试。

InfoQ：请列举在开发过程中，您认为还值得改进或提升的 5 件事情（请配以具体事例，分别说明）？

魔道：在我看来，产品开发不是一个单纯的项目，而更象一个始终“在路上”的旅程。在体验了魔道旅程的甜酸苦辣之后，我们学到了许多宝贵的经验，同时也留下不少的遗憾和教训。单就开发过程而言，简单总结如下。

首先我觉得团队协作是一门很大的学问，特别是对于搞技术的同事而言，因为大多比较腼腆沉静。有的时候团队中的几个人都在考虑解决同一个问题，但是所采用的思路和方法很可能截然不同。在这种情况下，如果无法互相沟通好的话，就不能能力往一处使了。更何况在这种时候，沟通也是件很累人的事情。于是，慢慢的我们发现，在做开发实现的过程中，设定一套被大家都接受的代码规范往往能起到事半功倍的效果。这就好比一家人有了共同的方言，这样凑在一起唠嗑才会开心。我们在一个小组中做了这样的尝试，发现效果很好，正在不断的改进和推广这个方法。

然后我觉得人工测试远不如机器的自动化测试有效。魔道的测试最初是采用完全的人工进行。可能是因为测试的同事对魔道都已经太熟悉的缘故，测试行为中的差异化就消失了，所以查不出意料以外的问题。后来我们编写了大

量的测试脚本，用自动化流程来进行覆盖，使得产品质量大幅改善。但是由于魔道的客户端主要基于 ActionScript 这个编译语言而开发 编写异步测试的工作变得复杂很多。相比之下，我们在服务器端采用 Node.js 来撰写测试则变得无比的灵活和轻盈。在这点上，虽然 ActionScript 语言提供了单元测试框架（比如 ASUnit, FlexUnit），但是如果 Adobe 不在 AS 虚拟机层面提供更加灵活的支持的话，恐怕“测试驱动开发”对于 ActionScript 开发者来说还将是一种奢望，当然更不用提“行为驱动的开发”了。

第三，和第三方开源代码库的对接。网上有很多 ActionScript 的开源项目，象珍珠一样散落在互联网的各个角落。找到和学会使用一个靠谱的第三方 AS 代码库是一件费时费神的事情。ActionScript 严重缺乏一个统一的中央的第三方模块管理和发布系统（参考 Node.js 的 npm、RubyGems、Python eggs 等）。正因为这样的系统的缺失，全世界 AS 开发者之间的工作难以有效的分享和复用。不过，这未必是一个技术性问题，在我看来，可能和 Adobe 之前一贯的基于工具产品销售的商业模式不无关系。Flex Builder 4.8 版本开始将由 Apache 基金会托管，同时 Adobe Game 开发工具的商业模式也将从基于工具产品收费，而转变为基于许可证书收费。相信这样的转变将会给 ActionScript 的生态环境带来许多新的气氛。

InfoQ：未来在游戏开发和游戏平台上还有哪些规划？

“魔道：说到规划，我觉得互联网为大家开启了一个非常神奇的时代。

以前，我曾在国外的一个大型公司工作过几年。当时的软件行业中，项目管理和瀑布模型的软件工程还很吃香。一个软件开发项目做上一年半载十分正常。再加上各种必然发生的需求变更，“产品交付”总让人感觉是一个距离遥远的目标。为了能够顺利达到这个目标，我的团队每周都要检查和调整月规划，季度规划，年度规划……在那样的环境下，开发本身很容易被形式化，而使开发者很快忘记了软件的本质是为了满足用户不断改变的使用需求。

现在，我们把产品的交付周期缩短到三周之内。每天，我们的策划都通过和游戏玩家的交流来调整和修订产品的功能设计。所以在我看来“未来的规划”已经不再是一个常量，而是变数了。不过，即便如此，我想有一些东西是不会变的。比如，无论玩家是在网页中进行游戏，还是在手机上玩，终端用户的总量正在急速膨胀。换句话说，玩家在魔道游戏中的总时间输入是一个持

续加速累积的资源。而我们所想要做的就是服务好玩家在游戏中所投入的时间。尽我们所能完善游戏的玩法，让玩家在游戏中享受到更好的体验。

原文链接：<http://www.infoq.com/cn/articles/muodao-dev-thoughts>

相关内容：

- [使用 Flash Builder 中的分析工具改善 Flash Professional 项目的性能](#)
- [使用 Flash Builder 4.5 进行多平台游戏开发](#)
- [Flex 4.6 SDK 和 Flash Builder 4.6 中的新功能](#)

推荐文章

Hadoop 的现在和未来

作者 [Boris Lublinsky](#) 译者 [夏雪](#)

现今，大数据和 Hadoop 在计算机工业里正如暴风骤雨般开展着。从 CEO、CIO 到开发人员，每个人对其用法都有自己的看法。据 Wikipedia 所述：

“Apache Hadoop 是一个开源的软件框架，它支持数据密集型的分布式应用，许可授权隶属于 Apache v2 license.[\[1\]](#) 它使应用程序以拍字节 (petabytes) 级数据进行工作，并可以在成千上万台独立的计算机上运行。Hadoop 源自于 Google 的 MapReduce 和 Google File System (GFS) 两篇论文。现在通常认为完整的 Apache Hadoop ‘平台’ 由 Hadoop 内核、MapReduce 和 HDFS 组成，以及若干相关的项目——包括 Apache Hive、Apache Hbase 等等”

可惜这个定义并没有真正解释 Hadoop 及其在企业中的角色。

在本次虚拟座谈会中，InfoQ 采访了多位 Hadoop 提供商和用户，他们就 Hadoop 的现在和将来发表了看法，并讨论了 Hadoop 继续走向成功并进一步推广的关键。

参加者：

- Omer Trajman , Cloudera 技术解决方案副总裁
- Jim Walker , Hortonworks 产品总监
- Ted Dunning , MapR 首席应用架构师
- Michael Segel , 芝加哥 Hadoop 用户群创始人

问题：

- 1、你如何定义 Hadoop ? 作为架构师，我们对服务器、数据库等术语有更专业的思考。在你的心里 Hadoop 属于哪个层面？
- 2、尽管人们实际谈论的是 Apache Hadoop ,但他们却很少直接从 Apache 网站上下载。如今大多数人都使用 Cloudera、Hortonworks、MapR、Amazon 等等的 “发行版” 进行安装。你认为这一现象的原因是什么，

这些“发行版”有哪些不同（请供应商们客观一点，我们知道你的是最好的）。

- 3、你认为如今 Hadoop 最普遍的用法是什么？将来呢？
- 4、除了 Flume、Scribe 和 Scoop 之外，Hadoop 与其他企业计算的集成非常少。你认为在企业 IT 基础架构中，Hadoop 会开始扮演更大的角色吗？
- 5、除了著名的 [Percolator](#) 之外，现在大部分 Google 项目都是用 Hadoop 实现的。你认为这样的项目(应该)在 Apache 的跟踪范围内吗？你了解实时 Hadoop 的其他方向吗？
- 6、许多人设法提高使用 Hadoop 的技能。还有很多人去找熟悉 Hadoop 的人。但是仍然搞不清楚，如何掌握 Hadoop 技术？阅读[这本书](#)？参加培训？还是考认证？

问题 1：你如何定义 Hadoop？作为架构师，我们对服务器、数据库等术语有更专业的思考。在你的心里 Hadoop 属于哪个层面？

Omer Trajman： Hadoop 是一个新型数据管理系统，它通过计算网络的处理能力将传统的非结构化领域或非关系型数据库联合起来。虽然它从传统大规模并行处理（MPP）数据库设计模式借鉴了大量经验，但 Hadoop 有几个关键的不同。首先，它是为低成本字节的经济而设计的。Hadoop 几乎可以在任意硬件上运行，可以非常宽容地应对异构配置和不时发生的故障。第二，Hadoop 非常容易扩展。Hadoop 第一个版本就可以扩展到数千个节点，当前版本试验表明可以持续增加到上万个节点以上。使用主流的两插槽 8 核处理器，那就是 80,000 核的计算能力。第三，Hadoop 可以非常灵活地存储和处理数据的类型。Hadoop 可以接受任何格式、任何类型的数据，并具有一组功能丰富的 API，用来读取和写入任何格式的数据。

Jim Walker： Hadoop 是一个高度可扩展的开源数据管理软件，使您轻松地获取、处理、交换任何数据。Hadoop 几乎可以连接到传统企业数据栈的每一层，因此将占据数据中心的中心位置。它将在系统和用户之间交换数据并提供数据服务。在技术层面，Hadoop 也就是做这些事情，但因为它为大众带来了超级计算能力，它也造成了商业的转变。它是开源软件，所创建的社区带来大规模并行处理，以及水平扩展所有在商品硬件上的存储。它不能替代系统，它迫使现有工具更加专业化，并占有流行的数据架构工具箱的一席之地。

Ted Dunning : 非常精确地定义 Hadoop , 至少让每个人都同意你的意见恐怕是不可能的。即使如此 ,假设你考虑这两个定义 ,你可以得到非常接近的答案 :

- a. 同名的 Apache 项目 , 该项目已经发布了一个 map-reduce 的实现和一个分布式文件系统。
- b. 一组项目集合 , Apache 和一些其他项目 , 它们使用或者以某种方式关联到 Apache Hadoop 项目。

第一个定义也经常用来暗指由 Apache Hadoop 项目发布的软件 , 但是 , 一个软件到底要与发布版本多接近才能 (或应该) 称之为 Hadoop 或 Hadoop 衍生品 , 是一个颇有争议的话题。

对我来说 , 作为社区中 Hadoop 相关软件的主要使用者或开发者 , 我更喜欢一个不太常用的 Hadoop 的定义。对于术语 “Hadoop” , 我更加喜欢先用它来代表社区 , 其次是主要的代码或项目。对我来说 , 社区比任何单个项目代码更重要。

Michael Segel : 我把 Hadoop 看作是一个框架以及进行分布式或并行处理的一组工具。你在 HDFS 中的分布式存储 , 在 Job Tracker 和 Task Trackers 中的分布式计算模型 , 以及在 HBase 中的分布式持久对象存储。按照 Hadoop 的定位 , 我认为这取决于特定的解决方案。

很难将 Hadoop 归为单独的类别。在一些场景中 Hadoop 用来做中间处理 , 这是难以在传统的 RDBMS 中完成的事情 , 于是使用 Hadoop 作为中间步骤 , 最后使用他们已有的 BI 工具执行分析。还有人用 Hadoop 来提供 “实时” (主观认为的) 数据处理。集成 Lucene / SOLR 和已有的 Hadoop / HBase 作为实时搜索引擎的一部分。关键是 Hadoop 被用于解决不同组织不同类型的问题 ; 即使在同一个企业中。这可能 Hadoop 最大的一个优势 , 它是一个基础框架 , 可以用来解决不同类型的问题。更多的人在使用 Hadoop , 并将其推进到极限 ; 这将产生大量各种各样的解决方案。

问题 2 : 尽管人们实际谈论的是 Apache Hadoop , 但他们却很少直接从 Apache 网站上下载。如今大多数人都使用 Cloudera、 Hortonworks、MapR、Amazon 等等的 “发行版” 进行安装。你认为这一现象的原因是什么 , 这些 “发行版” 有哪些不同 (请供应商们客观一点 , 我们知道你的是最好的) 。

Omer Trajman : “发行版”的需求有两个主要来源。至于 Cloudera，四年前我们开始与客户交流时首次遇到一个要求。每一位客户运行在不同的基础版本上，并且打了不同的小补丁，还使用了不同版本的客户端库。一旦我们涉及多个客户，就无法提供支持和解决代码问题的架构原则。产生“发行版”的第二个理由是，合作伙伴需要能够测试客户正在运行的代码库，以发现它们的不利影响。如果一个合作伙伴在 CDH4 认证，一个客户在 CDH4 上运行，他们要确信他们的软件在工作中不会产生冲突。为了给每一个人(不仅仅是我们的客户)创建一个标准基线，Cloudera 在 2009 年首次创建了 CDH。如果客户运行在 CDH 4.2 上，每个人都确切地了解代码库中有什么，并且可以测试它的不利影响。因为它完全开源并经 Apache 许可，任何人都可以按需改变安装程序，并且，他们可以获得任何人的支持。

Jim Walker : 我们认为基于 Hadoop 平台的下载趋势是一个较新的现象。因为越来越多的组织意识到，他们将受益于 Hadoop，当他们首次开始使用这项技术时，他们在寻找易于使用和消费的体验。后来，他们着手感兴趣的工具，工具易于管理和监控 Hadoop 在生产环境中的运行。这些“发行版”有很多相同点，相比开源的 Apache Hadoop，它们更易于使用和运维。这些“发行版”之间最主要的区别是，大多数(而非所有)包括了专属的软件组件。这么做的确把用户锁定在一个特定的“发行版”上，不允许他们充分利用开放源码社区过程。但是，Hadoop 及其相关项目都有自己的发布周期和版本结构，每个“发行版”都为消费者提供了重要价值，“发行版”已经把 Hadoop 和所有已知相关的项目打包到一起，负责部署的人就不需要去单独获取各个 Hadoop 相关的项目、单独测试并自行维护这样一个复杂的解决方案的网络了。

Ted Dunning : 事实上，人们去下载已打包的发行版无非是出于节省时间和节省脑力的考虑——否则，他们需要一个一个去下载必要的组件，一个一个地安装，并想办法让它们保持兼容。通常人们关注个别项目的功能或 Hadoop 社区项目，然后从 Apache 上直接下载项目源码，然而，还有一种做法是使用标准化的“发行版”。这使他们可以集中精力去处理那些他们认为最重要的事。

实际上，搭建和测试完整的 Hadoop “发行版”是一个最重要的任务。使用标准的“发行版”大大简化这个过程。另外，所有重要的“发行版”各自还有一些专属的 Hadoop 附件，使某些事可以更好地工作。 Hortonworks 有基于 VMWare 的软件，他们试图使 namenode 更加强壮，Cloudera 有专属管理接

口，MapR 有专属文件系统、表存储和管理功能。用户可以很轻易地找到这些附件的价值。

Michael Segel：首先，我想要指出的是，还是有很多人从 Apache 网站直接下载 Hadoop 的。那些人可能是想要研究最新版本的可用性，可能是 Apache 新的访问者。讨论组上的人仍在询问兼容性的问题，像“我有 X 版本的 HBase，我需要哪个版本的 Hadoop……”他们使用某个供应商的“发行版”，就不会有这个问题了。

企业倾向于选择供应商，因为他们需要供应商的支持。他们希望降低在 Hadoop 集群上的投入，以专注于解决手边的问题。供应商提供免费版本和支持版本，还有工具去简化他们产品的下载。(他们也有自己的网站，并有很多有用的帮助信息)。

当讨论供应商时，我必须提前声明，所有供应商实际上都是 Apache Hadoop 的衍生品。虽然 Cloudera 和 Hortonworks 承诺 100% 是 Apache 的，但我仍然认为它们是衍生品。供应商必须决定应用哪些补丁，亦或回退他们的版本。虽然所有代码都 100% 是 Apache 的，但这还是造成了一些轻微的差异。另外，供应商的产品中还包括一些附加工具和专属软件。我并没有说衍生品不好，相反它是好东西。衍生品也有利于区分供应商之间的差异。

我恰巧在供应商间保持着中立。你提到所有的供应商都支持 Apache Hadoop API，所以你编写的 map/reduce 程序，最多只做一次编译就可以在每一个版本中运行。

正如我之前所说，Hortonworks 和 Cloudera 承诺 100% 是 Apache 的，但采用了是 Hadoop 提供的内核。Cloudera 长期占有大部分的市场份额。 Hortonworks 借他们新出的 1.0 发行版发起强劲的营销攻势。我认为随着时间的推移，将难以在这两个之间做出选择。归根结底，它将成为支持和辅助工具。Hadoop 还在不断地发展。像诸如 YARN、HCatalog、Impala 和 Drill，在此我只列出了一些最近的创新。供应商不能自满，必须确定他们将来打算进一步支持的功能。

MapR 稍有不同。自全面启动以来，MapR 已决定替换内核，把 HDFS 替换为自己的 MapRFS，同时 MapRFS 实现了所有的 HDFS API，它是一个兼容 POSIX 的文件系统，可以通过 NFS 安装。此外，他们解决了在 Apache 的内核产品中发现的 SPOF Name Node 问题。他们还在一年前的初始版本中提供 HA(高可

用性)。虽然 MapR 支持 Apache Hadoop API , 但他们的软件是自有的、闭源的。MapR 有三种版本 , M3 (免费版) 、 M5 (含有支持的版本 , 并启用了所有 HA 特性) 和 M7 (在最近和自己重写的 HBase 一起发布) , MapR 采取了一种与其他供应商不同的方法 , 它肯定会有属于自己的追随者。

亚马逊拥有他们自己的产品 , 在他们提供的 EMR 中包括了对 MapR 的 M3 和 M5 的支持。因为他们不卖产品 , 所以我不会把他们归类到供应商。而且我认为你还要把 Google 加进来 , 因为他们在最近发布了他们的竞争产品。在此我们看到又一个 MapR 的合作伙伴进入到 Hadoop 市场。

在所有考查的选项和各种各样的供应商之中 , 我不得不说消费者将是最终的胜利者。早在 90 年代的时候 , 因为 Informix 、 Oracle 和 Sybase 间的激烈竞争 , 我们目睹了 RDBMS 的演变。今天 ? 我认为市场还相当不够成熟 , 我们在进行着一段疯狂之旅。

问题 3 : 你认为如今 Hadoop 最普遍的用法是什么 ? 将来呢 ?

Omer Trajman : 如今 , 人们用 Hadoop 应对各种行业间各自不同的挑战。最为普遍的用法是用来加速 ETL 。例如金融服务 , 不再是针对每个事务从众多源系统中拉数据 , 而是由源系统将数据推至 HDFS , ETL 引擎处理数据 , 然后保存结果。ETL 流程可被写入 Pig 或 Hive 中 或者使用商业解决方案如 Informatica 、 Pentaho 、 Pervasive 等等。结果可以将来用 Hadoop 分析 , 也可以提交到传统报表和分析工具来分析。经证实 , 使用 Hadoop 存储和处理结构化数据可以减少 10 倍的成本 , 并可以提升 4 倍处理速度。比传统 ETL 更突出的是 , Hadoop 还可以用来收集内部系统 (比如应用和 web 日志) 以及远程系统(在网络和全球上)的遥测数据。把精细的感应数据提供给公司的能力模型 (如通讯和移动载流能力模型) , 预测网络和设备上可能发生的问题 , 并主动地采取措施。 Hadoop 还可以作为集中式数据集线器 , 执行从跨组织的数据集分析到预测分析平台的任何工作。这些应用如今广泛地部署在生产环境中 , 为收集所有组织数据提供了可能性 , 很好地驱动了业务的发展。

Jim Walker : 大部分组织刚刚开始他们的 Hadoop 旅程。他们用它来提炼大量的数据 , 为业务分析实践提供价值。有些人用它来采集和使用一些曾经废弃的数据 , 或者仅仅从已有系统中去采集比以前更多的数据。更先进的组织开始走向数据科学研究之路 , 从事大数据和传统输入源的探索。在 2013 年 , Hadoop 将成为主流 , 人们将慎重地考虑将其作为传统企业数据架构的一部分 , 同 ETL 、

RDBMS、EDW 和目前所有那些为组织提供数据的现有工具一样成为“一等公民”。

Ted Dunning：我不了解当前的主要用途，但我看到了两个无限增长的领域，以至于我打算减少大部分大数据其他领域的使用。它们是：

- a. 世界度量系统。这些系统包括各种产品，它们拥有巨大的度量能力，并极有可能产生大量的数据。例如，涡轮机供应商使用仪器检测喷气式发动机，所以每架飞机将成为巨大的数据来源，又如磁盘驱动器供应商要在单独的磁盘驱动器中建设家用电话系统。同样地，零售商通过文字去察看客户对店内商品的反应。所有这些应用程序有可能比现有的大部分大数据系统产生更多的数据。
- b. 基因组织系统。单人的人类基因组测序就可以产生约四分之一 TB 的数据。癌细胞的增长包括细胞群落，经常有成千上万的突变遍布数以百计(至少)的变体发育谱系。这些不同的细胞系表示它们的基因不同，这些也是可以度量的。这意味着在不久的将来，一个人的医疗记录很有可能增长到几个 TB 的规模。再乘以每年进行医疗护理的人数呢，这表明当前的电子医疗系统的规模可能小了四到六个数量级。

还有一些我们暂时还不知道系统，它们可能会产生更多的数据。

Michael Segel：这是一个很难回答的问题。我认为我们会看到更多的公司从 Hive 的实施开始，因为它是“挂得最低的水果”。许多公司的员工都了解 SQL，对他们来说，Hive 的学习曲线最短。所以公司可以用很短的时间实现价值，能够很容易地部署 Hive 去解决问题。随着更多的公司内部 Hadoop 能力的提升，我想他们将会利用其他的 Hadoop 组件。

因为 Hadoop 是一个相当通用的框架，它可以用于许多不同行业中各种各样的解决方案。例如，通过采集和处理传感器数据以确定在你商店的货架上摆放哪些品牌的洗衣粉。你也可能有不同的用法。它是用来作为解决问题的中间步骤，还是用来提供实时数据？因为可以通过一些额外的工具来扩展 Hadoop 我认为，我们可以期待看到这些工具被公司更多的应用，这些公司的需求很难使用已有工具集来实现。在 Hadoop HDFS 框架之上，有 HBase(包含在 Hadoop 发行版)和 Accumulo (由美国 DoD 社区创建，是‘Big Table’的另一个衍生品)。而在 HBase 之上，你有 OpenTSDB 和 Wibidata 去扩展 HBase 的功能。随着这些工具的成熟和增强，我认为它们将会得到更多的应用。

我认为，随着更多的公司采用 Hadoop 并开始理解它的潜力，我们将继续看到 Hadoop 用于不同的方面，解决更为复杂的问题。

问题 4：除了 Flume、Scribe 和 Scoop 之外，Hadoop 与其他企业计算的集成非常少。你认为在企业 IT 基础架构中，Hadoop 会开始扮演更大的角色吗？

Omer Trajman： Hadoop 正在迅速地成为 IT 模型的数据中心。由于 Flume 中有用于任何事件数据丰富的连接，Sqoop 可用于所有结构化数据，HttpFS 可用于 SOA 集成和 ODBC、JDBC 报表工具，所以任何现有数据管理工作流程在产生数据或请求数据时都可以无缝地、安全地与 Hadoop 接口通讯。这种扩展使 Hive 进一步发展为 Hadoop 元仓库的标准。最初的目的就是在非结构化数据上映射 SQL 模式，Hive 已经增强了安全功能（针对定义验证和数据访问控制），而且集成了 Flume、Sqoop 以及其他系统接口。如今所有 Hadoop 部署都用这些数据集成的功能与 Hadoop 交换数据。未来，他们将能够与 Hive 安全地交换元数据。

Jim Walker： 你忘了，在 Hadoop 堆栈还有一个重要的组件。该组件被称为 Apache HCatalog，它为 Hadoop 提供了一个集中式元数据服务，这样就能够更容易地与传统的系统传递和交换数据了。HCatalog 分解了结构化数据与 Hadoop 之间的阻抗不匹配（impedance mismatch），这样它们就可以使二者深度集成。Teradata 的大量分析设备和微软的 HDInsights 产品都是很好的例子。在企业数据架构中，Hadoop 最终在现有系统旁扮演了属于它自己的角色。

Ted Dunning： 我自己的 MapR 公司，有一个业务是销售定制的 Hadoop “发行版”，它比其他“发行版”更容易集成企业基础设施系统。客户发现它们构建计算系统的能力真的很强，它们可以交叉扩展并与现有应用程序一起工作。在大部分大数据应用中，只有部分情况是真正的大规模计算。一个大数据系统的许多组件实际上并非那么大。因此，对于大数据系统的小部件，它可以通过使用小数据技术来获得回报，并节省出时间去开发系统中的大数据部件。

Michael Segel： 我认为这有点误导。Flume、Scribe 和 Scoop 都是开源的（Apache）项目，旨在整合 Hadoop 与其他公司的基础设施。IBM（数据阶段）和 Informatica 在他们的产品中添加了 Hadoop 集成。此外，Quest Software 也同样创建了一些解决方案以帮助与 Hadoop 的集成。所以业界也出现了在企业产品中适应并采用 Hadoop 的尝试。事实上如果我们与所有主要的硬件和软件

公司进行交流，我们会发现他们都有一些 Hadoop 解决方案，作为他们产品的一部分。

总体而言，我们仍然处于 Hadoop 增长曲线的早期。随着更多的公司采用 Hadoop 作为基础设施的一部分，我们将看到来自传统供应商更多的工具。许多现有的 BI 工具供应商利用 Hive Thrift 服务器连接他们的应用。在不足的方面，数据可视化算是其中的一个领域，我认为我们可以期待看到更多的工具进入市场。许多大型企业寄希望于 BI 报表和 dashboard 工具。与其购买一个完全独立的工具，也许不如为了支持 Hadoop 去扩展现有的基础设施更加让人关注。

问题 5 除了著名的 Percolator 之外 现在大部分 Google 项目都是用 Hadoop 实现的。你认为这样的项目(应该)在 Apache 的跟踪范围内吗？你了解实时 Hadoop 的其他方向吗？

Omer Trajman : 谷歌有许多项目，它们都是特定于谷歌的需求而创建的。除 Percolator 之外，还有许多项目、一些已公开的以及其他仍受严密保护的秘密。并非所有项目都是有用的或者可适用于其他的组织。从历史观点上说，社区和赞助组织指望从谷歌和其他大数据公司寻找灵感，以解决如何处理非常巨大的数据管理问题。BigTable 启发了 HBase、Chubby 启发了 Zookeeper，F1 启发了 Impala 的一些命名。如今，Hadoop 已经提供了若干解决方案，用于实时数据抽取(Flume)、实时数据存储(HBase)和实时数据查询(Impala)。未来的发展很可能被 Hadoop 用户和开发社区直接驱动。

Jim Walker : 我们认为，这里有 Hadoop 的应用案例，如果需要较快的数据访问，那么就适合用 Hadoop 来收集和分析。当然有很多其他不需要快速交互式访问数据的应用案例。我们相信充分利用社区驱动过程，整个 Hadoop 市场会最好的服务于广大全开放式项目。这个过程已经被证实，它可以满足企业的预期，为企业创造稳定和可靠的软件。Apache 软件基金会保证了这一点。

Ted Dunning : 其实我觉得 Hadoop 社区只是实现了很小一部分谷歌的项目。在应用程序级别，暂时没有 F1、Spanner、Dremel 和 Sawzall 等替代品，Borg 在开源社区中也没有好的替代品，谷歌内部使用基于开关的 Open Flow、操作系统虚拟层或各种各样庞大的基础设施服务。还有些类似的项目，他们一般都是一些逊色的仿制品。例如，我以为 Mahout 无法抗衡 Google 的机器学习系统，Lucene 要想比得上谷歌的搜索系统也有很长的路要走，甚至 Hadoop 比不上谷歌内部同一阵营的 map-reduce 系统。

开源社区终于有了适合于谷歌源代码管理系统的 Github 形式，但是，如果我们想为全世界提供一些功能，那么我们在开源社区中还要做更多大量的工作。

Michael Segel：我想，如果我们要看近期在扩展和加强 Hadoop 上的某些讨论，我认为你可以参考一下 Percolator。在 Quora 上，我们可以看到一些活跃的专题讨论。

当我们在 HBase 中有协处理器时，这里有一个相对较新的项目——Drill，其中 Ted Dunning 也积极参与在内。(我会尊重 Ted 针对 Drill 的讨论)。最近 Cloudera 也公布了 Impala。(我会尊重 Cloudera 针对 Impala 的讨论。)

有一些公司如 HStreaming，也参与了入站数据的实时处理。HStreaming 是一个本地公司，作为我们的芝加哥 Hadoop 用户组(CHUG)代表出席了会议。(他们已经搬到了加利福尼亚。)

实时处理方面，我认为协处理器的使用，以及 Lucene 和 Solr 更紧密的集成，我们将看到更多 HBase/Hadoop 的应用案例。

问题 6：许多人设法提高 Hadoop 的应用技能。还有很多人去找熟悉 Hadoop 的人。但是仍然搞不清楚，如何掌握 Hadoop 技术？阅读这本书？参加培训？考认证？

Omer Trajman：获取 Hadoop 技能最好的方式，是参与一个 Hadoop 项目。刚开始做 Hadoop 项目时最好的方式是参加培训班，参加认证考试并保持在项目中有一两本可以随时取用的好书。找一些专门研究 Hadoop 和更大生态系统的公司，跟他们讨论也非常有用。虽然大多数项目以 Hadoop 为核心开始启动，但很快就扩展到数据抽取处理、数据服务、分析和工作流。我们推荐与一个组织合作，它们可以提供一个从培训到服务完整的 Hadoop 平台，跨所有堆栈和整个 Hadoop 的生命周期支持和管理软件。虽然 Hadoop 是一个具有挑战性的技术，但它如今已经为组织解决了一些最大的问题，根据我们的经验，这很值得投资。

Jim Walker：培训是快速学习 Hadoop 基础的最好方式。一旦掌握了这些，就可以做些 Hadoop 实践并开始学习更多的概念了。

Ted Dunning：我们最大的一些客户开始喜欢在令人意想不到的地方寻找人才。而不是在通常几所顶级的大学中争夺那几个水平相仿的聪明人，他们可以在二流学校中找到绝对聪明的学生，尤其(大致而言)是在非计算机科学领域的技术人

员。然后对他们培训大数据方面的技术，在我曾经见过的人中，这些人都做得很好。

我还遇到相当多的人去参加 HUG (Hadoop 用户组) 或其他兴趣小组见面会，他们努力的从事自我训练。这些人利用所有可能的方式来学习技能，包括在线课程(Andrew Ng 机器学习方面的课程很受欢迎)、参加会议、创建个人项目、努力在工作中寻找大数据分析需求。

我感觉，这两个趋势给我们的启示是：很难把更多的聪明人放在公司里，而且在大数据方面，幕后更为广泛的称职人员的工作效果可能比灵光一闪式的想法更好。这个世界仍在快速发展，全能型人才的技能在开拓性事业中非常具有价值。但是，专家最终可能会占主导地位，但也要再等几年之后。与此期间，会涌现出各种优秀的人才。

Michael Segel：我认为你漏掉了最重要的……动手实践的经验。虽然读书、在线指南和示例很重要。培训有助于加速我们学习的过程，但是在 Hadoop 方面，动手实践的经验是不可替代的。

有几个成本较低的方式可以获得这种经验。首先，大部分供应商的产品都有一个可以下载的免费版本。它可以在一台机器上在虚拟分布模式下运行 Hadoop。虽然它们大多数都需要 Linux，但有一个可以在微软上运行的产品。我们不要忘了亚马逊。它可以加速和减速单独的机器和集群，以运行 Map/Reduce 任务和测试你的代码。

另外，像 Infochimps 这类公司将数据集放在 S3 上供公共使用和下载。这就使那些没有硬件和基础设施的人可以非常容易地去运行 Hadoop。

此外，世界各地还有许多 Hadoop 相关的用户组。所以可以很容易找到一个离你很近的用户组，如果还没有，那么你也可以创建一个。找到同样有志于学习 Hadoop 的人，能使经历更加有趣，而且过程也不会那么痛苦。

最后同样重要的是，你可以到讨论组和论坛中提问，你会得到答案的。

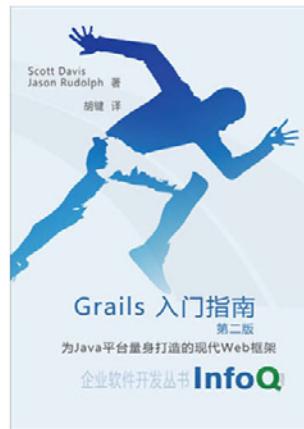
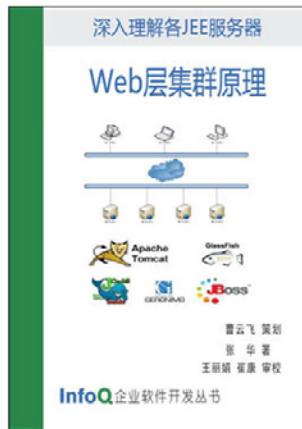
英文原文链接：<http://www.infoq.com/cn/articles/HadoopVirtualPanel>

相关内容：

- [Netflix 的云中数据仓库架构和 Hadoop PaaS——Genie](#)
- [Hadoop + SQL Server + Excel = 大数据分析](#)

InfoQ 软件开发丛书

欢迎免费下载



商务合作: sales@cn.infoq.com

读者反馈/内容提供: editors@cn.infoq.com

新品推荐

JavaScript 成为 GNOME 的首选语言

作者 Jeff Martin 译者 李彬

Linux 操作系统中流行的 GNOME 桌面环境正在将 JavaScript 变成其主要的应用程序开发语言。尽管其他语言也将得到支持，但 JavaScript 将享有最多的支持。这个决定引发了人们对这一选择是否值得的争论。

原文连接：http://www.infoq.com/cn/news/2013/02/javascript_gnome

Facebook 引入 Chef 来管理其 web 层

作者 Matthias Marschall 译者 李彬

Facebook 引入了 Private Chef (Opscode 的商业基础架构自动化产品) 来管理其 Web 层。为保证 Chef 满足 Facebook 的可伸缩性要求，他们帮助设计了 Chef 服务器的最新版本，该版本用 Erlang 完全重写了。

原文连接：<http://www.infoq.com/cn/news/2013/02/facebook-chef>

Xamarin 2.0 带来新的 IDE、支持 iOS 的 Visual Studio 插件和组件商店

作者 Abel Avram 译者 臧秀涛

Xamarin 在完成其为跨平台移动开发提供一组通用工具的愿景方面又迈出了重要一步。伴随 Xamarin 2.0 的宣布，Xamarin 重新命名了其产品，带来了新的 IDE Xamarin Studio、支持 iOS 开发的 Visual Studio 插件和组件商店。

原文连接：<http://www.infoq.com/cn/news/2013/02/Xamarin-2>

Miguel de Icaza 对 InfoQ 详细介绍了组件商店。

Cloud Foundry Core——保持云应用的可移植

作者 Kostis Kapelonis 译者 郑洁

Cloud Foundry Core 是一个 Web 应用，该应用针对一系列通用的运行时和服务对公共云实例（Cloud Foundry 端点）进行验证。该应用为提供 Cloud Foundry 实例的企业提供了跨平台的可移植性。同时，Cloud Foundry 还发布了一个新版本的 Micro Cloud Foundry，支持 Java 7.0 ,JRuby ,Play 2.0 框架等。

原文连接：<http://www.infoq.com/cn/news/2013/02/cloud-foundry-core>

微软为 Windows Azure 创建基于 Linux 的虚拟机的目录

作者 Richard Seroter 译者 李彬

微软开放技术公司公布了其第一个主要服务，即在 Windows Azure 云服务中提供存储库，以使预配置的虚拟机可以快速部署。据称，VM Depot 适用于发现和部署 Azure 友好的虚拟机，并可能成为微软面对热门的“亚马逊网络服务（AWS）”所作出的竞争手段。

原文连接：<http://www.infoq.com/cn/news/2013/02/microsoft-vm-depot>

Android 运行 Windows 应用，黑莓运行 Android 应用，以及 Ubuntu 手机的消息

作者 Jonathan Allen 译者 李彬

对移动开发者社区而言，二月份有许多新闻。我们已经见到在黑莓上运行 Android 应用，在安卓设备上运行 Windows 应用，也看到了 Ubuntu 手机发布日期的公布。

原文连接：<http://www.infoq.com/cn/news/2013/02/Mobile-Roundup>

MyGet 为 CodePlex、GitHub 和 BitBucket 提供免费构建服务

作者 Jonathan Allen 译者 邵思华

拥有 MyGet 帐户的开发者现在可以免费使用 MyGet 构建服务的公开 Beta 版本了，这项服务允许使用 CodePlex、GitHub 和 BitBucket 等服务的开发者将 MyGet 用作他们的自动构建服务器。

原文连接：<http://www.infoq.com/cn/news/2013/02/MyGet>

Twitter 发布基于组件的轻量级 JavaScript 框架——Flight

作者 郑柯

Twitter 日前发布了 Flight 项目，这是一个轻量级的、基于组件的 JavaScript 框架，可以将行为映射到 DOM 节点上。Twitter 将其用在自己的 Web 应用上。HackerNews 上对该项目有不少有趣的讨论。

原文连接：

<http://www.infoq.com/cn/news/2013/02/Twitter-Flight-Framework>

Visual Studio 拥抱 Git

作者 Jeff Martin 译者 邵思华

目前流行的开源分布式源代码控制系统 Git，得到了 Visual Studio 2012 和 Team Foundation Service 的原生支持。Microsoft 最新发布的插件提供了对 Git 的原生支持，同时也为集中式源代码管理模型提供了一种替代方案。

原文连接：http://www.infoq.com/cn/news/2013/02/vs2012_git

Meteor 0.5.3 发布：改进的性能与实时的反应式更新

作者 Tim Heckel 译者 张龙

近日，Meteor 开发团队发布了其框架的 0.5.3 版，该版本对新特性与增强的后端稳定性做了一些平衡。

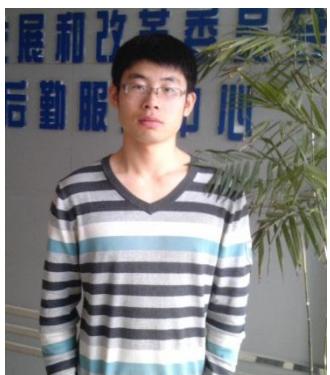
原文连接：<http://www.infoq.com/cn/news/2013/02/meteor-0.5.3>

1kg.org 多背一公斤

爱自然 | 更爱孩子



推荐编辑 | 翻译团队编辑孙镜涛



很荣幸能够成为本期的推荐编辑，感谢加入 InfoQ 以来贾国清、马国耀、郑柯、臧秀涛以及其他同仁对我的帮助和信任，使我能够快速的融入到这个大家庭中。

算起来，自己成为 InfoQ 的关注者已经很久了，早在 2009 年刚开始工作的时候就把 InfoQ 设置为每日必读，虽然当时也有想加入 InfoQ 的冲动，但是怯于自己的能力，一直也没有付诸行动。转眼间自己已经工作了将近四年，四年的时间让自己在解决方案、设计理念、技术

积累和组织沟通方面成熟了很多，自感借助于现在的经验和知识不至于向他人传递错误的信息；同时通过对《WebMatrix ASP.NET Web 开发入门经典》一书的翻译，使得自己的英文水平勉强达到可以实用的要求；再三考量之下最终于 12 年下半年付诸行动，并在以上同仁的帮助下实现了自己加入 InfoQ 的想法。IT 从业者真的很不容易，无论谁都不能固步自封，必须时刻关注业界动态，掌握最新的技术趋势，摸准最终用户的需求，关注同行的产品发布，否则偌大的市场也必然没有自己的生存空间。这就需要一个平台，使我们能够掌控这些信息，而 InfoQ 恰好就提供了这样一个平台，这也是我加入其中的原因，在这里你不仅是一个信息的受益者，同时还是一个信息的传播者，不仅能够及时获取最新的信息，还能够通过自己的表述让更多的人受益，能够让自己不断的成长，还能够见证社区的发展，如此是多么令人兴奋的一件事情呀！

编辑如同老师，“师者，传道授业解惑也”，作为信息的传播和分享者，必须有认真负责、谦虚谨慎、执着严肃的态度，否则就难以胜任这一职责。InfoQ 编辑团队便秉承了此文化，每一个人都尽所能的将事情做到最好，每一篇文章的发布都需要经过仔细的审校，也正是这样一种文化使得 InfoQ 得到了大家的认可，成为 IT 从业者的每日新闻，并吸引越来越多志同道合的人加入到这个大家庭里面。如果你有这样的态度，如果你符合这样的文化，如果你我志同道合，那么请加入到这个团队，一个优秀的团队离不开新鲜的血液，离不开每个人的积极性，而一个活跃向上的团队必将带动个人的发展，引领信息的传播。

最后说两句闲话，来个自我介绍，本人孙镜涛，毕业至今就职于浪潮软件技术中心，从事工作流引擎及设计工具和楼上平台 studio 的研发，对流程化、移动和云技术感兴趣。个人博客：<http://jingtao.cnblogs.com>，<http://sunjingtao.com>

架构师

www.infoq.com/cn/architect

每月8号出版

时刻关注软件开发领域的变化与创新

架构师

11月 ARCHITECT

特别专题
光棍节狂欢的背后——
电商系统深探
1号店B2C电商系统深造之路
百万点推荐引擎——从需求到架构
麦当劳购物系统浅谈分享
REST的远程API设计案例
大型Rails与VoIP系统架构
与部署实践
什么是Node.js
扩展Oozie
浅谈dojox中的一些小工具

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

架构师

10月 ARCHITECT

特别专题
大数据时代
大数据
大数据时代的数据管理
阿里巴巴数据架构设计经验与挑战
大数据时代的创新者们
关系数据库还是NoSQL数据库
向Java开发者介绍Scala
HTML 5 or Silverlight?
解析JDK 7的Garbage-First收集器
了解云计算的基础

Steve Jobs
1955-2011

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

架构师

9月 ARCHITECT

特别专题
QCon全球企业大会精华点滴
QCon在中国的三年回顾
麦肯锡对阿里巴巴国际站架构演进
畅销书《IPS》和《技术流年》
新浪微博团队建设的虚与实
沐泽宁谈主观决策架构
跟着李树学Oozie
如何查看我的订单—
REST的远程API设计案例
通用系统思考，走上改善之路
Redis内存使用优化与存储

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

架构师

8月 ARCHITECT

特别专题
云计算的安全风险
圆桌会议：云计算的安全风险
设计一种云级别的身份认证结构
云应用和平台的现状：
云采纳者如是说...

Java虚拟机家族考
专家视角看IT转型
为什么使用 Redis及高产品定位
架构演化之谜

InfoQ 每月8号出版

时刻关注企业软件开发领域的变化与创新

架构师

7月 ARCHITECT

特别专题
深入理解Node.js
为什么要用后端工程语言Node.js
虚拟研讨会：Node.js生态系统之
概览、棒、最佳实践
使用JavaScript和Node.js构建Web应用
Node.js的起源和实践应用—
专访Node.js创始人Ryan Dahl

Java深度剖析十：Java对集群划分与热切换
将数据打散之一：关于松散的数据设计
微服务平台部署与PaaS
社区驱动的源码控制
来自Padmanab的真言良言

InfoQ 每月8号出版

封面植物



桃花，即桃树盛开的花朵，属蔷薇科植物。叶椭圆状披针形，核果近球形，主要分果桃和花桃两大类。桃花原产于中国中部、北部，现已在世界温带国家及地区广泛种植，其繁殖以嫁接为主。桃花可制成桃花丸、桃花茶等食品。其具有很高的观赏价值，是文学创作的常用素材。此外，桃花中元素有疏通经络、滋润皮肤的药用价值。其花语及代表意义为：爱情的俘虏。每年3月份，各地会以桃花为媒，举办桃花节盛会。桃花为落叶乔木。叶椭圆状披针形，叶缘有粗锯齿，无毛，叶柄长1~1.5cm。高可达6~10米。树干灰褐色，粗糙有孔。小枝红褐色或褐绿色，平滑。花单生，有白、粉红、红等色，重瓣或半重瓣，花期3月。核果近球形，表面密被短绒毛，因品种不同，果熟6~9月。主要分果桃和花桃两大类。变种有深红、绯红、纯白及红白混色等花色变化以及复瓣和重瓣种。较重要的变种有：油桃、蟠桃、寿星桃、碧桃。其中油桃和蟠桃都作果树栽培，寿星桃和碧桃主要供观赏，寿星桃还可作桃的矮化砧。树高4~5米。一年生枝条红褐色。叶多呈披针形，叶缘有锯齿，叶柄基部常生蜜腺。花型有蔷薇型和铃型两种。核果除蟠桃外，多为圆形或长圆形，果面除油桃外，均布有茸毛。果肉白、黄色或夹红晕，少数呈红色；肉质柔软、脆硬或密韧；核表面具不同沟点纹路，均为种和品种群的重要分类依据。桃花是中国传统的园林花木，其树态优美，枝干扶疏，花朵丰腴，色彩艳丽，为早春重要观花树种之一。桃的果实是著名的水果；桃核可以榨油；其枝、叶、果、根俱能入药；桃木细密坚硬，可供雕刻用。



架构师 3 月刊

每月 8 日出版

本期主编：霍泰稳

美术/流程编辑：水羽哲

总编辑：贾国清 发行人：霍泰稳

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

InfoQ 中文站新浪微博：<http://weibo.com/infoqchina>

商务合作：sales@cn.infoq.com 15810407783



本期主编：霍泰稳，InfoQ 中文站的联合创始人兼 CEO

有多年的软件开发经验和媒体从业经历，以技术传播为己任，关注企业软件开发领域的变化与创新。曾先后参与《程序员》杂志、《MSDN 开发精选》杂志、《开源大本营》图书和《开源技术选型手册》2008 版图书的策划编辑工作。