时刻关注企业软件开发领域的变化与创新

架构师

11月 ARCHITECT

特别专题

解读NoSQL

企业中的NoSQL

解读NoSQL技术代表之作Dynamo

图形数据库、NOSQL和Neo4j

摘录与访谈：《HTML5专业开发》

加速你的Hibernate引擎（上）

吴磊畅谈Silverlight
在中国人寿的应用

SOA分析和设计中的错误处理要点

InfoQueue
Enterprise Software Development Community

每月8号出版

篇首语 | **Editor's letter**

银光渐远，孰之过？

最近社区里讨论的比较热烈的当属微软青睐 HTML 5，而忽略自家产品 Silverlight（银光）了，InfoQ 中文站也对其做了深入报道，并引来评论无数。虽然随后微软主管服务器与工具业务部门的总裁 Bob Muglia 发布博客澄清此事，表态说微软会加大对 Silverlight 的投入。但正所谓无风不起浪，再结合微软最近和 Silverlight 相关的高层人员变动，难免给人留下一些猜想的空间。

今年 5 月份 Silverlight 视频技术开发的微软产品部门经理 Brad Abrams 在自己的博客中说将离开微软，加盟 Google；同一月微软致全体员工邮件说娱乐与设备部门总裁 Robbie Bach，首席体验官兼首席技术官、XBox 之父 J Allard 将离开微软。虽然此后离职的商业软件部门总裁 Stephen Elop 和首席软件架构师 Ray Ozzie 和 Silverlight 关系均不大，但多少也印证了微软在努力调整自己的产品结构。

回想一下 Silverlight 面世时对 Flash 的攻击，以及在前端市场上和 Flash 的互相倾轧，再结合最近 Steve Ballmer 和 Adobe 首席执行官 Shantanu Narayen 把酒言欢，希望能联合击败苹果 iOS 平台。怎能让人不怀疑 Silverlight 在 Steve 的心目中已经渐行渐远？

抛开微软的整体战略不谈，八卦一些 Silverlight 在国内一些网站上的实施情况。有好事者此前曾走访国内一些互联网巨头，比如淘宝网、腾讯等，询问 Silverlight 和 Flash 在其企业内部中的应用。负责此事的经理往往是深叹一口气，评论说 Silverlight 其实挺好的，很有想法的一个产品，可惜比较奇怪的是，微软在和他们合作了某个版本之后，从此就销声匿迹了，即使 Silverlight 推出新的版本，微软也无人和他们沟通。加上此前自己的诸多项目也是基于 Flash 研发的，所以他们和 Silverlight 也渐行渐远。一叶落而知天下秋，想来微软内部的同仁应该是最早感觉出 Silverlight 的失宠，也许他们为此努力过，争取过，希望公司能够多追加一些资源，和这些互联网巨头搞好关系。可是公司摇摇头，讳莫如深。

这又能怪谁呢，记得阿里巴巴刚一开始收购雅虎中国的时候，雅虎中国的员工到杭州体验阿里巴巴的文化，结果被钢管舞、肚皮舞等震得目瞪口呆。直到现在，还有人留恋雅虎的工程师文化，认为那才是软件研发应有的味道，是王道，可事实是雅虎中国逐渐没落，阿里巴巴冉冉升起。不论雅虎中国是被玩死的，还是大势所趋，结果是它正在远离这个世界的中心。有人说，互联网时代需要社区、需要草根、需要口碑、需要……，可是黑暗中有人回应说"Apple 也活的挺好的啊"；是啊，那你能说这全是那个首席执行官的错？好像也不公平的。

算了，其实也不是谁的错，只是这个世界变化太快了，让我们所有人都有那么点晕头转向，对不对？

本期主编： 霍泰稳

目　　录

[篇首语]

银光渐远，孰之过？ 1

[人物专访]

蔡学镛谈复杂事务处理（CEP） 4

[热点新闻]

HTML5 已经起飞 9

ADOBE 展示 HTML5 动画制作 IDE 10

MEEGO 1.1 发布 11

PERCOLATOR：大数据集增量更新系统 12

[特别专题]

企业中的 NOSQL 15

解读 NOSQL 技术代表之作 DYNAMO 28

图形数据库、NOSQL 和 NEO4J 34

[推荐文章]

摘录与访谈：《HTML5 专业开发》 52

加速你的 HIBERNATE 引擎（上） 57

吴磊畅谈 SILVERLIGHT 在中国人寿的应用 70

SOA 分析和设计中的错误处理要点 74

[新品推荐]

MEMBASE 公司公开发布 MEMBASE 服务器.....81

LLVM 2.8 发布.....81

GOOGLE WEBP--让图片更小，让页面访问速度更快.....81

ADOBE 展示 HTML5 动画制作 IDE.....82

ADOBE 发布 64 位 FLASH PLAYER 预览版，支持 IE9.....82

微软发布 **TEAM FOUNDATION SERVER** 备份工具.....82

[推荐编辑]

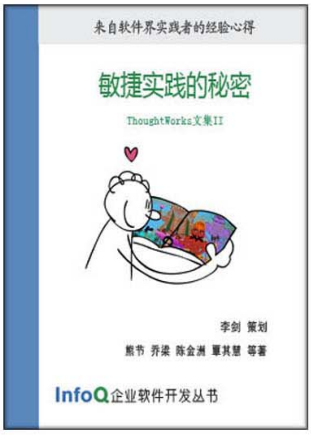
.NET 社区编辑 赵颉.....83

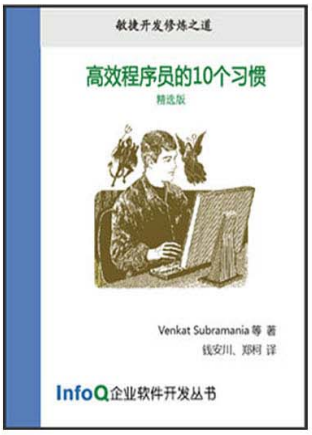
[封面植物]

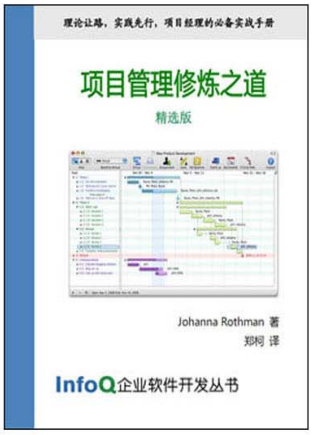
天女木兰.....84

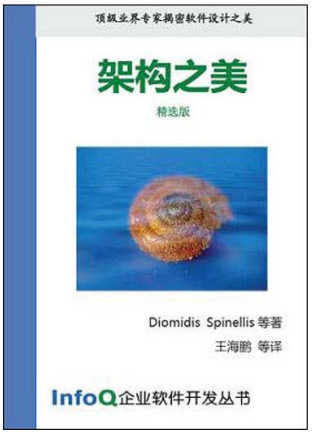
InfoQ企业软件开发丛书

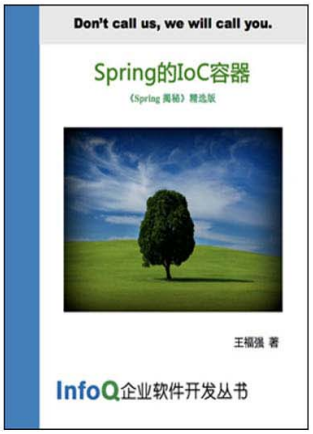
欢迎免费下载











商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

蔡学镛谈复杂事务处理（CEP）

蔡学镛，台湾清华大学硕士，曾任程序员、技术经理、技术总监。2001 年进入.net 领域，近年则倾向于使用动态语言。因为工作关系，他接触到 CEP），并完成了一套相应的系统。我们这期人物专访就专门刊出一篇在 CEP 技术方面对他作出的专访。

蔡学镛，台湾清华大学硕士，曾任程序员、技术经理、技术总监。除了将技术知识用于软件开发相关的工作之外，他也担任过培训班讲师、研讨会讲师、技术图书翻译与编辑、技术专栏作家等工作。他撰写的《Java 夜未眠》一书，至今仍常被 IT 圈提及。蔡学镛于 1995 年就开始玩 Java，在 2001 年进入.NET 领域，近年则倾向于使用动态语言。因为工作关系，他接触到 CEP（Complex Event Processing），并完成了一套相应的系统。

InfoQ：按惯例，请您先做一下自我介绍。然后，再向我们的读者介绍一下您在这次大会上讨论的主题？

蔡学镛（以下简称学镛）：大家好，我是支付宝技术部的蔡学镛（编辑注：现已离职），这次到 QCON 上演讲的主题是 CEP，刚好也算是我在支付宝的一个项目的心得体会。基本上，CEP 系统能够让我们把很多来自四面八方，各个系统的一些事件搜集起来，从中萃取出一些更大、更有意义、更有用的信息，这些信息不管是对制订公司、运营和行销的决策，还是对发现系统潜在的问题、漏洞和威胁都有帮助。这种系统就是 CEP 系统，它是准实时的，跟其他一些系统，比方说数据挖掘，又不太一样。那些系统是在事后才对过去的历史数据进行挖掘。这次跟大家分享的就是 CEP 这个技术。

InfoQ：在未来的企业中，**CEP** 会充当一个什么样的角色，它如何跟我们企业中其他的系统进行交互，完成企业的业务目标？

学镛：大家之所以会对 CEP 比较陌生的原因，是因为 CEP 过去主要用在银行业、金融业。例如，他们可以通过 CEP 系统来判断某一些持卡人的信用卡操作是有问题的，可能是这个卡掉了，或者可能因为卡所处的状态而就把卡给锁住了。但是这样就太局限了，我认为 CEP

的用途非常广，它可以用在任何企业里面。任何一家企业里面都有许多系统，系统之间交互和系统内部都会产生大量信息，如果我们能把这些信息善加分析、处理和使用的话，会对企业经营有很大帮助。至于怎样应用，就很难讲了，必须要看企业是哪种类型。但我相信，不管是什么样的企业都可以使用到 CEP。

InfoQ：在选择 **CEP** 产品和解决方案的时候，我们要考虑哪些因素？市面上主要的 **CEP** 产品有哪些，各有什么优缺点？

学镛：市面上提供 CEP 产品的厂商，基本上就是做数据库的那几家，我就不特别提及了。另外一些较小的厂商，如 TIBCO，也不能说它小，它其实是 CEP 最大的厂商，号称有 40%的市场占有率，而且我看他们的 CEP 确实也不错。还有一些小厂商，像 StreamBase；另外一些开源的，像 EsperTech，都可以在产品选型时考虑。

那么，要考虑哪些因素呢？我觉得有两大重点。第一，所提供的 EPL 语言。EPL 语言可以让我们描述哪些事件，在什么状况发生了我们感兴趣的符合条件的事件。这个描述语言本身如果不够灵活，表达能力不够好的话，那势必这个系统买回来对你的作用也不大，甚至可能会造成很严重的误报警率。误报警率一高，这个系统就不堪使用，所以我觉得 EPL 是最重要的一个考量点。除了 EPL 之外，我认为引擎本身的效率也非常重要，因为 CEP 面对的是海量事件、海量数据，如果没有一个很强有力的算法和引擎的话，整个系统效率绝对会是一个非常大的问题。

InfoQ：既然市面上已经有了这么多 **CEP** 的产品，为什么支付宝还选择自行开发？这个自行开发的解决方案，跟这些产品相比，有什么独到之处？

学镛：其实，早期我们评估过是否采用开源 CEP 框架。但我的评估结果是它的能力基本上达不到我们的要求，原因就是刚才我提到的第一点，EPL 的描述能力。所以，我认为可能还是自行开发会好一些。我们采用了状态机的设计方案，后来确实证明这个方案完全符合支付宝现金流的需求，我们现在不管是担保交易，还是更复杂的业务，再复杂的规则，即便整个状态图大如地图，这个引擎都能够把它描述出来。

确实，现在回想起来，当时决定自行开发引擎的做法是对的。但我不建议所有企业都自行开发，还是得量力而为。因为，如果你做的东西是别人已经做好的话，你可能得评估，自己有没有办法做得更好，或者是自己花一点钱就买得到的东西，是不是得自己开发？

InfoQ：这个 **CEP** 解决方案目前在支付宝内部的应用情况怎样，目前处于什么样的阶段，大概主要应用在哪些业务中？

学镛：支付宝内部其实很多部门都在用 CEP 了，虽然我们不会特别强调 CEP 这个词，但其

实都已经在用这样的概念了。采用的解决方案有很多，有买来的，有自行开发的。例如，支付宝里面有一个非常有名、非常经典的例子，CTU。CTU 大家应该都看过，这是一个 24 小时反恐的政府组织，他们做的事情就是要针对恐怖分子去做一些应对措施。支付宝因为牵涉到金融，所以我们也对这部分特别谨慎，这一直是支付宝非常核心的技术，大量地在使用 CEP。而且每隔一段时间就会发布新的供给模型，让支付宝会更加完善。这是一个例子，它们是用来防止网络攻击以及洗钱等这些行为。另外，支付宝还有一些比较小的 CEP 应用。比如，在营销、协助公司决定生产或判断趋势方面都有应用。至于我开发的这个总督系统，其实是比较新的，它的目的就是用来侦测我们各个系统之间的信息交互有没有问题，属于比较动态的系统校验。这些都是支付宝里面用到的 CEP。

InfoQ：您在演讲时候也提到准确性的问题。您开发的这个 **CEP** 解决方案，在性能和准确性方面表现如何？有没有相应的数据或者案例来支撑您的说法？

学镛：由于目前这个产品还属于比较前期的阶段，所以还没有特别去针对它收集相关数据，这个数据我给不出来。但是它的效率的确非常好，因为我们现在所有的计算，都是在内存里面而不是在数据库中进行的，它的效率绝对会比数据库方式来得快。我们现在就是几台服务器，处理支付宝目前的那些系统已经足够了，因此也没有特别针对项目进行调整。当前的效率都还可以，但我没办法给出比较确切的数据。

InfoQ：那准确性呢？

学镛：关于它，我倒是可以给出些数据。我们做过一些像担保交易、即时到帐等这类支付宝比较经典的应用，基本上所有的支付宝应用都是担保交易和即时到帐，它们大概一直占了百分之八九十，所以已经算是涵盖支付宝大部分的应用。监控这些业务，之前的误报警率是 3%，已经很低了。而且现在，一旦发生误报警，我们就会检查为什么会误报警，看看规则里面少了哪些判断，然后再去补充规则，所以现在又降低了，可能是 1%多点，而且又在不断地持续降低，这是我们这个总督系统的现状。至于其他像 CTU 的方式，那边的数据我就不清楚了，那边数据的效率可能会高一些。

InfoQ：您曾经说过，在您开发的 **CEP** 解决方案中有一个规则 **DSL**。**DSL** 现在相当热门，您能不能对此详细介绍一下？

学镛：我认为状态机的描述，最好能够让它非常简单，这样就可以让更多人能够去写规则，放到我们的这个平台上，这个 CEP 就能应用到更多场合。如何让这个规则可以很简单、很容易地写呢？DSL 是关键。DSL 是第一种方式，另一种方式就是利用易用的绘图工具，让我们可以去做一些表单设置或者是状态图的绘制。我们先是采取 DSL 的方式，未来当人力更充足

时，我们就会去做这个图形工具，我相信这两个阶段都达到之后，使用我们的系统或者是在上线之后再修改规则，都会更容易、更快。

InfoQ：您刚才说过支付宝的 **CEP** 解决方案，山头林立，未来是否有可能出现一个统一的 **CEP** 平台？

学镛：有可能，这就是我的野心了。我期望我做的这个平台，能够尽量多的发挥它的价值。我认为，因为这个平台有非常大的潜力，所有希望借目前的这些应用，慢慢扩展到未来也能够去做网络攻击、洗钱的侦测。当系统做到更好、更稳定的时候，能够把它推荐到阿里巴巴的其他子公司，像淘宝或 B2B，让他们也能够用。当然，这是我的目标啦，都还是未知数，现在言之过早。

InfoQ：祝您早日完全心愿。如果我想说服一家企业去采用 **CEP** 的话，您有何建议？回顾当初支付宝内部决定采用 **CEP** 解决方案的过程，有没有一些经验可以跟我们分享？

学镛：企业如果想采用 CEP，不应该是为了用 CEP 而用 CEP，应该是确切发现公司里面的某些地方能够通过 CEP 带来价值。不管是营销，还是其他方面，先要找到应用点。找到之后，然后再去评估这个应用点值不值得花大钱去做 CEP。因为提供 CEP 系统的厂商都是大厂商，东西都挺贵，这个可能要考量的第一点。如决定采用，那这个应用值不值得投入这么多钱去买一个 CEP，或者是投入这么多人力开发一个 CEP，说到底还是应用。你刚才的第二个问题是？

InfoQ：因为支付宝现在已经有经验了，有什么可供借鉴的？

学镛：支付宝倒是没什么问题，在支付宝内部要推行 CEP 非常容易，因为公司本身的属性虽说是网络业，但有一大部分是金融业，所以大家的安全意识非常非常的强。就以 CTU 来说，在支付宝里面算是最重要、最大的 CEP 了。其实，由于支付宝一成立就立刻有了这个 CTU，它等于是伴随着支付宝公司成长的，在支付宝没有推广不下的可能。

InfoQ：对于想要了解和学习 **CEP** 的开发者，您有什么建议，能否给出一些参考资料和学习方法？

学镛：我觉得可以先从各个 CEP 厂商的技术白皮书着手。技术白皮书一般都较为简短，可能会比较好读。像 TIBCO、 StreamBase、Oracle，这几家的 CEP，都可以从这里着手。另外，有本书其实挺好的，去年刚出版，国内可能没有引进，或许可以通过英文站点从国外购买，书名是《Event Processing: Designing IT System for Agile Companies》，McGraw Hill 出版社出版，作者名字不记得了。这本书非常棒。而且这本书里面提到的不只是 CEP，基本上 CEP 只占一章，其他讲的都是事件处理，重点是怎么结合起来成为一个整体，我觉得 CEP 不是单独存在

的，它会搭配很多其他的事件处理的概念，所以这本书整个对它进行了一个完整的说明。另外，今年开始也会陆续有一些面向程序员，不是着重架构的，让程序员看的 CEP 书籍。像是 Manning 出版社今年就会出一本，里面应该有 CEP，这些都是不错的参考资料。相对其他领域，如 SOA，大家对它可以说是耳熟能详了，说知道 CEP 的确实是少得可怜。

InfoQ：非常感谢。

学镛：谢谢。

原文链接：http://www.infoq.com/cn/articles/cxy-cep

相关内容：

- DSL的演进
- 开发复杂的外部DSL
- CEP是EDA或SOA的必要条件吗？
- 关于复杂事件处理和事件驱动架构的争论

时刻关注企业软件开发领域的变化与创新

InfoQ中文站
www.infoq.com/cn

我们的**使命**：成为关注软件开发领域变化和创新的专业网站

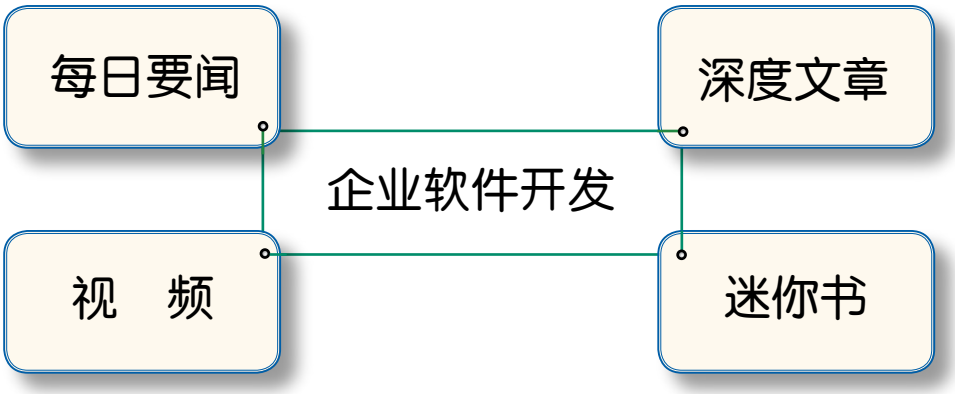
我们的**定位**：关注高级决策人员和大中型企业

我们的**社区**：Java、.NET、Ruby、SOA、Agile、Architecture

我们的**特质**：个性化RSS定制、国际化内容同步更新

我们的**团队**：超过30位领域专家担当社区编辑

……



讨 论 组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
广告合作：sales@cn.infoq.com

HTML5 已经起飞

作者 Abel Avram 译者 侯伯薇

根据 Mefeedia 的调查，当前 Internet 上发布的视频中，有 54%都提供了 HTML5 的格式；并且 Adobe 和 Sencha 都发布了新的 HTML5 编辑工具，这些都显示 HTML5 已经起飞。

Mefeedia 是一家在线视频门户网站，他们最近做了一项研究，想要知道已经有了多少 HTML5 的内容。 该项研究调查了拥有上百万条内容的视频索引，并且涉及到超过 33,000 名视频发布者，最终做出的结论是，现在以 HTML5 格式提供的在线视频内容与五个月前相比已经翻倍，从 26%提升到 54%，而与今年年初相比则增长了五倍，那时的比例只有 10%。 这种增长的驱动力在于移动设备，Flash 仍然是桌面计算机中的首选播放器。大多数站点都在提供 HTML5 视频的同时也提供 Flash 格式的视频，从而用户会基于播放视频的设备来选择合适的格式。

同时，几家厂商已经发布了新的可视化的 HTML5 编辑器。其中一种是来自于 Adobe 的 Edge，它是一种针对 Photoshop、Illustrator 和 Flash 专业用户用来创建 HTML5 动画的原型工具。Adobe 是受引入 HTML5 标准影响最大的公司，他们最初反对这项标准，但在对其有了深入了解之后，现在想要成为 HTML5 编辑工具中的领导者。InfoQ 在之前的新闻中提到过 Edge。

最近另一个发布的工具是 Sencha Animator，它是一种基于 GUI 的编辑器，针对的是想要创建 HTML5 动画的交互设计师。动画是使用 Ext JS 创建的，这是一种跨浏览器的 JavaScript 库，为 RIA 应用程序提供了 widget，并且它会生成能够与所有 JS 库协作的纯粹的 CSS3 动画代码。我们可以创建 2D 或者 3D 的对象，然后对其进行移动、伸缩，倾斜和旋转，并且添加像渐变色、模糊、反射和阴影之类的特效。代码能够在 Apple iOS 上进行硬件加速，以创建平滑的动画。

发布在 web 上的 HTML5 内容的数量，以及新工具——特别是来自于创建了 Flash 的公司 Adobe 的工具——显示，HTML5 全盛时期即将来临。

原文链接：http://www.infoq.com/cn/news/2010/09/big-ball-of-mud

相关内容：Adobe 为 Illustrator CS5 推出 HTML5 扩展包、HTML5 应用于生产为时尚早

Adobe 展示 HTML5 动画制作 IDE

作者 Dionysios G. Synodinos 译者 崔康

Adobe 在今年的开发者大会上向公众展示了一款用于 HTML5 动画制作的 IDE。这款工具（内部称为 Edge）采用 WebKit 渲染引擎来预览动画并提供类似 Dreamweaver 的源代码编辑模式。

Edge UI 遵循和其他 Adobe 工具一样的原则，因此 Flash 开发人员不会感到陌生。它提供一个 Stage 窗口显示页面布局，底部的时间轴窗口显示各种元素的动态变化。在 Stage 窗口右侧是 DOM 元素的布局面板。选择一个层次会显示该层的属性并支持 2D 和 3D 的 CSS 转换。还有一些类似 Flash 的元素，如 Action、Symbol 等。

而在底层，Edge 使用 jQuery 和 CSS 来管理过渡：

```
// Action executed on a specific event , e.g. button is clicked
$.Edge.symbol.get('#pageHolder').showContent('coffe');
```

从演示 demo 看，Edge 好像也在使用 jQuery UI 和 jQuery Easing plugin。

感兴趣的读者朋友可以观看 AdobeTV 视频《Mark Anderson 演示 Edge IDE 原型》。

目前还不清楚 Adobe 需要多久才能完成这个工具以及是否会很快支持更多 HTML5 的核心技术，如<canvas>和<video>。

原文链接： http://www.infoq.com/cn/news/2010/10/adobe-html5-animation-ide-edge

相关内容：

- Adobe AIR登陆Android平台
- Adobe为Flash Builder发布ActionScript代码覆盖插件
- 跨平台部署和企业应用：Adobe Flash Platform 最新进展
- Adobe发布 64 位Flash Player预览版，支持IE9
- HTML 5 案例研究：使用WebSockets、Canvas与JavaScript构建noVNC客户端

热点新闻 | News

MeeGo 1.1 发布

作者 崔康

MeeGo 是诺基亚和英特尔联合推出的 Linux 系统，其 1.1 版最新发布，采用了 Linux Kernel 2.6.35 和 Qt 4.7，包含了核心 OS 1.1 和针对上网本、车载资讯系统、手机的 UX 更新等。

主要功能包括：

- 核心 OS 1.1——针对所有 UX 的统一基础操作系统。
- 上网本 UX 1.1——针对上网本的完整核心应用集。
- 车载资讯系统（In-Vehicle Infotainment）UX 1.1——包含采用 Qt 4.7 构建的 IVI 示例主屏幕和任务栏以及语音识别。
- 手机 UX 1.1——实现了基本的开发 UX，包括语音通话、短信、浏览网页、音乐和视频播放、图片查看、连接管理等。
- SDK 1.1 beta——针对 MeeGo 核心 OS 和支持 UX 的 SDK 将在 MeeGo 都柏林高峰会议之前发布。

MeeGo 1.1 实现包括 Linux Kernel 2.6.35、X.org server 1.9.0、Web Runtime、Qt 4.7、Qt Mobility 1.0.2，除此之外还引入了一些先进的工具，如 oFono 电话协议栈、ConnMan 连接管理器等。

开发人员可以通过 MeeGo API 使用这些技术，并且通过最新的 Qt 4.7，他们能够利用 QML 提供的脚本技术更方便的自动化触摸型 UI 应用。

未来的 MeeGo 1.2 预计在 2011 年 4 月份推出，将包含手机 UX 的完整应用集，并且还有成百上千的功能正在开发中，它的路线图将很快对外发布。

InfoQ 中文站最近采访了和诺基亚合作密切且在移动开发领域活跃的开发商，他们大多表示对 Meego 保持关注甚至已经将其作为开发平台，感兴趣的朋友可以下载最新的 MeeGo 1.1 系统。

原文链接：http://www.infoq.com/cn/news/2010/10/meego1.1

热点新闻 | News

Percolator：大数据集增量更新系统

作者 Jean-Jacques Dubray 译者 陈曦 & 赖翥翔（Jason）

随着需要收集和处理的数据规模以惊人的速率增长，曾经只有 Google 级别的系统才会遇到的可伸缩性需求变得更普遍，并常常需要专门的解决方案。Daniel Peng 和 Frank Dabek 最近发表了一篇论文，介绍 Google 索引系统 Percolator 的技术细节。Percolator 目前运行在数千台服务器上，存储了数十 PB 的数据，并且每天要处理数十亿次的更新。

> 在抓取网页的同时进行索引更新，意味着在新文档不断加入时，需要对已有的总文档库进行持续地更新。这是通过小规模、独立的变换实现海量数据转换任务的一个典型范例。现有的技术基础平台恰恰不能胜任这样的任务：传统 DBMS 无法满足存储量和吞吐率的需求，而 MapReduce 和其它批处理系统无法逐个处理小 规模更新，因为它们必须依赖于创建大量的批处理任务才能获得高效率。

Daniel 和 Frank 解释说，尽管索引的过程是一项批处理任务，可以通过一系列的 MapReduce 操作来表现。但每次重新爬完一些页面后要更新索引 的时候，由于新增文档和已有文档之间存在链接引用的关系，只对增量部分运行 MapReduce 操作是远远不够的，实际上必须基于整个文档库进行 MacReduce 操作。事实上在 Percolator 出现之前，索引就是以上述的方式更新的。这样带来的主要问题就是由于要对整个文档库重新处理而产生的延迟。

解决此问题的关键是优化增量数据的处理方式。Percolator 的一个关键设计理念是：提供对库中文档的随机访问，以实现对单个文档的处理，从而避免了像 MapReduce 那样对文档全集进行处理。Percolator 通过“快照隔离”实现了遵从 ACID 的跨行及跨表事务，从而满足多线程在多台服务器上对文档库进行转换操作的需求。Percolator 还提供了“观察者（observer）”机制，在用户指定的列发生更新之后，这些观察者会被系统触发，以帮助开发者追踪计算过程所处的状态。

论文作者补充到：

> Percolator 是专门针对处理增量更新而设计，但不是用于取代大多现有的数据处理解决方案。那些不能被拆分为单个微小更新的计算任务（比如对一个文件排序）仍然最好由 MapReduce 承担。

Percolator 更适合于在高一致性及在数据量和 CPU 等方面有很高需求的计算任务。对于 Google 来说，它的主要用途是将网页实时地添加到 Web 索引中。运用 Percolator，Google 可以在抓取网页文档的同时来对文档进行处理，从而将平均延迟降低为原来的百分之一，平均文档寿命 （document age）降低 50%。

Percolator 建立于分布式存储系统 BigTable 之上。集群里的每台服务器上运行着三个可执行文件：worker，BigTable tablet 服务器和 Google File System chunkserver 服务器

> 所有观察者都被关联到 Percolator worker 上，后者会对 BigTable 进行扫描，一旦发现更新过的列就会在 worker 进程中以函数调用的方式触发（"notification"） 相应的观察者。观察者通过向 BigTable tablet 服务器发送读、写 RPC 请求来运行事务，继而触发后者向 GFS chunkserver 服务器发送读、写 RPC 请求。

Percolator 没有提供用于事务管理的中心服务器，也没有全局锁侦测器。因为 Percolator 不需要像运行 OLTP 任务的传统 DBMS 一样，对低延迟有很高要求，所以它采取了一种延迟的方式来清理锁，也因此在事务提交时造成了数十秒的延迟。

> 这种方法增加了事务冲突时的延迟，但保证了系统可以扩展到几千台服务器的规模……尽管增量数据处理在没有强事务的情况下也能进行，但事务使得开发者更容易地去分析系统的状态，并避免将错误引入到长时间运行的文档库中。

Percolator 的架构可以在普通廉价服务器集群上线性扩展多个数量级。在性能方面，Percolator 处于 MapReduce 和 DBMS 之间。和 DBMS 相比，在处理同样数量的数据情况下，Percolator 由于其分布式架构，资源消耗远大于 DBMS，同时它还引入了约 30 倍的额外性能开支。和 MapReduce 相比，Percolator 可以以低很多的延迟来处理数据，同时需要额外的资源来支持随机查找。Percolator 自 2010 年 4 月开始为 Google web 搜索提供索引，它利用合理的额外资源消耗，获得了更低的延迟。

不知道读者们是否看见或者预见了对处理海量数据集的快速增长的需求了没有？前不久 Phil Wehlan 问了同样的问题，希望大家给他提供反馈。

原文链接：http://www.infoq.com/cn/news/2010/10/google-percolator

相关内容：设计 Web 应用程序时要注意可伸缩性、Web 2.0 应用客户端性能问题十大根源

Adobe Flash Builder 4 简体中文正式版

Adobe® Flash® Builder™ 4（之前称为 Adobe Flex® Builder™）软件是一款专业级的开发工具，可以帮助软件开发人员使用开放源 Flex 框架快速开发跨平台富 Internet 应用程序 (RIA) 和内容。。Flash Builder 4 有许多新的特性，可以与 Adobe Flash Catalyst 集成、以数据为中心的开发、与 Adobe Creative Suite 设计工具集成、交互式数据可视化、智能代码编辑和调试等。

现在就来下载使用

即可体验流畅的开发过程，集成更多基础设施

开发更具表现力的应用程序

Adobe Flash Builder 4 简体中文正式版285M

立即下载

InfoQ中文站提供以下Adobe Flash Platform相关产品和工具的高速下载：

- Flash Platform一览表
- Flash Player
- Flash Catalyst Beta 2 292M 立即下载

特别专题 策划：InfoQ 中文站； 执行：霍泰稳 葛明见

NoSQL，即非关系型数据库。传统关系型数据库在面对 Web2.0 网站，尤其是那些超大规模和高并发 SNS 类型的纯动态网站时，已经显得力不从心。NoSQL 由此应运而生，它在处理超大量数据方面的优势很快得到了大家的认可。目前 Google 的 BigTable 和 Amazon 的 Dynamo 使用的就是 NoSQL 型数据库。本期《架构师》从 Infoq 网站上挑选几篇关于 NoSQL 方面的文章，希望对大家了解非关系型数据库方面有所帮助。

解读 NoSQL

企业中的 NoSQL

解读 NoSQL 技术代表之作 Dynamo

图形数据库、NOSQL 和 Neo4j

特别专题

企业中的 NoSQL

作者 Stefan Tilkov 译者 苑永凯

引子

作为企业架构师，我的职业习惯之一，就是不断的探求各种新的有前景的概念和思想，看其是否有潜力为我所服务的来自各行各业的企业客户带来价值。同样出于对这种理念的追求，我对 NoSQL 领域的关注了也有一段时间了，甚至从这个术语产生(或者错误的产生?)之前就开始了。Google 首先在这方面点了一把火，发布了论文 Big Table 架构，对关系数据库是银弹这种普遍的信念提出了质疑，而 Amazon 关于 Dynamo 的论文则紧随其后。过去的一年中我们见证了 NoSQL 强劲的势头，在这一领域有多达 25 种产品/解决方案发布，并且 NoSQL 的触角已经伸向了业界的各个角落。在此前提下，我最近考虑深入这一领域，评估一下我的客户究竟如何才能从这种 NoSQL 运动中获益。不仅如此，我还想探究对于企业来说，是否是到了该认真考虑采纳 NoSQL 的合适时机了。

什么是 NoSQL——快速回顾

像许多关注这一领域的人一样，我不喜欢从本质上将 SQL 与 NoSQL 这一术语对立起来。同时我对该术语现有的解释"Not Only SQL"也不甚满意。对我来说，我们这里所讨论的并非是是否使用 SQL。(相反的是，我们仍然可以选择类似 **SQL** 这样的查询接口(缺少对 **join** 等的支持)来与这些数据库交互，使用现有的资源和技术来管理开发伸缩性和可维护性。) 这一运动是要找到存储和检索数据的其他高效的途径，而不是盲目地在任何情况下都把关系数据库当作万金油。因此，我认为'Non Relational Database'(非关系型数据库)能够更好的表达这一思想。

无论采用哪个名字，“非关系型数据库”这一范围所传达出来的“囊括所有”类型的意味，使得这一概念比较模糊(并且它还是否定型的)。这又使得人们(特别是企业中的决策者)对于哪些是属于这个范围，哪些不是，更重要的是，对他们来说这到底意味着什么，感到非常迷惑。

为了解答这些疑问，我尝试通过以下几点特征的描述，来刻画“非关系型数据库”的内在本质。

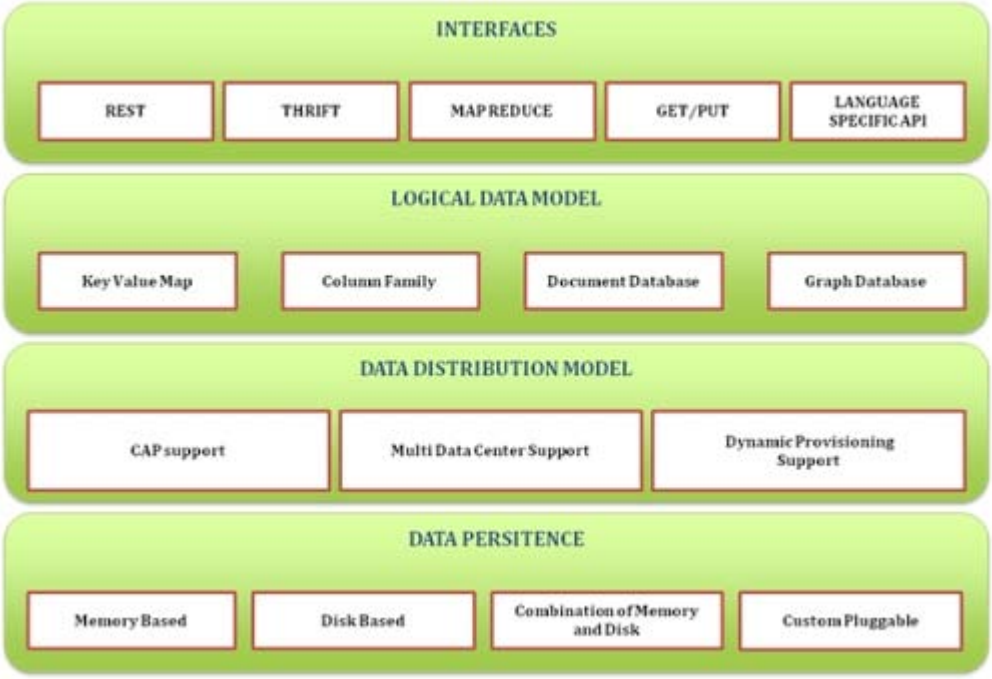
所谓“非关系型数据库”指的是

1、 使用松耦合类型、可扩展的数据模式来对数据进行逻辑建模(Map，列，文档，图表等)，而不是使用固定的关系模式元组来构建数据模型。

2、 以遵循于 CAP 定理（能保证在一致性，可用性和分区容忍性三者中中达到任意两个）的跨多节点数据分布模型而设计，支持水平伸缩。这意味着对于多数据中心和动态供应（在生产集群中透明地加入/删除节点）的必要支持，也即弹性(Elasticity)。

3、 拥有在磁盘或内存中，或者在这两者中都有的，对数据持久化的能力，有时候还可以使用可热插拔的定制存储。

4、 支持多种的'Non-SQL'接口(通常多于一种)来进行数据访问。



围绕着图中四个特征的（数据持久性、逻辑数据模型、数据分布模型和接口）“非关系型数据库”的各种变形，在最近的一些文章中有详尽的描述，并且在因特网上有着广泛的传播。所以我就不做过多繁复的描述，而是通过一些例子对关键的方向进行总结，供快速参考。

接口——REST (HBase ,CouchDB ,Riak 等) ,MapReduce (HBase ,CouchDB ,MongoDB ,Hypertable 等)，Get/Put (Voldemort，Scalaris 等)，Thrift (HBase，Hypertable，Cassandra 等)，语言特定的 API(MongoDB)。

逻辑数据模型——面向键值对的(Voldemort，Dynomite 等)，面向 Column Family 的(BigTable，HBase，Hypertable 等)，面向文档的(Couch DB，MongoDB 等)，面向图的(Neo4j， Infogrid

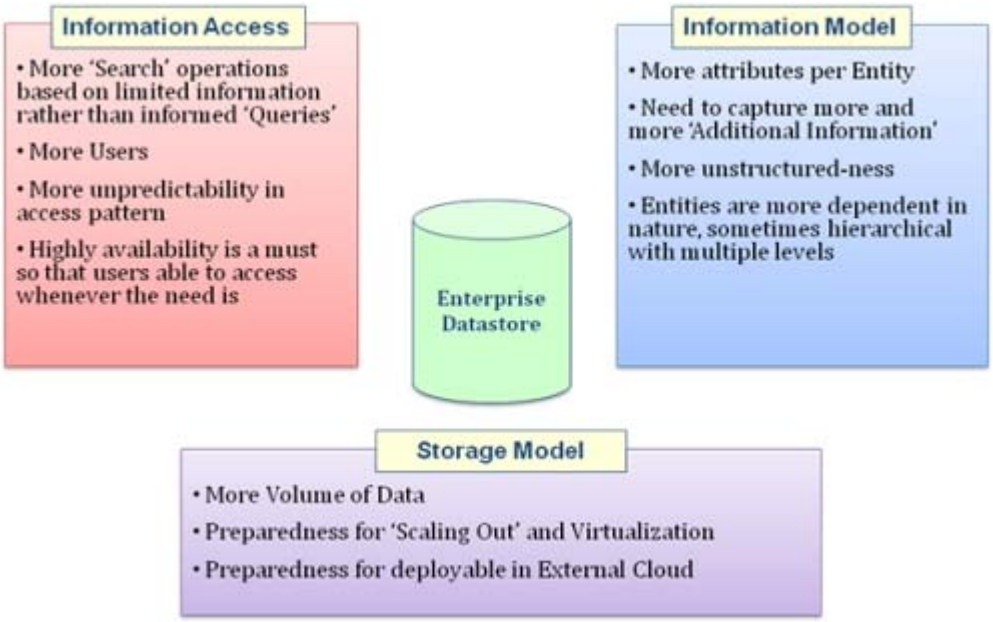
等)

数据分布模型——一致性和可用性(HBase，Hypertable， MongoDB 等)，可用性和可分区性(Cassandra 等)。一致性和可分区性的组合会导致一些非额定的节点产生可用性的损失。有趣的是目前还没有一个“非关系型数据库”支持这一组合。

数据持久性——基于内存的(如 Redis，Scalaris， Terrastore)，基于磁盘的(如 MongoDB，Riak 等)，或内存及磁盘二者的结合(如 HBase，Hypertable，Cassandra)。存储的类型有助于我们辨别该解决方案适用于哪种类型。然而，在大多数情况下人们发现基于组合方案的解决方案是最佳的选择。既能通过内存数据存储支持高性能，又能在写入足够多的数据后存储到磁盘来保证持续性。

如何将其与企业 IT 融合

如今的企业中，并非所有用例都直观地倾向于使用关系型数据库，或者都需要严格的 ACID 属性(特别是一致性和隔离性)。在 80 年代及 90 年代，绝大部分存储在企业数据库里的数据都是结构化的业务事务的“记录”，必须用受控的方式来生成或访问，而如今它已一去不复返了。无可争辩的是，仍有这一类型的数据在那里，并将继续也应该通过关系型数据库来建模，存储和访问。但对于过去 15 年以来，随着 Web 的发展，电子商务和社交计算的兴起所引起的企业里不受控的非结构化并且面向信息的数据大爆炸，该如何应对呢？企业确实不需要关系型数据库来管理这些数据，因为关系型数据库的特点决定了它不适用于这些数据的性质和使用方式。



上图总结了现今以 web 为中心的企业中信息管理的新兴模式。而“非关系型数据库” 是处理这些趋势的最佳选择(较之关系型数据库来说)，提供了对非结构化数据的支持，拥有支持分区的水平伸缩性，支持高可用性等等。

以下是支持这一观点的一些实际应用场景：

日志挖掘——集群里的多个节点都会产生服务器日志、应用程序日志和用户活动日志等。对于解决生产环境中的问题，日志挖掘工具非常有用，它能访问跨服务器的日志记录，将它们关联起来并进行分析。使用“非关系型数据库”来定制这样的解决方案将会非常容易。

分析社交计算——许多企业如今都为用户(内部用户、客户和合作伙伴)提供通过消息论坛，博客等方式来进行社交计算的能力。挖掘这些非结构化的数据对于获得用户的喜好偏向以及进一步提升服务有着至关重要的作用。使用“非关系型数据库” 可以很好的解决这一需求。

外部数据 **feed** 聚合——许多情况下企业需要消费来自合作伙伴的数据。显然，就算经过了多轮的讨论和协商，企业对于来自合作伙伴的数据的格式仍然没有发言权。同时，许多情况下，基于合作伙伴业务的变更，这些数据格式也频繁的发生变化。通过“非关系型数据库”来开发或定制一个 ETL 解决方案能够非常成功的解决这一问题。

高容量的 **EAI** 系统——许多企业的 EAI 系统都有高容量传输流(不管是基于产品的还是定制开发的)。出于可靠性和审计的目的，这些通过 EAI 系统的消息流通常都需要持久化。对于这一场景，“非关系型数据库” 再次体现出它十分适用于底层的数据存储，只要能给定环境中源系统和目标系统的数据结构更改和所需的容量。

前端订单处理系统——随着电子商务的膨胀，通过不同渠道流经零售商、银行和保险供应商、娱乐服务供应商、物流供应商等等的订单、应用、服务请求的容量十分巨大。同时，由于不同渠道的所关联的行为模式的限制，每种情况下系统所使用的信息结构都有所差异，需要加上不同的规则类型。在此之上，绝大部分数据不需要即时的处理和后端对帐。所需要的是，当终端用户想要从任何地方推送这些数据时，这些请求都能够被捕获并且不会被打断。随后，通常会有一个对帐系统将其更新到真正的后端源系统并更新终端用户的订单状态。这又是一个可以应用“非关系型数据库”的场景，可用于初期存储终端用户的输入。这一场景是体现“非关系型数据库”的应用的极佳例子，它具有高容量，异构的输入数据类型和对帐的"最终一致性"等等特点。

企业内容管理服务——出于各种各样的目的，内容管理在企业内部得到了广泛的应用，横跨多个不同的功能部门比如销售、市场、零售和人力资源等。企业大多数时间所面临的挑战是用一个公共的内容管理服务平台，将不同部门的需求整合到一起，而它们的元数据是各不相

同的。这又是“非关系型数据库”发挥作用的地方。

合并和收购——企业在合并与收购中面临巨大的挑战，因为他们需要将适应于相同功能的系统整合起来。“非关系型数据库” 可解决这一问题，不管是快速地组成一个临时的公共数据存储，或者是架构一个未来的数据存储来调和合并的公司之间现有公共应用程序的结构。

但我们如何才能准确的描述，相对于传统的关系型数据库，企业使用“非关系型数据库”带来的好处呢？下面是可通过非关系型数据库的核心特点(正如上一节所讨论的)而获得的一些主要的好处，即企业的任何 IT 决策都会参考的核心参数——成本削减，更好的周转时间和更优良的质量。

业务灵活性——更短的周转时间

“非关系型数据库”能够以两种基本的方式带来业务灵活性。

- 模式自由的逻辑数据模型有助于在为任何业务进行调整时带来更快的周转时间，把对现有应用和功能造成影响减到最少。在大多数情况下因任意的变更而给你带来的迁移工作几乎为零。
- 水平伸缩性能够在当越来越多的用户负载造成负载周期性变化，或者应用突然变更的使

用模式时，提供坚固的保障。面向水平伸缩性的架构也是迈向基于 SLA 构建(例如云)的第一步，这样才能保证在不断变化的使用情形下业务的延续性。

更佳的终端用户体验——更优越的质量

在现今企业 IT 中，应用的质量主要由终端用户的满意度来决定。“非关系型数据库”通过解决如下终端用户的考虑因素，能够达到同样的效果，而这些因素也是最容易发生和最难以处理的。

- “非关系型数据库”为提升应用的性能带来了极大的机会。分布式数据的核心概念是保证磁盘 I/O(寻道速率)绝不能成为应用性能的瓶颈。尽管性能更多的是由传输速率来决定。在此之上，绝大部分解决方案支持各种不同的新一代的高速计算的范式，比如 MapReduce，排序列，Bloom Filter，仅可追加的 B 树，Memtable 等。
- 现今用户满意度的另一个重要的方面就是可靠性。终端用户希望在想要访问应用时就能访问到，并且至少是在当他们分配到时间的时候能随时执行他们的任务。所以不可用的应用需要不惜代价的避免。许多现代的“非关系型数据库”都能适应并支持这一类有着严格和最终一致性的可用性的需求。

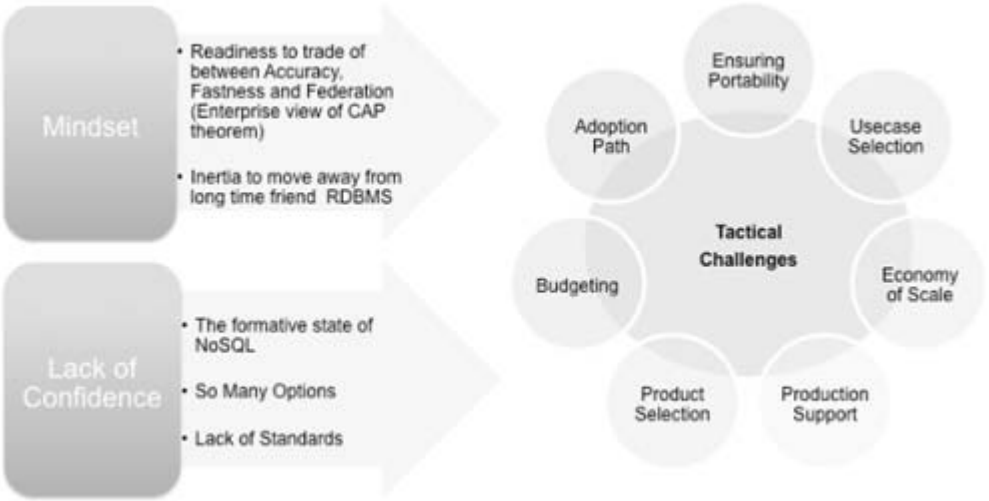
更低的所有者总成本

在如今的竞争市场中，企业 IT 支出随时都要仔细审查，以合理的成本获取合理的质量才值得赞许。在这一领域中“非关系型数据库”在一定程度上胜于传统的数据库，特别是当存储和处理的数据容量很大时。

- 水平伸缩性的基本前提保证了它们可以运行于廉价机器之上。这不仅削减了硬件资本的成本，同时还削减了诸如电力，维护等运维成本。同时这还进一步的为利用诸如云、虚拟数据中心等下一代低成本的基础设施打下了基础。
- 从长期来看，更少的维护能带来更多的运维成本优势。对于关系型数据库，这绝对是一个需要存储大容量数据的场景。为大容量的数据调优数据库需要高超的技艺，也就意味着更高的成本。相较之下，“非关系型数据库”始终提供快速和响应的特点，就算是在数据大幅上升的情况下。索引和缓存也以同样的方式工作。开发者不必过多担心硬件、磁盘、重新索引及文件布局等，而是把更多的精力投入了应用程序的开发上。

企业采用中所面临的挑战

抛开所有这些长远的好处，在企业拥抱“非关系型数据库”之前，当然还需要经历各种各样的挑战。



不考虑因现有思想的转换和缺乏信心而产生的来自高层的阻力，目前我认为的最主要的战术性挑战是：

为“非关系型数据库”认定正确的应用/使用场景

尽管从理论上容易论证并非所有的企业数据都需要基于关系和 ACID 的系统，然而由于关系型数据库与企业数据间多年的绑定关系，要作出所有的数据可以通过非关系的解决方案而解耦的决定仍然有很多困难。许多时候 IT 经理(以及其它对于应用程序负有核心的底线责任的各级人员)不明白他们将会失去什么，这样的担忧对于从关系型数据库转变出来比较不利。企业 IT 最有价值的资产就是数据。因此，要作出决定使用一种不太明确或者未被广泛采用的解决方案来管理同样的数据，这种能力不仅需要转换思维方式，同时还需要来自高层的强大的支持(和推动)。

我们如何选择最适合我们的产品/解决方案

另一个重大的挑战是找出合适的产品/工具来提供“非关系型数据库”。正如前面所提到的那样，现今业界里面有多于 25 种不同的产品和解决方案，它们在四个方面有着不同的特点。正因为每个产品在这四个方面特点各异，所以要选择一个产品来应对所有的需求显得尤为困

难。有的时候，可能在企业的不同部门使用到多种类型的非关系型数据库，最后人们可能会完全出于对标准的需要而转向关系型数据库。

如何获得规模经济

这一想法本质上是从前一个问题分支出来的。如果一个组织需要使用多个非关系型数据库解决方案(由于单个方案的适用问题)，那么保证在技术(开发者，管理者，支持人员)，基础设施(硬件成本，软件许可成本，支持成本，咨询成本)，以及工件(公共组件和服务)方面的规模经济就是一个大问题。这一方面与传统的关系型数据库解决方案比较起来确实更为严峻，因为大部分时间组织的数据存储都是以共享服务的模式在运行的。

我们如何保证解决方案的可移植性

从“非关系型数据库”的发展来看，我们可以很直观地推测在未来的几年中这一领域会有许多变化，比如供应商的合并，功能的进步以及标准化。所以对于企业来说一个更好的策略是不要把宝押在某个特定的产品/解决方案上，以后才可以更灵活的转换到一个更好的经过考验的产品。由于现在的非关系型产品/解决方案大部分是私有的，因此 IT 决策者在考虑尝试“非关系型数据库”之前，不得不认真考虑可移植性这一重要的问题。这纯粹是出于保护现有投资的需要。

我们如何获得合适的产品支持类型

现在的“非关系型数据库”能通过外部组织而提供支持方案的少之又少。就算有，也无法与 Oracle，IBM 或者微软等相比。特别是在数据恢复，备份和特定的数据恢复方面，由于许多“非关系型数据库”在这些方面未能提供一个健壮而易于使用的机制，对于企业决策者来说，仍存在很大的问题。

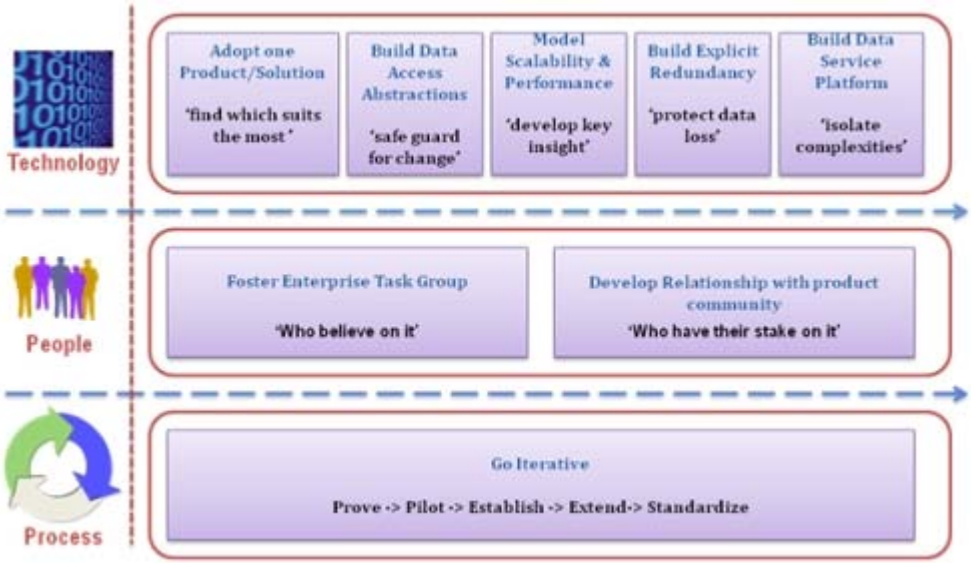
我们如何预算整体成本

与重量级的关系型数据库相比，“非关系型数据库”通常在性能和伸缩性特征方面能提供的数据更少。我也没有发现有 TPC 基准程序方面和类似的其它方面的数据。这将企业决策者置于了一个“没有方向”的情况下，因为他们不知道需要在硬件、软件许可、基础设施管理和支持等方面支出多大的费用。要得出一个预算估计，缺乏判断的数据就成了一个主要的障碍。因此在项目启动阶段，大部分情况下决策者还是会选择基于熟悉的关系型数据库的解决方案。

有时候，就算可以得到这些数字，但也不足以用来形成 TCO 模型并与传统的基于关系型数据库的数据存储和非关系型数据存储进行整体的成本分析（Capex+Opex)比较。通常情况下水平伸缩性所要求的大量的硬件机器(以及软件许可成本，支持成本)，如果与垂直伸缩性乍一比较，会让人觉得战战兢兢，除非由此带来的好处经过基于 TCO 模型的全方位比较仍然被证明是可以持续的。

关于如何采用 NoSQL 的两点思考

这是否意味着目前来看企业应该对 NoSQL 运动持观望的态度呢？并非如此。诚然，“非关系型数据库”对于广泛的采用来说还未到完全成熟的阶段。但“非关系型数据库”作为未来企业骨架的潜力仍不能忽视。特别是不远的将来企业将更多地处理大容量的半结构化/非结构化以及最终一致性的数据，而不是相对而言小容量的，严格结构化的遵循 ACID 的数据。所以现在而言至关重要的是做企业的关键决策人的思想工作，让他们明白企业的数据处理需要使用“非关系型数据库”。在这一过程中，要采取一些渐进的步骤把“非关系型数据库”应用到企业 IT 的一些关键的方面(技术，人员和流程)，并产生一定的价值。这样，就可以用一种缓慢而稳健的方式从整体上来解决我们之前所总结出来的一系列问题。



采用一个产品/解决方案

如今市场上的选择非常多样化，可根据“非关系型数据库”侧重的面不同而进行差异化的处理。与此同时，企业应用场景可能需要不同类型的特点。然而以不同的解决方案来处理不同的应用/使用场景从规模经济的角度出发对于企业是不适宜的。因此最好是根据目标应用的需要最终落实到某一个具体的产品/解决方案上。需要注意的是大多数的解决方案在特性上

都会有一些折中，有些特性可能在其它的产品中可以获得，有些可能只是在发展路线图当中暂时设定了一个位置。因为大部分的产品会在不久的将来不断趋于成熟，因此可以通过不同配置来提供不同的解决方案。所以只要现有的解决方案能适合目前大部分的需要，不妨作为一个起点将其采纳。

选择产品/解决方案的经验法则

- 支持所需要的逻辑数据模型应当被给予更高的权重。这将从实质上决定该解决方案在当前或未来能否灵活地适应不同的业务需求。
- 调查该产品所支持的物理数据模型的合适与否，据此对这一解决方案所需要的水平伸缩性、可靠性、一致性和分区性作出合理的评估。这同样能表明备份和恢复机制的可能性。
- 接口支持需要与企业标准的运行环境对齐。由于这些产品支持多样的接口，所以这一点可以得到很好的处理。
- 只要产品支持水平伸缩性，对于持久化模型的选择就不再重要了。

这里有一份一系列“非关系型数据库”的对照表。对于现在正认真考虑采用的企业来说，这是一个不错的起点。为了更贴近企业本身的情况，从 25+的集合中挑选出的子集所用到的的关键选择标准是：

1、 最重要的一点首先是企业应用程序必须支持有一定复杂程度的数据结构。否则的话，应用程序管理复杂性的责任将变得非常大。我认为比较合理的应当是介于纯粹的键值对与关系型模式中间的一种方案。出于这方面的考虑像 Vlodemort，Tokyo Cabinet 等产品就排除在了我的列表之外。

2、 第二点是以低成本的分片/分区为大容量数据提供水平支持。缺乏这样的支持就使得解决方案与任何关系型数据库无异了。因此像 Neo4J(尽管他有丰富的基于图的模型) ,Redis，CouchDB 等此时此刻就被过滤出我的列表之外了。

3、 最后一条评判标准，在企业级推广之前我会考虑一定程度的商业支持。否则的话，一旦出现生产环境的问题，我该去找谁呢？出于这一点，我不得不将现在的一些明星产品排除在外 ,比如 Cassandra(尽管有很大的可能不久的将来 Rackspace 或者 Cloudera 就会对其提供支持，因为它已经被用于一些生产环境里边了，比如 Twitter，Digg，Facebook)。

有了这些过滤标准，我可以精简这一列表，符合目前企业可用的产品有 MongoDB (下一版本就会提供 shards 支持)， Riak，Hypertable 和 HBase。下面这个表格中总结了这四个产品的主要特性。一个企业可以基于自己具体的实际情况从中作出选择，找到最适合自己需要的

特性。

特性	MongoDB	Riak	HyperTable	HBase
逻辑数据模型	富文档，并提供对内嵌文档的支持	富文档	列家族(Column Family)	列家族(Column Family)
CAP 支持	CA	AP	CA	CA
动态添加删除节点	支持(很快在下一发布中就会加入)	支持	支持	支持
多 DC 支持	支持	不支持	支持	支持
接口	多种特定语言 API(Java，Python，Perl，C#等)	HTTP 之上的 JSON	REST，Thrift，Java	C++，Thrift
持久化模型	磁盘	磁盘	内存加磁盘(可调的)	内存加磁盘(可调的)
相对性能	更优(C++编写)	最优(Erlang 编写)	更优(C++编写)	优(Java 编写)
商业支持	10gen.com	Basho Technologies	Hypertable Inc	Cloudera

数据访问抽象

为数据访问创建一个单独的抽象层对于“非关系型数据库”来说是必须的。它可以带来多方面的好处。首先，应用开发者可以与底层解决方案的细节完全隔离开来。这对于技术方面的伸缩性带来了好处。同时未来如果需要更改底层的解决方案也很方便。这也以一个标准的方式满足了多个应用的要求(即去掉了 Join，Group by 等复杂特性的 SQL)。

为性能和伸缩性创建模型

不管选择怎样的解决方案，使用标准技术(比如排队网络模型，分层排队网络等)来对性能和伸缩性进行建模都是高度推荐的。它能够为基本的服务器规划、拓扑以及整体的软件许可证成本，管理运行等提供必要的数据。这将实质上成为所有预算计划的主要参考数据，并对作出决策提供帮助。

构建显式的冗余

要防止数据丢失，除了将数据复制到备份服务器上，没有其它的办法了。尽管许多非关系型

数据库提供自动复制功能，但仍然存在主节点单点失效的风险。因此最好是使用次节点备份，并准备好用于数据恢复和自动数据修复的脚本。出于这样的目的，应当充分的了解目标解决方案的物理数据模型，找出可能的恢复机制备选方案，基于企业的整体需求和实践来对这些选项作出评估。

构建公共数据服务平台

就像公共共享服务的关系型数据库一样，也可以构建非关系型数据库的公共数据服务来促进规模经济效应，满足基础设施和支持的需要。这对于未来进一步演化和更改也有帮助。这可以作为愿望列表上的最终目标，通过中期或长期的努力来达到这一成熟水平。然而，初始阶段就设立这样的远景有助于在整个过程中作出正确的决策。

壮大企业的技术力量

每个组织都有一部分人对于学习新生的和非传统的事物充满热忱。成立这样的小组，并挑选人员(全职的或兼职的)，密切关注这方面的动向，了解问题和挑战，进行前瞻性的思考，能够为使用这些技术的项目提供方向和帮助。同时，这个小组还可以为决策者澄清炒作的疑云，提供来自真实数据的观点。

建立与产品社区的关系

选择了产品之后，与产品社区建立起良好的关系对于双方的成功都有极大的好处。许多非关系型数据库目前都有十分活跃的社区，非常愿意相互帮助。企业与社区之间的良好合作能给大家带来一个双赢的局面。如能提前对问题和解决方案有了解，那么企业在对某些特性或版本作出决策时就能成竹在胸。反过来，企业又能对产品特性的路线图产生影响作用，这对他们自身和社区都是有利的。另一方面，社区也能从实际层次的问题中得到反馈，从而丰富和完善产品。来自大型企业的成功案例同样能让他们处于领先。

迭代前进

考虑到非关系型数据库相对的成熟度，风险最小的采用策略就是遵循迭代开发的方法论。构建公共数据服务平台和标准化数据访问抽象不可能是一蹴而就的。相反，通过迭代和面向重构的方式能更好的达到目标。运用不太成熟的技术进行转型的过程，中途改变解决方案也不会太意外的。与此同时，采用敏捷的方式来看待事物，能够帮助建立起一个能从管理和实现

两方面不断吸引改进的开放态度。

然而，在这一问题上实现迭代，非常重要的一点是定义一个决策条件矩阵。比如操作指南(和例子)，来判断一个应用的对象模型是否适合关系型或非关系的范围，对基础设施规划作出指导，列出必需的测试用例等等。

结束语

企业的非关系型数据库采用过程中最大的挑战就是转变决策者的思想观念——让他们相信并非所有的数据/对象都适合关系型数据库。最能证明这一点就是选择合适的用例去尝试非关系型数据库，进而证实在合适的背景下，非关系型数据库是比关系型数据库更有效的解决方案。找到一些“非关键业务”(但能立竿见影的)适合于非关系型数据库的项目。这些项目的成功(甚至失败)都能有助于观念的改变。这也能有助于不断学习如何才能以一种不同的方式来更好的采用非关系型数据库。这些少儿学步般的尝试所作出的努力与投入都是值得的，如果企业想要在将来使用“非关系型数据库”来重塑其信息管理体系的话。

关于作者

Sourav Mazumder 目前是 InfoSys Technologies 的首席技术架构师。他在信息技术领域有 14 年以上的经验。作为 Infosys 技术顾问团的主要成员，Sourav 为 Infosys 在美国、欧洲、澳洲和日本的主要客户，提供保险、电信、银行、零售、安全、交通以及建筑、工程、施工等多个行业的服务。他曾参与 Web 项目的技术架构和路线图定义，SOA 战略实施，国际战略定义，UI 组件化，性能建模，伸缩性分析，非结构化数据管理等等。Sourav 参考的 Infosys 自身的核心银行产品 Finacle，也为他提供了丰富的产品开发经验。Sourav 还曾参与开发 Infosys 的 J2EE 可重用框架，和定义 Infosys 在架构方面和开发定制应用方面的软件工程方法。Sourav 的经历还包括在保证架构合规和开发项目的治理方面的工作。

Sourav 是 iCMG 认证的软件架构师，同时也是 TOGAF 8 认证的执行者。Sourav 最近在 LISA 伯克利全球化会议上发表了演讲。Sourav 关于 SOA 的最新白皮书在社区里十分流行。

Sourav 目前关注 NoSQL，Web 2.0，治理，性能建构和全球化。

原文链接：http://www.infoq.com/cn/articles/nosql-in-the-enterprise

特别专题

解读 NoSQL 技术代表之作 Dynamo

作者 陈臻

NoSQL 在过去的一年里，逐渐已经成为了家喻户晓的东西，我（54chen）自从去年开始人人网的 NoSQL 系统 Nuclear 的研发以来，一直看 NoSQL 越来越热，越来越引来大家的围观。受 InfoQ 中文站编辑之托，特作此文，一来作为过去一年的总结，二来希望对 NoSQL 系统在国内的发展和推 广尽绵薄之力。

NoSQL 背后的两种模式

NoSQL 其实并不是什么妖魔鬼怪，相反，NoSQL 的真谛其实应该是 Not Only SQL，其产生背景是在数据量和访问量逐渐增大的情况下下，人为地去添加机器或者切分数据到不同的机器，变得越来越困难，人力成本越来越高，于是便开始有了这样的项目，它们的本意是提高数据存储的自动化程度，减少人为干预的时间，让负载更加均匀等。在国际上，真正的代表之作有来自 Google 的 BigTable 和 Amazon 的 Dynamo，他们分别使用了不同的基本原理。

MapReduce

这是历史最久的一种模型，典型的代表是 BigTable。Map 表示映射，Reduce 表示化简。MapReduce 通过把对数据集的大规模操作分发给网络上的每个节点实现可靠性（Map）；每个节点会周期性地把完成的工作和状态的更新报告回来（Reduce）。大多数分布式运算可以抽象为 MapReduce 操作。Map 是把输入 Input 分解成中间的 Key/Value 对，Reduce 把 Key/Value 合成最终输出 Output。这两个函数由程序员提供给系统，下层设施把 Map 和 Reduce 操作分布在集群上运行。

Dynamo

这里我把 Dynamo 专门归纳成为了一种，其原因是它与 MapReduce 有很大的不同，自成一派。先说一下历史，Amazon 于 2006 年推出了自己的云存储服务 S3，2007 年其 CTO 公布了 S3 的设计方案，从此江湖中就不再太平了，开源项目一个个如雨后春笋般地出现了。比较

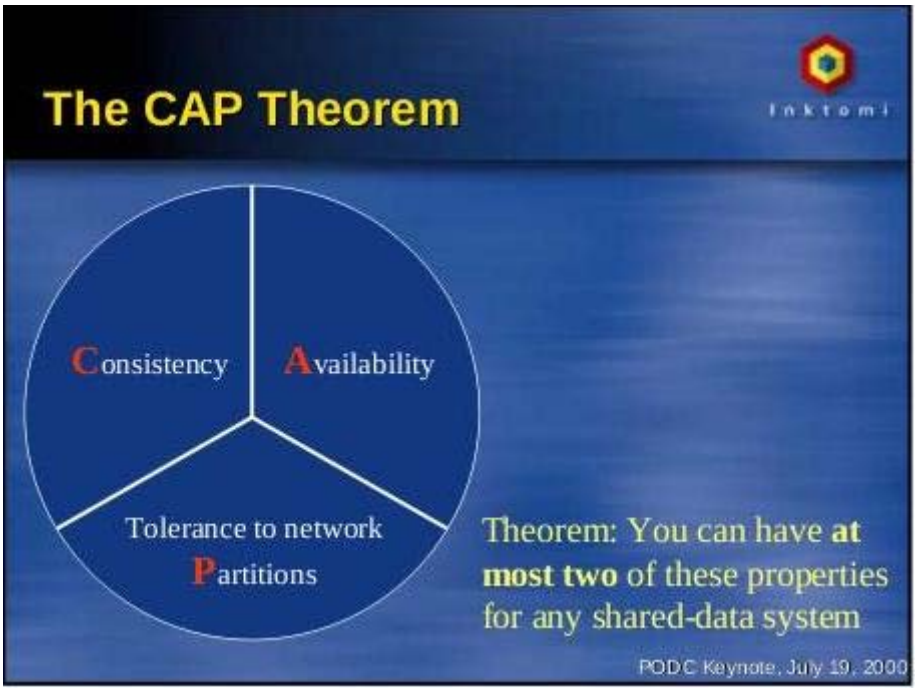
常见的有 Facebook 开发的 Cassandra（如果没有记错，在去年浏览他们项目网页的时候，上面还写着他们之中的一个开发人员是 Dynamo 的设计人员，现在风头紧，去掉了），还有 Linkedin 的 voldemort，而在国内话，有豆瓣网的 beansDB，人人网的 nuclear 等等。这里我主要讨论的也是 Dynamo 的方案细节。

入门基础

Dynamo 的意思是发电机，顾名思义，这一整套的方案都像发电机一样，源源不断地提供服务，永不间断。以下内容看上去有点教条，但基本上如果你要理解原理，这每一项都是必须知道的。

CAP 原则

先来看历史，Eric A. Brewer 教授，Inktomi 公司的创始人，也是 berkeley 大学的计算机教授，Inktomi 是雅虎搜索现在的台端技术核心支持。最主要的是，他们 （Inktomi 公司）在最早的时间里，开始研究分布计算。CAP 原则的提出，可以追溯到 2000 年的时候（可以想象有多么早！），Brewer 教授在一次谈话中，基于他运作 Inktomi 以及在伯克利大学里的经验，总结出了 CAP 原则（文末参考资料中有其演讲资料链接）。图一是来自 Brewer 教授当年所画的图：



图一：CAP 原则当年的 PPT

Consistency（一致性）：即数据一致性，简单的说，就是数据复制到了 N 台机器，如果有更新，要 N 机器的数据是一起更新的。

Availability（可用性）：好的响应性能，此项意思主要就是速度。

Partition tolerance（分区容错性）：这里是说好的分区方法，体现具体一点，简单地可理解为是节点的可扩展性。

定理：任何分布式系统只可同时满足二点，没法三者兼顾。

忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。

DHT——分布式哈希表

DHT（Distributed Hash Table，分布式哈希表），它是一种分布式存储寻址方法的统称。就像普通的哈希表，里面保存了 key 与 value 的对应关系，一般都能根据一个 key 去对应到相应的节点，从而得到相对应的 value。

这里随带一提，在 DHT 算法中，一致性哈希作为第一个实用的算法，在大多数系统中都使用了它。一致性哈希基本解决了在 P2P 环境中最为关键的问题 ——如何在动态的网络拓扑中分布存储和路由。每个节点仅需维护少量相邻节点的信息，并且在节点加入/退出系统时，仅有相关的少量节点参与到拓扑的维护中。至于一致性哈希的细节就不在这里详细说了，要指明的一点是，在 Dynamo 的数据分区方式之后，其实内部已然是一个对一致性哈希的改造了。

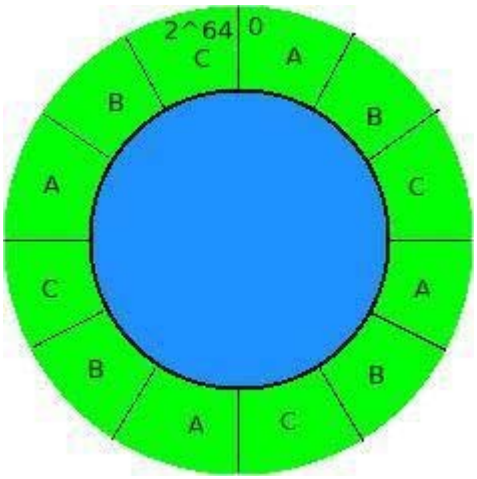
进入 Dynamo 的世界

有了上面一章里的两个基础介绍之后，我们开始进入 Dynamo 的世界。

Dynamo 的数据分区与作用

在 Dynamo 的实现中提到一个关键的东西，就是数据分区。 假设我们的数据的 key 的范围是 0 到 2 的 64 次方（不用怀疑你的数据量会超过它，正常甚至变态情况下你都是超不过的，甚至像伏地魔等其他类 Dynamo 系统是使用的 2 的 32 次方），然后设置一个常数，比如说 1000，将我们的 key 的范围分成 1000 份。然后再将这 1000 份 key 的范围均匀分配到所有的节点（s 个节点），这样每个节点负责的分区数就是 1000/s 份分区。

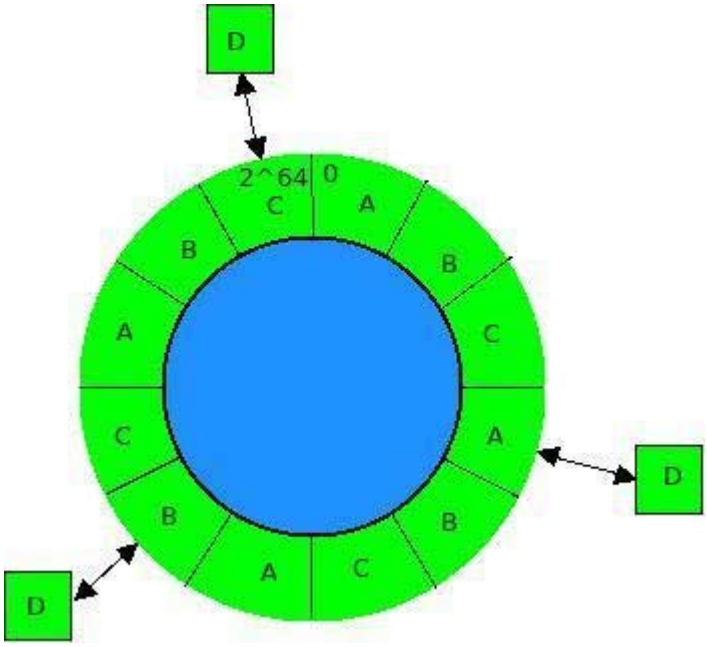
如图二，假设我们有 A、B、C 三台机器，然后将我们的分区定义了 12 个。



图二：三个节点分 12 个区的数据的情况

因为数据是均匀离散到这个环上的（有人开始会认为数据的 key 是从 1、2、3、4.....这样子一直下去的，其实不是的，哈希计算出来的值，都是一个离散的结果），所以我们每个分区的数据量是大致相等的。从图上我们可以得出，每台机器都分到了三个分区里的数据，并且因为分区是均匀的，在分区数量是相当大的时候，数据的分布会更加的均匀，与此同时，负载也被均匀地分开了（当然了，如果硬要说你的负载还是只集中在一个分区里，那就不是在这里要讨论的问题了，有可能是你的哈希函数是不是有什么样的问题了）。

为什么要进行这样的分布呢，分布的好处在于，在有新机器加入的时候，只需要替换原有分区即可，如图三所示：



图三：加入一个新的节点 D 的情况

同样是图二里的情况，12 个分区分到 ABC 三个节点，图三中就是再进入了一个新的节点 D，从图上的重新分布情况可以得出，所有节点里只需要转移四分之一的数据到新来的节点即可，同时，新节点的负载也伴随分区的转移而转移了（这里的 12 个分区太少了，如果是 1200 个分区甚至是 12000 个分区的话，这个结论就是正确的了，12 个分区只为演示用）。

从 **Dynamo** 的 **NRW** 看 **CAP** 法则

在 Dynamo 系统中，第一次提出来了 NRW 的方法。

N：复制的次数；

R：读数据的最小节点数；

W：写成功的最小分区数。

这三个数的具体作用是用来灵活地调整 Dynamo 系统的可用性与一致性。

举个例子来说，如果 R=1 的话，表示最少只需要去一个节点读数据即可，读到即返回，这时是可用性是很高的，但并不能保证数据的一致性，如果说 W 同时为 1 的 话，那可用性更新是最高的一种情况，但这时完全不能保障数据的一致性，因为在可供复制的 N 个节点里，只需要写成功一次就返回了，也就意味着，有可能在读的这一次并没有真正读到需要的数据（一致性相当的不好）。如果 W=R=N=3 的话，也就是说，每次写的时候，都保证所有要复制的点都写成功，读的时候也是都读到，这样子读出来的数据一定是正确的，但是其性能大打折扣，也就是说，数据的一致性非常的高，但系统的可用性却非常低了。如果 R + W > N 能够保证我们“读我们所写”，Dynamo 推荐使用 322 的组合。

Dynamo 系统的数据分区让整个网络的可扩展性其实是一个固定值（你分了多少区，实际上网络里扩展节点的上限就是这个数），通过 NRW 来达到另外两个方 向上的调整。

Dynamo 的一些增加可用性的补救

针对一些经常可能出现的问题，Dynamo 还提供了一些解决的方法。

第一个是 hinted handoff 数据的加入：在一个节点出现临时性故障时，数据会自动进入列表中的下一个节点进行写操作，并标记为 handoff 数据，在收到通知需要原节点恢复时重新把数据推回去。这能使系统的写入成功大大提升。

第二个是向量时钟来做版本控制：用一个向量（比如说[a,1]表示这个数据在 a 节点第一次写入）来标记数据的版本，这样在有版本冲突的时候，可以追溯到出现问题的地方。这可以使数据的最终一致成为可能。（Cassandra 未用 vector clock，而只用 client timestamps 也达到了同样效果。）

第三个是 Merkle tree 来提速数据变动时的查找：使用 Merkle tree 为数据建立索引，只要任意数据有变动，都将快速反馈出来。

第四个是 Gossip 协议：一种通讯协议，目标是让节点与节点之间通信，省略中心节点的存在，使网络达到去中心化。提高系统的可用性。

后记

Dynamo 的理论对 CAP 原则里的可扩展性做到了很方便的实现，通过创造性的 NRW 来平衡系统的可用性和一致性，增加了系统在实际情况下遇到问题的可选择方案。可以相像，在 NoSQL 的道路上，这只是个开端，在分布式计算的道路上，已经是 MapReduce 之后的再次革命。

关于作者

54chen（陈臻），人人网分布式存储研究人员，业余时间混迹于各技术组织且乐此不疲。目前关注实施 PHP 培训。对 flex 等前端技术有一点研究。个人技术站点：http://www.54chen.com/ 。可以通过电子邮件 czhttp@gmail.com 联系到他。

参考资料：

- LinkedIn 公司的伏地魔项目主页
- FaceBook 公司的 cassandra 项目主页
- Dyanmo 最原始的一个 pdf 文档
- Brewer 教授的主页
- 伏地魔项目的中文文档、提出 CAP 原则的文档、DHT 的介绍、一致性哈希的介绍

原文链接：http://www.infoq.com/cn/articles/nosql-dynamo

特别专题

图形数据库、NOSQL 和 Neo4j

作者 Peter Neubauer 译者 胡键

在众多不同的数据模型里，关系数据模型自 80 年代就处于统治地位，而且有不少实现，如 Oracle、MySQL 和 MSSQL，它们也被称为关系数据库管理系统（RDBMS）。然而，最近随着关系数据库使用案例的不断增加，一些问题也暴露了出来，这主要是因为两个原因：数据建模中的一些缺陷和问题，以及在大数据量和多服务器之上进行水平伸缩的限制。两个趋势让这些问题引起了全球软件社区的重视：

1. 用户、系统和传感器产生的数据量呈指数增长，其增长速度因大部分数据量集中在象 Amazon、Google 和其他云服务这样的分布式系统上而进一步加快。
2. 数据内部依赖和复杂度的增加，这一问题因互联网、Web2.0、社交网络，以及对大量不同系统的数据源开放和标准化的访问而加剧。

在应对这些趋势时，关系数据库产生了更多的问题。这导致大量解决这些问题某些特定方面的不同技术的出现，它们可以与现有 RDBMS 相互配合或代替它们 - 亦被称为混合持久化（Polyglot Persistence）。数据库替代品并不是新鲜事物，它们已经以对象数据库（OODBMS）、层次数据库（如 LDAP）等形式存在很长时间了。但是，过去几年间，出现了大量新项目，它们被统称为 NOSQL 数据库（NOSQL-databases）

本文旨在介绍图形数据库（Graph Database）在 NOSQL 运动里的地位，第二部分则是对 Neo4j（一种基于 Java 的图形数据库）的简介。

NOSQL 环境

NOSQL（Not Only SQL，不限于 SQL）是一类范围非常广泛的持久化解决方案，它们不遵循关系数据库模型，也不使用 SQL 作为查询语言。

简单地讲，NOSQL 数据库可以按照它们的数据模型分成 4 类：

1. 键-值存储库（Key-Value-stores）

2. BigTable 实现（BigTable-implementations）
3. 文档库（Document-stores）
4. 图形数据库（**Graph Database**）

就 Voldemort 或 Tokyo Cabinet 这类键/值系统而言，最小的建模单元是键-值对。对 BigTable 的克隆品来讲，最小建模单元则是包含不同个数属性的元组，至于象 CouchDB 和 MongoDB 这样的文档库，最小单元是文档。图形数据库则干脆把整个数据集建模成一个大型稠密的网络结构。

在此，让我们深入检阅 NOSQL 数据库的两个有意思的方面：伸缩性和复杂度。

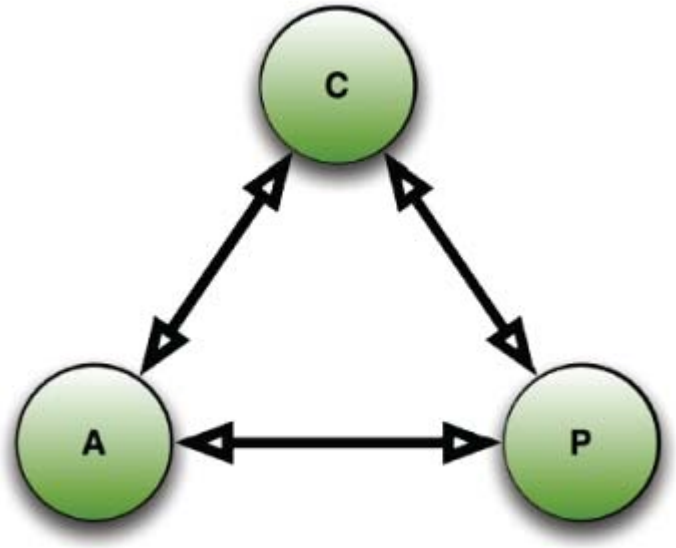
1. 伸缩性

CAP: ACID vs. BASE

为了保证数据完整性，大多数经典数据库系统都是以事务为基础的。这全方位保证了数据管理中数据的一致性。这些事务特性也被称为 **ACID**（**A** 代表原子性、**C** 表示一致性、**I** 是隔离性、**D** 则为持久性）。然而，ACID 兼容系统的向外扩展已经表现为一个问题。在分布式系统中，高可用性不同方面之间产生的冲突没有完全得到解决 - 亦称 **CAP** 法则：

- 强一致性（**C**）：所有客户端看到的数据是同一个版本，即使是数据集发生了更新 - 如利用两阶段提交协议（XA 事务），和 ACID，
- 高可用性（**A**）：所有客户端总能找到所请求数据的至少一个版本，即使集群中某些机器已经宕机，
- 分区容忍性（**P**）：整个系统保持自己的特征，即使是被部署到不同服务器上的时候，这对客户端来讲是透明的。

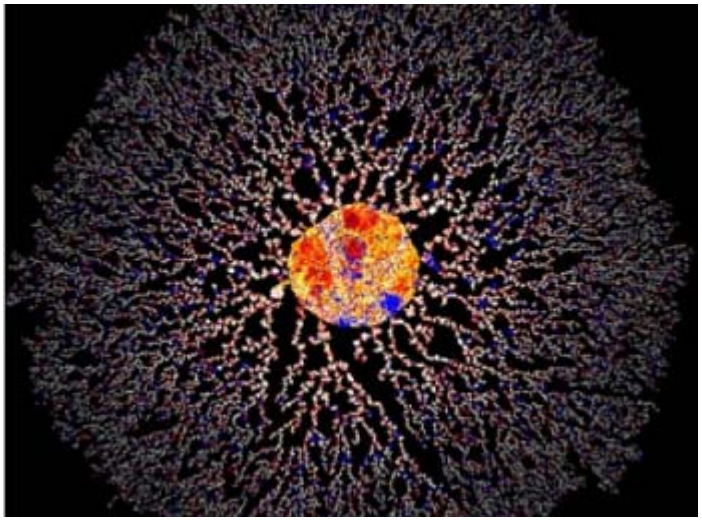
CAP 法则假定向外扩展的 3 个不同方面中只有两个可以同时完全实现。



为了能处理大型分布式系统，让我们深入了解所采用的不同 CAP 特征。

很多 NOSQL 数据库首先已经放宽了对于一致性（**C**）的要求，以期得到更好的可用性（**A**）和分区容忍性（**P**）。这产生了被称为 BASE（基本（**B**）可用性（**A**）、软状态（**S**）、最终一致性（**E**））的系统。它们没有经典意义上的事务，并且在数据模型上引入了约束，以支持更好的分区模式（如 Dynamo 系统等）。关于 CAP、ACID 和 BASE 的更深入讨论可以在这篇介绍里找到。

2. 复杂度



蛋白质同源网络（**Protein Homology Network**），感谢 **Alex Adai**：细胞和分子生物学院 - 德州大学

数据和系统的互联性增加产生了一种无法用简单明了或领域无关（domain-independent）方式进行伸缩和自动分区的稠密数据集，甚至连 Todd Hoff 也提到了这一问题。关于大型复杂数据集的可视化内容可以访问可视化复杂度（Visual Complexity）。

关系模型

在把关系数据模型扔进故纸堆之前，我们不应该忘记关系数据库系统成功的一个原因是遵照 E.F. Codd 的想法，关系数据模型通过规范化的手段原则上能够建模任何数据结构且没有信息冗余和丢失。建模完成之后，就可以使用 SQL 以一种非常强大的方式插入、修改和查询数据。甚至有些数据库，为了插入速度或针对不同使用情况（如 OLTP、OLAP、Web 应用或报表）的多维查询（星形模式），对模式实现了优化。

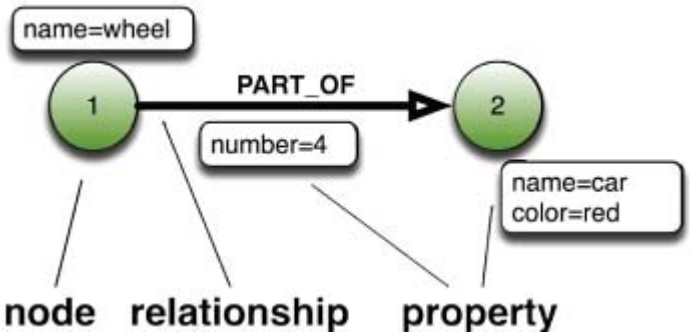
这只是理论。然而在实践中，RDBM 遇到了前面提到的 CAP 问题的限制，以及由高性能查询实现而产生的问题：联结大量表、深度嵌套的 SQL 查询。其他问题包括伸缩性、随时间的模式演变，树形结构的建模，半结构化数据，层级和网络等。

关系模型也很难适应当前软件开发的方法，如面向对象和动态语言，这被称为对象-关系阻抗失配。由此，象 Java 的 Hibernate 这 样的 ORM 层被开发了出来，而且被应用到这种混合环境里。它们固然简化了把对象模型映射到关系数据模型的任务，但是没有优化查询的性能。尤其是半结构化数 据往往被建模成具有许多列的大型表，其中很多行的许多列是空的（稀疏表），这导致了拙劣的性能。甚至作为替代方法，把这些结构建模成大量的联结表，也有问 题。因为 RDBMS 中的联结是一种非常昂贵的集合操作。

图形是关系规范化的一种替代技术

看看领域模型在数据结构上的方案，有两个主流学派 - RDBMS 采用的关系方法和图 - 即网络结构，如语义网用到的。

尽管图结构在理论上甚至可以用 RDBMS 规范化，但由于关系数据库的实现特点，对于象文件树这样的递归结构和象社交图这样的网络结构有严重的查询性能影 响。网络关系上的每次操作都会导致 RDBMS 上的一次"联结"操作，以两个表的主键集合间的集合操作来实现 ，这种操作不仅缓慢并且无法随着这些表中元组数量的增加而伸缩。



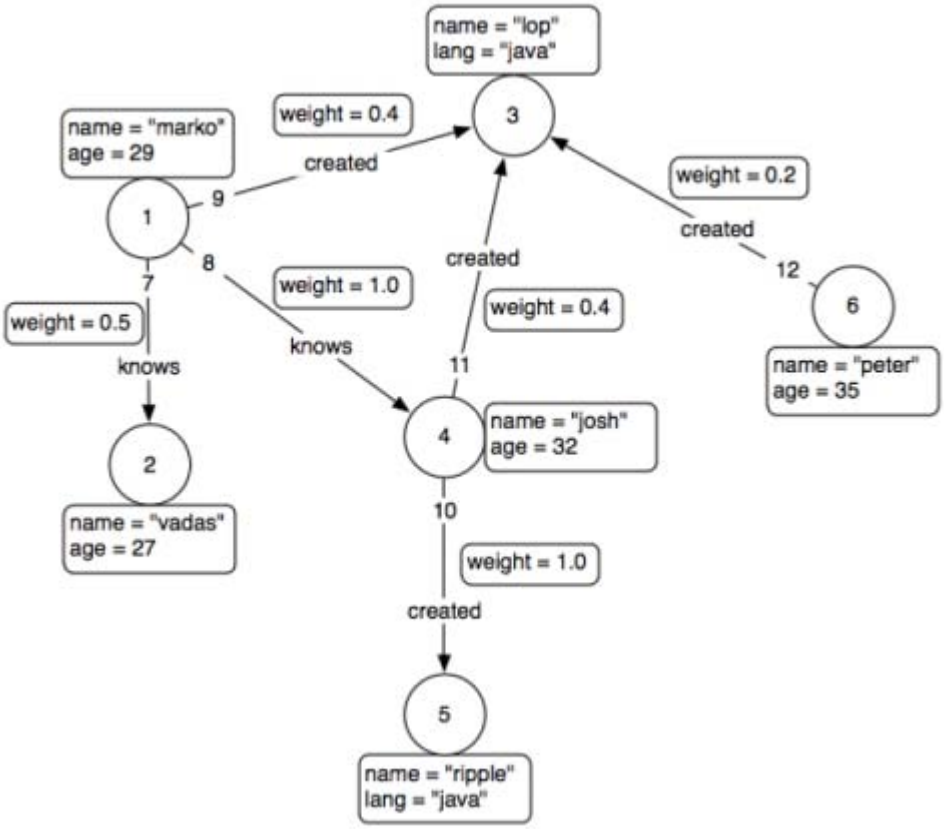
属性图形（**Property Graph**）的基本术语

在图的领域，并没有一套被广泛接受的术语，存在着很多不同类型的图模型。但是，有人致力于创建一种属性图形模型（Property Graph Model），以期统一大多数不同的图实现。按照该模型，属性图里信息的建模使用 3 种构造单元：

- 节点（即顶点）
- 关系（即边） - 具有方向和类型（标记和标向）
- 节点和关系上面的属性（即特性）

更特殊的是，这个模型是一个被标记和标向的属性多重图（multigraph）。被标记的图每条边都有一个标签，它被用来作为那条边的类型。有向图允许边有一个固定的方向，从末或源节点到首或目标节点。属性图允许每个节点和边有一组可变的属性列表，其中的属性是关联某个名字的值，简化了图形结构。多重图允许两个节点之间存在多条边。这意味着两个节点可以由不同边连接多次，即使两条边有相同的尾、头和标记。

下图显示了一个被标记的小型属性图。

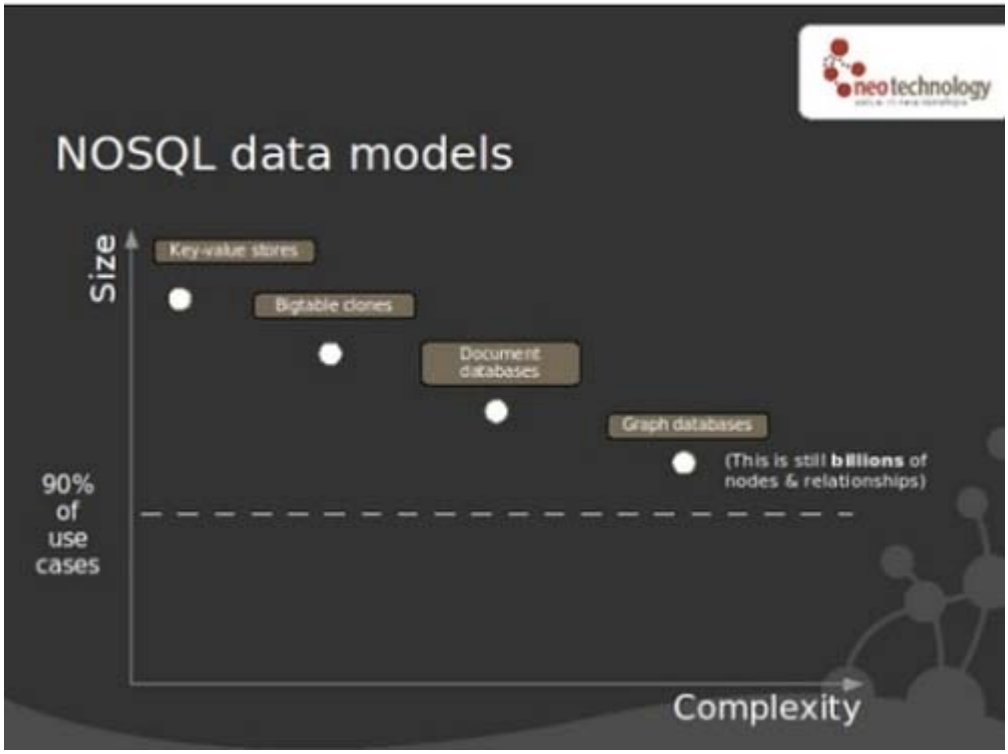


TinkerPop 有关的小型人员图

图论的巨大用途被得到了认可，它跟不同领域的很多问题都有关联。最常用的图论算法包括各种类型的最短路径计算、测地线（Geodesic Path）、 集中度测量（如 PageRank、特征向量集中度、亲密度、关系度、HITS 等）。然而，在很多情况下，这些算法的应用仅限制于研究，因为实际中没有任何 可用于产品环境下的高性能图形数据库实现。幸运的是，近些年情况有所改观。有几个项目已经被开发出来，而且目标直指 24/7 的产品环境：

- Neo4j - 开源的 Java 属性图形模型
- AllegroGraph，闭源，RDF-QuadStore
- Sones - 闭源，关注于.NET
- Virtuoso - 闭源，关注于 RDF
- yergraphDB - 开源的 Java 超图模型
- Others like InfoGrid、Filament、FlockDB 等。

下图展示了在复杂度和伸缩性方面背景下的主要 NOSQL 分类的位置。



关于“规模扩展和复杂度扩展的比较”的更多内容，请阅读 Emil Eifrem 的博文。

Neo4j - 基于 Java 的图形数据库

Neo4j 是一个用 Java 实现、完全兼容 ACID 的图形数据库。数据以一种 针对图形网络进行过优化的格式保存在磁盘上。Neo4j 的内核是一种极快的图形引擎，具有数据库产品期望的所有特性，如恢复、两阶段提交、符合 XA 等。自 2003 年起，Neo4j 就已经被作为 24/7 的产品使用。该项目刚刚发布了 1.0 版 - 关于伸缩性和社区测试的一个主要里程碑。通过联机备份实现的高可用性和主从复制目前处于测试阶段，预计在下一版本中发布。Neo4j 既可作为无需任何管理开销的内嵌数据库使用 ,也可以作为单独的服务器使用 ,在这种使用场景下，它提供了广泛使用的 REST 接口，能够方便地集成到基于 PHP、.NET 和 JavaScript 的环境里。但本文的重点主要在于讨论 Neo4j 的直接使用。

开发者可以通过 Java-API 直接与图形模型交互，这个 API 暴露了非常灵活的数据结构。至于象 JRuby/Ruby、Scala、Python、Clojure 等其他语言，社区也贡献了优秀的绑定库。Neo4j 的典型数据特征：

- 数据结构不是必须的，甚至可以完全没有，这可以简化模式变更和延迟数据迁移。
- 可以方便建模常见的复杂领域数据集，如 CMS 里的访问控制可被建模成细粒度的访问控制表，类对象数据库的用例、TripleStores 以及其他例子。

- 典型使用的领域如语义网和 RDF、LinkedData、GIS、基因分析、社交网络数据建模、深度推荐算法以及其他领域。

甚至“传统”RDBMS 应用往往也会包含一些具有挑战性、非常适合用图来处理的数据集，如文件夹结构、产品配置、产品组装和分类、媒体元数据、金融领域的语义交易和欺诈检测等

围绕内核，Neo4j 提供了一组可选的组件。其中有支持通过元模型构造图形结构、SAIL - 一种 SparQL 兼容的 RDF TripleStore 实现或一组公共图形算法的实现。

要是你想将 Neo4j 作为单独的服务器运行，还可以找到 REST 包装器。这非常适合使用 LAMP 软件搭建的架构。通过 memcached、e-tag 和基于 Apache 的缓存和 Web 层，REST 甚至简化了大规模读负荷的伸缩。

高性能？

要给出确切的性能基准数据很难，因为它们跟底层的硬件、使用的数据集和其他因素关联很大。自适应规模的 Neo4j 无需任何额外的工作便可以处理包含数十亿节点、关系和属性的图。它的读性能可以很轻松地实现每毫秒（大约每秒 1-2 百万遍历步骤）遍历 2000 关系，这完全是事务性的，每个线程都有热缓存。使用最短路径计算，Neo4j 在处理包含数千个节点的小型图时，甚至比 MySQL 快 1000 倍，随着图规模的增加，差距也越来越大。

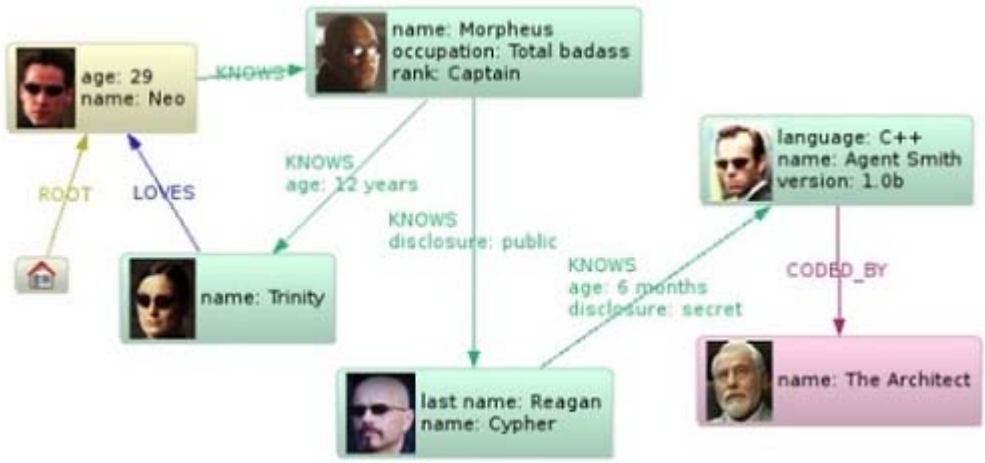
这其中的原因在于，在 Neo4j 里，图遍历执行的速度是常数，跟图的规模大小无关。不象在 **RDBMS** 里常见的联结操作那样，这里不涉及降低性能的集合操作。Neo4j 以一种延迟风格遍历图 - 节点和关系只有在结果迭代器需要访问它们的时候才会被遍历并返回，对于大规模深度遍历而言，这极大地提高了性能。

写速度跟文件系统的查找时间和硬件有很大关系。Ext3 文件系统和 SSD 磁盘是不错的组合，这会导致每秒大约 100,000 写事务操作。

示例 - 黑客帝国

图

前面已经说过，社交网络只是代表了图形数据库应用的冰山一角，但用它们来作为例子可以让人很容易理解。为了阐述 Neo4j 的基本功能，下面这个小型图来自黑客帝国这部电影。该图是用 Neo4j 的 Neoclipse 产生的，该插件基于 Eclipse RCP：



这个图链接到一个已知的引用节点（id=0），这是为了方便的从一个已知起点找到条路进入这个网络。这个节点不是必须的，但实践证明它非常有用。

Java 的实现看上去大概是这个样子：

在“target/neo”目录创建一个新的图形数据库

```
EmbeddedGraphDatabase graphdb = new EmbeddedGraphDatabase("target/neo");
```

关系类型可以动态创建：

```
RelationshipType KNOWS = DynamicRelationshipType.withName("KNOWS");
```

或通过类型安全的 Java Enum：

```
enum Relationships implements RelationshipType { KNOWS, INLOVE, HAS_CODED,
MATRIX }
```

现在，创建 2 个节点，给每个节点加上“name”属性。接着，把两个节点用一个“KNOWS”关系联系起来：

```
Node neo = graphdb.createNode();
node.setProperty("name", "Neo");
Node morpheus = graphdb.createNode();
morpheus.setProperty("name", "Morpheus");
neo.createRelationshipTo(morpheus, KNOWS);
```

任何修改图或需要数据隔离级别的操作要包在事务中，这样可以利用内置的回滚和恢复功能：

```
Transaction tx = graphdb.beginTx();
try {
    Node neo = graphdb.createNode();
    ...
    tx.success();
} catch (Exception e) {
    tx.failure();
} finally {
```

```
    tx.finish();
}
```

创建“黑客帝国”图的完整代码：

```
graphdb = new EmbeddedGraphDatabase("target/neo4j");
index = new LuceneIndexService(graphdb);
Transaction tx = graphdb.beginTx();
try {
    Node root = graphdb.getReferenceNode();
    // we connect Neo with the root node, to gain an entry point to the graph
    // not neccessary but practical.
    neo = createAndConnectNode("Neo", root, MATRIX);
    Node morpheus = createAndConnectNode("Morpheus", neo, KNOWS);
    Node cypher = createAndConnectNode("Cypher", morpheus, KNOWS);
    Node trinity = createAndConnectNode("Trinity", morpheus, KNOWS);
    Node agentSmith = createAndConnectNode("Agent Smith", cypher, KNOWS);
    architect = createAndConnectNode("The Architect", agentSmith, HAS_CODED);
    // Trinity loves Neo. But he doesn't know.
    trinity.createRelationshipTo(neo, LOVES);
    tx.success();
} catch (Exception e) {
    tx.failure();
} finally {
    tx.finish();
}
```

以及创建节点和关系的成员函数

```
private Node createAndConnectNode(String name, Node otherNode,
RelationshipType relationshipType) {
    Node node = graphdb.createNode();
    node.setProperty("name", name);
    node.createRelationshipTo(otherNode, relationshipType);
    index.index(node, "name", name);
    return node;
}
```

谁是 **Neo** 的朋友

Neo4j 的 API 有一组面向 Java 集合的方法可轻易地完成查询。这里，只消看看“Neo”节点的关系便足以找出他的朋友：

```
for (Relationship rel : neo.getRelationships(KNOWS)) {
    Node friend = rel.getOtherNode(neo);
    System.out.println(friend.getProperty("name"));
}
returns "Morpheus" as the only friend.
```

但是，Neo4j 的真正威力源自 Traverser-API 的 使用，它可以完成非常复杂的遍历描述和过滤器。它由 Traverser 和 ReturnableEvaluator 组成，前者计算 StopEvaluator 来获知何时停止，后者则用于在结果中包含哪些节点。此外，你还可以指定要遍历关系的类型和方向。Traverser

实现了 Java 的 Iterator 接口，负责延迟加载和遍历整个图，在节点被首次要求访问（如 for{...}循环）时进行。它还内置了一些常用的 Evaluator 和缺省值：

```
Traverser friends = neo.traverse(Order.BREADTH_FIRST,
StopEvaluator.DEPTH_ONE,
ReturnableEvaluator.ALL_BUT_START_NODE, KNOWS, Direction.BOTH);
for (Node friend : friends) {
    System.out.println(friend.getProperty("name"));
}
```

我们在继续访问更深一级的节点之前首先从起点访问处于同一深度的所有节点（Order.BREADTH_FIRST），在深度为 1 的一次遍历后停止（StopEvaluator.DEPTH_ONE），然后返回除了起点（"Neo"）之外的所有节点（ReturnableEvaluator.ALL_BUT_START_NODE）。我们在两个方向只遍历类型为 KNOWS 的关系。这个遍历器再次返回 Morpheus 是 Neo 仅有的直接朋友。

朋友的朋友？

为了调查谁是 Neo 朋友的朋友，KNOWS 网络需要再进行深度为 2 的步骤，由 Neo 开始，返回 Trinity 和 Cypher。实际编程中，这可以通过调整我们的 Traverser 的 StopEvaluator，限制遍历深度为 2 来实现：

```
StopEvaluator twoSteps = new StopEvaluator() {
    @Override
    public boolean isStopNode(TraversalPosition position) {
        return position.depth() == 2;
    }
};
```

还要定制 ReturnableEvaluator，只返回在深度 2 找到的节点：

```
ReturnableEvaluator nodesAtDepthTwo = new ReturnableEvaluator() {
    @Override
    public boolean isReturnableNode(TraversalPosition position) {
        return position.depth() == 2;
    }
};
```

现在“朋友的朋友”遍历器就成了：

```
Traverser friendsOfFriends = neo.traverse(Order.BREADTH_FIRST,

    twoSteps, nodesAtDepthTwo, KNOWS, Direction.BOTH);
for (Node friend : friendsOfFriends) {
    System.out.println(friend.getProperty("name"));
}
```

它的结果是 Cypher 和 Trinity。

谁在恋爱？

另一个有趣的问题是，这个图上是否有人正在热恋，比方说从架构师（Architect）开始。

这次，整个图需要沿着由架构师（假定他的节点 ID 是已知的，但要到很晚才知道）开始的任何关系开始检查，返回拥有向外 LOVE 关系的节点。一个定制的 ReturnableEvaluator 可以完成这件事：

```
ReturnableEvaluator findLove = new ReturnableEvaluator() {
@Override
    public boolean isReturnableNode(TraversalPosition position) {
        return                  position.currentNode().hasRelationship(LOVES,
Direction.OUTGOING);
    }
};
```

为了遍历所有关系，需要知道整个图的所有关系类型：

```
List<Object> types = new ArrayList<Object>();
// we have to consider all relationship types of the whole graph
// (in both directions)
for(RelationshipType type : graphdb.getRelationshipTypes()) {
    types.add(type);
    types.add(Direction.BOTH);
}
//let's go!
Traverser inLove = architect.traverse(Order.BREADTH_FIRST,
StopEvaluator.END_OF_GRAPH, findLove, types.toArray());
for (Node lover : inLove) {
    System.out.println(lover.getProperty("name"));
}
```

上述代码的返回结果只有一个节点：Trinity，因为我们只返回拥有向外 LOVE 关系的节点。

给图建立索引

尽管沿着所有关系的遍历操作是 Neo4j 的亮点之一，但也需要在整个图之上进行面向集合的操作。所有节点属性的全文检索就是一个典型的例子。为了不重新发明轮子，Neo4j 在这里使用了外部索引系统。针对常见的基于文本的搜索，Neo4j 已经跟 Lucene 和 Solr 进行了深度集成，在 Lucene/Solr 里增加了给具有事务语义的任意节点属性创建索引的功能。

在黑客帝国的例子里，如给“name”属性创建索引：

```
GraphDatabaseService graphDb = // a GraphDatabaseService instance

IndexService index = new LuceneIndexService( graphDb );

//create a new node and index the "name" property
Node neo = graphDb.createNode();
```

```
neo.setProperty( "name", "Neo" );
index.index( neo, "name", neo.getProperty( "name" ) );

//search for the first node with "name=Neo"
Node node = index.getSingleNode( "name", "Neo" );
```

Lucene 是图的外部索引的一个例子。但是，作为一个快速图形引擎，有大量的策略来构建图本身内部的索引结构，针对特殊数据集和领域缩短遍历模式。例如，有针对一维数据的 timeline 和 B 树，给二维数据（在空间和 GIS 社区非常普遍）建立索引的 RTrees 和 QuadTrees 等。另一个常见的有用模式是将重要的子图直接连接到根节点，以创建重要开始节点的快捷路径。

Java 太麻烦了。拜托，有没有简短点的？

Neo4j 甚至还提供了一组优秀的语言绑定来简化图结构操作。这个例子使用了优秀的 Neo4j-JRuby-绑定，它极大地减少了整个代码量：

先安装 neo4j gem

```
>gem install neo4j
```

这时，整个黑客帝国的图和前面提到的查询，用 JRuby 代码编写就成了这个样子：

```
require "rubygems"

require "neo4j"

class Person
  include Neo4j::NodeMixin

  #the properties on the nodes
  property :name


  #the relationship types
  has_n :friends

  # Lucene index for some properties
  index :name
end


#the players

neo      = Person.new :name => 'Neo'
morpheus  = Person.new :name => 'Morpheus'
trinity   = Person.new :name => 'Trinity'
cypher    = Person.new :name => 'Cypher'
smith     = Person.new :name => 'Agent Smith'
architect = Person.new :name => 'Architect'
```

```
#the connections

cypher.friends << morpheus
cypher.friends << smith
neo.friends << morpheus
morpheus.friends << trinity
trinity.rels.outgoing(:loves) << neo

architect.rels.outgoing(:has_coded) << smith


#Neos friends

neo.friends.each { |n| puts n }


#Neos friends-of-friends

neo.friends.depth(2).each { |n| puts n }


#Who is in love?
architect.traverse.both(:friends, :has_coded, :loves).depth(:all).filter do
  outgoing(:loves).to_a.size > 0
end.each do |n|
 puts 'In love: ' + n.name
end
```

图编程语言 – **Gremlin**

直到最近，还没有任何查询语言涉及大型的图领域和图相关项目。在语义网/RDF 领域，有 SPARQL，受 SQL 启发的查询语言，专注于描述用来匹配元组集合的样本图。但是，大量的图并不兼容 RDF，而且采用不同或更侧重于更实用的方式进行数据建模，象本文中的黑客帝国例子，以及其他领域特定的数据集。其他查询语言都是面向 JSON 的，如 MQL，一种用于 Freebase 的查询语言。这些语言只工作于它们自己定义的数据模型，完全不支持或只非常有限地支持深度图算法和启发式分析方法，而这又是当今大型图里不可或缺的内容。

至于针对各种图数据模型（如 RDF）的更复杂有趣的查询，Gremlin - 一种面向 XPath，图灵完备的图形编程语言 - 正由 TinkerPop 团队开发，主要由 Marko A. Rodriguez 推动。借助引入属性图模型，它创造了一个针对现有大多数模型的超集，以及最小的公共操作集合。此外，

它允许连接其他图形框架（如 Gremlin 使用 JUNG），同时支持在不同的图实现上都能表达图的遍历。已支持的一组实现包括，从简单的如内存中的 TinkerGraph，到其他通过针对 AllegroGraph、Sesame 和 ThinkerPop LinkedData SAIL（最开始由 Josh Shinavier 为 Ripple 编程语言开发）的 RDF-SAIL 适配器，一直到 Neo4j。

Gremlin 的语法建立在 XPath 基础之上，这是为了可以简单地表达整个图的深度路径描述。很多简单的例子几乎就像普通的 XPath。

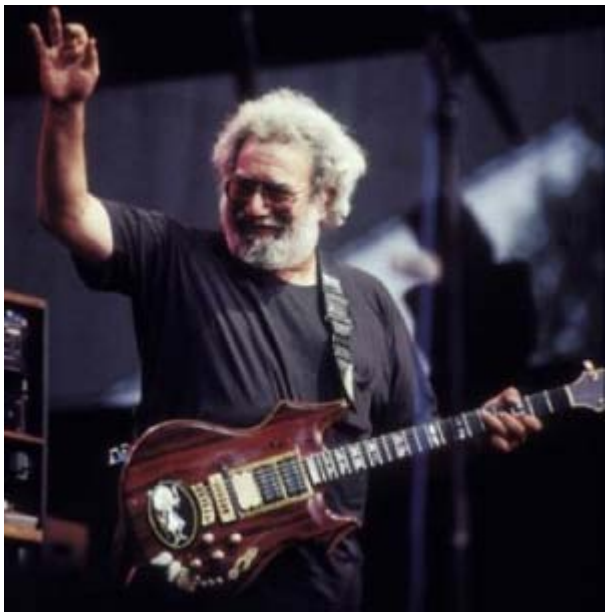
在安装 Gremlin 或在线试用之后，黑客帝国例子里的图的 Gremlin 会话大致是：

```
peterneubauer$ ~/code/gremlin/gremlin.sh


        \,,,/
        (o o)
-----oOOo-(_)-oOOo-----
gremlin> #open a new neo4j graph as the default graph ($_g)
gremlin> $_g := neo4j:open('tmp/matrix')
==>neo4jgraph[tmp/matrix]
gremlin> #the vertices
gremlin> $neo := g:add-v(g:map('name','Neo'))
==>v[1]
gremlin> $morpheus := g:add-v(g:map('name','Morpheus'))
==>v[2]
gremlin> $trinity := g:add-v(g:map('name','Trinity'))
==>v[3]
gremlin> $cypher := g:add-v(g:map('name','Cypher'))
==>v[4]
gremlin> $smith := g:add-v(g:map('name','Agent Smith'))
==>v[5]
gremlin> $architect := g:add-v(g:map('name','The Architect'))
==>v[6]
gremlin> #the edges
gremlin> g:list($cypher,$neo,$trinity)[g:add-e($morpheus,'KNOWS',.)]
==>v[4]
==>v[1]
==>v[3]
gremlin> g:add-e($cypher,'KNOWS',$smith)
==>e[3][4-KNOWS->5]
gremlin> g:add-e($trinity,'LOVES',$neo)
==>e[4][3-LOVES->1]
gremlin> g:add-e($architect,'HAS_CODED',$smith)
==>e[5][6-HAS_CODED->5]
gremlin> #go to Neo as the current root ($_) via a full-text index search
gremlin> $_ := g:key('name','Neo')
==>v[1]
gremlin> #is this Neo?
gremlin> ./@name
==>Neo
gremlin> #what links to here and from here?
gremlin> ./bothE
==>e[0][1-KNOWS->2]
==>e[4][3-LOVES->1]
```

```
gremlin> #only take the KNOWS-edges
gremlin> ./bothE[@label='KNOWS']
==>e[0][1-KNOWS->2]
gremlin> #Neo's friend's names
gremlin> ./bothE[@label='KNOWS']/inV/@name
==>Morpheus
gremlin>

gremlin> #Neo's Friends of friends, 2 steps
gremlin> repeat 2
            $_ := ./outE[@label='KNOWS']/inV
         end
==>v[4]
==>v[3]
gremlin> #What are their names?
gremlin> ./@name
==>Cypher
==>Trinity
gremlin> #every node in the whole graph with an outgoing LOVES edge
gremlin> $_g/V/outE[@label='LOVES']/../@name
==>Trinity
```

深度图算法 - 关系的价值

鉴于 Gremlin 的能力，黑客帝国的例子显得相当幼稚。更有趣的是开发和测试大型图上的算法。象特征向量集中度和 Dijkstra 这类穷举算法并不会扩展到这些图上，因为它们需要了解网络里的每个顶点。对于针对这些问题的如基于语法的随机访问器和扩散激活（Marko Rodriguez 和这里有更深入的解释）这类概念，启发式方法更合适。Google PageRank 算法就是启发式的，而且可以用 Gremlin 建模，代码如下（Greatful Dead 的歌曲、演唱会和相册图的一个例子，图从这里装入，2500 个循环，每次重复能量损失 15%）：



```
$_g := tg:open()

g:load('data/graph-example-2.xml')

$m := g:map()

$_ := g:key('type', 'song')[g:rand-nat()]

repeat 2500

  $_ := ./outE[@label='followed_by'][g:rand-nat()]/inV

  if count($_) > 0

    g:op-value('+',$m,$_[1]/@name, 1.0)

  end

  if g:rand-real() > 0.85 or count($_) = 0

    $_ := g:key('type', 'song')[g:rand-nat()]

  end

end

g:sort($m,'value',true())
```

它返回如下的歌曲权重列表：

```
==>DRUMS=44.0
==>PLAYING IN THE BAND=38.0
==>ME AND MY UNCLE=32.0
==>TRUCKING=31.0
==>CUMBERLAND BLUES=29.0
==>PROMISED LAND=29.0
==>THE MUSIC NEVER STOPPED=29.0
==>CASSIDY=26.0
==>DARK STAR=26.0
==>NOT FADE AWAY=26.0
==>CHINA CAT SUNFLOWER=25.0
==>JACK STRAW=25.0
==>TOUCH OF GREY=24.0
==>BEAT IT ON DOWN THE LINE=23.0
==>BERTHA=23.0
```

其底层图的另一个有趣的例子是 LinkedData 图，在互联网上可在线了解：LinkedData 和 DBPedia 的音乐推荐算法。

总结

就像 RDBMS 和其他持久化解决方案一样，图不是解决所有问题的银弹。数据才是最重要的东西，它是关键，然后是查询类型、要处理的操作结构，以及什么需求要考虑伸缩性和 CAP。

将采用 NOSQL 数据库的高伸缩性方面作为使用非关系解决方案的唯一考量往往是不必要的，而且也不符合预期。使用 Neo4j 和当今的硬件，大多数情况下，完全可以在单实例里容纳整个领域模型和查询数十亿领域对象。

如果那还不够，总是有办法引入针对领域优化过的数据分区概念，无需引入文档或键/值系统的硬数据建模限制。最终导致的结果是一个文档模型、或领域特定的“对象数据库”还是其他模型则取决于领域上下文和应用场景。

本文的代码可以从这里下载。

我要感谢 Michael Hunger 和 Marko Rodriguez 给我提供了优秀的反馈和有帮助的建议。

作者简介

Peter Neubauer 是 Neo Technology 的 COO ,他同时还是几个基于 Java 和图开源项目的合伙创办人 ,如 Neo4j、Gremlin、LinkedProcess、OPS4J、Qi4j。你可以通过 peter@neotechnology.com. 联系到 Peter。

原文链接： http://www.infoq.com/cn/articles/graph-nosql-neo4j

SYBASE

Sybase在线课堂：深度剖析

---Sybase ASE 15.5 的实时数据处理

这次活动我们邀请到了来自Sybase的资深的咨询顾问工程师范宏伟与IBM系统与科技事业部Power产品经理谷建一起，来为我们深度剖析Sybase ASE 15.5的实时数据处理与IBM POWER在数据管理领域的强大威力。

关键型的应用程序用户数不断增加，保持或提高应用服务水平的实时响应能力面临着更大的挑战。传统的基于一致性模型的数据库往往不堪重负，而Sybase为大家带来了最新的内存数据库技术，配合IBM强大的POWER硬件，使任务关键型的应用程序能更好地从ASE数据库中获取极致的高性能。

- 全程免费参与
- 与专业讲师零距离交流
- 参与提问更有机会获得精美礼品
- 分享知识，相聚课堂！

Sybase ASE 15.5 视频介绍 点击下载

现在就来报名！

推荐文章 | Articles

摘录与访谈：《HTML5 专业开发》

作者 Dionysios G. Synodinos 译者 徐涵

由 Peter Lubbers、Brian Albers 和 Frank Salim 编著的《HTML5 专业开发》（Pro HTML5 Programming）一书，旨在帮助开发者们构建能够挖掘当今浏览器全部能力的 HTML5 应用。

该书主要涉及领域有：

- **Communication APIs**：探讨构建实时和跨源（cross-origin）通讯的两大基础： 跨文档通讯（Cross Document Messaging）与 XMLHttpRequest Level 2。
- **Convas API**：介绍有关动态生成与渲染图形、图表、图像和动画的 API。
- **Geolocation API**：使用这个 API，用户便可共享自己的地理位置，并在 Web 应用的帮助下享用位置感知的服务（location-aware services）。
- **Web Sockets API**：定义了一个全双工传输的通讯通道，它在 Web 之上通过单个套接字进行工作。
- **Web Storage API**：探讨 HTML5 Web 存储（也就是 DOMStorage），该 API 令在多个 Web 请求间持久化数据变得简单。
- **Web Workers API**：该 API 为 Web 应用提供了后台处理的能力。
- **HTML5** 音频与视频：这些是 HTML 里新添的元素，它们为开发者提供了一套通用的、集成的、脚本式的处理音频与视频的 API，而无需安装任何插件。

Apress 出版社特许 InfoQ 发布该书第六章《使用 HTML5 Web Sockets API》，并介绍该书作者之一接受 InfoQ 的访谈。

InfoQ 与该作者进行了一系列问答：

InfoQ：你好，**Peter**！能否简单介绍一下自己，并谈谈你们为什么会写一本关于 **HTML5** 的书？

Peter 我在 Oracle 工作了大约 8 年，然后在 2008 年成为 Kaazing 的第四名员工。从 WebSocket API 与协议（作为 TCPConnection）被加入到 HTML5 规范开始，Kaazing 就一直涉入其中，并且对 HTML5 的发展起到了重要作用。早在 2008 年，我们就已经开始提供有关 WebSocket 的培训了，那时还没什么浏览器想到要支持它。我们的 WebSocket 网关能够完全（并且迅速）地模拟标准，所以你在 IE6 这种老式浏览器上也可以使用 HTML5 WebSockets。

在 Kaazing，我负责文档与培训的工作。 Kaazing 的创始人（Jonas Jacobi 与 John Fallows）是我在 Orcale 的同事，我曾帮助他们编校了 2006 年 Apress 出版的《JSF 与 Ajax》一书。正是他们建议我们三人—— Brian Albers（Kaazing 的开发副总裁）、Frank Salim（Kaazing 的“编码机器”)和我——写一本有关 HTML5 WebSockets 的书，可是我们发现市面上并没有 HTML5 的书，而且有关 HTML5 开发的资源也很少。

本书不是 HTML5 的入门书，也不是回顾 HTML 历史的书。 现在，我想我们都知道 HTML5 是个大项目，而本书会帮助你使用目前大多浏览器已采用的简单而又强大的 API。

InfoQ：**HTML5** 是什么？它要解决什么问题？你能介绍一下 **HTML5** 的各个部分以及它们之间的联系吗？

Peter：HTML5 不仅仅是 HTML 的更新换代，它已经孕育很久了。之前最新的 HTML（4.01）还是 10 年前发布的！

当我们谈及 HTML5 时，实际上它指的是包括 HTML、CSS 和 JavaScript 在内的一套技术组合。比如说，HTML5 引进了新的画布（convas）和视频元素。虽然它们自身已经非常强大，但你需要将其与 JavaScript 和 CSS 一同使用，其全部能力方能显现。

HTML5 讲究简单性。它的一个理念叫做“沿着脚印铺路（pave the cowpaths）” [译注：这里字面意思为“沿着牛走过的脚印铺路”，在这里的意思是，如果一种做法已在开发者中间被广为采用，那么与其禁止它或发明新的做法，还不如考虑接受它并采纳为标准]，这意味着它致力于让之前困难的事情变得非常简单。HTML5 还讲究原生的（native）浏览器功能。在 HTML5 里，你可以完成之前依赖于插件（如 Flash 和 Silverlight）或者必须用复杂的 JavaScript 脚本才能做到的事情。比如，HTML5 引进了原生的表单控件类型（日期、拾色器）与表格验证、原生的套接字（用于全双工通信）、原生的绘图 API（SVG、画布）以及内置的位置感知（使用 Geolocation API）。

InfoQ：目前的浏览器对各种 **HTMl5 APIs** 的支持如何？你认为未来会如何发展？

Peter：浏览器对 HTML5 功能特性的支持正在以惊人的速度迅速增加——这是十分令人振奋的。许多流行浏览器（如 Chrome、Safari、Firefox 以及 Opera）已经能够支持 75%到 90%不

等的 HTML5 特性了，而这个数字还在不断增加。

你可以通过 html5test.com 或 caniuse.com 网站来查询你所使用的浏览器支持哪些功能特性；这是检查飞速变化的 HTML5 支持情况的好地方。

即使是一直处于落后的 IE（IE6 仅支持 3%，IE8 支持 28%），在 IE9 中也“翻倍了”，支持将近 60%的 HTML5 特性。其中一些功能还有硬件优化，这将推动其他浏览器加入进来。

InfoQ：你们的书中介绍了好几种 **HTML5 APIs**。你能否举一些例子来说明，它们将如何帮助开发者构建更好的应用？

Peter：当然！Web 应用已经越来越与桌面的功能不相伯仲。 比如，在 Kaazing，一位非常有才华的同事 Frank Salim 刚刚使用 WebSockets 与画布（convas）开发了一款 VNC 客户端——这在几年前（或者几月前）是无法想象的。

大体说，HTML5 便于我们开发出更具状态性和事务性的 Web 应用。你可以使用 WebSockets 直接连接到远程服务器，你也可以使用 Geolocation API 开发出位置感知的应用，并将他们与 Map APIs 相结合。你可以使用 SVG 和画布（convas）开发出令人惊奇的动画游戏（2D；但如果你愿意构建自己的浏览器，也可以是 3D）。

各个 APIs 都很强大，你只要将他们结合起来，就可以开发出非常棒的应用。比如，HTML5 支持用新的应用缓存（Application Cache）来开发脱机应用，也支持客户端 Web 存储（甚至是持久化存储，称为 localStrorage），那么当然，你可以将脱机应用与存储漂亮地结合起来；你可以进行脱机检查，而不必依赖不保险的浏览器缓存能力（点击“返回”后，之前的页面就没了）。现在，用户可以按他们所习惯的方式来使用整个应用了：尚未发出的数据被保存在客户端，待应用检测出你在线后，数据才被发到服务端。

InfoQ:你认为目前 **HTML5** 中还缺少什么？还应该添加些什么？是否有新的 **APIs** 出现？现有 **APIs** 有修改吗？

Peter：当然，还有很多东西需要打磨，浏览器也需要关注于互操作性。我们不应忽视 HTML5 原始的设计理念。我觉得，对各种 APIs 的二进制支持（跨文档通讯与 WebSockets）是需要改善的，还有对 3D 画布的支持也需改进。此外，不同特性的无障碍性（accessibility）还需改善。

WebGL 不仅将 OpenGL 引入到 JavaScript 中，它也创造了一个通用的二进制数据 API。高速的 3D API 需要二进制缓存，所以有人提议新推出一套用于操作二进制数据的 API，称为 TypedArray。如果该提议被采纳，TypedArray 将在 HTML5 中无处不在。

InfoQ：人们如何开发 **HTML5** 应用？有哪些主要工具可用于开发、调试和测试应用？

Peter：目前还没有很多集成开发环境（IDE）支持 HTML5。DreamWeaver CS5 的 HTML5 包和 Visual Studio 都提供了语法补全支持，不过你也可以直接使用文本编辑器（如 UltraEdit）或通用的 IDE（如 Eclipse）。

如今的浏览器调试功能非常强大，浏览器厂商们都在致力于为它们的开发工具增添一些很酷的功能。比如，Google Developer Tools 与 Speed Tracer、Safari Web Inspector、Firefox Firebug 以及 Opera Dragonfly，这些都是功能强大的浏览器调试工具。如果你有一段时间未跟踪此领域的话，你会对如此多的强大功能感到非常惊讶。

我发现我的浏览器每周都有功能更新。比如，Chrome 刚刚支持对脱机应用的调试以及应用缓存的查看功能，此外，还可以通过控制台来访问 Web SQL 数据库——太强了！

InfoQ：对于准备使用 **HTML5** 进行项目开发的团队，你有什么建议？有什么特别需要注意的地方吗？有什么“最佳实践”吗？

Peter：不要相信任何炒作，然后，再买本《HTML5 专业开发》就没问题了 ;-)

说真的，现在有很多不错的资源。可以看看 Google 的 html5rocks.com 网 站。另外，Paul Irish 的 html5boilerplate 工具也很不错，它将多年经验融汇其中，并且采用了可以令你的网站与老式浏览器完美兼容的技巧与技术（一般来 说，这是个大问题）。我认为，html5boilerplate 工具的文档（注释与代码片段）里包含了目前最好的最佳实践。此外，还有很多其他好网站也可 参考。

我也在全球范围内提供 HTML5 培训（HTML5 快速追踪、Bootcamps 以及 WebSocket 课程），所以，如果需要快速入门的话，尽管联系我。

InfoQ：对于 **HTML5** 新手来说，这本书哪些部分最有价值？对于有经验的开发者呢？

Peter：HTML5 具有强大的简单性。

正像本书书名一样，《HTML5 专业开发》面向的读者群是对 HTML 和 JavaScript 有一定了解、并且希望使用新的 HTML5 API 的开发者。

不过，由于 HTML5 讲究简单性，所以对于 HTML、JavaScript 和 CSS 新手来说，HTML5 实际上要比看起来容易得多。我们希望本书能引发大家按有创意的方式来使用 HTML5 APIs。

《HTML5 专业开发》讲解了如何使用新的 HTML5 APIs，并通过实际例子展示了 HTML5 的功能特性，以及目前的浏览器支持其中哪些。我们挑选出了一些最常用的、实用而且强大的

HTML APIs 进行讲解，以便于读者快速上手。你将学习到如何使用 HTML5 的各种功能特性来开发 Web 应用，比如地理位置、Web 存储、 WebSockets、Web Workers、画布、音频和视频等等。

你会看到这些功能如何结合起来，无缝地与其他标准 Web 技术工作。我们的例子使用了 HTML5 文档，并充分利用了浏览器的基本功能。

你可以在 InfoQ 找到更多关于 HTML5 的信息。

原文链接：http://www.infoq.com/cn/articles/pro-html5-programming

相关内容：

- HTML5 Web Sockets与代理服务器交互
- Ajax应用开发：实践者指南
- 书摘和采访：Deploying HTML5
- HTML 5 案例研究：使用WebSockets、Canvas与JavaScript构建noVNC客户端

推荐文章 | Articles

加速你的 Hibernate 引擎（上）

作者 Yongjun Jiao and Stewart Clark 译者 丁雪丰

1.引言

Hibernate 是最流行的对象关系映射（ORM）引擎之一，它提供了数据持久化和查询服务。

在你的项目中引入 Hibernate 并让它跑起来是很容易的。但是，要让它跑得好却是需要很多时间和经验的。

通过我们的使用 Hibernate 3.3.1 和 Oracle 9i 的能源项目中的一些例子，本文涵盖了很多 Hibernate 调优技术。其中还提供了一些掌握 Hibernate 调优技术所必需的数据库知识。

我们假设读者对 Hibernate 有一个基本的了解。如果一个调优方法在 Hibernate 参考文档（下文简称 HRD）或其他调优文章中有详细描述，我们仅提供一个对该文档的引用并从不同角度对其做简单说明。我们关注于那些行之有效，但又缺乏文档的调优方法。

2.Hibernate 性能调优

调优是一个迭代的、持续进行的过程，涉及软件开发生命周期（SDLC）的所有阶段。在一个典型的使用 Hibernate 进行持久化的 Java EE 应用程序中，调优会涉及以下几个方面：

- 业务规则调优
- 设计调优
- Hibernate 调优
- Java GC 调优
- 应用程序容器调优
- 底层系统调优，包括数据库和 OS。

没有一套精心设计的方案就去进行以上调优是非常耗时的，而且很可能收效甚微。好的调优方法的重要部分是为调优内容划分优先级。可以用 Pareto 定律（又称“80/20 法则”）来解释这一点，即通常 80%的应用程序性能改善源自头 20%的性能问题[5]。

相比基于磁盘和网络的访问，基于内存和 CPU 的访问能提供更低的延迟和更高的吞吐量。这种基于 IO 的 Hibernate 调优与底层系统 IO 部分的调优应该优先于基于 CPU 和内存的底层系统 GC、CPU 和内存部分的调优。

> 范例 1
>
> 我们调优了一个选择电流的 HQL 查询，把它从 30 秒降到了 1 秒以内。如果我们在垃圾回收方面下功夫，可能收效甚微——也许只有几毫秒或者最多几秒，相比 HQL 的改进，GC 方面的改善可以忽略不计。

好的调优方法的另一个重要部分是决定何时优化[4]。

积极优化的提倡者主张开始时就进行调优，例如在业务规则和设计阶段，在整个 SDLC 都持续进行优化，因为他们认为后期改变业务规则和重新设计代价太大。

另一派人提倡在 SDLC 末期进行调优，因为他们抱怨前期调优经常会让设计和编码变得复杂。他们经常引用 Donald Knuth 的名言“过早优化是万恶之源”[6]。

为了平衡调优和编码需要一些权衡。根据笔者的经验，适当的前期调优能带来更明智的设计和细致的编码。很多项目就失败在应用程序调优上，因为上面提到的“过早优化”阶段在被引用时脱离了上下文，而且相应的调优不是被推迟得太晚就是投入资源过少。

但是，要做很多前期调优也不太可能，因为没有经过剖析，你并不能确定应用程序的瓶颈究竟在何处，应用程序一般都是这样演化的。

对我们的多线程企业级应用程序的剖析也表现出大多数应用程序平均只有 20-50%的 CPU 使用率。剩余的 CPU 开销只是在等待数据库和网络相关的 IO。

基于上述分析，我们得出这样一个结论，结合业务规则和设计的 Hibernate 调优在 Pareto 定律中 20%的那个部分，相应的它们的优先级更高。

一种比较实际的做法是：

1. 识别出主要瓶颈，可以预见其中多数是 Hibernate、业务规则和设计方面的（其数量视你的调优目标而定；但三到五个是不错的开端）。
2. 修改应用程序以便消除这些瓶颈。

3. 测试应用程序，然后重复步骤 1，直到达到你的调优目标为止。

你能在 Jack Shirazi 的《Java Performance Tuning》[7]一书中找到更多关于性能调优阶段的常见建议。

下面的章节中，我们会按照调优的大致顺序（列在前面的通常影响最大）去解释一些特定的调优技术。

3. 监控和剖析

没有对 Hibernate 应用程序的有效监控和剖析，你无法得知性能瓶颈以及何处需要调优。

3.1.1 监控 SQL 生成

尽管使用 Hibernate 的主要目的是将你从直接使用 SQL 的痛苦中解救出来，为了对应用程序进行调优，你必须知道 Hibernate 生成了哪些 SQL。JoeSplosky 在他的《The Law of Leaky Abstractions》一文中详细描述了这个问题。

你可以在 log4j 中将 **org.hibernate.SQL** 包的日志级别设为 DEBUG，这样便能看到生成的所有 SQL。你还可以将其他包的日志级别设为 DEBUG，甚至 TRACE 来定位一些性能问题。

3.1.2 查看 Hibernate 统计

如果开启 **hibernate.generate.statistics**，Hibernate 会导出实体、集合、会话、二级缓存、查询和会话工厂的统计信息，这对通过 **SessionFactory.getStatistics()**进行的调优很有帮助。为了简单起见，Hibernate 还可以使用 MBean“**org.hibernate.jmx.StatisticsService**”通过 JMX 来导出统计信息。你可以在这个网站找到配置范例 。

3.1.3 剖析

一个好的剖析工具不仅有利于 Hibernate 调优，还能为应用程序的其他部分带来好处。然而，大多数商业工具（例如 JProbe [10]）都很昂贵。幸运的是 Sun/Oracle 的 JDK1.6 自带了一个名为“Java VisualVM” [11]的调试接口。虽然比起那些商业竞争对手，它还相当基础，但它提供了很多调试和调优信息。

4. 调优技术

4.1 业务规则与设计调优

尽管业务规则和设计调优并不属于 Hibernate 调优的范畴，但此处的决定对后面 Hibernate 的调优有很大影响。因此我们特意指出一些与 Hibernate 调优有关的点。

在业务需求收集与调优过程中，你需要知道：

- 数据获取特性包括引用数据（reference data）、只读数据、读分组（read group）、读取大小、搜索条件以及数据分组和聚合。
- 数据修改特性包括数据变更、变更组、变更大小、无效修改补偿、数据库（所有变更都在一个数据库中或在多个数据库中）、变更频率和并发性，以及变更响应和吞吐量要求。
- 数据关系，例如关联（association）、泛化（generalization）、实现（realization）和依赖（dependency）。

基于业务需求，你会得到一个最优设计，其中决定了应用程序类型（是 OLTP 还是数据仓库，亦或者与其中某一种比较接近）和分层结构（将持久层和服务层分离还是合并），创建领域对象（通常是 POJO），决定数据聚合的地方（在数据库中进行聚合能利用强大的数据库功能，节省网络带宽；但是除了像 COUNT、SUM、AVG、MIN 和 MAX 这样的标准聚合，其他的聚合通常不具有移植性。在应用服务器上进行聚合允许你应用更复杂的业务逻辑；但你需要先在应用程序中载入详细的数据）。

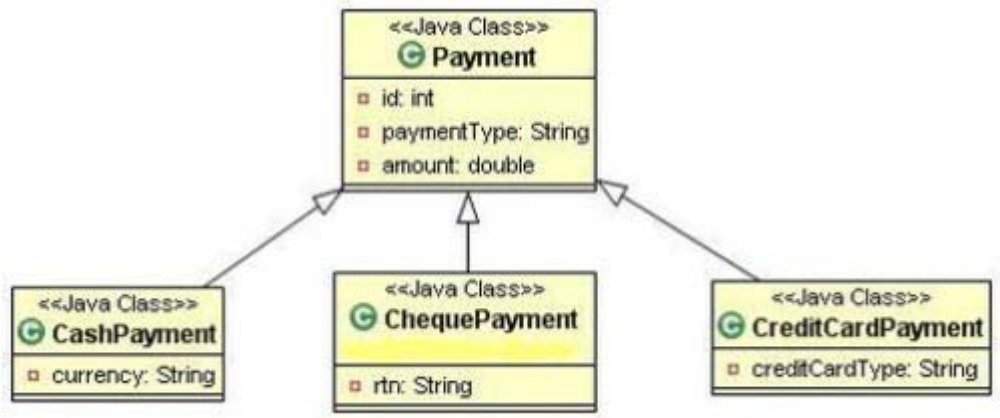
> 范例 **2**
>
> 分析员需要查看一个取自大数据表的电流 ISO（Independent System Operator）聚合列表。最开始他们想要显示大多数字段，尽管数据库能在 1 分钟内做出响应，应用程序也要花 30 分钟将 1 百万行数据加载到前端 UI。经过重新分析，分析员保留了 14 个字段。因为去掉了很多可选的高聚合度字段，从剩下的字段中进行聚合分组返回的数据要少很多，而且大多数情况下的数据加载时间也缩小到了可接受的范围内。
>
> 范例 **3**
>
> 过 24 个“非标准”（shaped，表示每小时都可以有自己的电量和价格；如果所有 24 小时的电量和价格相同，我们称之为“标准”）小时会修改小时电流交易，其中包括 2 个属性：每小时电量和价格。起初我们使用 Hibernate 的 select-before-update 特性，

就是更新 24 行数据需要 24 次选择。因为我们只需要 2 个属性，而且如果不修改电量或价格的话也没有业务规则禁止无效修改，我们就关闭了 select-before-update 特性，避免了 24 次选择。

4.2 继承映射调优

尽管继承映射是领域对象的一部分，出于它的重要性我们将它单独出来。HRD [1]中的第 9 章“继承映射”已经说得很清楚了，所以我们将关注 SQL 生成和针对每个策略的调优建议。

以下是 HRD 中范例的类图：



4.2.1 每个类层次一张表

只需要一张表，一条多态查询生成的 SQL 大概是这样的：

```
select id, payment_type, amount, currency, rtn, credit_card_type from payment
```

针对具体子类（例如 CashPayment）的查询生成的 SQL 是这样的：

```
select id, amount, currency from payment where payment_type='CASH'
```

这样做的优点包括只有一张表、查询简单以及容易与其他表进行关联。第二个查询中不需要包含其他子类中的属性。所有这些特性让该策略的性能调优要比其他策略容易得多。这种方法通常比较适合数据仓库系统，因为所有数据都在一张表里，不需要做表连接。

主要的缺点整个类层次中的所有属性都挤在一张大表里，如果有很多子类特有的属性，数据库中就会有太多字段的取值为 null，这为当前基于行的数据库（使用基于列的 DBMS 的数据仓库处理这个会更好些）的 SQL 调优增加了难度。除非进行分区，否则唯一的数据表会成为热点，OLTP 系统通常在这方面都不太好。

4.2.2 每个子类一张表

需要 4 张表，多态查询生成的 SQL 如下：

```
select id, payment_type, amount, currency, rtn, credit_card type,
      case when c.payment_id is not null then 1
           when ck.payment_id is not null then 2
           when cc.payment_id is not null then 3
           when p.id is not null then 0 end as clazz
rom payment p left join cash_payment c on p.id=c.payment_id left join
  cheque_payment ck on p.id=ck.payment_id left join
  credit_payment cc on p.id=cc.payment_id;
```

针对具体子类（例如 CashPayment）的查询生成的 SQL 是这样的：

```
select id, payment_type, amount, currency
from payment p left join cash_payment c on p.id=c.payment_id;
```

优点包括数据表比较紧凑（没有不需要的可空字段），数据跨三个子类的表进行分区，容易使用超类的表与其他表进行关联。紧凑的数据表可以针对基于行的数据库做存储块优化，让 SQL 执行得更好。数据分区增加了数据修改的并发性（除了超类，没有热点），OLTP 系统通常会更好些。

同样的，第二个查询不需要包含其他子类的属性。

缺点是在所有策略中它使用的表和表连接最多，SQL 语句稍显复杂（看看 Hibernate 动态鉴别器的长 CASE 子句）。相比单张表，数据库要花更多时间调优数据表连接，数据仓库在使用该策略时通常不太理想。

因为不能跨超类和子类的字段来建立复合索引，如果需要按这些列进行查询，性能会受影响。任何子类数据的修改都涉及两张表：超类的表和子类的表。

4.2.3 每个具体类一张表

涉及三张或更多的表，多态查询生成的 SQL 是这样的：

```
select p.id, p.amount, p.currency, p.rtn, p. credit_card_type, p.clazz
from (select id, amount, currency, null as rtn,null as credit_card type,
          1 as clazz from cash_payment union all

     select id, amount, null as currency, rtn,null as credit_card type,
          2 as clazz from cheque_payment union all

     select id, amount, null as currency, null as rtn,credit_card type,
          3 as clazz from credit_payment) p;
```

针对具体子类（例如 CashPayment）的查询生成的 SQL 是这样的：

```
select id, payment_type, amount, currency from cash_payment;
```

优点和上面的“每个子类一张表”策略相似。因为超类通常是抽象的，所以具体的三张表是必须的[开头处说的 3 张或更多的表是必须的]，任何子类的数据修改只涉及一张表，运行起

来更快。

缺点是 SQL（from 子句和 union all 子查询）太复杂。但是大多数数据库对此类 SQL 的调优都很好。

如果一个类想和 Payment 超类关联，数据库无法使用引用完整性（referential integrity）来实现它；必须使用触发器来实现它。这对数据库性能有些影响。

4.2.4 使用隐式多态实现每个具体类一张表

只需要三张表。对于 Payment 的多态查询生成三条独立的 SQL 语句，每个对应一个子类。Hibernate 引擎通过 Java 反射找出 Payment 的所有三个子类。

具体子类的查询只生成该子类的 SQL。这些 SQL 语句都很简单，这里就不再阐述了。

它的优点和上节类似：紧凑数据表、跨三个具体子类的数据分区以及对子类任意数据的修改都只涉及一张表。

缺点是用三条独立的 SQL 语句代替了一条联合 SQL，这会带来更多网络 IO。Java 反射也需要时间。假设如果你有一大堆领域对象，你从最上层的 Object 类进行隐式选择查询，那该需要多长时间啊！

根据你的映射策略制定合理的选择查询并非易事；这需要你仔细调优业务需求，基于特定的数据场景制定合理的设计决策。

以下是一些建议：

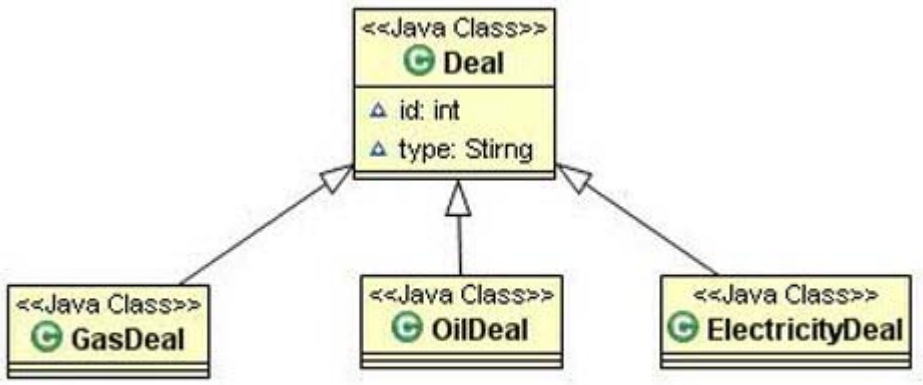
- 设计细粒度的类层次和粗粒度的数据表。细粒度的数据表意味着更多数据表连接，相应的查询也会更复杂。
- 如非必要，不要使用多态查询。正如上文所示，对具体类的查询只选择需要的数据，没有不必要的表连接和联合。
- “每个类层次一张表”对有高并发、简单查询并且没有共享列的 OLTP 系统来说是个不错的选择。如果你想用数据库的引用完整性来做关联，那它也是个合适的选择。
- “每个具体类一张表”对有高并发、复杂查询并且没有共享列的 OLTP 系统来说是个不错的选择。当然你不得不牺牲超类与其他类之间的关联。
- 采用混合策略，例如“每个类层次一张表”中嵌入“每个子类一张表”，这样可以利用不同策略的优势。随着你项目的进化，如果你要反复重新映射，那你可能也会采用该策

略。

- “使用隐式多态实现每个具体类一张表”这种做法并不推荐，因为其配置过于繁缛、使用“any”元素的复杂关联语法和隐式查询的潜在危险性。

范例 4

下面是一个交易描述应用程序的部分领域类图：



开始时，项目只有 GasDeal 和少数用户，它使用“每个类层次一张表”。

OilDeal 和 ElectricityDeal 是后期产生更多业务需求后加入的。没有改变映射策略。但是 ElectricityDeal 有太多自己的属性，因此有很多电相关的可空字段加入了 Deal 表。因为用户量也在增长，数据修改变得越来越慢。

重新设计时我们使用了两张单独的表，分别针对气/油和电相关的属性。新的映射混合了“每个类层次一张表”和“每个子类一张表”。我们还重新设计了查询，以便允许针对具体交易子类进行选择，消除不必要的列和表连接。

4.3 领域对象调优

基于 4.1 节中对业务规则和设计的调优，你得到了一个用 POJO 来表示的领域对象的类图。我们建议：

4.3.1 POJO 调优

- 从读写数据中将类似引用这样的只读数据和以读为主的数据分离出来。
- 只读数据的二级缓存是最有效的，其次是以读为主的数据的非严格读写。将只读 POJO

标识为不可更改的（immutable）也是一个调优点。如果一个服务层方法只处理只读数据，可以将它的事务标为只读，这是优化 Hibernate 和底层 JDBC 驱动的一个方法。

- 细粒度的 POJO 和粗粒度的数据表。

  基于数据的修改并发量和频率等内容来分解大的 POJO。尽管你可以定义一个粒度非常细的对象模型，但粒度过细的表会导致大量表连接，这对数据仓库来说是不能接受的。

- 优先使用非 final 的类。

  Hibernate 只会针对非 final 的类使用 CGLIB 代理来实现延时关联获取。如果被关联的类是 final 的，Hibernate 会一次加载所有内容，这对性能会有影响。

- 使用业务键为分离（detached）实例实现 equals()和 hashCode()方法。

  在多层系统中，经常可以在分离对象上使用乐观锁来提升系统并发性，达到更高的性能。

- 定义一个版本或时间戳属性。

  乐观锁需要这个字段来实现长对话（应用程序事务）[译注：session 译为会话，conversion 译为对话，以示区别]。

- 优先使用组合 POJO。

  你的前端 UI 经常需要来自多个不同 POJO 的数据。你应该向 UI 传递一个组合 POJO 而不是独立的 POJO 以获得更好的网络性能。

  有两种方式在服务层构建组合 POJO。一种是在开始时加 3.2 载所有需要的独立 POJO，随后抽取需要的属性放入组合 POJO；另一种是使用 HQL 投影，直接从数据库中选择需要的属性。

  如果其他地方也要查找这些独立 POJO，可以把它们放进二级缓存以便共享，这时第一种方式更好；其他情况下第二种方式更好。

4.3.2 POJO 之间关联的调优

- 如果可以用 one-to-one、one-to-many 或 many-to-one 的关联，就不要使用 many-to-many。

- many-to-many 关联需要额外的映射表。

  尽管你的 Java 代码只需要处理两端的 POJO，但查询时，数据库需要额外地关联映射表，修改时需要额外的删除和插入。

- 单向关联优先于双向关联。

  由于 many-to-many 的特性，在双向关联的一端加载对象会触发另一端的加载，这会进一步触发原始端加载更多的数据，等等。

  one-to-many 和 many-to-one 的双向关联也是类似的，当你从多端（子实体）定位到一端（父实体）。

  这样的来回加载很耗时，而且可能也不是你所期望的。

- 不要为了关联而定义关联；只在你需要一起加载它们时才这么做，这应该由你的业务规则和设计来决定（见范例 **5**）。

  另外，你要么不定义任何关联，要么在子 POJO 中定义一个值类型的属性来表示父 POJO 的 ID（另一个方向也是类似的）。

- 集合调优

  如果集合排序逻辑能由底层数据库实现，就使用“order-by”属性来代替“sort”，因为通常数据库在这方面做得比你好。

  集合可以是值类型的（元素或组合元素），也可以是实体引用类型的（one-to-many 或 many-to-many 关联）。对引用类型集合的调优主要是调优获取策略。对于值类型集合的调优，HRD [1]中的 20.5 节“理解集合性能”已经做了很好的阐述。

- 获取策略调优。请见 **4.7** 节的范例 **5**。

> 范例 **5**
>
> 我们有一个名为 ElectricityDeals 的核心 POJO 用于描述电的交易。从业务角度来看，它有很多 many-to-one 关联，例如和 Portfolio、Strategy 和 Trader 等的关联。因为引用数据十分稳定，它们被缓存在前端，能基于其 ID 属性快速定位到它们。
>
> 为了有好的加载性能，ElectricityDeal 只映射元数据，即那些引用 POJO 的值类型 ID 属性，因为在需要时，可以在前端通过 portfolioKey 从缓存中快速查找 Portfolio：
>
> ```
> <property name="portfolioKey" column="PORTFOLIO_ID" type="integer"/>
> ```
>
> 这种隐式关联避免了数据库表连接和额外的字段选择，降低了数据传输的大小。

4.4 连接池调优

由于创建物理数据库连接非常耗时，你应该始终使用连接池，而且应该始终使用生产级连接池而非 Hibernate 内置的基本连接池算法。

通常会为 Hibernate 提供一个有连接池功能的数据源。Apache DBCP 的 BasicDataSource[13]是一个流行的开源生产级数据源。大多数数据库厂商也实现了自己的兼容 JDBC 3.0 的连接池。举例来说，你也可以使用 Oracle ReaApplication Cluster [15]提供的 JDBC 连接池[14]以获得连接的负载均衡和失败转移。

不用多说，你在网上能找到很多关于连接池调优的技术，因此我们只讨论那些大多数连接池所共有的通用调优参数：

- 最小池大小：连接池中可保持的最小连接数。
- 最大池大小：连接池中可以分配的最大连接数。

  如果应用程序有高并发，而最大池大小又太小，连接池就会经常等待。相反，如果最小池大小太大，又会分配不需要的连接。
- 最大空闲时间：连接池中的连接被物理关闭前能保持空闲的最大时间。
- 最大等待时间：连接池等待连接返回的最大时间。该参数可以预防失控事务（runaway transaction）。
- 验证查询：在将连接返回给调用方前用于验证连接的 SQL 查询。这是因为一些数据库被配置为会杀掉长时间空闲的连接，网络或数据库相关的异常也可能会杀死连接。为了减少此类开销，连接池在空闲时会运行该验证。

4.5 事务和并发的调优

短数据库事务对任何高性能、高可扩展性的应用程序来说都是必不可少的。你使用表示对话请求的会话来处理单个工作单元，以此来处理事务。

考虑到工作单元的范围和事务边界的划分，有 3 中模式：

- 每次操作一个会话。每次数据库调用需要一个新会话和事务。因为真实的业务事务通常包含多个此类操作和大量小事务，这一般会引起更多数据库活动（主要是数据库每次提交需要将变更刷新到磁盘上），影响应用程序性能。这是一种反模式，不该使用它。

- 使用分离对象，每次请求一个会话。每次客户端请求有一个新会话和一个事务，使用 Hibernate 的“当前会话”特性将两者关联起来。

  在一个多层系统中，用户通常会发起长对话（或应用程序事务）。大多数时间我们使用 Hibernate 的自动版本和分离对象来实现乐观并发控制和高性能。

- 带扩展（或长）会话的每次对话一会话。在一个也许会跨多个事务的长对话中保持会话开启。尽管这能把你从重新关联中解脱出来，但会话可能会内存溢出，在高并发系统中可能会有旧数据。

你还应该注意以下几点。

- 如果不需要 JTA 就用本地事务，因为 JTA 需要更多资源，比本地事务更慢。就算你有多个数据源，除非有跨多个数据库的事务，否则也不需要 JTA。在最后的一个场景下，可以考虑在每个数据源中使用本地事务，使用一种类似“Last Resource Commit Optimization”[16]的技术（见下面的范例 **6**）。
- 如果不涉及数据变更，将事务标记为只读的，就像 **4.3.1** 节提到的那样。
- 总是设置默认事务超时。保证在没有响应返回给用户时，没有行为不当的事务会完全占有资源。这对本地事务也同样有效。
- 如果 Hibernate 不是独占数据库用户，乐观锁会失效，除非创建数据库触发器为其他应用程序对相同数据的变更增加版本字段值。

范例 **6**

我们的应用程序有多个在大多数情况下只和数据库“A”打交道的服务层方法；它们偶尔也会从数据库“B”中获取只读数据。因为数据库“B”只提供只读数据，我们对这些方法在这两个数据库上仍然使用本地事务。

服务层上有一个方法设计在两个数据库上执行数据变更。以下是伪代码：

```
 //Make sure a local transaction on database A exists
@Transactional (readOnly=false, propagation=Propagation.REQUIRED)
public void saveIsoBids() {
  //it participates in the above annotated local transaction
  insertBidsInDatabaseA();

  //it runs in its own local transaction on database B
  insertBidRequestsInDatabaseB(); //must be the last operation
```

因为 **insertBidRequestsInDatabaseB()**是 saveIsoBids ()中的最后一个方法，所以只有下

面的场景会造成数据不一致：

在 saveIsoBids()执行返回时，数据库“A”的本地事务提交失败。

但是，就算 saveIsoBids()使用 JTA，在两阶段提交（2PC）的第二个提交阶段失败的时候，你还是会碰到数据不一致。因此如果你能处理好上述的数据不一致性，而且不想为了一个或少数几个方法引入 JTA 的复杂性，你应该使用本地事务。

（未完待续）

关于作者

Yongjun Jiao 是 SunGard Consulting Services 的技术主管。过去 10 年中他一直是专业软件开发者，他的专长包括 Java SE、Java EE、Oracle 和应用程序调优。他最近的关注点是高性能计算，包括内存数据网格、并行计算和网格计算。

Stewart Clark 是 SunGard Consulting Services 的负责人。过去 15 年中他一直是专业软件开发者和项目经理，他的专长包括 Java 核心编程、Oracle 和能源交易。

[译注：由于原文较长，中译版分两次发布]

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

相关内容：

- 揭秘淘宝自主研发的文件系统--TFS
- 使用Java构建高伸缩性组件
- 极端事务处理模式：Write-behind缓存
- 为网站和智能手机构建FlightCaster前台应用

吴磊畅谈 Silverlight 在中国人寿的应用

作者 朱永光

InfoQ：大家好，这里是全球企业软件开发大会 QCOM 北京 2010 的现场，我们有幸邀请到了中国人寿的技术主管吴磊，给我们做一个采访，你能向大家做个自我介绍吗？

吴磊（以下简称吴）：我叫吴磊，从 03 年开始进行软件开发，在中国人寿工作已有七年，主要从事核心业务系统和管理系统的开发，就包括这次我们用 Silverlight 进行开发的养老金咨询平台，代号就叫 PACS。

InfoQ：那么对于这个 **PACS** 平台，在使用 **Silverlight** 开发过程中，你觉得最大的收获是什么？

吴：最大的收获，我认为是整个开发模式的变化，因为原来对于这种传统的核心业务系统或者其他类似管理系统的开发，我们更多是使用一种面向功能的开发过程。但现在我们基于 RIA 思路去开发的话，就是基于一种以用户体验为中心的开发过程。那么在整个需求分析的过程中， 我们的需求分析人员从原来更多依托于 UML 或需求规格说明书去分析问题，变成了基于原形基于用户体验的分析过程了，这就是我们最大的一个收获。第二个收获是，我们据此了解了眼下所有主流的 RIA 技术，知道了目前在 RIA 这个应用领域里面，有多少的技术可以被企业所用到，而且也沉淀出了基于 Silverlight 的开发框架。

InfoQ：那么在开发这个系统过程中，遇到的最大挑战是什么？以及如何应对这些挑战呢？

吴：在整个 PACS 的开发过程中，我们遇到的最大挑战，实际上是 Silverlight 技术本身。因为它比较新，在整个开发过程中，需要有一段的时间去学习，并培训我们团队的成员。这个过程我们花费了大量时间，也得到了微软公司的支持，他们给我们进行了两到三次的现场培训。之后，我们也去参观和学习了一些专门的视觉团队。在这些视觉团队中，他们具有很丰富的用户体验设计能力。从而，我们团队也学习了一些用户体验方面的经验。就这样，我们克服了开发遇到的一个很大难点。

InfoQ：对于这个业务系统，你感觉在当前的版本当中，技术层面还存在那些方面不足，打算如何来解决这些问题呢？

吴：PACS 目前存在的最大一个问题，还是我们用户习惯上的问题。因为我们原来的一些互联网客户，主要使用到或接触到的都是一些基于 HTML 的页面。而对于 PACS，首先就需要他下载安装一个插件并等待启动的过程，对此需要一个接受的过程。普通的用户可能很快就能适应，但对于企业中年龄比较大的用户就是个问题，他们对于互联网不是很了解，对于 RIA 应用就更不了解了。在这种情况下，他们需要要去接受一种新的事物是比较难的。我们认为在这个困难面前，需要努力加强的就是，怎么让整个体验过程更加简单。，我们内部也是提出了一种方法，就是让我们这个系统变成更加傻瓜的系统。操作上就像使用傻瓜相机一样简单，而功能上就像单反相机一样专业。这就是我们考虑的主要内容，也是现在 Silverlight 存在的一些问题。

InfoQ：你的意思目前存在的主要还是用户体验方面的问题，那么技术层面有没有不足？

吴：技术层面实际上也有一些局限，包括 Silverlight 应用程序下载包（XAP）的大小问题。因为我们在做企业应用开发的时候，可能面临的功能点非常多，这样的应用会非常庞大，随着这个应用的加大，功能的增多，如果只是在单一的 XAP 包里面进行部署，这个 XAP 包会越变越大。目前我们的 XAP 包已经有接近 1.3M 的大小了。如果我们后面再往上追加功能的话，XAP 包的大小会越来越大。目前，我们也在通过很多渠道，包括跟微软美国总部 Silverlight 产品组进行直接的交流，看能不能直接找到一些让 XAP 包分包下载，按需下载的方式【编者注：吴磊使用的 Silverlight 3 开发，在 Silverlight 4 中已经可以实现简单的按需下载功能】，这样来克服里面的困难。

InfoQ：就是说在目前你们是基于 **Silverlight 3** 来进行开发的，那么有打算升级到 **Silverlight 4** 吗？如果有的话，你觉得会遇到什么样的困难，以及需要特别注意什么东西？

吴：实际上我们一直在关注 Silverlight 4 的技术趋势，包括 Silverlight 4 最早于去年发布的第一个 BETA 版本的时候，我们就已经在试用了。我们团队里面有专门的人去研究 Silverlight 4 的一些特性。而且我们现在已经有一个详细的计划，就是什么时候去升级到这个新版本，因为现在大家都知道 Silverlight 4 才刚刚发布 RTM 版本，未来可能会有一个专门针对中文的语言包。那么我们要等到这个中文语言包的 Silverlight 出来以后，我们就会进行一个系统的预迁移。那么在这个预迁移过程中，可能我们会首先去解决一些命名空间上的问题。因为当时我们在做 Silverlight 2 到 Silverlight 3 的升级过程中，我们就遇到了一些命名空间上的命名不一样的问题。所以这一块需要我们有一个详细的计划。

InfoQ：那在下一步系统升级过程中，你们迫切需要解决的问题是什么？刚才你提到的这些问题？

吴：刚才提到最关键的是客户端这个 XAP 包的大小问题，这个问题实际上我们现在已经有了一些思路。包括像使用.NET 里面最新的 MEF 框架的插件式功能。这种功能是可以随机去按需去索取这个 XAP 的，我们可以把它分开打包，然后通过 MEF 的插件式方式，把它加载到我们对应的页面上去。但这里面也有一些问题，比如说不同的 XAP 包之间，它的资源（包括图片与样式）是不能共享的，所以我们现在也在考虑，通过 TCP 后台推送的方式直接把公共的资源文件打包下载到客户端独立存储中，下载下来以后，在客户端进行解析，实现 XAP 包之间的资源共享。

InfoQ：最迫切需要解决的还是性能问题？

吴：还是一个用户下载过程中的体验问题。对于性能方面的问题，我们刚才一直没有谈到。在性能方面，实际上我们因为在客户端这边运用了一个多线程分配的技术，就是我们所有任务是通过这种并行的线程去分派的，所以目前从性能上来说，还没有很大的缺陷，还是比较稳定的。特别是在一些大数据量的客户端操作的时候，目前来看还是比较满意。

InfoQ：我们知道 **Silverlight 4** 刚刚发布了正式版，你觉得 **Silverlight 4** 最大的进步是什么方面？你会在新的系统应用它什么特性？

吴：实际上，对于 Silverlight 大家可以看到，从 1.0、1.1 到目前的 4.0，它整个走的过程是非常的快，如果和我们用到的其他一些同类 RIA 技术相比的话，这种升级速度是非常快的。这样既有好处，也有坏处。好处就是它可以持续的改进，坏处就是我们需要不断的去学习，去掌握新版本的一些新特性。但是我们在 Silverlight 4 里面，看到了企业应用的一些特性。之前我们跟微软的一些产品组也进行了深入的沟通，希望加入一些包括调用 COM 组件的新特性。当时我们提出了需要在 Silverlight 里面去调用我们的一些针对 Office 的 COM 组件。在 Silverlight 4，当然微软做的比较更进一步，直接让所有的 COM 都可以在 Silverlight 里面去调用了，这个是一方面。第二个方面，Silverlight 4 内置了 WCF RIA Services，对企业级应用的业务数据处理支持力度更大。在 Silverlight 3 里面，企业开发者需要手写代码去调用后端数据服务，但是在 Silverlight 4 里，它的整个开发环境可以不写任何代码，就能够实现商业化应用程序里面数据的转换和处理，可以说是 Silverlight 的 WCF RIA Services 是面向商业应用中非常大的一个进步，我们也是关注这方面的一个动态。

InfoQ：你刚才提到的，**Silverlight** 对脱离浏览器运行的这种支持越来越好，认为 **Silverlight 4** 有可能会成为开发简单桌面程序的一种技术选择吗？或者说，就是 **Silverlight** 会成为未来桌面平台上一种统一的轻量级 **UI Toolkit** 吗？

吴：这个问题我觉得提得非常好，因为 Silverlight 本身，它的平台技术，是脱胎于整个 WPF 框架的，WPF 刚好是在整个 CS 架构里面微软主要推崇的技术。为什么 Silverlight 要有脱离

浏览器的方式呢？为什么不直接使用 WPF 呢？主要的一个原因是浏览器就是我们客户端的主要环境，而 Silverlight 不会对这个环境有任何的依赖性，因为它不会依赖于客户端是否安装了.NET Framework。他自己有一整套.NET CoreCLR 运行环境，所以能让我们所有的应用可以很正常地在客户端可以得到安装运行。那么在脱离了浏览器以后，就可以使的我们原来的一个 BS 架构的应用变成一个 CS 架构的应用，当然也可以使我们原有的 CS 架构的应用更加高效，更加适用于在客户端运行。在这个方面，我也了解到国外的一些厂商已经开始在这方面做一些产品了，包括像英国某个航空公司，可以在客户端上安装一个售票系统；包括我们国内的新浪财经，他可以在客户端上安装他的某种金融软件，而这个软件是不需要任何.NET Framework 去支持它的。我们这边也有这方面的打算，包括以后考虑在客户端上做一些数据处理方面的一些轻量级的应用，包括客户的一些员工数据的处理、检查，都可以通过我们脱机浏览的方式在他本机运行，这就是我们对于 OOB 的一些认识。

InfoQ：那对于下一个版本的英文，你最期待是什么新特性呢？

吴：下一个版本的 Silverlight，实际上我们看到它在版本 3 到版本 4 的演进过程中，已经加入非常多的特性。对于下一个版本，我很难想象 Silverlight 会是一个什么样子，或者说他功能到底有哪些。但是我认为他应该更加去注重在企业级应用这块的功能加强。特别是，可以考虑把 WCF RIA Services 这个功能嵌入到 Silverlight 里面，以便可以通过这种通用架构更加快速地开发商业应用。对于其他方面，我还没有想的更清楚。

InfoQ：最后一个问题，你认为现有的，或者你用过的一些 **RIA** 技术，他们各自的特点，以及自己的一些体会呢？

吴：实际上我刚才在介绍的时候，已经提到了我在以前、开发过一些使用 Flash、Flex 的一些技术的系统。包括当时在 Flex 1.0 的时候，我使用 Flex 开发过一个公司的资产管理平台。当时对于 Flex 这个技术，还是 Macromedia 公司在进行开发的。Flex 我认为他更多是一种设计的思路，用的编程语言也还是基于解释型的脚本方式去执行。所以说当时在开发第一个资产管理系统的时候，就遇到很多性能上的问题。但是后面，我看到 Flex 已经不断推出新版本，功能和性能都有所加强了。我认为这也是现在 Silverlight 的出现促使这个 RIA 整个产品线有了一个竞争的关系了。因为原来可能是 Adobe 一家独大的势态，但是现在有了 Silverlight 的介入，有了 JavaFX 的介入，我们可以看到有更多的 RIA 技术可以供我们去选择，所以说我认为这是一个好事。因为出现竞争以后，大家彼此在技术方面的更新频率就会更快，对于我们终端用户来说，可以有更多的选择，也可以去开发更炫更酷的应用。

原文链接：http://www.infoq.com/cn/articles/wl-silverlight-chinalife

推荐文章 | Articles

SOA 分析和设计中的错误处理要点

作者 Hari Poolla 译者 吴宇

在 SOA 分析和设计阶段进行全面的错误处理需求分析对于正确完成设计和实施服务至关重要。缺少具体的需求以明确出错处理场景，或者缺乏将这些场景纳入 SOA 分析及设计阶段的理解，都将导致在开发阶段致力于将设计服务放在首位，只要完成功能，走正确路径（不考虑出错场景）即可。这种方法往往藏有隐患，会 导致项目超支，因为之后补加的错误处理方案会产生大量的返工甚至需要对某些组件进行重新设计。

本文探讨了多个与可重用服务设计相关的错误处理要点，提供了一个框架用以说明在 SOA 分析和设计阶段应该使用何种错误处理，另外也描述了一组最佳实践，以及如何将这些最佳实践应用到设计中以确保最终的服务得以最完整的设计和实施。

简介

不同于单一整体应用，在 SOA 环境中，错误处理已成为应用设计中一个非常重要的步骤，因为 SOA 应用集成了各种异构（跨各组织边界、厂商及合作伙伴 IT 资 产）的 IT 系统。在分析和设计的初期就关注错误的处理分析，能够确保在不同的平台中为不同的模块提供适合的错误处理标准/准则。本文所罗列的是一般性的错 误处理要点，也是架构师和设计师们在 SOA 方案设计中需要解决的。SOA 分析和设计任务大致可分为三大阶段：服务标识、服务规范、和服务实现，具体可参考 Ali Arsanjani 的面向服务的建模和架构。后续关于该话题的讨论所围绕的错误处理要点都适用于这三大阶段。

服务标识中的错误处理

服务标识的目的是为了确定某一侯选服务组合，使其能够识别可重用的服务组合。该阶段包含了对业务工件包（其中包括关键需求、业务目标、功能模块、业务流程分析模块、用例等）的分析。

错误类型

错误大致可分为两类：

- 可恢复性错误——可恢复性错误是指那些可通过恰当的可选执行路径来恢复用户程序的错误。这些错误是由于不遵守某一特定业务规则所导致的。
- 不可恢复性错误——这些错误是客户程序无法恢复的。此类错误是在运行时中由某些不可预期的错误产生的，诸如空指针、资源不可用等编程性错误。

业务错误的标识

分析整个业务工件包为发现与服务关联的业务错误提供了很多机会。假设已存在某资产正用于某业务服务，那些组件接口可用于发掘那些通过自上而下分析所不能发现的其他业务错误。所谓业务错误实际上是一种可恢复性出错。一旦服务组合进入了最后的定稿阶段，那么请从以下几个错误处理的方面来评估可重用服务：

- 业务错误场景——详细描述会标记某业务操作无效的条件。
- 错误概述——当某一业务错误发生时服务消费者收到的业务出错提示，它提供对业务出错的简单描述。
- 错误代码——可用于查询和某错误相关的附加信息的代码。
- 建议——给服务消费者的反馈，诸如有效输入、或显示有关出错的具体信息等等。
- 服务区域——标识某一服务区域，该区域可接收到与服务系统错误相关的所有通知。

这些定义了业务出错的属性既可以涵盖在服务合同中也可以按照需求打包进服务响应中。

流程错误恢复场景

- 标识新操作——业务流程（长流程）或者任何微流程（短流程）都需要根据各自服务在其流程中可能抛出的业务错误来加以分析。这种分析可以引导发现更新的操作，而这些操作在传统的自上而下的流程任务分解中是无法被发现的。
- 更新流程模型——服务操作模型/依赖性可随着在前一步中发现更新的操作而得到更新。

服务规范中的错误处理

服务规范阶段包含以下任务：定义输入及输出消息、服务和操作名称、模式、服务组合、非功能需求以及其他诸如同步/异步、调用方式等被标记出来作为服务的特征。

错误处理相关的特征

与错误处理相关的一般服务特征有：

- 有保障的交付——决定了某一服务是否要求有保障的 QOS（服务质量）交付类型。该需求帮助设计者们设计合适的异步消息处理设计模式，或者如果采用 Web 服务来实现的话使用可靠的消息处理机制。
- 监控需求——决定了是否需要为服务业务关键性错误建立事前监控。
- 错误映射/转换规则——建立由服务提供者所返回的错误代码/信息的转换规则，以及这些返回的信息该如何提供给服务消费者。就处理服务出错而言，标准的业务错误代码有助于应用程序更得心应手地使用这些服务。
- 更新处理流程——随着较新的操作发生或者在标识阶段处理业务错误时所发现的可选执行路径，现有的处理流程将得到进一步更新。
- 事务属性和边界——各种错误各具特色，例如系统出错与程序出错，这种错误的特性影响不同的运行时平台如何处理自动回滚。在某一流程中的事务属性和边界将根据发生在单个服务调用/事务时的错误进行分析。

错误的传递

可以使用多种方法来向服务消费者们传递错误，而其中非常重要的就是要给企业作出一个架构设计决定用以选择一个最为合适的传递方式。

两种最受欢迎的返回错误信息的方式就是使用 SOAP fault 或者定制化错误消息体。两种方式各有利弊，人们选择时大多基于现有的开发和运行时平台。使用面向 Web 服务的平台所实现的服务找到 SOAP fault 作为其必然选择是因为有大量基于 Web 服务工具的支持，而定制化错误消息体则更加适合那些使用传统的面向消息的中间件（MOM）平台所实现的服 务。有关这个问题的进一步的探讨可参见 Boris Lubinsky 所做的比较。 总而言之，鉴于客户工具在各种不同程度上的支持以及服务器对各种不同概要（profiles）的支持，不可恢复性的系统

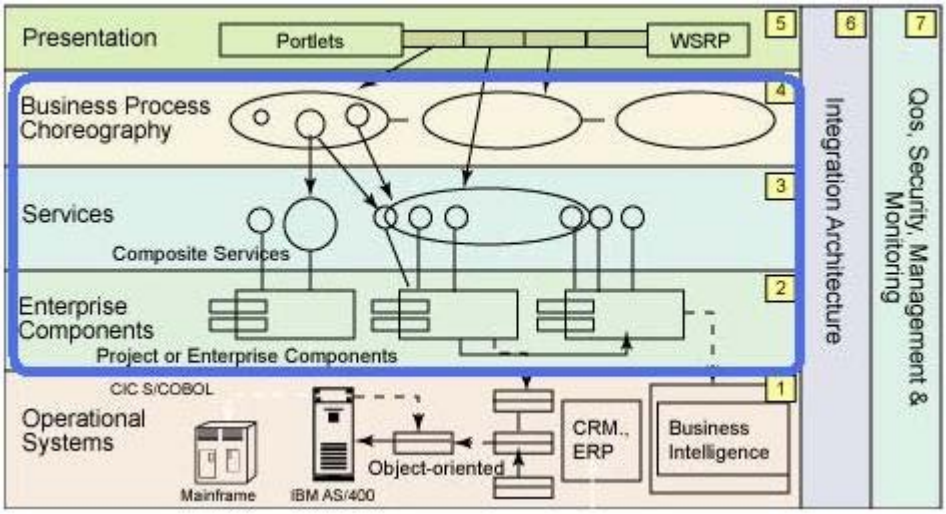
类错误更宜作为 SOAP faults 返回，而可恢复性的业务错误则更适合使用定制化错误消息体进行描述，因其能灵活和扩展地定义定制化错误模式（schema）。尽管如此，使用 定制化错误消息体仍然要求服务消费者做额外的客户端处理程序来解析响应消息从而判断调用是否成功。

一般企业范围的自定义模式

通过标识元数据及一般模式来持续描述企业中出现的各类错误。该数据可包含常用的属性，这些属性包括日期、时间、错误代码、描述、严重级别、消息来源、相关 ID 等等。对于该元数据的全方面分析将十分有助于建立有效的服务监测。

服务实现中的错误处理

服务实现阶段是指服务模型影射于服务组件和运行时/部署模型时期。该步骤一般包括设计服务组件、分配组件至 SOA 各堆栈层、选择组件交互方式、运行时平 台，以及制定架构设计决策(ADD)。关于此话题的后续讨论将着重围绕一些最佳实践，其中将错误处理方案实施到典型的企业 SOA 堆栈三个层次中：业务流程或编排、中介/总线以及组件，如下图所示



业务流程/编排层的错误处理

该层所部署的组件实施业务流程或编排。以下是可用的错误处理要点：

- 错误处理器——错误处理器是解决编排向外调用过程中返回的错误的最受欢迎的方式之一。错误处理器在某一流程中与特定任务绑定，或者作为整个流程的一种全方位错误

处理工具。当流程中有错误发生时，错误处理器就会被调用，由它实施更正的任务。补偿事务和手动回滚都设置了错误处理器，所以恰当的纠正措施可应用于处理流程错误。使用错误处理器时，不应将其用于可选执行路径上，而应该用于对流程抛出的错误进行恢复。

- 服务状态信息——流程编排场景通常包含调用多个服务。这些在流程内部所调用的服务可能会因多种严重程度等级（从信息、警告、错误到致命）的出错而终止。从每个调用中收集状态描述是很好的实践，例如将错误代码等返回到一个可重复的数组中然后原样返还给服务消费者。这样的做法使服务消费者能够判断整个流程是否包含由某些服务中流程所调用所引起的任何警告/错误。
- 阀值错误严重级别——定义错误严重级别阈值，并在服务编排中根据这些阈值设计容错级别。阀值级别可根据任意属性或某个定义错误的组合（如出错严重级别、定制状态代码等）来确定，而这些确定因素不能仅仅依靠 SOAP faults 来判断流程的失败。

服务/协调/**ESB** 层的错误处理

企业服务总线(ESB) 层是典型企业 SOA 堆栈的核心。该层支持所需的转换及路由能力独立于企业的可复用服务。该层组件通过使用恰当的消息和协议转换为各种提供商实现（如现有相关资产及基于合作伙伴或厂商的服务）提供了一种明确的接口。由协调组件完成的错误处理一般都包含将提供者错误结构转换成在业务域的上下文中明确定义的错误结构。这些组件也可以处理一些复杂的转换和错误的影射规则，这些错误由后台功能组件返回，在企业内部为服务消费者们提供了更为简单的错误信息。

- 转换提供者错误代码——不同的服务提供者会使用不同的语义来返回服务错误。范围涵盖很广，从流行的 SOAP faults 到相当专用的结构。适当的转换规则可应用于此，以便可重用的企业服务能使企业应用程序以更一致的方式返回错误，轻松解析和实施合适的处理器。
- 过滤敏感信息——当内部服务组件抛出致命错误时，堆栈跟踪往往包含敏感信息，比如所使用的协议、服务器 IP 等等。这一层将设立适当的过滤规则来过滤所有在堆栈里跟踪的敏感信息。当通过可信赖的网络传达服务响应时这一策略变得尤为重要。
- 捕获应用错误——服务组件所经历的任何技术性错误（如资源不可用或是一些运行时异常等）会转换成某种简单的技术错误消息。如果本地组件不记录这些出错，那么协调层可以将所有堆栈跟踪信息传递到日志，但却只是将通用文本消息返还给服务消费者并通

知服务暂时不可用。

这一层所提及的许多出错处理的方案也可以应用于组件层。但是市场上有大量 ESB 和框架已经包含这些东西，其配置性和灵活性比起独立平台开发人员实现其功能组件实现方案要更佳。ESB 提供的出错处理仲裁关注点的分离使平台开发者们摆脱了必须应对各种各样出错处理方案的窘境，使他们把注意力更多地放在实施业务功能，提高生产力。

组件层的错误处理

这一层的组件出错处理包括处理异常操作的情况(如某一资源不可用或者运行时组件不按指令处理又或者背离逻辑)。组件需处理类似事件以通知用户程序，并且做适当的日志以便促进排除故障和服务监测。在 Java 编程语言中，类似事件的抛出称作异常，并且 API 提供了两种不同的异常：已检查异常和未检查异常。已检查异常继承自 Exception 类并用于处理可恢复性出错诸如业务出错场景。未检查异常作为运行时 Exception 类是理想的异常侯选处理不可恢复性出错，如资源不可用或者空指针。

组件层出错处理的第二个部分就是要做适当的日志。在离错误源发生最近的地方做日志是很好的做法。当组件抛出应用出错时，他们可以在组件边界内通过适当的接口记录异常，然后再抛出异常。使用关联性 id 来标识事件并将其传递给调用应用程序将可以通过链接不同平台日志的方式大大增强出错跟踪和监测。

事务回滚和补偿

设计合理的容错机制以维持流程中的 ACID（原子性、一致性、隔离性和持久性）特性是设计 SOA 解决方案的一大挑战。这些解决方案一般包含业务流程，可调用的服务可跨不同平台、多种交互模式及由不同的资源提供者提供。一个业务流程中所参与到的所有服务多半不是事务性的。一旦有某个特定的事务在某一流程中失效，就会有相应的恢复性措施用以维持数据的完整性。事务回滚和补偿事务是解决问题的两种方法。

事务回滚可通过一个事务监测器从而协调各事务来实施，前提是业务流程跨越一个限定域且所有资源都是事务型的。如业务流程更复杂些，无法完成业务流程不仅需要回滚，将数据返回至一致状态，而且需要流程调用一定的补偿事务（诸如发送警告，或在调用撤消操作来撤消前一步的服务调用等）。这远远超出了本文所论述的问题的范围。读者们可参考此空间中即将上传的 Web 服务标准：Web 服务协作（WS-C）和 OASISWeb 服务事务（WS-T）。

小结

本文提供了一些 SOA 架构技巧以用于从业务工件包中发现错误处理的需求以及在整个 SOA 分析和设计阶段如何分析这些需求。同时也提供了一些在 SOA（解决方案堆栈的）三层（如服务编排、协调及组件层）之中进行错误处理的最佳实践。

对各种错误处理要点的全面直接的分析有助于架构师们在设计和实施阶段、平台和 SOA 堆栈产品方面作出正确的决策。

关于作者

Hari Poolla 有着丰富的 SOA 实践经验 就职于位于美国中西部的一大型的保险金融服务公司。他主要关注于应用架构、BPM、解决方案设计、以及企业应用领 域的集成与合作，专攻建立企业可重用服务。为企业 SOA 采用路线图及设计定制 SAO 解决方案发展理论提供专业咨询。他具有 IBM 的 SOA 解决方案设计师认 证以及 Sun 认证的 Java2 平台企业架构师。他曾经参与电子移动基金(EFT)在各个不同行业构建和实施一个基于 SOA 的业务解决方案。你可以通过 hari_poolla@yahoo.com 与他联系。

原文链接：http://www.infoq.com/cn/articles/error-handling-soa-design

相关内容：

- 访谈：Paul Fremantle谈WSO2 Stratos
- 面向资源的架构：信息，而非容器
- 业务SOA治理
- REST和SOAP：谁更好，或者都好？

时刻关注企业软件开发领域的变化与创新

InfoQ中文站
www.infoq.com/cn

Java — .NET — Ruby — SOA — Agile — Architecture

Java社区：企业Java社区的变化与创新

.NET社区：.NET和微软的其它企业软件开发解决方案

Ruby社区：面向Web和企业开发的Ruby，主要关注Ruby on Rails

SOA社区：关于大中型企业内面向服务架构的一切

Agile社区：敏捷软件开发和项目经理

Architecture社区：设计、技术趋势及架构师所感兴趣的话题

讨 论 组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
广告合作：sales@cn.infoq.com

新品推荐 | New Products

Membase 公司公开发布 Membase 服务器

作者 Michael Hunger 译者 马国耀

从今天起，企业版和社区版 Membase 服务器可从 Membase 公司（原 NorthScale 公司）获得。Membase 公司是广泛使用的 Memcached 缓存基础设施的提供商。

原文链接：http://www.infoq.com/cn/news/2010/10/membase

LLVM 2.8 发布

作者 Alex Blewitt 译者 李明（nasi）

LLVM 团队昨日发布了 LLVM 2.8，包括下一代 C/C++编译器、优化器和运行时的低级虚拟机基础架构。VMKit 使用 LLVM 来实现 CLR 和 JVM 的运行时，MacRuby 和 Python 的 Unladen Swallow 也同样用到了 LLVM。另外，最近发布的 Mono 2.8 包含了一个 mono-llvm 运行时。那么，LLVM 2.8 都有哪些新特性？

原文链接：http://www.infoq.com/cn/news/2010/10/llvm-2.8

Google WebP--让图片更小，让页面访问速度更快

作者 Abel Avram 译者 霍泰稳

Google 日前对外宣布了一种新的图片压缩格式 WebP，可以让通过网络传输的图片变得更小。Google 声明说，WebP 格式可以让图片的大小减少 39%，从而加快了页面的装载速度。

原文链接：http://www.infoq.com/cn/news/2010/10/WebP

Adobe 展示 HTML5 动画制作 IDE

作者 Dionysios G. Synodinos 译者 崔康

Adobe 在今年的开发者大会上向公众展示了一款用于 HTML5 动画制作的 IDE。这款工具（内部称为 Edge）采用 WebKit 渲染引擎来预览动画并提供类似 Dreamweaver 的源代码编辑模　式。

原文链接：http://www.infoq.com/cn/news/2010/10/adobe-html5-animation-ide-edge

Adobe 发布 64 位 Flash Player 预览版，支持 IE9

作者 Dionysios G. Synodinos 译者 霍泰稳

近日 Adobe 发布了代号为"Square"的 Adobe Flash Player 的预览版，对 Linux、Mac 和 Windows 等操作系统均进行 64 位本地支持。另外，它还支持微软的 IE9 Beta 版本。

原文链接：http://www.infoq.com/cn/news/2010/10/flash-square

微软发布 Team Foundation Server 备份工具

作者 Jonathan Allen 译者 霍泰稳

在 Team Foundation Server 发布过三个主要版本后，微软终于为其发布了一个用于执行备份和恢复的工具。这个和 TFS Power 工具九月版一同发布的工具，很大程度上简化了备份 TFS/SharePoint 数据库的过程。

原文链接：http://www.infoq.com/cn/news/2010/10/TFS-Backup

推荐编辑 | .NET 社区编辑 赵颉

大家好，很高兴能在这期《架构师》和大家见面。我是赵劼，洋名 Jeffrey Zhao，网名老赵、赵姐夫，在 InfoQ 担任.NET 社区编辑。职业是钢琴手，可惜无法养活自己，因此混迹于 IT 行业补贴家用。目前就职于盛大创新院产品开发部，研究员。关注前沿技术，并致力于开源社区与微软平台的组合优化。对函数式编程，并行程序开发，代码之美以及程序员能力与修养等相关问题也有着浓厚的兴趣，同时非常希望能够写程序到 60 岁。最近致力于 F#，Scala 语言及 mono 平台在社区中的推广。

无论从哪方面来看，我都是个十足的年轻人，年轻的程序员。从小对编程有浓厚的兴趣，上大学前在编程方面的经验主要来自于信息学竞赛，到了大学里，学习（或自学）了 Java，少许 LISP，数据结构与算法，操作系统，计算机体系结构，计算机网络，编译原理等最传统的科班课程，但似乎学的不太好，现在想来颇是后悔。也学过几次 C++，但智商有限，几次下来都没有坚持到底，现在也忘得差不多了。总体来说，我的专业程序员生涯的成长离不开微软与.NET。

加入微软是个偶然，因为我在大学里学习和使用的都是 Java，包括一些稀奇古怪的“企业项目”，从现在看来，那时对.NET 和 C# 1.0 的了解只能算是一些皮毛。后来通过校招加入了微软，自然全面转向.NET。在微软我只呆了一年半的时间，但这一年半给我最大的帮助就是开阔了眼界，知道技术领域有多么广阔，知道单凭学校里获得的信息是多么的不靠谱。可以说在微软的这段时间对我来说是个突破，从那时起我就对各种技术都抱有强烈的兴趣。

我时常从一些前辈程序员那里听说微软善于淘汰自己的技术，例如使用.NET 代替旧有的 COM 技术，让程序员追逐地很累，且难以积累。可能年轻也是一种幸运，我的程序员生涯是随.NET 成长起来的。虽然前辈们一再警告.NET 终究会被微软抛弃，但是我并没有从.NET 这将近 10 年的发展过程中找到衰退的蛛丝马迹。如今在 mono 的帮助下，.NET 环境已经进入了 iOS，Android，以及 Mac OS 原生平台，可以说.NET 从未处在一个如此辉煌的时代。

从微软和.NET 里我学到了很多，因为它的每一步发展都是渐进的，踏实的，富有底蕴，我没有丝毫“追逐”之感。最重要的是，它为我打开了眼界，促使我去接触更多技术领域，Python，Scala，JVM，Rails，Linux.....我实在无法抵挡这些缤纷的魅力，时刻保持 Keep Stupid，Keep Hungry 的感觉。在我眼中，技术就是技术，我不会将它分为“微软”与“非微软”两部分。在我目前的工作中，我便是将.NET 与 Linux 上的 NoSQL 存储相结合，配合 C#强大的表现能力体现编程之美。

InfoQ 是一个综合性的技术媒体，不局限于某个特定技术领域，这有助于帮助我保持开放的眼界与心态。正因为如此，我当时向编辑团队毛遂自荐，希望有机会可以加入 InfoQ 的建设中来。InfoQ 也从未令我失望过，我虽然是.NET 社区的编辑，但只要愿意，人人都可以贡献任何内容。更重要的是，加入 InfoQ 以后我认识了许多优秀的朋友，从他们身上也学到很多。

我很年轻，我的路还很长，我会努力。

封面植物

天女木兰

天女木兰，木兰科木兰属植物，因花的花瓣洁白、雄蕊紫色、香气迷人、形似天女而得名天女木兰。天女木兰乃珍奇花木，系太古第四纪冰川期幸存的珍稀名贵花卉，被列入国家濒危植物名录，是我国重点保护的世界珍惜植物，叶如翠雕，花似玉琢，具有浓郁的芳香，是重要的园林绿化树种和提取香料的轻工业原料。

生长分布：天女木兰分布于长江以南和长城以北,多产于辽宁、安徽、江西、广西北郊,也是中国东北地区惟一的野生木兰属植物。朝鲜、日本也有分布。生于海拔 700-1800m 的山坡杂木林中。

植物形态：落叶小乔木或灌木，高 1.5-3m 或更高。树皮淡灰褐色;小枝细长，疏生细长毛，嫩枝密被短柔毛。冬芽椭圆形或长圆形，被棕色有光泽的短毛。单叶互生；叶柄长 1-4cm；托叶痕长为叶柄的 1 / 2；叶片披针状椭圆形或倒卵状椭圆形，长 6-14cm， 宽 4-10cm，先端尖或短渐尖，基部楔形，上面深绿色，无毛，下面灰绿色，除沿叶脉疏生绒毛外，余无毛。花与叶同时开放，单生于枝顶，有香气，呈杯状， 直径约 6cm；花被 9 片，肉质，乳白色或淡粉红色至粉红色，倒披针形或近匙形；雄蕊多数，长约 1cm，花丝长约 4mm，呈紫红色，花药药隔先端尖呈黄色； 雌蕊群椭圆形.雌蕊柄长约 5mm.花柱细而弯曲。聚合果长圆柱形，熟时深红色，木质。种子呈不规则的扁圆形，外种皮肉质，呈红色。花期 5-6 月，果期 8-10 月。花盛时节，满树银花，随风飘荡，芳香无比，宛如天女散花，深受人们喜爱。它属落叶小乔木，树高可达 10 米，树皮灰白色，小枝褐色，叶宽椭圆 形，单生于小枝顶端。

用途：为著名的庭园观赏树种。花朵冰清玉洁，香味醇厚致远，其花、叶、茎可提炼高级香料，具有极高观赏和经济值。可广泛用于食品、化工、医药、卷烟 、糖果、化妆品等，是增香剂制品的最好原料，国家轻工部已把它列为得料的重要原料之一。花可入药。

1kg.org 多背一公斤

爱自然 | 更爱孩子

架构师 11 月刊

每月 8 日出版

本期主编：霍泰稳

总编辑：霍泰稳

编辑：李明 胡键 宋玮 郑柯 朱永光 池建强

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

交流群组：
http://groups.google.com/group/infoqchina

商务合作：sales@cn.infoq.com 13911020445

本期主编：霍泰稳，**InfoQ** 中文站创始人兼总编辑

霍泰稳，有多年的软件开发经验和媒体从业经历，以技术传播为己任，关注企业软件开发领域的变化与创新。大学毕业后先是在杭州从事软件开发工作，基于多种语言和技术开发过 MIS、GIS 等系统，但现在大多技术已经遗忘。2004 年步入技术媒体行业，并渐渐沉溺其中不能自拔，在 CSDN 工作期间曾先后参与《程序员》杂志、《MSDN 开发精选》杂志、《开源大本营》图书和《开源技术选型手册》2008 版图书的策划编辑工作。后参与创建 InfoQ 中文站，并主导运营至今！

InfoQ Enterprise Software Development Community

《架构师》月刊由 InfoQ 中文站出品。

所有内容版权均属 C4Media Inc.所有，未经许可不得转载。

时刻关注企业软件开发领域的变化与创新

架构师

www.infoq.com/cn/architect

每月8号出版

InfoQ
Enterprise Software Development Community

商务合作：sales@cn.infoq.com
读者反馈/投稿：editors@cn.infoq.com