

架构师

12月 ARCHITECT



特别专题

NoSQL，走向成熟

从关系数据库向NoSQL迁移

CouchDB是什么？为什么要关注它？

Julien 谈Apache Nutch 2的特性

注重实效的架构师

深入分析ConcurrentHashMap

Calatrava：构建UI的跨平台移动框架

QA部门将会消亡

消灭神出鬼没的Heisenbug

arrayDB,全新而且简单的PHP ORM库

卷首语

NoSQL，走向成熟

在玛雅预言中的末世之年，整个 IT 业界却似乎可以用“波澜不惊”四个字来概括。没有惊天动地的大新闻来攫取眼球，对于媒体人而言，或许是一种不幸；但对于程序员，却可以减少太多技术选择带来的茫然。没有你方唱罢我登场的喧嚣，没有概念经济的浮夸与炒作，许多技术于是从绚烂归于平淡。

技术由绚烂归于平淡，并不意味着它将落伍于时代，退出历史舞台，而是大浪淘沙炼出真金，乃返璞归真的升华。业界颇有几种技术正是沿着这样的发展脉络不屈地前进，例如本期《架构师》专题讨论的技术 NoSQL。

编者认为，NoSQL 在褪去浮华之后，已经逐渐迈向了成熟。一个标志是用户群的增加。虽然，Martin Fowler 在去年[接受 InfoQ 的采访](#)时，认为：“关系数据库仍然是大多数情况下的选择，至少未来几年是这样的。毕竟，关系数据库产品很成熟，有丰富的支持工具，而且相对来说人们对它们已经有很好地理解了。”但他同样对采用 NoSQL 技术的项目表示信心满满。

可以看到，许多 NoSQL 技术已经在互联网应用以及大数据处理方面露出峥嵘，即使是在企业应用领域，它们也逐渐崭露头角。以文档数据库为例，作为个中翘楚的 MongoDB，就被 SourceForge、MTV 等互联网站广泛使用。而作为全球排名第一的 ERP 软件开放商 SAP，也将 MongoDB 作为其 PaaS 平台的核心组件，用以支持企业内容管理（Enterprise Content Management）。Neo4j 作为图形数据库的领跑者，目前已有很多财富 500 强的企业如 Adobe，Cisco 成为了 Neo4j 的客户。

提及 NoSQL 技术，就不能不提到它与关系型数据库之间的关系，尤其是我们在实际运用中，会面临一个棘手的问题，那就是如何实现关系数据库到 NoSQL 的平滑迁移。很显然，只要支持这一特性，就能更好地将 NoSQL 运用到遗留系统的技术栈迁移场景，拓宽 NoSQL 技术的应用范围。MongoDB 很好地做到了这一点，例如[Server Density](#)就选择了从 MySQL 迁移到 MongoDB。

不仅是 MongoDB 支持这种迁移，Couchbase 支持的特性也不让 MongoDB 专美于前。本期《架构师》的专题文章《[从关系数据库向 NoSQL 迁移：采访](#)

[Couchbase 的产品经理主管 Dipti Borkar](#)》介绍了使用 Couchbase 进行这种迁移的时机、步骤、困难以及选择 NoSQL 的优点。

NoSQL 产品之间的良性竞争也可以从另一个侧面证明 NoSQL 技术的成熟。例如 CouchDB 与 MongoDB 之间的竞争。事实上，CouchDB 这种使用 JSON 作为文档，运用 JavaScript 完成 MapReduce 查询的方式，以及侧重于 Availability（可用性）与 Partition-Tolerance（分区容忍度）的表现（相对而言，MongoDB 更侧重于一致性与分区容忍度），使得它迅速成为了诸多互联网站的新宠。本期专题文章《[CouchDB 是什么？为什么我们要关注它？](#)》非常详细地介绍了 CouchDB。

或许，我们还可以从各种 NoSQL 产品的版本来判断 NoSQL 的成熟度，例如 MongoDB 的版本已经发展到 2.2.2，Neo4j 的版本发展到了 1.9。不过，这种判断未免有些草率。判断成熟度的另一个佐证则是该技术是否产生了大量相关的衍生产品。Martin Fowler 在展望 NoSQL 数据库市场的发展趋势时，就认为“现在的一切都是在添加工具和成熟度，使这些数据库更容易很好地使用。”对于 NoSQL 技术，我们确实看到了这种趋势，如 Apache GORA 与 Spring Data 的出现。Spring Data 提供了访问数据库的统一 API，因为它既能支持关系型数据库，又能支持 REST、大数据以及主流的 NoSQL 数据库，如 MongoDB、Neo4j 等。Apache GORA 则是 NoSQL 数据库的 ORM 框架，提供了各种 NoSQL 数据存储之上的统一前端。专题文章《[Julien Nioche 谈 Apache Nutch 2 的特性及产品路线图](#)》在介绍 Web 搜索框架 Nutch 时，谈到了它与 NoSQL 以及 ORM 框架 GORA 之间的关系。

此外，还有许多创新的 NoSQL 产品的出现，让人眼睛一亮。例如提供不可变数据库服务器的 [Datomic](#)，具备事务管理和部署的特性，并能在云中将数据库作为服务提供给用户。还有 [VoltDB](#)，它尝试在提供可伸缩性的同时使用 SQL 范型。或许它们不能完全划归到 NoSQL 类别中，而应该称之为 NewSQL，甚至是 MoreSQL。[451 Group](#) 的高级分析师 Matthew Aslett 就将 NoSQL 定义为“旨在满足分布式体系结构的可扩展性需求和/或无模式数据管理需求”，NewSQL 则被定义为“旨在满足分布式体系结构的需求，或提高性能以便不必再进行横向扩展”。Alexander Tatiyants 撰写的文章

《[NoSQL No More: Let's double down with MoreSQL](#)》，则掀起了 MoreSQL 的热潮。个人认为，这些技术事实上还是 NoSQL 技术的一种发展。

我们还可以预见 在很长一段时间内 ,会存在关系数据库与 NoSQL 并存的状态 ,
甚至在同一个系统中出现 ,这正是 Martin Fowler 所谓的 [Polyglot Persistence](#)。
这显然是由这两种不同类型的数据库 ,甚至是不同类别的 NoSQL 数据库各自适
应的场景不同导致的必然结果。无论如何 ,种种迹象证明 , NoSQL 技术已经走
向成熟 ,是到了该收获果实的季节了。

本期主编 张逸



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

[卷首语]

[人物专访]

阿里云大赛 20 万大奖获得者江林锦：ECSMate 开发心得	9
---------------------------------------	---

[热点新闻]

Dropbox 可伸缩性设计最佳实践分享.....	12
电商网站的宕机案例分析	16
Facebook 如何提高软件质量 ?	19
为 Web 应用程序提速的 50 条秘技	22
Matz : 如何成为语言的设计者 matz.....	26

[特别专题]

从关系数据库向 NoSQL 迁移.....	40
CouchDB 是什么 ? 为什么我们要关注它 ?	46
Julien Nioche 谈 Apache Nutch 2 的特性及产品路线图.....	76

[推荐文章]

注重实效的架构师——大胆行前人未行之路	80
聊聊并发 (四) ——深入分析 ConcurrentHashMap	87
Calatrava : 自由构建 UI 的跨平台移动框架	96
QA 部门将会消亡	99
消灭神出鬼没的 Heisenbug.....	103
arrayDB , 全新而且简单的 PHP ORM 库	112

[特别专栏]

SpringOne 大会采访 : Mark Pollack 博士专访.....	112
---	-----

采访与书评 : Spring Integration in Action.....118

[新品推荐]

RabbitMQ 3.0 版本有所简化 , 改进了对 STOMP 和 MQTT 的支持	120
Moscrif : 用 JavaScript 进行跨平台移动开发.....	120
Pex : 来自微软研究院的单元测试工具.....	120
ASP.NET 的新特性.....	120
ModelMapper:从对象到对象的映射库	121
Google 发布支持 Java 7 的 App Engine 预览版.....	121
使用 C++/CX 开发 Windows Store 应用程序的注意事项	121
用 Visual Studio 2012 Power Tools 来提高生产力	121
微软开源 Reactive Extensions.....	122
JustDecompile 已支持 C#5 和 WinRT	122
Visual Studio 如何提高 C++ 性能	122

[封面植物]

鹅耳枥.....	124
----------	-----

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 |

人物专访 | Interview

阿里云大赛 20 万大奖获得者江林锦 :ECSMate 开发心得

作者 水羽哲

阿里云今年举办了首届开发者大赛，大赛一共收到了 1000 多份作品，其中 20 件作品最终入围复赛，经过一天的评比，11 月 9 号来自与福州的开发者江林锦（[@江林锦](#)）凭借“ECSMate”获得了阿里[云开发者大赛](#)20 万元超级工具大奖，InfoQ 对他进行了采访。

InfoQ：首先请您做下自我介绍并介绍本次的参赛作品 ECSMate？

江林锦：我是来自福州的开发者/商，本次参加大赛是偶然也是必然。

近两年来一直在关注云计算和大数据的发展，直到今年才正式决定离职创业。我八月初才离职的，正好赶上了阿里云首届的开发者大赛，所以有了ECSMate。ECSMate (ECS 是阿里云弹性计算服务 Elastic Computing Service 的缩写) 是一款用于集中管理阿里云服务器的 Web 管理面板，设计目的是为了降低云服务器使用门槛、提升云服务器管理效率，它通过调用 ECS API 实现服务器级别的管理，通过在云服务器上安装 VPSMate 实现了对各类系统管理功能（软件安装、文件管理、网站管理等）的扩展。

InfoQ：你提到 ECSMate 是使用 Python 的 Tornado Web 框架进行开发的，那么选择这个语言和框架是基于哪些考虑？其中你觉得这个框架还不尽理想的地方是什么？

江林锦：ECSMate 使用 [Tornado Web 框架](#)（以下简称 Tornado）进行开发，主要是考虑三点：

1. 基于 Python，大部分发行版都有默认安装，而且能够与系统较好地进行粘合
2. Tornado 自带了一个 HTTP Server，无须另行安装配置 HTTP Server 环境
3. Tornado 的异步处理机制异步处理在 ECSMate 中发挥了重要作用，API 接口的调用和返回、系统操作、软件安装，都是异步过程，借助 Tornado

的异步处理机制，能够避免在执行这些操作过程中发生阻塞，很好地提升了用户体验。

除了 Tornado ,ECSMate 的快速成型还需要感谢 [Bootstrap](#) 和 [AngularJS](#)。ECSMate 的设计使用前后端完全隔离的方式，后端使用 Tornado 驱动，封装好各类 API。而前端则是采用 Bootstrap 和 AngularJS 构建所有 UI 逻辑。

坦白说我使用 Python 的时间并不是太长，不敢对 Tornado 妄作评价。过去几年主要还是使用 PHP 进行开发，Python 只是简单地接触，还没有正式用于生产项目编码，在 ECSMate 开发过程中其实是边上手 Python 边开发的。Tornado 提供的功能已经足够满足 ECSMate，如果非要找不足，恐怕只能说是个个人习惯问题了。

InfoQ：你认为在云时代，工具类应用将会主要有哪些需求？

 **江林锦**：工具类应用这个名词很好，其实在每个时代，工具市场都是非常旺盛的，加上不断开放化、平台化的趋势，从事工具类应用的开发能让小团队乃至个人非常容易地从这个市场中分一杯羹。

在电商时代，有大量围绕电商开发的各类工具，这个我们可以在淘宝开放平台上找到许多优秀的产品和开发商。

在云时代，也将会有大量围绕云平台开发的各类云周边工具和各类上层应用，国外已经有大量成功的云工具开发商。

结合云的特点和国内市场形势，除了 ECSMate 这类服务器/集群管理工具外，个人还看好以下两个方向：

- 迁移工具：传统服务向云的迁移、云际迁移
- 数据分析：越来越多的行业将通过云进行数据分析和决策

实际上好好对比下云与非云的差别，很容易就能发现更多机会的。

InfoQ：请分享一下 ECSMate 中让你比较得意的技术实现？

 **江林锦**：除了前后端完全隔离和异步处理，ECSMate 技术上其实并没有太多值得称道的地方，参赛中有许多作品在技术上要优秀得多。

几年前，也许我还会为一个厉害的技术实现而疯狂，现在的我更关心市场和用户的需求，关心的是我需要在多久时间内对市场做出响应，关心的是用户对产品的需求，我需要更多地在市场和技术之间做出权衡地选择。

InfoQ：在开发的过程中，你用到了阿里云的哪些服务？请列举这些服务的优点和需要改进的地方？

江林锦：ECSMate 主要是通过调用 ECS API 和扩展系统管理功能实现对云服务器的管理。

目前阿里云可能是出于安全、产品功能或财务结算的考虑，API 文档中仍有许多接口的权限还未开放，但是我相信离开放的日子不远了。

不过目前阿里云仍缺少一些我觉得必要的 API 的功能，如：

- 权限类：限定各个 AccessKey 的权限，实现通过一个 Key 管理单台或多台服务器而不是全部服务器。
- 日志类：系统启动日志、带宽日志、磁盘 IO 日志等，方便用户更好地跟踪追溯云服务器的使用情况。

InfoQ：请你分享一下获奖的感受？

江林锦：其实在参赛前我就已经做好了大赛后的计划，不论获得的是哪个奖项。能够赶上这次大会结识许多业内的朋友已经是很幸运的事情，而又幸运地赢得了这个奖项，缓解了我当前不少的资金压力。

不过目前更大的挑战摆在我的眼前：如何利用好这笔为数不多的资金继续我的创业进程？如何维系 ECSMate 的发展？

InfoQ：ECSMate 下一步的发展计划是什么？

江林锦：ECSMate 仍将定位于为云服务器用户提供易于使用的管理工具，今后 ECSMate 将成为一套云服务器综合管理平台，你可以通过它快速创建和管理你的各类服务器，而不仅限于 WEB 服务器，对各类 Linux 发行版的支持也将会不断完善，同时也会不断跟进与整合阿里云的各类云服务，完善和扩展更多系统管理功能。

热点新闻

Dropbox 可伸缩性设计最佳实践分享

作者 [李洋](#)

Dropbox 的运维工程师 Rajiv，跟大家分享了可伸缩性设计的最佳实践第一讲。众所周知，Dropbox 是一款非常易用的网络存储云端产品，现已达到 40,000,000 的用户。令人惊奇的是，Dropbox 公司对于服务器集群的运维人员投入在一到三个人。Rajiv 就系统的可伸缩性设计，尤其在资源有限、流量快速增长的情况下，将最佳实践分享给大家。

Run with extra load(通过额外加载发现系统故障)

在生产环境最常用的一个技巧就是，人为制造一些额外的数据进行加载。举个例子，生产环境常常针对缓存服务器进行额外的数据读取加载(Memcached Read)，如果缓存服务器 down 机，运维工程师就可以马上切断流量进行故障排查。为什么不能事先计划好，而是采用一种“通过加流量试错”的方式发现系统的问题呢？答案是：从长时间积累的运维经验来看，引起系统故障的原因范围太大，而且常常是突发性的，无法通过监控实时捕获。另一点需要注意的是，尽量不要模拟数据写入加载，这会破坏生产环境的数据一致性以及导致无法控制的锁竞争。

App-specific metrics(面向具体应用的度量图表)

针对特定应用，汇总不同类目集群所定制的数据统计图表变得越来越重要。一般现有的图表监控都是针对具体某一类数据(CPU, Mem, IO 等)，将各个维度的统计数据汇总到一个图表的需求很迫切。Dropbox 的解决方案是充分利用 memcached(缓存服务器)，cron(后台计划任务管理)以及 ganglia(集群监控管理)三方的功能：每当有统计数据是我们想要的，会将该数据存到线程安全的内存块里面，每秒钟内存块里面的数据都会发送到缓存服务器，以时间戳作为 Key 存储。每分钟，缓存服务器的统计数据会被清除、汇总、发送到集群监控服务器。举个实际的例子，下面是生产环境中最好用的图表：

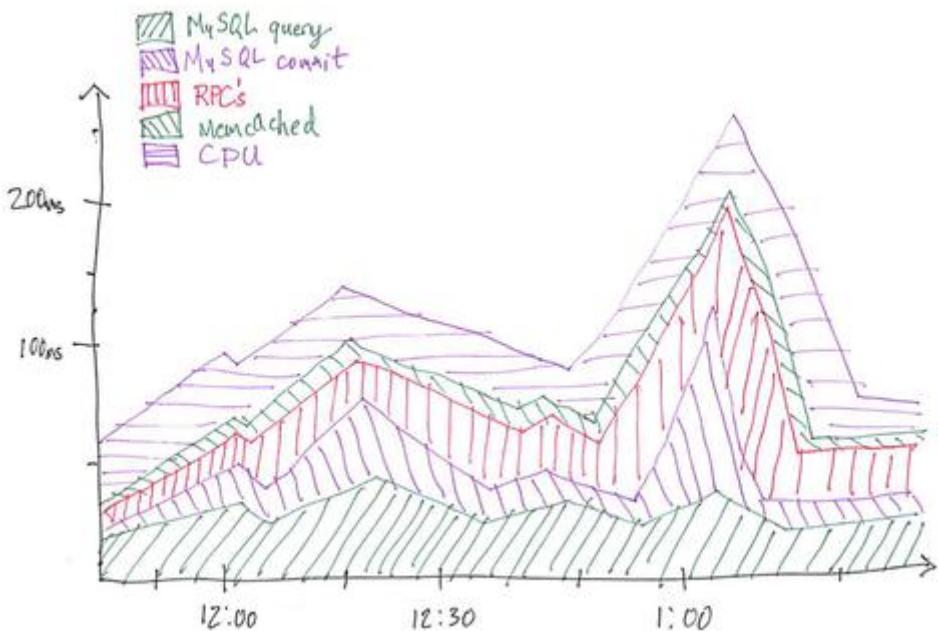


图 1：系统响应时间度量图表

横轴是时间段，纵轴是站点服务器响应时间，响应时间分五段(MySQL Query, MySQL Commit, RPC' s, Memcaced, CPU)，我们很容易看出在 1:00 左右，有根很长的刺，由 MySQL Commit 所导致。实际图表响应时间不仅仅只有五段，很多因素被剔除了，都包含在“CPU”里面。

Poor man' s analytics with bash(善于利用 bash 脚本分析系统问题)

熟练运用 bash shell 可以很大程度提高工作效率，举个例子，现有服务器端的日志，想从中提取最近某个时间段内流量的波峰是否超过阈值，当然现有的服务器图表可以满足大部分的需求，但很多系统的数据实时性不高(一到五分钟更新一次数据)。面对 ad hoc 需求，可以使用如下脚本：

```

Apr 8 2012 14:33:59 POST ...
Apr 8 2012 14:34:00 GET ...
Apr 8 2012 14:34:00 GET ...
Apr 8 2012 14:34:01 POST ...

cut -d' ' -f1-4 log.txt | xargs -L1 -I_date +%s -d_ | uniq -c | (echo "plot '-' using 2:1 with
lines" ; cat) | gnuplot

```

上述命令会将现在的系统状况以图形化的形式表现出来。

Log spam is really helpful(垃圾日志还是帮得上忙的)

垃圾日志并不是说一无是处，很多时候，它帮了我们大忙。垃圾日志其实可以作为跟踪代码的一种方式，举个例子，一段代码逻辑如果在正常情况下会打印“FUUUUCCKKKKAsdjkfnff”的字符串，那么在出问题的时候我们可以根据日志找到问题的源头。日常的运维工作常常会维护两份日志文件，一份干净的日志文件，一份填充着垃圾日志的文件，它常常会不经意的帮到我们。

Keeping a downtime log(记录故障日志)

记录故障日志是一个很好的习惯，记录故障的开始、结束时间以及故障的原因，过后通过分析故障日志我们可以很客观的判断如何最小化故障发生的时间。每个故障发生的原因不尽相同，而每个故障的解决方案也不一样，将故障详情记录下来确实是一个明智之举。

UTC(使用世界标准时间)

一定要使用世界表尊时间，不管是服务器时间还是数据库时间。很多系统对于非 UTC 时间的处理很不好，这会带来很多问题，切记在呈现给用户数据之前做时区转换。

Technologies we used(我们使用的技术)

很多朋友对于 Dropbox 使用什么技术非常感兴趣，我们来列举下我们使用的技术：

- 1) 开发语言使用 Python
- 2) 数据库 MySQL
- 3) Web 框架 Paster/Pylons/Cheetah
- 4) 亚马逊云服务 S3/EC2 存储文件
- 5) memcached 用来减小数据库的压力，以及用来处理内部系统的协作
- 6) ganglia 做集群监视管理以及生成定制化监控报表

- 7) nginx 做前端服务器
- 8) haproxy 做 web 服务器负载均衡
- 9) nagios 做内部服务器的健康检查
- 10) Pingdom 做外部服务的监控
- 11) GeoIP 用来将 IP 转化为地理位置

上述选择的技术是业界标准，没有什么新奇，最大的原因是可靠、风险低。即便是 memcached 这么通用的技术，都存在一些令人头痛的 bug，选择太新的技术会让事情变得复杂。我们对于技术选型的建议是，尽量选择轻量级的、业界知名公司都在使用的技术，当然也可以花时间成为该项目的“技术先驱”。

The security-convenience tradeoff(系统安全和用户体验的权衡)

为了保护用户文件信息，系统安全对于 Dropbox 非常重要。但增加安全性，不管对于用户还是程序员，都会造成不方便，举个例子：很多网站在用户输入错误的用户名或者密码的时候，提示说您有一项输入错误，但不会告诉你具体是哪个。这种安全性策略，大大减小了破解用户名、密码的概率，但对于那些在不同网站都使用不同用户名的用户，会带来一些麻烦。

在内部服务器集群设置防火墙，是一个很棒的想法，但在实践中，对于互相没有通信的服务器集群可以省略此步骤。最常用的隔离，一般针对的是搭建在内部网络的第三方论坛，它更容易遭到攻击。

我们的安全策略可能有些争议，但系统安全都是对用户行为的假想和抽象，很多系统甚至是银行系统，都存在很大的安全隐患，所以在对系统做安全性投入的时候，先想一想这些工作是不是那么的必要。

对于运维技术感兴趣的读者朋友，可以关注 Rajiv 的博客，或者通过邮箱 rajiv.eranki@gmail.com 与作者联系。

原文链接：

<http://www.infoq.com/cn/news/2012/11/dropbox-scale-bestpractice>

电商网站的宕机案例分析

作者 [崔康](#)

性能调优社区 dynatrace 在其博客中分享了客户案例，电商网站在假日客流峰值期间数次崩溃，经过 SQL 优化和调整负载均衡算法解决了相关问题，值得读者借鉴。

按照博文的描述，该电子商务网站在圣诞节期间崩溃了七次，每次宕机的时间都超过 5 个小时。这种情况让企业损失了大量的收入和名誉。

我们的一个客户就曾经遭遇到这种情况，我们在此分享下他们的经历。宕机的原因有很多，不过我们这里只强调比较突出的一点，即负载均衡器在采用 Round-Robin (轮询) 算法时比 Least-Busy (最休闲) 算法更容易导致应用服务器因堆内存耗尽而崩溃。

在分析性能问题之前，先看一看该网站的拓扑结构。该电子商务网站部署在 6 个 Tomcat 应用服务器上，前面是 3 个 Apache Web 服务器。

面临的问题是，在负载高峰时刻，每个 Tomcat 实例处理的响应时间开始增长，等待队列中的请求数目增多。一段时间之后，这些 Tomcat 实例由于 OutOfMemory 异常而崩溃，随后其他实例也因为承受不住由此增加的负载而宕机。

“即使在采用平均分布负载的算法 (Round Robin) 下，有些 Tomcat 应用服务器也会出现响应时间的跳跃。一旦服务器开始拒绝客户连接，我们会发现一些连锁反应。数据库层出现大量的异常，同时应用层之间会抛出异常，Web 服务器会返回给浏览器 HTTP 500 的错误。

比如，在实际的分析过程中，我们发现某一个 Tomcat 实例在 30 分钟内对 43000 个页面返回了 HTTP 500 错误。

原因分析：来自数据库层 (JDBC) 的异常是分析该问题的关键入口。仔细查看这些异常会发现连接池已经耗尽了，从而导致应用的各个部分出现问题。由于连接池的原因，每一个请求都需要平均等待 3.8 秒来从池里获取连接。

不光是连接池大小的设置问题，而且不少低效的数据库语句在执行应用的一些业务逻辑事务时花费了太长时间。这导致应用服务器维护这个连接的时间也较长。

如果把负载均衡器设为 Round Robin 算法，应用服务器会继续获得其他的请求。最终，由于客户请求的随机性，某个应用服务器收到了不少执行低效数据库处理的请求。一旦连接池耗尽，应用服务器就开始抛出异常，并最终导致 JVM 崩溃。一旦第一个应用服务器出现问题，用不了多久其他服务器也挂掉了。

解决办法：优化应用程序和负载均衡器

首先要分析执行最慢的数据库语句，并做性能优化，比如增加索引等。同时也优化了连接池大小来满足高峰时刻的需求。然后，企业把负载均衡器的算法从 Round-Robin 改为了 Least-Busy，在生产环境中这个配置经常被人遗忘。自从对应用程序和负载均衡器做了修改之后，网站再也没有崩溃。

该企业之前做过负载测试，但是存在两个问题：

1. 没有使用预期的峰值负载长时间测试。
2. 没有完全模拟用户行为，测试脚本太少、太简单，缺少了用户的高交互性访问场景。

由此等到的经验教训：在生产环境的低访问量时段（凌晨 2 点到 6 点）执行负载测试，这样可以虽然有小的交易风险，但是可以避免大的经济损失。高峰时间长为 10 小时，不要测试太短时间。

关于负载均衡的基本算法，主要有以下几种（[参考 F5 产品](#)）：

- 随机：负载均衡方法随机的把负载分配到各个可用的服务器上，通过随机数生成算法选取一个服务器，然后把连接发送给它。虽然许多均衡产品都支持该算法，但是它的有效性一直受到质疑，除非把服务器的可运行时间看的很重。
- 轮询：轮询算法按顺序把每个新的连接请求分配给下一个服务器，最终把所有请求平分给所有的服务器。轮询算法在大多数情况下都工作的不错，但是如果负载均衡的设备在处理速度、连接速度和内存等方面不是完全均等，那么效果会更好。
- 加权轮询：该算法中，每个机器接受的连接数量是按权重比例分配的。这是对普通轮询算法的改进，比如你可以设定：第三台机器的处理能力是第一台机器的两倍，那么负载均衡器会把两倍的连接数量分配给第 3 台机器。

- 动态轮询：类似于加权轮询，但是，权重值基于对各个服务器的持续监控，并且不断更新。这是一个动态负载均衡算法，基于服务器的实时性能分析分配连接，比如每个节点的当前连接数或者节点的最快响应时间等。
- 最快算法：最快算法基于所有服务器中的最快响应时间分配连接。该算法在服务器跨不同网络的环境中特别有用。
- 最少连接：系统把新连接分配给当前连接数目最少的服务器。该算法在各个服务器运算能力基本相似的环境中非常有效。
- 观察算法：该算法同时利用最小连接算法和最快算法来实施负载均衡。服务器根据当前的连接数和响应时间得到一个分数，分数较高代表性能较好，会得到更多的连接。
- 预判算法：该算法使用观察算法来计算分数，但是预判算法会分析分数的变化趋势来判断某台服务器的性能正在改善还是降低。具有改善趋势的服务器会得到更多的连接。该算法适用于大多数环境。

有关电商网站在假期峰值解决方案的经验分享，InfoQ 中文站以前曾专门采访过淘宝的几位专家：

- 淘宝双十一事件的前中后技术事
- 江枫谈淘宝“双十一”事件中的数据库架构优化
- 伏威谈淘宝网的高并发处理与压力测试

国内一年一度的光棍节电商促销又要开始了，开发和运维人员将面临巨大的挑战。有关技术分享，InfoQ 中文站将持续关注。

原文链接：

<http://www.infoq.com/cn/news/2012/11/ecommerce-load-balance>

相关内容：

- [开源应用架构之 Selenium WebDriver（中）](#)
- [开源应用架构之 Selenium WebDriver（上）](#)
- [腾讯入股艺龙，在线旅游市场引发关注](#)

Facebook 如何提高软件质量？

作者 郑柯

[刘彪](#)是微软测试技术团队的一名软件设计工程师，他在自己的博客上分享了 Facebook 如何提高软件质量的原则、手段和背后的原因。

在文章开头，刘彪指出：

“虽然 facebook 已经早已不是创业公司，但是不难看出它在产品研发和质量控制仍然保持着创业公司的风格。”

他提到，Facebook

“以小的研发团队为核心，遵循几个非常重要的原则：

- Be there from start to ship: 每个工程师自始至终负责产品。从最开始的一个想法，到开发原型，到内部审核，反馈，到产品开发，上线和维护，全部有工程师自己搞定。
- Show work early and often: facebook 非常看重反馈，尤其早期内部反馈。他们鼓励工程师有了想法后，尽快开发出原型，尽快得到反馈。
- Gets your hands dirty: 动手去做，去实现。
- Don't fall in love: 互联网产品是不断变化的，不需要等到把一个产品设计的很完美了才发布。

接下来，他举出了 Facebook 为了遵循上述原则而采取的质量控制机制：

“● 开发对质量负责：开发从设计，实现，测试，到部署都要自己做。其它做工具，流程的工程师通过开发工具和流程来帮助开发人员更为简单方便地做测试，做部署和做监控。每个开发人员有自己单独的测试环境，测试环境就是运行在开发本地机器上，部署非常简单快速。测试环境用的是真实的用户数据。

● 持续集成和测试自动化：每周发布一次。星期天晚上，要发布的构建从主线上分支出来到发布分支，到星期二的中午如果没有大的问题，就可以上线了。所有的测试运行控制在 10 分钟以内，所以不需要考虑不运行哪些测试用例。运行所有测试用例。（只是听说，没有经过考证。）

- 内测（dog food）：发布之前，公司员工使用要发布的功能。2 - 3天之内可以有几百个或上千个人在使用新功能。负责要发布功能的开发人员在星期天晚上到星期二中午之间会做大量的测试。
- 发布风险控制：新功能本身质量可能有问题，新功能也可能影响其它现有功能。为了减少或控制这些风险。Facebook 开发了一整套完善的发布，控制，监控流程和工具。做到：1. 测试通过后，产品质量基本有保证。2. 即使有漏测的 bug，只会影响很少量的用户。3. 及时监控到问题。4. 及时修复。
- 产品监控：监控产品的系统的运行状态。

刘彪指出：



Facebook之所以采取这种质量控制策略和它的产品特点密切相关：

1. 用户对社交产品质量的容忍度相对较高。比如发微博，现在连不上，等一会在连接也可以，现在发布不出去可以等一会再发，粉丝数量统计有误，没有人太关心。其实 Facebook 并不认为自己的质量差。他们认为产品的质量高低不是有多少个 failed 测试用例，有多少个 bug 来确定的，而是有用户对质量的期望值来决定的。如果用户对产品质量的期望值很高很高，一个 bug 漏掉了都会照成质量差的印象，用户很有可能放弃使用。相反，如果用户的期望值一般，100 个 bug 漏掉了都不会影响用户继续使用。所以 facebook 产品发布的条件是满足用户对质量的期望值即可。
2. 相对宽松的产品发布周期。不像微软或 google 很多产品已经在市场上，用户对下一版本的发布时间和新增加功能的期望很高，这往往给产品开发组的压力很大。Facebook 基本没有这个问题，它有适合自己的发布期限，不用受到外界干扰。
3. 产品发布和监控流程比较完善，即使有漏测的 bug，对用户的影响可以控制在最小而且可以及时发现及时修复。

“没有专职测试工程师” 刘彪认为这是 Facebook 质量控制中引以为豪而且倍受瞩目的的一点：

1. 什么是“专职测试工程师”？头衔里面有“测试”的工程师？专门找bug的工程师？专门做质量控制的工程师？等等。
2. Facebook 的确没有带“测试”头衔的工程师，也没有专门运行产品找bug的工程师。每个人都是开发工程师。但是他们的实际工作有区别，有的专门做面对用户的产品，有的专门做测试，开发工具，有的专门做产品的构建和持续集成工具和流程，有的专门做发布和监控的工具和流程。如果按照传统意义上的开发和测试的划分的话，除了第一类外，其他都可以看做专职测试工程师。
3. Facebook 不是惟一一个没有带“测试”头衔工程师的公司，很多软件公司都没有，比如 Twitter。
4. 很多人把专职测试工程师指专门运行产品找 bug 的工程师。微软在 2005 年去掉 STE (software test engineer) 岗位，就已经没有这一类型的专职测试工程师了。

对于专职测试工程师，刘彪的看法是：

“专职测试工程师是个非常模糊的结论。尤其现在我们对产品质量控制方法的不断演变和提高，“测试”的概念不仅仅是指找 bug 了，所有围绕提高产品质量的工作都是测试。头衔上有没有“测试”不重要，有没有“测试”岗位不重要，重要的是如何有效保证和提高产品质量。”

刘彪曾在多个技术大会上做过微软相关开发技术和流程的演讲，同时也主编了《[详解 Windows Azure 云计算平台](#)》一书，在他的博客中还分享了[Amazon](#)、[Google](#) 和[微软](#)如何提升软件质量，感兴趣的读者可以移步一观。

原文链接：

<http://www.infoq.com/cn/news/2012/11/Facebook-on-software-quality>

相关内容：

- [500TB——Facebook 每天收集的数据量](#)
- [Facebook 元老王淮谈科技公司应有的工具文化](#)
- [Facebook 是如何发布代码的](#)
- [Facebook 已将 HHVM/JIT 用于其开发和产品中](#)
- [Facebook 使用 Corona 提升 Hadoop 的可伸缩性](#)

为 Web 应用程序提速的 50 条秘技

作者 Abel Avram 译者 臧秀涛

Jatinder Mann 是微软 Internet Explorer 产品的一名项目经理，在 BUILD 2012 大会上，他做了题为 “提高 HTML5 应用和网站性能的 50 条秘技 ([50 performance tricks to make your HTML5 apps and sites faster](#))” 的演讲，介绍了很多为 Web 应用程序提速的技巧。

Mann 的建议是按照下面六个原则组织的。

1. 快速响应网络请求。

- 避免重定向。排名前 1000 的网站中，63% 使用了重定向。如果不执行重定向的话，页面速度可以提高 10%。
- 避免 [Meta-refresh](#)。世界上 14% 的 URL 使用了 Meta-refresh。
- 尽可能通过 CDN 定位用户，使服务器响应时间最小化。
- 从不同的域下载资源，使并发连接的应用最大化。
- 复用连接。不要在响应请求时关闭连接。
- 确保页面加载不会因合作伙伴网站提供的数据而延迟。
- 了解耗时的网络组件，如重定向、缓存、DNS、请求和响应等。在 IE 9 和 10 中可以使用 [Navigation Timing API](#) 来测量浏览器花在每个操作上的时间。

2. 最小化下载的字节数。

- 加载页面时，要尽量减少下载的数据量。平均而言，每个页面要下载的数据量达 777KB，其中有 474KB 的图片、128KB 的脚本和 84KB 的 Flash。
- 请求 gzip 压缩的内容。
- 将资源保存在本地的包中，比如为 Windows 商店应用生成的包资源索引 ([Package Resource Index](#)) 文件。这样当需要这些资源时就可以很容易地获取到。

- 使用 HTML5 App Cache 缓存动态资源。App Cache 会只下载一次资源，从而避免多次网络行程。当应用的版本号发生变化时，它会自动重新下载相应资源。
- 尽量在响应中使用“Expires”字段来提供可缓存的内容。
- 通过设定请求的 `If-Modified-Since` 字段来使用条件请求。
- 缓存数据请求，如 HTTP、XML 和 JSON 等，因为大约 95-96% 的请求不会整天变化。虽然这个想法很合理，但实际缓存接收到的请求的网站所占比重还不到 1%。
- 用大写将文件命名标准化。虽然服务器可能把 Icon.jpg 当作 icon.jpg，但是对于 Web 平台而言，它们是不同的资源，对应不同的网络请求。

3. 高效地组织标记。

- 对于 IE 而言，请使用最新的标记标准，因为它速度最快。IE 10 也能识别早期的 IE6-IE9 标记风格，但是其速度不如新的标记风格。
- 特定的业务 Web 应用可能需要强制 IE 运行于传统模式，请使用 HTTP 头字段“X-UA-Compatible: IE=EmulateIE7”来代替 HTML 标签，这样速度会快一些。
- 为了平滑地渲染，样式表应该链接在页面顶部的<head>之中的<title>后面。
- 绝对不要在页面底部链接样式表。否则加载页面时可能会出现闪烁。
- 对于分层样式，不要使用“@import”，因为它是同步的，会阻塞 CSS 数据结构的创建和屏幕绘制。
- 避免样式的嵌入和内联，因为它会强制浏览器在 HTML 和 CSS 解析器之间进行上下文切换。
- 仅包含必要的样式。不要下载和解析用不到的样式。
- 仅在页面底部链接 JavaScript。这可以确保脚本执行时图片和 CSS 等资源已经加载，无需等待，也避免了上下文切换。
- 不要在页面开头链接 JavaScript。如果某些脚本必须在开始处加载的话，请使用“defer”属性。
- 不要内联 JavaScript，这样可以避免上下文切换。
- 使用“async”属性加载 JavaScript，这样整个脚本就可以异步加载和执行。
- 避免冗余代码。世界上 52% 的网页包含 100 行甚至更多的冗余代码，比如一个 JavaScript 文件被链接了两次。

- 将一个 JS 框架标准化 ,无论是 jQuery ,Dojo ,Prototype.js 还是其他框架。浏览器没有必要加载多个功能基本相同的框架。
- 不要加载 FB 和 Twitter 等网站的脚本 ,只是看起来很酷而已 ,它们会争用资源。

4. 优化多媒体资源的使用。

- 图片是最常用的资源 ,每个页面平均会下载 58 张图片。
- 尽量避免下载太多图片 ,根据页面加载时间将图片最大数量控制在 20-30 之间。
- 使用 Image Sprites 将多个图片组合成一个。该技术可以减少网络连接数 ,也会减少下载的字节数并节省 GPU 处理周期。
- 手动创建 Image Sprites ,因为工具创建的可能会留下较大的空洞 ,这会加大需要下载的数据量 ,也需要更多的 GPU 处理周期。
- 使用 PNG 格式的图片 ,该格式在下载大小、解码时间、兼容性和压缩率之间实现了完美的折中。JPEG 格式可以用于照片。
- 使用原始的图像分辨率 ,这样可以避免下载不必要的数据以及缩放所需的 CPU 处理。
- 尽可能使用 CSS3 Gradient 替代图片。
- 尽可能使用 CSS3 border radius 替代图片。
- 使用 CSS3 Transform 来创建移动、旋转和倾斜效果。
- 对于小型的单个图片 ,可以使用 [Data URI](#)。这样可以节省一张图片的下载量。
- 避免复杂的 SVG ,因为它们会延长下载和处理时间。
- 当包含 HTML5 时 ,指定一个预览图片。这样浏览器就不必下载整个视频文件来确定预览图片了。
- 使用 HTML5 来代替 Flash、Silverlight 或 QuickTime。HTML5 速度更快 ,而且其他几种形式的运行时插件会消耗系统资源。
- 主动地以异步方式下载富媒体资源并将其保存在 App Cache 中。

5. 编写快速的 JavaScript。

- 在 JavaScript 中进行数学运算时尽量使用整型。JavaScript 的浮点运算比相应的整型运算耗费的时间要多得多。在进行数学运算，特别是计算密集型运算时，请使用 Math.floor 和 Math.ceil 将浮点数转换为整型数。
- 降低 JavaScript 代码量，这样不但可以减少下载的数据量，而且能够提供更好的运行时性能。
- 按需初始化 JS。当需要时动态加载 JS。
- 通过缓存变量（如 document 和 body 等）使 DOM 交互减到最少。
- 使用内置的 DOM 代码，如 element.firstChild 或 node.nextSibling 等。这些代码都是高度优化的，相对于第三方库能提供更好的性能。
- 访问大量 DOM 元素时，使用 querySelectorAll。
- 使用.innerHTML 来构建动态页面。
- 批量标记更改。
- 维护小巧而健壮的 DOM——将其元素数目控制在 1000 以内。.
- JSON 快于 XML。
- 使用浏览器的 JSON 原生方法。
- 不要滥用正则表达式。

6. 知道你的应用在做什么。

- 理解 JavaScript 定时器，了解 setTimeout 和 clearInterval。除非确定要使用定时器完成一些功能，否则不要启动定时器。组合定时器也是如此。
- 如果监视器的刷新率是 60Hz，请将显式帧的定时器调整为 16.7 ms。
- 在 IE 10、Chrome 和 Firefox 中，图形处理请使用 requestAnimationFrame 动画函数。其绘制通过回调实现，因此不需要定时器。
- 使用可见性 API (document.hidden 和 Visibilitychange) 来确定应用程序的可见状态，如果页面是隐藏的，就关闭该活动。这样可以节省 CPU 和电池寿命。

Mann 建议在 IE 中使用 [Windows Performance Tools](#) 来测试 Web 页面的性能，并以减少 CPU 时间和增加并发性为目标进行优化。

原文链接：

<http://www.infoq.com/cn/news/2012/11/Browser-Web-App-Performance>

Matz：如何成为语言的设计者 matz

作者 [丁雪丰](#)

在今天举行的 [RubyConfChina 2012 大会](#)上，Ruby 语言之父松本行弘 (Yukihiro "Matz" Matsumoto ,@yukihiro_matz)第四次来到中国参加大会，为大家带来了一场名为《Be a language designer》的主题演讲，分享了作为语言设计者的心得，还带来了 [mruby](#) 与 [Ruby](#) 2.0 的最新信息。

Ruby 诞生于 1993 年 2 月 24 日，Matz 只是出于兴趣，对编程语言的热爱，开发了 Ruby，当时是一个 Unix 的脚本语言，但是，当时很多人对 Ruby 并不感冒，因为已经有了 Perl。Matz 并没有放弃，因为他相信只有不断挑战才能成功：

Only chanllege again and again will bring you the success.

他提到了三个能在 IT 时代生存下来的角色，其中之一就是语言的设计者，他在场做了简单的调查，与会者中没有语言的设计者，仅有位语言的实现者。Matz 例举了一些知名语言的发明者——Perl、Python 等等，有兴趣的人可以去访问 language inventor or serial killer? 这个网站。他告诉大家，**其实开发一门语言并没有大家想象的这么难**。每个开发者平时设计的代码、API 和接口其实都是一门“语言”。语言就是用来进行交流沟通的，他举了个例子，Ruby 甚至能用于人与人的交流。

人人都能成为语言的设计者，只要关心你身边的人，关心你自己，关心你的未来。Ruby 的热心布道者 Dave Thomas 说过：

Programming is a process of designing your own dsl.

因此，**如果想要设计一门语言，不妨先从 DSL 写起**，用它来构建你的应用程序，从内部 DSL 写起，慢慢发展为外部 DSL，这成就了著名的 Ruby on Rails。

作为 Ruby 之父，Matz 亲手重新实现了 Ruby，这就是 mruby。mruby 可以算是 Ruby 的一个子集，它在语法上完全兼容 Ruby 1.9，非常小巧精悍(<500K)，适用于嵌入式环境，提供了软实时，能够运行于小型设备(比如 SAKURA Board) 中，甚至嵌入在别的语言或者应用之中。Matz 介绍了几个与 mruby 相关的项目：

- MobiRuby——mruby for iOS

- mruby for Android 仍在开发之中
- mod_mruby——Apache mruby Extension
- mruby_nginx——Nginx mruby Extension
- mruby-libuv——适用于 mruby 的异步 I/O

mruby 的应用非常广泛，能被用于各种智能设备上，比如能够被用于太阳能面板控制器、智能网络路由、自动售货机控制器甚至汽车上，它同样能用来开发游戏和编辑器。Matz 表示，如果有人用 mruby 开发了一款编辑器，那么他原因抛弃使用多年的 emacs，使用这款编辑器。

2013 年是 Ruby 诞生 20 周年的日子，在日本 20 岁算是成年，因此 Ruby 2.0 将于 2013 年 2 月 24 日发布。从 2001 年第一次公开讲到 Ruby 2.0，10 年后，它终于要与人们见面了。此次 Ruby 2.0 带来了大量重大变化，他重点例举了以下四个：

第一，Keyword Formal Argument

```
defownto(from, to, step:n )
...
End
```

第二，Enumerable#lazy（下例中使用了 lazy，不会消耗太多内存）

```
(1..Float::INFINITY).lazy.map {|i|
  i.to_s
}.select { |s|
  /3/ === s
}.first(5)
```

第三，Module#prepend

```
class Foo
  def foo; p :foo; end
end
```

```

module Prepend

  def foo

    p :before

    super

    p:after

  end

end

class Foo

  prepend Prepend

end

Foo.new.foo

```

第四，Refinement

```

module R

  refine String do

    def foo

      ...

    end

  end

end

"".foo # => error!

using R

"".foo

```

在提问环节中，很多与会者都对 mruby 和 Ruby 2.0 表示了浓厚的兴趣。Matz 表示 Ruby 的性能在不断改善之中，2.0 的性能有比较不错的改善，比如 64 位

系统里的浮点数性能得到了大大的改善 ,但是目前还不会考虑多线程方面的优化 ,mruby 会在不久的未来支持 Fiber ; mruby 能运行于各个平台之上 ,但它不会替代 cruby ,也不想代替 cruby , mruby 语法上兼容 1.9 ,但是由于一些原因 ,mruby 中还不能很好地使用 c extension 。移动领域是下一个重要的战场 ,Ruby 会在这个方面继续下功夫。

本次大会上 Matz 反复提到想让世界更美好 ,让 Ruby 更美好 ,Matz 希望 Ruby 2.0 能成为 Ruby 历史上最好的版本 ,也希望大家能够积极参与 ,共同让 Ruby 变得更美好。

原文链接 : <http://www.infoq.com/cn/news/2012/11/rubyconf-matz>

相关内容 :

- [Ruby 2.0 Preview 1 发布 , 正式版将于 2013 年 2 月发布](#)
- [Google Dart 精粹 : 应用构建 , 快照和隔离体](#)
- [Ruby on Rails : 3.2 RC1 发布 , 4.0 将会放弃 Ruby 1.8.7](#)
- [带有基于 Smalltalk 的 Ruby VM 的 NoSQL OODB : MagLev 1.0 发布了](#)
- [NodeJS 的异步编程风格](#)
- [ClojureScript 通过 Javascript 将 Clojure 引入到浏览器端](#)

特别专栏

SpringOne 大会采访：Mark Pollack 博士专访

作者 [丁雪丰](#)

SpringOne 大会今年首次落户中国，将会于 12 月 7-8 号在北京召开，大会结合了 Spring 和 Cloud Foundry 大热门技术，将会针对于 Spring 的最新发展、Spring 在移动、大数据领域的应用等做主题分享。InfoQ 就 Spring 以及 Spring Data、Spring Batch 等话题采访到了 Spring 社区专家 Mark Pollack 博士。

InfoQ：欢迎来到北京，我是 InfoQ 中文站的编辑丁雪丰，感谢您接受我们的采访。您能否先为我们的读者做个自我介绍？

“Mark：我就职于 SpringSource，它是 VMWare 的子公司，多年以来都在从事 Spring 开源项目相关的工作，早在成立公司支持它以前就是团队的成员了。我做过 JmsTemplate 相关的事情，早期还做过 Spring.NET 项目——这是.NET 版本的 Spring 框架。这些年数据访问技术方面发生了很多变化，大约三年前，我开始围绕 NoSQL、大数据这些方面为大家提供支持，还为关系型数据库添加了更多支持。

InfoQ 您刚提到了 NoSQL 技术，现在有很多数据存储，比如 RDBMS、NoSQL 和 NewSQL。在只有 RDBMS 时，Hibernate 可以应对大部分的问题。现在我们要面对 MongoDB、Redis 还有 Cassandra，Spring 如何让开发者的生活变得更轻松一点呢？

“Mark：我们的确为其中一些提供了支持，但不是全部 NoSQL 数据库都有支持，MongoDB 和 Redis 就有支持，Cassandra 目前还不在其列。Spring 从很多方面减轻了开发者的负担。首先，使用 Spring 容器来配置应用程序，通过一致的方式来使用多种技术，在 Spring 早期，你要把 MVC、模版引擎和 Hibernate 结合在一起，就得益于一致的使用方式；NoSQL 数据库和关系型数据库很类似，我们可以用一个通用的配置模型将所有技术整合起来，这只是纯粹的配置方面的。

其次 我们为很多 NoSQL 数据库 还有 Hadoop 提供了很多特定的辅助类，和 JdbcTemplate 很类似，我们提供了 MongoTemplate，能更方便地使

用 MongoDB，如果你只是使用标准 API 集合，它同样提供了高层 API，比如把 POJO 映射成文档数据模型，这是基本的值属性方面的。

第三，要创建一个针对 POJO 的数据访问类库，以 MongoTemplate 和 RedisTemplate 为例，能很方便地通过 MongoTempalte 来使用一些特性，比如 MapReduce、累加计数器，后者在 Redis 里也很常用，这都很容易。如果你还想映射 POJO 来做领域层，也就是 Repository 层，我们也提供了一些基础设施，帮你对各种 NoSQL 数据库进行 POJO 的增、删、改、查操作。

InfoQ：在您的演讲中，您提到了 Spring Data、Spring Integration，以及一些 Spring 的子项目，这些项目满足了开发者在大数据时代下的各种日常工作需要。您能否向国内的开发者介绍下 Spring Data 项目，虽然国内 Spring 的应用很广泛，但 Spring Data 还未广为人知。

“Mark：我很早就开始参与 Spring Data 项目了，我们的目标是提供熟悉的基于 Spring 的编程模型，涵盖大数据、NoSQL 和关系型数据库。如此一来，需要对 NoSQL 数据源做一个抽象，因此我们有两类支持，一个是一系列的模版类，让你更方便地与下层的 NoSQL 存储进行交互；另一个是 Repository 抽象，让你更方便地编写数据访问层，把 POJO 写到数据存储里。

另一大块内容是大数据 我们有 Spring Hadoop 项目 协助进行与 Hadoop 的交互，在 HDFS 里读写文件。Hadoop 中经常用到 Hadoop 任务，你有一系列顺序执行的任务，Spring Batch 能帮你组织这些任务。

InfoQ：这些项目在国外的使用情况是怎么样的，比如美国和英国，在那里大家都用这些 Spring 子项目么？

“Mark：我们一直在持续追踪 Maven 上的下载统计信息，这些项目的下载量都是呈指数级增长的。我最近没有关注，但我认为，Spring Integration 和 Spring Batch 的流行程度比较接近。Spring Data JPA 是最流行的项目，虽然 NoSQL 的讨论很热烈，但企业开发者还是在使用关系型数据库，所以对这个结果我们并未感到太吃惊。我们的 MongoDB 支持、Redis 支持最近也越来越流行了，还有 Neo4j，它们的成长曲线很漂亮，终会和 Spring Integration 和 Spring Batch 一样流行的。

InfoQ : 在你看来 , Java EE 6 和 Spring 框架是个什么关系 ? 有些人认为 Java EE 6 比 Spring 要好 , 你是怎么想的 ?

“ Mark : Java EE 6 就是一堆 JSR 的集合 , 它的版本基本就是标明用了哪个版本的 JSR 实现。 Spring 通常都会部署在 Java EE 容器里 , 提供一些容器里没有的功能 , 举例来说 , 可以让 Spring 在老的不支持 CDI 的容器里提供依赖注入 , 很多人都看到了其中的价值。 Spring 和 Java EE 6 有很多共同点 , 比如都实现了 Bean Validation 、依赖注入注解等等 , 的确有很多重复的地方。 Spring 最出名的是它的依赖注入 , CDI 是 Java EE 6 里的依赖注入规范。我认为依赖注入现在已经是个标配了 , 我想人们不会从一大堆特性里把它挑出来。 Spring 给你带来的是更多的灵活性 , 你可以在 Tomcat 3 里用依赖注入 , 然后升到 Tomcat 7 , 不会受限于 Java EE 的版本 , 不受限于 JSR 。

InfoQ : Java 7 已经被广泛使用了 , 明年 Java 8 很快就要来了 , Spring 是否已经为新的 JDK 做好准备了 ?

“ Mark : Java 8 很令人振奋 , 我们对一系列的特性都很感兴趣 , 比如闭包。我们的目标是成为第一个支持 Java 8 的现代 Java 框架 , 因此我们非常期待 Java 8 的到来。

InfoQ : 我们应该如何使用开源软件来构建基于 Hadoop 的大数据平台 , 让整个解决方案更加易于维护 ?

“ Mark : 在 Spring Hadoop 是个很好的项目 , 其中你能找到很多东西帮助创建 Hadoop 项目。如何用它来创建一个更可靠的解决方案 , 人们使用 Hadoop 的传统方式是用 MapReduce 任务来和 HDFS 交互 , 用 Pig 或者 Hive 这种构建于 MapReduce 核心之上的语言 , 所有这些操作基本都是通过命令行、 Shell 、命令行应用的方式 , 很多都是在 JavaScript 、 Perl 和 Ruby 脚本里 , 一个接一个的进行调用 , 这种方式很乏味 , 严重依赖于脚本 , 在用的时候要十分小心。

在 Spring Hadoop 里 , 我们有个使用 Hadoop 的编程模型 , 易于使用 , 和现有的配置 Spring 的方式保持一致 , 提供了很多辅助 API 和工具类 , 让你从脚本驱动的编写 Hadoop 应用的方式转变到一种使用 Java 类的方式 , 你有了一套早已习惯的结构 , 只需配置到 Hadoop 的连接 , 然后执行 Hadoop 任务。比如 , 你现在可以把那个任务注入到你的代码里并运行。这种方式比

意大利面条式的脚本更具可维护性。可以先从一个简单的任务和 Spring 应用上下文开始，以后要是有了很多步骤，比方说 20 个步骤，这时可以在 Spring Batch 任务里复用那些 Hadoop 任务的定义，也就是说 Spring Batch 里的任务就是 Hadoop 任务的引用。也许这么多“任务”有点混淆，但通过这种方式应用程序能更加可靠，可维护性更强。

InfoQ：非常感谢您接受我们的采访。

 Mark : 谢谢。

原文链接：

<http://www.infoq.com/cn/articles/springone-interview-mark-pollack>

特别专栏

采访与书评：Spring Integration in Action

作者 [Sriini Penchikala](#) 译者 [张卫滨](#)

Mark Fisher , Jonas Partner , Marius Bogoevici 以及 Iwein Fuld 合著的 [《Spring Integration in Action》](#) 一书介绍了 Spring Integration 框架 , 该框架基于 Spring 编程模型 提供了众所周知的 [企业级集成模式 \(Enterprise Integration Pattern , EIP \)](#) 的一种实现。在基于 Spring 的应用中它实现了轻量级的消息传递并支持使用声明式的适配器实现应用集成。

作者对 Spring Integration 框架的讨论是从异步消息架构的核心组件开始的 : Message、Channel 以及 Message Endpoint。正如他们在第 1 章中所言 , Spring Integration 就是 “企业级集成模式遇到了控制反转 (Inversion of Control) ” 。作者通过示例应用和代码样例详细讨论了企业级集成模式以及 Spring Integration 框架是怎样实现这些模式的。

本书还涵盖了 Spring Integration 提供的对邮件和文件适配器、Web 服务以及 使用 Spring Batch 框架对文件进行批处理的支持。为了使讨论更完整 , 作者还谈及了使用 JMX 实现对消息组件的监控和管理。最后 , 本书讨论了测试的话题 , 谈及了怎样通过模拟 (mock) 服务来实现对异步系统和消息组件的测试。

InfoQ 与 Mark Fisher 和 Iwein Fuld 讨论了这本书以及 Spring Integration 框架的优势和局限性。

InfoQ: 有这样一种趋势 , 就是将应用集成作为基于云的服务。你能介绍一下这种 “集成即服务 (Integration as a Service) ” 的新理念吗 ? Spring Integration 框架能够怎样提供帮助 ?

 **Iwein:** Spring Integration 本身并不是什么 “服务 (as a Service) ” 。这是一件好事儿。作为开发人员 , 你感兴趣的是与其他*aas API 集成 , 而使用 ESB 作为混合服务通常并不会简化什么事情。在服务架构中 , 使用 Spring Integration 是很容易的 , 在云中何尝不是如此呢。 SI 的优点在于集成而不是服务。就是说 , Cloud Foundry 对 Spring 应用提供了原生的支持因此也就支持了 Spring Integration , 它还提供了 Spring Integration 支持的服务

——如 RabbitMQ、Redis、MongoDB 以及关系型数据库——所以对你来说所有的可能性和例子都是信手拈来的。

InfoQ: 在消息架构中 使用 NoSQL 数据存储是不是通常比关系型数据库更好，比如在 Queue Channel 和 Messages Store 中存储消息？在消息应用中，使用 NoSQL 数据库进行持久化的优缺点是什么？

“ Iwein: 如果你使用 NoSQL 存储的话，在写以及简单的“按键值”查询的时候通常会有点优化。NoSQL 存储与消息模型的配合更为自然。也就是说，对于关系型数据库来说针对这种使用方式进行优化也不是很难的事情，所有说选择 Spring Integration 作为消息框架并不是选择 NoSQL 存储的决定性因素。

Mark: 另外，NoSQL 存储并不是放之四海皆准的，因此更有意义的讨论是通过为不同的 NoSQL 存储提供适配器，让 Spring Integration 支持更为广泛的应用。例如，使用我们在 2.2 版本中添加的 Channel Adapter，你可以很容易地以消息驱动方式更新 Redis ZSET 中的 score。

InfoQ: 在有效且高效地处理和分析不同类型的业务数据方面，大数据、数据挖掘以及数据处理最近得到了很多关注。Spring Integration 和 Spring Batch 框架是如何适应大数据领域的呢？

“ Iwein: 如果你想基于用后就扔掉的代码快速得到结果的话，那 Spring Integration 和 Spring Batch 可能并不适合你。但是，如果你想要的是一个可维护的解决方案，那它们就适合了。长期存在的应用程序通常更看重可维护性而不是快速推向市场，Spring 组合产品正是能够在这类应用中发挥其价值。在大数据领域，很多事情都是容易发生变化的。通常的解决方案会快速完成、收回投资然后丢弃掉。如果你们的应用是这样的话，那尽可以忽略 Spring Integration。

Mark: 但是，我们开始设计的 3.0 版本的一个主题就是对大数据的支持。它不仅会包含 HDFS 适配器、为 Apache Hadoop 项目无缝集成 Spring，还可能会包含将 disruptor model 应用于 socket 层次的连接工厂这样的特性。结合 Spring Integration 的低开销特性，应该能够满足很多大数据解决方案的吞吐量需求。另外，在探索“现实世界”大数据场景的时候，我们认识到，最终需要解决的通常是集成问题：结合来自多个数据源的数据、定义跨不同

系统的数据流动、创建多个处理步骤的流水线等等。很明显，在处理“传统”企业级集成时所用的工具和框架是可以用在这里的。

InfoQ: 安全是企业级应用开发中很重要的一个方面，在集成用例中更是如此。您能介绍一下 Spring Integration 为保护各种消息组件所提供的安全支持吗？在这个领域的最佳实践是什么？

Iwein: 在消息系统的安全要么在传输层做，要么在消息层做，要么在两者上都做。很酷的一点是 Spring Integration 在这两方面都涉及到了。鉴于 SSL HTTPS 之下就是 HTTP 了，所以 SI 本身不需要知道这些。在消息层，Spring Integration 将安全问题委托给 Spring WS 的安全组件。这样很好，我们本身躲过了这个问题。在我看来，如果关心安全的话，你要保护好消息。你只需要保证最终的接受者和最终的发送者是可信的，而不需要为整个网络拓扑负责。很多组织走的是另外一条路，他们完全忽略了消息层的安全，因为他们认为“管好传输层就好了”。这是很愚蠢的做法，就像安装操作系统的人没有给你安装键盘记录器那样，但现实中这种做法却广泛存在。安全领域的最佳实践就是时刻保持警惕，别相信什么最佳实践。

Mark: Spring Integration 还提供了 Channel Interceptor，它只是简单地委托给了一个 Spring Security 授权管理器，但是如果 SecurityContext 上已经填充了来自传输层的可用信息的话，通常还会加以检查。

InfoQ: 在书中你们也讨论了如何测试消息应用中的不同组件。能介绍一些这方面的技术吗？Spring 和 Spring Integration 框架怎样简化对消息组件的测试的？

Iwein: 相当清楚，第 18 章就是关于这一点的。在我进行 SI 测试并编写这一章时，最让我大开眼界的事情就是若想了解多线程应用程序是如何工作的，使用调试器并不是什么好主意。意识到这一点后，我不再使用调试器，转而按照自己的思路来修复测试和日志。在学习调试上浪费了太多时间，而且在意识到问题之后相关知识就被直接丢到九霄云外了，这是让人震惊的。

另一件值得一提的事情是，大多数的并发问题都能通过精心设计的测试用例触发。我和 Mark 经常因为无法重现彼此的问题而工作到深夜，但是只要我们愿意付诸行动，总会有办法确定测试用例。相对于一般的功能测试，这会花费更多的时间，你要有所准备，因为在紧要关头这可是会救你一命的。

InfoQ: EIP (企业级集成模式) 的局限性是什么，换句话说集成架构师或开发人员在他们的应用程序中使用这些模式时要注意什么？

“**Iwein:** 写在书中的事情有一个问题就是，你不必保证它们解析和编译成实际代码时是否严谨。在写这本书中，我们纠结过很多次，但是在实现 SI 的时候，我们也纠结于 EIP 这本书。当设计架构时，架构师需要记住的是将来的代码会看起来有所不同，而开发人员应当记得新兴的框架要像低层次的代码那样干净。EIP 更大意义上是开始时的指南，实际工作的代码要胜于它，即使代码中的模式与书中的模式有所不同。

Mark: 另外，当涉及到设计模式时，通常语言是最重要的。语言使得开发人员能够将其所学转化为给定 API。因此，当要确定某个属性的名字或配置特性的时候，我们会查看 EIP 这本书并希望从那里使用的单词上获取灵感。一致性也是很重要的，所以每当要将特定的模式映射到我们的 API 并遇到困难时，我们会尝试使用类似的技术和术语。

InfoQ: 你们能否介绍一下企业级应用集成领域的发展趋势吗？

“**Iwein:** 主要的趋势（依然）是 REST。因为 JavaScript 框架和浏览器已经规范化了，所以 HTTP 才刚刚开始发挥其全部潜力。另一个有意思的趋势就是通过远离既有的做事方式来优化效率，这个趋势我见过好几次了。这可能对移动设备、片状网络（flaky network）有所助益，但是对于那些看似不太可能的设备（如家庭能源测量装置）也是有所裨益的。Spring Integration 的 TCP/UDP 适配器可能在这样的情况下派上用场。

Mark: 除了 REST API，基于 Web 并使用 Web Socket 的消息系统以及诸如 SockJS 的高层次抽象也是越来越流行了。这种技术为应用集成打开了一个全新的世界，在这里客户端/服务器的关系戏剧性地发生了变化。在 Spring Integration 3.0 的路线图中这也是我们关注的一个焦点。

作者们还谈论了应用集成的通用知识和最佳实践。

“**Iwein:** 我不赞成复杂的应用，尽管它可能是通过复用我的代码实现的。很多开发人员认为他们可能需要异步处理，所以他们使用了 SI、Camel 或 Akka。最后，往往没有严格的评估来确认单线程的解决方案是否满足需求。不要因为你能使用一项很酷的技术就干这种傻事。不要误会我的意思，SI 是很不错，但是如果你能远离并发的话，那你最好还是远离它。

Mark: 也就是说，当消息驱动的模式确实能够满足你需求的话，Spring Integration 为你隐藏了大多数的复杂性。为了优化性能，你还是需要理解如何配置线程池和触发器，但是你不需要编写处理任务调度和异步调用的代码。而是像其他的 Spring 应用那样，你只需要关注业务域的 POJO。这也意味着你的代码能够与基础设施解耦合进而便于测试，这也是我们可能都认可的“最佳实践”。

关于作者：



Mark Fisher 是 Spring Integration 项目的创建者。目前他在 VMware 仍然领导着 Spring Integration 的开发，同时还探索大数据和消息系统的结合。他是许多 Spring 项目的提交者，包括 Spring 框架本身和他参与创建的 Spring AMQP。Mark 经常在一些会议和关于消息系统、数据、集成以及云计算的用户组发表演讲。



Iwein Fuld 是一个独立顾问，关注高质量开发以及团队培训。他是一个多面手，但是总能保持对服务端集成问题和算法的关注。Iwein 从 2008 年初就是 Spring Integration 项目的提交者。除了是 TDD 和并发方面的专家，Iwein 尤其喜欢组建敏捷团队和精益创业。

特别专题

从关系数据库向 NoSQL 迁移：采访 Couchbase 的产品管理主管 Dipti Borkar

作者 [Abel Avram](#) 译者 [臧秀涛](#)

尽管关系数据库用于存储数据已经有几十年的历史，而且对很多用例而言，这仍然代表着一类可行的方案，但 NoSQL 正在成为人们当前的选择，尤其是考虑可伸缩性和性能的时候。本文是对 Couchbase 的产品经理主管 Dipti Borkar 的采访，主要谈到了从关系数据库向 NoSQL 迁移的挑战、收益和过程。

InfoQ：何时才是放弃 SQL 转而寻求 NoSQL 解决方案的时机呢？

“**Dipti Borkar**：这个问题可能有些尖锐——事实上，大多数情况下并不是放弃 SQL 转而寻求 NoSQL 解决方案，而是为了让应用和用例满足需求的变化，从一种方案转向另一种方案。一般而言，在构建现代 Web 和移动应用时，不管是伸缩模型还是数据模型，对灵活性都有特定的需求，而这种需求正是从 SQL 向 NoSQL 迁移的推动因素。

典型的 Web 应用程序是采用三层架构构建的。应用程序要向外扩展时，一般是简单地在负载均衡器之后添加更多的商品化 Web 服务器来支持更多用户。而对于越来越重要的云计算模型而言，向外扩展能力是其核心原则。在云计算模型中，虚拟机实例很容易根据需求进行相应地添加或删除。

然而，当涉及到数据层时，关系数据库（RDBMS）不但无法向外扩展，也没有提供灵活的数据模型，这方面有很多挑战。要处理更多的用户，这意味着要加入一台更为大型的服务器，而大型的服务器复杂度很高，一般是专有的而且非常昂贵，不像基于 Web 或云的架构中所使用的商品化硬件那样廉价。因此，当公司开始发现现有应用或新应用所用的关系数据库存在性能问题时，特别是这一切与用户数目的增长有关时，他们意识到需要一个更快的、有弹性的数据库层。现在是时候评估 NoSQL 技术并将其作为交互式 Web 应用的数据库层了。

InfoQ：在从 SQL 向 NoSQL 迁移时，需要哪些主要步骤？

“**Dipti Borkar**：在使用 NoSQL 数据库时，不同的组织或项目追求的目标是五花八门的。所以很多迁移还是取决于具体的使用情况。下面是迁移时的一些通用指导原则：

#1 理解应用的关键需求：

某些与 NoSQL 匹配的需求如下：

- 快速应用开发
 - 变化的市场需求
 - 变化的数据需求
- 可伸缩性
 - 未知的用户需求
 - 访问、添加和更新数据使吞吐量持续增长而带来的需求
- 一致的性能
 - 低响应时间，以便支持更好地用户体验
 - 高吞吐量，以便处理快速地增长
- 运行可靠性
 - 高可用性，能够优雅地处理失效并尽量减小对应用的影响
 - 内置监控 API，便于运行时维护

#2 理解 NoSQL 产品的不同类型：

有一个常见的误区，就是认为所有的 NoSQL 数据库都是等同的。比如，Cassandra 的列存储数据模型可能适用于分析类应用，而图形数据库 Neo4j 则适用于需要访问实体间关系的应用。

我们会特别关注分布式的、面向文档的 NoSQL 技术，Couchbase 和 MongoDB 就是两个最常见的、应用最广泛的例子。

#3 证明理念的可行性

一旦将潜在的选择缩小到数据库层，在集成应用程序的关键特性时，要计划好如何证明相关理念的可行性。可以看一下响应时间、吞吐性能和易扩展能力。

#4 文档建模与开发

对于文档数据库，数据模型会从固定的表格模式转为灵活的文档对象，因此需要在数据建模上花些时间。

#5 分阶段向产品部署

对交互式 Web 应用程序而言，运行的稳定性是非常重要的。当推出 Web 应用程序时，应该像对待采用传统关系数据库系统的应用程序一样进行测试和阶段部署。请确保所选的数据库支持跨集群监控，并且能够方便地在线伸缩以支持按需扩容，还需要支持其他数据库管理工具。

#6 跟上最新的趋势

在美国有很多高质量、免费的实践式 NoSQL 培训课程。要确保成功实现 NoSQL 方案，最好是有一支受过培训的开发团队，而且该团队应该了解最新的服务器发型版本和供应商的产品。

下面是几个最大的培训机构的链接：

[-CouchConf](#)

[-NoSQL Now](#)

InfoQ：从 SQL 向 NoSQL 迁移时，有哪些主要的困难？

“Dipti Borkar：主要困难基本上可以归结为理解传统的关系数据库系统和文档数据库的差异。最重要的区别是数据模型：



Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.



Document data model

Collection of complex documents with arbitrary, nested data formats and varying “record” format.

如上图所示，关系数据库中的每条记录都要符合某一模式——字段（列）数是固定的，而且每个字段都应该有一个明确的目的和数据类型。每条记录都有一样的模式。多个表之间的数据是规范化的。优点是数据库中的数据很少出现冗余。缺点是模式的改变需要执行一些代价很高的“alter table”语句，因为要避免数据库出现不一致状态，这种操作需要同时锁住很多表。

而使用文档数据库时，每个文档的结构可以与其他文档完全不同。数据库不需要额外管理文档模式的改变。

InfoQ：NoSQL 文档数据库有什么优点？

“ Dipti Borkar：文档数据库的主要优点包括以下几个方面：

- 灵活的数据模型

数据不需要明确的模式就能插入，而且插入数据的格式可以随时变化——这为应用带来了极大的灵活性，最终会带来实际的业务敏捷性。

- 容易扩展

有些 NoSQL 数据库不需要应用参与就能自动跨服务器传播数据。通过数据与 I/O 的跨服务器传播，可以在不停掉应用的情况下添加和删除服务器。

- 一致性和高性能

先进的 NoSQL 数据库技术能够在系统内存中缓存数据，这对开发者和运维团队是完全透明的。

InfoQ：如果告诉开发者采用了 NoSQL，他们会作何反应？

“ Dipti Borkar：对于 NoSQL 技术，开发者会感到非常激动，尤其是因为某些数据库所带来的开发的简洁性。文档数据库有极为灵活的模式，而且易于使用。

因为不需要修改底层数据库的模式，对于应用的变更，开发者可以进行更为快速地迭代。如果开发者构建应用程序时所用的数据为稀疏数据，或者是不断变化的数据，或者是他们无法掌控的来自第三方的数据，文档数据库尤为有用。

InfoQ：与现有的开发人员一起工作并让他们学习新技术，这是否可行？还是需要寻找新的掌握 NoSQL 技术的开发人员？

Dipti Borkar : 应用开发者会发现某些 NoSQL 技术是非常容易使用的，特别是那些支持 JSON 文档格式的技术。越来越多的开发者在应用中使用 JSON 为对象建模。因此，将数据直接以 JSON 格式保存在数据库中能够减少跨栈的阻抗失配。

严重依赖 SQL 的开发者可能需要适应并学习文档建模方法。重点是，他们需要反思如何使用文档来对数据进行逻辑组织，而不是将数据规范化为固定的数据模型。

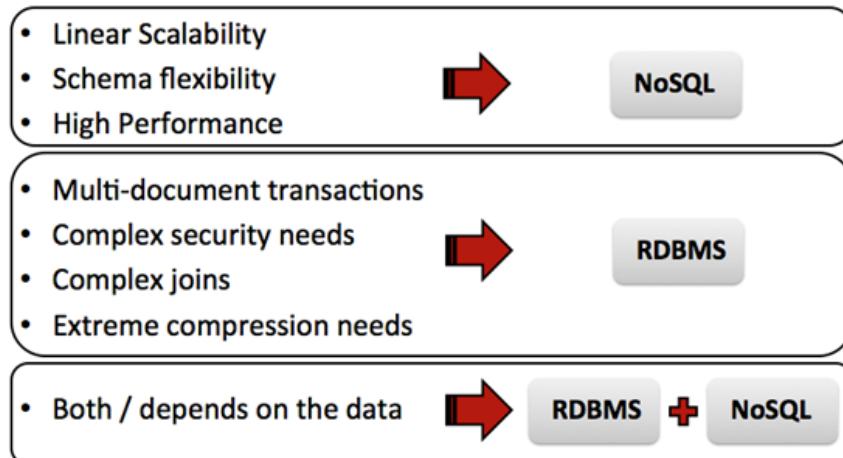
InfoQ : 你是否有过或者听过向 NoSQL 迁移时的失败案例？如果有的话，错在什么地方？

Dipti Borkar : 架构师和开发者应该确保所选的方案或数据库能够满足他们的关键需求。例如，如果所选的数据库更适合于分析类应用，那么这种数据库可能无法满足交互式应用的延迟和吞吐量需求。如果没有研究所有需求就匆忙地做出选择，这样的项目可能会因数据访问的响应时间过长而导致用户体验很差。对于可伸缩性，用户提前应该有所计划。还有一个问题更为严重的例子，在某些情况下，应用规模已经疯长了，但所选的数据库却跟不上这种规模，无法向外扩展。

同时，对于更适合于 OLTP 风格应用的数据库，如果将其应用于高级数据分析或复杂处理，效果可能也不好。大数据方案估计是更好的选择。

InfoQ : 向 NoSQL 迁移时，有哪些主要的教训？

Dipti Borkar : 向 NoSQL 迁移时，开发者会受益良多。数据模型更为灵活并且不需要僵硬的模式，这就是一个很大的优点。你可能也会看到性能的明显提升以及数据层水平向外扩展的能力。但是大多数 NoSQL 产品仍处于产品周期的前期阶段。虽然像复杂连接或多文档事物等功能可以在应用中模拟，但这时开发者使用传统的关系数据库可能更舒服。对某些项目而言，混合方法或许是最好的选择。



关于被采访人



Dipti Borkar 是 Couchbase 的产品管理主管，她负责 Couchbase 服务器（一种 NoSQL 数据库）的产品路线图，并与客户和用户一起工作来理解对低延迟、可伸缩数据存储方案的新兴需求。在数据库方面，Dipti 拥有深厚的技术经验，她曾经在 IBM 担任过软件工程师，还曾是 DB2 服务器团队的项目经理，后来在 MarkLogic 担任过高级产品经理。Dipti 从加州大学圣地亚哥分校获得了计算机科学硕士学位，她的主攻方向就是数据库。她还获得了加州大学伯克利分校哈斯商学院的 MBA 学位。

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

相关内容：

- [采访与书评：NoSQL Distilled](#)
- [CouchDB 与 Couchbase：区别何在，Membase 又将如何？](#)
- [新调查结果表明：NoSQL 采用呈上升趋势](#)
- [James Phillips 谈从关系型数据库转到 NoSQL](#)
- [分布式缓存能否作为 NoSQL 数据库？](#)

CouchDB 是什么？为什么我们要关注它？

作者 Warner Onstine and Leo Pryzbylski 译者 孙镜涛

CouchDB 是众多称作 NoSQL 解决方案中的一员。与众不同的是，CouchDB 是一个面向文档的数据库，在它里面所有文档域（Field）都是以键值对的形式存储的。域（Field）可以是一个简单的键值对、列表或者是 map。

CouchDB 会为存储到数据库中的每一个文档分配一个文档级别的唯一标识符（*id*），同时每次将变动保存到数据库中时还会分配一个修订号（*rev*）。

NoSQL 数据库的出现代表着传统的关系型数据库的转变，它能够提供很多好处，当然其自身也面临着挑战。CouchDB 为我们提供了下面的特性：

- 容易地在多个服务器实例之间进行数据库复制
- 快速地索引和检索
- REST 风格的文档插入、更新、检索和删除的接口
- 基于 JSON 的文档格式（更容易地在不同语言之间转换）
- 为用户选择的语言提供多个库（指一些流行的语言）
- 通过`_changes` 订阅数据更新

从 [NoSQL 系统可视化向导](#) 中可以找到一个非常出色的工具，它能帮你决定哪一个数据存储适合你。该指南描述了选择数据库系统时应该关注的三个方面（NoSQL 和关系型数据库都是如此）。在我们的项目中使用该指南筛选数据库时会关注下面的特性：

- 可用性
- 一致性
- 分区容忍度

CouchDB 侧重于 AP（可用性和分区容忍度），这正是满足我们的数据关注点所要寻找的数据库（更不用说在连续的或者点对点的设备间进行数据复制的能力）。相比之下，MongoDB 侧重于 CP（一致性和分区容忍度），像 Neo4J 这样的数据库则提供了特有的面向图形的结构。

另一个出色的工具是这篇[博客文章](#)，它对 Cassandra、MongoDB、CouchDB、Redis、Riak、Hbase 和 Membase 做了比较。

当然，对于一个给定的项目你很可能有多个工具，换言之，这就需要明确需求并找到合适的工具以满足这些需求。

我们将如何使用 CouchDB ?

我们将要构建一个简单的本地事件数据库，用于存储一些事件以及这些事件发生的位置。我们将会把它分为两个文档，通过它们的文档 id 将两者关联起来。这两个文档是：

- 事件
- 位置

(本文稍后会为这两个文档创建 Java 类)

Jcouchdb

我们将使用 [jcouchdb](#) 与 CouchDB 数据库交互。这是一个经过良好测试并且易于使用的 Java 库，它会自动地将 Java 对象序列化、反序列化进 CouchDB 数据库。选择 jcouchdb 的另一个原因是它和 CouchDB 自身的 API 非常相似。

Jcouchdb 的替代方案有那些？

如果你不喜欢 jcouchdb，或者想要尝试其他的库，那么可选项也有很多，如：

- [Ektorp](#)
- [JRelax](#)
- [CouchDB4J](#)
- [DroidCouch](#)

其中有少数已经很久没有更新了，所以，如果你要做一些测试，请确保预留一些时间解决程序的问题。

开始

从哪儿开始呢？我们将会使用 Maven3 构建这个示例项目。哪怕不知道 Maven 也能理解代码，但是为了构建并运行示例项目你需要安装它。可以从 Maven [站](#)找到 Maven3。

指南的这个部分假定你具有一定的 Maven3 知识，但是如果你不了解 Maven，你可以直接使用从我们的库中下载的 pom.xml 文件并直接使用它。

我们将会跳过 POM 创建的初始部分，但是如果需要创建 POM 文件的细节或者仅仅想要开始编码可以从我们的 [github 库](#)中下载它。首先要做的就是指定需要的 jcouchdb 和 Spring 组件。

```
<properties>
    <spring.framework.version>3.1.0.RELEASE</spring.framework.version>
    <spring-xml.version>2.0.0.RELEASE</spring-xml.version>
    <jcouchdb.version>0.11.0-1</jcouchdb.version>
    ...
</properties>
```

在文件顶部指定版本信息的一个原因是，这样可以很容易地一次性将一个库（或者一组库，如 Spring）快速地更新到新版本。

```
<dependencies>
    <dependency>
        <groupId>com.google.code.jcouchdb</groupId>
        <artifactId>jcouchdb</artifactId>
        <version>${jcouchdb.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.framework.version}</version>
    </dependency>
```

```
</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-aop</artifactId>

    <version>${spring.framework.version}</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-test</artifactId>

    <version>${spring.framework.version}</version>

</dependency>

<dependency>

    <groupId>org.springframework.ws</groupId>

    <artifactId>spring-xml</artifactId>

    <version>${spring-xml.version}</version>

</dependency>

...

</dependencies>
```

在初始化依赖设置完成之后，我们需要为项目设置剩下的目录结构。我们将遵循标准的 Maven 设置：

```
-src
  -main
    -java
    -resources
  -webapp
```

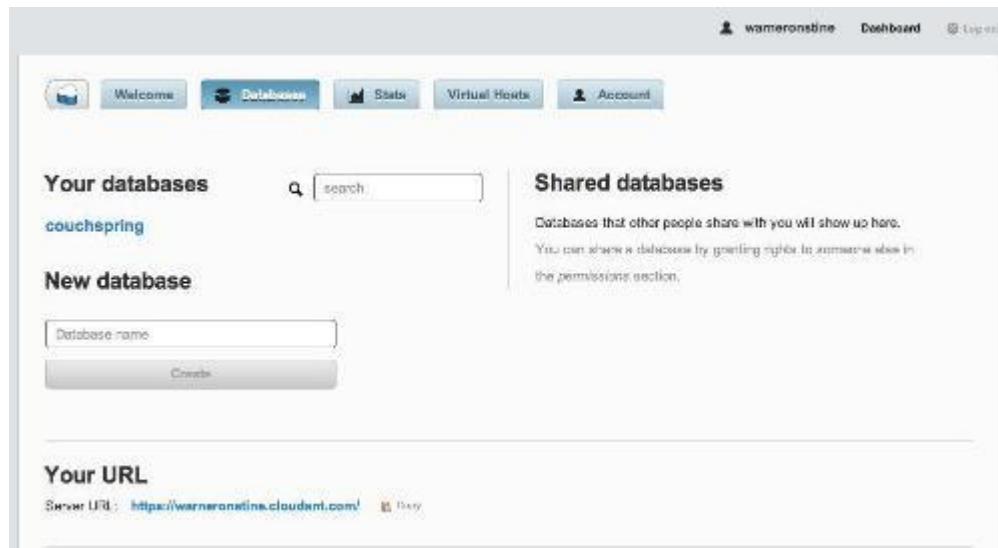
```
-test
  -java
  -resources
```

设置 CouchDB

完成了初始化设置之后，接下来就需要设置 CouchDB 数据库了。幸运的是，有一些非常好的解决方案可以帮助我们快速地启动并运行 CouchDB。

- [Cloudant](#)
- [Iris Couch](#)

它们都提供了免费的账号，能够完美的设置好数据库，以便我们开始开发工作。



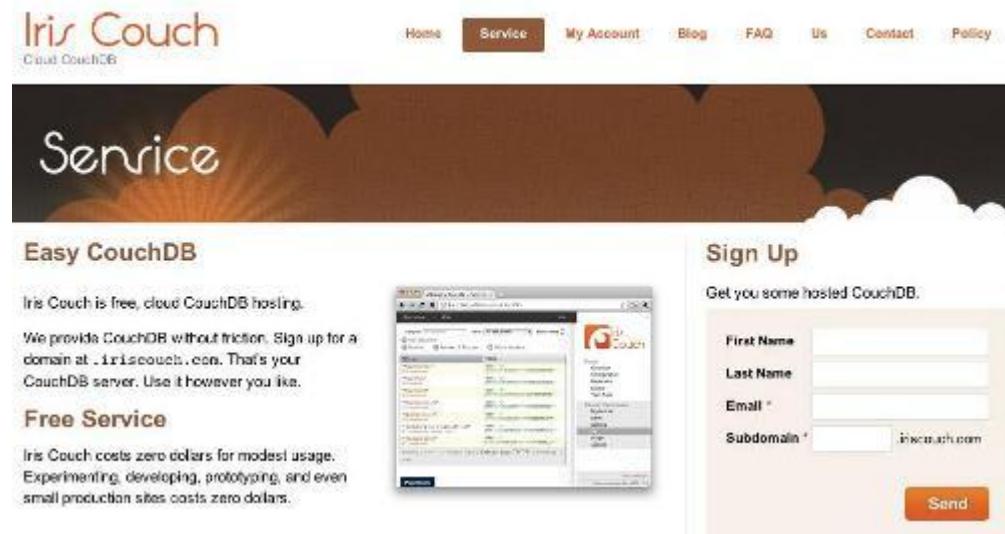
The screenshot shows the CouchAnt web interface. At the top, there's a navigation bar with links for 'Welcome', 'Databases' (which is currently selected), 'Stats', 'Virtual Hosts', and 'Account'. Below the navigation, there are two main sections: 'Your databases' and 'Shared databases'. In 'Your databases', there's a search bar and a list containing 'couchspring'. Below this is a 'New database' section with a 'Database name:' input field and a 'Create' button. In 'Shared databases', there's a note about shared databases and a link to the permissions section. At the bottom, there's a 'Your URL' section showing the server URL as <https://werneronline.cloudant.com/>.

图 1.CouchAnt 首页



The screenshot shows the CouchDB management interface. On the left, there's a sidebar with 'Overview' and 'werneronline:couchspring' selected. It includes links for 'New Document', 'Delete Database', 'Edit Document', 'View 24 documents', and a search bar. The main area displays a table with columns for 'Key', 'Value', and 'Status'. A red 'Delete' button is visible at the bottom of each row. On the right, there's a 'CouchDB relax' logo and a 'Tools' menu with options like 'Overview', 'Replicator', and 'Status'. Below that is a 'Recent Databases' section with a link to 'werneronline:couchspring'.

图 2.CouchAnt Futon 页面



The screenshot shows the CouchAnt Futon interface. At the top, there's a navigation bar with links: Home, Service (which is highlighted in red), My Account, Blog, FAQ, Us, Contact, and Policy. Below the navigation is a large banner with the word "Service" in white. To the left, under "Easy CouchDB", it says "Iris Couch is free, cloud CouchDB hosting. We provide CouchDB without friction. Sign up for a domain at .iris couch.com. That's your CouchDB server. Use it however you like. Free Service". It also mentions that Iris Couch costs zero dollars for modest usage. In the center, there's a screenshot of the Futon interface showing a list of databases. To the right, there's a "Sign Up" form with fields for First Name, Last Name, Email, and Subdomain, followed by a "Send" button.

图 3。Iris Couch 注册页面



The screenshot shows the Iris CouchFuton interface. On the left, there's a sidebar with a "Tools" section containing "Overview", "Configuration", "Replicator", "Status", and "Verify Installation". Below that is a "Recent Databases" list. The main area has a title "Overview" and a table titled "Cloud Databases". The table has columns: Name, Size, Number of Documents, and Update Seq. It lists three databases: "_replicator" (4.1 KB, 1 doc, 1 seq), "_users" (8.1 KB, 2 docs, 2 seq), and "couchspring" (79 bytes, 3 docs, 0 seq). At the bottom of the table, there are links for "Previous Page", "Next page", and "View Page".

图 4. Iris CouchFuton 页面

另一个选择是在本地机器（或主机）上安装 CouchDB。我们并不会带你在你的操作系统上安装它，但是在 [CouchDB 的 wiki](#) 上有一些非常好的说明。

在账号创建完成之后（或者在设置并启动 CouchDB 之后），将需要创建一个的数据库。在我们的应用程序中选择了 couchspring 作为数据库名。你可以随意取名，但是当我们开始配置设置时需要将其修改为对应的名字。

在 CloudAnt 中，可以在数据库截图（图 1）中创建数据库，而对于 Iris Couch 来说可以直接在 Futon 页面（管理 CouchDB 实例的用户界面）中创建。关于管理页面的更多信息可以在 [CouchDB wiki](#) 中找到。本文并不会过多的使用管理页面，但是这是一个非常好的操作视图的工具。



图 5.在管理页面中创建数据库的步骤 1

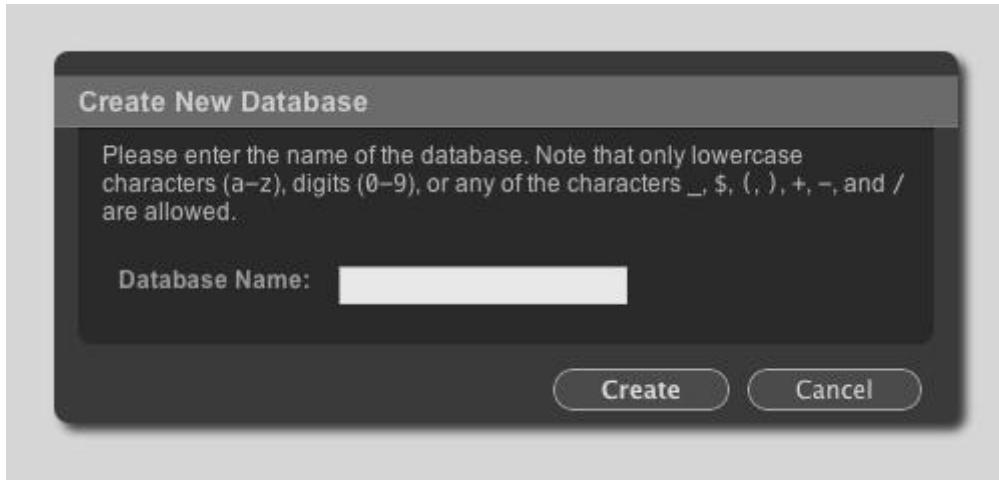


图 6.在 Futaon 页面中创建数据库的步骤 2

配置 jcouchdb、Spring 和 POJOS

在新数据库设置完成之后，我们需要：

- 创建基础 POJO 对象
- 提供一个 json 配置映射 ,它能够将 CouchDB 使用的 Java 对象和 JSON 对象自动地进行转换
- Spring 配置

首先，让我们创建一些对象！

带有自定义注解的 POJOs

我们将要为事件系统创建的基础对象是哪些？

- Event——存储来自于外部源（如 [Eventful.com](#)）或 Web 界面的事件
- Place——存储事件的发生位置。

同时，还会结合使用一些其他的对象（从外部源中抽取数据时做一些额外的数据处理）：

- AppDocument ——由 json 映射实用程序所使用的用来定义文档类型识别域的基础对象
- Description——用于格式化并过滤事件的描述
- Location——用于记录给定位置/地点的经纬度

首先，需要创建基础类 AppDocument

AppDocument.java

```
package com.clearboxmedia.couchspring.domain;

import org.jcouchdb.document.BaseDocument;
import org.svenson.JSONProperty;

public class AppDocument extends BaseDocument {

    /**
     * Returns the simple name of the class as doc type.
     *
     * The annotation makes it a read-only property and also
     * shortens the JSON name a little.
     *
     * @return document type name
     */
    @JSONProperty(value = "docType", readOnly = true)
    public String getDocumentType()
    {
        return this.getClass().getSimpleName();
    }
}
```

```
}
```

该对象继承了 jcouchdb 自身的 BaseDocument 对象，同时它还提供了一种区分不同的文档类型的方法。CouchDB 并没有提供处理这些内容的默认方式，而是将其留给了开发者，让他们去实现各自的处理方式。我们选择使用类名作为识别符，例如 Event 对象的 docType 会输出 Event，而 Place 对象会输出 Place。

接下来需要创建 Event 类。

Event.java (为简单起见，我们省略了一些域和方法)

```
package com.clearboxmedia.couchspring.domain;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlElement;

import org.svenson.JSONProperty;

public class Event extends AppDocument {
    private String id;
    private String title;
    private String description;
    private String startTime;
```

```
private String stopTime;  
  
private String venueId;  
  
private Map
```

这里有几件有趣的事。首先就是我们将会在对象中存储的是 venueId 而非 venue ,为什么要这样做呢 ?

因为 CouchDB 并不是关系型数据库 ,它没有一种直接的方式去定义两个不同文档之间的关系 ,因此我们在 Event 对象中存储 venue 的 id。我们可以在 event 对象中存储 venue 对象 ,但是将这些对象分开存储更清晰 ,尤其对于一个给定的地点可能会有多个事件。因此 ,我们将会提供一个动态的获取器 ,仅在我们需要的时候才会检索 venue 对象 ,而不是存储关系。我们将会在查询文档部分介绍这如何实现。 [todo: 动态查询]

现在 ,我们来定义 Place 类。

Place.java

```
package com.clearboxmedia.couchspring.domain;  
  
import java.util.LinkedHashMap;  
  
import java.util.List;  
  
  
public class Place extends AppDocument {  
  
    private String id;  
  
    private String name;  
  
    private String address1;  
  
    private String address2;  
  
    private String city;
```

```
private String state;  
  
private String postalCode;  
  
private String lastUpdated;  
  
private Boolean active;  
  
private Location location;  
  
private String venueType;  
  
private List<String> tags;  
  
  
public Place() {  
}  
  
  
public String getId() {  
    return this.id;  
}  
  
  
public void setId(final String id) {  
    this.id = id;  
}  
  
  
public String getName() {  
    return this.name;  
}  
  
  
public void setName(final String name) {  
    this.name = name;  
}
```

```
public String getAddress1() {
    return this.address1;
}

public void setAddress1(final String address1) {
    this.address1 = address1;
}

public String getAddress2() {
    return this.address2;
}

public void setAddress2(final String address2) {
    this.address2 = address2;
}

public String getCity() {
    return this.city;
}

public void setCity(final String city) {
    this.city = city;
}

public String getState() {
```

```
        return this.state;  
  
    }  
  
    public void setState(final String state) {  
        this.state = state;  
    }  
  
    public Location getLocation() {  
        return this.location;  
    }  
  
    public void setLocation(final Location location) {  
        this.location = location;  
    }  
  
    public String getVenueType() {  
        return this.venueType;  
    }  
  
    public void setVenueType(final String venueType) {  
        this.venueType = venueType;  
    }  
  
    public String getPostalCode() {  
        return this.postalCode;  
    }
```

```
public void setPostalCode(final String postalCode) {  
    this.postalCode = postalCode;  
}  
  
public String getLastUpdated() {  
    return this.lastUpdated;  
}  
  
public void setLastUpdated(final String lastUpdated) {  
    this.lastUpdated = lastUpdated;  
}  
  
public Boolean getActive() {  
    return this.active;  
}  
  
public void setActive(final Boolean active) {  
    this.active = active;  
}  
  
public List<String> getTags() {  
    return this.tags;  
}  
  
public void setTags(final List<String> tags) {
```

```
    this.tags = tags;  
  
}  
  
}
```

我们不再详细介绍其他的辅助对象 Description 或者 Location , 因为它们实在是太简单了。如果你对它们感兴趣 , 可以从 [GitHub 仓库](#) 中检出它们。

配置 jcouchdb 和 JsonConfigFactory

在配置之前 , 需要创建一些即将使用的类。JsonConfigFactory 用于 json 数据 (CouchDB) 和 Java 类之间的映射 , CouchDbServerFactory 为将要连接的服务器创建新的实例。

```
JsonConfigFactory.java public class JsonConfigFactory {  
  
    /**  
     * Factory method for creating a {@link JSONConfig}  
     *  
     * @return {@link JSONConfig} to create  
     */  
  
    JSONConfig createJsonConfig() {  
  
        final DateConverter dateConverter = new DateConverter();  
  
        final DefaultTypeConverterRepository typeConverterRepository =  
            new  
DefaultTypeConverterRepository();  
  
        typeConverterRepository.addTypeConverter(dateConverter);  
  
        // typeConverterRepository.addTypeConverter(new LatLongConverter());  
  
        // we use the new sub type matcher  
  
        final ClassNameBasedTypeMapper typeMapper = new
```

```
ClassNameBasedTypeMapper();

typeMapper.setBasePackage(AppDocument.class.getPackage().getName());

    // we only want to have AppDocument instances

    typeMapper.setEnforcedBaseType(AppDocument.class);

    // we use the docType property of the AppDocument

    typeMapper.setDiscriminatorField("docType");

    // we only want to do the expensive look ahead if we're being told to

    // deliver AppDocument instances.

    typeMapper.setPathMatcher(new SubtypeMatcher(AppDocument.class));

final JSON generator = new JSON();

generator.setIgnoredProperties(Arrays.asList("metaClass"));

generator.setTypeConverterRepository(typeConverterRepository);

generator.registerTypeConversion(java.util.Date.class, dateConverter);

generator.registerTypeConversion(java.sql.Date.class, dateConverter);

generator.registerTypeConversion(java.sql.Timestamp.class,
dateConverter);

final JSONParser parser = new JSONParser();

parser.setTypeMapper(typeMapper);

parser.setTypeConverterRepository(typeConverterRepository);

parser.registerTypeConversion(java.util.Date.class, dateConverter);

parser.registerTypeConversion(java.sql.Date.class, dateConverter);

parser.registerTypeConversion(java.sql.Timestamp.class, dateConverter);
```

```

        return new JSONConfig(generator, parser);

    }

}

```

该类会创建一个生成器，它将 Java 类 (Event 或 Place) 转换成对应的 json，而解析器执行相反的过程。在 typeMapper (生成器和解析器都会用到它) 中有几个关键点需要注意，特别是基础类型和鉴别器域。

typeMapper.setEnforcedBaseType(AppDocument.class) 仅会转换继承自 AppDocument 类的文档。

typeMapper.setDiscriminatorField("docType") 将使用 docType 域和值来鉴别不同的文档类型。可以自由地将该域修改为其他的名字，但是这需要在 AppDocument 类中改变方法和 json 映射。为了刷新内存，可以参考下面的方法：

```

@JSONProperty(value = "docType", readOnly = true)

public String getDocumentType()

{
    return this.getClass().getSimpleName();
}

```

要注意的最后一个点是 typeMapper.setPathMatcher(new SubtypeMatcher(AppDocument.class))，它会自动查找子类型从而确保我们在继承自 AppDocument 的对象间转换。可以为检索或查询数据库的几个 jcouchdb 方法调用提供自己的解析器，但是在本教程不打算探讨那些内容。

既然有了我们需要的类，是配置 spring 上下文的时候了。我们将我们的 CouchDB 特有的配置项分离到 couchdb-config.xml 中。

```

couchdb-config.xml

<beans
    xmlns="http://www.springframework.org/schema/beans"

```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:util="http://www.springframework.org/schema/util"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.1.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang-3.1.xsd">

<context:annotation-config />

<bean id="properties"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
/>

<bean id="jsonConfigFactory"
class="com.clearboxmedia.couchspring.json.JsonConfigFactory"/>

<bean id="jsonConfig" factory-bean="jsonConfigFactory"

```

```

factory-method="createJsonConfig"/>

<!-- If my db requires username/password, I will need to set up a Principal -->
<bean id="couchPrincipal"
      class="org.apache.http.auth.UsernamePasswordCredentials">
    <constructor-arg value="${couchdb.username}" />
    <constructor-arg value="${couchdb.password}" />
</bean>

<bean id="serverFactory"
      class="com.clearboxmedia.couchspring.couch.CouchDbServerFactory" />

<bean id="couchDbServer" factory-bean="serverFactory"
      factory-method="createCouchDbServerInstance">
    <constructor-arg value="${couchdb.url}" />
    <constructor-arg name="credentials" ref="couchPrincipal" />
</bean>

<bean id="systemDatabase" class="org.jcouchdb.db.Database">
    <constructor-arg ref="couchDbServer" />
    <constructor-arg value="couchspring-dev" />
    <property name="jsonConfig" ref="jsonConfig" />
</bean>
</beans>

```

我们需要做的第一件事就是使用设置注解，它设置了 spring 上下文的注解。接下来的两个部分设置了 jsonConfigFactory ,使之做好了在服务器实例中使用的准备。最后，创建了用于创建 couchDbServer 实例的 serverFactory ，该实例随后和 jsonConfig 以及想要连接的数据库名称一起传入了 jcouchd database 实例。所有属性 (username,password 和 url) 现在都是通过命令行传入的，但是可以简单到仅提供一个指定的属性文件。

现在，我们已经配置好一切，是时候编写一些测试了。

创建、保存、检索、更新和删除

在深入探讨视图创建之前，先从一些基础测试开始，如：创建、更新、检索和删除。因为在每一个测试中我们都会对它们做一些事情。下面是 CouchSaveTest 类的定义，对于其他的测试来说也是如此。

```
CouchSaveTest.java (header)

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration("/root-context.xml")

public class CouchSaveTest {

    @Autowired

    protected Database database;

    ...

}
```

第一个注解@RunWith 告诉 Maven 使用 SpringJUnit4ClassRunner 运行该测试（不是标准的 JUnit 类运行器）。这样下一个注解 @ContextConfiguration("/root-context.xml") 才能为这个测试启动一个 Spring 上下文。该上下文会加载所有的 CouchDB 实体、POJOs 和它们的 JSON 注释以及会自动将视图更新到 CouchDB 服务器的 CouchDBUpdater。在下面的 Views 部分我们将会介绍最后一个注解。

最后，我们告诉 Spring 将数据库自动装配到该测试类中，使我们能够使用它。

Document creation **文档创建

无论在哪种类型的数据库存储系统中，创建新纪录的能力（本例中是文档）都是首要步骤之一。 使用 jcouchdb 的 API 如何实现这些呢？

```
CouchSaveTest.java (testEventSave())  
  
@Test  
  
public void testEventSave() {  
  
    Event document = new Event();  
  
    document.setTitle("Test");  
  
    assertTrue(document.getId() == null);  
  
  
    database.createDocument(document);  
  
    assertTrue(document.getId() != null);  
}
```

这里，我们创建了一个新的 Event 对象，然后将其作为参数调用了 database.createDocument() 方法。之后 JsonConfigFactory 会将我们的域映射到 CouchDB 文档。 [插入屏幕截图]

文档的检索和更新

```
CouchSaveTest.java (testEventSave_Update())  
  
@Test  
  
public void testEventSave_Update() {  
  
    Event document = database.getDocument(Event.class, "2875977125");  
  
    assertTrue(document != null);  
  
  
    document.setDescription("Testing out save");
```

```

        database.createOrUpdateDocument(document);

        Event newdocument = database.getDocument(Event.class, "2875977125");
        assertTrue(document != null);
        assertTrue(document.getDescription().equals("Testing out save"));
    }
}
    
```

该方法实际上测试了两件事情，首先通过调用 Event document = database.getDocument(Event.class, "2875977125"); 检索文档，检索时传入了文档的 id——“2875977125”。其次还测试了更新方法 database.createOrUpdateDocument(document);，它的作用正如其名，或创建一个新的文档，或更新一个已有的文档（意味着如果在数据库中能够找到匹配该 id 的文档时就会更新它）。

```

CouchSaveTest.java (testEventSave_Exists2())

@Test(expected = IllegalStateException.class)
public void testEventSave_Exists2() {
    Event document = database.getDocument(Event.class, "2875977125");
    assertTrue(document != null);
    database.createDocument(document);
    assertFalse(document.getId().equals("2875977125"));
}
    
```

如果我们试图创建一个已经存在的文档，最后一个测试会抛出一个异常（注意，我们没有使用 createOrUpdateDocument() 方法）。

文档删除

删除文档和创建、更新文档一样简单。

```

CouchDeleteTest.java (testEventDelete())

@Test
    
```

```
public void testEventDelete() {  
  
    Event document = database.getDocument(Event.class, "3083848875");  
  
    assertTrue(document != null);  
  
    database.delete(document);  
  
  
    try {  
  
        document = database.getDocument(Event.class, "3083848875");  
  
    }  
  
    catch (Exception e) {  
  
        assertTrue(e instanceof org.jcouchdb.exception.NotFoundException);  
  
    }  
  
}  
  
  
@Test(expected = org.jcouchdb.exception.NotFoundException.class)  
public void testEventDelete_NotExists() {  
  
    Event document = database.getDocument(Event.class, "-2");  
  
    assertTrue(document != null);  
  
    database.delete(document);  
  
    document = database.getDocument(Event.class, "-2");  
  
    assertTrue(document == null);  
  
}
```

这两个测试调用了 delete()方法分别对存在的和不存在的文档进行了删除，第二种情况会抛出 NotFoundException 异常。

查询文档

现在已经介绍完基础的 CRUD 操作，那么接下来需要做一些稍微复杂的工作。通过更多的方式查询数据库，而不仅仅是通过要查找的文档的 id。在本文中我们仅会探究视图的冰山一角，因为它们可以非常复杂。关于视图的更多内容可以在 [CouchDB wiki](#) 以及在线版的 [CouchDB: 权威向导](#) 中找到。

视图介绍

首先，CouchDB 视图究竟是什么，它如何工作？

视图是过滤或查询数据库中数据的一种方式。视图通常使用 JavaScript 编写，当然使用其他语言也可以编写视图，但那是另一话题，我们并不会在此进行介绍。每一个视图都将文档内的键映射到值。直到文档存取时才会更新视图索引，但是如果你愿意，你也可以通过外部脚本改变这个行为。当查询设计文档中的某一个视图时该文档中的所有视图都会被更新。

设计文档

在介绍视图创建之前我们应当讨论如何自动上传应用（并保持视图最新）。所有的视图都捆绑到一个设计文档。本例中我们将有两个设计文档：

event

place

`org.jcouchdb.util.CouchDBUpdater` 类会自动地创建这两个设计文档，该类是在 `couchdb-services.xml` 文件中配置的。

couchdb-services.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:context="http://www.springframework.org/schema/context"
```

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:annotation-config />

<bean id="systemUpdater"
      class="org.jcouchdb.util.CouchDBUpdater"
      init-method="updateDesignDocuments">
    <property name="createDatabase" value="true"/>
    <property name="database" ref="systemDatabase"/>
    <property name="designDocumentDir" value="designdocs"/>
  </bean>
</beans>
```

CouchDBUpdater 会监听 designdocs 目录中的变化，同时自动地将这些变化推到配置的 CouchDB 数据库。

那么 designdocs 目录实际上包含什么内容呢？

```
-designdocs
-event
```

```

-allByDate.map.js
-allByIdParent.map.js
-allByVenueId.map.js
-list.map.js
-place
-list.map.js

```

实际上每一个目录都映射到 CouchDB 中的一个设计文档。



Field	Value
_id	_design/event
_rev	1-0b959727fe402bf8a577e12a2f650f
language	javascript
views	<pre> allByParentId map function(doc) { if (doc.docType == 'Event') { emit(doc.parentId, null); } } allByVenueId map function(doc) { if (doc.docType == 'Event') { emit(doc.venueId, null); } } list map function(doc) { if (doc.docType == 'Event') { emit(doc.id, null); } } </pre>

图 7 DesignDocs 事件

很好，接下来就让我们编写这些视图。

我们的第一个视图

下面是一个简单的视图，它会查找所有的 “event” 类型的文档。

```

function(doc) {
  if (doc.docType == 'Event') {
    emit(doc.id, doc);
  }
}

```

该视图会简单的返回所有具有 docType 域并且该域值为 Event 的文档的 id。让我们稍微检查一下它做了什么。第一行是一个 JavaScript 函数定义，它仅接受一个 doc 参数。然后就能够检查存储在文档内的值(本例中是 doc.docType)。

最后，调用了 emit 函数，它有两个参数 key 和 value，其中 value 可以为 null。在本例中，key 是 doc.id 域，value 是整个文档。

在接下来的几个视图示例中，emit 函数才是我们实际上用到的查询数据库的方法。关于 emit，我们需要理解的另外一关键点是，它会对返回的文档按照 key 值进行排序。

下面是调用 list 视图的测试用例。

```
CouchViewTest.java (testListingQuery())
```

```
@Test

public void testListingQuery() {

    String viewName = "event/list";

    ViewResult results = database.queryView(viewName, Event.class, null, null);

    assertNotNull(results);

    assertEquals(27, results.getRows().size());

}
```

通过 venue id 检索 events

为了方便使用，首先需要创建的视图之一就是通过关联的 venueId 检索给定事件集合的视图。为此，需要编写一个 key 为 venueId、值为 document 的视图（尽管 jcouchdb 的函数并没有严格的需求）。那么，该视图是什么样的呢？

```
function(doc) {

    if (doc.type == 'Event') {

        emit(doc.venueId, {doc});

    }

}
```

它和之前编写的那个简单视图非常相似，但是这次从应用程序中调用该视图进行查询时需要传递一个 venueId。

```
CouchViewTest.java (testQueryByVenueId())
```

```
@Test

public void testQueryByVenueId() {

    String viewName = "event/allByVenueId";

    ViewAndDocumentsResult
```

此处关于如何调用视图的关键区别之一就是使用 queryViewAndDocumentsByKeys() 方法传入了 viewName、映射的 Event 类以及要查询的键（这种情况下只会查询出键值为指定 venueId 的事件）。

通过日期查询事件

这两个视图都比较简单。而通过日期查询这种稍微复杂一点功能的如何实现呢？首先，我们需要定义视图。

```
function(doc) {

    if (doc.docType == 'Event') {

        var startDate = new Date(doc.startTime);

        var startYear = startDate.getFullYear();

        var startMonth = startDate.getMonth();

        var startDay = startDate.getDate();

        emit([
            startYear,
            startMonth,
            startDay
        ]);

    }
}
```

现在，我们如何调用这个函数呢？

[todo: 在 java 中调用视图的代码示例]

从事件中检索 venue 的动态查询

CouchViewTest.java (testQueryByDate())

```

    @Test

    public void testQueryByDate() {

        String viewName = "event/allByDate";

        Options opts = new Options();

        opts.startKey(Arrays.asList(2013, 7, 11));
        opts.endKey(Arrays.asList(2013, 7, 11));

        ViewAndDocumentsResult
    }

```

此处有一个名为 Options 的新对象，通过它可以指定想要传入视图的查询选项。在本例中，我们提供了一个 startKey 和一个 endKey 去检索对象集合。需要注意的一件事情就是，你要返回或匹配的数据需与传入数据的类型保持一致。在本例中处理的是 int 类型，所以传入的键必须是 int 类型的域。当然顺序也是键，我们依次传入了年、日、月从而匹配视图中的年、日、月。

那么，endKey 是什么呢？实际上，endKey 参数允许我们指定查询的范围。在本例中我们选择了相同的日期，但是可以很容易地选择不同的值从而返回更多或更少的文档。CouchDB 会简单地依次比较每一个键，直到再没有匹配的数据时才会返回结果文档集合。

从事件中检索 venue 的动态查询

除了通过 Event id 查询地址 (place) 之外，此处所要做的事情就是编写和之前 queryByVenueId 相同的逻辑。

```

    @JsonProperty(ignore = true)

    public Place getVenue() {

        String viewName = "place/allByEventId";

        ViewAndDocumentsResult
    }

```

你仅需要为 place 文档编写一个和 allByVenueId 相似的视图，仅此而已。

接下来去向何方？

视图（或 map）只是 CouchDB 所提供的 map/reduce 功能的第一部分。那么什么是 reduce（和 re-reduce）功能，它能给我们带来什么呢？



Reduce 允许我们从之前的 map 中获取结果集，同时能对其执行附加的操作将结果集分解成更加简洁的形式。

Reduce 和 re-reduce 的功能需要你自己去探索，但是你可以通过它们做一些非常有趣的事情。探索并感受 CouchDB 带来的乐趣！

关于作者

Leo Pryzbylski 在 ClearBox 媒体是技术创新的标杆人物。他在挥舞着一个创造性解决问题的巨大棒槌的同时，还痴迷于软件体系结构，富有与软件不切题的很多问题战斗的经验。Leo 具有广泛的技能，这得益于他作为游戏开发者、QA 工程师、发布管理者、配置管理员、开发管理者、计算机科学家、网络入侵专家、嵌入式软件工程师、软件架构师、科学程序员和系统管理员的经验。

Warner Onstine 的 IT 生涯开始于 90 年代早期对 Intuit 的技术支持工作。那个时候他学会了如何开发 web 应用程序，同时开始了自己的软件工程师生涯。在那之后，他在很多地方工作过，包括 Intalio、亚利桑那大学，现在作为开发经理在 rSmart 工作。ClearBox Media 的萌芽始于 Warner 学习 ARG 的时候，同时在几年之前就开始从中找乐了。它们非常吸引人，因为无论是在现实生活还是虚拟环境中它们都是良好的调味剂。Warner 以及其他发现了 ARG 能让社会变得更好的潜能，同时也是 “The Human Mosaic Project” 的指导性原则之一——感受乐趣。做得好。

原文链接：<http://www.infoq.com/cn/articles/warner-couchdb>

相关内容：

Julien Nioche 谈 Apache Nutch 2 的特性及产品路线



作者 [Frank Buschmann](#) 译者 [赵震一](#)

开源的 Web 搜索框架 Apache Nutch 的 [2.1](#) 版本已于 2012 年 10 月 5 日发布，该版本的新特性包括：支持一些改进属性，用于更好地配置 Solr；更新到各个 Gora 依赖；可以选择构建[弹性搜索](#)中的索引。Nutch 既可以运行在单台服务器上，也可以用作大规模抓取平台运行在 Hadoop 集群上。

Nutch 框架的 2.0 版本在经历了两年开发之后，已于今年 7 月[发布](#)，该版本以 [Apache Gora](#) 框架作为其存储抽象而构建。Apache Gora 开源框架提供了一种内存数据模型，并支持大数据的持久化。它支持将数据持久化到列存储、键值存储、文档存储和关系数据库中，还可以利用大量 Apache Hadoop 的 MapReduce 支持工具来分析数据。在今年早些时候，Gora 已成为 Apache 的顶级项目。

Nutch 2 支持大数据存储方案，如分布式键值存储 [Apache Accumulo](#)、数据序列化系统 [Apache Avro](#)、列族数据存储 [Apache Cassandra](#)、分布式大数据存储 [Apache HBase](#) 和 Hadoop 分布式文件系统（HDFS）。

InfoQ 采访了 Apache Nutch 项目的副总裁 Julien Nioche，他也是 DigitalPebble Ltd 的主管。他将于 2012 年 11 月 7 日在 [Apache Conference Europe](#) 上介绍如何使用 Nutch 框架进行大规模抓取。

InfoQ :Apache Nutch framework 的哪些方面使其适合 NoSQL 数据库和大数据领域？

 **Julien :** Nutch 的确是被打上了“大数据”的标签。一方面，Apache Hadoop 就是产生自 Nutch 项目，而 Apache Hadoop 现在已经是用于大规模数据处理的事实上的标准框架。Nutch 是为大规模 Web 数据抓取而设计的。有些用户使用由数以百计的服务器组成的集群来运行 Nutch 并保存了数十亿的页面。

至于它与 NoSQL 的关系，这正是 Nutch 2 要解决的问题。Nutch 1.x 分支依赖于 Hadoop 数据结构，这非常适合批处理任务；而 Nutch 2 则依赖于 Apache GORA 来提供一个在各种 NoSQL 数据存储之上的统一前端。

InfoQ : Apache Gora 框架也是出自 Nutch 项目。作为 NoSQL 数据库的一个 ORM 框架，你能否谈论一下它对应用程序开发人员有何帮助？

 **Julien** : 我喜欢把 GORA 当作 “NoSQL 数据库的 JDBC” ，因为它在存储之上提供了一层抽象，允许开发者编写独立于任何特定 API 的代码。GORA 的部分 API 也提供了一种在不同后端之上的 MapReduce API，此外还有一种基于 Apache AVRO 的序列化机制。当然，它也支持基本的 GET-PUT-DELETE 等原子操作。

Apache GORA 现在是 2.1 版本了，支持 HBase、Cassandra 和 Accumulo 等数据存储，而且它还有一个 SQL 模块！这意味着用户可以在 MySQL 数据库之上运行 MapReduce，一些 Nutch 2 用户正是这么做的。实际上，通过 Nutch 2 我们也发现，人们喜欢不同的存储方式，因此 GORA 是非常有用的。

InfoQ 最新的版本也支持 HTML 解析，这是利用 Apache Tika 框架来处理的。你能详细描述一下这一特性是如何工作的吗？

 **Julien** : Apache Tika 是一个用 Java 实现的开源库，支持从多种格式中（如 HTML、PDF 和 Word 等）抽取文本和元数据，也能用于语言和 MIME 类型识别。实际上它就是现有的第三方解析器（如 PDFBox）的包装器，只是提供了一个统一的 API 来使用这些解析器。在 Nutch 1.x 分支和 Nutch 解析器遗留系统中，Tika 已经有所应用了，因此它并不是 Nutch 2.0 中的新东西。有趣的是，就像 Hadoop 和 GORA 一样，Apache Tika 是也是一个脱胎于 Nutch 的项目。

InfoQ : 在即将发布的版本和特性方面，Nutch 项目未来的路线图是什么样的？

 **Julien** : Nutch 的发布并没有遵循一个严格的时间表。基本上是这样，当我们认为大部分工作已经完成时，就会发布新的版本，而工作的完成又要依赖于有多少贡献者参与和用户最快要多久才能采用新项目等因素。Nutch 1.x 和 2.x 必定会共存一段时间，直到 2.x 完全成熟为止。尽管如此，它们的发

布不太可能完全同步。近来，我们平均每年发布两个版本，但随着 2.x 版吸引力的增加，我们可能会加快新版本发布的频率。

至于特性，最重要的一个就是升级到 SOLR 4，再就是它的云功能。我们可能也会看到更多的功能将委托给第三方项目，比如 [Crawler Commons](#)，这样其他项目就能复用并改进代码了。我们也考虑过将索引后端变为可插拔的：虽然目前只支持 SOLR（2.x 版支持 ElasticSearch），但是我们希望开发者能够使用插件机制编写新的索引后端，这样就无需捆绑 Nutch 代码了。把网页排名功能交给 [Apache Giraph](#) 等图库，我们可以少编写很多代码，而且更有效率。我希望将大部分精力放在巩固 2.x 的代码上。

他也谈到了完成这一项目所用的十年：

Julien : Apache Nutch 最近已经 10 岁了，对软件而言，这已经算很老了。

“为什么它仍然存在呢？我想原因是，它能够把要做的事情做好，并且没有试图重复发明轮子。有趣的是，现在很多源自 Nutch 的项目，如 Hadoop 或 Tika 等，它们的发展也让 Nutch 获益匪浅。我希望这一切也会出现在 GORA 身上。Nutch 2 的开发非常激动人心，而且我们看到非常多的新用户选择了它。在开发过程中，不断有新的贡献者和提交者加入进来，而这正是一个项目健康的标志。

7 月份，Apache Nutch 团队[宣布](#)发布了 Nutch v1.5.1。这是 Nutch 框架 1.5.x 主干版本的一个维护性发布。具体信息见该版本的[修改列表](#)。该搜索框架可以从网络上[下载](#)。Nutch 的文档和教程等资料请参见项目的[wiki](#) 页面。



关于被采访人

Julien Nioche 是 DigitalPebble Ltd 的创始人，这是一家位于英国布里斯托尔的咨询公司，专注于为文本工程提供开源解决方案。Julien 的专长涉及信息检索、文本分析、信息抽取、自然语言处理和机器学习等领域。他还是 Apache Nutch 项目的副总裁、Apache Tika 和 Apache Gora 的提交者以及其他一些开源项目的贡献者。

原文链接：<http://www.infoq.com/cn/articles/nioche-apache-nutch2>

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
跟着李航学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转型构
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
虚拟机设计：Node.js生态系统之
秘密，操作实践
使用Java连接JavaScript并行编程
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
分布式平台的组件化PaaS
社区驱动的开源计划
来自Padmanab的真言

InfoQ 每月8号出版

推荐文章 | Article

注重实效的架构师——大胆行前人未行之路

作者 [Frank Buschmann](#) 译者 [赵震一](#)

本文首次发表在 *IEEE Software*，并由 InfoQ 和 IEEE 计算机协会为您引进

是什么让架构师们精通自己的技艺？熟练的架构师是如何进行设计的？一次次，有人问起我这些问题，而我也不止一遍的问我自己。很明显，这不只是软件工程过程、设计方法、技术或是编程的专业程度所决定的。很多架构师具备令人钦佩且完备的技术知识，这确实是使设计成功的必要条件。但是，还是有很多的软件项目失败了，或是在项目的架构中遭受到了严峻的挑战。掌握此道的关键在于架构师是以什么方式实现设计，他们重视什么，他们关注哪些方面以及在这些方面努力着。

(缺失的) 线条能告诉我们什么

图 1 展示了摘自以前项目的一张高层次图表。那个项目的架构师创建了该图以阐述系统的基本设计。

该图大致描述了系统的关键组件、各组件的职责以及它们核心的模块化原理。架构师使用这些概要图来交流设计，以管理开发的过程，并以此讨论它们在业务方面产生的影响，例如：成本、时间和精力等。然而，一段时间后，管理层会想要知道项目为什么会有重大延期以及预算为什么会超支，他们无法从架构师的报告中获得任何提示或指标。

问题来自于该图上的点状黑线以及那些“尚未显示的线条”。这些点状黑线表示了该项目开发的一个专有的消息中间件。在图上无法找到所罗列的这些组件之间的通信关系，特别要强调的是，这些关系是通过这个中间件运作的。该项目的架构师并不认为值得对这些方面值建模或报告，因为从他们的视角来看，它们代表的是基本技术的基础设施，并不会有助于系统的领域和业务用例。然而，中间件的开发消耗了大量预算，因为这个过程中涉及到很多健壮性和性能的问题，我们不得不从系统的其他部分中抽调最好的开发人员来解决这些缺陷。另外，支撑这些中间件问题也需要在领域组件中付出相当多的努力。

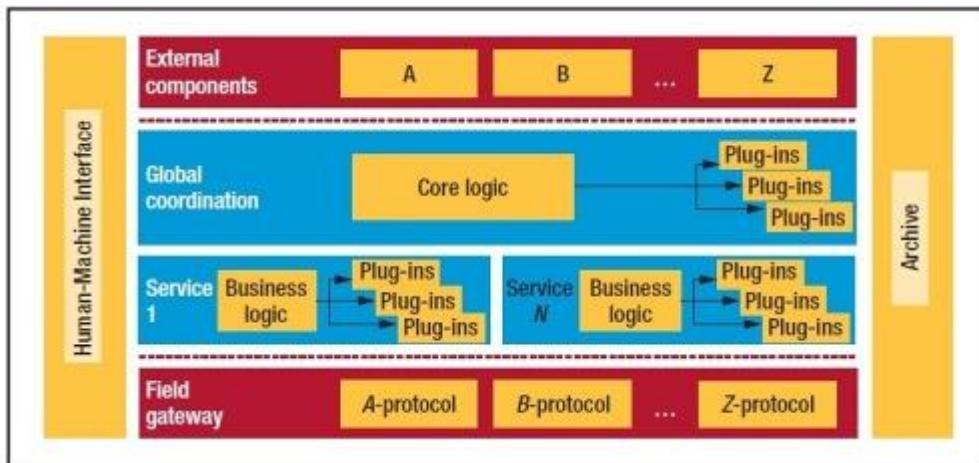


图 1.一幅用于和业务利益相关者交流关键设计的高层次架构概要图

事物间的设计

过往的经验显示，系统的架构师们常常将注意力过多地集中于一些“明显”的事物上：用户接口、领域特定组件、数据管理以及持久化等。然而架构的问题并不在组件之内，而是在组件之间：与其他系统之间的接口，交互以及集成——包括底层的技术基础设施。但是架构规范里几乎不涉及这些方面，所以在这些方面发生问题之前，架构师和开发人员都不会给予关注。

与此相反，注重实效的架构师们更关注事物之间的事物——就是说，组件之间的事物以及代码行（LOC）之间的事物，例如标准数据类型背后的领域思想。当然，他们也会指导系统实际组件的设计，但是当指导通常足够充分以支持更进一步拆分并且可由开发团队实现时，这些事物之间的事物实际上需要架构师们亲手处理。而且必须是他们，这是他们的领域——特别是在我们还不确定如何实际地设计这些“事物”时。下面让我们去探究一下注重实效的架构师们的秘密。

发现隐藏的领域概念

最近我在我们的几个系统的代码上运行 tag-cloud 生成器。我想通过这种方式对这些系统中重要的领域概念有个大致的印象。出人意料的是，在这些系统中最顶端的数据类型是 string 和 int。然而我怀疑是不是真的如此，因为在针对工业工程或能源管理的系统中，其他概念会更加重要：设备、电力线、传感器、制动器、标签、警报等等。当我更深入地查看代码时，就发现了这些领域概念——

但是它们分散在表面上几乎不怎么相关的诸如 string 和 int 这样基本类型的配置中。

如此隐藏的领域概念可能需要开发者们花费相当多的努力来理解和实现系统，以保证产品的质量。我如何才能知道一个 int 实际表示的是某个特定的领域概念？我如何保证在某个特定的计算上下文中使用 int 时会执行特定领域概念的契约？我不能，除非通过注释和约定，其他的都对实践没有实质帮助。图 2 展示了摘自导致 Ariane 5 在其 1996 年首飞时坠毁的（标有注释的）代码片段，1 其根本原因是源于对整型数溢出的保护不足。²

Why visibility matters—the Ariane 5 crash

- Velocity was represented as a 64-bit float
- A conversion into a 16-bit signed integer caused an overflow
- The current velocity of Ariane 5 was too high to be represented as a 16-bit integer
- Error handling was suppressed for performance reasons

```

-- Vertical velocity bias as measured by sensor
L_M_BV_32 := 
TDB.T_ENTIER_32S ((1.0/C_M LSB_BV) *
G_M_INFO_DERIVE(T_ALG.E_BV));
-- Check, if measured vertical velocity bias can be
-- converted to a 16 bit int. If so, then convert
if L_M_BV_32 > 32767 then
    P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elseif L_M_BV_32 < -32768 then
    P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
    P_M_DERIVE(T_ALG.E_BV) :=
        UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));
end if;
-- Horizontal velocity bias as measured by sensor
-- is converted to a 16 bit int without checking
P_M_DERIVE(T_ALG.E_BH) :=
    UC_16S_EN_16NS(TDB.T_ENTIER_16S ((1.0/C_M LSB_BH) *
G_M_INFO_DERIVE(T_ALG.E_BH)));

```

*Source: <http://moscova.inria.fr/~levy/talks/10enslongo/enslongo.pdf>

图 2. 摘自 Ariane 5 的代码片段。因为对整型数溢出的保护不足导致了 Ariane5 在其 1996 年首飞时坠毁。

我们有足够的勇气说，如果开发者以定义好的方式使用契约，将速度建模成一种合适的类型，那么 Ariane 5 的软件错误原本是可以避免的。然而这个例子也很好的把握了显式建模的重要性。

即使领域概念很明显、很清楚，但也可能埋藏在无数的细节之下。举个例子，我曾经研究过一个系统的复杂性。该软件包含了一个命名服务，该服务拥有一个包括 300 多种方法的接口。该服务的实际契约已经几乎看不清楚，而开发者们需

要非常努力才能正确和有效地使用它。有分析表明，不超过 20 个方法就完全可以说明一个服务的必要契约。

注重实效的架构师因此会非常重视并做好相关的工作，在他们的架构中对所有领域概念进行明确的描述，例如那些常被粗粒度描述的组件，细微的特定领域的数据类型，有意义的接口，等等。^{3,4} 在对概念的建模中，注重实效的架构师总是专注于精简而避免混乱或复杂，并着重强调概念的本质。⁴ 这样一来，系统中那些隐藏的概念或是相关的性能都会立即明朗起来，这将会帮助开发人员更好的识别它们，并把它们看作独特的、明确的、有意义的类型(types)。用途(Usage)转变成类型——对于创建有表现力的软件设计和健壮的实现来说是一种重要的实践。

在事物衔接之处

什么是架构？Eoin Woods 对于该问题思考了相当长一段时候后找到了一个答案。“那些很多被架构师每天在使用着的思想：框架，代理，层次，接口，消息通知，连接器...这都是与间隙（ gaps ）相关的！ [...] 架构是一种用于连接软件设计师们一起工作的粘合剂，共同创造一个弹性的、灵活的、可扩展的以及最终可用的系统。”⁵

在这个结论中包含着很多道理。我确定所有人都知道关于组件接口不支持工作流，而我们系统不得不支持的事。在很多项目中，集成——无论是系统集成还是“仅仅”是系统构成组件的集成——是成本最高的地方。而间隙（ gaps ）是指事物间衔接处的空间，组件交互的地方——没有一方会对其负责，除了注重实效的架构师！所以设计简洁及有意义的组件接口来支持组件间工作流的实现确实是不平凡的任务。在组件间定义符合用户在系统上执行任务那样的交互是一件困难的事。⁶ 甚至更难的是将一个系统与其他系统集成，使其在不丧失任一相关系统独立质量的情况下支持跨系统的工作流。你可以快速浏览一下支持该结论主题方面的模式著作。^{7,8}

“空隙（ void ）”事实上需要架构师更多的关注和亲身实践。然而重点并非事物之间的适配——接口，交互，组件，系统——这些肯定是要连接的；重要的是随着对工作流的支持之后这些事物之间的协调性。⁶ 目标是最精益（ leanest ）的适配，并且最深刻（ impressive ）的映射！这就是架构产生的地方；此处的决

策将对系统的生命周期成本产生非常大的影响。本段开篇提到的战争故事也就恰好定格在这里：事物在哪里衔接。

我们可以对独立系统间那些“在中间（in-between）”的所有类型的工件的设计思想进行扩展——举个例子，驻留于不同计算节点上，在不同的进程中或是在不同的线程中各个部分。我曾看到过一些失败的项目，之所以失败是因为在他们的处理实体（processing entities）之间粘合的时候采用了一种幼稚的方式。其实在跨越计算机、进程和线程的分布式实体之间建立工作交互比较简单。这就是对设计的掌控，无论如何，创建（分布式）进程之间的交互可以最小化网络交互和对线程间同步的需求。⁷

以不确定作为驱动

我们都会抱怨那些模糊的系统需求。然而纵使业务的利益相关者们进行了最仔细的需求捕获工作，这都无法完全解决所有的歧义和不确定性。这就是事实。同样，软件工程师可能需要针对特定的需求费心选择各种可选解决方案，并在讨论该采用哪种方案上花费大量时间。但是架构师则必须要面对来自设计、开发以及交付系统带来挑战，并且在系统发布和后续的整个生命周期中都能保证满足相关的业务需求。系统开发的时间越长，系统的生命周期越长，不确定性也就越大。

在这种形势下，架构师们会选择一种最典型的规避方式，那就是泛型——它可以最大程度地带来灵活性。架构通过弹性机制来应对过载，这在理论上支持所有可想象的系统配置，但是并不能满足任何有意义配置的具体（非功能性）需求。

⁹

注重实效的架构师能察觉到危险地带并以不确定性为驱动来做好决策。¹⁰首先，他们承认需要在各种可选项中做好选择，并针对他们设计中的可变方面做出工作，以此来限制变化或选择造成的影响。他们会深入探索系统的使用场景，以此来澄清需求中的不确定性或对一种特殊设计选项的选择，从而实现出场景的原型或可运行骨架系统（walking skeleton）的一个部分。⁹并且，他们非常欢迎来自于原型和可运行骨架系统的循环反馈，以驱动或调整他们的决策。架构师们重复地在事物之间（本例中是指在有歧义和不确定的需求或可选设计选项之间）进行着设计。

本文的标题是“大胆行前人未行之路”，架构设计恰好就需要如此。来到事物之间：在代码行上或其间，发现隐藏的领域概念；在你的系统和其他系统的组件之间，可以引导你设计好接口和工作流；在不确定性及可选项之间，驱动你的决策。架构师的工作就是去尽早地发现这些“在中间的”事物，使它们更加明确，从中做出决策。以上这些加上扎实的有关架构方法和技术的专业知识，以及谨慎的实践，就是对架构的精通之道：在软件系统的痛点上进行深思熟虑并最终决定它的成败。

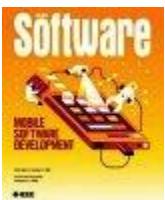
参考文献

1. J-J. Levy, "[Un Petit Bogue, Un Grand Boum!](#)" (in French), 2010.
2. J-L. Lions et al., "[Ariane 501 Inquiry Board Report,](#)" 1996.
3. E. Evans, Domain Driven Design, Addison-Wesley, 2004.
4. F. Buschmann and K. Henney, "Five Considerations for Software Architecture, Part 1," IEEE Software, vol. 27, no. 3, 2010, pp. 63-65.
5. E. Woods, "[Architecting in the Gaps,](#)" 2011.
6. F. Buschmann, "Unusable Software Is Useless, Part 1," IEEE Software, vol. 28, no. 1, 2011, pp. 92-94.
7. F. Buschmann, K. Henney, and D.C. Schmidt, Pattern-Oriented Software Architecture-A Pattern Language for Distributed Computing, John Wiley and Sons, 2007.
8. G. Hohpe and B. Woolf, Enterprise Integration Patterns-Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, 2003.
9. F. Buschmann, "Learning from Failure, Part 2: Featuritis, Permititis, and Other Diseases," IEEE Software, vol. 27, no. 1, 2010, pp. 10-11.
10. K. Henney, "Use Uncertainty as a Driver," 97 Things Every Software Architect Should Know, R. Monson-Haefel, ed., O'Reilly, 2009, pp. 321-361.

关于作者



Frank Buschmann 是西门子公司研究院高级首席工程师，他同时也是系统架构和平台部门的首席架构师。可以通过 frank.buschmann@siemens.com 与他联系。



本文首先发表在 [IEEE Software](#) 杂志。[IEEE Software](#) 的使命是打造一个引领未来软件实践的社区。该杂志传递可靠，有用，前沿的软件开发信息，以保持工程师与管理者对技术变革快节奏的把握。

聊聊并发（四）——深入分析 ConcurrentHashMap

作者 方腾飞

术语定义

术语	英文	解释
哈希算法	hash algorithm	是一种将任意内容的输入转换成相同长度输出的加密方式，其输出被称为哈希值。
哈希表	hash table	根据设定的哈希函数 $H(key)$ 和处理冲突方法将一组关键字映象到一个有限的地址区间上，并以关键字在地址区间中的象作为记录在表中的存储位置，这种表称为哈希表或散列，所得存储位置称为哈希地址或散列地址。

线程不安全的 HashMap

因为多线程环境下，使用 HashMap 进行 put 操作会引起死循环，导致 CPU 利用率接近 100%，所以在并发情况下不能使用 HashMap，如以下代码

```

final HashMap<String, String> map = new HashMap<String, String>(2);

Thread t = new Thread(new Runnable() {

    @Override

    public void run() {

        for (int i = 0; i < 10000; i++) {

            new Thread(new Runnable() {

                @Override

                public void run() {

```

```
        map.put(UUID.randomUUID().toString(), "");  
    }  
}, "ftf" + i).start();  
}  
}  
}, "ftf");  
t.start();  
t.join();
```

效率低下的 HashTable 容器

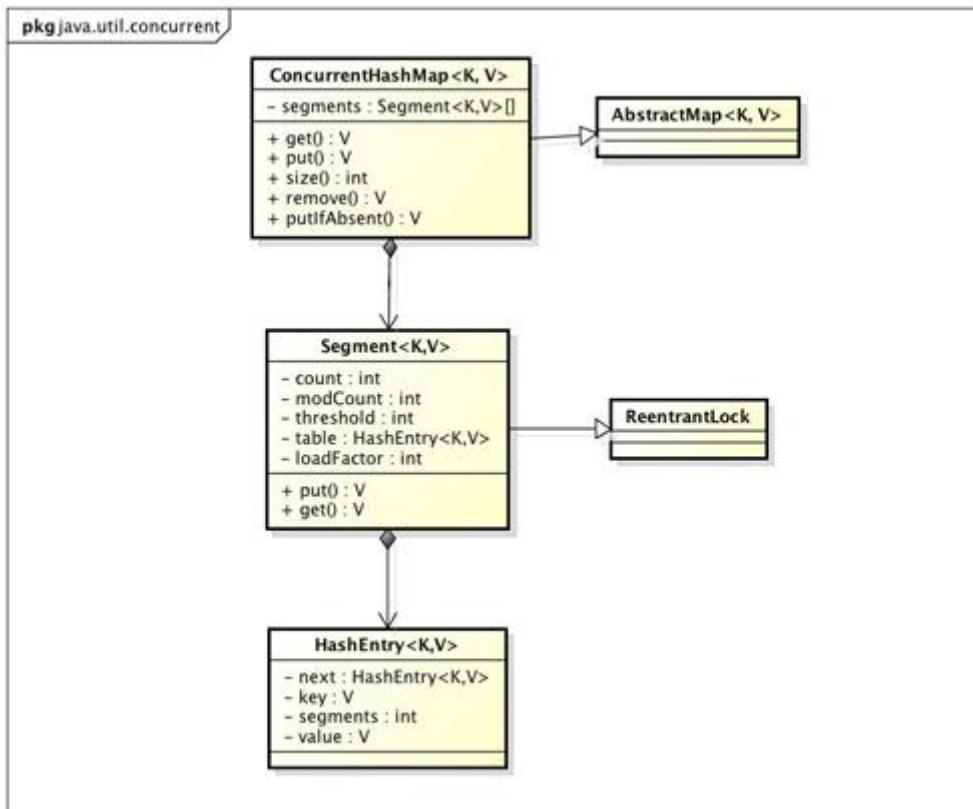
HashTable 容器使用 synchronized 来保证线程安全 ,但在线程竞争激烈的情况下 HashTable 的效率非常低下。因为当一个线程访问 HashTable 的同步方法时 , 其他线程访问 HashTable 的同步方法时 , 可能会进入阻塞或轮询状态。如线程 1 使用 put 进行添加元素 , 线程 2 不但不能使用 put 方法添加元素 , 并且也不能使用 get 方法来获取元素 , 所以竞争越激烈效率越低。

锁分段技术

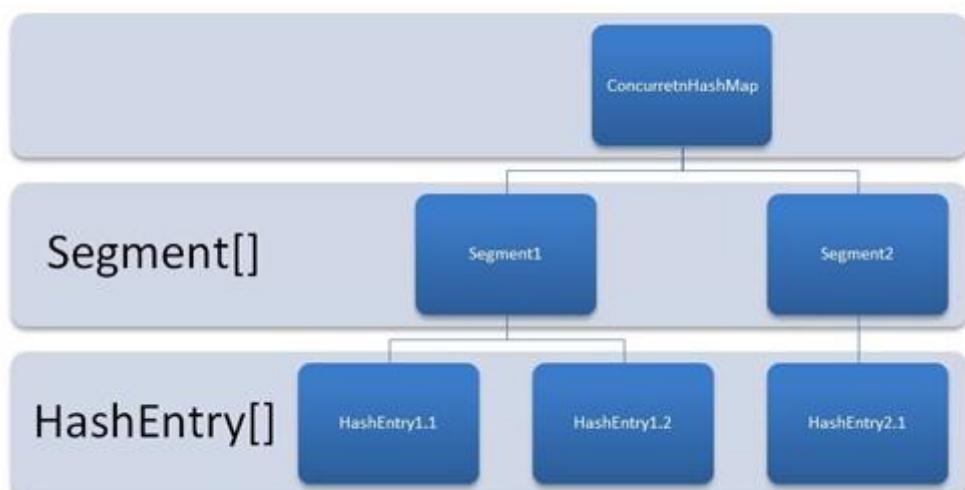
HashTable 容器在竞争激烈的并发环境下表现出效率低下的原因是所有访问 HashTable 的线程都必须竞争同一把锁 , 那假如容器里有多把锁 , 每一把锁用于锁容器其中一部分数据 , 那么当多线程访问容器里不同数据段的数据时 , 线程间就不会存在锁竞争 , 从而可以有效的提高并发访问效率 , 这就是 ConcurrentHashMap 所使用的锁分段技术 ,首先将数据分成一段一段的存储 , 然后给每一段数据配一把锁 , 当一个线程占用锁访问其中一个段数据的时候 , 其他段的数据也能被其他线程访问。

ConcurrentHashMap 的结构

我们通过 ConcurrentHashMap 的类图来分析 ConcurrentHashMap 的结构。



ConcurrentHashMap 是由 Segment 数组结构和 HashEntry 数组结构组成。Segment 是一种可重入锁 ReentrantLock ,在 ConcurrentHashMap 里扮演锁的角色 , HashEntry 则用于存储键值对数据。一个 ConcurrentHashMap 里包含一个 Segment 数组 , Segment 的结构和 HashMap 类似 , 是一种数组和链表结构 , 一个 Segment 里包含一个 HashEntry 数组 , 每个 HashEntry 是一个链表结构的元素 , 每个 Segment 守护者一个 HashEntry 数组里的元素,当对 HashEntry 数组的数据进行修改时 , 必须首先获得它对应的 Segment 锁。



ConcurrentHashMap 的初始化

ConcurrentHashMap 初始化方法是通过 initialCapacity , loadFactor, concurrencyLevel 几个参数来初始化 segments 数组 段偏移量 segmentShift , 段掩码 segmentMask 和每个 segment 里的 HashEntry 数组 。

初始化 segments 数组。让我们来看一下初始化 segmentShift ,segmentMask 和 segments 数组的源代码。

```
if (concurrencyLevel > MAX_SEGMENTS)
    concurrencyLevel = MAX_SEGMENTS;

// Find power-of-two sizes best matching arguments

int sshift = 0;
int ssize = 1;

while (ssize < concurrencyLevel) {
    ++sshift;
    ssize <<= 1;
}

segmentShift = 32 - sshift;
segmentMask = ssize - 1;
this.segments = Segment.newArray(ssize);
```

由上面的代码可知 segments 数组的长度 ssize 通过 concurrencyLevel 计算得出。为了能通过按位与的哈希算法来定位 segments 数组的索引，必须保证 segments 数组的长度是 2 的 N 次方 (power-of-two size) ，所以必须计算出一个大于或等于 concurrencyLevel 的最小的 2 的 N 次方值来作为 segments 数组的长度。假如 concurrencyLevel 等于 14 , 15 或 16 , ssize 都会等于 16 , 即容器里锁的个数也是 16。注意 concurrencyLevel 的最大大小是 65535 , 意味着 segments 数组的长度最大为 65536 , 对应的二进制是 16 位。

初始化 segmentShift 和 segmentMask。这两个全局变量在定位 segment 时的哈希算法里需要使用，sshift 等于 ssize 从 1 向左移位的次数，在默认情况下 concurrencyLevel 等于 16，1 需要向左移位移动 4 次，所以 sshift 等于 4。segmentShift 用于定位参与 hash 运算的位数，segmentShift 等于 32 减 sshift，所以等于 28，这里之所以用 32 是因为 ConcurrentHashMap 里的 hash() 方法输出的最大数是 32 位的，后面的测试中我们可以看到这点。segmentMask 是哈希运算的掩码，等于 ssize 减 1，即 15，掩码的二进制各个位的值都是 1。因为 ssize 的最大长度是 65536，所以 segmentShift 最大值是 16，segmentMask 最大值是 65535，对应的二进制是 16 位，每个位都是 1。

初始化每个 Segment。输入参数 initialCapacity 是 ConcurrentHashMap 的初始化容量，loadfactor 是每个 segment 的负载因子，在构造方法里需要通过这两个参数来初始化数组中的每个 segment。

```

if (initialCapacity > MAXIMUM_CAPACITY)
    initialCapacity = MAXIMUM_CAPACITY;
int c = initialCapacity / ssize;
if (c * ssize < initialCapacity)
    ++c;
int cap = 1;
while (cap < c)
    cap <<= 1;
for (int i = 0; i < this.segments.length; ++i)
    this.segments[i] = new Segment<K,V>(cap, loadFactor);
    
```

上面代码中的变量 cap 就是 segment 里 HashEntry 数组的长度，它等于 initialCapacity 除以 ssize 的倍数 c，如果 c 大于 1，就会取大于等于 c 的 2 的 N 次方值，所以 cap 不是 1，就是 2 的 N 次方。segment 的容量 threshold = (int)cap*loadFactor，默认情况下 initialCapacity 等于 16，loadfactor 等于 0.75，通过运算 cap 等于 1，threshold 等于零。

定位 Segment

既然 ConcurrentHashMap 使用分段锁 Segment 来保护不同段的数据，那么在插入和获取元素的时候，必须先通过哈希算法定位到 Segment。可以看到 ConcurrentHashMap 会首先使用 Wang/Jenkins hash 的变种算法对元素的 hashCode 进行一次再哈希。

```
private static int hash(int h) {
    h += (h << 15) ^ 0xffffcd7d;
    h ^= (h >>> 10);
    h += (h << 3);
    h ^= (h >>> 6);
    h += (h << 2) + (h << 14);
    return h ^ (h >>> 16);
}
```

之所以进行再哈希，其目的是为了减少哈希冲突，使元素能够均匀的分布在不同的 Segment 上，从而提高容器的存取效率。假如哈希的质量差到极点，那么所有的元素都在一个 Segment 中，不仅存取元素缓慢，分段锁也会失去意义。我做了一个测试，不通过再哈希而直接执行哈希计算。

```
System.out.println(Integer.parseInt("0001111", 2) & 15);
System.out.println(Integer.parseInt("0011111", 2) & 15);
System.out.println(Integer.parseInt("0111111", 2) & 15);
System.out.println(Integer.parseInt("1111111", 2) & 15);
```

计算后输出的哈希值全是 15，通过这个例子可以发现如果不进行再哈希，哈希冲突会非常严重，因为只要低位一样，无论高位是什么数，其哈希值总是一样。我们再把上面的二进制数据进行再哈希后结果如下，为了方便阅读，不足 32 位的高位补了 0，每隔四位用竖线分割下。

0100 0111 0110 0111 1101 1010 0100 1110
1111 0111 0100 0011 0000 0001 1011 1000
0111 0111 0110 1001 0100 0110 0011 1110

1000 | 0011 | 0000 | 0000 | 1100 | 1000 | 0001 | 1010

可以发现每一位的数据都散列开了，通过这种再哈希能让数字的每一位都能参与到哈希运算当中，从而减少哈希冲突。 ConcurrentHashMap 通过以下哈希算法定位 segment。

```
final Segment<K,V> segmentFor(int hash) {
    return segments[(hash >> segmentShift) & segmentMask];
}
```

默认情况下 segmentShift 为 28，segmentMask 为 15，再哈希后的数最大是 32 位二进制数据，向右无符号移动 28 位，意思是让高 4 位参与到 hash 运算中， $(hash >> segmentShift) \& segmentMask$ 的运算结果分别是 4, 15, 7 和 8，可以看到 hash 值没有发生冲突。

ConcurrentHashMap 的 get 操作

Segment 的 get 操作实现非常简单和高效。先经过一次再哈希，然后使用这个哈希值通过哈希运算定位到 segment，再通过哈希算法定位到元素，代码如下：

```
public V get(Object key) {
    int hash = hash(key.hashCode());
    return segmentFor(hash).get(key, hash);
}
```

get 操作的高效之处在于整个 get 过程不需要加锁，除非读到的值是空的才会加锁重读，我们知道 HashTable 容器的 get 方法是需要加锁的，那么 ConcurrentHashMap 的 get 操作是如何做到不加锁的呢？原因是它的 get 方法里将要使用的共享变量都定义成 volatile，如用于统计当前 Segement 大小的 count 字段和用于存储值的 HashEntry 的 value。定义成 volatile 的变量，能够在线程之间保持可见性，能够被多线程同时读，并且保证不会读到过期的值，但是只能被单线程写（有一种情况可以被多线程写，就是写入的值不依赖于原值），在 get 操作里只需要读不需要写共享变量 count 和 value，所以可以不用加锁。之所以不会读到过期的值，是根据 java 内存模型的 happen before 原则，对

volatile 字段的写入操作先于读操作，即使两个线程同时修改和获取 volatile 变量，get 操作也能拿到最新的值，这是用 volatile 替换锁的经典应用场景。

```
transient volatile int count;  
  
volatile V value;
```

在定位元素的代码里我们可以发现定位 HashEntry 和定位 Segment 的哈希算法虽然一样，都与数组的长度减去一相与，但是相与的值不一样，定位 Segment 使用的是元素的 hashCode 通过再哈希后得到的值的高位，而定位 HashEntry 直接使用的是再哈希后的值。其目的是避免两次哈希后的值一样，导致元素虽然在 Segment 里散列开了，但是却没有在 HashEntry 里散列开。

```
hash >>> segmentShift) & segmentMask // 定位 Segment 所使用的 hash 算法  
  
int index = hash & (tab.length - 1); // 定位 HashEntry 所使用的 hash 算法
```

ConcurrentHashMap 的 Put 操作

由于 put 方法里需要对共享变量进行写入操作，所以为了线程安全，在操作共享变量时必须得加锁。Put 方法首先定位到 Segment，然后在 Segment 里进行插入操作。插入操作需要经历两个步骤，第一步判断是否需要对 Segment 里的 HashEntry 数组进行扩容，第二步定位添加元素的位置然后放在 HashEntry 数组里。

是否需要扩容。在插入元素前会先判断 Segment 里的 HashEntry 数组是否超过容量(threshold)，如果超过阀值，数组进行扩容。值得一提的是，Segment 的扩容判断比 HashMap 更恰当，因为 HashMap 是在插入元素后判断元素是否已经到达容量的，如果到达了就进行扩容，但是很有可能扩容之后没有新元素插入，这时 HashMap 就进行了一次无效的扩容。

如何扩容。扩容的时候首先会创建一个两倍于原容量的数组，然后将原数组里的元素进行再 hash 后插入到新的数组里。为了高效 ConcurrentHashMap 不会对整个容器进行扩容，而只对某个 segment 进行扩容。

ConcurrentHashMap 的 size 操作

如果我们要统计整个 ConcurrentHashMap 里元素的大小，就必须统计所有 Segment 里元素的大小后求和。Segment 里的全局变量 count 是一个 volatile 变量，那么在多线程场景下，我们是不是直接把所有 Segment 的 count 相加就可以得到整个 ConcurrentHashMap 大小了呢？不是的，虽然相加时可以获取每个 Segment 的 count 的最新值，但是拿到之后可能累加前使用的 count 发生了变化，那么统计结果就不准了。所以最安全的做法，是在统计 size 的时候把所有 Segment 的 put，remove 和 clean 方法全部锁住，但是这种做法显然非常低效。因为在累加 count 操作过程中，之前累加过的 count 变化了，所以 ConcurrentHashMap 的做法是先尝试 2 次通过不锁住 Segment 的方式来统计各个 Segment 大小，如果统计的过程中，容器的 count 发生了变化，则再采用加锁的方式来统计所有 Segment 的大小。

那么 ConcurrentHashMap 是如何判断在统计的时候容器是否发生了变化呢？使用 modCount 变量，在 put，remove 和 clean 方法里操作元素前都会将变量 modCount 进行加 1，那么在统计 size 前后比较 modCount 是否发生变化，从而得知容器的大小是否发生变化。

参考资料

1. JDK1.6 源代码。
2. 《Java 并发编程实践》。
3. [Java 并发编程之 ConcurrentHashMap](#)。

作者介绍

方腾飞，花名清英，淘宝资深开发工程师，关注并发编程，目前在广告技术部从事无线广告联盟的开发和设计工作。个人博客：<http://kiral.javaeye.com> 微博：<http://weibo.com/kirals> 欢迎通过我的微博进行技术交流。

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

相关内容：

- [聊聊并发（五）——原子操作的实现原理](#)
- [聊聊并发（三）——JAVA 线程池的分析和使用](#)

Calatrava：自由构建 UI 的跨平台移动框架

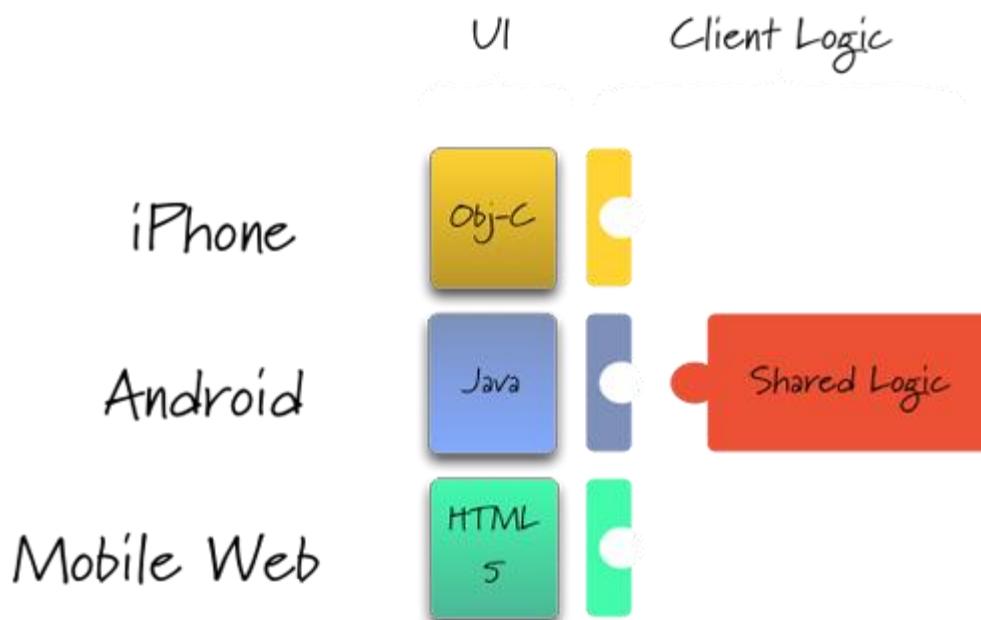
作者 [申健](#)

移动是未来计算的趋势，越来越多的人使用移动设备来访问互联网。但是目前至少三大平台：iOS、Android、移动 Web。相比桌面 Web，移动用户需要更好的体验、界面和设计。然而移动设备受限于电池、不可靠的网络连接和小尺寸屏幕。

几个月前，Martin Fowler 写过一篇关于开发跨设备移动应用的[文章](#)。他指出，要获得最佳用户体验，最直接的办法就是针对每个移动平台开发一个本地应用，但这却将带来极大的成本。或者选用跨平台工具箱，“一次编写随处运行”？然而没有几个产品真正获得成功。Web 应用程序倒是可以以较低成本运行在跨平台之上，但是用户界面又受到限制。因此，不可避免地要在用户体验和成本之间做出选择。

或者，折中的办法就是开发混合式(hybrid)应用程序：结合 Web 和原生应用。

ThoughtWorks 探索这条路并取得一些进展，最近发布了[Calatrava 开源框架](#)。Calatrava 的思路是开发者用跨平台的 JavaScript 编写客户端逻辑，这部分代码完全相同，运行在 iOS、Android、移动 Web 的 JavaScript 解释器中。Calatrava 提供本地桥接，允许逻辑来驱动原生 UI。



它将移动应用程序分成两部分来考虑：UI 和客户端逻辑主体(headless-body)。开发者使用 JavaScript 代码开发通用的控制器逻辑，而原生代码处理 UI 部分生成原生外观 (native veneer)，可移植的 JavaScript 代码和原生代码之间能够互操作，进而开发出混合移动应用。

Calatrava 本身并不提供任何 UI 框架或建立 UI 抽象层。在 iOS 上，就用 Objective-C 和 Cocoa Touch 框架来构建 UI；在 Android 上，就用 Java 和 Android 库；在移动 Web 上，就用 HTML5、CSS3 和你喜欢的 JavaScript 库。跨平台逻辑层与代表着应用表现层 Page 对象进行交互，该对象提供了从显示和交互机制中分离出的 API 接口。很多时候，对于移动应用来说，HTML5 UI 已经足够好，所以 Calatrava 也允许在 iOS 和 Android 上桥接 HTML5，但是应用绝不会绑定在 HTML UI 上。

这样，每个平台都有具有了自己独有的 UI 设计，避免了“恐怖谷效应”(uncanny valley effect)。应用开发可以从 HTML UI 开始，当你觉得某些地方不够好的时候，就用原生 UI 替换掉，且只对部分平台进行替换，而不影响其他部分。同时你也可以享受某些平台特有的 UI 特性。

Calatrava 适合包含较多复杂的客户端逻辑的应用，且当应用偏重于成为现有产品的新渠道(Channel)，而非产品本身。如果应用的大部分代码是 UI 部分（如游戏），或是已经提供非常好的用户体验的 Web 应用，Calatrava 就不太适合了。Calatrava 编写的核心 JavaScript 逻辑支持使用 [Jasmine](#) 进行单元测试，以及使用 [cucumber.js](#) 进行功能测试。

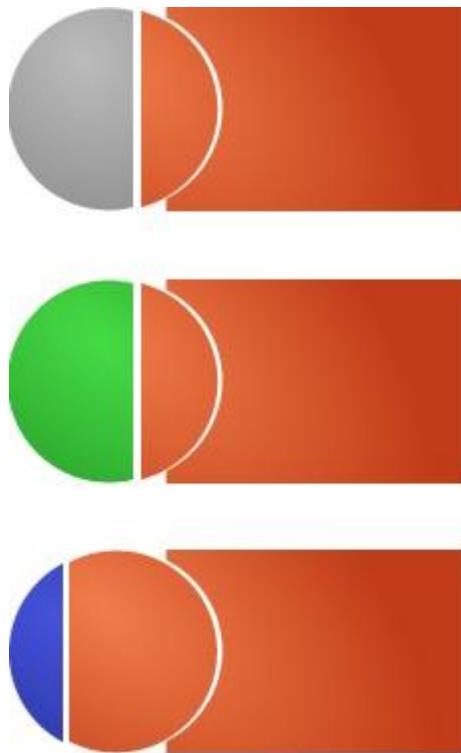
想试试吗？从 github 的 [Calatrava 主页](#)可以找到说明，简单来说就是下面几步：

1. 安装依赖：Node.js, Ruby, Xcode, Android.
2. 安装 Ruby 插件 gem install calatrava
3. 使用 Calatrava 工具创建项目 calatrava create sample
4. 编译并运行项目

Calatrava 还处在早期开发阶段，估计还是会有很多 bug，另外插件、模块和文档也需要完善。

Martin Fowler 分析了用户体验和可负担能力之间的动态平衡，他认为混合式 (hybrid) 解决方案介于纯本地应用和纯 Web 应用之间，更适合增量式发布。

即首次发布(release)采用纯 Web UI ,之后的发布(release)逐步将 Web UI 转为本地 UI 特性 ,或逐步地增加本地 UI 特性的比例。



Martin Fowler 认为 Caltrava 最有价值的地方就在于它适合增量式发布策略。比如 cover-your-bases 策略 ,即当你已经有大量用户基础 ,而移动应用定位为现有产品的新渠道(channel)。由于现有的用户 ,最重要的事情是将新渠道尽快地推到尽可能多的用户面前。很明显 ,平台覆盖率最重要。然而在移动平台上 ,体验也非常重要 ,所以应该用最小功能集提供简化的体验 ,而不是提供退化的体验。

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

相关内容 :

- [PhoneGap 2.0 发布](#)
- [ThoughtWorks 王秋 : 渐进增强在移动开发中应用](#)
- [傲游勾三股四解析 HTML5 中的 Web Storage 使用](#)

QA 部门将会消亡

作者 [Eli Lopian](#) 译者 [王瑜珩](#)

革命始于 250 年前，在工厂中、农田里、矿井上，机器开始代替人类进行生产劳动。这在极大的促进了经济增长的同时，也深深的伤害了那些技能一般、无法找到新工作或者没有足够知识去转行的人，这与目前 QA 所处的境地有着惊人的相似。上世纪九十年代，由于互联网泡沫的出现，对软件开发的需求急速膨胀，这就需要大量 QA 来进行测试，以保证软件能够顺利发布。因此，像 Mercury Interactive 这样的测试公司便如雨后春笋般出现。然而，当互联网泡沫破裂时，公司纷纷消减预算，敏捷开发变得更加普及，自动化测试开始接手（QA 的手工测试）。就像工业革命时手工劳动者被机器所淘汰一样，以手工测试为生的 QA 也面临同样的困境。很多 QA 人员从质量保证转移到了需要编程技能的岗位上去，独立的 QA 团队消失，QA 融合到开发团队中产生了跨职能团队。柏林墙倒了。

让我们回过头来，看看以前我们是怎么开发软件的。自五十年代开始，瀑布模型被大多数软件开发团队所采用。开发人员首先对整个系统进行预先设计，然后集中编写代码，再把写好的软件交给 QA 进行测试，最后修复 QA 找到的缺陷。当进度压力总是伴随着开发人员，使得开发人员永远无法按时交付时，开发人员对 QA 的依赖也越来越强。恶性循环由此产生，雇佣的 QA 越多，开发人员就越多的依赖 QA，同时也越少对他们自己的代码进行测试，由此又需要雇佣更多的 QA……最后开发人员干脆就不测试自己的代码了。

这种方式对于开发人员和测试人员都是低效的，软件交付被拖延，最终产品无法及时交付到客户手中。

在互联网泡沫破裂的同时，2001 年 2 月敏捷宣言发布了，一种新的开发思想浮出水面。敏捷开发方法将开发人员带入了崭新的世界，适应变化和快速交付可工作的软件成为开发团队的关注点。敏捷还使得整个团队都参与到开发的各项工作中，特别是对开发人员测试的重视胜过 QA 的手工测试。随着敏捷更加普及，效率持续提高，对 QA 的需求变得更少，QA 已经有一只脚踏出了（软件行业）这扇们。

QA 越多，问题就越多

企业级软件开发是复杂并且昂贵的，管理层经常会发现无法达成计划的目标，从而需要决定如何应对这种情况。他们有三个选择来解决按时交付的问题。

1. 增加预算 —— 你不是每次都能给项目增加预算的，但如果可以，这或许可以帮助你按时完成项目，同时也需要考虑效果递减法则。
2. 减少特性 —— 开发人员和管理层一般都不倾向于让客户花同样的钱却买到更少的东西，对很多公司来说这不是一个可行的办法。
3. 降低质量 —— 虽然我们无法摆脱软件缺陷，但软件质量可能对任何产品来说都是最重要的部分，不幸的是，降低质量通常是人们最先选择的方案。

当开发人员面临压力（或者仅仅因为懒惰），而写出质量不高的代码时，管理层却在减少 QA 的数量。这就是问题的根本所在，也是敏捷将要解决的问题。

敏捷就是答案

随着敏捷方法的流行，开发人员和管理者似乎找到了构建高质量软件的钥匙。虽然两方之间仍有很多不同的观点，但至少大家都希望能够在最短的时间内将软件构建出来，当然管理者还希望尽可能少花钱。

互联网泡沫破裂后，随着经济形势的回落，企业也意识到他们需要降低软件开发的成本，但他们该如何减少 QA 部门的高昂花销呢？

敏捷软件开发让开发人员自己测试自己的代码，单元测试通过就表示单元级别的代码能够正确完成预期的功能，当然这还需要整个团队一起努力。产品经理负责保证产品能够满足客户的需要，开发人员和测试人员一起编写测试用例，然后开发人员编写单元测试来保证质量。构造良好的测试是保证软件交付的唯一方法，也是敏捷软件开发的核心，它可以保证代码按照开发人员的意图工作。在开发人员承担测试自己代码的职责后，QA 仍有存在的价值，他们需要寻找那些非常难以发现的问题。敏捷软件开发的一个支柱就是可以工作的软件，有些敏捷方法包含了测试驱动开发和开发人员执行的单元测试，而单元测试被用来检查你写的代码。通过这种保证局部的方式来让整体获益，以便得到持续的、及时的反馈，是一种快速、高效的抵御缺陷的方式。如果没有足够的单元测试覆盖，你很难敏捷起来，因为设计是在持续变化的，而变化会引入缺陷，如果无法在开发时发现这些缺陷，项目就会陷入困境。

单元测试 - 潜在的 QA 杀手

单元测试可以用来测试一小段代码，以保证代码做了正确的事，并能够被集成到整个软件系统中去。单元测试已经被证明可以达到 90%以上的代码覆盖率，和 QA 使用的手工测试工具相比，构建良好、自动执行的单元测试可以随着代码一起演进，实时对代码进行测试。

我并不是说单元测试可以解决所有的开发问题，但作为测试代码（和促进设计）的工具，单元测试可以用更低的成本快速提升软件质量。自动化单元测试和测试驱动开发也是构建高质量软件所需要的，它们可以让开发人员更快速的适应新需求和其他的变化。

QA 手工测试可能是九十年代互联网泡沫时期的救世主，但公司纷纷意识到 QA 团队阻碍了对变化的适应。单独的 QA 团队只会拖延开发人员修复错误的时间，敏捷开发和单元测试则保证了开发人员写完代码时，软件就可以工作了。

我们将去往何处？

我打赌你一定很奇怪为什么我会选择这样一个标题，这可能与我是 Don McLean 的粉丝有关。此外，我觉得对于一篇关于 QA 是如何从软件开发的主力走向边缘的文章，这会是一个很贴切的标题，QA 即将消亡。可现实是，QA 部门依然存在于很多公司之中，这种情况还将持续多久？我个人认为这只是时间问题，测试的任务迟早会转移到开发人员身上，开发人员需要测试自己的代码，而不依赖于任何 QA 部门。

当然，还会有很多专业 QA 继续呆在这个行业之中，不同的是他们不再是独立的部门，而是成为开发团队的一分子。敏捷需要跨职能角色之间的沟通，推倒降低效率的部门隔阂。传统的 QA 部门只会降低开发速度，拉高开发成本，只有一种方法可以解决：开发人员测试。这不意味着不再需要 QA，而是需要 QA 重新定义自己的角色，与时俱进。不能再一成不变了，他们需要融入开发团队，在自动化单元测试中贡献价值；或者加入到产品管理中去，致力于定义让客户满意的产 品。

当我们将(测试)职责转移到开发人员身上时 ,开发人员将会写出更干净的代码 ,构建出缺陷更少、质量更高的软件。这一切都取决于公司如何组织 ,以便在不降低质量的情况下降低成本。

关于作者



Eli Lopian是单元测试工具公司Typemock的创始人和首席执行官。在创建 Typemock 之前 ,Eli 曾在 AMDOCS (NYSE:DOX)和 DEC 等国际公司工作 ,拥有 17 年的研发经验 ,并曾负责优化开发流程和改善开发环境与工具。

原文链接 :<http://www.infoq.com/cn/articles/day-qa-dept-died>

相关内容 :

- [3000 万代码行敏捷项目的每日发布](#)
- [调查表明敏捷模式的最大挑战在于沟通问题与持续改进](#)
- [采访和书评 : Google 如何做测试](#)
- [如何控制单元测试的粒度 ?](#)
- [虚拟座谈会 : 专家眼中的 QA、敏捷测试、探索式测试及测试的开放性](#)
- [在敏捷项目中实施自动化测试之我见](#)

消灭神出鬼没的 Heisenbug

作者 [Victor Grazi](#) 译者 [王丽娟](#)

Webster 最常用的词汇列表上可能并没有术语 “Heisenbug” 。但不幸的是，作为软件工程师，我们对这个可恶的家伙是再熟悉不过的了。量子物理学里有个海森堡不确定性原理（Heisenberg Uncertainty Principle），认为观测者观测粒子的行为会影响观测结果，术语 “Heisenbug” 是海森堡不确定性原理的双关语，指生产环境下不经意出现、费尽九牛二虎之力却无法重现的计算机 Bug。所以要同时重现基本情形和 Bug 本身几乎是不可能的。

但要是照着下面的方法去做，保证会出现 Heisenbug：

1. 编写并发程序，但一定要忽略并发概念，比如发布、逸出、Java 内存模型和字节码重排序。
2. 全面测试程序。（不要担心，所有的测试都会通过！）
3. 把程序发布到生产环境。
4. 等待生产环境瘫痪，并检查你的收件箱。你马上就会发现有大量尖酸刻薄的电子邮件在痛批你和你满是 Bug 的应用。

在想方设法规避这些 Heisenbug 之前，思考一个更根本的问题可能会更合适：既然并发编程如此困难，为什么还要这么做呢？事实上，进行并发编程的原因有很多：

并行计算——随着处理器和内核的增加，多线程允许程序进行并行计算，以确保某个内核不会负担过重、而其他内核却空闲着。即使在一台单核机器上，计算不多的应用也可能会有较多的 I/O 操作，或是要等待其他的一些子系统，从而出现空闲的处理器周期。并发能让程序利用这些空闲的周期来优化性能。

公平——如果访问一个子系统的客户端有两个甚至更多个，一个客户端必须等前面的客户端完成才能执行是很不可取的。并发可以让程序给每个客户端请求分配一个线程，从而缩短收到响应的延迟。

方便——编写一系列独立运行的小任务，往往比创建、协调一个大型程序去处理所有的任务要容易。

但这些原因并不能改变并发编程很难这一事实。如果程序员没有彻底考虑清楚应用里的并发编程，就会制造出严重的 Heisenbug。在这篇文章里，我们将介绍用 Java 语言架构或开发并发应用时需要记住的十个建议。

建议 1—自我学习

我建议精读由 Brian Goetz 编著的《Java 并发编程实践》一书。这部自 2006 年以来就畅销的经典著作从最基本的内容开始讲解 Java 并发。我第一次读这本书的时候有醍醐灌顶的感觉，并意识到了过去几年犯的所有错误，后来发现我会不厌其烦地提起这本书。要想彻底深入了解 Java 并发的方方面面，可以考虑参加[并发专家培训](#)，这门课程以《Java 并发编程实践》为基础，由 Java 专家 Heinz Kabutz 博士创建，并得到了 Brian Goetz 的认可。

建议 2—利用现有的专业资源

使用 Java 5 引入的 `java.util.concurrent` 包。如果你没怎么用过这个包里的各个组件，建议你从 SourceForge 上下载能执行的 Java Concurrent Animated 应用（运行 `java -jar` 命令），这个应用包含一系列动画（事实上是由你定制的），演示了 concurrent 包里的每个组件。下载的应用有个交互式的目录，你在架构自己的并发解决方案时可以参考。

Java 在 1995 年底问世时，成为最早将多线程作为核心语言特性的编程语言之一。并发非常难，我们当时发现一些很优秀的程序员都写不好并发程序。不久之后，来自奥斯威戈州立大学的并发大师 Doug Lea 教授出版了他的巨著《Java 并发编程》。这本书的第二版引入了并发设计模式的章节，`java.util.concurrent` 包后来就是以这些内容为基础的。我们以前可能会把并发代码混在类里面，Doug Lea 让我们把并发代码提取成能由并发专家检验质量的单独组件，这能让我们专注于程序逻辑，而不会过度受困于变化莫测的并发。等我们熟悉这个包、并在程序中使用的时候，引入并发错误的风险就能大大降低了。

建议 3—注意陷阱

并发并不是个高级的语言特性，所以不要觉得你的程序不会有线程安全方面的问题。所有的 Java 程序都是多线程的：JVM 会创建垃圾回收器、终结处理器

(Finalizer)、关闭钩子等线程，除此以外，其他框架也会引入它们自己的线程。比如 Swing 和 Abstract Window Toolkit(AWT)会引入事件派发线程(Event Dispatch Thread)。远程方法调用(RMI)的 Java 应用编程接口，还有 Struts、Spring MVC 等所谓的模型-视图-控制器 (MVC) Web 框架都会为每个调用分配一个线程，而且都有明显的不足之处。所有这些线程都能调用你代码里的编程钩子，这可能会修改你应用的状态。如果允许多个线程在读或写操作中访问程序的状态变量，却没有正确的同步机制，那你的程序就是不正确的。要意识到这一点，并在并发编码时积极应对。

建议 4—采用简单直接的方法

首先保证代码正确，然后再去追求性能。封装等技术能确保你的程序是线程安全的，但也会减慢运行速度。不过 HotSpot 做了很好的优化工作，所以比较好的做法是先保证程序运行正确，然后再考虑性能调整。我们发现，比较聪明的优化方式往往会使代码比较费解、不易维护，一般也不会节省时间。要避免这样的优化，无论它们看上去多么优雅或聪明。一般来说，最好先写不经优化的代码，然后用剖析工具找出问题点和瓶颈，再尝试着去纠正这些内容。先解决最大的瓶颈，然后再次剖析；你会发现改正一个问题点后，接下来最严重的一些问题也会自动修复。一开始就正确地编写程序，要比后面再针对安全性改造代码容易一个数量级，所以要让程序保持简单、正确，并从一开始就认真记录所有为并发采取的措施。在代码使用 @ThreadSafe、@NotThreadSafe、@Immutable、@GuardedBy 等并发注释是一种很好的记录方式。

建议 5—保持原子性

在并发编程里，如果一个操作或一组操作在其调用和返回响应之间，对系统的其他部分来说就像在一瞬间发生的，那这个操作或这组操作就是原子的。原子性是并发处理互相隔离的保证。Java 能确保 32 位的操作是原子的，所以给 integer 和 float 类型的变量赋值始终都是完全安全的。当多个线程同时去设置一个值的时候，只要能保证一次只有一个线程修改了值，而不是好几个线程都修改了这个值，那就自然而然地做到了安全性。但涉及 long 和 double 变量的 64 位操作就没有这样的保证了：Java 语言规范允许把一个 64 位的值当成两个 32 位的值，用非原子的方式去赋值。结果在使用 long 和 double 类型的变量时，多个线程

要是试图同时去修改这个变量，就可能产生意想不到和不可预知的结果，变量最终的值可能既不是第一个线程修改的结果，也不是第二个线程修改的结果，而是两个线程设置的字节的组合。举例来说，如果线程 A 将值设置成十六进制的 1111AAAA，与此同时，线程 B 将值设置为 2222BBBB，最后的结果很可能是 1111BBBB，两个线程可都没这么设置。把这样的变量声明成 volatile 类型可以很容易地修复这个问题，volatile 关键字会告诉运行时把 setter 方法作为原子操作执行。但 volatile 也不是万能的；虽然这能保证 setter 方法是原子的，但要想保证变量的其他操作也是原子的，还需要进一步的同步处理。举例来说，如果“count”是个 long 类型的变量，调用递加功能 count++ 完全有可能出错。这是因为++看起来是个单一操作，但事实上是“查询”、“增加”、“设置”三个操作的集合。如果并发线程不稳定地交叉执行++操作，两个线程就可能同时执行查询操作，获得相同的值，然后算出相同的结果，导致设置操作互相冲突、生成同样的值。如果“count”是个计数器，那某次计数就会丢失。当多个线程同时执行检查、更新和设置操作时，要确保在一个同步块里执行它们，或者是使用 java.util.concurrent.locks.Lock 的实现，比如 ReentrantLock。

很多程序员都认为只有写值需要同步、读取值则不用同步。这是一种错误的认知。举例来说，如果同步了写值操作，而读取值没进行同步，读线程极有可能看不到写线程写入的值。这看起来似乎是个 Bug，但它实际上是 Java 的一个重要特性。线程往往在不同的 CPU 或内核中执行，而在处理器的设计里，数据从一个内核移到另一个内核是比较慢的。Java 意识到了这一点，因而允许每个线程在启动时对状态进行拷贝；随后，如果状态在其他线程里的变化不合法，那仍然可以访问原始状态。虽然把变量声明成 volatile 能保证变量的可见性，但仍然不能保证原子性。你可以在必要的地方对代码进行正确地同步，你也有义务这么做。

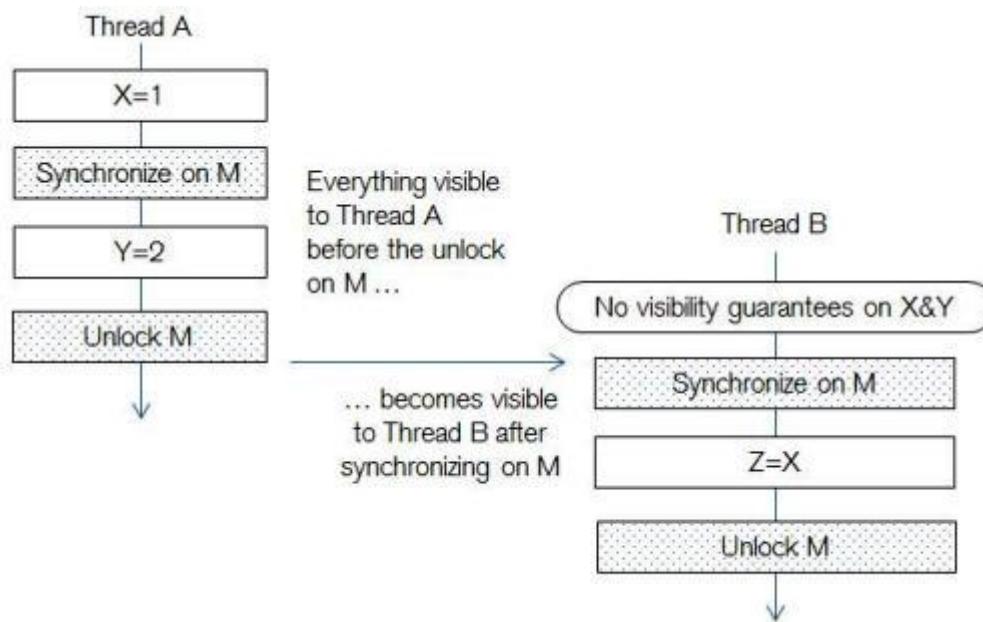
建议 6—限制线程

要防止多个线程互相竞争去访问共享数据，一种方式就是不要共享！如果特定的数据点只由单个线程去访问，那就没有必要考虑额外的同步问题。这种技术称为“线程限制（Thread Confinement）”。

限制线程的一种方式是让对象不可变。尽管不可变的对象会生成大量的对象实例，但从维护的角度来说，不可变对象确实是专家要求的。

建议 7—注意 Java 内存模型

了解 Java 内存模型对变量可见性的约束。所谓的“程序次序法则”是指，在一个线程里设置的变量不需要任何同步，对这个线程之后的所有内容来说都是可见的。如果线程 A 调用 synchronized(M) 的时候获得了锁 M，锁 M 随后会被线程 A 释放、被线程 B 获取，那线程 A 释放锁之前的所有动作（包括获取锁之前设置的变量，也许是意料之外的），在线程 B 获得锁 M 之后，也对线程 B 可见。对线程可见的所有值来说，锁的释放就意味着内存提交。（见下图）。请注意，java.util.concurrent 里新的加锁组件 Semaphore 和 ReentrantLock 也表示相同的意思。volatile 类型的变量也有类似的含义：线程 A 给 volatile 变量 X 设置值，这类似于退出同步块；线程 B 读取 volatile 变量 X 的值，就类似于进入同一变量的同步块，这意味着在线程 A 给 X 赋值的时候，对线程 A 可见的那些内容在线程 B 读取 X 的值之后，也会对线程 B 可见。



建议 8—线程数

根据应用情况确定合适的线程数。我们使用下面的变量来推导计算公式：

假设 T 是我们要推导出的理想线程数；

C 是 CPU 个数；

X 是每个进程的利用率（%）；

U 是目标利用率（%）。

如果我们只有一个被百分之百利用的 CPU，那我们只需要一个线程。当利用率是 100% 时，线程数应该等于 CPU 的个数，可以用公式表示为 $T=C$ 。但如果每个 CPU 只被利用了 $x(\%)$ ，那我们就可以用 CPU 的个数除以 x 来增加线程数，结果就是 $T=C/x$ 。例如，如果 x 是 0.5，那线程数就可以是 CPU 个数的两倍，所有的线程将带来 100% 的利用率。如果我们的目标利用率只有 $U(\%)$ ，那我们必须乘以 U ，所以公式就变为 $T=UC/x$ 。最后，如果一个线程的 $p\%$ 是受处理器限制的（也就是在计算）， $n\%$ 是不受处理器限制的（也就是在等待），那很显然，利用率 $x=p/(n+p)$ 。注意 $n\%+p\%=100\%$ 。把这些变量代入上面的公式，可以得出如下结果：

$$T=CU(n+p)/n \text{ 或}$$

$$T=CU(1+p/n).$$

要确定 p 和 n ，你可以让线程在每次 CPU 调用和非 CPU 调用（比如 I/O 调用和 JDBC 调用）的前后输出时间日志，然后分析每次调用的相对时间。当然，并不是所有的线程都会显示出相同的指标，所以你必须平均一下；对调整配置来说，求平均值应该是个不错的经验。得到大致正确的结果是很重要的，因为引入过多的线程事实上反而会降低性能。还有一点也不容小觑，就是让线程数保持可配置，这样在你调整硬件配置的时候，可以很容易地调整线程数。只要你计算出线程数，就可以把值传给 ThreadPoolExecutor。由于进程间的上下文切换是由操作系统负责的，所以把 U 设置得低一点，以便给其他进程留些资源。不过公式里的其他计算就不用考虑其他进程了，你只考虑自己的进程就可以了。

好消息是会有一些偏差，如果你的线程数有 20% 左右的误差，那可能不会有较大的性能影响。

建议 9—在 server 模式下开发

在 server 模式下进行开发，即便你开发的只是个客户端应用。server 模式意味着运行时环境会进行一些字节码的重排序，以实现性能优化。相对 client 模式来说，server 模式会更频繁地进行编译器优化，不过在 client 模式下也会发现这些优化点。在测试过程中使用 server 模式能尽早进行优化处理，就不用等到发布到生产环境里再进行优化了。

但要分别使用-server、-client 和-Xcomp 参数进行测试（提前进行最大优化，以避免运行时分析）。

要设置 server 模式，在调用 java 命令时使用-server 选项。在 Java Specialists 享誉盛名的 Heinz Kabutz 博士指出，一些 64 位模式的 JVM（比如 Apple OSX 最近才推出的 JVM）会忽略-server 选项，所以你可能还要使用-showversion 选项，-showversion 选项会在程序日志的开始显示版本，并指明是 client 模式还是 server 模式。如果你正在用 Apple OSX，你可以用-d32 选项切换到 32 位模式。简言之，使用-showversion -d32 -server 进行测试，并检查日志，以确保使用的 Java 版本正是你期望的那个。Heinz 博士还建议，用-Xcomp 和 -Xmixed 选项各进行一次独立的测试（-Xcomp 不会进行 JIT 运行时优化；-Xmixed 是缺省值，会对 hot spots 进行优化）。

建议 10—测试并发代码

我们都知道怎么创建单元测试程序，对代码里方法调用的后置条件进行测试。在过去这些年里，我们费了很大的劲儿才习惯执行单元测试，因为我们完成开发、开始维护的时候，才明白单元测试节省的时间所带来的价值。不过测试程序时，最重要的方面还是并发。因为并发是最脆弱的代码，也是我们能找到大部分 Bug 的地方。但具有讽刺意味的是，我们往往是在并发方面最疏忽大意，主要是因为并发测试程序很难编写。这要归因于线程交叉执行的不确定性——哪些线程会按什么样的顺序执行，通常是预测不出来的，因为在不同的机器上、甚至在同一机器的不同进程里，线程的交叉执行都会有所不同。

有一种测试并发的方法是用并发组件本身去进行测试。比如说，为了模拟任意的线程时间安排，要考虑线程次序的各种可能性，还要用定时执行器确切地按照这些可能性给线程排序。测试并发的时候，线程抛出异常并不会导致 JUnit 失败；JUnit 只在主线程遇到异常时才会失败，而其他线程遇到异常的时候则不会。解决这个问题的方法之一是把并发任务指定为 FutureTask，而不是去实现 Runnable 接口。因为 FutureTask 实现了 Runnable 接口，可以把它提交给定时执行的 Executor。FutureTask 有两个构造函数，其中一个接受 Callable 类型的参数，一个接受 Runnable 类型的参数。Callable 和 Runnable 类似，不过还是有两个显著的不同之处：Callable 的 call 方法有返回值，而 Runnable 的 run 方法返回 void；Callable 的 call 方法会抛出可检查型异常

ExecutionException , Runnable 的 run 方法只会抛出不可检查型异常。

ExecutionException 有个 getCause 方法 ,这个方法会返回触发它的实际异常。

(请注意 ,如果你传入的参数是 Runnable 类型 ,你还必须传入一个返回对象。

FutureTask 内部会将 Runnable 和返回对象包装成 Callable 类型的对象 ,所以你仍然可以利用 Callable 的优势。) 你可以用 FutureTask 安排并发代码 ,让 JUnit 的主线程调用 FutureTask 的 get 方法获取结果。 Callable 或 Runnable 对象异步抛出的所有异常 ,get 方法都会重新抛出来。

但也有一些挑战。假设你期望一个方法是阻塞的 ,你想测试它是不是真的和预期一样。那你怎么测试阻塞呢 ?你要等多久才能确定这个方法确实阻塞了呢 ?

FindBugs 会定位出来一些并发问题 ,你也可以考虑使用并发测试框架。 Bill Pugh 是 FindBugs 的作者之一 ,他参与创建的 MultithreadedTC 提供了一套 API ,可以指定、测试所有的交叉执行情况 ,以及其他并发功能。

在测试的时候要牢记一点——Heisenbug 并不会在每次运行时都出现 ,所以结果并不是简单的 “通过” 或 “不通过” 。多次循环(数千遍)执行并发测试程序 ,根据平均值和标准差得出统计指标 ,再去衡量是否成功了。

总结

Brian Goetz 告诫过大家 ,可见性错误会随着时间的推移越来越普遍 ,因为芯片设计者设计的内存模型在一致性方面越来越弱 ,不断增多的内核数量也会导致越来越多的交叉存取。

所以千万不要想当然地处理并发。要了解 Java 内存模型的内部工作机制 ,并尽量使用 java.util.concurrent 包 ,以便消灭并发程序里的 Heisenbug ,然后就等着表达满意和好评的电子邮件纷至沓来吧。

链接

- [《Java 并发编程实践》](#)
- [Java Specialists 新闻中心](#)
- [并发专家课程](#)
- [Java Concurrent Animated 应用](#)

作者简介



Victor Grazi 今年五月成为一名 Oracle Java Champion , 他从 2005 年起就职于瑞士瑞信银行的投资银行架构部 处理平台架构的相关事宜 , 他还是一名技术咨询师和 Java 技术布道者。此外 , 他经常在技术会议上演讲 , 给大家讲解 Java 并发和其他 Java 相关的主题。Victor 是 “并发专家课程” 的贡献者和教练 , 他还在 SourceForge 上提供了开源项目 “Java Concurrent Animated” 。他和他的家人住在纽约布鲁克林区。

原文链接 : <http://www.infoq.com/cn/articles/exterminating-heisenbugs>

相关内容 :

- [快讯 :今日 9:45 分将直播 SpringOne 大会 “Spring 框架的未来” 等主题演讲](#)
- [Lambda 项目公开邮件列表](#)
- [Jigsaw 项目延后的群众反响](#)

arrayDB , 全新而且简单的 PHP ORM 库

作者 [Mustafa Dokumaci](#) 译者 [雷慈祥](#)

过许多 PHP ORM 库。它们中大多数都要你为想保存在数据库里的每一项写一个类。无缘无故地继承这个、继承那个，而那些内容通常都是重复和反复无常的。

既然条目包含相似数据类型以及相似关系的字段，一个编写良好的类就可适用于所有情况。如果你需要一个库来简化这些事情，这就是我的方法。

arrayDB ORM 库只包含 5 个类。你基本上只使用其中一个，而其余类则在内部使用，仅此而已。缓存及其与数据库之间的同步全是自动的，你无需对其进行跟踪。

要开始使用这个库，你只需简单地定义：

- 你的数据模型（你需要保存哪几项，他们的字段以及字段间的关系）。
- 你的 MySQL 连接方式。
- 你的缓存配置。

定义数据模型

所有的数据模型定义被写成像这样的数组：

```
$model=array(  
  
    'user'=>array(  
  
        'conf'=>array('len'=>7),  
  
        'fields'=>array(  
  
            'name'=>array('len'=>50)  
        ),  
  
        'has_many'=>array(  
  
            'posts'=>array('type'=>'post',  
  
                'foreign_name'=>'writer'  
            ),  
    ),
```

```

'many_to_many'=>array(
    'liked_posts'=>array('type'=>'post',
'foreign_name'=>'likers'),
),
'self_ref'=>array('friends')
),

'post'=>array(
    'conf'=>array('len'=>10),
    'fields'=>array(
        'text'=>array('len'=>200),
        'view_count'=>array('type'=>'numeric', 'len'=>5)
        // 默认的字段类型是 text , 这里将类型定义成 numeric
    )
)
);

```

这里我们有两张表：用户和帖子。

用户有姓名，有些用户发帖子而有些用户关注帖子。帖子有文本内容，浏览量以及关注者。

用户还有其他许多用户作为好友。

使用这个模型，我们想通过\$user['posts']获取用户发布的所有帖子，通过\$post['writer']获取帖子的作者。这是一对多关系。

我们还想通过\$user['liked_posts']获取用户关注过的帖子，通过\$post['likers']获取帖子的关注者。这是多对多关系。

最后我们想通过\$user['friends']获取用户的好友列表。这是自引用关系。

定义 MySQL 连接方式

定义 MySQL 连接方式也写成一个像这样的数组：

```
$db_config=array(  
    'hostname'=>'localhost', 'database'=>'social',  
    'username'=>'root', 'password'=>  
)
```

定义缓存配置

目前，我们有三种缓存类型实现：APC、Memcached 以及普通文本文件。要使用 APC，这样的配置数组就够了：

```
$cache_config=array('type'=>'apc');
```

要使用 Memcached，你需要提供一些参数：

```
$cache_config=array('type'=>'memcached', 'host'=>'127.0.0.1',  
'port'=>11211, 'timeout'=>1);
```

要使用普通文本文件，你需要创建一个可读可写的目录并提供其绝对路径：

```
$cache_config=array('type'=>'file', 'path'=>'/tmp/my_project_cache')
```

还有一个可选参数 “prefix” 。如果给定，它将用作缓存的键名。

开始使用

现在是时候使用那些我们定义过的数据了。初始化该库，我们只需要这几行代码。

```
DB::init($db_config);  
  
CACHE::init($cache_config);  
  
ADB::init($model);
```

```
$adb=ADB::get_instance();
```

创建表

这项任务只需执行一次。我们告诉库去创建所需的数据库表。它就会负责完成关系等相关的复杂工作。

```
$adb->create_tables();
```

我们只需在发布的时候执行该方法一次。如果在插入数据之后，调用该方法将导致数据丢失。

使用记录

我们手头拥有这个\$adb 实例。我们将通过它来获取所有数据。

创建记录

我们提供表名和由字段名称及数据组成的键值对数组来创建一条记录。

```
$uid1=$adb->create('user', array('name'=>'John'));  
  
$uid2=$adb->create('user', array('name'=>'Marry'));  
  
  
$pid1=$adb->create('post', array(  
    'writer'=>$uid1,  
    'text'=>'What a wonderful world'  
));  
  
  
$pid2=$adb->create('post', array(  
    'writer'=>$uid2,  
    'text'=>'Life is beautiful'  
));
```

创建多对多关系

第一个参数是表名。第二个参数是被关联记录的局部名称。第三个参数是记录的 ID，第三个参数是被关联记录的 ID。

```
$adb->relate('user', 'friends', $uid1, $uid2);

$adb->relate('user', 'liked_posts', $uid1, $pid1); // 关注自己的帖子 :)

$adb->relate('user', 'liked_posts', $uid1, $pid2);

$adb->relate('user', 'liked_posts', $uid2, $pid1);
```

列举数据

我们可在简单的循环中列举所有写过帖子的用户以及帖子的关注者：

```
foreach ($adb->id_list('user') as $uid) {

    // load user
    $user=$adb->load('user', $uid);
    echo '<h1>' . $user['name'] . '</h1>' . "\n";

    echo '<h2>Posts: </h2>' . "\n";
    echo '<ul>' . "\n";

    foreach ($user['posts'] as $pid) {

        //load post of user
        $post=$adb->load('post', $pid);
        $likers=array();
```

```

foreach ($post['likers'] as $lid) {
    // load liker of post
    $liker=$adb->load('user', $lid);
    $likers[]=$liker['name'];
}

$likers=(count($likers)) ? '<br />' . implode(', ',
$likers) . ' liked.' : '';

echo '<li>' . $post['text'] . '' . $likers . '</li>' .
"\n";
}

echo '</ul>' . "\n";
}

```

更新记录

我们能够像下面这样更新任意一个记录：

```

$user1=$adb->load('user', $uid1);
$user1['name']='Jack';
// 无需调用任何保存方法，保存以及缓存的同步更新全是自动的。

```

如果我们需要一次更新多个字段，这是另一种方法：

```

$post1=$adb->load('post', $pid1);
$post1->update(array('writer'=>$uid2, 'text'=>'Not a wonderful
world!'));

```

删除关联

这和创建关联一样：

```
$adb->unrelate('user', 'friends', $uid1, $uid2);  
  
$adb->unrelate('user', 'liked_posts', $uid1, $pid1);
```

删除记录

我们可以删除记录，同时保存或删除与该记录相关联的数据。

```
$adb->delete('user', $uid1);  
  
// 用户删掉了，帖子成为匿名的了。  
  
  
$adb->delete('user', $uid1, true);  
  
// 用户以及用户的帖子都删掉了。
```

查询更多

作为示例，我想获得最受关注的 5 篇帖子。如下是我们所有要做的：

```
foreach ($adb->id_list('post', false, 'likers DESC', 5) as $pid) {  
  
    $post=$adb->load('post', $pid);  
  
    // 可对帖子做任何操作
```

抑或是我们想要获取用户被关注次数超过 5 次的所有帖子，我们可以执行以下操作：

```
$user=$adb->load('user', $uid1);  
  
foreach ($user->id_list('post', 'view_count>5') as $pid) {  
  
    $post=$adb->load('post', $pid);  
  
    // 可对帖子做任何操作
```

```
}
```

主要目标

对于单独的帖子页面，我们的代码将会是这么简单：

```
$post=$adb->load('post', $pid1);

$writer=$adb->load('user', $post['writer']);

echo $writer['name'] . ' wrote' . "<br />\n";

echo $post['text'] . "<br />\n";

$post['view_count']++;

// 是的，增大浏览量就是这么简单
```

这些代码里有任何查询或者缓存逻辑吗？没有，主要目标是保持简单。

市面上有很多著名的替代产品。他们文档完备，支持得也更好。该库不是他们中的一员，它目前还不是一部状态良好的机器。在我看来，这是最简单且容易上手的方法。如果 ORM 库的目的是为了让编码人员不用关心数据库逻辑，这个库是其他新兴库中最自信的一个。jQuery 是最简单的 javascript 框架，也因此成了标准。所以，易于使用的 PHP ORM 库也有这样的机会。



这个库的下载地址：[arrayDB github](#)

欢迎提出意见和建议。

关于作者

Mustafa Dokumaci 是一位来自土耳其伊斯坦布尔的软件工程师。他的专业是环境工程和会计学，但他目前就职于 [sporcum.com](#) 和 [hipsin.com](#)。Mustafa 拥有六年的 PHP、MySQL、Apache、Nginx、Python、CodeIgniter、Magento 等的使用经验。

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

相关内容：

新品推荐

RabbitMQ 3.0 版本有所简化 ,改进了对 STOMP 和 MQTT 的支持

作者 [Roopesh Shenoy](#) 译者 [张卫滨](#)



RabbitMQ 最近释放了 3.0 版本，提供了对 STOMP 1.2 的支持、支持 Web-STOMP 和 MQTT 的新插件、更加友好的集群命令、对每条消息的 TTL 控制以及多个方面的性能提升。

原文链接：<http://www.infoq.com/cn/news/2012/11/rabbitmq-3-0>

Moscrif：用 JavaScript 进行跨平台移动开发



Moscrif 是构建在定制虚拟机上的跨平台移动开发环境。尽管该平台提供了访问原生设备的功能，但编程语言却是 JavaScript 的一个定制版本。

原文链接：<http://www.infoq.com/cn/news/2012/11/Moscrif>

Pex：来自微软研究院的单元测试工具

作者 [Anand Narayanaswamy](#) 译者 [臧秀涛](#)



Pex 是微软研究院开发的一个 Visual Studio 插件与测试工具。该工具能检查源代码，还能做诸如给出测试建议、选取参数值等许多工作。

原文链接：<http://www.infoq.com/cn/news/2012/11/pex>

ASP.NET 的新特性

作者 [Roopesh Shenoy](#) 译者 [姚琪琳](#)

Scott Hanselman 和 Jon Galloway 在这次 Build 大会上做了一次演讲 ,介绍了 ASP.NET 在 2012 秋季预览版中出现的一些有趣的特性。

原文链接 : <http://www.infoq.com/cn/news/2012/11/aspnet-build-2012>

ModelMapper:从对象到对象的映射库

作者 [Kostis Kapelonis](#) 译者 [潘志明](#)

ModelMapper 是一个对象到对象的映射库 ,可以消除将对象从一种形式复制为另一种形式时的重复代码。通过观察属性名称 ,它能够执行自动映射 ,或定义描述该映射的提示。

原文链接 : <http://www.infoq.com/cn/news/2012/11/modelmapper>

Google 发布支持 Java 7 的 App Engine 预览版

作者 [Charles Humble](#) 译者 [臧秀涛](#)



Google 的平台即服务 (Platform-as-a-Service) 产品 App Engine 在其 10 月份的更新中包含了对即将到来的 Java 7 支持的预览。

原文链接 : <http://www.infoq.com/cn/news/2012/11/java7-app-engine>

使用 C++/CX 开发 Windows Store 应用程序的注意事项

作者 [Jonathan Allen](#) 译者 [李永伦](#)

在《Diving deep into C++ /CX and WinRT》演讲里 , Marian Luparu 谈到针对 Windows 8 的应用程序的异常处理和性能问题。对于开发者来说 ,最重要的东西是理解 WinRT 和标准 C++ 代码之间的边界如何影响异常处理和性能。

原文链接 : <http://www.infoq.com/cn/news/2012/11/Cpp-WinRT>

用 Visual Studio 2012 Power Tools 来提高生产力

作者 [Anand Narayanaswamy](#) 译者 [姚琪琳](#)

微软最近发布了 Visual Studio 2012 的 Productivity Power Tools , 除了包含 2010 版本的所有扩展外 , 还包括 Power Commands、Color Printing 和 Quick Tasks 三个新的扩展。

原文链接 :

<http://www.infoq.com/cn/news/2012/11/productivity-power-tools-vs2012>

微软开源 Reactive Extensions

作者 [Roopesh Shenoy](#) **译者** [朱永光](#)

Reactive Extensions (Rx)已经由 Microsoft Open Technologies 开源。这增加了它不久之后出现在 Mono 中的可能性。

原文链接 : <http://www.infoq.com/cn/news/2012/11/rx-net-open-source>

JustDecompile 已支持 C#5 和 WinRT

作者 [Anand Narayanaswamy](#) **译者** [臧秀涛](#)

Telerik 的 JustDecompile Q3 2012 支持 C#5 和 WinRT。它包含了两个插件 , 而且可以通过 JustCode 与 Visual Studio 集成。

原文链接 : <http://www.infoq.com/cn/news/2012/11/JustDecompile>

Visual Studio 如何提高 C++ 性能

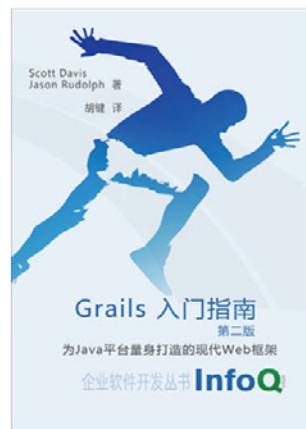
作者 [Jeff Martin](#) **译者** [孙镜涛](#)

相对于之前的版本 , Visual Studio 2012 中优化器的体积几乎增加了一倍。这对开发者是有意义的 , 因为重新编译其 C++ 代码就能获得执行速度的明显提升。将 VS2012 的其他特性作为目标还能获得更大的性能收益。

原文链接 : http://www.infoq.com/cn/news/2012/11/vs2012_CPP

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

封面植物

鹅耳枥



鹅耳枥，为[桦木科](#)乔木植物，稍耐阴，喜肥沃湿润土壤，也耐干旱瘠薄。属于鹅耳枥属。该属植物全世界约有 40 余种，我国约 30 种。分布相当广泛，在华北、西北、华中、华东、西南一带都曾有过它们的足迹。其中有些种类木材坚硬，纹理致密美观，可制家具、小工具及农具等。鹅耳枥种子可榨油，供食用以及工业用。有些种类叶形秀丽，果穗奇特，枝叶茂密，为著名园林观赏植物。

形态特征：乔木，高 5-10 (-15) 米；树皮暗灰褐色，粗糙，浅纵裂；枝细瘦，灰棕色，无毛；小枝被短柔毛。叶卵形、宽卵形、卵状椭圆形或卵菱形，有时卵状披针形，长 2.5-5 厘米，宽 1.5-3.5 厘米，顶端锐尖或渐尖，基部近圆形或宽楔形，有时微心形或楔形，边缘具规则或不规则的重锯齿，上面无毛或沿中脉疏生长柔毛，下面沿脉通常疏被长柔毛，脉腋间具髯毛，侧脉 8-12 对；叶柄长 4-10 毫米，疏被短柔毛。果序长 3-5 厘米；序梗长 10-15 毫米，序梗、序轴均被短柔毛；果苞变异较大，半宽卵形、半卵形、半矩圆形至卵形，长 6-20 毫米，宽 4-10 毫米，疏被短柔毛，顶端钝尖或渐尖，有时钝，内侧的基部具一个内折的卵形小裂片，外侧的基部无裂片，中裂片内侧边缘全缘或疏生不明显的小齿，外侧边缘具不规则的缺刻状粗锯齿或具 2-3 个齿裂。小坚果宽卵形，长约 3 毫米，无毛，有时顶端疏生长柔毛，无或有时上部疏生树脂腺体。花期 4-5 月。坚果，果序下垂，长 6-20 毫米，果期 8-9 月。



架构师 12 月刊

每月 8 日出版

本期主编：张逸

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：张逸，InfoQ 中文站原创团队编辑



怀揣梦想的架构师，沉迷于设计之美，希望写程序能写到老。游走于.NET与Java之间，但更偏好关注架构与设计本质，偶尔还会玩玩Ruby和Python。四川大学软件工程硕士，是一只有着十余年IT从业生涯的老鸟，但还不是专家。著译作包括《[软件设计精要与模式](#)》、《[WCF 服务编程](#)》等，个人网站为：
<http://www.agiledon.com>。