

架构师

12月 ARCHITECT



特别专题

内核那些事

大数据处理之如何确保断电不丢数据

UNIX环境高级编程：Stephen Rago访谈

Linux可插拔认证模块的基本概念与架构

敏捷与结构性模块化

敏捷与结构性模块化（一）

敏捷与结构性模块化（二）

敏捷与结构性模块化（三）

大数据安全：Hadoop安全模型的演进

如何让你的内存NoSQL数据库为企业应用做好准备



卷首语

创业和预测未来

“历史和社会不是缓慢爬行的，而是在一步步地跳跃。它们从一个断层跃上另一个断层，其间极少有波折。而我们（以及历史学家）喜欢相信那些我们能够预测的小的逐步演变。我们只是一台巨大的回头看的机器。”

——《黑天鹅：如何应对不可预知的未来》

今天我们所见到的世界，有多少是你在2010年就预见到了的？有多少是你在2008年就预见到了的？有多少是你在2006年就能预见到的？

出来创业的人，在创业过程中大概总有某一个阶段会在心里描绘一幅三年之后、五年之后或者七年之后的市场画面。他们试图找到这个“想象中的世界”与现实世界之间的差距，并作出假设：如果能够填补这个差距，就能到达那个“想象中的世界”。那个世界中也许有一整个今天还不存在的市场，也可能只不过是一些细微的技术改进或模式改进。不管“想象中的世界”和今天的差别是大是小，它总归是一个新的世界。

这是一个预测未来的过程。

未来是一个概率事件。历史告诉我们，什么事都有可能发生，只是概率不同。作为一位预测者，如果你收集到尽可能多的、相关的可靠情报，你对“这件事可能会发生的几率”的判断会比其他人更准确；如果你这时去赌博（比如投资），就有更大的胜率。

但是，预测未来只是创业过程中很小的一部分。创业者跟大部分预测者的不同在于：

- 1、创业者有必要进入大部分人认为“不太可能发生的未来”
- 2、创业者的大部分工作是为了增加“这件事可能发生的几率”，或者“提前这件事真实到来的时间”

比如，IBM创始人曾预测人类只需要几台计算机。但是当计算机足够便宜、易用之后，市场自己解决了“计算机能做什么”这个问题。

比如在站长自己架站的时代，大部分人想不通一个普通人或者店铺做一个网站有什么用；但是当建立一个自己的页面变得足够简单之后，市场自己解决了这个问

题。

比如我自己在2012年之前，一直坚持认为我需要一个全键盘的、待机时间长的手机，直到我入手了我的第一台Android手机之后，我才发现一个真正智能的系统完全可以“一美遮百丑”。

众所周知，创业的成功率很低，因为你不仅需要在正确的时间站在正确的地方，还需要找到推动事件前进的“杠杆”才能增加事情发生的几率。但是在事情真正发生前，你很难猜到这个“杠杆”是什么。所以市场的规则是，创业者们前仆后继的涌入，各自拿出各自猜测的可行道路，最后有极少数的道路走通了，于是我们进入了新时代。

（当然，极少数的道路成为了未来，并不意味着其他道路无法走通。这一切只因为大部分可能发生的未来都是小概率事件。）

在往新世界前进的道路上，大家心里那个“想象中的世界”不会一成不变，往往会进行一些修订。

有些修订出于对现实的妥协，这并不是理想的，因为妥协可能意味着一系列目标下滑的一个开端，结果是你折回到了旧世界当中，沦为平庸。

有些修订出于早期对时间和地点的预判偏离，这是应该的，这会有助于你减少浪费。

有些修订出于你看到了以前没看到的一些东西，这是必要的，因为未来往往来自意外。比如，发现宇宙背景微波辐射的两位科学家本来只是在找天线上的鸟粪。

我觉得旅途当中最重要的是，绝对不要忘记你的“新世界”将带来哪些新的价值。你在行走的路上会遇到各种各样的干扰：我要去学习谁的模式？我要去抢谁的生意？我要去打造什么样的差异化？

不要因此而分心。

我们学习历史，为的是学习他“为什么会发生”，而不是学习他“如何发生的”。没有发生的未来并非不可能发生，我们只不过在赌一个几率。你要做的是专心寻找那只杠杆。


记住你的愿景，反复在脑海中描绘你的新世界，你有更大的可能性会到达那里。

还有，我所说的，很可能是错的。

“如果你是石器时代的哲学家，你的部落首席计划官要求你在一份综合报告中预测未来，你必须预测到车轮的发明，否则你就会错过大部分人类进展。……

预测要求我们知道将在未来发现的技术。但认识到这一点几乎会自动地让我们立即开始开发这些技术。因此，我们不知道我们将知道什么。”

最后给大家赠送一句摘抄：



“我们制造玩具，有些玩具改变了世界。”

本期主编：杨赛

InfoQ^{ueue}

促进软件开发领域知识与创新的传播



中文 | 英文 | 日文 | 葡文 | ……

目录

人物 | People

Codenvy背后的技术--一次与CEO TylerJewell的深入访谈

观点 | View

关于MongoDB你需要知道的几件事

IT领域2014年发展趋势

本期专题：内核那些事 | Topic

大数据处理之如何确保断电不丢数据

UNIX环境高级编程：Stephen Rago访谈

Linux可插拔认证模块的基本概念与架构

推荐文章 | Article

大数据安全：Hadoop安全模型的演进

如何让你的内存NoSQL数据库为企业应用做好准备

特别专栏 | Column

敏捷与结构性模块化（一）

敏捷与结构性模块化（二）

敏捷与结构性模块化（三）

避开那些坑 | Void

Spring Web应用的最大瑕疵

你应该远离的6个Java特性

新品推荐 | Product

在Twitter，Netty 4 GC开销降为五分之一

Android 4.4 KitKat新特性介绍

谷歌发布Dart 1.0

Spring Data Neo4j简介

TypeScript综述：新功能、工具和路线图

Apigee现在支持Node.js 并开源了Volos

QClub

我们影响有影响力的人

非盈利、非商业、纯技术交流的技术社区活动

We are QClub

QClub作为InfoQ线下技术沙龙品牌，定期在全国主要城市举办免费的技术沙龙，邀请国内外知名公司技术总监，项目经理，高级研发工程师等走入各地技术社区，分享他们的经验与对行业趋势的预测与讨论，为中国技术人员搭建交流、分享的平台，尽自己微薄之力架起中高端技术人员之间的桥梁，为中国技术社区的发展与价值的传播贡献自己的力量。

非盈利 非商业 纯技术交流

QClub的话题

QClub的话题可能包括任何当地技术人员感兴趣的话题，比如：编程语言、架构设计、企业级开发、运维、基础架构、过程与实践、云计算、大数据、互联网、移动开发、安全、敏捷、性能、业务流程、SOA、.....



参会人群：

开发人员、技术或团队负责人、技术爱好者、学生等

活动城市：

北京、长沙、成都、大连、福州、广州、上海、太原、天津、温州、西安、郑州、.....

举办周期：

每个城市
每1-2个月一次活动

活动形式多样：

- 讲师演讲
- Open Space
- 户外活动
- 线上交流
- 粉丝 QQ 群
- 等等.....

QCon

International
Software Development
Conference

全球软件开发大会

2014年4月25—27日 北京国际会议中心

北京站 2014



7折 火热抢票中

截止12月31日

技术专题精彩呈现

• 知名网站案例分析

成功的网站发展历程中的经验教训

• 构建高效能团队

适应公司发展的，有效推动业务的，持续改进的高效能团队

• 敏捷测试

怎样持续敏捷地创造价值，为谁创造价值

• 云计算架构案例

设计云计算，不只是部署

• 推荐系统工程实践

探讨推荐系统的系统、架构、算法和策略，分享各大公司在应用推荐系统时的工程技术和实战经验

• 安全专题

白帽子分享知名互联网公司安全经验，黑帽子同台竞技见招拆招

• 跨终端的Web

从浏览器端到服务器端、PC端到移动端；开发到测试、简单应用到富客户端Web应用

• 扩展性、可用性与高性能

讨论大型复杂系统中，如何在架构设计、代码、运维体系等方面达到扩展性、可用性与高性能

• 大数据处理与大数据应用

关注大数据处理和分析的最新技术和工具，以及互联网公司的创新数据产品

• 移动应用案例分析

移动浪潮席卷全球，设计、技术和商业的完美融合才是最重要的产品元素

• 文化专题

团队文化究竟是什么？如何被创建、传承与发扬？

• 未来的开发者工具

IDE、编辑器、编程语言的未来

• 尖端之上的Java

企业级开发核心生产力的最新信息与框架分享

• 自动化运维

系统架构设计，CDN加速、日志搜集、故障定位，都是当今自动化运维追求的目标

• 持续集成与持续交付

促进开发流程中的协作，来达到更快、更可靠、低成本的自动化软件交付

人物 | People

Codenvy背后的技术--一次与CEO Tyler Jewell的深入访谈

作者 [Abel Avram](#)，译者 [陈菲](#)

与[Eclipse Orion](#)，[Cloud9](#)及其它新一代IDE一样，[Codenvy](#)也是其中一员。它们都试图转变开发人员创建软件的方式，为他们提供可以在任何主流浏览器上执行的前端界面，以及可以在服务器或云中执行的后端，该部分将处理所有的力气活：编译、代码分析、构建、预备、发布等等。

Codenvy可以支持多种语言的在线开发，其中有Java、JavaScript、HTML5、CSS、PHP、Python、Ruby及XML。同时更多的工作也在进行中，以便支持更多语言，同时为社区提供所需工具用来添加想要的语言。对于框架也是一样的，Codenvy目前支持Spring、Node.js、JSP、Rails、GWT和Django等框架。对于Android开发的支持也在Google I/O 2013对外发布了。

应用程序在IDE里就可以往不同PaaS上发布：Amazon (Elastic Beanstack)、CloudBees、Cloud Foundry、GAE、Heroku、OpenShift、AppFog和Ti er3。

对于代码仓库，Codenvy一开始就支持GitHub和[BitBucket](#)，随后支持Git和Mercurial代码库，但是最近添加了对Assembla、BeanStalk、Codebase、Deveo、GitEnterprise、Gitorious、PikaCode、ProjectLocker和Unfuddle的支持。Codenvy使用Jenkins、Hudson或CloudBees来持续集成。

Codenvy支持结对编程，它含有实时多光标协作功能。其代码编辑器包含有语法高亮，代码补全和重构功能。

我们采访了Tyler Jewell，Codenvy的CEO，以便挖掘出那些为该IDE提供动力的技术背后所存在的更多细节。

InfoQ: Codenvy是由什么构造的？其背后技术、服务器和客户端都是什么？

TJ: Codenvy作为一个云系统，包含有多个节点来处理路由，负载平衡，身份管理，IDE，生成器，运行器和分析器。核心IDE本身是用Java编写的，并由GWT生产优化后的客户端JavaScript。对于大部分浏览器和服务器之间的沟通，我们则采用WebSockets。其它在系统中广泛使用的技术包括：Apache Web Services，CloudFoundry，Gluster，CodeMirror，Collide（协同

编辑器)，Hadoop和Apache Pig。其实Codenvy的核心是作为构建云IDE的平台。在内部，我们利用一个运行时框架来管理发现，加载，协调和激活分层扩展。扩展是以插件形式打包好提供语言、框架、编辑器、文件管理、生成器指令和运行器指令。然后我们创建一个IDE框架链接各个插件并部署到云框架，用来提供租赁、灵活性、日志记录、身份管理和隔离控制。我们建立了自己的多租户IP，这样在单个JVM上就能运行最多250个IDE，同时也防止IDE在执行生成和运行流程时产生拥堵现象。我们针对OAuth创建了自己的拓展，用来允许入站身份验证请求，以及向外拓展身份到云下游以简化用户工作流。我们使用puppet和大量的Bash脚本来处理与Devops相关的问题。而我们的网站，作为独立于我们云生命周期的产品，是通过JavaScript、Yeoman、Grunt和Jekyll撰写的。而对于工具，我们则大量使用GitHub、Jira、Confluence和Selenium。Codenvy的原始代码是通过使用Eclipse和IntelliJ来撰写的，但是现在大部分Codenvy是通过Codenvy本身来撰写的。

我们使用超过100多个开源库来创建运行现有Codenvy的各种元素。

InfoQ: Codenvy是在哪里运行的，在您自己的服务器上，还是在云里？到底是哪个？

TJ: 我们有开发（验收测试），临时（完整的测试环境），准产品（允许市场营销、技术支持和文档参与进来的最终确定版本），以及产品等不同环境。我们在AWS上运行产品。准产品则在我们自己位于欧洲的数据中心内，执行于Eucalyptus之上。我们将临时环境运行于AWS，这样就能进行Devops配置，测试多种不允许内部访问的云API，以及执行大规模的测试。我们的开发服务器环境都是包装有Codenvy的单服务器，它们能根据配置执行于1到2个虚拟机上。针对每个功能特征，我们使用开发环境（能同时运行6个）驱动从PM和用户那所获取的验收标准。在Sprint中，验收服务器会在我们自己的数据中心中运行每个功能。当Sprint结束后，临时环境和预产品环境将通过Puppet程序自动更新。

InfoQ: 你们是否有打算支持其它语言呢？如果是，那将是哪些？

TJ: 目前就我看来，我们不可能支持太多的语言。在这里，我们有两个方向可以考虑：一是全新的语言，需要专门的构建和运行环境来执行；另外就是可以运行于VM的语言，类似那些执行于JVM的。如果单靠我们自己团队，有太多的环境需要支持，因此我们把社区引入进来。我们一直在努力完成内部所使用的SDK，然后将其发布出去，这样大家就能创建自己的扩展和插件。它的架构和Eclipse插件相符，只会更广泛一点。Codenvy内部是个完整的构建子系统

和执行Cloud Foundry的运行环境。因此，如果开发人员想往系统中添加新语言，比如：C，就需要添加常用的IDE部件（文档、语法、调试、重构和宏）。同时也需要给Codenvy相应指令去操作构建系统（make）和运行环境。

我们的首要任务就是将该SDK发布出去，而这将是在今年；随后，我们的第二个重点是将围绕不同框架和语言的社区组织起来。我们也有来自外部的人有兴趣参与到C、C++，Objective C、Go、Scala、Plan等语言中。

InfoQ：那对框架/PaaS/CI的支持呢？

TJ：在每个Sprint中我们都尝试添加新的东西。我们执行为期两周的Sprint，因此这方面的列表总是在变化。之所以Codenvy这么快能走红，是因为我们有个广泛的集成，而且也没有放缓的倾向。我们最新列表也是广泛分布的。我们为GitHub、Bitbucket、AWS、Google App Engine、Heroku、Cloud Foundry、OpenShift、CloudBees、Tier3和AppFog所提供的高度支持，感到非常自豪。我们也刚刚完成大概10个git供应商的认证，因此我们将更新所有规范和how-to，以更好地与各供应商协作。

对于CI的支持，我们启用了post-comment hooks，以及让Web Service直接调用CI供应者。

InfoQ：该SDK提供了什么？我理解是开发人员能够自己添加语言或平台。还有该SDK将在什么时候发布？

TJ：该SDK提供了一个API用于编写分层拓展；一个API用于分离UI、数据（文档）和事件模型，以便单独考虑；API文档；一组插件实例，比如我们给Java所设计的那样；一个专门的SDK云允许开发人员上传插件测试。插件本身将通过Java和GWT来编写。除非是在开发模式下运行，否则GWT需要11分钟编译。我们打算让GWT为Codenvy设置一个开发模式，但是从简化创建流程的最初目的上看，我们需要开发人员上传插件到Codenvy的伙伴云中，在那里我们可以编译、打包以及集成到运行时中。

该SDK云的最初版本会发布于2013年夏天。我们已经签订了好几个ISV合作伙伴，他们对此非常期待，因此这是我们内部的最高优先级。

在首次发布后，我们将创建Codenvy WAR文件包。这样开发人员就可以在Tomcat中运行Codenvy的单服务器版本，同时本地构建和发布插件。该版本将会有点棘手，因为它没有包含对运行器的支持。由于我们使用的是Cloud Foundry执行运行器，目前为止还没有找到一个简单的方法将运行器包装到单服

务器模式。我们将允许开发人员映射他们的“云运行器”到Tomcat中运行来处理localhost系统。因此如果在localhost的浏览器中执行“Run”，也将触发你在同一机器中所定义的进程命令。

InfoQ: Codenvy能离线工作吗？如果不能，你们的计划是什么？

TJ: 我们不支持离线工作。在某种程度该话题本身就能成为一个初步招股说明书（a red herring）。从UserVoice获知，该特性是排名第4或第5个大家所需要的。但是当我们询问开发人员他们为什么需要这个特性时，答案却每个都不一样。有些人担心飞机旅行，需要完全离线；而有些则考虑如果只有部分网络访问能力的话，情况会怎么样；但大部分人并不认为如果没有该功能会是个大问题，而提出该需求本身也是件非常自然的事。

提供离线支持的一部分在于决定哪些可以离线。我们可以很容易地提供一个文本编辑器，但不会有对Git指令、构建、调试、运行或部署的支持。因此，从最基本层面看，初始阶段允许开发人员离线时有个只读模式是有一定道理的。而这在将来也非常容易做到。除此之外，不同层次的离线功能将取决于你是否可以或愿意允许在客户端安装它们。如果系统完全基于浏览器，那么我们可以同步Git代码库，并提供编辑功能。如果允许我们在桌面上安装软件，那么我们也可以在本地上创建一个完全同步的构建和运行器。

在这方面，我们还在评估哪方面是我们想做的，并希望与社区进行更多的讨论。

InfoQ: 你能就Codenvy对JRebel、Rally Dev和Tasktop的集成上提供更多的细节吗？

TJ: 为协调Java工程我们提供了JRebel。首先建立一个单一且永久的运行器，然后在生成器和运行器集群间激活JRebel。

我们会在今年年底讨论我们将为Rally和TaskTop做哪些工作。

InfoQ: 目前只有社区版是可用的。你们打算什么时候推出其它已经宣布定价的版本呢？

TJ: 其实，我们已经有一些付费服务可供出售。现在，我们为那些个人使用者提供了“[早期采用者项目](#)”，并提供66%的折扣。随着时间的展开，我们对于在

每一个版本中增加的高级特性更有自信,价格将会逐年增加.

InfoQ: 企业版将包含什么?

TJ: Codenvy企业版将会是一个安装在防火墙后能管理大量开发者的云系统. 迄今为止使用的客户都是那些有监管, 合规, 安全, 外包管理需求的. 企业版除了包含公共云的特性之外还加入了一些特殊功能: a) LDAP(轻量级目录访问协议)&认证集成, b) 集群, 负载均衡, 和高可用性策略控制, 和c) 分析和汇报以衡量采购, 参与度, 以及开发团队、工作空间和项目的活力. 现在, 客户们自己提供硬件, 然后在Tier3、AWS、Eucalyptus或OpenStack上运行Codenvy Enterprise.

有趣的是, 很多Codenvy Enterprise的早期使用者并没有把它当作Eclipse的替代品. 相反, 它被作为一个辅助的系统. 很多情况下, 我们会替换离岸开发人员运行缓慢的VDI或TS环境. 在其他情况下, 当花时间在有限窗口下编程比学会如何配置开发环境更重要的时候, 我们正在变成新员工、实习生或合作伙伴的培训基地. 就算作为辅助使用, 开发人员还是可以推广自己的云, 只是他们需要自己负责管理桌面, 并让IT负责管理Codenvy.

InfoQ: 对于将来, 你们有什么计划?

TJ: 我们非常有幸拥有30个工程师在我们的开发团队. 他们都是非常出色的系统工程师, 在构建大而复杂的系统上有很丰富经验. 我想这也正是他们的动力之源, 促使他们开发设计出了大量能帮助开发人员自动化复杂工作流的项目. 我们的精力将分散到以下这几个分类中:

1) 支持开发人员. 更多的语言, 更多的框架, 更多的云集成和其他云服务将被引进. 我希望在年底能集成250个技术到Codenvy中. 此外, 由于与Triggler.IO和Google的合作, 我们已公开承诺将添加Android和iOS开发.

2) 支持独立软件开发商 (ISV). 在第三季度, 我们将交付各种ISV软件包, 这样用户就能自定义模板, 推广示例应用程序, 以及内部构建/测量/跟踪开发活动. 最终, 我们希望所有ISV都能在线配置自定义的、按需的IDE. 这一点, 在很多方面, 需要结合他们在SDK方面的情况.

3) 支持企业. 我们寻求更多的POC, 及在大企业中的安装. 我们已经在一些大公司上完成安装, 但是我们想再完成大概25个后, 再宣称Codenvy Enterprise GA. 任何想自己运行开发云的公司, 都可以联系我们, 我们将非常荣幸.

关于被访者



Tyler Jewell是Codenvy的CEO。同时也是Toba Capital的投资合伙人，他主要关注中间件和应用开发投资。他是WSO2、Exo Platform和Codenvy的董事会成员，同时也投资了Cloudant、ZeroTurnaround、InfoQ和AppHarbor。

参考英文原文：[The Technology behind Codenvy. An Interview with Tyler Jewell, CEO](#)

感谢[陈菲](#)对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（[@InfoQ](#)）或者腾讯微博（[@InfoQ](#)）关注我们，并与我们的编辑和其他读者朋友交流。

原文链接：<http://www.infoq.com/cn/articles/codenvy-interview>

相关内容

- [Webix JavaScript UI 库可以帮你构建跨平台的HTML5 和 CSS3 程序](#)
- [WebStorm 7.0 支持更多的Web技术](#)
- [Bootstrap 3拥有全新视觉方案和更多的组件](#)
- [Web响应图像综述](#)
- [Fries: 使用HTML、JavaScript和CSS搭建Android本地接口](#)

观点 | View

关于MongoDB你需要知道的几件事

作者 [张龙](#)

Henrique Lobo Weissmann是一位来自于巴西的软件开发者，他是itexto公司的联合创始人，这是一家咨询公司。近日，Henrique在博客上撰文谈到了关于MongoDB的一些内容，其中有些观点值得我们，特别是正在和打算使用MongoDB的开发者关注。

到目前为止，MongoDB在巴西是最为流行的NoSQL数据库（至少根据关于MongoDB的博客数量以及文章所判断）。MongoDB是个非常棒的解决方案，不过困扰我们的是很少有人了解过关于它的一些限制。这样的事情正在不断上演：人们看到MongoDB的限制，心里却认为这些是它的Bug。

本文列举了颇让作者困惑的一些MongoDB限制，如果你也打算使用MongoDB，那么至少要提前了解这些限制，以免遇到的时候措手不及。

消耗磁盘空间

这是我的第一个困惑：MongoDB会消耗太多的磁盘空间了。当然了，这与它的编码方式有关，因为MongoDB会通过预分配大文件空间来避免磁盘碎片问题。它的工作方式是这样的：在创建数据库时，系统会创建一个名为[db name].0的文件，当该文件有一半以上被使用时，系统会再次创建一个名为[db name].1的文件，该文件的大小是方才的两倍。这个情况会持续不断的发生，因此256、512、1024、2048大小的文件会被写到磁盘上。最后，再次创建文件时大小都将为2048Mb。如果存储空间是项目的一个限制，那么你必须要考虑这个情况。该问题有个商业解决方案，名字叫做[TokuMX](#)，使用后存储消耗将会减少90%。此外，从长远来看，repairDatabase与compact命令也会在一定程度上帮到你。

通过复制集实现的数据复制效果非常棒，不过也有限制

MongoDB中数据复制的复制集策略非常棒，很容易配置并且使用起来确实不错。但如果集群的节点有12个以上，那么你就会遇到问题。MongoDB中的复制集有12个节点的限制，这里是问题的描述，你可以追踪这个问题看看是否已经被解决了。

主从复制不会确保高可用性

尽管已经不建议被使用了，不过MongoDB还是提供了另外一种复制策略，即主从复制。它解决了12个节点限制问题，不过却产生了新的问题：如果需要改变集群的

主节点，那么你必须得手工完成，感到惊讶？看看这个[链接](#)吧。

不要使用32位版本

MongoDB的32位版本也是不建议被使用的，因为你只能处理2GB大小的数据。还记得第一个限制么？这是MongoDB关于该限制的[说明](#)。

咨询费非常非常昂贵（至少对于巴西的开发者与公司来说如此）

我不清楚其他国家的情况，不过至少在巴西MongoDB的咨询费是个天价。对于“[Lightning Consult](#)”计划来说，每小时的价格是450美金，而你至少需要购买两个小时的，换句话说，对于任何一家公司来说，每次咨询的价格至少是900美金。相比于RedHat和Oracle来说，这个价格太高了。

差劲的管理工具

这对于初学者来说依然是个让人头疼的问题，MongoDB的管理控制台太差劲了。我所知道的最好的工具是[RoboMongo](#)，它对于那些初次使用的开发者来说非常趁手。

了解官方的限制

让我感到惊讶的是，很少有人会查询关于他们将要使用的工具的限制。幸好，MongoDB的开发人员发布了一篇MongoDB所有限制的[博客](#)，你可以提前了解相关信息，避免在使用过程中难堪。

各位读者，现在使用MongoDB的公司也越来越多了，不妨与大家分享你在使用这个NoSQL数据库时的一些经验与教训。

原文链接：<http://www.infoq.com/cn/news/2013/11/mongodb-things>

相关内容

- [MongoDB价值12亿美元的奥秘](#)
- [NoSQL基准对比Aerospike、Cassandra、Couchbase和MongoDB](#)
- [Simba Technologies正将SQL的强大能力带向Cassandra、Hadoop、BigQuery和MongoDB](#)
- [在MongoDB中实现聚合函数](#)
- [MongoDB、Java及ORM](#)

观点 | View

IT领域2014年发展趋势

作者 [崔康](#)

虽然2013年还有将近两个月才结束，但是有关2014年的发展趋势已经开始见诸于各个媒体。可以预见，在新的一年里，云服务、大数据、安全问题、内存计算等都将是发展的热点区域。

InfoWorld主编Eric Knorr[总结](#)了九个方面的发展趋势，其中云计算领域的热点最多。

云将是新硬件。相信Pivotal公司的CEO Paul Maritz的观点吧：新的计算平台驱动着产业的巨变——从PC到客户端到服务器到互联网。对于服务器、存储器、网络设备这些要迎合应用程序扩展性的“大机器”，我们必须对其基础设施进行可视化和集中控制——也就是软件定义。最后这个趋势将超越SDN，会包括数据中心的每一个系统，一直到HVAC。公共云供应商提供的先进的软件控制方案将继续涌向企业。

“参与系统（systems of engagement）”引领潮流。为什么你需要大范围的云的伸缩性呢？这不同于老式的企业“记录系统（systems of record）”，如ERP，在这种系统里数据模型很少变化，并且你很难知道有多少人在使用这个系统。现在云系统的趋势是“systems of engagement”：面向用户的网络或移动端应用，这些应用的用法可能随量级而变化。

优化用户交互是当今最热门的研究领域，这一研究推动着弹性基础设施、新数据库技术的发展，也推动着大数据分析收集技术的发展（主要针对网络点击流量以及一些其它用户数据）。基于Hadoop应用的数据分析技术是上个十年中最伟大的企业级技术。在其之后是NoSQL数据库，如MongoDB、Cassandra、Couchbase。这些数据库伸缩性好，并且可以对数据模型进行动态修改。

云计算集成移至前端。伴随着基于云的分析数据存储量的增加，大数据倾向于维持现状。一般的云计算，特别是存储自身数据的SaaS应用，很有可能重蹈覆辙，采用孤立组织——在这种组织中，关于同一个产品和用户的两个有些许差异的副本分散在孤立的数据存储中（一同存储的还有应该共享的、有价值的过程）。

解决方案有两个：云端整合与提供更多、更好的API接口。云端整合服务商比

比皆是，包括Cordys, Dell Boomi, IBM Cast Iron, Informatica, Layer 7, MuleSoft, SnapLogic, 和WSO2等。此外，API现在也有了它们自己的“军团”——如Apigee公司提供的API解决方案，这些API允许企业发布与保持企业自己的公有API。

企业开发用户开始转向PaaS。到目前为止，PaaS的主要用户是商用软件开发者和专业服务公司。但是伴随着越来越多的企业推出他们自己的网络和移动应用，企业开发者将看到诸如Microsoft Azure、Pivotal Cloud Foundry、Red Hat OpenShift、SalesForce Heroku等PaaS供应商能为他们带来的益处。这些供应商在云端提供应用程序敏捷编码、测试、部署所需的工具和服务。

除了云计算之外，大数据领域也位列其中。**大数据将超越自身。**大数据分析具有光辉的前景，但短期来看，很多大数据解决方案还不能解决实际问题。从长远来看，大数据技术将不止限于优化电子商务，更加推动各种产业——从制造业到交通运输到电网——的发展。

但是这些产业需要工业网络（也被成为物联网）上线。在这个网络中，相互连接的传感器将通过传递大量的遥测信息来改进产品设计，提高错误预测准确度等。通用电气和IBM是这个领域早期的领导者，我们在这个领域只是刚刚起步。若干年后当工业网全面展开时，大数据会变得非常非常大。届时人们将急需大量的大数据解决方案。同时，如果2014年要有什么泡沫要破灭的话，大数据首当其冲。

接下来就是安全问题。**身份（Identity）**是新的安全问题。这种说法有一点夸张，但事实是我们必须拓展身份以满足内部部署应用和SaaS应用的需求。管理什么人可以访问哪些内容——并移除离职职工的账户——这一工作已经变得越来越重要，越来越复杂。微软、Okta、Salesforce等公司正在推出这类解决方案。没有云端身份认证管理，企业将无法安全、有效的实施公共云解决方案。

内存是新的存储方式。大内存存在两个方面迅速发展。在软件方面，每一个相关的数据库供应商都在添加内存功能（主要用于分析），同时他们也在急剧缩减处理庞大过程工作的时间。硬件方面，像Conduktiv Technologies和PernixData一类的解决方案商在服务器端装配一个大的使用闪存的分布式缓存，通过这种方式他们可以减少实现SAN的读写操作量。

JavaScript的势头不可阻挡。由于源源不断的移动设备（包括新电视、汽车等等）的推出，当今硬件设备的用户多样性高于之前任何时刻。大家都不想给每一个平台开发单独的、原生的用户应用程序。如果你想要使用单一的代码库，你必须要保证你的应用程序可以运行在一个浏览器中——也就是说，这个程序必须是

一个基于JavaScript/HTML5的程序。

有着Famo.us这种机构不断推动Javascript的发展，我们不难理解貌似每周都有新的Javascript框架推出。此外，像PhoneGap这样的跨平台的移动开发环境允许JavaScript应用程序和本地应用程序的快捷转换。

开发者依旧是主导。如果说这些预言中有什么共通之处的话，那就是两年前Marc Andreessen的预言——软件正在吞噬整个世界。有这么多平台要写——甚至数据中心的基础设施也变成可编程化的了——我们已经没有足够的开发者来完成这些代码了。贪得无厌的需求带来了高薪水和高职位，至少对于那些拥有对应技能的人，情况是如此的。我们能不能找更有效、更好的方式来训练这方面的人才呢？

Eric最后总结道：这九个趋势展示了短时间内的大量变化。至少这些预测中对于IT产业的暗示是非常有意思的。

举个例子说，如果软件定义了基础设施，那么包括网络设备在内的硬件将会变得更商品化。如果应用程序最终可以实现开发者只写一次代码就能在多个平台运行，那么客户端的选择将会变得越来越不重要。此外，软件的交付方式永远的改变了。IBM、Oracle、SAP以及其它的供应商深深地陷入到了这种能够持续从用户身上获得大量利益的组织模式，因此它们要保持这些商品。但是这些令人兴奋的新东西呢？主要是开源的SaaS或者便宜的手机应用程序。在我看来这些产业需要重新校准他们的收入预期，虽然其中有很多“暴发户”已经获得了高额收入。

当云技术和移动世界不再有商业技术垄断之后，企业IT将如何来应对这一局面呢？许多公司逐步认识到，公司需要为用户布置各种各样的网络和移动应用程序，并观察哪一种效果最好。企业IT能够有足够的技术和技能来让这些成为现实么？还是企业将被迫转向SaaS供应商、敏捷开发公司或者其他外部供应商——然后企业IT仅仅处理一些简单的工作。

原文链接：<http://www.infoq.com/cn/news/2013/11/2014-trend>

相关内容

- [社区驱动研究：NoSQL数据库的采用趋势](#)
- [企业移动性的当前趋势](#)
- [关于Hadoop你需要知道的几件事情](#)
- [全球顶尖技术会议QCon上海2013开幕在即](#)

本期专题：内核那些事 | Topic

大数据处理之如何确保断电不丢数据

作者 [黄浩松](#)

今年7、8月份杭州实行拉闸限电时，导致阿里余杭机房的机器意外断电，造成HDFS集群上的部分数据丢失。

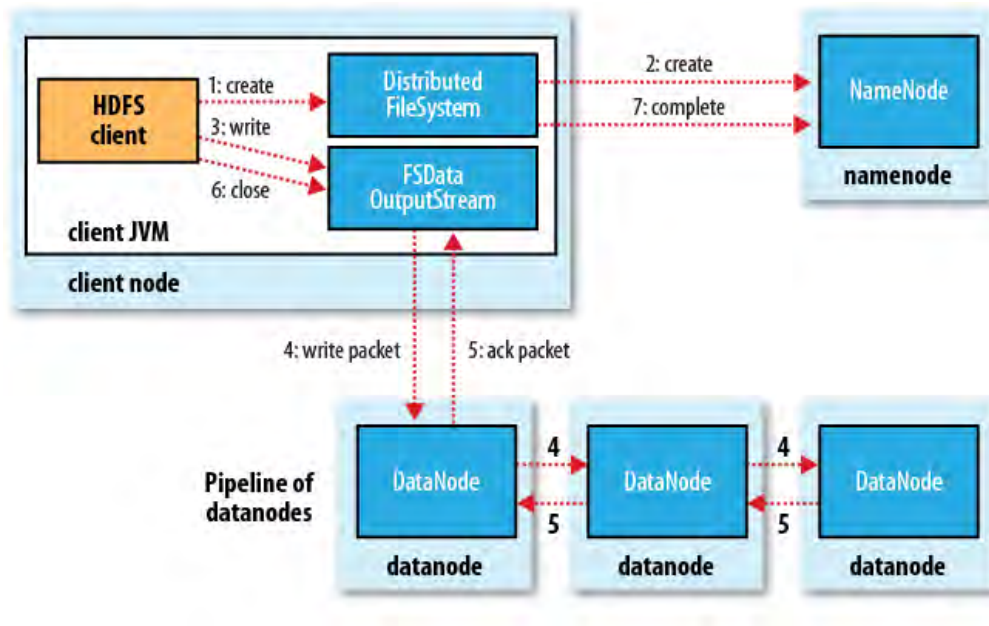
在Hadoop 2.0.2-alpha之前，HDFS在机器断电或意外崩溃的情况下，有可能出现正在写的数据丢失的问题。而最近刚发布的CDH4中HDFS在Client端提供了hsync()的方法调用([HDFS-744](#))，从而保证在机器崩溃或意外断电的情况下，数据不会丢失。这篇文章将围绕这个新的接口对其实现细节进行简单的分析，从而希望找出一种合理使用hsync()的策略，避免重要数据丢失。

HDFS中sync(), hflush()和hsync()的差别

在hsync()之前，HDFS就已经提供了sync()和hflush()的调用，单从方法的名称上看，很难分辨这三个方法之间的区别。咱们先从这几个方法之间的差别介绍起。

在HDFS中，调用hflush()会将Client端buffer中的存放数据更新到Datanode端，直到收到所有Datanode的ack响应时结束调用。这样可保证在hflush()调用结束时，所有的Client端都可以读到一致的数据。HDFS中的sync()本质也是调用hflush()。

hsync()则是除了确保会将Client端buffer中的存放数据更新到Datanode端外，还会确保Datanode端的数据更新到物理磁盘上，这样在hsync()调用结束后，即使Datanode所在的机器意外断电，数据并不会因此丢失。而hflush()在机器意外断电的情况下却有可能丢失数据，因为Client端传给Datanode的数据可能存在于Datanode的cache中，并未持久化到磁盘上。下图描述了从Client发起一次写请求后，在HDFS中的数据包传递的流程。



hsync()的实现本质

hsync()执行时，实际上会在对应Datanode的机器上产生一个fsync的系统调用，从而将内存中的相关文件的数据更新到磁盘。

Client端执行hsync时，Datanode端会识别到Client发送过来的数据包中的syncBlock_字段为true，从而判定需要将内存中的数据更新到磁盘。此时会在BlockReceiver.java的flushOrSync()中执行如下语句：

```
((FileOutputStream)cout).getChannel().force(true);
```

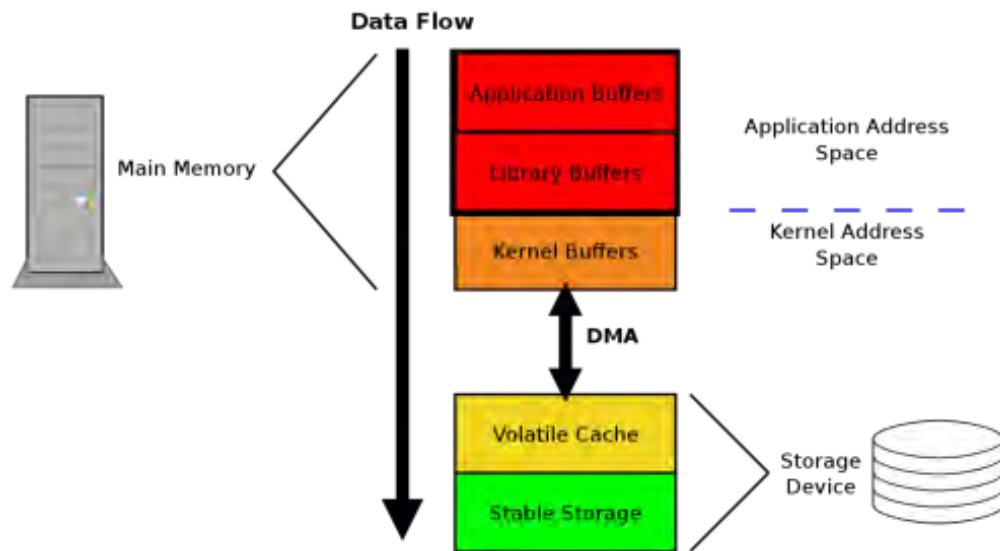
而FileChannel的force(boolean metadata)方法在JDK中，底层为于[FileDispatcherImpl.c](#)中调用fsync或fdatsync。metadata为true时执行fsync，为false时执行fdatsync。

```
Java_sun_nio_ch_FileDispatcherImpl_force0(JNIEnv *env, jobject this,
jobject fdo, jboolean md)
{
    jint fd = fdval(env, fdo);
    int result = 0;

    if (md == JNI_FALSE) {
        result = fdatsync(fd);
    } else {
        result = fsync(fd);
    }
    return handle(env, result, "Force failed");
}
```

当Datanode将数据持久化到磁盘上后，会发ack响应给Client端。当收到所有Datanode的ack响应时，hsync()的调用结束。

值得注意的是，fsync或fdatasync本身是一个非常耗时的调用，因为磁盘的读写速度远低于内存的读写速度。在不调用fsync或fdatasync的情况下，数据可能保存在各级cache中。



最开始笔者在测hsync()的读写性能时，发现不同机器上测试结果hsync()耗时差别巨大，有的集群平均调用耗时为4ms，而有的集群平均调用耗时则需25ms。后来在公司各位大神的点拨下才意识到是跟Linux文件系统的机制有关。在这种情况下，只有一探Linux相关部分的源码才能解开心中的疑惑，下面这节就将从更底层的角度来解析与hsync()密切相关的系统调用fsync及fdatasync方法。

fsync和fdatasync的大致实现过程

对ext4格式的文件系统来说，fsync和fdatasync方法的实现代码位于fs/ext4/fsync.c这个文件中。在追加写文件的情况下，fsync和fdatasync的流程几乎一致，因为对HDFS的写操作基本都是追加写，下面我们只讨论追加写文件下的情景。ext4格式的文件系统中布局大致如下：

Group 0 Padding	Super Block	Group Descriptors	Reserved GDT Blocks	Data Block Bitmap	Inode Bitmap	Inode Table	Data Blocks
1024 bytes	1 block	many blocks	many blocks	1 block	1 block	many blocks	many more blocks

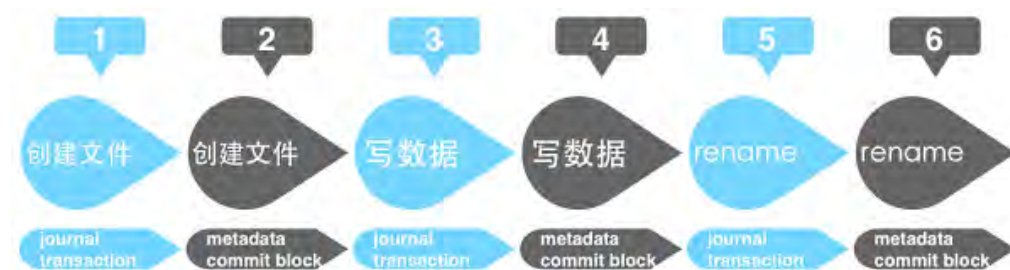
在我们追加写文件时，涉及到修改的有DataBlock BitMap、inode BitMap、i

node Table、Data Blocks。但从代码中来看，实际上对文件的追加会被合并成两次写(这里是指逻辑意义上的两次写，实际在从系统Cache刷新到磁盘时，读写操作会被再次合并)，第一次为写DataBlock和DataBlock Bitmap，第二次为写inode BitMap和更新inode BitMap中的inode。ext4为了支持更大的容量，使用了extend tree来实现块映射。在追加文件的情况下，fsync和fdatsync除了更新inode中的extend tree外，还会更新inode中文件大小，块计数这些metadata。对fsync来说，还会修改inode中的文件修改时间、文件访问时间（在mount选项不含noatime的情况下）和inode修改时间。

写障碍和Disk Cache的影响

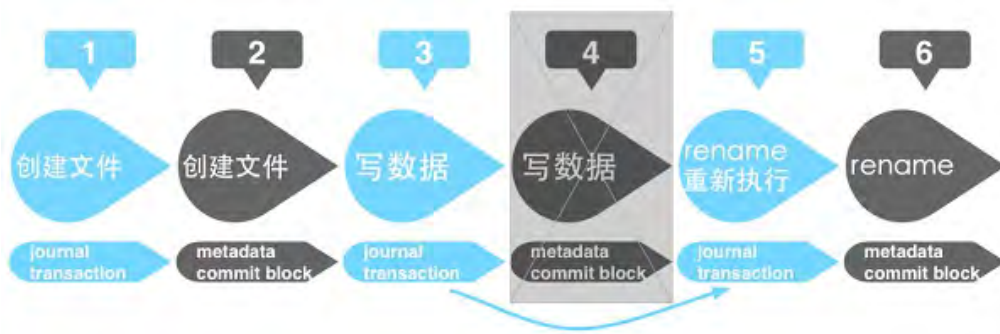
在了解了fsync()和fdatsync()方法会对文件系统进行的改动后，离找出之前为什么在不同集群上hsync()的调用平均耗时的原因仍还有一段距离。这时我发现了不同的磁盘挂载选项会影响到fsync()和fdatsync()的执行时间，进而确定是写障碍和Disk Cache在搞怪。下面这节就将分析写障碍和Disk Cache对hsync()方法调用耗时的影响。

由于市面上大部分的磁盘都是带Disk Cache的，这导致在不开启写障碍的情况下，机器意外断电可能会对其造成metadata的不一致。对ext4这种journal文件系统来说，journal写入一个事务后，会对metadata进行更新，更新完成后会将该事务标记从未执行修改为完成。举个例子，加入我们要创建并写一个文件，那么在journal中可能会产生三个事务。那么创建并写一个文件的执行流程如下：



在磁盘没有Disk Cache的情况下，即时机器意外断电，那么重启自检时，可通过journal中最后事务的状态来对metadata进行重新执行修复或者废弃该事务。从而保证了metadata的一致性。但在磁盘有Disk Cache的情况下，IO事件会当数据写到Disk Cache中就响应完成。虽然journal按上图的流程进行执行，但是执行完成后这些数据仍可能有部分并未持久化到磁盘上。假如在执行第6个步骤的时候机器意外断电，同时第4个步骤中的数据暂未更新到磁盘，而第1，2，3，5个步骤的数据已经同步到磁盘的话。这时机器重启自检时，由于第5个步骤中journal的执行状态为未完成，会重新执行第6个步骤一次。但第6个步骤对metadata的修改是建立在第4个步骤已经完成的基础之上的，由于第4个步骤并未持久化到磁盘，所以重新执行第6个步骤时会发生异常，造成metadata的错误。

。



Linux中为了避免这一情况，可以在ext4的mount选项中加barrier=1,data=ordered开启写障碍，来确保数据持久化到磁盘的顺序。在写障碍前的数据会先于写障碍后的数据刷新到磁盘，Linux会在journal的事务写到Disk Cache中后放置一个写障碍。这样journal的事务位于写障碍之前，而对应的metadata的修改数据位于写障碍之后。避免了Disk Cache中合并IO时，对读写操作进行重排序后，由于读写操作执行顺序的改变而造成意外断电后metadata无法修复的情况。

关闭写障碍，即ext4的mount选项为barrier=0时，除了有可能造成在机器断电或异常崩溃重启后metadata错误外，fsync和fdatasync的调用还会在数据更新到Disk Cache时就返回，而非等到数据刷新到磁盘上后才结束调用。因为在不开写障碍的情况下，Linux会将此时的磁盘当做没有Disk Cache的磁盘来处理，当数据只是更新到Disk Cache，就会认为该IO操作已完成，这也正是前文中提到的不同集群上hsync()的平均调用时长差别巨大的原因。所以关闭写障碍的情况下，调用fsync或fdatasync并不能确保数据在机器断电或异常崩溃时不丢失。

Disk Cache的存在可以提高磁盘每秒的吞吐量，通过重排序IO，尽量将IO读写变成顺序读写提高速率，同时减少文件系统碎片。而通过开启写障碍，可避免意外断电情形下metadata异常，同时确保调用fsync或fdatasync时Disk Cache中的数据持久到磁盘。

开启journal的影响

除了写障碍和Disk Cache会影响到hsync()的调用时长外，Datanode上文件系统有没有打开journal也是影响因素之一。关闭journal的情况下可以减少hsync()的调用时长。

在不开启journal的情况下，调用fsync或fdatasync主要是由[generic_file_fsync](#)这个方法来实现将数据刷新到磁盘。在追加写文件的情况下，不论是fsync还是fdatasync，在[generic_file_fsync](#)这个方法中都会先更新Data Block数据，

再更新inode数据。如果执行fsync或fdatasync的文件为新创建的文件，在不开启journal的情况下，还会在更新完文件的inode后，更新该文件的父结点的Data Block和inode。

而开启journal的情况下，调用fsync或fdatasync会先写Data Block，然后提交journal的事务。虽然调用fsync或fdatasync是指定对某个文件进行操作，但在ext4中，整个文件系统只有一个journal文件，提交journal的修改事务时会将整个文件系统的metadata的修改事务一并提交。在文件系统写入操作频繁时，这一步操作会比较耗时。

fsync及fdatasync耗时测试

测试使用的代码如下：

代码中以追加的方式向一个已存在的文件写入4k数据，4k刚好为内存页和磁盘块的大小。下面分别以几种模式来测试fsync和fdatasync的耗时。

```
#define BLOCK_LEN 1024

static long long microseconds(void) {
    struct timeval tv;
    long long mst;

    gettimeofday(&tv, NULL);
    mst = ((long long)tv.tv_sec) * 1000000;
    mst += tv.tv_usec;
    return mst;
}

int main(void) {
    int block = open("./block", O_WRONLY|O_APPEND, 0644);
    long long block_start, block_end, fdatasync_time, fsync_time;

    char block_buf[BLOCK_LEN];
    int i = 0;
    for(i = 0; i < BLOCK_LEN; i++){
        block_buf[i] = i % 50;
    }

    if (write(block, block_buf, BLOCK_LEN) == -1) {
        perror("write");
        exit(1);
    }
    block_start = microseconds();
    fdatasync(block);
```

```

        block_end = microseconds();
        fdatsync_time = block_end - block_start;

        if (write(block, block_buf, BLOCK_LEN) == -1) {
            perror("write");
            exit(1);
        }
        block_start = microseconds();
        fsync(block);
        block_end = microseconds();
        fsync_time = block_end - block_start;

        printf("fdatsync spent: %lld, fsync spent: %lld\n",
               fdatsync_time,
               fsync_time);

        close(block);
        exit(0);
    }

```

测试准备

- 文件系统：ext4
- 操作系统内核：Linux 2.6.18-164.el5
- 硬盘型号：WDC WD1003FBYX-1 1V02, SCSI接口
- 通过sdparm--set=WCE /dev/sdx开启Disk Write Cache, sdparm--clear=WCE /dev/sdx关闭Disk Write Cache
- 通过barrier=1,data=ordered开启写障碍, barrier=0关闭写障碍
- 通过tune4fs-O has_journal /dev/sdxx开启Journal, tune4fs-O ^has_journal /dev/sdxx关闭Journal

关闭Disk Cache, 关闭Journal

类型				耗时 (微秒)				
fdatsync				8368				
fsync				8320				
Device	wrqm/s	w/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
sdi	0.00	120.00	480.00	8.00	1.00	8.33	8.33	100.00

可以看到, iostat为8ms, 对inode、Data Block、inode Bitmap、Data Block Bitmap的数据更新合并为了一次写操作。

关闭Disk Cache, 开启Journal

类型				耗时 (微秒)				
fdatsync				33534				
fsync				33408				

Device	wrqm/s	w/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
sdi	37.00	74.00	444.00	11.95	1.22	16.15	13.32	99.90

通过使用blktrace跟踪对磁盘块的读写，发现此处写journal会比较耗时，下面的记录为fsync过程中对磁盘发送的写操作，已预处理掉了大部分不重要的信息，可以看到，后面三条记录都是journal的写操作（通过此处kjournald的进程id为3001来识别）。

0,0	13	1	0.00000000	8835	A	W	2855185 + 8 <- (8,129) 2855184
0,0	4	5	0.00031300	3001	A	W	973352281 + 8 <- (8,129) 973352280
0,0	4	1	0.00030532	3001	A	W	973352273 + 8 <- (8,129) 973352272
0,0	4	12	0.01478035	3001	A	WS	973352289 + 8 <- (8,129) 973352288

开启Disk Cache，开启写障碍，开启Journal

类型	耗时（微秒）
fdatasync	23759
fsync	25006

从结果可以看到，Disk Cache的开启可以合并更多IO，从而减少耗时。

值得注意的是，在开启Disk Cache时，iostat的await是按照从内存写完到Disk Cache中来统计耗时，并非是按照写到磁盘上来计时，所以此种情况下iostat的await参数会比较小，并无参考意义。

小结

从这次测试结果可以看到，虽然CDH4提供了hsync()方法，但是若我们对每次写操作都执行hsync()，会严重加剧磁盘的写延迟。通过一些策略，比方说定期执行hsync()或当存在于Cache中的数据达到一定数目时，执行hsync()会是更可行的方案，从而尽量减少机器意外断电所带来的影响。

附：术语解释

- Hadoop: Apache基金会的开源项目，用于海量数据存储与计算。
- CDH4: Cloudera公司在Apache社区发行版基础之上进行改进后的发行版，更稳定更适用于生产环境。
- Namenode: Hadoop的HDFS模块中管理所有文件元数据的组件。
- Datanode: Hadoop的HDFS模块中存储文件实际数据的组件。
- HDFS Client: 这里指连接HDFS对其中文件进行读写操作的客户端。

作者简介

黄浩松，华南农业大学学生，现于阿里巴巴数据平台实习。微博ID：[@华农金中菊](#)

原文链接：<http://www.infoq.com/cn/articles/large-data-processing-ensuring-data-not-lost-when-power-off>

本期专题：内核那些事 | Topic

UNIX环境高级编程：Stephen Rago访谈

作者 [Jeff Martin](#)，译者 [臧秀涛](#)

《UNIX环境高级编程》（Advanced Programming in the UNIX Environment, APUE）被誉为基于UNIX的编程环境的圣经。本书全面介绍了UNIX环境上的C语言编程，涵盖文件I/O、进程、信号、线程、进程间通信和套接字（Socket）等主题。第3版结合FreeBSD 8、Linux 3.2.0、OS X 10.6.8和Solaris 10讨论了这些概念。

APUE的[第3版](#)已于近期出版。InfoQ有机会采访了其作者Stephen Rago，谈到了这一最新版本以及UNIX开发。

InfoQ：本书第1版出版于1992年，2005年更新过一版。在第3版中，你的主要目标和动机是什么？

Stephen Rago：在第2版出版之前，我一直受困于没有足够的时间来更新某些平台相关的内容。尤其是我想用Linux 2.6代替2.4，因为2.6对pthread的支持更好一些，而且其表现与其他平台更为接近。但当时2.4仍然有很大的装机量，所以我保留了它。第3版用了我两年的时间才得以出版，因为在这期间，书中覆盖的平台频繁更新了好多次，我感觉自己一直在追赶。

InfoQ：你感觉哪部分写起来最有意思？

Rago：这就好像问我最喜欢自己的哪个孩子。从遗传学角度讲，好父母不能厚此薄彼。我也一样。相对于我的工作，在APUE第3版上的工作是一次让我耳目一新的改变。我本质上是一位C程序员和操作系统开发人员，而UNIX系统的优雅超越了其他所有操作系统，所以每部分工作我都非常喜欢。

InfoQ：与第2版相比，面向的读者有变化吗？比如说，这本书是面向职业开发人员的，还是面向学习相关编程知识的人员的，这方面是不是有所改变？

Rago：APUE最初是作为“Addison-Wesley专业计算丛书”（Addison-Wesley Professional Computing Series）的一部分出版的，所以我假定读者是职业程序员。不过从我最近收到的提问问题的电子邮件来看，很多是来自学术

界（对于赤裸裸地要答案的请求，我尽量不直接回复）。我知道一些系统编程类课程用到了这本书，我猜随着UNIX系统及其克隆产品在业务中越来越常见，越来越多的开发者都掌握了书中材料，所以本书面向的读者某种程度上也向学术界迁移了。或许也可以看做对这种改变的反映，我目前正在编写APUE第3版配套的教师手册，其中包括了书中所有习题的答案，还添加了一些书中没有的新习题。

在描述实际编程问题以及揭示很多UNIX系统接口的背景方面，这本书做得不错。所以我认为这本书可以很好地服务这两类读者。

InfoQ: 你有没有发现基于UNIX系统比较适合学术研究？这是因为其设计内在的特性，还是说只是因为相对于商业系统，其源代码很容易获得？

Rago: 这两方面的原因使基于UNIX的系统用于学术研究非常理想。其设计简洁清晰，各种实现的源代码也可以免费获得，所以我们可以看到抽象的概念是如何映射为实际实现的。你可能需要把UNIX系统包含在“商业系统”中，因为很多业务都运行在UNIX系统之上。

InfoQ: 你感觉你的书在哪种环境上更受欢迎，是UNIX、OS X还是其他系统？

Rago: 很难说。尽管我并没有一种很好的方法来衡量各种环境的受欢迎程度，但是从我收到的电子邮件来看，大部分人运行的都是某个版本的Linux。

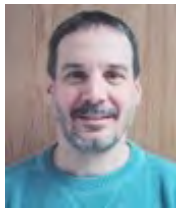
InfoQ: 考虑到C语言与UNIX的历史关系，所有例子都是用C写的。你有没有发现哪种语言能挑战C语言？C语言足够好了吗，还是说有新语言可以改进系统编程？

Rago: 40年来，C语言一直是够用的。我工作过的地方曾经把C++当做更好的C，因为它会进行更强的类型检查。但是C++如此庞大和复杂，使用起来非常困难。编写C程序时，我不需要不断地查参考书，以确定这样那样的特性应该如何使用；编写C++程序就不是这样了。我更喜欢C，因为可以用同一种语言处理高层和低层的东西，语言规范也足够简单，可以记在脑子里。操作系统需要与硬件交互，并基于高层构造提供功能，所以C语言可以很好地满足需求。使用底层操作系统所用的语言来编程会比较轻松。我还没研究过Go语言，但会找机会看一下，因为该语言尝试解决用C和C++之类的语言构建大型项目时所遇到的某些软件工程问题。

InfoQ: 在花时间写书的时候，你有没有发现UNIX有何不足？

Rago: 或许UNIX系统需要一些简单的特性使执行更安全。比如，在IPC通道（如命名管道或套接字）的一端很难获得一个进程的标识信息。但是UNIX系统起源于协同环境，所以它没有提供更多内置的认证基础设施这一点也很容易理解。但是相对于使用该系统能做的所有事情，这只是小瑕疵了。

关于APUE一书作者



Stephen A. Rago是《UNIX® System V网络编程》（Addison-Wesley, 1993）一书的作者。Rago是贝尔实验室参与构建UNIX System V Release 4的开发者之一。他曾经是《UNIX环境高级编程》第1版的技术审校者。Rago目前是NEC美国实验室存储系统组的一名研究人员。

W. Richard Stevens是公认的UNIX和网络专家，也是一位备受尊重的作家，还是广受欢迎的讲师和咨询师。他最著名的是《UNIX网络编程》、《UNIX环境高级编程》和《TCP/IP详解》等一系列书籍。

查看英文原文：[Advanced UNIX Programming: An Interview with Stephen Rago](#)

原文链接：http://www.infoq.com/cn/articles/apue_interview

相关内容

- [红帽企业版Linux新添SQL Server驱动](#)
- [AIDE 2.0引入对原生C/C++应用的支持](#)
- [今时今日，C还适合当下之所需么？](#)
- [LLVM提议向C语言中加入模块机制](#)
- [网络报文处理的两个模式](#)

本期专题：内核那些事 | Topic

Linux可插拔认证模块的基本概念与架构

作者 [王基立](#)

Linux用户认证方法简介

当今IT环境中,任何计算机系统都要充分考虑设计、使用和运行过程中的安全性。所以在目前主流操作系统的各个环节当中都增加了很多安全方面的功能和特性,而在众多的安全特性和功能中有相当多的技术是确保用户鉴别和身份认证方面的安全性的。

所谓用户鉴别,就是用户向系统以一种安全的方式提交自己的身份证明,然后由系统确认用户的身份是否属实的过程。换句话说,用户鉴别是系统的门户,每个用户进入到系统之前都必须经过鉴别这一道关。而所谓认证安全,简而言之就是计算机系统确认了用户是经过授权的合法用户之后才能允许访问。安全认证最常用的方式是对用户输入和预存于数据库中的密码。

不过在用户进行身份鉴别和安全认证的过程中,肯定会涉及几个核心问题。例如:

- 如何实现正确鉴别用户的真实身份?
- 在鉴别用户合法身份之后,如何确定用户可以对哪些资源进行访问?
- 如何控制用户以何种方式来访问计算机资源?
- 如何对用户的安全访问随时随地按需调整?

上述这些问题都是在设计鉴别和认证程序过程中需要充分考虑和精心设计的。而在Linux类的操作系统中,这些问题的处理实际上有一套完整的流程和机制。

在Linux类的操作系统中,最初用户鉴别过程就像各种Unix操作系统一样:系统管理员为用户建立一个帐号并为其指定一个口令,用户用此指定的口令登录之后重新设置自己的口令,这样用户就具有了一个只有它自己知道的口令或者密码。一般情况下,用户的身份信息在Linux系统中存放在/etc/passwd文件当中,这实际上是一个拥有简单格式的数据库表,通过": "作为分隔符分隔出多个字段,其中包括用户的名称、用户ID、组ID、用户说明、主目录和登录使用的shell等相关信息。而用户口令经过加密处理后存放于/etc/shadow 文件中。也是一个格式类似的数据库表,除了用户名和经过加密之后的密码之外,还包括多个对密码有效期进行定义的字段,包括密码有效时间、密码报警时间等。

用户登录的时候，登录服务程序提示用户输入其用户名和口令，然后将口令加密并与/etc/shadow 文件中对应帐号的加密口令进行比较，如果口令相匹配，说明用户的身份属实并允许此用户访问系统。这种思想基于只有用户自己知道它的口令，所以输入的口令是正确的话，那么系统就认定它是所声称的那个人。

在Linux类操作系统中，定义用户信息和密码信息的字段和格式都需要符合标准的Linux Naming Service Switch定义，即NSS定义。因此用户信息只要保证满足NSS规范，就可以来源于本地passwd和shadow之外的其它信息数据库和认证源。所以在此基础上还派生出一些其它认证解决方案。例如NIS、LDAP等，都可作为存放用户信息的数据库，而存放用户口令或者鉴别用户身份的数据库，可以采用专用于网络环境的Kerberos以及智能卡鉴别系统等方式。

这一整套的鉴别和认证方案貌似无懈可击，但是将这种解决方案真正应用到操作系统中的话就会发现一些问题：

第一，在操作系统上所包含的认证不仅仅只涉及到系统登录和访问，在系统外围往往提供了众多的应用程序，相当多的应用程序在访问过程中是有认证需求的。那么是否需要针对每一个应用程序都得加入认证和鉴别的功能？如果要，那么无论从程序的开发和使用管理角度来讲，工作量都将成倍增加；如果不要，则系统级的用户鉴别和安全认证与应用程序没有任何关系，意味着不管用户是否需要登录系统，但是对应用程序的访问都将缺乏最基本的安全性。

第二，如果针对每一个应用程序都开发用户鉴别和认证的功能，那么一旦发现所用的算法存在某些缺陷或想采用另一种鉴别和认证方法时，开发者或者用户都将不得不重写（修改或替换）应用程序，然后重新编译原程序。

所以，尤其是当实现鉴别功能的代码以通常方法作为应用程序一部分一起编译的时候，上述问题将十分突出。很明显，传统的身份鉴别和用户认证方式一旦整合到实际的操作系统中，在实用当中缺乏灵活性。

鉴于以上原因，Linux操作系统的开发者和设计人员开始寻找一种更佳的替代方案：一方面，将鉴别功能从应用中独立出来，单独进行模块化的设计，实现和维护；另一方面，为这些鉴别模块建立标准的应用程序接口即API，以便众多的应用程序能方便地使用它们提供的各种功能；同时，鉴别机制对上层用户（包括应用程序和最终用户）要求一定要是透明的，这样可以对使用者隐藏其中比较复杂的实现细节。

可插拔认证模块PAM的基本概念

事实上直到1995年的时候，SUN的研究人员才提出了一种满足以上需求的方案，这就是可插拔认证模块（Pluggable Authentication Module--PAM）机制

，并首次在其操作系统 Solaris 2.3上部分实现。

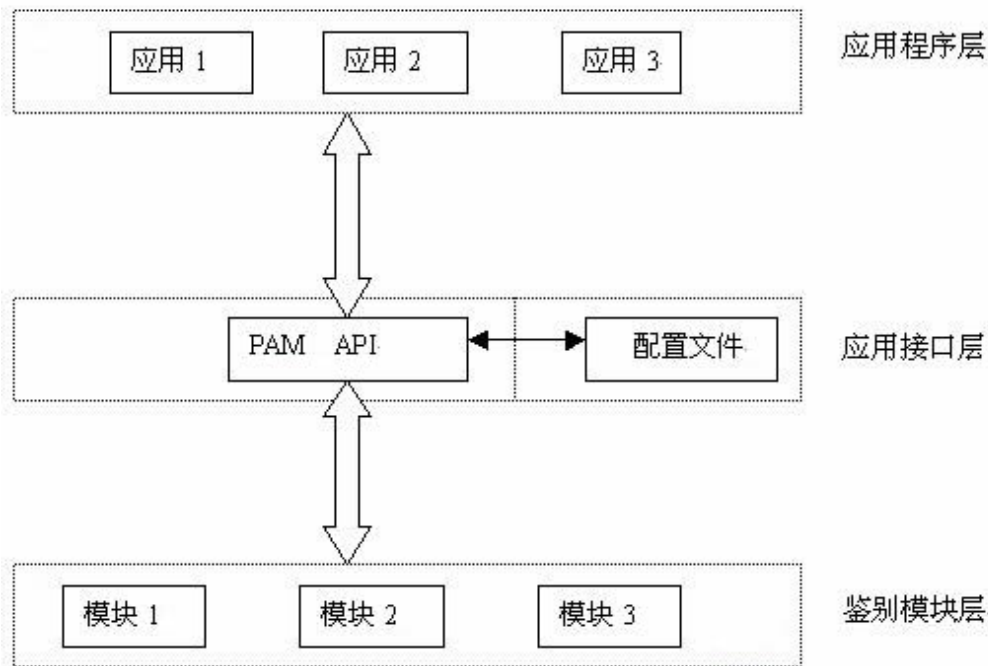
可插拔认证模块（PAM）机制采用模块化设计和插件功能，使用户可以轻易地在应用程序中插入新的认证模块或替换原先的组件，同时不必对应用程序做任何修改，从而使软件的定制、维持和升级更加轻松。因为认证和鉴别机制与应用程序之间相对独立。所以应用程序可以通过PAM API来方便地使用PAM提供的各种鉴别功能而不必了解太多的底层细节。此外PAM的易用性也较强，主要表现在它对上层屏蔽了鉴别和认证的具体细节，所以用户不必被迫学习各种各样的鉴别方式，也不必记住多个口令；又由于它实现了多鉴别认证机制的集成问题，所以单个程序可以轻易集成多种鉴别机制，如Kerberos和Diffie - Hellman等认证机制，但用户仍可以用同一个口令登录而且感觉不到采取了各种不同的鉴别方法。

在广大开发人员的努力下，各版本的UNIX系统陆续增加和提供了对PAM应用的支持。其中Linux-PAM是专门为Linux操作系统实现的，众多的Linux操作系统包括Caldera、Debian、Turbo、Red Hat、SuSE 及它们的后续版本都提供对PAM的支持。而FreeBSD从3.1版本也开始支持PAM。而且除了具体实现方法上多少有些不同外，各种版本Unix系统上PAM的框架是相同的。所以我们在这里介绍的Linux的PAM框架知识具有相当的普遍性，而且在下文介绍其框架过程中可以看到，我们并没有刻意区分Unix PAM与Linux PAM这两个技术术语。

PAM的分层体系结构

PAM 为了实现其插件功能和易用性，采取了分层设计思想。就是让各鉴别模块从应用程序中独立出来，然后通过PAM API作为两者联系的纽带，这样应用程序就可以根据需要灵活地在其中"插入"所需要的鉴别功能模块，从而真正实现了在认证和鉴别基础上的按需应变。实际上，这一思路也非常符合软件设计中的"高内聚，低耦合"这一重要思想。

PAM 的体系如下简图所示：



从上面的结构图可以看出，PAM 的API起着承上启下的作用，它是应用程序和认证鉴别模块之间联系的纽带和桥梁：当应用程序调用PAM API 时，应用接口层按照PAM配置文件的定义来加载相应的认证鉴别模块。然后把请求（即从应用程序那里得到的参数）传递给底层的认证鉴别模块，这时认证鉴别模块就可以根据要求执行具体的认证鉴别操作了。当认证鉴别模块执行完相应的操作后，再将结果返回给应用接口层，然后由接口层根据配置的具体情况将来自认证鉴别模块的应答返回给应用程序。

上面描述了PAM的各个组成部分以及整体的运作机理。下面将对PAM中的每一层分别加以介绍。

第一层：模块层。模块层处于整个PAM体系结构中的最底层，它向上为接口层提供用户认证鉴别等服务。也就是说所有具体的认证鉴别工作都是由该层的模块来完成的。对于应用程序，有些不但需要验证用户的口令，还可能要求验证用户的帐户是否已经过期。此外有些应用程序也许还会要求记录和更改当前所产生的会话类的相关信息或改变用户口令等。所以PAM在模块层除了提供鉴别模块外，同时也提供了支持帐户管理、会话管理以及口令管理功能的模块。当然，这四种模块并不是所有应用程序都必需的，而是根据需要灵活取舍。比如虽然login可能要求访问上述所有的四种模块,但是su可能仅仅需要使用到鉴别模块的功能即可。至于如何取舍则涉及到接口层的PAM API和配置文件，这部分内容将在后文中加以介绍。

第二层：应用接口层。应用接口层位于PAM结构的中间部分，它向上为应用程序屏蔽了用户鉴别等过程的具体细节，向下则调用模块层中的具体模块所提供的特定服务。由上图可以看出，它主要由PAM API和配置文件两部分组成，下面将逐

一介绍。

PAM API可以分为两类：一类是用于调用下层特定模块的接口，这类接口与底层的模块相对应，包括：

- 鉴别类接口：pam_authenticate () 用于鉴别用户身份，pam_setcred () 用于修改用户的私密信息。
- 帐号类接口：pam_acct_mgmt () 用于检查受鉴别的用户所持帐户是否有登录系统许可，以及该帐户是否已过期等。
- 会话类接口：包括用于会话管理和记帐的 pam_open_session () 和 pam_close_session () 函数。
- 口令类接口：包括用于修改用户口令的 pam_chauthtok ()。

第二类接口通常并不与底层模块一一对应，它们的作用是对底层模块提供支持以及实现应用程序与模块之间的通信等。具体如下：

- 管理性接口：每组 PAM 事务从 pam_start () 开始，结束于 pam_end () 函数。接口 pam_get_item () 和 pam_set_item () 用来读写与 PAM 事务有关的状态信息。同时，能够用 pam_strerror () 输出 PAM 接口的出错信息。
- 应用程序与模块间的通讯接口：在应用程序初始化期间，某些诸如用户名之类的数据可以通过 pam_start () 将其存放在PAM接口层中，以备将来底层模块使用。另外底层模块还可以使用 pam_putenv () 向应用程序传递特定的环境变量，然后应用程序利用pam_getenv () 和pam_getenvlist () 读取这些变量。
- 用户与模块间的通讯接口：pam_start () 函数可以通过会话式的回调函数，让底层模块通过它们读写模块相关的鉴别信息，比如以应用程序所规定的方式提示用户输入口令。
- 模块间通讯接口：尽管各模块是独立的，但是它们仍然能够通过pam_get_item () 和pam_set_item () 接口共享某些与鉴别会话有关的公用信息，诸如用户名、服务名、口令等。此外，这些API还可以用于在调用pam_start () 之后，让应用程序修改状态信息。
- 读写模块状态信息的接口：接口pam_get_data () 和pam_set_data () 用以按照PAM句柄要求访问和更新特定模块的信息。此外，还可以在这些模块后附加一个清除数据函数，以便当调用 pam_end () 时清除现场。

由于 PAM 模块按需加载,所以各模块始化任务在第一次调用时完成。如果某些模块的清除任务必须在鉴别会话结束时完成，则它们应该使用 pam_set_data () 规定清除函数，这些执行清除任务的函数将在应用程序调用 pam_end () 接口时被调用。

以上介绍了Linux可插拔认证模块PAM的基本概念和分层体系结构，在后续文章里，将会介绍常见的PAM模块应用以及相关实例。

关于作者

王基立 (Jerrywjl)，红帽软件 (北京) 有限公司资深解决方案架构师，熟悉红帽所有平台类产品和解决方案，拥有多年的售前架构规划与售后技术支持经验。现主要负责华为、中兴等大型电信企业以及金融、政府、教育等行业客户在生产环境中的咨询、培训、现场实施和技术支持等工作。

原文链接: <http://www.infoq.com/cn/articles/wjl-linux-pluggable-authentication-module>

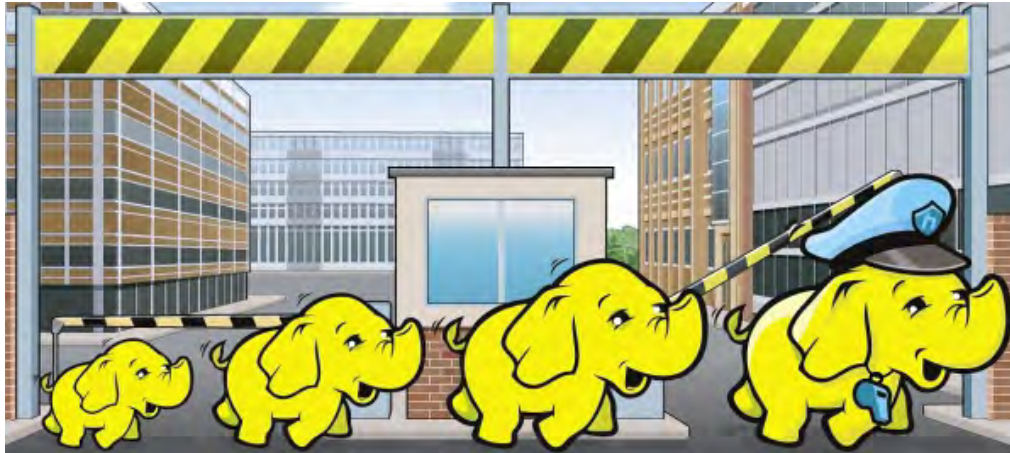
相关内容

- [Linux Container概述，以及其存在的问题和挑战](#)
- [用更好的工具提升Linux开发](#)
- [访问及书评:DevOps中Linux服务器疑难追踪最佳实践](#)
- [微软为Windows Azure 创建基于Linux的虚拟机的目录](#)
- [Linux现可运行于Windows Azure上](#)

推荐文章 | Article

大数据安全：Hadoop安全模型的演进

作者 [Kevin T. Smith](#)，译者 [吴海星](#)



敏感信息的安全和保护是当今人们最关心的问题之一。进入大数据时代，很多组织都在从各种源头收集数据，进行分析，并基于对海量数据集的分析做出决策，因此这一过程中的安全问题变得愈发重要。与此同时，HIPAA和其他隐私保护法之类的法律法规也要求组织加强对这些数据集的访问控制和隐私限制。来自内部和外部攻击者的网络安全漏洞与日俱增，通常都要数月之后才能发现，而那些受此影响的人正在为此付出代价。没能对他们的数据做出恰当访问控制的组织将受到起诉，出现在负面报道中，并将面临监管机构的罚款。

请想一想下面这些让人大开眼界的统计数据：

- 赛门铁克和Ponemon研究所今年公布的一项研究表明，一个安全漏洞在美国的平均组织化成本是540万美元¹。另据[最近一项研究](#)表明，仅仅网络犯罪在美国造成的损失每年就有140亿美元之多。
- 2011年索尼游戏机网络中出现的漏洞可以算是近代最大的安全漏洞之一，专家们估计索尼与该漏洞相关的损失大约在27亿到240亿美元之间（范围很大，但这个漏洞太大了，所以几乎难以对其进行量化）。²
- Netflix和AOL已经因为其管理的大量数据和对个人信息的保护而受到金额达数百万美元的起诉（某些已经立案），尽管他们已经对这些数据做了“匿名化”处理并且是为了研究才公布的。³
- 跟安全漏洞相关的除了可量化的成本（客户和业务合作伙伴的损失，诉讼，监管罚款），经历此类事件的组织的可信度和声誉还会受到影响，甚至可能会导致公司歇业。⁴

简而言之，如果没有恰当的安全控制，大数据很容易变成花费巨大的大问题。

对于处理大数据的组织来说这意味着什么？意味着你拥有的数据越多，对数据的保护就越重要。意味着不仅要安全有效地控制离开自有网络的数据，还必须做好网络内部的数据访问控制。依据数据的敏感程度，我们可能要确保数据分析师能看到的数据是可以让他们分析的数据，并且必须明白发布这些数据及其分析结果可能产生的后果。仅[Netflix数据泄漏](#)一个案例就足以表明，即使已经试图对数据做了“匿名化”处理，也可能会发布一些意料之外的信息——一些在[差异化隐私](#)领域标明的东西。

Apache Hadoop是最流行的大数据处理平台之一。尽管最初设计Hadoop时根本没考虑安全问题，但它的安全模型在不断地演进。Hadoop的兴起也招致了很多批判，并且随着安全专家不断指出其潜在的安全漏洞及大数据的安全风险，使得Hadoop一直在改进其安全性。“Hadoop安全”市场曾出现过爆炸性的增长，很多厂商都发布了“安全加强”版的Hadoop和对Hadoop的安全加以补充的解决方案。这类产品有[Cloudera Sentry](#)、[IBM InfoSphere Optim Data Masking](#)、[英特尔的安全版Hadoop](#)、[DataStax企业版](#)、[DataGuise for Hadoop](#)、[用于Hadoop的Protegrity大数据保护器](#)、[Revelytix Loom](#)、[Zettaset 安全数据仓库](#)，此外还有很多，这里就不再一一列举了。与此同时，Apache也有[Apache Accumulo](#)这样的项目，为使用Hadoop提供了添加额外安全措施机制。最终还出现了[Knox网关](#)（由HortonWorks贡献）和[Rhino项目](#)（由英特尔贡献）这样的开源项目，承诺要让Hadoop本身发生重大改变。

要让Hadoop达到安全性要求的巨大需求使得Hadoop一直在发生着变化，这也是我要在本文中重点讨论的内容。

Hadoop安全（简）史

Doug Cutting和Mike Cafarella最初为Nutch项目开发Hadoop时并没有考虑安全因素，这是众所周知的事实。因为Hadoop的最初用例都是围绕着如何管理大量的公共web数据，无需考虑保密性。按照Hadoop最初的设想，它假定集群总是处于可信的环境中，由可信用户使用的相互协作的可信计算机组成。

最初的Hadoop中并没有安全模型，它不对用户或服务进行验证，也没有数据隐私。因为Hadoop被设计成在分布式的设备集群上执行代码，任何人都能提交代码并得到执行。尽管在较早的版本中实现了审计和授权控制（HDFS文件许可），然而这种访问控制很容易避开，因为任何用户只需要做一个命令行切换就可以模拟成其他任何用户。这种模拟行为非常普遍，大多数用户都会这么干，所以这一已有的安全控制其实没起到什么作用。

在当时，考虑到安全问题的组织把Hadoop隔离在专有网络中，只有经过授权的用户才能访问。然而由于Hadoop内部几乎没有安全控制，在这样的环境中也会

出现很多意外和安全事故。善意的用户可能会犯错（比如用一个分布式删除在几秒内就会删除大量数据）。所有用户和程序员对集群内的所有数据都有相同的访问权限，所有任务都能访问集群内的任何数据，并且所有用户都可能会去读取任何数据集。因为MapReduce没有认证或授权的概念，某个顽劣的用户可能为了让自己的任务更快完成而降低其他Hadoop任务的优先级，甚至更坏，直接杀掉其他任务。

随着Hadoop在数据分析和处理平台中的地位日益凸显，安全专家们开始关心来自Hadoop集群内部的恶意用户的威胁。恶意开发人员能轻易写出假冒其他用户Hadoop服务的代码来（比如写一个新的TaskTracker并将其注册为Hadoop服务，或者冒充hdfs或mapred用户，把HDFS里的东西全删掉等等）。因为DataNode没有访问控制，恶意用户可以绕过访问控制从DataNode中读取任意数据块，或将垃圾数据写到DataNode中破坏目标分析数据的完整性。所有人都能向JobTracker提交任务，并可以任意执行。

因为这些安全问题，Hadoop社区意识到他们需要更加健壮的安全控制，因此，雅虎的一个团队决定重点解决认证问题，选择Kerberos作为Hadoop的认证机制，这在他们2009年的[白皮书](#)上有记录。

在Hadoop发布2.0.20x版本时他们实现了自己的目标，该版本采用了下面这些机制：

- **用Kerberos RPC (SASL/GSSAPI) 在RPC连接上做相互认证**——用SASL/GSSAPI来实现Kerberos及RPC连接上的用户、进程及Hadoop服务的相互认证。
- **为HTTP Web控制台提供“即插即用”的认证**——也就是说web应用和web控制台的实现者可以为HTTP连接实现自己的认证机制。包括（但不限于）HTTP SPNEGO认证。
- **强制执行HDFS的文件许可**——可以通过NameNode根据文件许可（用户及组的访问控制列表（ACLs））强制执行对HDFS中文件的访问控制。
- **用于后续认证检查的代理令牌**——为了降低性能开销和Kerberos KDC上的负载，可以在各种客户端和服务经过初始的用户认证后使用代理令牌。具体来说，代理令牌用于跟NameNode之间的通讯，在无需Kerberos服务器参与的情况下完成后续的认证后访问。
- **用于数据块访问控制的块访问令牌**——当需要访问数据块时，NameNode会根据HDFS的文件许可做出访问控制决策，并发出一个块访问令牌（用HMAC-SHA1），可以把这个令牌交给DataNode用于块访问请求。因为DataNode没有文件或访问许可的概念，所以必须在HDFS许可和数据块的访问之间建立对接。
- **用作业令牌强制任务授权**——作业令牌是由JobTracker创建的，传给TaskTracker，确保Task只能做交给他们去做的作业。也可以把Task配置成当用户提交作业时才运行，简化访问控制检查。

- 把这些整合到一起让Hadoop向前迈出了一大步。自那之后，又实现了一些值得称道的修改：
- 从“即插即用的认证”到**HTTP SPNEGO认证**——尽管2009年的Hadoop安全设计重点是即插即用的认证，但因为RPC连接（用户、应用和Hadoop服务）已经采用了Kerberos认证，所以Hadoop开发者社区觉得如果能跟Kerberos保持一致更好。现在Hadoop web控制台被配置成使用HTTP SPNEGO这一用于web控制台的Kerberos实现。这样可以部分满足Hadoop亟需的一致性。
- **网络加密**——采用了SASL的连接可以配置成使用机密保护质量（QoP），在网络层强制加密，包括使用Kerberos RPC的连接和使用代理令牌的后续认证。Web控制台和MapReduce随机操作可以配置成使用SSL进行加密。HDFS文件传输器也能配置为加密的。

自对安全性进行重新设计以来，Hadoop的安全模型大体上没发生什么变化。随着时间的推移，Hadoop体系中的一些组件在Hadoop之上构建了自己的安全层，比如Apache Accumulo，提供单元级的授权，而HBase提供列和族系一级的访问控制。

Hadoop当前所面临的安全挑战

组织在保证Hadoop的安全性时会面临一些安全方面的挑战，在我和Boris Lubli nsky 及 Alexey Yakubovich写的新书中，我们用了两章的篇幅集中讨论Hadoop的安全问题，其中一章的重点是Hadoop本身的安全能力，另外一章的重点是对Hadoop的安全性进行补充的策略。

常见的安全问题有：

- 如何强制所有类型的客户端（比如web控制台和进程）上的用户及应用进行验证？
- 如何确保服务不是流氓服务冒充的（比如流氓TaskTracker和Task，未经授权的进程向 DataNode 出示ID 以访问数据块等？）
- 如何根据已有的访问控制策略和用户凭据强制数据的访问控制？
- 如何实现基于属性的访问控制（ABAC）或基于角色的访问控制（RBAC）？
- 怎么才能将Hadoop跟已有的企业安全服务集成到一起？
- 如何控制谁被授权可以访问、修改和停止MapReduce作业？
- 怎么才能加密传输中的数据？
- 如何加密静态数据？
- 如何对事件进行跟踪和审计，如何跟踪数据的出处？
- 对于架设在网络上的Hadoop集群，通过网络途径保护它的最好办法是什么？

这其中很多问题都能靠Hadoop自身的能力解决，但也有很多是Hadoop所无能为力的，所以行业内涌现出了很多Hadoop安全补充工具。厂商们发布安全产品来弥补Hadoop的不足有几个原因：

1. 没有“静态数据”加密。目前HDFS上的静态数据没有加密。那些对Hadoop集群中的数据加密有严格安全要求的组织，被迫使用第三方工具实现HDFS硬盘层面的加密，或安全性经过加强的Hadoop版本（比如今年早些时候英特尔发布的版本）。
2. 以 **Kerberos**为中心的方式——Hadoop依靠 Kerberos做认证。对于采用了其他方式的组织而言，这意味着他们要单独搭建一套认证系统。
3. 有限的授权能力——尽管Hadoop能基于用户及群组许可和访问控制列表（ACL）进行授权，但对于有些组织来说这样是不够的。很多组织基于XACML和基于属性的访问控制使用灵活动态的访问控制策略。尽管肯定可以用Accumulo执行这些层面的授权过滤器，但Hadoop的授权凭证作用是有限的。
4. 安全模型和配置的复杂性。Hadoop的认证有几个相关的数据流，用于应用程序和Hadoop服务的Kerberos RPC认证，用于web控制台的HTTP SPNEGO认证，以及使用代理令牌、块令牌、作业令牌。对于网络加密，也必须配置三种加密机制，用于SASL机制的保护质量，用于web控制台的SSL，HDFS数据传输加密。所有这些设置都要分别进行配置，并且很容易出错。

如果Hadoop如今还不具备实现者所要求的安全能力，那么他们只能转而集成第三方工具，或使用某个厂商提供的安全加强版Hadoop，或采用其他有创造性的办法。

即将发生的大变化

2013年开端之际，英特尔发起了一个开源项目[Rhino](#)，以提升Hadoop及其整个体系的安全能力，并将代码贡献给了Apache。这有望显著加强Hadoop当前的贡献。这一开源项目的总体目标是要支持加密和密钥管理，一个超越Hadoop当前提供的用户及群组ACL的通用授权框架，一个基于认证框架的通用令牌，改善HBase的安全性，改善安全审计。这些任务都被记录在Hadoop、MapReduce、HBase 和 Zookeeper的JIRA中，择重点摘录如下：

- 加密的静态数据——JIRA 任务 [HADOOP-9331](#) (Hadoop加密编码解码器框架及加密编码解码器的实现) 和 [MAPREDUCE-5025](#) (支持MapReduce中的加密编码解码器的密钥发行和管理) 有直接的关系。第一个侧重于创建一个加密框架及其实现，以支持对HDFS上文件的加密和解密；第二个侧重于为MapReduce提供密钥发行和管理框架，以便能在MapReduce操作过程中对数据加密和解密。为此向Hadoop中引入了一个可分割AES编码解码器的实现，可以对磁盘上分散的数据加密和解密。密钥发行和管理框架可以在MapReduce操作过程中解析密钥的上下文，因此MapReduce作业能够进行加解密操作。他们已经发展出的需求包括MapReduce作业不同阶段的不同选项，并且要支持灵活的密钥获取办法。在一些相关的任务中，[ZOOKEEPER-1688](#) 将提供透明的快照加密能力，并在硬盘记录日志，防止敏感信息从静态文件中泄漏出去。
- 基于令牌的认证及统一授权框架——JIRA 任务 [HADOOP-9392](#) (基于令牌的认

证及单点登录) 和 [HADOOP-9466](#) (统一授权框架) 也是相互关联的。第一项任务展示了一个跟Kerberos耦合不是那么紧密的基于令牌的认证框架。第二项任务会用基于令牌的框架支持灵活的授权强制引擎, 以取代 (但能向后兼容) 当前的ACL式访问控制。对基于令牌的认证框架, 第一项任务计划支持多种认证机制的令牌, 比如LDAP 用户名/密码认证, Kerberos, X.509证书认证, SQL认证 (基于SQL数据库的用户名/密码认证) 和SAML。第二项任务要支持一个先进的授权模型, 侧重于基于属性的访问控制 (ABAC) 和XACML标准。

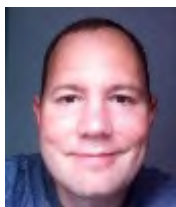
- **提升HBase的安全性**——JIRA 任务 [HBASE-6222](#) (增加每-键值安全) 向HBase添加Apache Accumulo具备但HBase还没有的单元级授权。开发出构建在加密框架上的[HBASE-7544](#), 把它扩展到HBase, 提供透明的表加密。

这些就是Hadoop的主要变化, 但有望解决有这些安全需求的组织的安全问题。

结论

在我们这个步履匆匆而又相互关联的世界里, 大数据就是王道, 在我们对海量数据进行处理和分析时, 明白安全的重要性至关重要。这要从弄懂数据及相关的安全策略开始, 也要明白组织的安全策略, 并知道如何强制执行。本文介绍了Hadoop的安全简史, 重点讲了常见的安全问题, 并介绍了Rhino项目, 给出了一个未来的快照。

关于作者



凯文T.史密斯是Novetta解决方案应用任务方案分部的技术方案及推广指导, 他负责向客户提供战略性的技术领导力, 开发具有创新性的、数据为本并且高度安全的解决方案。他经常在各种技术会议上演讲, 发表过很多技术文章, 还编写过许多技术书籍, 包括即将出版的《专业Hadoop解决方案》, 以及《应用SOA: 面向服务的架构及设计策略》, 《语义Web: XML, Web服务及知识管理的未来发展指南》等等。可以通过KSmith@Novetta.com联系到他。

致谢

特别感谢Stella Aquilina, Boris Lublinsky, Joe Pantella, Ralph Perko, Praveena Raavicharla, Frank Tyler 和 Brian Uri 对本文的审阅和部分内容的评论。此外还要感谢克里斯·贝利制作了不断发展的Hadoop大象之“艾比路”这幅插图。

¹ Ponemon 研究所, 2013数据泄露的成本研究: 全球分析, 2013年5月

² 商业内幕, PlayStation网络危机可能让索尼花费了数十亿

³ 请参见“CNN/Money -5数据泄露 - 从尴尬到致命”,及维基百科上关于 AOL在匿名化记录上泄漏的研究数据的页面

⁴ Ponemon 研究所, “你的公司为大数据泄漏做好准备了吗?”, 2013年3月

原文链接: <http://www.infoq.com/cn/articles/HadoopSecurityModel>

相关内容

- [OWASP发布Web应用程序的十大安全风险](#)
- [Oracle发布了51个Java安全补丁](#)
- [JRuby核心开发者Ola Bini谈语言的设计, 多语言编程与信息安全](#)
- [应用安全测试: 双面的黑盒](#)
- [云计算时代的运维与安全](#)

推荐文章 | Article

如何让你的内存NoSQL数据库为企业应用做好准备

作者 [Yiftach Schoolman](#)，译者 [邵思华](#)

对于每一个关注于用户体验的web与移动应用程序来说，诸如开源的[Redis](#)与[Memcached](#)等基于内存的NoSQL存储系统正在成为事实上的标准。但是，近几年间，大型企业对于这些数据库的使用仍发展缓慢，其原因主要归结于性能、可伸缩性及可用性等方面的挑战。

幸运的是，现代编程语言（Ruby，Node.js和Python等）以及开发平台（Rails，Sinatra和Django等）已经直接创建好了一系列的工具和类库，它们能够充分利用基于内存的数据存储系统（Redis体现得尤为明显）的高性能以及各种操作命令类型，实现了一系列常见的用例。

这些开源软件项目的用例包括了任务管理、论坛、实时分析、twitter clone，地理位置搜索及缓存高级应用。

不过，对于每一个应用程序来说，数据库的可用性、可伸缩性以及性能对于整个应用的成败都有着莫大的影响。

本文概括介绍了为了将你的内存NoSQL数据库为企业应用做好准备所需的各种知识，以及在云端管理这些数据库时，如何克服七项最大挑战的提示与建议。

1.可用性

无论你进行何种操作，你的数据集对应用程序来说应该是始终可用的。这对内存数据库来说尤其重要，因为如果没有应用正确的策略，当以下状况发生时，你就会丢失你的数据集的部分或全部内容：

1. 节点失败（这点在云端尤其常见）。
2. 进程重启（你或许时常需要进行重启）。
3. 系统扩展（希望你会需要到它）。

对于状况1与状况2的情形（本文稍后会讨论状况3的内容），你必须应用两种主要的机制：

- **分发（Replication）**：你至少必须确保在另一个云端实例中托管一份数据集的拷贝，而如果你要确保在整个数据中心故障发生时（2012年间，Amazon W

eb Services就至少发生了四次这样的情况) 仍能够保护你的数据, 你最好在一个不同的数据中心保存一份拷贝。遗憾的是做到这一点并不容易。以下这种场景就是在分发时所面临的一项挑战:

- 当你的应用程序中直接写入磁盘的操作大量增长时, 你将会发现你的应用服务器的写入速度要高于分发的速度, 尤其是你的主结点和分发结点出现网络拥堵时, 这种问题会更加严重。一旦这种问题开始产生, 如果你的数据集非常大的话, 那么你的分发结点有很大的可能性完全停止同步操作。
- **自动故障转移 (Auto failover)**: 为什么需要自动故障转移? 这是因为你的内存数据库通常每秒处理的请求数量达到其它数据库的100倍之多, 因此每一秒的停机都意味着你的应用堆积了更多的处理延迟, 因此导致了糟糕的用户体验。在实现你自己的自动故障转移机制时, 请按照以下列出的建议来做:
 - 请确保你的主结点一旦故障时, 分发结点能够立刻进行故障转移, 这一点应该基于一个健壮的看门狗机制, 它持续地监控你的各个结点, 并在发生故障时自动转移至某个情况最良好的结点上。
 - 这一过程应该尽量对你的应用程序保持透明, 理想的情况下应该不需要任何配置的变更。最高级的解决方案是修改DNS中数据存储节点的IP地址, 这可以确保你的恢复过程只需几秒钟就可以完成。
 - 你的自动故障转移应该基于请求集 (Quorum), 并且实现完全一致性或者最终一致性。关于这一点的更多信息请见下文。

2. 网络分裂期间与之后的一致性

网络分裂 (network splits) 在云端频繁发生, 它或许是世界上任何一个分布式数据库系统中最复杂的部分。一旦发生分裂, 你的应用程序也许只能见到你的全部内存NoSQL结点中的一部分, 并且你的任意一个内存NoSQL结点也只能见到其它内存NoSQL结点的一部分。

为什么说这是一个很大的问题呢? 如果你的数据库隐含着某些方面的设计缺陷的话, 那么当网络分裂发生时, 你或许你发现你的应用程序将数据写入了错误的结点。这就意味着, 一旦分裂状况恢复时, 在此阶段你的应用程序所发出的数据写入请求都将消失。这对于内存NoSQL数据库来说是个极大的问题, 因为它每秒钟所产生的“写”操作远远大于其它任何NoSQL数据库系统。

那么如果你的内存NoSQL数据库设计正确呢? 很不幸, 你将不得不在两种非常糟糕的替代方案 (实际上是一种.....) 中进行选择, 如下所示:

1. 如果你的内存NoSQL数据库是完全一致的, 你需要了解的是, 在某些情况下它将不允许你写入任何数据, 直到网络分裂恢复为止。
2. 如果你的内存NoSQL数据库是最终一致的, 你的应用程序大概在发送读请求时会使用一个请求集向量, 它或者返回一个值 (基于请求集), 或者被阻塞 (等待请求集)。

请注意：由于目前市面上还没有任何一种最终一致的内存NoSQL数据库存在，实际上你只能选择第1个选项。

3.数据持久化

即使你的内存NoSQL解决方案允许多种分发方案，你仍应该考虑数据持久化与备份问题，出于以下几个原因：

- 或许你不愿意为内存分发投入更多的资源，但仍然希望在结点故障发生时能够确保能够在某处保留你的数据集，并能够通过它进行故障恢复（即使恢复速度缓慢）。
- 假设你希望能够从某种故障情况（例如节点故障、多节点故障、数据中心故障等等）进行恢复，并且可以在某个安全所在保留着你的数据集的备份，即使它并不包含你最新的某些变更记录。
- 还有其它诸多原因促使你使用数据持久化，例如将你在生产环境中的数据导入到预发布环境中，以满足调试的需要。

希望我已经使你了解到数据持久化是必要的，在多数云端环境中，你应该为你的云端实例附加一个存储设备（例如AWS的EBS和Azure的Cloud Drive等等），如果你依然使用本地磁盘保存数据，那么下次节点故障时数据就会丢失。

一旦你启用了数据持久化机制之后，你的头号挑战就是如何在实时地将数据写入你的持久化存储介质时，保持你的内存NoSQL数据库依然高速运作。

4.稳定的性能

诸如Redis与Memcached等内存NoSQL数据库在设计时的指标是：每秒能够处理超过10万次请求，并保证延迟小于毫秒级。但如果你不按照以下方法去做的话，你在云端环境中的速度是达不到这些值的：

- 确保为你的解决方案选择性能最强大的云端实例（例如AWS的[m2.2*large/m2.4*large](#)实例或者Azure的A6/A7实例），并且将它们保留为专用的环境。作为替代，你也可以实现某种机制，只要该机制能够阻止不同的云端帐号间发生相互影响。这种机制应该基于某种标准对你的数据集性能进行实时监控，并且覆盖每个命令。该机制还需要同时应用一系列其它机制，举例来说，当它发现延迟已超过某个阈值时，能够自动将数据集迁移至某个其它节点。
- 为了避免存储介质的I/O瓶颈，请确保为你的解决方案选择一个强大的持久化存储设备，最好是配置了RAID。随后要确保你的解决方案在请求数量突然爆发时不会阻塞你的应用程序。举例来说，在开源的Redis中，你可以配置slave节点，让它将数据写入到某个持久化存储设备中，而让主服务器处理你的应用程序的请求，以避免峰值时的请求超时。
- 对于云提供商所建议的存储I/O优化手段，例如[AWS的PIOPS](#)进行全面测试。大

多数情况下，这些方案在随机访问（读/写）时具有良好的表现，但在顺序写操作的场景下，例如那些内存NoSQL数据库系统中所使用的方式，这种方案比起标准的存储配置并没有带来额外的优势。

- 如果你的内存数据库像Redis一样，是基于某种单线程架构的，请确保不要在一个单线程进程中运行多个数据库。这种配置会潜在性地产生某种阻塞式的场景，即某个数据库会阻塞另外的数据库执行命令。

5.网速

多数云端实例都配置了一块独立的1G网卡。在内存NoSQL数据库的情况下，这1G需要处理以下内容：

1. 应用程序请求
2. 集群内部通信
3. 分发
4. 存储介质访问

这1G流量会很容易成为各种操作的瓶颈，以下是解决该问题的一些建议：

- 使用10G流量的云端实例（但请做好准备，它们可是相当昂贵的）。
- 选择能够在某些特殊配置的情况下（例如在VPC中）提供多块1G网卡的云服务，例如AWS。
- 建立一种能够有效地在多个内存NoSQL节点之间分配资源的解决方案，将网络阻塞降至最低。

6.可伸缩性

对于简单的键/值缓存解决方案来说（例如Memcached或者Redis的简单应用），一般而言都不会把扩展当作一个大问题，因为在多数情况下，只需将一台服务器加入服务器列表（或从列表中移除），并修改哈希方法即可。不过，富有经验的用户会意识到，扩展仍然可能成为一项令人头疼的任务。以下是处理此问题的一些建议：

1. 使用一致性哈希算法。使用像modulo这样的简单哈希算法会导致在扩展时丢失全部的key。另一方面，多数使用者并未察觉到，即使使用一致性哈希算法，在扩展时仍然会丢失部分数据。举例来说，在横向扩展时你就会丢失 $1/N$ 的key，其中N代表扩展后的节点数目。因此如果N的数目较小的话，这一过程仍旧是令人头痛的（例如你使用一致性哈希算法对一个包含2个节点的集群进行横向扩展，那么将意味着扩展后整个数据集的 $1/3$ 将会丢失）。
2. 创建一项机制，在扩展发生时对你所有的内存NoSQL客户端进行同步，以避免在扩展过程中不同的应用服务器对不同的节点进行写操作。

在处理复杂的命令，例如Redis的[UNION](#)或者[INTERSECT](#)时，扩展会成为一个真正的难题。这些命令的作用相当于SQL语句中的JOIN命令，在对一个多分片（multi-shard）的架构进行操作时，必然需要加入一定量的延迟以及复杂性。如果在应用程序级别进行分片则能够部分解决此问题，因为它允许你在分片级别运行一些复杂的命令。但这意味着，你的应用程序设计会与内存NoSQL节点的配置紧密相关，使整个设计变得非常复杂。比方说，支持分片的应用程序必需了解每个键存储在哪个节点上。并且像重新分片等扩展事件将导致大量的代码变更，并消耗运维部门的大量精力。

作为替代方案，有些用户声称新一代的超高性能RAM，例如AWS的High Memory Cluster Eight Extra Large 244GB内存（[cr1.8*large](#)）能够通过纵向扩展实例的方式解决多数复杂数据类型的扩展问题。但现实稍有不同，因为像Redis这样的内存NoSQL数据库，当它的数据集大小达到了25GB到30GB的规模后，会有许多其它操作上的难题出现，它们会使你执行纵向扩展的计划受阻。这些难题与本文之前描述的诸多挑战密切相关，例如分发、存储介质I/O、单核的单线程架构，网络开销等等。

7. 运维团队的巨大开销

处理内存NoSQL数据库的各种运维操作会消耗巨大的精力。它需要你对这些技术的所有细节都有深入的了解，以保证在各种紧要关头作出正确的决定。它同时要求你紧跟这些系统的最新变化与发展趋势，因为技术的变化是非常频繁的（或许太频繁了些）。

结论

正如我以上所阐述的一样，为了充分利用Redis和Memcached这些开源技术的各种优点，充分了解它们的各种问题也是非常关键的。对企业级IT团队来说，了解如何在企业级环境下以最佳的方式克服这些挑战，以最大程度发挥内存NoSQL数据库的作用，这一点是尤其重要的。我对开源项目并没有偏见，但我依然建议寻求一些能够克服可伸缩性及高可用性的种种限制，同时保证在功能与性能方面不会作出任何妥协的商业解决方案。因为执行内存NoSQL数据库运维操作需要尖端的领域专家，而这种专家是数量很少的。

当前市面上已经有一些基于Redis和Memcached的内存NoSQL即服务（NoSQL-as-a-service）解决方案存在了。我建议你对这些服务以及自己动手打造解决方案的方式做一个全面的对比，然后再决定对你的应用程序来说，怎样克服这些在云端管理内存NoSQL时所面临的挑战才是最佳的方式。对你心仪的解决方案最好能建立一些基于真实项目的经验，这也是为什么很多服务商会提供一个免费试用阶段的原因之一。

关于作者



Yiftach Schoolman是Garantia Data的联合创始人之一，并担任CTO，他是一位经验丰富的技术专家，在多个领域担任过软件开发与产品设计的领导者，包括了应用程序加速、云计算、软件即服务（SaaS）、Broadband Networks与Metro Networks。Yiftach也是Crescendo Networks公司（后被F5 – NASDAQ代码FFIV收购）的创始人、主席以及CTO，Native Networks公司（后被Alcatel – NASDAQ代码ALU收购）的VP软件工程师，并且是ECI电信宽带业务部门的创始团队成员之一，担任VP软件工程师。Yiftach拥有数学及计算机科学专业的学士学位，并且修完了Tel-Aviv大学的计算机科学专业的硕士学位。

原文链接：<http://www.infoq.com/cn/articles/make-imdg-enterprise-ready>

相关内容

- [NoSQL数据库面面观](#)
- [Peter Bell谈NoSQL的发展趋势](#)
- [NoSQL更适合担当云数据库吗](#)
- [如何使用多记录类型为NoSQL类型数据提供SQL访问](#)
- [NoSQL精粹](#)

Software AG webMethods BPMS 产品白皮书

作者：张去疾

版本 1.0 | 2013 年 9 月

推荐语

敏捷与结构性模块化

敏捷开发方法论日益流行，然而大多数“敏捷”专家和分析师都在孤立地讨论敏捷，也就是说忽视了系统“结构”（Kirk Knoernschild是一个例外，他编写了一本名为《Java Application Architecture》的图书阐述这一理念）。考虑到“敏捷”是基础实体的一个重要特性或属性，那么，这种疏忽令人感到很惊讶。一个实体要具有“敏捷”的特性，它必须具有高度的结构性模块化（structural modularity）特征（参见Scott Page的《Diversity & Complexity》）。也许正因为这种疏忽，许多组织在敏捷开发流程方面进行投入但忽略了应用程序的结构。除了“如何实现一个敏捷的系统？”这个问题以外，有人肯定还会问，“如何构建一个在结构上具备高度模块化的系统？”

这个系列的文章将从探讨结构性模块化和敏捷之间的关系开始。



特别专栏 | Column

敏捷与结构性模块化（一）

1 简介

作者 [Richard Nicholson](#)，译者 [刘剑锋](#)

敏捷开发方法论日益流行，然而大多数“敏捷”专家和分析师都在孤立地讨论敏捷，也就是说忽视了系统“结构”（Kirk Knoernschild是一个例外，他编写了一本名为《Java Application Architecture》的图书阐述这一理念）。考虑到“敏捷”是基础实体的一个重要特性或属性，那么，这种疏忽令人感到很惊讶。一个实体要具有“敏捷”的特性，它必须具有高度的结构性模块化（structural modularity）特征（参见Scott Page的《Diversity & Complexity》）。

也许正因为这种疏忽，许多组织在敏捷开发流程方面进行投入但忽略了应用程序的结构。除了“如何实现一个敏捷的系统？”这个问题以外，有人肯定还会问，“如何构建一个在结构上具备高度模块化的系统？”

这个系列的文章将从探讨结构性模块化和敏捷之间的关系开始。

2 结构、模块化与敏捷

业务主管和应用开发人员经常面临相同的挑战。无论是商业领域还是服务于商业的软件，它都必须在成本范围内构建和维护。如果要保持实体的持续运营，就必须能够以低成本的方式快速响应难以预料的变化。

如果我们希望高效地管理一个系统，就必须先理解该系统。只有理解了系统，可控制的变更和定向升级才能成为可能。

当然，我们并不需要理解系统的每个组成部分的详细情况和特性，只需要理解所负责的系统的相关参数以及相应等级层次的行为即可。

隐藏服务实现

从外部角度来看，我们仅仅关心系统暴露的行为、提供的服务类型以及该服务的属性。例如，服务可靠吗？与替代方案相比有竞争力吗？

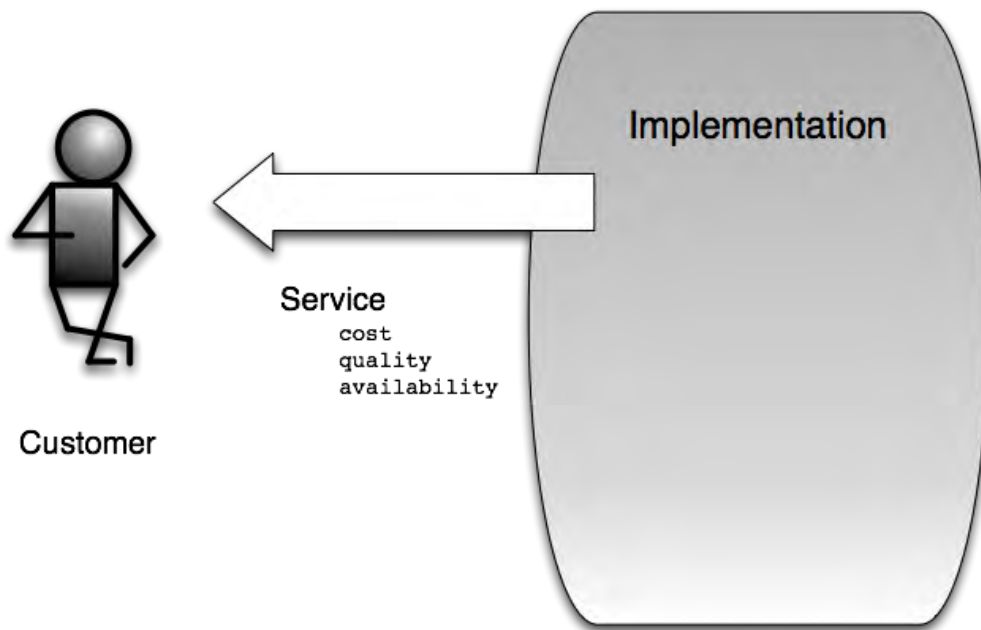


图1：服务的使用者

作为服务的使用者，我们并不关心服务的特性是如何实现的，我们只关心所提供的功能（Capability）是否可以满足我们的需求（Requirement）。

理解结构以便管理

不同于使用者，服务的实现方式对于服务的提供者而言，是极为重要的。为了更好的理解，我们为负责提供服务的系统建立概念模型，这是通过将系统分解成一组更小的相互关联的单元实现的。如果这个实体是某个公司，这张组件图就代表了“组织结构图（Organization Chart）”；如果这个实体是软件应用程序，那么这张图就是模块间依赖关系的映射图。

尝试理解抽象系统的第一步如下图所示。

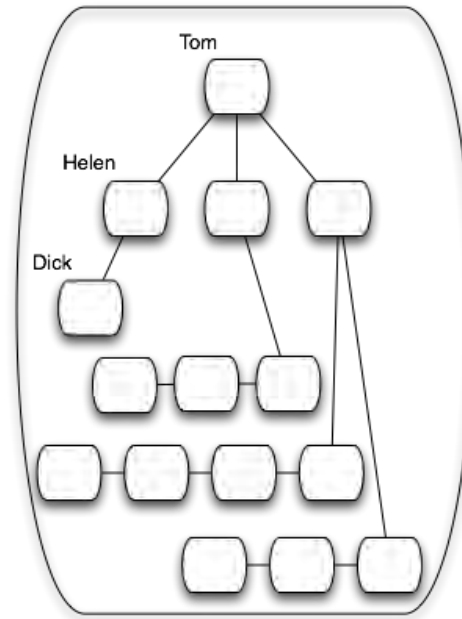


图2: 服务提供者和系统维护者

从上图的例子，我们可以马上知道：

1. 系统由15个组件组成。
2. 每个组件的名称。
3. 这些模块间的依赖，尽管我们无法知道这些依赖为何存在。
4. 尽管我们并不知道每个组件各自所承担的责任，从关联性出发，我们依然可以推断出“Tom”模块在系统中所占据的地位很有可能比“Dick”模块更为重要。

需要注意的是，这些组件可能并不是我们所创建的，我们也不必理解这些组件之间的内在结构。就像作为服务的使用者，只需关心服务所提供的功能，我们作为组件的使用者，只是需要它们的功能。

需求和功能 (Requirements & Capabilities)

到目前为止，我们仅仅知道组件之间存在依赖性，并不知道为什么会存在这些依赖性。另外目前的状态是与时间无关的。如果随着时间的推移，发生了变化又该怎样呢？

最初我们可能会借助于实体的名字，再加上版本号（version）或者版本范围（version range），结构的变化由版本的变化体现出来。然而，如图3所示，版本名称（versioned name）尽管表明了系统的改变，但却无法解释为什么Susan 2.0不能像Susan 1.0那样与Tom 2.1一起协同工作。

这是为什么呢？

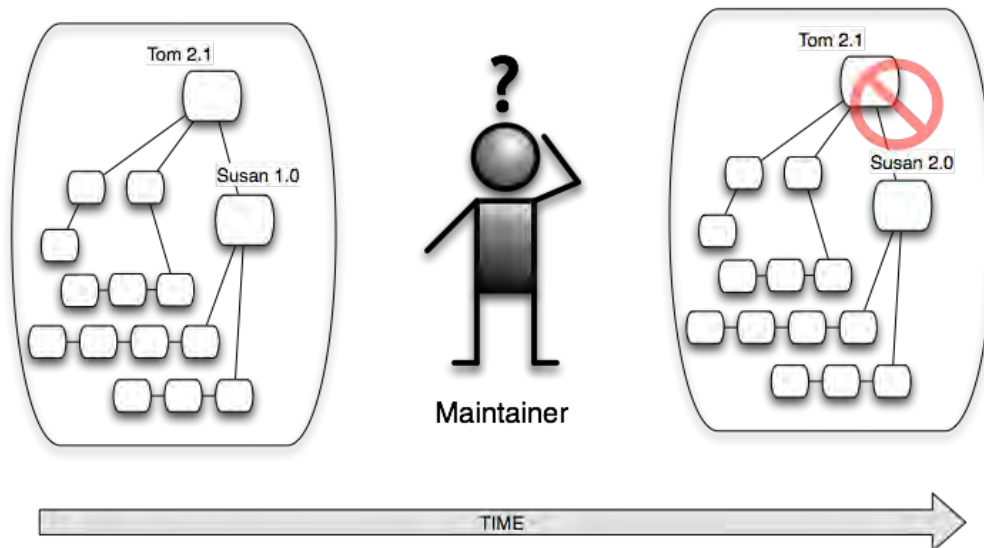


图3：如何跟踪结构随时间的变化？系统以前能够正确运行，后来由于一个组件的升级导致整个系统出错。为什么呢？

只有当我们仔细研究系统的功能和需求后，才能了解问题的原因。Tom 2.1需要管理者（Manager）的功能，这个功能在Susan 1.0中提供。然而稍后的Susan 2.0，由于她的职业规划，决定进行再培训，这时的Susan 2.0被赋予了新的Plumber 1.0功能，也就意味着其不再拥有管理者的功能了。

这个简单的例子向我们展示了模块间的依赖关系需要由需求和功能来表达，而不是它们的名字（Apache Maven项目最近正在讨论[为制品的名字采用版本范围](#)。）。尽管这是一个进步，但依然有缺陷，因为依赖还是用实体的名字来进行描述。）。这些描述应当能显示模块的本质，即模块应当能自我描述（需求、功能以及依赖应该进行文档化，但是随着时间的推移，这些描述会变得过时，如最初的文档制定之后，系统又发生了变化，而文档并没有得到更新。）。）。

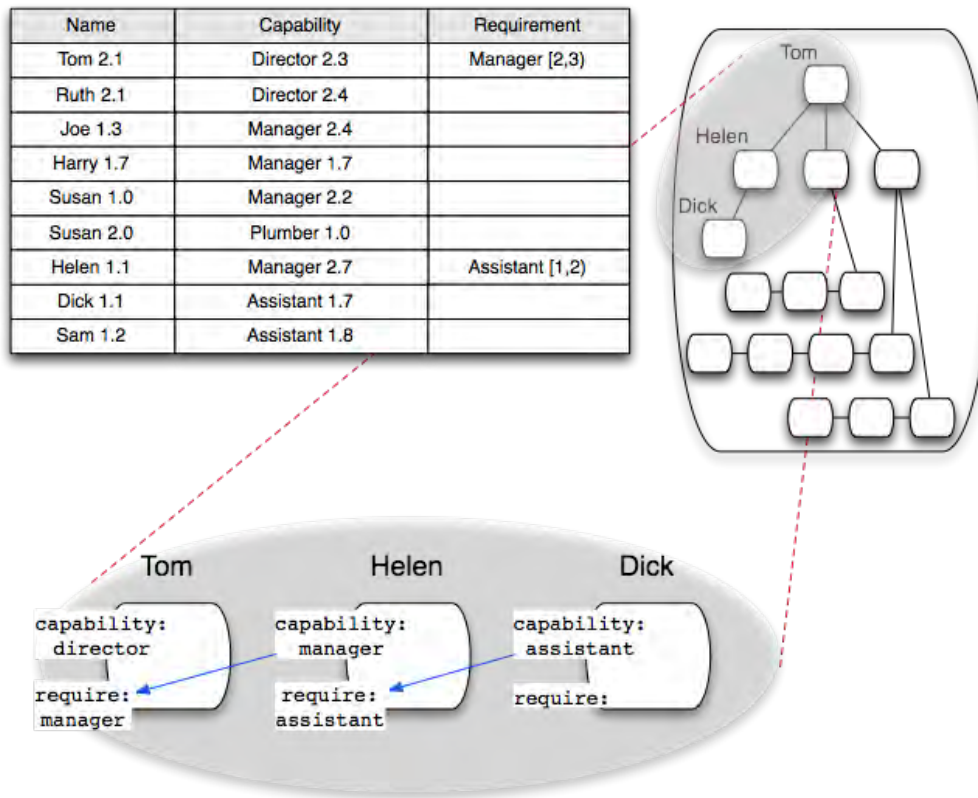


图4：组织化结构：按照功能、需求的术语以及语义化的版本来进行定义

如图所示，我们完全可以不引用具体实体的名字，而直接使用需求和功能来描述一个系统

系统演化和语义化版本的角色

目前，功能与需求是我们了解系统结构的主要途径。然而，要理解时间推移所带来的变化，我们依然还会遇到问题。

- 在组织结构图中，如果某个员工晋升了，那么原有的关联性是否依旧有效（功能增强）？
- 在一组互相关联的软件组件中，如果我们重构了其中的一个模块（可能改变其公开接口），原有的依赖是否依然有效？

通过简单的版本化，我们可以观察到系统所发生的变化，却无法了解这些变化所带来的影响。然而，如果采用语义化版本命名方式(见 <http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf>)，我们就能够传达系统变化而带来的潜在影响。

这可以通过以下方式实现：

- 将功能根据major.minor.micro 的版本模式来进行版本化。同时我们达成共识

， minor 或 micro 这两个版本域代表非破坏性的变化（non-breaking change）。例如，2.7.1→2.8.7。相反，major版本域的变化，例如，2.7.1→3.0.0表示有破坏性的变化（breaking change），组件的改变可能影响到它的使用者。

- 需求则使用可接受的功能的版本范围来表示。方括号“[”和“]”表示包含此值，而圆括号“(”和“)”表示不含此值。因此，范围[2.7.1,3.0.0) 表示任何版本高于或等于2.7.1并且低于3.0.0（不含3.0.0）的功能都是可接受的

使用这种方式，我们可以看到如果Helen代替了Joe，Tom的需求依然会得到满足。然而，同样有管理者功能的Harry却因为其功能仍是1.7版本，不在Tom的[2,3)需求范围内，所以无法进行替换。

通过使用语义化版本的命名方式可以表达系统变更所带来的影响。再加上需求和功能，我们具备了足够信息，能够保证在满足系统各部分依赖的前提下，进行模块的替换。

我们的工作到此告一段落，这样简单的系统是敏捷且易维护的！

敏捷——从上至下贯穿各层

最后的挑战与复杂性息息相关。试想如果下列情况出现时会发生什么：1）系统的规模和难度不断增长？2）系统的模块数量大幅增加，并且模块间的互相依赖性也大幅增加？有些读者在前面的例子中，可能已经注意到出现了某种程度的[自相似性](#)（self-similarity），你们或许已经从中猜到了答案。

服务的使用者选择我们的服务是因为服务所宣称的功能符合它们的需求（见图1）。而提供服务的系统的实现方式，对服务的使用者而言是不可见的。向下的每一层都沿用这一模式。系统的结构自然而然地根据各组件的功能和需求进行描述。（见图4）。这时组件的内部结构对于系统而言是不可见的。如图5所示，许多逻辑层都可能使用这种模式。

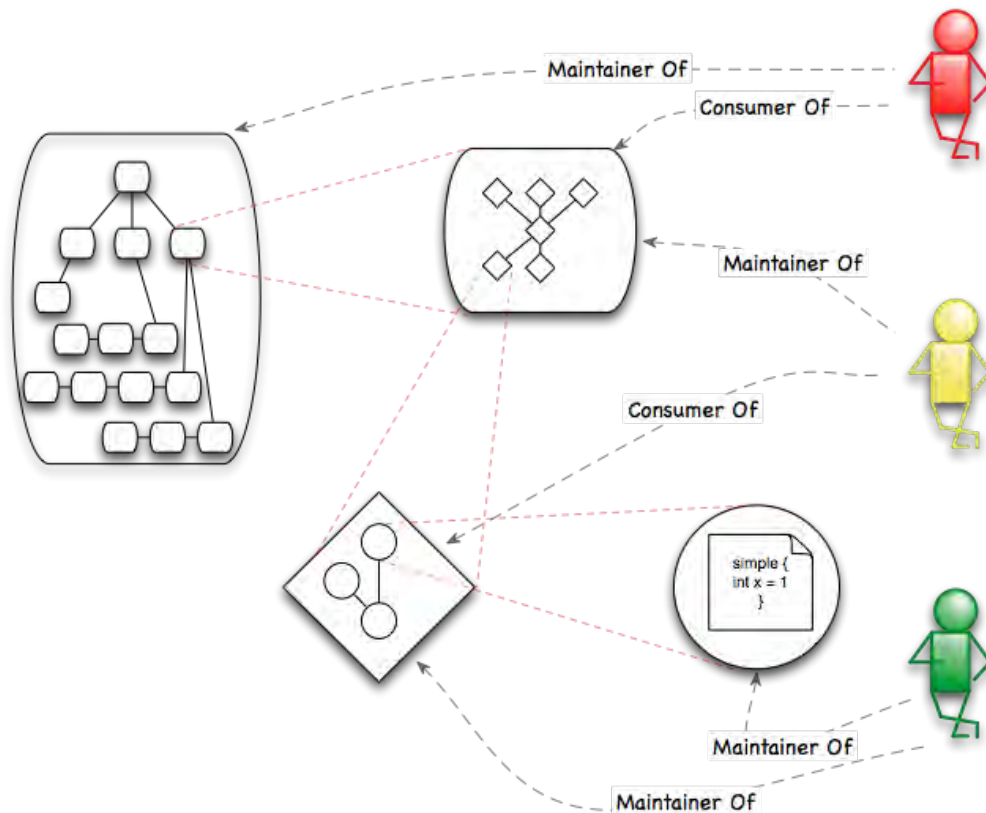


图5：敏捷的结构：每层只暴露必要的信息。每层都是由组件间的依赖所组成的，这些依赖通过需求和功能来进行表述。

所有真正敏捷的系统都是以分层的层级结构建立起来的。在每个结构化的层次中，各组件的自描述都遵循以下的规则：只描述当前层次的有关信息，关于更低层次的不必要细节是不会描述的。

这种模式不断出现于自然系统和人造系统中。自然生态系统中所构建的大量结构都是由嵌套的模块化组件所组成的。例如：

- 生物体
- 器官
- 组织
- 细胞

无独有偶，商业组织也有类似结构：

- 组织
- 部门
- 团队
- 个人

因此，我们也期望复杂的敏捷软件系统也参照这些最优的解决方案：

- 商业服务

- 粗粒度的业务组件
- 细粒度的子服务
- 代码级别模块

这项进程起源于20世纪90年代中后期，许多公司开始采用包括面向服务架构（Service Oriented Architecture, SOA）和企业服务总线（Enterprise Service Buses, ESB's）为代表的粗粒度模块化技术。商业程序通过已知的服务接口或消息传递的方式，较为宽松地联系在一起。SOA提倡更加“敏捷”的IT环境，即商业系统应该更加易于升级或替换。

然而在许多实际场景中，核心的应用程序一直没有更改。很多现有的程序只是简单地将接口暴露为SOA服务。从这一角度来看，SOA实质上并不能如其保证的那样节省开支和使商业快速化：<http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html>。

这是因为内部缺乏模块化，加入SOA之后的程序和没加之前一样难以修改

变得敏捷？

我们将在此章节，对目前为止的观点进行小结。

为了“敏捷”，系统必须符合以下的特性：

- 层级化的结构（Hierarchical Structure）：系统必须层级化，每一层由更低一层的组件构成。
- 隔离性（Isolation）：对于每个结构化的层级，高度的隔离性确保参与运行的组件的内部结构将是不可见的。
- 抽象化（Abstraction）：对于每一层，参与运行的组件的行为通过需求和功能加以表达。
- 自描述（Self-Describing）：在每层之中，参与运行的模块间的关系都必须是自描述的。也就是说，依赖性定义将通过需求和功能进行表达。
- 变化的影响（Impact of Change）：通过语义化的版本命名，变化对依赖的影响可以进行表述。

系统按上述原则建立，将是：

- 易于理解的（Understandable）：基于层级化的结构，系统在每一层的结构都易于理解。
- 高适应性的（Adaptable）：在每一层中，结构性模块化保证了变更的影响可以局限在那些相关的模块内部，高度模块化所建立起来的边界能够保护系统的其他部分不受影响。
- 可演化的（Evolvable）：每层中的组件都可以被代替。因此，系统可以支持多样化（diversity）并且是可演化的。

系统可以通过结构性模块化来实现敏捷。

在这个系列的下一部分，我们将会讨论OSGi™——Java™的模块化框架——如何满足结构性模块化的需求，从而为流行的敏捷方法学奠定基础，最终形成敏捷的企业。

原文英文地址：[Agility and Structural Modularity - part I](#)

作者简介

Richard Nicholson是Paremus 的CEO和创始人，这是一个2001年成立的软件公司，总部位于英国。

在意识到高度可维护以及高度敏捷的系统在本质上必须是高度模块化的之后，Paremus在2004年开始研究下一代的软件系统。这种持续的努力体现在了Paremus Service Fabric产品之中，这是一个高度可适应的、基于OSGi的自装配运行时，可用于企业级和云环境。作为OSGi联盟的主席（2010-2012），Richard开始推进OSGi Cloud并鼓励OSGi联盟参与到敏捷软件社区中。

Richard在很多的研究领域都保持了浓厚的兴趣，这支撑了Service Fabric的研发，他的研究领域包括复杂的适应性系统（Complex Adaptive System）以及敏捷（Agility）、模块化组装（Modular Assembly）、结构化多样性（Structural Diversity）和适应性（Adaption）之间的关系。

成立Paremus之前，Richard在花旗集团/Salomon Smith Barney，领导着欧洲系统工程（European System Engineering）相关的工作。Richard获得了曼切斯特大学的物理学荣誉学位，并在格林尼治皇家天文台（Royal Greenwich Observatory）获得天体学物理博士。

Richard的博客：<http://adaptevolve.paremus.com>。

Paremus的博客：<http://blogs.paremus.com>。

原文链接：<http://www.infoq.com/cn/articles/agile-and-structural-modularity-part1>

相关内容

- [敏捷与结构性模块化（一）](#)
- [敏捷转型中的看板](#)
- [敏捷时代的建模：敏捷团队的扩张除了代码还需要什么？](#)
- [诺基亚娱乐部门如何用DevOps补敏捷之不足](#)

特别专栏 | Column

敏捷与结构性模块化（二）

作者 [Richard Nicholson](#)，译者 [刘剑锋](#)

在[上一篇文章](#)中，介绍了结构性模块化与敏捷之间的关系，在这个系列的第二篇文章中，我们将会研讨OSGi™，在实现Java™的结构性模块化方面，OSGi扮演了核心的角色；OSGi与流行的敏捷方法论之间存在着自然的联系。

1 但我们已经实现了模块化！

绝大多数开发人员都同意程序应该模块化。尽管在面向对象的程序设计出现的早期，逻辑性模块化的要求就被迅速满足了（见http://en.wikipedia.org/wiki/Design_Patterns），但是软件行业花费了很长的时间才理解结构性模块化的重要性。特别是，结构性模块化可以提高程序的可维护性和灵活性，控制和减少环境带来的复杂性。

只是JAR文件的组合

在《Java Application Architecture》（本书已经由机械工业出版社引进出版，中文书名为《[Java应用架构设计：模块化模式与OSGi](#)》——译者注）一书中，Kirk Knoernschild探索研究了结构性的模块化，并建立了一套关于结构化设计的最佳设计模式。Knoernschild认为，在模块化风潮中并不需要开发模块化的框架；对于Java，JAR文件结构就足够了。



事实上，在“敏捷”开发团队中根据代码库的增长将应用分解为较小的JAR文件的做法并不罕见。随着JAR文件大小的增加，他们会被分解为更小的JAR文件的集合。从代码角度，特别是如果遵循Knoernschild的结构化设计模式，我们会认

为，从某个结构层看，程序是模块化的。

但这是否敏捷？

从创建应用的团队，以及负责随后维护人员的角度来看，应用是更敏捷的。团队了解依赖关系，以及变化的影响。然而，这种知识并没有关联到组件上。一旦团队成员离开了公司，程序和业务就有可能立刻受到影响。同样，对第三方（即使是同一个组织的不同团队）来说，该应用可能依然是单独的巨大代码库。

如果程序中只有一层结构性的模块，它必然不是自描述的。由于缺乏描述模块之间内在关系的元数据，由此产生的业务系统本质上是脆弱的。

那么MAVEN呢？

Maven文档（项目对象模型，Project Object Model——POM）也能表达组件之间的依赖关系。这些依赖关系由组件的名称定义。

鉴于此，基于Maven的模块化程序可以被任何第三方很容易地整合起来。然而，在[上篇文章](#)我们已经提到，由名称定义依赖关系是有缺陷的。由于组件之间的依赖关系并不能明确表明需求（Requirement）和功能（Capability），第三方不能推断出依赖关系的存在原因和可替代的部分。

程序可以被整合，但却不能变更。这种方式是否使得程序比之前的“JAR文件组合”更为“敏捷”还值得商榷。

对OSGi的需求

就如Knoernschild在《Java应用架构设计》中阐述的那样，一旦实现了结构性模块化，我们很容易就能将它迁移到OSGi之中，也就是Java领域中的模块化标准。

OSGi不仅帮助我们保证结构性的模块化，它同时提供了必须的元数据来保证我们建立的模块化结构也是敏捷的结构。

OSGi通过需求和功能表达依赖关系。因此，第三方可以立刻知道哪些组件可能是可替换的。因为OSGi同样使用语义化的版本控制，所以第三方可以立即推测出，某个组件的变更是否有破坏系统的潜在危险。

OSGi同样也具有展现结构化层次的能力。

在模块化的一端，我们使用面向服务的架构（Service Oriented Architecture

，SOA)；在另一端，我们使用Java包和类。然而，正如Knoernschild表述的那样，在这两端之间缺少了必要的层次。

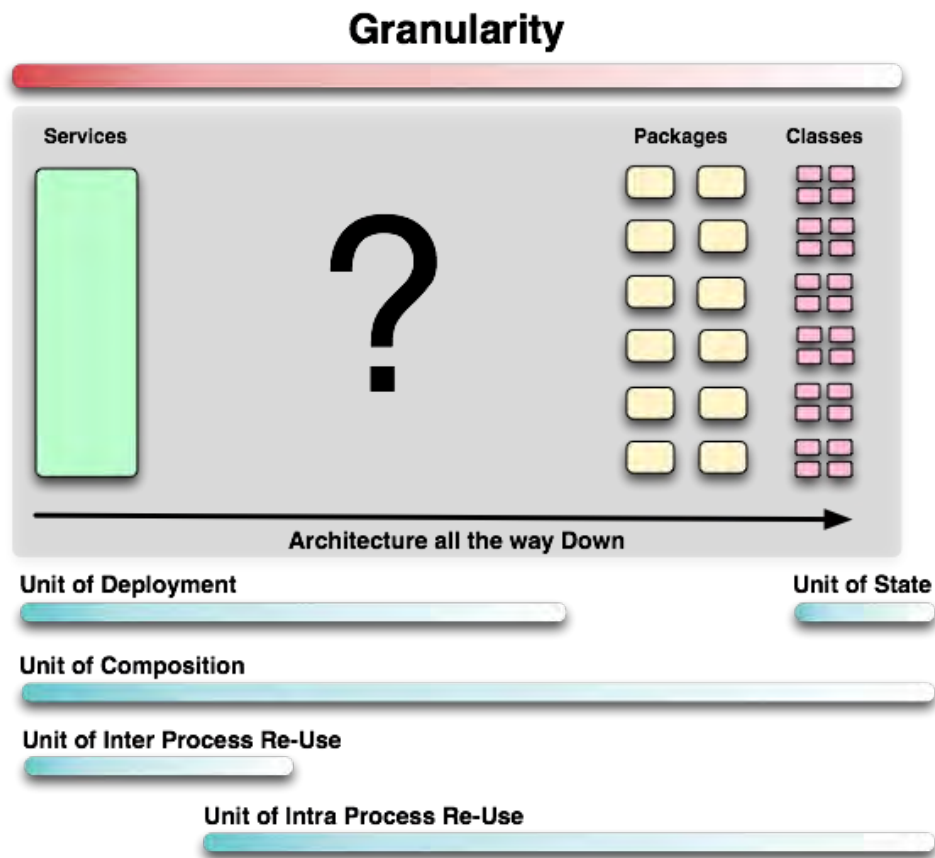


图1：结构化层次：缺失了中间的部分（Kirk Knoernschild – 2012）

中间缺少层次的问题，在OSGi中得到了直接的解决

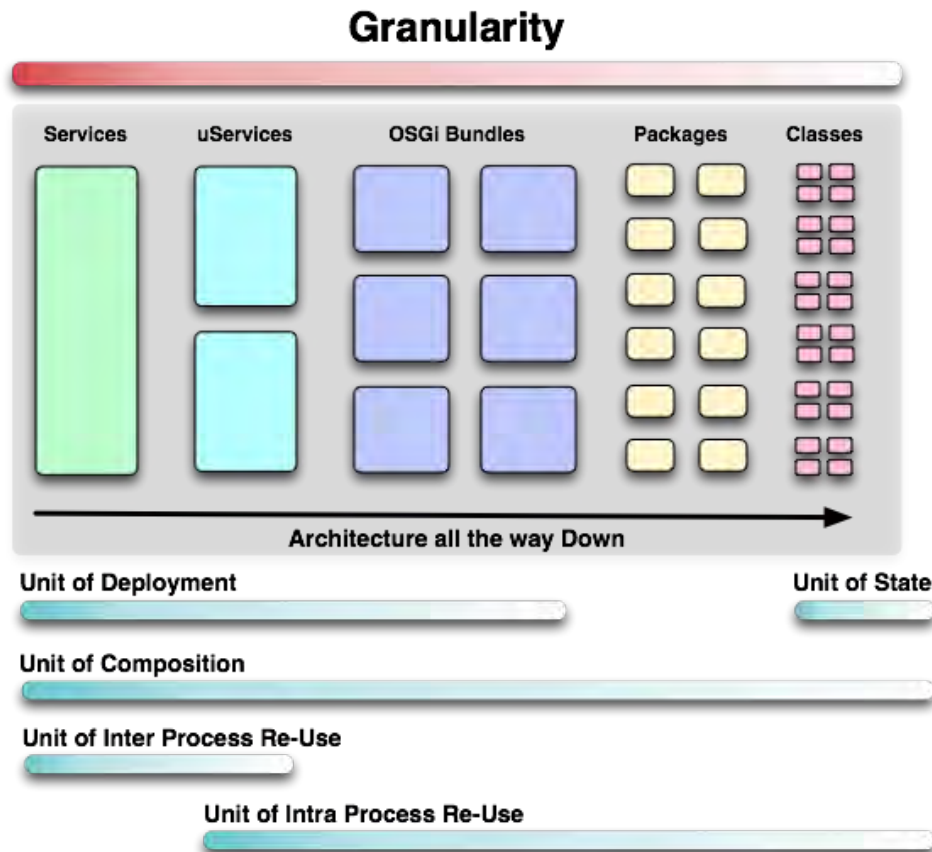


图2: 结构化层次: OSGi 服务和Bundle

正如Knoernschild 所描述的, OSGi所提供的模块化层次解决了一些比较关键的问题:

- 代码的重用: 通过使用OSGi Bundle, 能够促进代码的重用。
- 进程内和进程间的重用: OSGi服务是轻量级服务, 允许相互间动态寻找或绑定对方。OSGi服务可能发布在相同的JVM中, 也可以通过实现了OSGi远程服务规范的工具发布在网络内的分布式JVM中。粗粒度的业务程序可能是由数个更小粒度的OSGi服务组成的。
- 部署单元: OSGi Bundle提供对自然单元进行部署、更新和补丁的基础功能。
- 组合单元: OSGi Bundle和服务在组合的等级结构中都是基本要素。

因此, OSGi Bundle和服务作为OSGi联盟开放规范的主要部分, 为Java提供了之前缺失的、结构性模块化中的重要层次。原则上, OSGi技术使基于Java的业务系统实现“敏捷——从上至下贯穿各层”成为可能。

我们现在可以看到, OSGi的结构 (Bundle和服务) 与流行的敏捷方法论很匹配, 并且能够促进它们得以实现。

拥抱敏捷

敏捷运动主要关注于实现敏捷的产品开发和交付相关的流程 (Process), 当前

已经存在一些精益与敏捷（Lean&Agile）的方法，其中的每一种都是两种广为人知的方案的变种、混合或延伸。这两种方案就是Scrum和Kanban（http://en.wikipedia.org/wiki/lean_software_development）。

为了每种方案更加有效，都需要在某种程度上具有结构性的模块化。

Scrum

客户一直在改变他们的想法。Scrum承认“需求改动（requirement churn’）”的存在，并且采用基于经验的（[empirical](http://en.wikipedia.org/wiki/empirical)，<http://en.wikipedia.org/wiki/empirical>）方式完成软件的交付。Scrum承认问题不能完全被理解或被预先定义。Scrum着重于最大限度地提高团队能力，以更快完成交付并解决更多新的需求。

Scrum是迭代式渐进的增长过程，以“Sprint”作为基本的开发单元。每个Sprint都是在“固定时间（time-boxed）”（<http://en.wikipedia.org/wiki/Timeboxing>）内的工作，即它有一个明确的期限。对于每个Sprint，它的期限是事先固定的，一般在一周到一个月之间。Sprint开始之前会有一个明确任务和目标的计划会议，Sprint之后会有一个回顾会议，这个会议上要评估进度并总结本Sprint的经验教训。

每个Sprint的期间，团队将完成产品的一部分。纳入Sprint的特性来源于产品的backlog，这是一个按顺序排列的需求（<http://en.wikipedia.org/wiki/Requirement>）列表。

Scrum鼓励创建自组织的团队，这通常需要团队所有成员的协同工作和口头交流实现。

Kanban

“Kanban”起源于日语，意思是“布告板”或“看板”。它的根源可以追溯到20世纪40年代后期的日本丰田汽车制造公司（见<http://en.wikipedia.org/wiki/Kanban>）。Kanban鼓励团队对工作、工作流程、进程、风险等建立共同的理解，以此团队将取得在某个问题上的共识，然后在此基础上提出能够达成一致的改进方案。

从结构性模块化的角度来看，Kanban的重点在于在制件（Work-In-Progress, WIP）、有限的发布和反馈，可能这是该方法中最为有意思的：

1. 在制件（WIP）一定是处在一个多级工作流的某个步骤。当目前的WIP所规定的流程完全走完后，在下一阶段有处理能力的情况下，工作将被“拉”到下一阶段。
2. 工作在每个工作流程的进度都是受监视的、可评估、可报告的。通过积极地管理

流程（flow），我们可以对那些持续发生的、增量式的以及渐进的升级变更对系统所产生的积极或消极影响进行评估。

由此可见，Kanban鼓励持续的、增量式的以及渐进的系统变更。随着结构性模块化的程度得到提高，流动率也同时增加，而每个更小的模块将消耗更少的时间处于“在制品”状态。

敏捷成熟度模型

随着结构性模块化程度的提高，Scrum和Kanban的目标将更容易实现。能力成熟度模型（Capability Maturity Model，参见http://en.wikipedia.org/wiki/Capability_Maturity_Model），主要被公司或项目用于估测它们在软件开发过程获得的改进。同样的，模块化成熟度模型（Modularity Maturity Model）更倾向于描述组织或项目在模块化方面的进展，这个概念是由Graham Charters博士在OSGi Community Event 2011 上(参加 <http://slidesha.re/ZzyZ3H>) 提出的。现在我们扩展一下这个理念，分析模块化成熟度模型对企业在敏捷方面的影响。

模块化成熟度模型的阶段总共分为以下六个阶段：

特定的（Ad Hoc）——没有任何正式的模块化内容。依赖关系不明。Java程序没有或很少有什么结构可言。在这种环境下，敏捷管理流程将无法实现商业目标。

模块（Module）——不再直接使用类（或者包含类的JAR），而是使用有清晰的版本规范的模块，依赖关系可由模块的标识确定（包括版本名）。Maven、Ivy和RPM就是这类模块化解决方案的例子，在这里依赖通过版本化的标识符进行管理。企业一般都有存储这些模块的库。然而这些库的价值并没有完全体现出来，因为这些模块没有描述它们的功能和需求。

许多企业内部的研发团队都在这一级别的成熟度。象Scrum这样的敏捷流程是可能的，而且也对企业提供了一些价值。然而，流程在有效性和扩展性方面的最终收益将会因为结构性模块化而受到限制。例如模块间的需求和功能通常通过口头进行交流。这些没有良好定义的结构化依赖同时会影响到持续集成（Continuous Integration, CI）方面的能力。

模块化（Modularity）——模块的标识并不同于真正的模块化。我们已经知道模块的依赖关系可以通过契约关系（即功能和需求）而不是组件名来进行表述。在这一点上，基于功能和需求表述依赖关系的解决方案，成为动态软件组建机制的基础。这个层次的结构化模块化依赖同时会使用语义化的版本。

通过采用如OSGi之类的模块化框架，Scrum过程中扩展性的问题也得到了解决

。由于强制封装，并使用功能和要求来定义依赖关系，OSGi使得众多小团队可以进行独立且高效的开发，而且这些开发可以并行。Scrum管理工作的效率也能获得相应的提高。每个Sprint能够关注于一个或多个事先定义好结构实体，也就是OSGi Bundle的开发或重构工作。同时，语义化版本命名规则，使得不同的团队之间能高效地交流。因为OSGi Bundle 提供了有效的模块化和隔离机制，并行的研发团队可以在同一应用中的不同结构范围内安心地工作。

服务 (Service) ——对于用户而言，基于服务的协作方式隐藏了服务的构建细节。客户端与提供者的实现之间能够实现解耦。由此可知，服务提倡松散耦合。具有动态寻找和绑定功能的OSGi服务直接支持了松耦合、动态结构、应用的整合或装配等功能。更为重要的是，服务是实现运行时敏捷的基础，包括快速增强业务功能或自动适应环境的改变。

结构性的模块化达到了这一层次，企业可以简单而自然地运用Kanban原理，并达到可以持续集成的目标。

委托 (Devolution) ——文件的所有权移交给模块化的资源库，资源库鼓励协作和统一管理。资产会根据它们所声明的功能自动选择。这样做的好处包括：

- 对已有的模块更加了解
- 减少重复工作并提高质量
- 协作和授权
- 质量和操作控制

因为软件具有一套完整的需求和功能说明，开发人员能通过语义化的版本与第三方交流模块的变更（破坏性的或非破坏性的）。委托机制允许开发团队按其需求快速找到第三方伙伴的软件。委托机制保证了灵活的软件创建方式，使分布的团队能够以更为有效的方式进行交流。软件可以由同一组织的其他团队创建，也可以使用外部第三方的软件。委托机制阶段提高了代码重用度和效率，降低了外包、众包、内包在创建软件制件方面的风险。

动态化 (Dynamism) ——这个等级建立在模块化、服务和委托的基础之上，并且是敏捷的顶点。

- 可以根据模块化组件快速地装配出业务程序。
- 因为保证了高度的结构性模块化（通过OSGi Bundle的边界进行隔离），组件可能被众多小的开发团队——本地、近岸和离岸（on-shore, near-shore, off-shore）——高效率地创建并维护。
- 因为每个程序都是自描述的，即便最复杂的业务系统也可以很容易地被理解、维护和改进。
- 因为使用了语义化的版本，变更的影响可以在所有参与人员之间得到有效的交流，这包括管理和变更控制流程等。

- 软件的补丁可以热部署到产品中，而不需要重启业务系统。
- 可以快速地扩展和启用程序功能，同样不需要重启业务系统。
- 最后，因为动态装配进程知道运行时环境的功能，程序的结构和功能会根据环境自动调节，实现了在共有云或传统数据中心环境中的透明部署和优化。

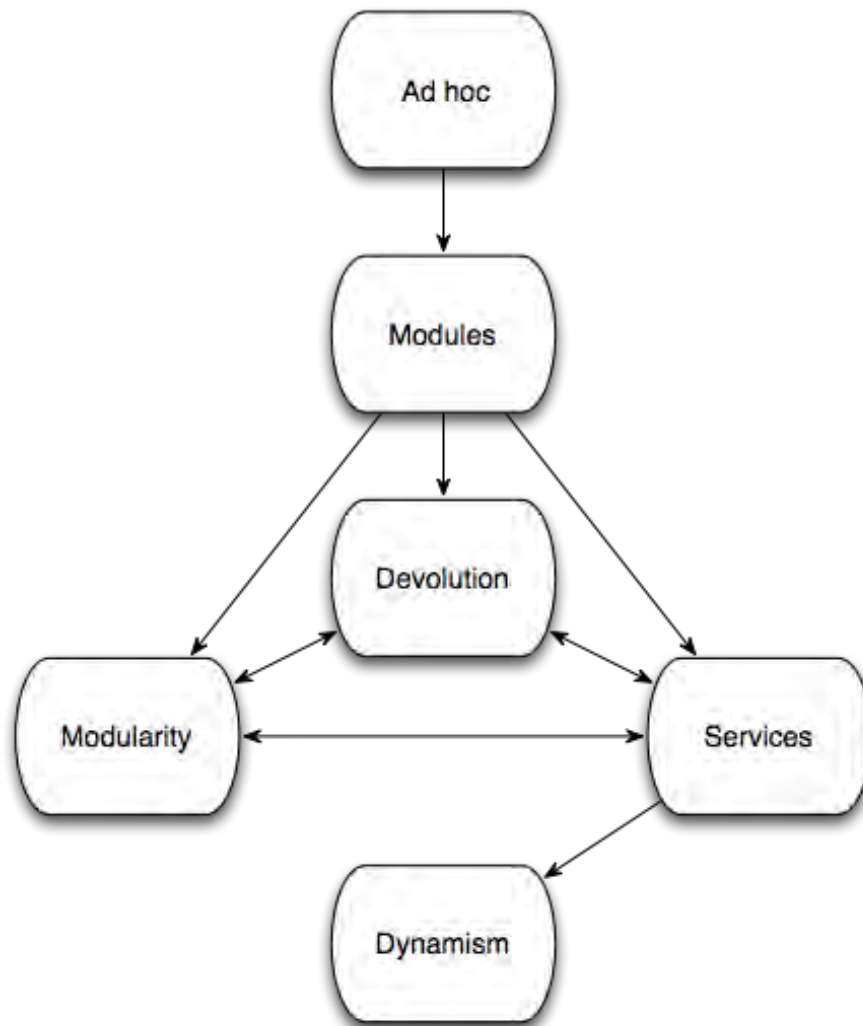


图3 模块化成熟度模型

组织的模块化演变（Modularisation Migration）策略，将由贯穿这些模块化等级的路径所决定。大多数组织已经从最初“特定的”阶段发展到“模块”阶段。同时，有些重视高度敏捷的组织希望跨入终点，即动态化。每个组织可能通过数种路径，配合必要的发展策略，从“模块”达到“动态化”。

- 为了尽快地实现收益，组织可能会通过将现有代码重构为OSGi Bundle，从而直接发展到“模块化”阶段。随后自然就能获得“委托”和“服务”的好处。对于未开发的应用显然也要采取这种策略。
- 对于遗留的程序，可选择先达到“服务”阶段。首先，把粗粒度的软件组件用OSGi服务来表达，随后，在每个服务的层面上推动代码级的模块化（即OSGi Bundle）。对于那些拥有许多遗留程序的大型企业，这是一条更加容易的路径。
- 最后，通过对现有的软件采用OSGi元数据，企业可以开始初步到达“委托”阶段

。需求与功能的采用，加上语义化版本的使用，都能为第三方明确地描述现存的结构以及变更的影响范围。尽管结构性模块化并没有得到提高，但是“委托”阶段使得企业以后更便于到达“模块化”和“服务”的层级。

拥有不同的选择以及根据具体的场景实践这些选择，正是我们所期望达到的更加敏捷的环境。

原文英文地址：[Agility and Structural Modularity - part II](#)

作者简介

Richard Nicholson是Paremus 的CEO和创始人，这是一个2001年成立的软件公司，总部位于英国。

在意识到高度可维护以及高度敏捷的系统在本质上必须是高度模块化的之后，Paremus在2004年开始研究下一代的软件系统。这种持续的努力体现在了Paremus Service Fabric产品之中，这是一个高度可适应的、基于OSGi的自装配运行时，可用于企业级和云环境。作为OSGi联盟的主席（2010-2012），Richard开始推进OSGi Cloud并鼓励OSGi联盟参与到敏捷软件社区中。

Richard在很多的研究领域都保持了浓厚的兴趣，这支撑了Service Fabric的研发，他的研究领域包括复杂的适应性系统（Complex Adaptive System）以及敏捷（Agility）、模块化组装（Modular Assembly）、结构化多样性（Structural Diversity）和适应性（Adaption）之间的关系。

成立Paremus之前，Richard在花旗集团/Salomon Smith Barney，领导着欧洲系统工程（European System Engineering）相关的工作。Richard获得了曼切斯特大学的物理学荣誉学位，并在格林尼治皇家天文台（Royal Greenwich Observatory）获得天体学物理博士。

Richard的博客：<http://adaptevolve.paremus.com>。

Paremus的博客：<http://blogs.paremus.com>。

原文链接：<http://www.infoq.com/cn/articles/agile-and-structural-modularity-part2>

相关内容

- [敏捷与结构性模块化（一）](#)
- [敏捷转型中的看板](#)
- [敏捷时代的建模：敏捷团队的扩张除了代码还需要什么？](#)
- [诺基亚娱乐部门如何用DevOps补敏捷之不足](#)

特别专栏 | Column

敏捷与结构性模块化（三）

作者 [Richard Nicholson](#)，译者 [刘剑锋](#)

该系列的[第一篇文章](#)介绍了结构性模块化和敏捷的根本性的关系，在[第二篇](#)中我们了解到如何使用OSGi实现高度敏捷和高度可维护的软件系统。

第三篇文章基于标题为“现实世界的挑战：基于OSGi/Bndtools的开发、发布和版本控制的工作流程”（*Workflow for Development, Release and Versioning with OSGi / Bndtools: Real World Challenges*, <http://www.osgi.org/CommunityEvent2012/Schedule>）的演讲。在这篇演讲中西门子团队展示了这些业务驱动的方案。这些方案实现了基于OSGi的高度敏捷的持续集成环境。

需求

西门子公司技术研究部门由许多具有不同技能的工程师所组成，他们的领域涵盖计算机科学、数学、物理、机械工程和电气工程。该部门向西门子的业务单位提供了基于神经网络技术和其他的机器学习算法的解决方案。比起理想化的概念，西门子的业务部门更需要可以运行的样例产品。所以该部门需要为业务部门快速地建立解决方案的原型。

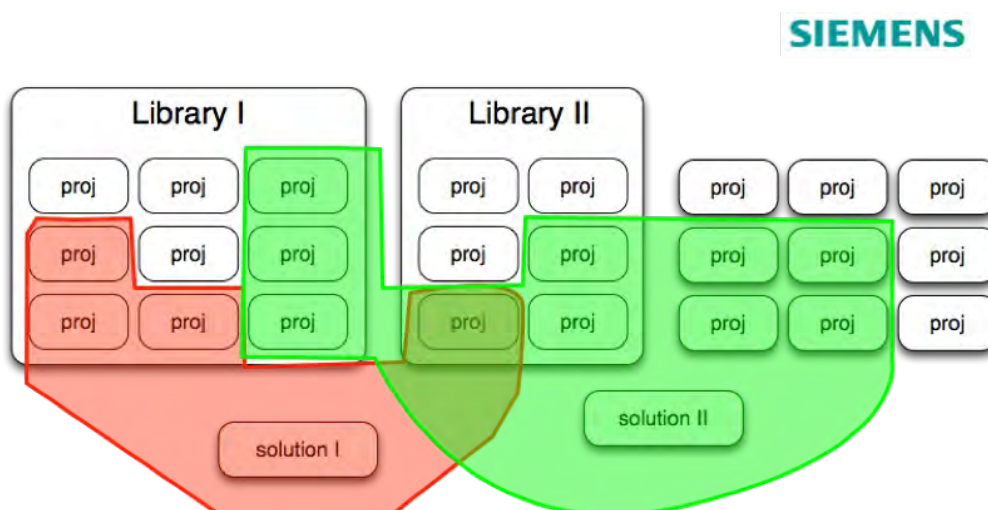


图 1：西门子的产品库

为了快速地建立原型，理想的方案是建立一个软件组件库，并以此为核心，这样西门子的研发团队就可以快速地发布新的功能，业务部门也能够很快地提供新的产品。

实现这一目标所采用的解决方案必须满足以下要求：

1. **可重复构建**：解决方案必须能够保证即使过了很多年，依然能够基于完全相同的源码和依赖进行重新构建。这样西门子可以持续支持多个被不同客户使用的发行版本。
2. **可靠的版本**：通过使用OSGi的语义化版本命名，构建过程中所有发布的组件版本永远能够正确地对应用其语义，西门子可以快速且可靠地组装出组件的集合（包括他们自己的软件、第三方软件和开源软件），并且高度确信它们可以正常运作。
3. **全程可追踪**：软件所发布的所有组件，与QA测试过的组件，总是完全相同的，并且，能够追溯到它们的源码和依赖性。这使得从测试状态变为可发布状态的过程中，不再需要进行重新构建工作。

最后，单独的软件组件和最终所组成的产品，必须有统一的应用启动、生命周期和配置方式。

解决方法

他们选择OSGi作为实现模块化的框架，这个决定基于OSGi技术的成熟度、支撑OSGi实现的开放行业规范以及OSGi联盟的技术管理。这个**持续集成**的解决方案基于开发和发布/产品的OSGi Bundle 库（Development and Release/Production OSGi Bundle Repository, OBR）。由于OSGi的组件完全是自描述的（**需求和功能的元数据**），特定的业务功能可以动态地根据模块间的依赖关系从有关的库中自动加载。

西门子的团队也想实施“所测即所发布”（*What You Test Is What You Release*, WYTIWYR）的最佳实践。用于发布的软件不应该在测试以后重建，在测试过程中，构建环境有可能会发生改变。许多组织确实在发布过程中重新构建软件产品，比如从1.0.0.BETA变为1.0.0.RELEASE。这一常用但不算太好的方法是因为依赖关系是由组件的名字来实现的。

最后从技术角度来看，解决方案必须有以下特点：

- 可以与标准的开发工具一起使用，如Java中的Eclipse；
- 对OSGi强有力的支持；
- 支持不同软件库的理念；
- 支持自动的语义化版本（即自动计算需要导入的版本范围，并且自动增加输出的版本号）——因为这一过程对人类来说实在太繁琐了！

基于这些原因西门子公司选择了[Bndtools](#)。

解决方案

下面一系列的图示解释了西门子公司解决方案的关键属性。

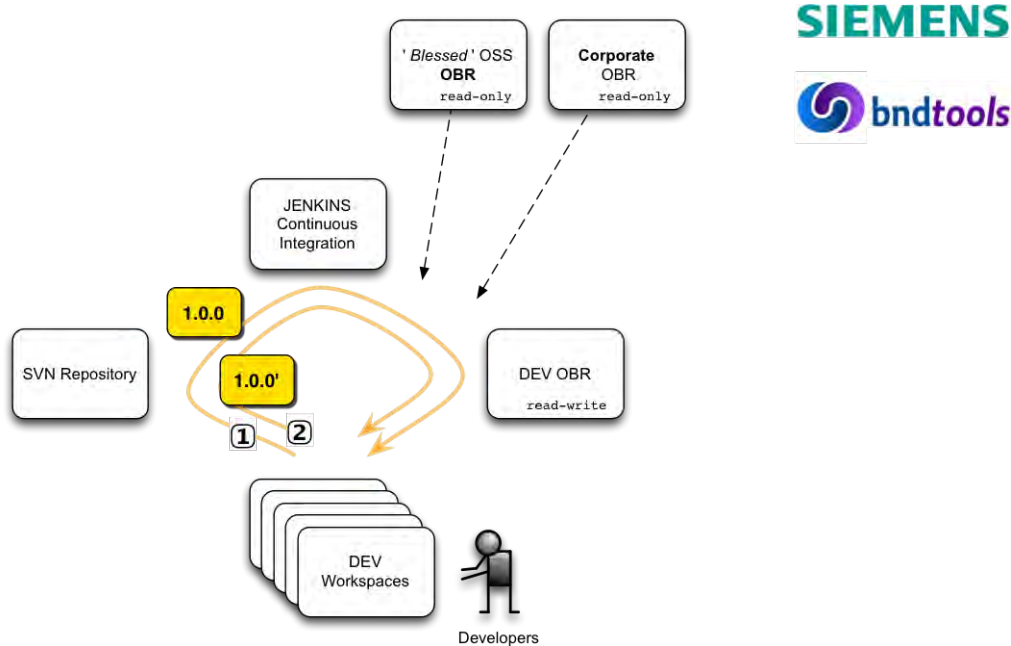


图 2：以库为中心、快速迭代并且在开发过程实现版本重用

Bndtools是一个以库为中心的工具，让开发者可以从多个*OSGi Bundle库* (OBR) 中取得OSGi的bundle。除了本地用于开发的读写库之外，开发者也可以从其他只读库中寻找OSGi bundle，比如，组合使用公司内部的开源库、公司专有的库和经批准的第三方库等。开发者可以很容易地从一个经认证的库列表中选择所需的库，并从中选取所需的组件，并把它们拖到Bndtools的工作区之中。

开发者将本地工作的代码放入SVN库。SVN库只存放在制品 (Work-In-Progress, WIP)。Jenkins的持续集成服务器不停地构建、测试并将建好的OSGi组件加入共享的只读开发库中。这些组件可以马上通过Bndtools被所有的开发人员使用。

随着开发人员的快速开发，每天会进行多次的构建，这将会变得难以管理，对每次开发构建都增加版本号确实是没有意义的。由于这个原因，在开发环境中我们允许重复地使用版本号。

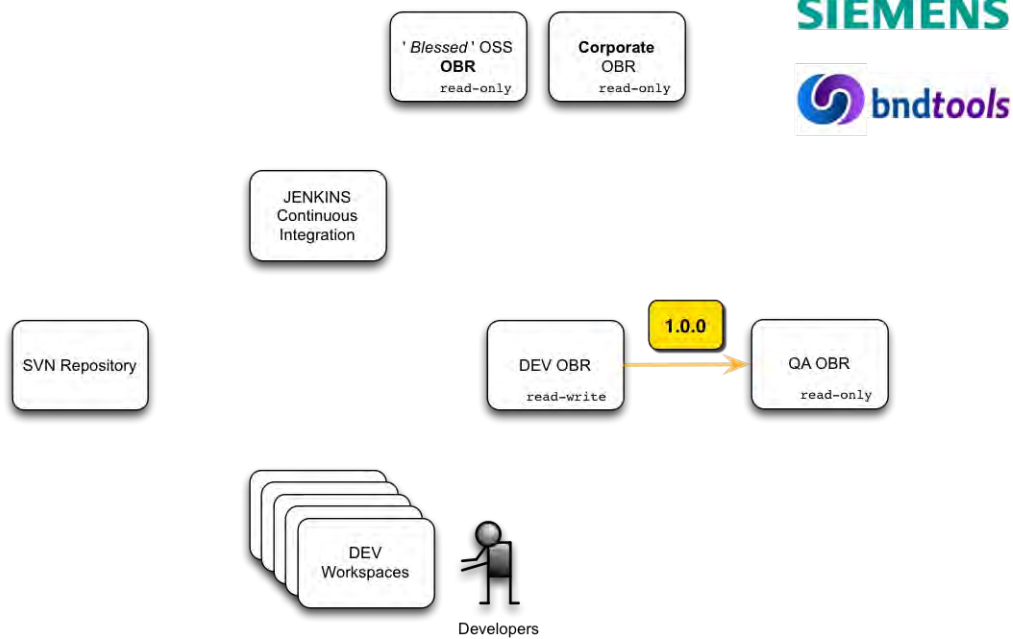


图 3: 发布

就绪之后，开发团队可以将模块发布到只读的QA库。

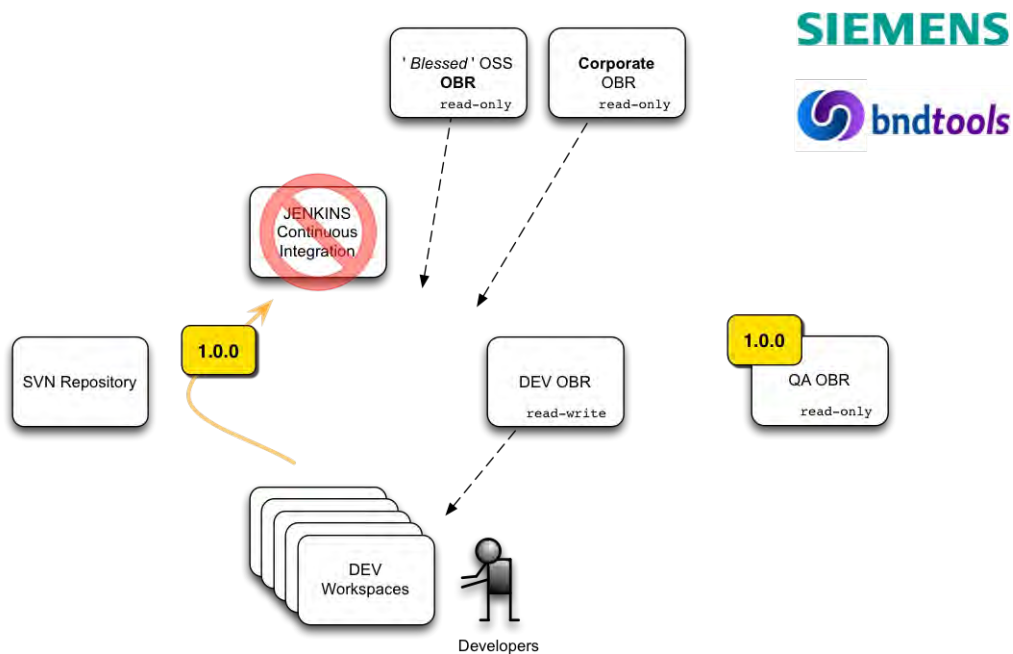


图 4: 锁定

当模块一旦进入QA后，在开发库中它就变成只读的了。任何修改或重新构建都会失败。如要修改，开发人员必须增加版本号。

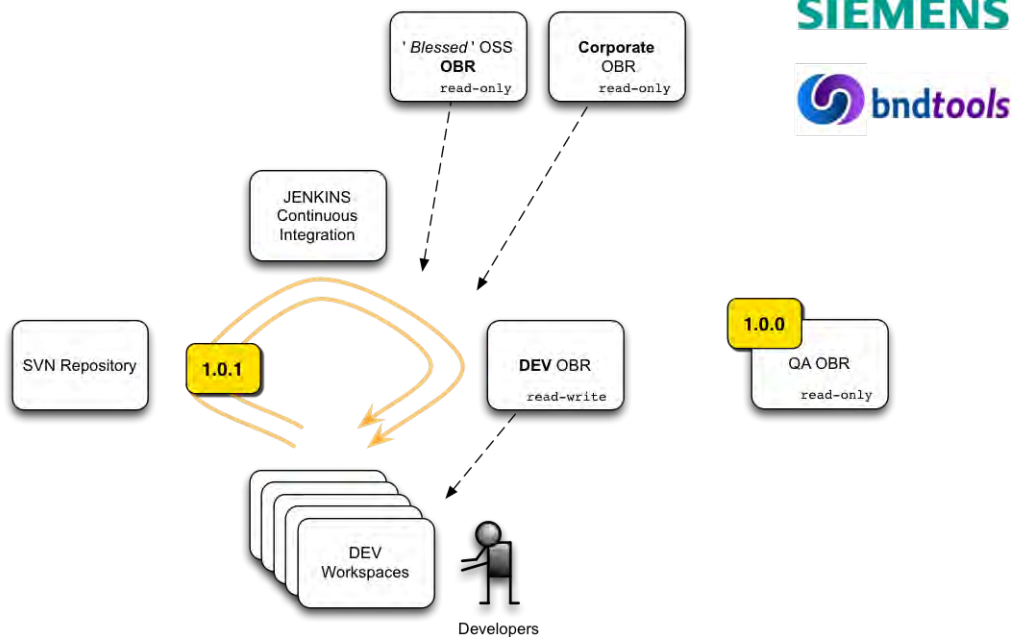


图 5：版本递增

开发人员可以使用Bndtools的自动化语义版本功能来实现版本的递增，从而确保当前的版本号能够表达出目前的WIP版本与之前版本的区别。根据前面文章中对语义版本规则的讨论：

1. 1.0.0 => 1.0.1 ... “缺陷修正”
2. 1.0.0 => 1.1.0 ... “新的功能”
3. 1.0.0 => 2.0.0 ... “破坏性的变化”

我们可以看到新的1.0.1版本是一个缺陷修正版本。

再看敏捷成熟度模型

在前面的文章中我们讨论了敏捷成熟度模型。根据该模型对西门子的解决方案进行评估，所有高度敏捷系统所必需的特征它都满足。

- *委托 (Devolution)*：通过Bndtools对OSGi库进行方便的使用
- *模块化及服务 (Modularity&Service)*：整体的解决方案。决策中必不可少的一部分就是采用以OSGi为中心的方案。

正如Kirk Knoernschild在DEVOXX2012上的演讲“[自上而下的架构 \(Architecture All the Way Down\)](#)”所讲的，敏捷运动专注于实现在敏捷开发时社交和流程 (Social and Process) 的方面，但根本性的因素——“结构性的模块化”——并没有被重视。那些想在庞大的代码库中实现“敏捷”的人对此应该深有体会。西门子公司通过OSGi实现结构性的模块化，由此达到敏捷的目标，与此同时也实现了社交和流程方面的敏捷开发。

[Bndtools](#)是促成这一解决方案的关键。同时，西门子公司的业务需求也反过来促进和形成了Bndtools的关键功能。在此我感谢西门子公司允许这些工作成果被Paremus和OSGi联盟所使用。

Bndtools的更多信息

Bndtools基于[Peter Kriens](#)的**bnd**项目，这个GITGHUB项目由Neil Barlett在2009年早期开始，bnd是业界创建OSGi bundle的事实标准。Bndtools包括了Neil在培训时帮助学生开发的一系列工具，以及[Paremus在SIGIL项目](#)上的一些工具。

[Neil Barlett已经多次陈述](#)过Bndtools的目标，即让开发敏捷和模块化的Java应用变得更容易，而不是更难。西门子的项目显示Bndtools能够迅速达到这个核心的目标。Bndtools得到了越来越多的开源社区和软件供应商的支持，这其中就包括了Paremus的长期支持。现阶段Bndtools的目标是支持OSGi Blueprint、与Maven更好的集成以及在OSGi云计算环境里便捷地加载运行时发布版本的适配器（runtime release adaptor），这样的环境包括Paremus的Service Fabric。

更多关于OSGi / Bndtools的理念可以在Neil Barlett 2013年5月在日本OSGi用户组的演讲材料找到：[NeilBartlett-OSGiUserForumJapan-20130529](#)。对那些想实现Java/OSGi敏捷开发的公司来说，Paremus提供这方面的工程咨询服务，以帮助他们实现该目标。Paremus也为各公司的开发人员提供现场的高级OSGi培训。如有兴趣可以[联系我们](#)。

最后的章节

在敏捷和模块化系列的最后一篇里，我会讨论敏捷和运行时平台（Runtime Platform）。敏捷的运行时平台是Paremus从2004年来就专注的领域，那时Service Fabric刚刚发布最初的版本，当时还被称为Infiniflow。对于敏捷运行时环境的追求使Paremus从2005年起采用了OSGi，并在2009年成为了OSGi联盟的会员。

但是OSGi的运行环境并不统一。尽管OSGi在基础上使敏捷的运行时环境成为可能，但单纯地使用OSGi并不能保证运行环境的敏捷。用OSGi建立脆弱的系统也是可能的。下一代的动态模块化平台，如Paremus的Service Fabric，不但必须使用OSGi，而且必须要采纳OSGi本身所基于的根本性的设计理念。

原文地址：<https://adaptevolve.paremus.com/?p=1380>

作者简介

Richard Nicholson是Paremus 的CEO和创始人，这是一个2001年成立的软件公司，总部位于英国。

在意识到高度可维护以及高度敏捷的系统在本质上必须是高度模块化的之后，Paremus在2004年开始研究下一代的软件系统。这种持续的努力体现在了Paremus Service Fabric产品之中，这是一个高度可适应的、基于OSGi的自装配运行时，可用于企业级和云环境。作为OSGi联盟的主席（2010-2012），Richard开始推进OSGi Cloud并鼓励OSGi联盟参与到敏捷软件社区中。

Richard在很多的研究领域都保持了浓厚的兴趣，这支撑了Service Fabric的研发，他的研究领域包括复杂的适应性系统（Complex Adaptive System）以及敏捷（Agility）、模块化组装（Modular Assembly）、结构化多样性（Structural Diversity）和适应性（Adaption）之间的关系。

成立Paremus之前，Richard在花旗集团/Salomon Smith Barney，领导着欧洲系统工程（European System Engineering）相关的工作。Richard获得了曼切斯特大学的物理学荣誉学位，并在格林尼治皇家天文台（Royal Greenwich Observatory）获得天体学物理博士。

Richard的博客：<http://adaptevolve.paremus.com>。

Paremus的博客：<http://blogs.paremus.com>。

原文链接：<http://www.infoq.com/cn/articles/agile-and-structural-modularity-part3>

相关内容

- [敏捷与结构性模块化（一）](#)
- [敏捷转型中的看板](#)
- [敏捷时代的建模：敏捷团队的扩张除了代码还需要什么？](#)
- [诺基亚娱乐部门如何用DevOps补敏捷之不足](#)
- [适合敏捷开发的合同模式：目标价格，风险均摊](#)



图灵社区 iTuring.cn



图灵教育微信



图灵访谈微信

图灵教育推荐

在线出版

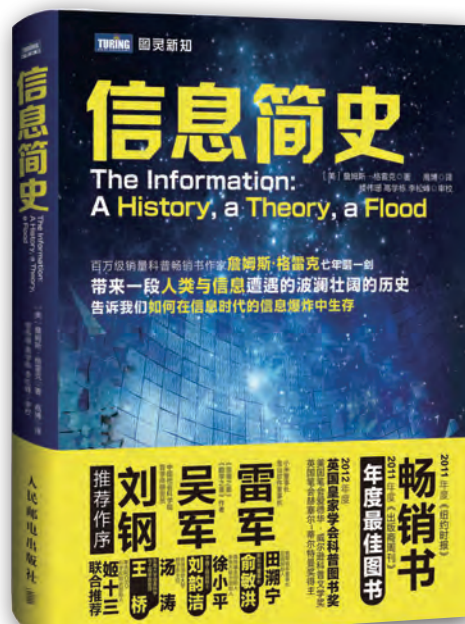
电子书

《码农》杂志

图灵访谈

线下活动

技术交流



- 第一次为信息作史，媲美《时间简史》、《万物简史》的杰作
- 2011 年度《纽约时报》畅销书
- 2011 年度《出版商周刊》年度最佳图书
- 2012 年度英国皇家学会科普图书奖
- 美国笔会爱德华·威尔逊科普文学奖
- 英国笔会赫塞尔-蒂尔特曼奖得主

雷军、吴军、刘钢推荐作序

田溯宁、俞敏洪、徐小平、刘韵洁、汤涛、王桥、姬十三联合推荐

书名：信息简史
 作者：[美] 詹姆斯·格雷克
 译者：高博
 书号：978-7-115-33180-9
 定价：69.00 元

详情请点击：<http://www.ituring.com.cn/book/731>

- 从零开始设计和实现 CPU
- CPU、总线、内存、I/O，组装出你自己的计算机系统
- 超强实践性，从硬件到软件，统统自己动手

《CPU 自制入门》可以帮助软件工程师深入了解硬件与底层，开发出高效代码。硬件工程师可以在本书基础上设计定制硬件，开发高速计算机系统。相信所有读者都可以在本书的阅读过程中，体会到自制计算机系统的乐趣与热情。

详情请点击：<http://www.ituring.com.cn/book/1142>



书名：CPU 自制入门
 作者：[日] 水頭一壽，米澤遼，藤田裕士
 译者：赵谦
 书号：978-7-115-33818-1
 定价：99.00 元

避开那些坑 | Void

Spring Web应用的最大瑕疵

作者 [张龙](#)

众所周知，现在的[Spring框架](#)已经成为构建企业级Java应用事实上的标准了，众多的企业项目都构建在Spring项目及其子项目之上，特别是Java Web项目，很多都使用了Spring并且遵循着Web、Service、Dao这样的分层原则，下层向上层提供服务；不过[Petri Kainulainen](#)在其博客中却指出了众多Spring Web应用的最大瑕疵，请继续阅读看看文中所提到的问题是否也出现在你的项目当中。

使用Spring框架构建应用的开发者很乐于谈论[依赖注入](#)的好处。但遗憾的是，他们很多人并没有在其应用中很好地利用其优势，如[单一职责原则](#)和[关注分离原则](#)。如果仔细看看基于Spring的Web应用，你会发现很多都是使用如下这些常见且错误的设计原则来实现的：

- 领域模型对象只是用来存储应用的数据。换句话说，领域模型使用了贫血模型这种反模式。
- 业务逻辑位于服务层中，管理域对象的数据。
- 在服务层中，应用的每个实体对应一个服务类。

可问题是：如果这种做法很普遍，那为什么说是不对的呢？下面来阐述一下。

旧习难改

Spring Web应用之所以看起来是这个样子原因在于这是人们长久以来的做法，旧习难改，特别是在高级开发者或是软件架构师强制开发人员这样做的时候。问题在于这些人非常擅于捍卫自己的观点。他们喜欢的一个论调就是应用应该遵循关注分离原则，因为它被划分成了几个层次，每个层次都有自己具体的职责。

一个典型的Spring Web应用会有如下几个层次：

- Web层：负责处理用户的输入并向用户返回正确的响应。Web层只会与服务层通信。
- 服务层：作为事务边界。它还负责授权并包含了应用的业务逻辑。服务层管理着领域对象模型并且与其他服务及存储层通信。
- 存储/数据访问层：负责与所用的数据存储进行通信。

关注分离原则的定义是这样的：关注分离（Soc）是一种将计算机程序划分到不同部分的一种设计原则，这样每一部分都会有单独的关注点。虽然一个典型的Sp

ring Web应用也在一定程度上遵循了这个原则，不过实际情况却是应用拥有一个整体的服务层，它包含了太多的职责了。更具体一些，服务层主要有两个问题：

首先，应用的业务逻辑来自于服务层。

这是个问题，因为业务逻辑散落在服务层。如果需要查看某个业务规则是如何实现的，我们需要先找到它才行，这可不是那么轻松的事情。此外，如果有多个服务类都需要相同的业务规则，那么开发人员很可能将这个业务规则从一个服务复制到另一个服务中，这会导致维护的梦魇。

其次，每个领域模型类在服务层中都有一个服务类。

这违背了单一职责原则：单一职责原则表明每个类都应该只有一个职责，这个职责应该完全被这个类所封装。它的所有服务都应该与这个职责保持一致。

服务类存在大量的依赖和大量的[循环依赖](#)。一个典型的Spring Web应用的服务层没有包含只拥有一个职责的[松耦合](#)的服务，它更像是一个紧耦合的大量服务的集合。这使得它很难理解、维护与重用。看起来有点苛刻，不过服务层经常是Spring Web应用最容易出现问题的一环。幸好对我们来说还存在着希望。

推翻

目前的状况并不好，不过也不是完全没有希望的。下面我们来看看如何打破旧有的习惯。

首先，我们需要将应用的业务逻辑从服务层移动到领域模型类中。

为何要这么做呢，看看下面这个例子：

假设我是个服务类，你是个领域模型对象。如果我告诉你从房顶跳下来，那么你是否会拒绝呢？

将服务层的业务逻辑移动到领域模型类中有如下3个好处：

- 根据合理的方式划分代码的职责。服务层会负责应用的逻辑，而领域模型类则负责业务逻辑。
- 应用的业务逻辑只会位于一处。如果需要验证特定的业务规则是如何实现的，我们总是知道该去哪里寻找。
- 服务层的源代码将会变得更加整洁，再不会包含任何复制粘贴的代码了。

其次，我们需要将特定于实体的服务划分为更小的服务，每个服务只有一个目标。

比如说，如果应用有一个服务类，它为与用户帐户相关的人与操作提供了CRUD操作，那么我们就应该将其划分到两个单独的服务类中：

- 第1个服务提供人的CRUD操作。
- 第2个服务提供与用户帐户相关的操作。

这么做有如下3个好处：

- 每个服务类都有一套合理的职责。
- 每个服务类的依赖会更少，这意味着他们不再是紧耦合的庞然大物了。他们是更加小巧且松耦合的组件。
- 服务类更易于理解、维护与重用。

这两个简单的步骤可以帮助我们清理应用的架构，提升开发者的生产力和幸福度。现在，我们想知道如果所有这些都是必要的，那么该何时解决这些问题呢？

有时生命是黑白的

我经常听到有人说我们不应该过多的关注于“架构”，因为我们的应用很小并且很简单。虽然这个论调有一定的正确性，不过我们必须记住[一开始很小的项目最后会变得很大](#)。如果不考虑这种情况，那么一旦发生状况，我们会陷入到巨大的麻烦当中。在未知的水域中航行可不是个好做法，但我们必须要知道，泰坦尼克号在撞到冰山沉没时是在熟悉的航线中航行的。这种事情也会发生在我们的应用中。当事情变得无法控制时，我们必须要有勇气说不。

如果你打算改变，那么我推荐你阅读一下Olivier Gierke所写的“[Whoops! Where did my architecture](#)”（或是观看他在SpringOne2GX上关于这个项目的[演讲](#)）。但请注意，习惯的力量还是很强大的。

原文链接：<http://www.infoq.com/cn/news/2013/11/spring-web-flaw>

相关内容

- [Moden Java Programming with Spring](#)
- [Spring Data与MongoDB：不协调的设计](#)
- [Spring宣布4.0第一个里程碑版本完成](#)
- [Apache Struts 1宣告退出舞台](#)
- [Spring Framework 4.0相关计划公布---包括对于Java SE 8 和Groovy2的支持](#)

避开那些坑 | Void

你应该远离的6个Java特性

作者 [张龙](#)

[Nikita Salnikov Tarnovski](#)是plumbr的高级开发者，也是一位应用性能调优的专家，他拥有多年的性能调优经验。近日，Tarnovski撰文谈到了普通开发者应该尽量避免使用的6个Java特性，这些特性常见于各种框架或库当中，但对于普通的应用开发者来说，使用这些特性也许会给你所开发的应用带来灾难。

我曾花费了无数个小时为各种不同的应用排错。根据过往的经验我可以得出这样一个结论，那就是对于大多数开发者来说，你应该远离几个Java SE特性或是APIs。这里所说的大多数开发者指的是一般的Java EE开发者而不是库设计者或是基础设施开发者。

坦白地说，从长远来看，大多数团队都应该远离如下的Java特性。不过凡事总有例外的情况。如果你有一个强大的团队，总是能够清楚地意识到自己在做什么，那就按照你的想法去做就行。但对于大多数情况来说，如果你在项目的开发中使用了下面这几个Java特性，那么从长远来看你是会后悔的。

这些应该远离的Java特性有：

- 反射
- 字节码操纵
- ThreadLocal
- 类加载器
- 弱引用与软引用
- Sockets

下面对这些特性进行逐个分析，看看为什么普通的Java开发者应该远离他们：

反射：在流行的库如Spring和Hibernate中，反射自然有其用武之地。不过内省业务代码在很多时候都不是一件好事，原因有很多，一般情况下我总是建议大家不要使用反射。

首先是代码可读性与工具支持。打开熟悉的IDE，寻找你的Java代码的内部依赖，很容易吧。现在，使用反射来替换掉你的代码然后再试一下，结果如何呢？如果通过反射来修改已经封装好的对象状态，那么结果将会变得更加不可控。请看看如下示例代码：

```

public class Secret {
    private String secrecy;
    public Secret(String secrecy) {
        this.secrecy = secrecy;
    }
    public String getSecrecy() {
        return null;
    }
}

public class TestSecrecy {
    public static void main(String[] args) throws Exception {
        Secret s = new Secret("TOP SECRET");
        Field f = Secret.class.getDeclaredField("secrecy");
        f.setAccessible(true);
        System.out.println(f.get(s));
    }
}

```

如果这样做就无法得到编译期的安全保证。就像上面这个示例一样，你会发现如果getDeclaredField()方法调用的参数输错了，那么只有在运行期才能发现。要知道的是，寻找运行期Bug的难度要远远超过编译期的Bug。

最后还要谈谈代价问题。JIT对反射的优化程度是不同的，有些优化时间会更长一些，而有些甚至是无法应用优化。因此，有时反射的性能损失可以达到几个数量级的差别。不过在典型的业务应用中，你可能不会注意到这个代价。

总结一下，我觉得在业务代码中唯一合理（直接）使用反射的场景是通过AOP。除此之外，你最好远离反射这一特性。

字节码操纵：如果在Java EE应用代码中直接使用了CGLIB或是ASM库，那么我建议你好好的审视一下。就像方才我提到的反射带来的消极影响，使用字节码操纵所带来的痛苦可能是反射的好几倍之多。

更糟糕的是在编译期你根本就看不到可执行的代码。从本质上来说，你不知道产品中实际运行的是什么代码。因此在面对运行期的问题以及调试时，你要花费更多的时间。

ThreadLocal：看到业务代码中如果出现ThreadLocal会让我感到颤抖，原因有二。首先，借助于ThreadLocal，你可以不必显式通过方法调用就可以传递变量，而且会对这种做法上瘾。在某些情况下这么做可能是合理的，不过如果不小心，那么我可以保证最后代码中会出现大量意想不到的依赖。

第二个原因与我每天的工作有关。将数据存储在ThreadLocal中很容易造成内存泄漏，至少我所看到的十个永久代泄漏中就有一个是由过量使用ThreadLocal导致的。连同类加载器及线程池的使用，“java.lang.OutOfMemoryError:Perm gen space”就在不远处等着你呢。

类加载器：首先，类加载器是个很复杂的东西。你必须首先理解他们，包括层次

关系、委托机制以及类缓存等等。即便你觉得自己已经精通了类加载器，一开始使用时还是会出现各种各样的问题，很可能会导致类加载器泄漏问题。因此，我建议大家还是将类加载器留给应用服务器使用吧。

弱引用与软引用：关于弱引用与软引用，你是不是只知道他们是什么以及简单的使用方式而已？现在的你对Java内核有了更好的理解，那会不会使用软引用重写所有的缓存呢？这么做可不太好，可不能手里有锤子就到处找鼓敲吧。

你可能很想知道我为什么说缓存不太适用使用软引用吧。毕竟，使用软引用来构建缓存可以很好地说明将某些复杂性委托给GC来完成而不是自己去实现这一准则。

下面来举个例子吧。你使用软引用构建了一个缓存，这样当内存行将耗尽时，GC会介入并开始清理。但现在你根本就无法控制哪些对象会从缓存中删除，很有可能在下一次缓存中不再有这个对象时重新创建一次。如果内存还是很紧张，又触发GC执行了一次清理，那么很有可能会出现一个死循环，应用会占用大量CPU时间，Full GC也会不断执行。

Sockets: [java.net.Socket](#)简直太难使用了。我认为它的缺陷归根结底源自其阻塞的本质。在编写具有Web前端的典型的Java EE应用时，你需要高度的并发性来支持大量的用户访问。这时你最不想发生的事情就是让可伸缩性不那么好的线程池呆在那儿，等待着阻塞的Sockets。

现在已经出现了非常棒的第三方库来解决这些问题，别自己写了，尝试一下[Netty](#)吧。

各位InfoQ读者，Java出现至今经历了多次版本更迭，每次也都会有诸多新特性的加入。在日常的Java开发中，你认为存在哪些Java特性是很容易导致问题的呢？作者提到不建议在普通的应用开发中使用反射，不过对于一些框架或库的开发，离开反射实际上是无法实现的，例如Spring、Struts2等框架，那么在一般的Java项目开发中，你觉得哪些地方有使用反射的必要呢？换句话说，如果不使用反射就实现不了功能或是需求。文中作者也不建议使用字节码操纵，实际上一些框架在实现某些功能时是必须要使用的，比如说Spring在实现AOP时就使用了Java的动态代理与CGLib库两种方式达成的。那么对于一般的Java项目来说，哪些地方需要用到字节码操纵呢？欢迎各位读者畅所欲言，一起讨论这些有趣的话题。

原文链接：<http://www.infoq.com/cn/news/2013/11/six-java-features-to-avoid>

相关内容

新品推荐 | Product

在Twitter, Netty 4 GC开销降为五分之一

作者 [Matt Raible](#) , 译者 [马德奎](#)

Netty项目在7月份发布了Netty 4的第一个版本, 其性能的显著提升主要来源于垃圾收集开销的降低。在Twitter, Netty 4经过完善已经获得了5倍的性能提升, 但也有一些代价。

原文链接: <http://www.infoq.com/cn/news/2013/11/netty4-twitter>

Android 4.4 KitKat新特性介绍

作者 [Abel Avram](#) , 译者 [孙镜涛](#)

Google发布了Android 4.4 (KitKat) 特性: 更低的内存占用、沉浸式模式、半透明样式、屏幕打印及一些框架——打印、存储、转换和Chromium WebView。

原文链接: <http://www.infoq.com/cn/news/2013/11/android-4-4-kitkat>

谷歌发布Dart 1.0

作者 [Zef Hemel](#) , 译者 [马德奎](#)

在初次公告并发布预览版两年以后, Lars Bak于11月14日在Devvox比利时会议上宣布了Dart的第一个稳定版本。Dart是谷歌的新Web编程语言和平台, 用于开发现代Web应用程序。

原文链接: <http://www.infoq.com/cn/news/2013/11/dart-10>

Spring Data Neo4j简介

作者 [张龙](#)

Neo4j是一款非常流行的开源图型NoSQL数据库。它完全支持ACID数据库事务属性, 由于其良好的图数据模型设计, Neo4j的速度非常快。对于连接的数据操作, Neo4j的速度要比传统的关系型数据库快1000倍。Spring Data是Spring

的一个核心项目，其下涵盖了如Spring Data JPA、Spring Data MongoDB、Spring Data Redis、Spring for Hadoop等子项目，而Spring Data Neo4j也是Spring Data下的一个重要子项目，它提供了高级的特性以将注解的实体类映射到Neo4j图型数据库上。其模板编程模型类似于我们熟知的Spring模板，为与图的交互提供了基础，此外也用于高级的仓库支持。该项目旨在为NoSQL数据库操作提供便捷的支持。

原文链接：<http://www.infoq.com/cn/news/2013/11/spring-data-neo4j-intro>

TypeScript综述：新功能、工具和路线图

作者 [Jonathan Allen](#)，译者 [马德奎](#)

TypeScript的第一个生产版本已经接近完成，1.0版本之后的功能也已经开始规划。待添加功能列表的第一条是异步/等待，这在客户端和Node.js编程方面将会很有用。

原文链接：<http://www.infoq.com/cn/news/2013/11/TypeScript-0-9-1>

Apigee现在支持Node.js 并开源了Volos

作者 [Abel Avram](#)，译者 [孙镜涛](#)

Apigee Edge现在支持Node.js并且已经开源了Volos，一个包含了一组API管理模块的项目。

原文链接：<http://www.infoq.com/cn/news/2013/11/apigee-nodejs-volos>

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

架构师

www.infoq.com/cn/architect

每月8号出版





宝 典

企业BPM

消除流程孤岛
释放流程潜能

加速目标实现系列丛书

Software AG 制作

封面植物

雪松



雪松，又称香柏，是松科雪松属（学名：Cedrus）植物的统称。由于球果形状相似，与杉树最为接近。原产于喜马拉雅山脉海拔1,500—3,200米的地带和地中海沿岸1,000—2,200米的地带。树高40—50米（也有高60米的），木材带具刺激性的树脂香味，树皮粗糙、有脊状突出，树枝扁平。幼芽分为长（形成树枝，树叶独立地以开放螺旋叶序出现）、短（树叶多长于其上，呈密集螺旋丛生状）两类。叶子常绿，呈针状，长8—60公厘，叶色由亮草绿色到蓝绿色都有，视乎防止水份蒸发的蜡质层厚度而定。球果呈桶状，长6—12厘米，像杉树弓样在成熟时释出长10—15公厘的带翅（长20—30公厘）

种子。内有2—3腺体，能分泌树脂阻止松鼠侵袭。球果需一年时间成熟，每年9—10月受粉，翌年种子成熟。此树是某些鳞翅目幼虫的食物。

1kg.org 多背一公斤

爱自然 | 更爱孩子





架构师 12 月刊

每月8日出刊

本期主编：杨赛

美术/流程编辑：水羽哲

总编辑：霍泰稳

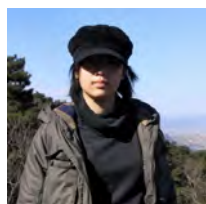
发行人：霍泰稳

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

商务合作：sales@cn.infoq.com

InfoQ 中文站：[新浪微博](#)



本期主编：杨赛

杨赛 (@lazycal)，InfoQ高级策划编辑。写过一点Flash和前端，现在只是个伪码农。在51CTO创办了《Linux运维趋势》电子杂志，偶尔也自己折腾系统。曾混迹于英联邦国家，学过物理，做过一些游戏汉化，练过点长拳，玩过足球、篮球、羽毛球等各类运动和若干乐器。喜欢读《失控》。



《架构师》月刊由InfoQ 中文站出品。

所有内容版权均属 C4Media Inc.所有，未经许可不得转载。