时刻关注企业软件开发领域的变化与创新

架构师

10月 ARCHITECT

Steve Jobs
1955-2011

特别专题

大数据时代

大数据，下一个新领域

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

了解云计算的漏洞

InfoQueue
Enterprise Software Development Community

每月8号出版

篇首语 | Editor's letter

谢谢你，乔布斯

这一天迟早要来。只是当它真的到来的时候，我们才知道，这种悲伤远比想象中要更加强烈。八年来，乔布斯一直在与疾病抗争。最终，病魔从我们的手中夺走了他，留下的满是遗憾和唏嘘，和一串在科技发展史上永不逝去的足迹。

乔布斯的离去让整个世界为之动容。奥巴马说：他改变了我们每个人看世界的方式。沃兹涅克说：我们失去了永远找不回的东西。蒂姆·库克说：从此我们失去了一位挚友，一位精神导师。比尔·盖茨说：他将继续影响今后几代人，和他共事是无比的荣耀。迈克尔·戴尔说：今天世界失去了一位富有想象力的领导者。拉里·佩奇说：他的专注一直启迪着我。谢尔盖·布林说：他追求完美的理念感动了所有接触过苹果产品的人。孙正义说：人们将像怀念达芬奇一样怀念他。这里有他的朋友，他的战友，甚至他的竞争对手。赢得整个世界的尊重，乔布斯做到了。

乔布斯为世界留下的，不仅仅是一个公司、一些产品，而是一种理念、一种信仰。是他让我们知道，优雅的美学可以与技术的发展完美的结合；是他让我们了解，执着地在设计中追求极致的价值所在；是他让我们看到，无止境的创造可以改变每个人的生活，乃至整个人类文明。他传奇般的人生经历，近乎偏执的追求完美，永不停息的自我超越，深深地震撼着每个人的心灵，烙印在我们的灵魂深处。而他的激情，他的教诲，他的坚持，也深深地鼓励着我们，成为我们追逐梦想的风帆，成为伴随我们前行的座右铭。

作为一个多年使用苹果产品的人，我十分感谢乔布斯为这个世界带来的美感和创想。它们就如同火种一般，点燃了艺术与技术的灵感，激发了无数的变革和创新，从而改变了整个世界。我们永远都不会忘记。谢谢你，乔布斯。

本期主编： 李明

目　录

[篇首语]

谢谢你，乔布斯 1

[人物专访]

争渡读屏杨永全谈网页无障碍设计与体验 4

[热点新闻]

WINDOWS 运行时的设计细节 9

GOOGLE APP ENGINE 涨价，开发者深受打击 12

用户抱怨 WINDOWS AZURE 14

MONO 将不会具备 METRO UI 17

"APACHE 杀手"一种利用 RANGE HTTP 头域的 DDOS 19

[特别专题]

大数据，下一个新领域 22

大数据时代的数据管理 25

阿里巴巴数据架构设计经验与挑战 30

大数据时代的创新者们 37

关系数据库还是 NOSQL 数据库 44

[推荐文章]

向 JAVA 开发者介绍 SCALA 49

HTML 5 OR SILVERLIGHT? 58

解析 JDK 7 的 GARBAGE-FIRST 收集器 63

了解云计算的漏洞 69

[新品推荐]

WINDOWS 8 将替换 WIN32 API 83

TWITTER STORM：开源实时 HADOOP 83

ECLIPSE 3.7.1 开始支持 JAVA 7 83

SYNC FRAMEWORK 打破了平台之间的屏障 84

SENCHA TOUCH 2：令人期待的新特性 84

SPRING SOCIAL 给 JAVA 带来 SOCIAL CONNECTIVITY 84

F# 3.0 -- LINQ + 类型提供程序 = 富信息编程 85

AMAZON 的全新浏览器 SILK 使用分离式架构 85

MAINTAINJ 3.2 业已发布，多项功能得到增强 85

[推荐编辑]

翻译团队编辑 高翌翔 86

[封面人物]

史蒂夫·乔布斯 87

InfoQ
Enterprise Software Development Community

时刻关注企业软件开发领域的变化与创新

我们的价值观：
创造可信赖的内容

我们的愿景：
帮助传播企业软件开发相关的知识和创新

我们的读者：
中高端技术人员，如架构师、项目经理、高级工程师等

我们的社区：
Java、.NET、Ruby、SOA、敏捷、架构和运维

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60位领域专家担当社区编辑

人物专访 | Interview

争渡读屏杨永全谈网页无障碍设计与体验

无障碍设计的目的就是希望让每个网站都可以很方便的被所有人访问，每个软件都可以很方便被所有人使用，让每个信息都可以让所有人很方便的获得，让所有人受益，是每个 IT 人共同的社会责任，也是一个社会文明的体现。本次采访了争渡读屏的团队负责人杨永全，将从无障碍体验及界面设计谈起，谈到了如何通过设计来 改善用户体验，同时举例说明了目前互联网产品中设计上的一些不足。此外还介绍了争渡读屏的技术原理和目前的发展情况。

杨永全，盲人按摩师，争渡读屏创始人之一，中国盲人在线网站站长。2008 年结识了朱兰强（争渡）开始开发争渡读屏软件。2011 年 5 月正式推出了争渡读 屏公益版，免费提供给盲人使用，成为内地首款免费读屏软件。从 2006 年至今，先后参与过自强盲人论坛、九州健康论坛、中国盲人在线等多个盲人网站的建设 和维护。长期关注和推动信息无障碍，曾推动了 QQ 空间、腾讯微博、新浪微博、淘宝网等的无障碍改造。

InfoQ：我们现在是在淘宝技术嘉年华会议现场，很荣幸邀请到争渡读屏的产品负责人杨永全，杨永全您好。

杨永全：您好。

InfoQ：下面想首先请您介绍下自己以及您所在的团队？

杨永全：大家好，我叫杨永全。现在在争渡读屏软件团队工作。主要负责软件的设计，包括测试，以及用户售后方面的一些工作。我们的团队是 2008 年组建的，到目前为止，团队成员之间还未曾谋面，十几位成员都是通过网络来联系，其中包括我们软件的设计、规划、讨论等工作，都是在网上通过 QQ、Skype 这些工具来进行交流、沟通的。我们的团队构成，有盲人，也有明眼人。

InfoQ：团队的分工大概是什么样的？

杨永全：盲人负责一些包括设计，还有一些体验、测试这些工作。我们程序的具体实现、编程是由我们一个明眼人朋友来做。

InfoQ： 那鉴于您的产品有一定产品的特殊性，还有您所面对的用户、用户群众也有一个特殊性，针对这两个特殊性，想请您说一下，你的团队在开发和测试的过程中，同其他的一些团队有哪些不同，或者说有自己的哪些特点？

杨永全：首先因为我们这个团队开发的产品是给盲人使用的，也就是说它的使用群体就是受到限制，只是这个视障人、盲人用户，所以我们的测试也会受到很多限制，所以我们就用了一个最简单的办法，直接是我们的用户承担了测试的工作。一个产品的版本开发完成，首先会在内部进行一个初步测试，通过后会提交给用户继续做测试，稍后用户再反馈意见，我们再进行修改。这样经过几个回合之后，基本上一个版本就可以完成了。

InfoQ： 在您产品的设计中提到了信息无障碍的概念，你能针对这个信息无障碍这个词，简单的介绍一下吗？

杨永全：信息无障碍（Information Accessibility），指的是能够让所有的人群，包括老年人、儿童、残障人，各种人群都能够方便的、平等的获取和使用互联网上的信息。需要强调的一点是所有人群，包括老年人、儿童、残障人，包括盲人、聋哑人、肢残人，还有智残人等等，都能够很方便的去获取到任何有用的信息，这就是信息无障碍的一个基本的概念。

InfoQ： 还想请您再说一下无障碍设计和普通的交互设计在具体的设计环境中有那些不同，这两者是否可以有一个有机的一个融合，比如说无障碍设计可以给普通的交互设计有一些启发或者说指导？

杨永全：其实我觉得这个无障碍设计方面，其实应该属于普通交互设计的一部分，而且现在成为了被忽略的一个环节，事实上，就在交互设计的之初，就应该考虑到无障碍设计，比如说一个产品，需要考虑它的受众，不仅仅是常规人群，还有一些特殊人群，比如说我们的一些软件，很多人在关注鼠标操作的同时，可能就忽略到了键盘操作，往往这一块，就可能会给一些其他人群，包括残障人，老年人，或者说其他人群带来一些的困扰，这个在设计之初，如果想到的话，在充分设计的前提下，就会使产品变得很完善，所以无障碍设计跟普通设计是不冲突的。

InfoQ： 这点您能举一个具体的例子吗，就拿咱们日常生活中用到最多的微博来说，你觉得就是，刚刚在跟您在沟通过程中，您也提到了，就是如果在使用微博的过程中，做了哪些改进，或者提升了哪些设计上的理念，就能够让用户更好的，或者说就实现了这种无障碍的这种方法？

杨永全：这个呢，其实这就是一个很小的一个体验，就比如说像今天刚才采访吧，那么我是在坐在这了，刚才你又把我送过来，那么你告诉我，前面是一个椅子，你可以坐下来。但是

呢，对于我来讲，我就不知道前面在哪，那么这个椅子距离我有多远，那如果你扶着我的手，你把我手放在这个椅子上，那我知道了，椅子在这，我就可以坐下了。其实是一个很细的一个动作，那么就解决了一个很大的问题。那么就是说这个微博其实也是这样的，就说很多时候，咱们都忽略了这个键盘操作，那么你们用微博的话，用鼠标去点的话会很快，很方便。那么说，你如果鼠标一旦鼠标坏了的话呢，那这个时候如果你用键盘的话，能不能一样去操作呢？

InfoQ： 那比如说如果要让您设计这样一个更好的交互这样的微博，您觉得从哪些方面可以改善一下？比如说就是不用鼠标，就只能用键盘操作？

杨永全：其实现在我觉得就微博的体验来讲呢，有这么几个方面，一个就是转发和评论，这两个最常用的功能。细微之处在于，当转发完成之后的焦点处理，键盘焦点会返回到哪里去？这个体验就显得很关键。因为如果用键盘的话，比如说当前页面有十个微博，当转发完第五条微博之后，想再继续看第六条，现在很多的体验是需要让你从第一条看起，从第一条再往后找，如果在看完第五条微博时将焦点保持住，接着从第五条往后找，这样体验一下就提高了，操作速度和时间也会提升很多。

InfoQ： 就拿盲人来说，最大的时间往往是花费在转发完一条微博到查找到下一条微博上，如果通过这些焦点的细节上的处理呢，能够把这个时间大大的缩短，从而在体验上也是有一个很好的提高。

杨永全：对于现在的微博来讲，实际上现在不是不能用，而是这种使用效率上的低下。

InfoQ： 最近讨论的比较热门一类软件，我们称之为读屏类软件，请您介绍一下读屏类的软件，基本的技术原理是什么？

杨永全：读屏软件顾名思义，就是朗读屏幕，屏幕上显示什么内容，软件就会通过语音朗读出来，这是最基本的功能。技术上主要是通过一些无障碍的接口，包括那些 MSA 或者其他一些方式去获取信息，然后朗读出来，我认为主要包括两个层面，一个是对鼠标操作的一些提示，另外是对键盘操作的一些提示。实际上用户的每一次操作，包括鼠标的点击，你这个键盘的按键操作，都会通过语音来给用户进行反馈。

InfoQ： 我从互联网上也了解到，您目前正在从事一款产品，叫“争渡读屏”的产品的开发。现在想请您介绍一下争渡读屏这款软件，在读屏类软件中有哪些优势，它目前这个产品主要取得了哪些进展，主要适合于哪些场景？

杨永全：这样，读屏软件在这个国内来讲主要有永德、阳光还有争渡，那么这三款是比较主流的读屏软件。争渡读屏是一个相对发展比较晚，到目前只有三年的时间，到目前为止它取

得的这个进展和这个成绩应该说是很不错。从功能上来讲，与其他两款也是相差无几，甚至在很多地方也超越了它们的一些功能。现在我们的盲人朋友，通过我们的争渡读屏软件，来进行包括上网看信息，玩儿微博，QQ 聊天，甚至包括在淘宝开店，购物，还包括炒股，都是没有问题的。

InfoQ： 争渡读屏最近发布了公益版，是什么让您有这种想法，要把这个软件做成一个公益版呢？

杨永全：其实做公益版这个软件，是我一直以来的一个梦想。因为作为一个盲人，体验们互联网的生活是个很痛苦的过程。因为之前为盲人服务的产品很少，加上功能上也很不理想。所以后来我就联系到一些朋友一起，做了这款争渡读屏软件。做了三年之后，我发现，其实虽然产品价格不是很高，但仍旧有很多朋友承受不起，所以我们觉得，我们有责任能让更多的盲人朋友能够有机会用上读屏类软件，能够享受到互联网的这种精彩生活，所以经过考虑之后决定推出这样一款免费的公益版本，而且这个免费版本，我们之所以叫做公益版也是希望通过我们的行动，来进行一个呼吁，希望有更多的爱心人士能够参与到关注，关爱残疾人，盲人的这个事业当中来。

InfoQ： 在发布了公益版之后，有没有一些数字上的，比如说用户数量上突增的这么一个现象？

杨永全：今年 5 月 15 号晚上，在网上开的发布会，当天到场的听众有三百多人。这个数量在盲人的群体里边，也称得上是一个非常大的参与的活动，第二天的下载量超过了一千八百次。截止到目前，几个月来，经过我们初步的统计，每天同时在使用的用户数将近两千人。这个数字，在盲人能够上网的用户当中，也已经是一个比较大的比例了。

InfoQ： 最后，还想请您再介绍一下，就是目前咱们的这个读屏的软件，或者说无障碍设计这块，遇到的最大挑战是什么？包括在您的开发过程中遇到了哪些比较棘手的问题，您最后是怎么解决的，未来的一个发展方向是什么？

杨永全：其实从读屏软件角度来讲，障碍主要来自于越来越多的软件开发方法，它的不规范性和它的非常强的这种自定义性，这个就是说，现在软件会尽量做得非常漂亮，但是这么做的结果就是这种漂亮的界面，很多都是通过一些自定义的框架，或者说公司内部自己开发的框架，那么这种框架往往就忽略掉了无障碍设计这方面，也就没有提供一些无障碍的接口，导致读屏软件几乎就获取不到他们的信息了，这样的话读屏软件就很难去操作它，这是我们现在碰到的最大一个困难。目前解决应该是有以下几个方面，一个是软件开发公司本身在无障碍设计意识上的提高，他们会去关注这个问题。另外，也还是希望将来会有一些法律法规

方面的规定去规范这些方面，当然还是需要我们这个读屏软件从本身的角度来增强我们的功能，加强产品的功能。

InfoQ： 那您觉得无障碍事情这块在未来会有一个什么样的发展？

杨永全：其实无障碍设计，我觉得在咱们国内虽然已经提了很多年，但是真正被重视起来，还是刚刚起步，所以未来的呼吁有越来越多的公司，或个人会关注这个领域，应该说将来也会有一个非常好的改善。所以我对将来还是有信心的。

InfoQ： 那您觉得技术上还会有一些什么样的变化？

杨永全：技术上来讲，我是觉得还是需要这个多交流，多沟通，就主要是通过什么？这个读屏软件开发者和这个软件开发者实际要多交流、多沟通。另外就是要提高技术人员的这个无障碍的意识。

InfoQ： 好，那最后非常感谢杨永全接受我们的采访，谢谢。

杨永全：谢谢。

原文链接：http://www.infoq.com/cn/interviews/yyq-barrier-free-design

InfoQ
Enterprise Software Development Community

时刻关注企业软件开发领域的变化与创新

Java社区：企业Java社区的变化与创新

.NET社区：.NET相关的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和Rails

SOA社区：关于企业内面向服务架构的一切

敏捷社区：面向敏捷软件开发的项目经理

架构社区：设计、技术趋势及架构师所感兴趣的话题

运维社区：关注IT运维中的存储、监控和设计

官方微博：http://weibo.com/infoqchina
豆瓣小组：http://www.douban.com/group/infoq
讨论组：groups.google.com/group/infoqchina
编辑电邮：editors@cn.infoq.com
商务合作：sales@cn.infoq.com

热点新闻 | News

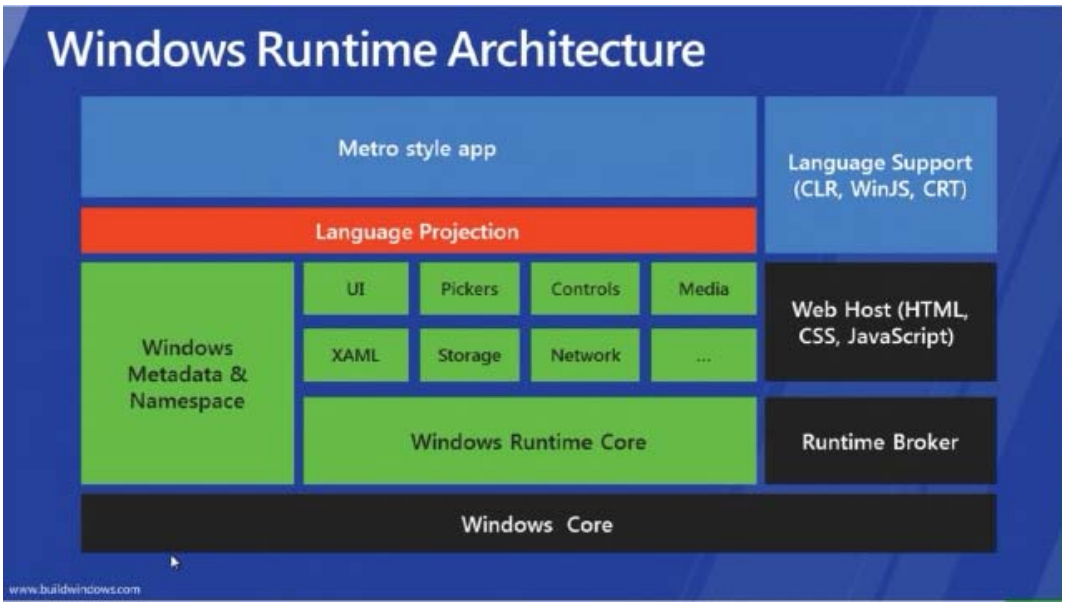
Windows 运行时的设计细节

作者 Abel Avram 译者 高翌翔

创建 Windows 运行时（WinRT）是为了在 Windows 上给用户提供一种流畅且安全的应用体验。WinRT 会受到.NET、C++、以及 JavaScript 三者的影响。WinRT 不会取代 CLR 或 Win32，而是为那些使用不同语言编写的应用程序提供统一支持，以便它们可使用新的 Metro 风格用户界面运行于 Windows 之上。

两年前，微软从一个愿望开始了 Windows 运行时（WinRT）的研发工作，此愿望是，建立一个更好的开发平台，让开发者在此平台上使用具有丰富智能感知功能和优秀调试功能的工具来创造快速、流畅、可靠的应用程序，而且开发者可自由选择所使用的语言和库。最终结果是产生了一种架构以及一组可以从.NET 语言（C#、VB.NET、F#）、C++、以及 HTML/JavaScript 调用的 API。所有这些语言都会对 WinRT 的设计产生影响。

WinRT 不是为了取代.NET 或 Win32 提供的所有功能，但是它是一个公共平台，以便那些使用不同语言编写的应用程序可使用新的 Metro 风格界面来 运行。当混合 C#应用程序基于 WinRT 创建 Metro 风格用户界面时，程序中将仍能执行 LINQ 查询，对于存储、网络、新式应用程序的安全性等方面同样 能执行 LINQ 查询。完整的运行时架构如下图所示：



语言投射（Language Projection）表示对于每种已支持语言的 WinRT API 视图。通过 Visual Studio 11 的智能感知，在“Windows”命名空间下可找到那些推荐的用于创建 Metro 风格应用程序的 API。

在 BUILD 开发者大会上，Windows 运行时开发经理 Martyn Lovell 通过题为“包围 Windows 运行时”的演讲介绍了 WinRT 背后的设计原则：

- 任何耗时超过 50 毫秒的事情都应该通过使用了 Async 关键字的异步调用来完成，以确保流畅、快速的应用体验。由于即便当异步调用的情况存在时，许多开发 者仍倾向于使用同步 API 调用，因此在 WinRT 深处建立了使用 Async 关键字的异步方法从而迫使开发者进行异步调用。
- 应用程序彼此之间被更好地隔离开，从而使得一个应用程序的性能不会影响到另一应用程序，同时也是为了获得更好的安全性。隶属于某个应用程序的运行时对象不能被暴露给另一应用程序，除非通过借助标准的操作系统级的通讯信道 Windows 契约（Windows Contracts）来完成。
- 基于平台的版本控制（Platform-based versioning）确保应用程序在不同版本的 Windows 上运行良好。版本控制信息包含在 WinRT 元数据中，而且智能感知会根据应用程序的目标版 本来公开功能，因此开发者无需查阅其他文档就能知道，对于某个特定版本的 Windows 而言，到底有哪些类和方法是可用的。

关于类型，WinRT 必须提供语言无关的类型——integer（整数）、enumerations（枚举）、structures（结构）、arrays（数组）、interfaces（接口）、generic interfaces（泛型接口）、以及 runtime classes（运行时类）。引入了被称之为 HSTRING 的新字符串类型，该类型允许在不进行任何数据复制的情况下，在应用程序与运行时环境之间传输字符串。

每个 WinRT 对象都会对应一些接口，其中有两个接口属于每个对象：IUnknown 接口，熟悉的 COM 接口；以及 IInspectable 接口，用于根据对象所包含的元数据来发现有关该对象的信息。一个对象可能通过接口提供其他功能，然而运行时类会把这些接口集中公开出来。例如，一个 FileInformation 对象拥有由 FileInformation 类公开的 IStorageItemInformation、IStorageItem、IStorageFile 三个接口。

WinRT 对象在编译时被公开给 C++应用程序，而对于 C#或 VB.NET 应用程序而言，对 WinRT 对象的绑定一部分是在编译时完成的，另一部分则是在运行时完成的。HTML 或 JavaScript 应用程序只有在运行时可以看到 WinRT 对象，而且元数据是动态生成的。

Metro 界面运行在一个不可重入的单线程之上，然而应用程序的其余部分可以从线程池中使

用由运行时环境所自动提供的多线程。

Windows 运行时体验团队的 Harry Pierson 和公共语言运行时团队的 Jesse Kaplan 在 BUILD 开发者大会的另一题为“在 C#和 Visual Basic 中使用 Windows 运行时”的演讲中，介绍了一些使用.NET 语言对 WinRT 进行编程的细节。

据 Pierson 透露，.NET 对于 WinRT 的重大影响在于，许多设计准则被从.NET 中借用过来。例如，通过使用基于.NET 元数据格式更新版本的元 数据增强了 WinRT 库。就像 Silverlight 一样，为了创建 Metro 风格应用程序，WinRT 会使用 XAML 框架。由于在运行时与.NET 之间 存在直接映射：基本类型（primitives）、类（classes）、接口（interfaces）、属性（properties）、方法 （methods）等等，并且开发者无法看到那些存在的差异，因此使用 WinRT 的.NET 应用程序将会有宾至如归的感觉。

Pierson 还表示，开发者可以用 C#语言创建可供 C++或 JavaScript 的 WinRT 应用程序使用的 Windows 运行时组件，然而须要遵守一系列规则：“结构体只能拥有公共数据字段；只允许对 XAML 控件使用继承，其它类型都必须使用 sealed 关键字；只支持系统提供的泛型。”

在 Windows 8、或是后续版本的 Windows 中将提供一种经典应用程序与新的触摸友好的 Metro 风格应用程序共存的混合环境。基于 Metro 风格的未来的 Windows 应用程序将受益于 Windows 运行时所提供的公共基础设施，开发者必须针对一套唯一的 API 进行编程，而对于不同语言会略有差异。在与过 去保持兼容性的同时，又为未来提供新功能方面，这是微软所做的最好尝试。

原文链接：http://www.infoq.com/cn/news/2011/09/Design-Details-Windows-Runtime

相关内容：

- Windows 8 将替换 Win32 API
- C++组件扩展：COM 的新面孔
- 微软将与 Joyent 合作将 Node.js 移植到 Windows 上
- System-on-a-Chip 上的 Windows
- 微软缘何认为 VB 与 C#需要异步语法

热点新闻 | News

Google App Engine 涨价，开发者深受打击

作者 Charles Humble 译者 贾国清

五月， Google 宣布即将调整旗下的云计算服务 App Engine 的收费标准，并在今年的晚些时候推出以 PaaS 为核心的产品。据 Google 称，价格调整将在九月中下旬进行，同时会取消产品的“预览”标记。Google 会提供一款工具来帮助用户计算相关的服务费用。

Google 同时提到， 届时仍会为当前活跃应用保留一定免费的配额，但大部分的付费用户将发现费用有所上升，作为回报，在提供更好的服务同时，Google 还会与付费用户签订服 务等级协议（SLA，Service Level Agreement），以及与高级用户签署关于运维支持的服务等级协议。

在 Google App Engine 官方论坛上，客户均表现出了很大程度上的消息态度。Google 论坛中一份 App Engine 前后价格对比的报告表明，调整后开发者的成本增加了 50%、100%甚至更多。还有一部分开发者迫于高额的价格则不得不放弃 App Engine。

无独有偶，同时也是 App Engine 用户的诺基亚研究员 Russell Beattie 也表达了对新收费标准的不满，他开发的基于 App Engine 的应用每日费用将从 2.63 美元上升到 34.38 美元，这一数字，在十一月五折优惠期过后还将会翻翻。

名为 Ugorji Nwoke 的开发者在博客中这样写道：

> Google 的这种调整 App Engine 收费标准的做法相当可笑，严重打击了开发者热情。在这样下去，不仅 Google 的品牌形象会受到影响，甚至还会失去开发者对其的信任。

VMware SpringSource 部 Groovy 开发负责人 Guillaume Laforge 写道：

> 最大的问题出现在“前端实例用时（Frontend Instance Hours）”的花费上。在新的收费标准下，一个不间断运行的、访问量较低的应用，在保证前端实例一直运行的情况下每月的费用将会在 30 美元左右。

同时，很大部分的开发者正在考虑转向其他的云计算服务，比如亚马逊的 EC2 和 VMware 的 Cound Foundry。另外，与传统 Java 服务托管商比起来，就更加显得有趣。例如，同样是

29 美元/月的花费，Kattare 可以提供独立的 Apache Tomcat 的 JAVA 虚拟机（JVM），且此虚拟机具有 256MB 的堆空间，每月高达 50GB 流量以及全面的支持。尽管 Kattare 的服务中没有像 Google 那样特有的 API 和基础设施，但其提供的服务一直被开发者所关注。一位用户在 Google 官方邮件组中写道：

> App Engine 中最令人担忧的是什么？是被套牢，是被 Google 任意摆布。尽管也有 TyphoonAE 这种服务，但其始终无法成为 App Engine 的替代品。
>
> Google 想要做什么？他们正在做的正是令人批判的，我们这群 GAE 使用者最不希望看到的事情，那就是不断的压榨我们。
>
> App Engine 的没落并不是由于我们即将投奔 EC2，而是正如那些使用 App Engine 的，仍处在担忧中的人们所亲眼看到的那样，价格的调整以及一直争论的封闭政策所带来的弃用 GAE。可以预见的是，新应用的数量会减少，最终，Google 将会发现 GAE 不仅没有成功而且还接结束其短短 3 年的生命。

尽管 Google 在 App Engine 的官网表示正在为为盈利性组织、教育机构以及开源项目等寻求更好的解决方案，但其至今仍未对外界的不满做出正面的回应。

原文链接：http://www.infoq.com/cn/news/2011/09/app-engine-price-hike

相关内容：

- 基于用户反应，Google 调整 GAE 价格条款
- Google App Engine 开始支持 Go 语言
- Google 期望使用 Chrome 和 GAE 商用版在企业级市场中更进一步
- 测量和比较 5 种云平台的性能
- 云计算的虚拟研讨会

热点新闻 | News

用户抱怨 Windows Azure

作者 Abel Avram 译者 郑柯

Appirio 从 Windows Azure 切换到了 Salesforce.com 的 Database 产品，他们提到了部署方面的困难，以及使用 Web Roles 和 DBA 的额外开销；同时，开发人员 Adron Hall 抱怨 SDK、价格和系统管理方面的问题。

Appirio 是 IT 和云咨询服务提供商，最近发布了 CloudSpokes，这是一个 web 门户，供开发人员就云相关的项目展开竞争，赢取奖金。他们最早使用 Windows Azure 服务开发了这个项目，但之后就切换到了 Database.com，这是 Salesforce.com 的数据库，可供云上任何用户使用。纽约时报公布了一些这次技术选型变更的细节。

提到的第一个问题，是关于一个众包项目的部署：

> CloudSpokes 的社区架构师 Dave Messinger 提到：部署过程非常痛苦，特别是考虑到 CloudSpokes 要构建自己的站点。因为 CloudSpokes 有众包的站点开发任务，使用自己的竞赛来吸引全世界的开发人员，因此这个过程中任何复杂度很快就会变得让任务难以为继。

Appirio 的首席战略官 Narinder Singh 认为：Azure 的 Web Role 缺少真正的平台支持。

> Singh 认为整个使用过程的特点很接近 IaaS，因为还要处理底层细节，而这不符合像 Windows Azure 这样的 PaaS 厂商的市场宣传。

但是，Appirio 迁移到 Database.com 的主要原因，看起来一直是 Azure 对数据库管理水平的高要求：

> Windows Azure 需要一定水平的数据库系统管理能力，要了解背后的机制，而这是 CloudSpokes 不想着手的。它想把重点放在前端和其他关键的业务方面，而不仅仅是 DBA 的工作。因此，自从了解了 Database.com 之后，Messinger 和 Singh7 月中旬就开始切换，并再也没有回头。

离开 Azure 的整体效果可归结为：效率提高、减少所需开发人员数量，以及整体上更快的项

目交付速度：

- 用 Windows Azure 需要 7 个全职开发人员，现在只要一个
- 第一次产品部署只用了一个月，相对 Windows Azure 估计需要 6 个月
- 估计两个半月上线，相对使用 Windows Azure 估计需要 7 个半月上线

另一个例子是 Adron Hall 提供的，他是 Russel Investments 的资深应用开发人员，他写了一篇博客文章，详细描述了使用 Windows Azure 的优劣之处。他还特别列出关于 SDK、价格和系统管理工具方面的多个问题：

> 好吧，我要被 SDK 逼疯了。它一直都有很多错误，sealed 类型的（糟糕）代码，而且与 Development Fabric **紧紧耦合**在 一起。我很职业，我可以 mock 这些东西，不需要像幼儿园的小朋友一样手把手地教我怎么做！如果我有一个很大的环境，有几千个 prospective 节点 （或是几十个实例），development fabric 不会有任何帮助。在企业环境中开发大规模应用，Windows Azure 是最难使用的平台，我会将 SDK 封闭的本质（sealed，没有接口）和 development fabric 视为头号原因。
>
> Windows Azure 是目前市面上价格最高的云平台，或者说基础设施。AWS 的一些价格可以排在第二或是第三，只相当于 Azure 的六分之一。Rackspace 在某些情况下的价格低得离谱，相同的处理能力，只有 Windows Azure 的八分之一。 我知道 Windows Azure 提供一些比较特别的东西，而且在某些很少见的特定情况下也许更便宜，但**真的是很少见**...
>
> Silverlight Interface 界面很漂亮，这个我认可；但在除 IE 之外的大多数浏览器中，它就变得稀奇古怪了。噢，等一下，我错了。它在所有的浏览器中都是稀奇古怪的！郁闷！现在这可能已经修复了，但在我和其他工作伙伴的经验里，我们在 Chrome、Opera、Safari、Firefox 还有 IE 里面都见到过问题。比如：启动实例时，它总是在转，貌似在启动，它转来转去，等刷新完成后，实例却完全消失了。我之前刷新了 Silverlight 的 UI，它就停止响应前面的通信请求了（这甚至不是在我自己的机器上）。
>
> 不管是互联网环境，还是 Web 开发，或是其他什么，实例的启动时间都是完全不可接受的。启动时间应该接近 Linux 物理服务器。我不管启动时应该做什么，但是实例应该清理干净，架构应该变化，如果需要的话，还要完成文件交换。我不关心云运行在什么操作系统之上，但是我的实例应该在 1 到 2 分钟、甚至更短时间内启动起来。Rackspace、Joyent、AWS，还有其他所有云供应商都能在 45 秒左右启动起来，有时要 1 分钟，可常常不用那么久。

Hall 继续指出了 Windows Azure 拥有的一些好特性，比如平台支持、.NET、PHP、Ruby on Rails 生态系统、SQL Server、服务总线、访问控制、Azure Marketplace、SQL Azure 等。可他还是以负面的评论结尾，基本上就是说 Azure 不能胜任。

> Windows Azure 从 beta 版本发布后，已经成长、成熟了很多。但是相对更成熟的解决方案，它还是有些大问题。不过，选择 Windows Azure 的人还是能看到一线曙光，或者是那些“被选择” Windows Azure 的人们……
>
> 我确实能想见自己在将来用 Windows Azure，也许不会很频繁，但是会用的。

我们想知道：其他用户使用 Windows Azure 的经验如何？您面临什么情况？使用 Windows Azure 带来的好处能超过不便吗？还是恰恰相反？

原文链接：http://www.infoq.com/cn/news/2011/09/Some-Users-Complain-about-Azure

相关内容：

- 基于 Windows Azure 的云计算应用设计
- 什么是 IronRuby？开发者如何在 Rails 中使用它？
- AppFabric 新特性：队列、主题和订阅功能现已发布
- 企业开发库即将支持 Windows Azue 应用的自动扩展功能
- Windows Azure Tools 1.4 中的多项新功能

热点新闻 | News

Mono 将不会具备 Metro UI

作者 Abel Avram 译者 郑柯

Miguel de Icaza 提及：Xamarin 将不会把 Metro 导入其他平台，这会是 Linux 在桌面上失败的原因之一。对开发跨平台应用感兴趣的.NET 开发人员将可以使用 Mono 编写业务代码，并为其他平台重写 UI 代码。

为了确保不会误解 Xamarin 对于 Mono 和 Windows Runtime（WinRT）UI（Metro）的计划，Miguel de Icaza 在博客上宣称： 他们"不会为 Linux 开发 WinRT UI，而且也没有计划要这么做。"De Icaza 认为：有可能使用一些 Moonlight 代码将 Metro UI 带入 Mono，但是他觉得这么费劲不值得，表示出他对于 Linux 的怀疑，认为 Linux 可能无法跟上"其他消费者环境的成长"。

同一时间，Tim Anderson 在博客中提到 de Icaza 的一些话，这是在 BUILD 2011 的私人谈话中发生的。Mono 的创始人表示了他对桌面 Linux 当前状态的担心，而且直接暗示了 Mono for Linux 的走向：

> 老实话，对于桌面 Linux，开源的好处一直在对抗它，因为我们一直在出问题。不只是 Red Hat、Ubuntu、Suse 之间的不兼容，甚至同样的发布版本都存在问题。Ubuntu 从这周起就不在于 9 个月之前的版本兼容。而且还有多个版本，KDE 版本、Gnome 版本，拥有新启动系统的版本。
>
> 如果你算一下在 Linux 上有多少出色的桌面应用，大概也就 10 个左右。你使劲儿想想，也许能说出 20 个。这一路上，我们在每一步都成功地激怒了开发人员，API 总是出问题。
>
> 我的心都碎了，这是底线啊……
>
> 我想 Linux 在桌面上面临着困难，而且桌面也已经开始不再那么重要了。

所以，现在很清楚：我们不会看到跨平台的 Metro UI 了。这与 Mono 的方式相同，Mono 不打算把 WPF 导入到其他平台，因为需要太多资金和工作量支持。但是，如果 Mono 打算跨平台，使用什么样的 UI 呢？De Icaza 提出如下建议：

- Windows 平台：WinRT、 Winforms、WPF（替代品：GTK#、Silverlight）
- MacOS 平台：MonoMac（替代品：Gtk#、Silverlight）
- Linux：Gtk#
- Android：MonoDroid API
- iOS：MonoTouch
- Windows Phone 7：Silverlight
- XBOX 360：基于 XNA 的 UI

为了编写跨平台应用，针对这么多平台要使用这么多不同的 UI，让人怀疑 Mono 是否还具有吸引力。De Icaza 认为 :在应用的业务代码和 UI 之间有明确分隔还是有价值的 ,而且 Mono 的业务代码可以在所有这些平台上运行，开发人员只需重写 UI 部分。这样更好，因为原生 UI 代码让应用看起来更好，使用跨平台 UI 的要差些。

但是 Web 开发框架在崛起，比如 jQuery、jQuery UI、jQuery Mobile、PhoneGap 和 Sencha，整体上有向 HTML5 和 web 技术发展的趋势，Metro 和 Windows 8 也强调了这个方向，这些让人开始思考 Mono 的空间还有多大 ,还有多少开发人员会选择 Mono 作为自己的跨平台解决方案。对于投入.NET 的开发人员来 说，开发跨平台应用 Mono 还有吸引力，但是这样就足够了吗？

原文链接：http://www.infoq.com/cn/news/2011/09/Mono-Metro-UI

相关内容：

- Wally McClure 谈 MonoTouch 与 Mono for Android 的未来
- MonoDevelop 发布 2.6 版本，支持 Git 和和 Mac 开发
- Xamarin 发布了其 MonoTouch 的首个版本
- Mono 和.NET：Medtronic 的 iPad 应用背后的秘密
- 在 MonoTouch 中自定义表格

热点新闻 | News

“Apache 杀手”一种利用 Range HTTP 头域的 DDoS

作者 Jean-Jacques Dubray 译者 黎伟

2007 年，Google 工程师 Michael Zalewski 在研究了 HTTP/1.1 Range 头域的实现后，发表了一篇备忘，详细说明了 Apache 和 IIS 网页服务器中存在的一个潜在漏洞。

> 在我印象中，一个单独的、短的请求就可用来诱使服务器无目的地地发出数以 GB 的伪数据，而不顾服务器文件大小、连接数以及管理员设定的持续（keep-alive）请求数的限制。

8 月 19 号，“Full Disclosure”安全邮件列表中发布了一段 Apache DDoS 工具原型的 Perl 脚本。8 月 24 号，Apache 安全小组就此发表了一篇备忘进行解释：

> 它最常见地表现在静态内容生成且通过 mod_deflate 模块进行压缩的时候，但其它在内存中缓存或生成内容的模块也可能会受到影响。这是一种非常普遍（缺省的，不是么？）的配置方式。
>
> 攻击可以远程进行，并且中等数量的请求就会导致内存和 CPU 占用率显著提高。
>
> 我们注意到该工具已经得到越来越多的应用。
>
> 目前还没有修复该漏洞的补丁和新版本的 apache，因此本公告将在长期修复补丁发布时更新。修复补丁预计将在接下来的 96 小时内发布。

星期五，Apache 发布了第二则公告， 其中他们解释了当服务器处理请求以返回多个（重叠的）范围时，Apache httpd 进程和它所谓的内部“桶队列（bucket brigades）” 是如何按照请求的顺序进行处理的。单个请求可以请求非常大的范围（例如从字节 0 到结尾），差距达到 100 倍。目前，这种类型的请求内部会扩展相当于 100 次大型的取指令操作，所有这些都以低效率的方式保存在内存中。

> 处理方式有两种，要么使事情更有效率，要么清除或者简化被认为过于笨重的请求。在最终修复方案出来前有几种直接选择可缓解这一状况。

Apache 提出的缓解措施包括从完全禁止 Range 头域到限制请求包的大小及部署定制的 Range 计数模块。Lori MacVittie 详细说明了如何通过使用 Big-IP 来实现缓解措施。

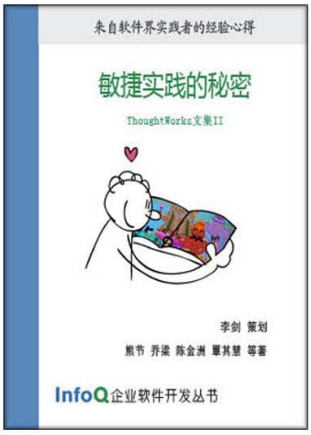
原文链接：http://www.infoq.com/cn/news/2011/09/apache-killer

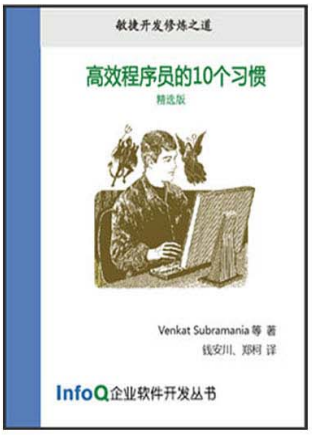
相关内容：

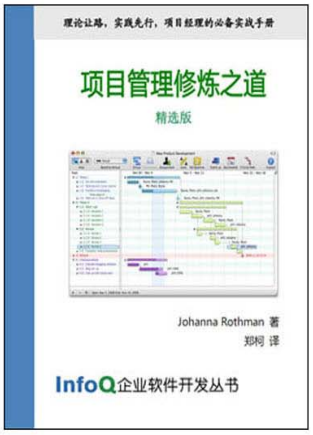
- REST 服务开发实战
- Web 是否应该被加密？
- World Wide Web 二十周年纪念
- 关于构建可进化的系统
- 加密因特网

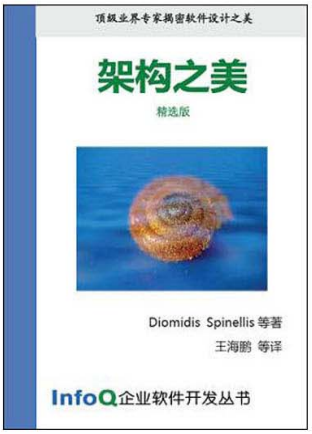
InfoQ企业软件开发丛书

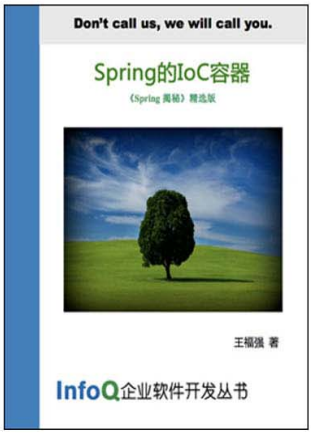
欢迎免费下载











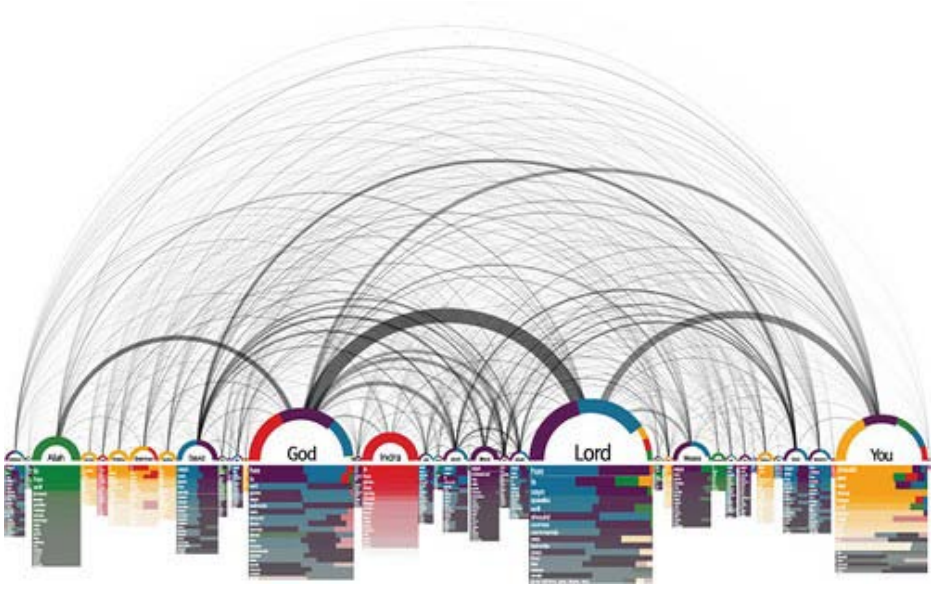
商务合作：sales@cn.infoq.com　　读者反馈/内容提供：editors@cn.infoq.com

特别专题 策划：InfoQ 中文站； 执行：李明

大数据时代

就在你看这句话的的短短数秒间，已有数以亿记 GB 的数据又生成了出来。是的，现在是一个数据爆炸的时代，过去3年里产生的数据量比以往4万年的数据量还要多。据美国《经济学人》2010年报道，沃尔玛的数据量是美国国会图书馆的167倍。eBay 的分析平台每天处理的数据量高达100PB，超过 Nasdaq 交易所每天的数据处理量。根据 IDC 的预测，2020年电子数据存储量将在2009年的基础上增加44倍，达到35万亿 GB。截止到2010年，电子数据存储量已经达到了120万 PB，或1.2ZB。大数据的浪潮正在席卷整个世界。

大数据不仅仅是数据量上的增长，数据的形态也在发生着根本性的改变。据调查显示，非结构化信息（比如文件、电子邮件和视频）将占到未来10年新生数据的90%。这也就意味着，传统的数据库和数据架构根本无法支撑如此迅猛的增长，技术的变革势在必行。如今，已经有越来越多的企业认识到了数据的价值。IBM 的调查报告中称，已经有83%的 CIO 拥有涵盖商业智能和数据分析的远期计划。大数据时代带给我们的不仅仅是一个概念，而是一种重新认识数据的机会。谁可以把握住这个机会，成为这个时代的宠儿，我们拭目以待。



大数据，下一个新领域　　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

特别专题

大数据，下一个新领域

作者 Boris Lublinsky 译者 孙爱鸿

上个月麦肯锡全球研究院（MGI）发布了一份报告——《大数据：创新、竞争和生产力的下一个新领域》，这份报告研究了数字数据和文档的状态，同时讲解了处理这些数据能够释放出的潜在价值。

该报告深入研究了五个领域，来观察大数据是如何创造出价值的和研究大数据的变革潜力。这五个领域包括美国医疗卫生、欧洲联合公共部门管理、美国零售业、全球制造业和个人地理位置信息等。根据这份报告，这五个领域不仅代表了广泛的全球经济的核心领域，也能说明一系列区域性的观点。当然，在对大数据使用的成熟度和复杂度上，这五个领域又各自不同，因此也能提供不同的商业经验教训。

通过对这几个领域的详细地分析，该研究提出了以下五个可广泛适用的方法，来利用大数据的变革潜力创造价值，同时这些方法也暗示了如何去设计、组织和管理个体组织。

- 创造透明度

> 让相关干系人更容易地及时获得大数据，能创造巨大的价值。实际上，创造价值的这一点是所有其他级别的前提条件，而且它是最直接的可以抓住该潜力的方法。在很多场合，机会存在于那些创造数据透明度的动机不对称，如缺少绩效驱动的地方。

- 通过实验来发现需求、呈现可变性和增强绩效

> 组织的装备世界——部署能够收集数据的技术——和感知世界的能力在持续地增强。越来越多的公司在以数字化的形式存储大量非常详细的商业交易数据。因为可以访问这些数据，并且有时还可以控制数据生成的条件，所以决策的方法就可以截然不同，也就是说将更加科学的方法引入到管理中——例如，将经典的科学方法应用到管理的实践中。特别是，经理们现在可以使用控制实验的科学流程，包括特定假设的公式等，来设计和实施实验，经过严格地分析定量的实验结果后再做出决策。

- 细分人群，采取灵活的行动

> 利用大数据，组织可以创建更窄的细分段、更精确地精简服务来满足顾客的需求。这种方法在市场和风险管理方面比较知名，但是也可以借鉴到像公共部门管理的地方。

- 用自动算法代替或者帮助人工决策。

> 精密的分析能够实质性地优化决策、减少风险以及发掘有价值的观点，反之则可能一直被忽视。大数据也能提供用于开发算法或者算法需要操作的原始数据。当今的基于大数据的分析包括基于规则的系统、统计分析和机器学习技术如神经网络等。

- 创新商业模式、产品和服务。

> 因为有了大数据，所有类型的企业都可以创建新产品和服务、改善现有的产品和服务以及发明全新的商业模式。

这份报告在互联网上引起了强烈的反响。经济学家 Schumpeter 说：

> 在一份用事实适当包装的新报告中……MGI 指出数据正在成为像有形资本、人力资本这类产品的一个因素。能够利用大数据的公司将会藐视数据利用无能者。数据 资本（自造一个词汇）将和品牌资本一样重要。MGI 强调这并不是无聊的未来学：商业已经在适应大数据了。数据革命正在瓦解已经建立的产业和商业模式。IT 公司率先嗅到了进入医疗卫生市场的方法：谷歌的 Health 和微软的 HealthVault 都能让消费者跟踪自己的健康情况以及记录治疗情况。制造商们也 正在加速转型为服务型的公司：通过传感器，他们可以监控其产品状态，从而在产品破坏之前就能早早判断出是否需要维修它们。

Richard Maddox 也在博客中写道：

> 该报告描述了已经进入到每一个部门和经济领域的数字型数据的状态和成长中的角色。充分的证据表明大数据能显著地为国民经济做出贡献，它为整个世界经济创造 实质性的价值。然而，MGI 也提到了组织在获得大数据全部潜力之后所面临的一些挑战，如能让大数据对商业更有利和更有价值的分析和管理人才还比较有限。

最后，Stowe Boyd 评论道 ：

> MGI 的分析说明了企业和政策制定者们必须克服面临的艰难险阻，去充分地抓住大数据带来的机会。仅美国就面临 140,000 至 190,000 分析和管理人才缺口，和 150 万具备理解和基于大数据研究做出决策的经理和分析师人才缺口。企业和政策制定者必须克服这些关于隐私安全、数据访问、技术部署等问题的不对称诱因。

每天随着越来越多的信息被收集和存储起来，大数据正呈爆炸式增长。MGI 估计 2010 年全

球的企业在磁盘上存储了超过 7EB（艾可萨字节）的新数据，而消费者则在个人电脑和笔记本（以及移动设备）等设备上存储了超过 6EB 的新数据。如果能够正确地使用，这些数据将能显著地促进当今经济的增长。

原文链接：www.infoq.com/cn/news/2011/06/BigData

相关内容：

- Platform 创始人王敬文谈云计算和大数据
- 林昊谈 HBase 技术在淘宝中的应用
- 向上扩展或向外扩展？还是两者兼顾？
- 支持大数据的 JasperSoft 4 发布了
- NASA 的 OODT 被选为 Apache 顶级项目

特别专题

大数据时代的数据管理

作者 刘庆

不知怎么地，大数据，Big Data 这个词就变得流行起来。

处理大数据惯常是属于商业智能（BI）的 事情。抽取数据、挖掘数据，制成报表、OLAP、仪表盘、挖掘模型，作为辅助决策之用。不过在 BI 领域都不这么叫法，大伙儿都说海量数据，Large- scale Data。这听起来还是略显学术气，不如 Big Data 来的通俗——大数据。这大概是因为如今随处可见的数据，一种爆炸效应带来的结果，已经脱离某种专业的范畴，人们需要用更简单的术语来命名这种数据 爆炸。这给不温不火的 BI 带来一些新的刺激，让 BI 人看到一些希望。

以前，不说国内，就算是国外，做 BI 也大多是局限在几个大行当，电信、金融、零售、政府，他们需要数据来帮助自己理性决策。在国内很长一段时间里，更是仅 限于电信和金融两个行当。可是尴尬的地方在于，决策者有时候更愿意相信自己的直觉，而非数据。这种意识虽然逐渐在变化，可从来没有发生过根本的变化。意识 的变化是艰难的。当一些新兴行业的介入，他们对数据的利用方式，价值的榨取，让人看到数据分析不仅仅用于辅助决策，而是可以从数据中获得收益了，它已经不 再是一种锦上添花的东西了，那正是因为大数据时代的到来。这得感谢互联网以及还未兴起的物联网，在这些行当里面，数据在爆发，不断增长。他们不甘心只是如 报表、OLAP、仪表盘之类的分析应用。数据分析部门可以按照推荐系统的点击效果利润分成；交易的数据可以包装成分析服务销售给商户，让他们自己去洞察市 场商机；根据用户的点击流行为和上网内容，个性化广告布放等等。

就在刚过去的 9 月，TDWI（数据仓库学院）发布了 2011 年第四季度最佳实践报告，而 这份最佳实践的主题正是大数据分析。TDWI 会通过调查问卷的方式，对全球范围的企业调查，目标对象既有 IT 人，有业务单位的人，也有咨询顾问。问卷的问 题一般都会询问企业应用 BI 技术的实际情况，现在如何，计划如何。所以，这类最佳实践报告可以反映出当下某项技术的现状和趋势。报告的内容也遵循一定结 构，一下定义，二看现状，三分长短，四谈趋势，最后再来个厂商介绍。同样，这份大数据分析的最佳实践报告也是如此结构。

其中关于“大数据”的定义，值得关注。如果我们仅仅从字面上看，大数据似乎跟海量数据差别不大，仅仅是变得更加通俗？并非如此，这份报告给出一些区别，TDWI 赋予这个术语更多的含义，更多符合目前数据爆炸时代的含义。

大数据的 3V

Big Data 的 3V。大数据有 3V 的特性。

Volume、Variety、Velocity。这 3V 表明大数据的三方面特质：量大、多样、实时。对，不光是数据量大了。对 TB、PB 数据级的处理，已经成为基本配置。还能处理多样性的数据类型，结构化数据和非结构化数据，能处理 Web 数据，能处理语音数据甚至是图像、视频数据。实时。以前的决策支持时代，可以用批量处理的方式，隔夜处理数据，等决策者第二天上班，可以看到昨天的经营数据。但现在的互联网时代，业务在 24 小时不间断运营，决策已经不是第二天上班才做出，而是在客户每次浏览页面，每次下订单的过程中都存在，都会需要对用户进行实时的推荐，决策已经变得实时。

这个定义非常完美，形式上也很漂亮，3 个 V。

可细细想想，这每个方面的 V，难道不是传统 BI 一直在试图征服的嘛？也许所谓大数据时代，是新瓶装旧酒。只是换了一个称呼，而具体要解决的问题，仍是那些存在已久的问题。可毕竟大数据时代轰轰烈烈地，踏着旧的海量数据浪潮而来，而且这将是更高一浪。平常人站在下面，是否会腿脚发软，或是识破浪头的力度，来个漂亮的转身冲浪呢？

大数据管理的需求与挑战

在这样的大数据时代，数据仍然是最关键的。如何将大数据管理好，仍然是对企业的考验。

无处不数据。手机通话、移动在产生数据，ATM 在产生数据，商品上的 RFID 在产生数据，包裹从一个城市到另一个城市在产生数据。就算是一个小小的店铺，当它销售出去一瓶水，也可能会记录到 Excel 里面，产生了数据。数据记录这世界的存在和变化。

当企业的某项资产非常重要，数量巨大时，就需要有效管理。如今，数据已经成为这种资产。以前人们还不会将它看做是资产，而是一种附属物。客户来办理业务，在系统中产生了这种附属物。而现在，发现在客户办理业务这条信息中，蕴含这一些客户的需求，成千上万条这类信息累积下来，就能洞察客户所需，为设计新产品，为客户个性化营销产生新的价值。数据变成一种资产了，需要被管理起来。

数据仓库是管理数据的工具。在近二三十年里，以某种类似蜗牛的速度爬行，它始终还是贵

族家的玩具。只有那些多金的买主才会为它买单。这让数据管理变得高高在上，数据当做资产只是停留在理念层面。人们还在争论着，数据仓库能够给我们带来什么？

我自己曾总结过一句话，体现数据仓库的六项价值——“能快速、及时、方便、准确而安全地访问整合过的数据。”现在看看，发现这个描述还蛮符合大数据时代，对数据管理的需求。

而这六方面价值也对应了不同的技术领域。

- 数据仓库硬件、软件、模型要保障对数据的快速访问。比如专用设备，按照数据温度选择数据是否高速存储，采用特殊存储技术；
- DW 模型确保数据的整合性，当你需要企业视图的数据，需要以年为周期的数据，需要数据模型的支持；
- ETL 保障数据及时性。批量的 ETL 已经不足够，需要准实时，甚至是数据流式处理；
- 元数据管理让数据访问更方便，不仅仅将数据以表、字段的方式管理，要将数据切分地更小，可管理；
- 数据质量管理保障数据的准确一致，让数据可信；
- 数据仓库架构、权限管理保障数据访问安全。

大数据时代对六项价值之一——快速访问数据的性能，有明显推动。人们最迫切的希望还是从无到有，从慢到快吧。让数据唾手可得。

数据库技术在变化

传统数据库并未专为数据分析而设计，数据仓库专用设备的兴起（Data Warehouse Appliance），如 Teradata、Netezza、Greeplum、Sybase IQ 等等，正表明面向事务性处理的传统数据库和面向分析的分析型数据库走向分离，泾渭分明。数据仓库专用设备，一般都会采用软硬一体，以提供最佳性能。这类数据库会采用更适于数据查询的技术，以列式存储或 MPP（大规模并行处理）两大成熟技术为代表。另外，新兴的互联网企业也在尝试一些新技术，比如 MapReduce 技术（这得感谢 Google 将它发扬光大），Yahoo 的开源小组开发出 Hadoop，就是一种基于 MapReduce 技术的并行计算框架。在 2008 年之前，Facebook 就在 Hadoop 基础上开发出类似数据仓库的 Hive，用来分析点击流和日志文件。几年下来，基于 Hadoop 的整套数据仓库解决方案已日臻成熟。目前在国内也有不少应用，尤其在互联网行业的数据分析，很多就是基于这个开源方案，比如淘宝的数据魔方。而在一些商业性的产品

中，也已经融入 MapReduce 技术，如 AsterData。

低廉的数据仓库解决方案降低了数据管理的门槛，长尾的中小企业不一定非得去跟 Oracle、IBM 这样的大公司去谈高高在上的价格。开源的产品，配置足够的硬件存储，有一支专业的服务团队，就可以架构一个数据仓库平台。在去年，就曾有多位朋友向我咨询的数据仓库方案，他们有一个不约而同的期望，价格不要太高。他们有服务团队。我没有其他推荐，只有推荐 Hadoop。

还有一些其他的技术可以让数据访问性能提高，比如数据温度技术，可以区分经常被访问和很少被访问的数据，经常访问的就是高温数据，这类数据将存储在高速存储区，访问路径会非常直接，而低温数据则可以放在非高速存储区，访问路径也可一些相对复杂一些。近两年，存储访问的技术也在变化着，比如 Teradata 前几年推出固态硬盘数据仓库，用接近闪存的性能访问数据，比原来在磁盘上顺序读取数据快很多。后来又兴起一批内存数据库产品，这类产品在 DBMS 软件上进行优化，规避传统数据库（数据仓库）读取数据时的磁盘 IO 操作，再次大大节省访问时间。比如 SAP 的 HanaBI、Oracle 的 TimesTen、SolidDB、extremeDB、Altibase。

文本、语音、图像、社交网络、地理位置...大数据时代的数据类型如此丰富。用关系型数据库存储这类数据，再深入去分析挖掘这些数据，开始有些负累。

于是，越来越多的 NoSQL 数据库涌现出来，其中很大一部分是用于分析用途。比如西班牙有个小厂商，叫 illumnate，他们拥有一个叫 Correlation DBMS 的数据库产品。它不像关系数据库那样按照表、字段存储，那样冗余很大。CDMBS 的做法是，针对每个不同的值，只有一个地方存储，而所有对这个值的引用，都在索引中记录。比如有个客户的姓名叫“张三”，而还有一个公司名字也叫“张三”，那么在 CDBMS 里面，只存有一个“张三”这个值，但在索引里面记录了有两个地方引用它。这种数据库是专门为分析而设计的。因为不存储冗余数据，所以它对于海量数据，非常节省空间。如果说这个有点不太吸引人的话，另一个据称的优点就是做 ad-hoc 查询非常快捷。

社交网络很火热，Facebook、Twitter、QQ、MSN，甚至是普通的电信通话、邮件，都构成社交网络。人们决策的一个重要依据其实就跟社交群体相关，周围人的决策会带动你的决策，用社交网络理论来做决策支持是一个重大方向。

用关系型数据库来存储社交数据有点吃力。我跟你打电话，“我”是一个“用户”的实体，“你”是另一个“用户”的实体，我们之间存在了“通话”的关系；“你”还可能跟“她”发生了关系。但社交网络的分析还需要关注圈子、关系紧密度..... 人们想从中找到人与人之间的关系、圈子，是不是一个家庭的，是不是一个公司的，是不是情侣关系。甚至还要去

发现一个人的重要程度，是否具备某种影响力。用实体关系来表述这种社交网络需要绕些弯路转换。所以，自然出现了一种图数据库（Graph DBMS）。数据按照节点、关系和属性键值存储。开源产品 Neo4j 就是这类 GDBMS。基本上这也是一种键值数据库，也就是说其最底层数据存储都是按照 key-value 存放的，这种存储方式是比较适合并行处理，适用于分析。而 graph database 的重要特点就是内置了常见的 graph 算法，它的存储结构让这类算法性能倍增。可想，未来也许会出现专为图像分析而出的数据库，专为视频分析的，等等。

数据的量越来越大，种类越来越丰富，大数据时代需要新的数据管理手段。列式、MPP 的关系型数据仓库在改变着，NoSQL 的 CDBMS、GDBMS 也试图在改变着。关系型数据库是企业 IT 建设时代的数据管理基石，而在 Big Data 时代，也许需要一种新的，正在探索中的数据管理基石。

作者简介

刘庆（网名：Q），定居合肥，BI 独立顾问，兼职于 Teradata，从事电信业的 BI 咨询服务工作，入 BI 一行 10 余年，早期研究 BI 架构，近些年偏重业务分析。另一身份为 ttnn BI 论坛创办人，写写文章，编编杂志。

相关内容：

- 年度技术回顾之数据库、NoSQL、开源软件
- 百万级访问量网站的技术准备工作
- 数据库在现代搜索技术中的应用
- 亚马逊开始提供 MySQL 服务
- Martin Fowler 看到了数据存储方式的复苏

特别专题

阿里巴巴数据架构设计经验与挑战

受访人 姜迅 采访人 霍泰稳

InfoQ：我现在是在阿里巴巴园区，采访阿里巴巴的数据架构专家姜迅。江迅您好，请先给大家做一下自我介绍。

姜迅：您好，我是阿里巴巴数据仓库架构师，我叫姜迅。08 年加入阿里巴巴，之前一直是 Oracle 的 DBA。加入阿里巴巴之后，一个很偶然的机会，从 DBA 转到了数据分析。现在想起来确实是个非常正确的选择，因为在这两年多时间里，我越来越感受到不同部门乃至整个社会对数字的迫切需求，随着需求量的增长，我们会发现很多原来的架构已经无法支撑现在数据分析的需求，这就迫使我们去利用其它方法来满足这样的分析。目前我在阿里巴巴负责三个团队，第一个团队负责数据模型，把业务抽象为实体。另外一个部门负责技术架构，主要负责搭建一个能够支撑海量数据的分析平台。那么第三个部门负责把数据推销出去，负责数据的展现，也就是把数据分析的结果让用户看到，这也是数据分析的价值。

InfoQ：其实在我们以前的沟通当中，感觉你是对数据比较痴迷的，那请给我们介绍一下数据的价值，包括为什么现在数据分析这么热？

姜迅：我觉得其实可以举个简单的例子，就拿阿里巴巴来说，可能大家都知道去年温总理来我们这里访问，当时做了一个汇报，汇报里提到很重要的一项，就是我们有一个很重要的指标是反映国家的出口趋势指标。比如中国出口在 2010 年 Q3 会有上升或是下降，为什么阿里巴巴能够知道，那是因为，在阿里巴巴我们能够拿到 12%左右的询盘，或者说交易是从阿里巴巴平台完成的，以服装行业为例，如果说 2010 年 Q2 大概有 20%的服装的询盘，可能在英国地区，我们大概能够推测经过 AQ 的生产，出口一定会上升，因为在三个月前，他已向我下了定单，经过三个月的生产，这个生产好的服装就一定会出口，那么我们只要通过 2010 年 Q2 询盘量，就能够去推测到我们出口上升，所以我们实际上是能够在整个国家的宏观经济上去做一下分析。

另外一个淘宝的淘宝魔方，也叫数据魔方，这个东西实际上我们现在可以做很多的分析，举一个简单例子，假设，上次我们还没有做这个事情，比如说通过你对衣服的尺码的分析，我

们可以分析出中国人现在是偏瘦还是偏胖。因为我可以通过多少人买均码，多少人买中码，多少人买大码，包括你鞋的尺码是能够分析出来的，我们越来越觉得数据是能够来对整个社会，甚至企业本身运营都是能够起到支撑作用的，所以在宏观经济上我们都觉得是有很大帮助，因为我们现在感觉很多社会行为是被数据所领导的，不管你做什么事情，上网买机票、买衣服、在银行里刷卡、打一个电话，很多信息是被记录的，记录的数据能够得到善用，而非去做恶，拿到这些数据之后能够把它所隐含的这种规律挖掘出来之后，能够指导整个企业运营也好，预测国家经济趋势也好，我们都希望能够拿到，因为对阿里巴巴来说，很重要的一点是我们的数据是非常非常有洞知力的，因为它是一个商业数据，不像是 QQ、盛大这些公司，他们可能以娱乐数据为主，那么他们可能信息噪音会比较多，但是在阿里巴巴平台和淘宝平台上，因为就是在做生意，就是要去买东西，所以用户的行为很单纯，而且这种行为跟钱是相关的，所以这些数据的商业价值是非常高的。

InfoQ： 这么大的数据量需要一个什么样的架构来支撑，在设计架构的过程中主要有哪些难点？

姜迅：我们对数据量感受很深，因为我 08 年刚来时，当时数据量大概在 20 个 TB 左右，到了 09 年，已经达到了 100 个 TB。到了 2010 年底，就达到了 300 个 TB，现在应该已经到了 500 个 TB。所以这种膨胀速度是非常快的。我们最早用的是一台 IBM 570，当时是 09 年，后来觉得不够，我们用了六个节点 Oracle Rac 去支持它。到了 09 年中就发现已经不够用了，所以在 09 年中我们引入了 Green Plum 做数据分析，再后来引入了 Hadoop。这样我们能够用比较少机器就能够满足我们现在需求。但对数据开放存储，我们把它放到 Hadoop 上，这是我们做的一个变化。总的来说，就是从一个单机到多机的 Rac，然后再到分布式系统，这个分布系统不管是 Green Plum 还是 Hadoop，一定越来越趋向分布式。因为单个 PC 的能力现在越来越强，但无论是小型机还是大型机，始终这种 SCSI 扩展方案总会有瓶颈，所以一定是批量做水平扩展，不管是用那种技术，我们一定会选择水平扩展方案做数据分析架构。

InfoQ： 那它的主要难点在什么地方？

姜迅：最主要的难点还是在系统的可伸缩性，就是怎样能够让一个系统很快速地随需增加计算能力或存储能力，之后有另外一个难点就是怎么能够把资源切片，然后分给不同部门，这是另外一个比较难的地方，因为现在我们发现，越来越多的部门都需要数据分析，它并不只是我们数据仓储一个部门的事情。如果有多个部门都需要来访问这些数据，那我们应该怎样把数据开放出去，这个是整个架构里面最难的部分。如果仅仅只是仓储一个部门使用，那我是比较容易做一些控制的，因为大家都知道，现在做数据分析绝大多数都是用 SQL 做分析，一个好的 SQL，完全相同的功能，执行的性能有可能相差一万倍，这点我们最近感受很深。

我们做了很多优化，经过简单优化尽量把一个任务从五个小时或十个小时变成五分钟，这种执行效率带给系统非常大的压力，如果这种随意分析任务能够上线，那么整个系统即使有再多资源也不够用，特别是分布式系统之后很多单机优化方式就更不适用了，而且每个查询就是一个简单的 SELECT COUNT 查询，可能涉及几个 TB 的数据。比如网络访问日志，一张表就有几个 T 或者十几个 T，单表查询就可以把整个系统搞得非常忙。

我们非常关注的一个难点，就是如何能够在数据开放时，既能够保证系统的稳定性，又能为每个部门提供一部分系统资源，同时还不会导致系统独占，每个人都能做分析。这种任务可以有优先级，这也是我们现在比较关注的一个部分。我们现在也还在做的比如自动任务解析，因为现在任务都是用 SQL 或 HQL 写的，通过解析 SQL 提取原数据，再通过分析对象与对象之间的关系，就可以自动建立依赖，换句话说就是把 SQL 本身解析出来，FROM 什么，WHERE 是什么，属性从哪里来。其实业界有数源血缘分析的说法，即通过分析一个指标的变化从哪些任务里来，如果原系统发生了变化会影响哪些指标。这是很难的一个部分。确实，我们现在将近有一万个任务在系统中运行，有几千个指标，原系统在不断的变化，当原系统发生变化后就一定会影响到指标，如果这些指标变了，我希望能很快知道原系统某个资源变了，或者它的值变了，还有哪些指标会受到影响，我必须能够快速识别，再做代码修正。举个简单的例子，MEMBER，一个用户名字段，原来可能是 4 个长度，现在变成 48 个长度，我需要知道有哪些表要把这个长度变到 48 个长度，因为很多字段并不是说名称一定相同，名字可能是会变的。这个时候我需要解析比如我知道 INSERT 的 COLUMN 是从哪个 FROM 的 COLUMN 中来，这么一层一层地解析，最后把整条链都打通。

InfoQ：做为阿里巴巴高级数据架构师，现在的数据架构是怎么设计的，正在做哪些方面的优化？计划设计成什么样子？

姜迅：我们现在整个系统结构基本是分成三层，最下面的数据层依赖于 Hadoop 解析存储，用 Hadoop 集成，中间的计算没有完全在 Hadoop 做，因为我们觉得这是我们实测的结果，也发现 Green Plum 的计算能力跟 Hadoop 的计算能力基本是 1：4 左右，如果同样的计算任务就需要四倍的 Hadoop 来支撑，因此计算是在 Green Plum 完成，然后再把它写回 Hadoop 去。这就是我们数据分析框架，更进一层，我们希望数据分析往前走，这是值得做的事情，原来说分析数据趋势与波动，这些数据一般都是给老板做一些专业决策支持或预测，如果真是这样，如果公司哪天不景气，有可能先砍掉的就是你这个部门，因为你确实只是作为辅助部门。所以我们现在想往前走，于是我们在结构上做了大的调整，我们把数据提供给千百万用户直接使用。这样系统就成为整个网站中不可或缺的环节，所以我们要把整个系统推着往前走，做推荐也好，做数据分析也好，把数据直接卖出去。

例如阿里巴巴中文站首页和国际站首页，有几个按钮是来自数据仓库中数据分析的结果，这种数据需要直接向数据仓库请求，因为，如果数据仓库服务器 down 掉，整个中文站首页就不能用了，这样的话，跟网站是基本上同样一个水平，我们是把对数据认知直接让最终客户感受到，所以我们会有往前走的目标，我们用分布式数据库来支持这种海量并发数据访问，我们能把对数据认知做成数据服务提供给最终用户，能够使他们直接看到我们对数据的理解，这也是我们想做的变化，我们会一直不断往前走。

InfoQ： 从现在的数据架构到你所设计的架构，过程是非常难的，你现在的工作过程中遇到的比较大的挑战是什么？刚才你也提到一个公用模型，也请给大家介绍一下。

姜迅：我刚也提到我们有一万多个任务，有几千个指标，这么多任务他们之间是相互依赖的，这种依赖关系非常复杂，我们尝试画过整个数据仓库之间的任务关系，结果是一张蜘蛛网，可读性非常差，所以我们想把每个指标做成服务，先对整个阿里巴巴的数据进行抽象，因为有一些实体是不会变化的。比如会员这个实体是不会变化的，不管这个公司今天举办什么狂风行动，明天发起什么春雷行动，但在五年内，我们总是基于会员做营销，这是整个阿里巴巴的根本，所以会员这个中心点是不变的，但这个中心点的很多属性是变化的，所以可能今天要分析旺铺的 PV，明天可能要分析支付宝的支出情况，它的每个属性是不断变化的，我们希望找到变化的一个中心点。找到中心点之后，它的每个属性就跟蒲公英一样，做成插拔式的，即能够插入进来又能够拔掉这个指标，它不会影响到整个任务，这就是指标服务化，这点是我们非常想去做的事情。

如何做到蒲公英模型呢？我们有一个概念叫列交换。因为这种指标化如果之间见不能相互解耦，最终目的还是无法实现，所以在某个指标过时的时候这个指标能够快速下线而不影响其他指标，这种功能在基于 SQL 的架构中是无法实现的，因为你必须把这个指标有关所有代码全都改一遍才能把它下线。我们希望做到的是每个指标都是一个独立服务，这个服务能够快速加入也能快速下线，这是我们想做的。这与我们目前基于关系型数据库设计有关，因为现绝大部分的数据分析都是基于关系型数据库模型，所以我们现在这个模型实际上是基于 KV 这种结构去设计的，我们认为所有数据都是基于 key/value/description 这么一个三元组，这个三元组足够描述一个对象的属性。你可以按照不同的方式组合这个三元组，假设这个数据你要去使用的时候，你是想按照对外提供访问的，那么你就可以把这个三元组，这个 KV 对应的三元组能。比如把一个 Key 的数据能够放在一起，那么这样的话，你就能够支撑，对这个人，你想要知道这个人他的所有数据，就能很快的拿到，那如果说，我是要把他做分析的，那么实际上你就可以把这个三元组，把某一个 Description16：29 放到一起，那这样的话，我就可以知道说，对某一个属性，它的有多少个不同的值，这个是说，我们可以，当你把所有的这个指标，把它都拆解，抽象化后，你都会得到这么一个 KV 对，这也是其实我现在理

解，为什么由此以后他会变得这么热的一个原因。

你可以把所有这种对象描述做成一个 KV 描述，这样你能够很方便地组合，我能够很方便地添加。例如我能够 set 一个对象，这个对象可以是没有这个架构的，这不像关系型数据库，它必须有一个列，如果女性增加一个属性比如服装的颜色，你就必须有一个字段颜色来支持，如果没有就必须增加一列。但如果是 KV 结构就不需要了，只要 set 属性就可以了，而且不需要预先定制。因为每个对象属性会越来越多，特别是现在，我们做用户分析的时候，实际上每个人对用户的理解都不一样，例如我们会分析，你是杭州人吗？你是个高价值的人吗？是男人吗？会给每个人打上各种不同的属性，每个人的属性都会不一样，因为每个人不一样。如果说一个对象要打这么多属性，可能那张表有几千几万个字段，数据库也能够支持，如果我们的模型能够把每个人不同的属性不断插进来，把出去，这个就变得很方便。我想看这个人的信息，我就可以把这个人所有值都拿出来，我想按照区域来看，就可以只把区域这个东西放在他一起，我就可以把杭州关于这个人的信息全部筛选出来，而且这个根据每个人的想法不同可以不停地增加，它不需要 DB 给你增加字段，要求做性能测试。一个做 PI 分析的人，或者一个业务发展人，他只有有想法就可以把值加进去，然后这个属性马上会回到系统里去，它能够被另外一个人筛选，这样整个数据统计就变成一个回环，前面系统收集到这个人的属性，紧接着这个数据就能被 BI 分析，而且整个过程不需要人工干预，系统自动支持。

InfoQ：也就是说架构越来越灵活？

姜迅：对，我们希望达到这样的效果，只有这样我们才能把人解放出来，我们不需要每天去应对字段变化而做变更，这样太痛苦，如果能够做到这点，我们会比较有弹性，所有的组织结构能够做到可插拔。

InfoQ：刚才提到目前比较热的一个技术是 NoSQL，能简单点评一下 NoSQL 这个技术的优缺点在些什么地方？

姜迅：根据我的理解，NoSQL 还是一个存储，是基于关系型数据库的设计，与我们现在的一些关系数据库 Oracle/DB2/PostgreSQL 不同，就像现在我们说的 Java 对象，其实这种对象设计都是基于一个 Key，它有一些属性，NoSQL 更自然贴近 Java，因为它不需要做序列化和反序列化，它自身就有 Key/Value。我觉得首先是关系模型支持越来越方便描述，这是它比较大的特点，例如你要描述一个人有多少件衣服，你必须有一张叫做人的表，另外有一张叫做衣服的表，那张表可能是一张素表，可能有很多不同的衣服，NoSQL 就不需要，他还是一个人一个 Key，你可以有三个 Key，第一个 Key 是他有件黄色的衣服，第二个 Key 是他有件蓝色的 T 恤衫，第三个 Key 是他可能有件红色的马甲，你可以把这些 Key 都放到同一张表里去，而不需要多张表就维护，你只需要有这样一个 Key 传进来，你就可以找到它了，它能更自然

地描述。我觉得这是一个特点，并不是最关键的点，最关键的是把对象都插到 KV 之后，NoSQL 的伸缩性做得非常好，而且简单，因为伸缩性对我们来说非常重要，特别是我们刚才提到的增长，每个这个数在不断地增长，基本上你根本没办法在今年预计到明年会怎样。

我们的数据库原来打算用 IBM 小机，从 590 升级，后来我们发现，590 已经插满了，你怎么升？没办法升级，往上扩展也不足以支持应用，所以只能水平扩展，这必然要求有一个健壮的分布式机制，我们现在来看 NoSQL，不管是 Hbase、Cassandra，还是 MongoDB，它们都是利用这种廉价 PC 来支撑整个系统，都是基于对象描述建立不那么复杂的关系模型，它对对象进行抽象、简单化，让整个系统获得非常好的扩展性。尤其对网站而言，我们愿意为了扩展性去牺牲比如一致性等，因为如果不能扩展，就不能支持业务增长。前段时间，Facebook 上有一家公司 Zynga，它号称一个星期要增加一千台服务器，这种增长真的是每天一个 PB，一个星期就一千台服务器，这种增长，它里面更多的是视频这种信息，但确实，我们会越来越多看到这种 PB 级的数据中心或者说数据量，雅虎已经有 60PB 的数据，像这种你没有办法，你只有用这种 KV 分布式支持，我并不是说完全是 NoSQL，我觉得我们现在谈 NoSQL 都是说它不仅仅是 SQL，如果说是 SQL，对我们分析而言或者对网站而言，SQL 是一个非常简单易用的工具，我觉得没有一个语言比 SQL 更简单更易懂了，比如我要做一个聚类，用 Group By 就行了，要做去重，用 Distinct 就行了，这整个语言都是非常简单的。90%的数据需求通过 SQL 都能快速支持，包括 Facebook 的，他们也提到说整个 Facebook 数据开放，把 HQL 开放给整个部门，90%用 Java 没有用 MapReduce，就是用 HQL 去写。因为这个任务非常简单，每个人都能理解它，只有被更多人使用，这个数据才能发挥它的价值。

但是 SQL 确实有很多场景支持得不好，这个时候我们需要有一些不同的方式来支持。我对 NoSQL 的理解是这样的，首先它不仅仅是一个 SQL，它并不是说排斥 SQL。另外，我觉得 NoSQL 有几个核心点或者说核心概念，它是一个可扩展的系统，我们要求系统的扩展性非常高。另外一点是它的架构，它是一个自由架构的结构，它不像关系模型一定要把架构制定得非常严格，这是我的理解。第三点是它基于大量 PC 结构来支持，因为现在 PC 很便宜，单机性能很高，它能够通过机柜堆叠来扩展以达到比一个小型机更高的性能。我理解 NoSQL 是一个过程，或者说，NoSQL 对我们来说是对现有架构的补充，并不是说它一定要把谁干掉。

InfoQ：对于现在市面上已有的 NoSQL 产品，你会从哪些方面去考量呢？

姜迅：其实我们现在最关心的还是它是不是够健壮，够稳定，特别是我们线上系统。健壮和稳定性是我们最关心的一个部分，另外一个就是性能。实际上，前段时间我们用过 Cassandra，也用过 MongoDB，也尝试用 Hbase 做一些业务支持，根据评测，包括我们自己还有开放的阿里云，我们 KV 引擎从单个性能来看，它一定比关系型数据库能力要弱，就单机相比，或

者同样这种以前的机器跟 PC 分布式比，会发现它比这种数据库要弱。因为毕竟数据库发展这么多年，它有成熟的索引机制，但 NoSQL 的可扩展性我觉得是非常重要的一点。另外现在绝大多数 NoSQL 都是开源的，这样我们有很多想法可以在上面实现。第三是我们希望这个系统像 Facebook 或雅虎这样的公司是不是在使用，因为要找到好案例，跟着大趋势走，而不是说自己找一个前沿的东西研究，对企业而言，要尽可能低成本支持更大的应用，这也是我们考量的一点。我们不愿意做第一个吃螃蟹的人，因为毕竟系统稳定性是比较重要的一点。

InfoQ： 好，非常感谢姜迅接受我们的采访，谢谢。

姜迅：谢谢。

原文链接：http://www.infoq.com/cn/interviews/jx-alibaba-data-architecture-design

相关内容：

- 面向开发的 MySQL 性能优化
- 百度贴吧的架构实践
- Sina App Engine 数据存储服务架构
- 百万级访问量网站的技术准备工作
- 百度数据库架构演变与设计

特别专题

大数据时代的创新者们

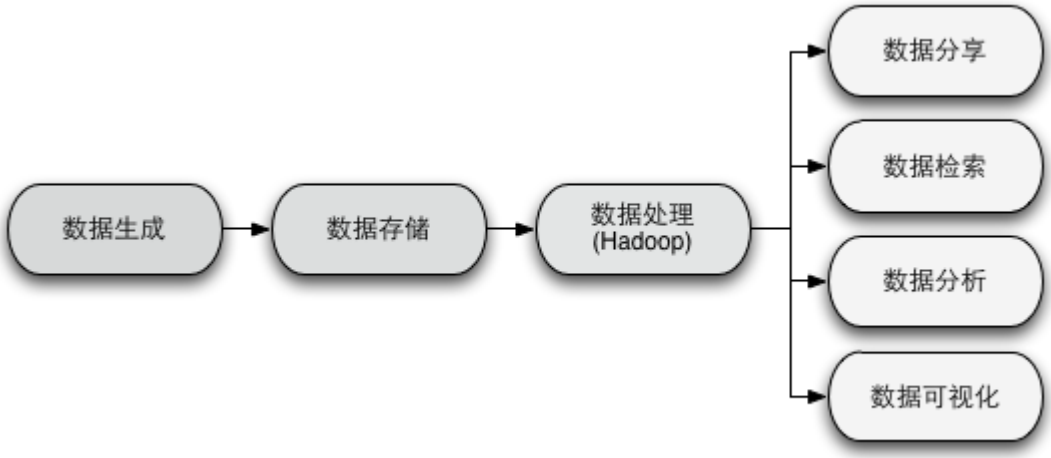
作者 李明（nasi）

大数据的时代已然来临。IDC 数据显示，在 2006 年全世界的电子数据存储量为 18 万 PB，而如今这个数字已经达到 180 万 PB，短短 5 年间就已经增长了一个数量级。而根据预测，2015 年这个数字则会达到如同天文数字般的 800 万 PB。就在此时此刻，海量数据依然源源不断地产生，从不停息。面对这些“大数据”，有些人叹息抱怨，害怕数据量的剧增对于现有 IT 架构的冲击；有些人积极主动，探寻应对海量数据的应对与解决之道；还有一些人，则是顺势而为，抓住时代发展的商业机会，成为富有活力的创新者。本文就将聊聊这第三种人。

大数据的发展催生了诸多商业机会和商业模式。而这些公司所面对的独特的时代背景，就注定了它们必会受到市场和资本的追捧。它们中的一些或是已经融资成功，进入高速发展期；或是被成功收购，帮助投资人和创始人成功从项目中退出。而很多上市公司，也开始在这一领域动作频繁，积极布局，这也从侧面反应了这一领域的广阔前景和巨大的利润空间。

大数据的生态系统

IBM 的架构师 Stephen Watt 曾在《Deriving new business insights with Big Data》一文中简单讲解过大数据的生态系统。简而言之，大数据的生态系统，就是数据的生存周期。数据从产生，到处理，再到价值提取，最后被消费掉，这整个过程就构成了大数据的生态系统。如下图所示。



在这个生态系统中，无论是数据的存储、数据的处理、数据的分享、数据的检索、数据的分析，还是数据的可视化，都存在着不同的商业需求。需求的出现必然会导致创新的产生。所以，在每个步骤都有不少初创公司在深耕自己所在的领域，试图通过新技术和新方法来实现新的商业模式。

数据的存储

Amazon 是大数据和云计算的先行者，它推出的 S3 云服务也 早已成为云端存储的业界标准。通过易于使用的 API，用户可以很方便地将各种数据对象放在云端，然后再像使用水电一般按用量收费。S3 根据用户所占用的存 储空间、请求数和数据流量进行阶梯定价收费。同时，S3 还为对数据可靠性的要求并不高的用户提供了更为便宜的去冗余存储模式。Amazon S3 服务是典型的付费服务商业模式，增长十分迅速。去年 Q4 共有 2620 亿个对象储存在 S3 上，而今年 Q3 这个数字已经翻了一倍，达到 5660 亿。更难得 的是，Amazon S3 的云服务真正让许多创业公司享受到了云计算带来的便捷。使用 S3 作为存储支持的文件分享服务 Dropbox 进行的最近一轮的融资，估值高达 80 亿美 元，每天上传的文件多达 2 亿个。

大数据时代另一个热点便是 NoSQL，不但诞生了很多 NoSQL 的数据库产品，还围绕着 NoSQL 产生了不少新技术新模式。也许提起 10gen 这家公司，读者们会觉得陌生，但是说起他们的产品 MongoDB， 则是鼎鼎大名。由于其易用性和高性能，MongoDB 在很多开发者眼中已然成为 NoSQL 的首选。10gen 公司提供基于 MongoDB 的服务，包括商业 支持、培训和技术咨询等等，像 Foursquare、Craigslist 这样知名的公司，都是 10gen 的客户。10gen 于今年 9 月完成 D 轮 2000 万美元的融资。

其他的 NoSQL 产品在大数据时代也广受注目。为企业提供基于 Cassandra 的 Hadoop 构建方案的创业公司 DataStax 近日宣布完成 1100 万美元的 B 轮融资；NoSQL 数据库技术提供商 Couchbase，则集合了 CouchDB 和 memcached 的设计者和开发人员，今日完成了一笔 1400 万美元的 C 轮融资；图形数据库厂商 Neo Technology 也凭借其开源项目 Neo4j 获得 1060 万美元的融资。这些公司主要将融资用于 NoSQL 旗舰产品的研发，并努力提升和拓展市场份额，然后基于它们的产品开展业务，它们在盈利的同时，也为社区提供了高质量的 NoSQL 数据库产品，从而实现共赢。

分布式文件系统也是大数据存储的方式之一。最早由 Powerset 开发的 HBase 就 是基于 HDFS（Hadoop Distributed Filesystem）的分布式数据库。虽然目前还没有专门的商业公司来做针对 HBase 的业务，但 HBase 在业界已经有众多使用者，许多知名公司比如 Facebook、Twitter、淘宝等都是 HBase 的用户。

数据的处理

Hadoop 是大数据时代数据处理的首选。脱胎于 Google MapReduce 的 Hadoop 凭借其开源和易用的特性，很快成为了大数据时代的最耀眼的主角。目前，Hadoop 已经成为大数据生态环境中不可或缺的一环，是拥有海量数据 处理需求的公司的标准配置，许多商业创新和产品创新也都是围绕着 Hadoop 展开的。Yahoo 也已经认识到了 Hadoop 的价值，将 Hadoop 拆分成 一个独立的商业公司 HortonWorks 进行运营。

虽然 Yahoo 是 Hadoop 最大的贡献者，也进行了 Hadoop 的商业化，但却没法阻止其他的颇具实力的竞争者进入这个前途无限的领域。Cloudera 便 是其中最耀眼的一个。且不说联合创始人中有 Facebook 和 Google 的精英们，就连 Hadoop 的创始人 Doug Cutting 也从 Yahoo 离职加入了 Cloudera，这一举动当时在业界还引起了不小的震动。Cloudera 最开始的模式是帮助企业管理数据，后 来则转型为软件厂商。他们推出的软件发布包可以帮助企业更方便地搭建以 Hadoop 为中心的数据管理平台。Cloudera 也是通过技术支持、培训和咨询 等付费服务来盈利的，目前融资已达 3600 万美元。

如果说 Cloudera 是依靠其华丽的精英团队来吸引客户的话，那么 MapR 则 是通过过硬的产品来让业界认识到他们的价值。据称，经过 MapR 改造的 Hadoop 的速度可达原来的 3 倍。对于 Hadoop 的 MapReduce 模式，相 信现在基本上已经没人提出质疑了，然而大家更关心的是，这玩意还能不能更快，MapR 则很完美地回答了这个问题。EMC 也宣布在一些产品使用 MapR 版本 的 Hadoop，而 MapR 也刚刚完成了 2000 万美元的融资。

除了速度以外，Hadoop 的易用性也是一个用户所关心的问题。虽然相比较其他的框架而言，Hadoop 已经简化了许多使用 MapReduce 技术时所需要做的工作，但是对于终端用户而言可能还算不得十分友好。近日宣布完成 570 万美元 A 论融资的海量数据管理软件商 Platfora，就在试图解决这个问题。Platfora 旨在提供一个更为友好且更具操作性的用户界面，而且这个产品可以兼容包括 Cloudera 和 MapR 在内的各个 Hadoop 版本，能够大大降低使用 Hadoop 的门槛，让更多的公司体验到 Hadoop 的技术优势。

不仅仅是 Hadoop 本身，就连 Hadoop 的周边也不乏成功的创新者。AsterData 已经成功地被老牌数据仓库厂商 TeraData 以 2.63 亿美元收购，他们的核心技术叫做 SQL-to-MapReduce，可以将海量非结构化数据的处理技术和结构化数据的数据仓库技术结合在一起。而这种高速处理海量非结构化数据的能力，恰恰是传统数据仓库的公司所欠缺的，这也是为什么 TeraData 肯花如此大的价钱买下 AsterData 的原因。

数据的分享

数据本身也非常有价值。虽然，大部分的公司所面对的数据都是由内部系统或者交易记录日志之类的东西所产生的，但是这并不意味着他们不需要一些自己无法获得，或者已经被处理过的外部数据。因此，能够下载或者访问数据集，自然而然也就成为了商业需求，甚至美国政府都推出了官方的数据集网站可供下载。

InfoChimps 正是一家在线的数据集市，吸引了不少才华横溢的数据开发者。数据提供者可以将数据集上传至 InfoChimps，可以供人免费下载，或者以一定的价格销售。另外，InfoChimps 还提供很多 API 可供用户调用，在超过一定数量的免费 API 调用限额后，InfoChimps 会向用户收取一定的费用。InfoChimps 的目标就是让每个人都能找到自己需要的数据集，目前这家公司已经完成了 A 轮 120 万美元的融资。

提供 API 服务的数据集分享公司并不止于此，Factual 就是一家开放数据平台的公司。它所提供的多种数据集汉涵盖了本地服务、娱乐、教育和医疗等多个方面，不但可以通过 API 访问，还可以很方便地通过 SDK 集成到移动应用当中，为依赖数据的移动创新带来了很大的便利。Factual 也是通过收费 API 调用的方式来盈利的，目前已经募集资金达 2700 万美元。

数据的检索

数据检索在搜索引擎时代已经不是什么新鲜事了，然而随着社交网络的盛行和大数据时代的到来，实时性检索的需求也就变得越来越强烈。事实上，实时性的需求一直以来都是存在

的，只是受囿于技术和成本的原因而没有什么实质性的突破。如今，随着实时数据处理技术的不断成熟，实现实时性数据检索也已经成为可 能。

实时搜索引擎 TopSy 是目前少有的独立运营的实时搜索引擎 ,他们号称可以每秒 钟索引 100 万份文档，这个速度基本上能够满足实时性的需求。目前 TopSy 主要索引的是 Twitter 的数据，它提供了 API 可供用户访问。在 2011 年 1 月间，TopSy 共收到 5 亿次请求，绝大多数是来自于 API 的调用。因此，公司也在考虑推出收费的 API 服务，以解决目前公司盈利模式不明朗的问题。 TopSy 已经完成了 C 轮融资，融资总额度高达 3000 万美元。

说到实时数据检索的问题，就不能不谈到 Twitter 刚刚推出的开源产品 Storm。这个产品一经推出就立刻吸引了大家的目光。然而却少有人知道，Storm 其实来源于 Twitter 刚刚收购的一家名为 BackType 的 公司。这家公司由大名鼎鼎的 YC 进行孵化，在被收购以前就计划推出 Storm，然而期间却经历了 Twitter 的收购，因此收购以后由 Twitter 发布 Storm 也是顺理成章的事。Storm 每秒钟可以处理数百万的消息，非常适合实时消息处理，而这也许是最为吸引 Twitter 的地方。

最近还有一件与实时数据检索相关的收购案颇为引人关注 ,全球最大的连锁零售商日前宣布收购了移动和社交广告公司 OneRiot， 然而这次收购的交易金额并未对外透漏，OneRiot 也被并入了沃尔玛实验室。OneRiot 最早是一家实时搜索公司，后来借此涉足广告领域，并关停了实 时搜索，专注于实时广告业务，并开始提供应用内移动广告的社交服务。OneRiot 最吸引沃尔玛的地方，应该就是所谓的 Big Data + Fast Data，将实时的数据处理与分析和广告联系起来，这也将是广告业未来发展的一个必然趋势。

数据的分析

在线数据分析服务平台是数据分析的趋势。Quantivo 的口号是“Big Data Analytics for Everyone”，该平台可以从多种来源组合业务数据，对其进行整理和合并，然后让客户通过专有接口来访问甚至提问，平台会帮你找到最好的答案。另外一 家提供在线分析平台的公司是最近刚刚完成 8400 万美元融资的 Opera Solutions，这次融资也使该公司的估值达到 5 亿美元。用户将数据上传到 Opera Solutions 的平台上，然后 Opera Solutions 会针对用户的不同需求，结合行业专家的建议来为用户提供服务。该公司虽颇为低调，但年营收早已突破 1 亿美元。

然而并不是每个公司都是服务导向性的公司，Palantir 就 是一家产品导向性的公司。这家由前 PayPal 员工和 Stanford 的一群科学家们所创建的公司，融资总额已接近 2 亿美元，估值

高达 25 亿美元。 Palantir 主要是为政府和金融机构提供高级数据分析平台，该平台源自 PayPal 的反欺诈分析平台，将人工算法和强大的数据库扫描引擎整合在一起， 帮助用户通过多种方式快速浏览相关的信息。更有趣的是，这家公司号称永远都不会有销售、营销和公关人员，坚持追究极致产品的乌托邦式工程师文化，完全通过 口碑来推动公司的业务发展。

随着社交网络的兴起，社交数据的分析也成为了热点。今年 Saleforce 就宣布以 3.26 亿美元的价格收购社交数据分析公司 Radian6。Radian6 的业务主要是围绕着各个社交网站所开展。通过对各个网站的监测和分析，Radian6 能够将客户关心的数据尽早呈现，从而使这些客户能够更 为主动地制定市场营销的战略。对于 Saleforce 而言，Radian6 最吸引它的地方，便是可以将现有的 CRM 与社交分析整合在一起，从而更好地满足 客户的需求。

数据分析的服务并不只是空中楼阁或是大佬们的玩物，也许它就在你我的身边，被 Next Jump 收购的公司 FlightCaster 就是这样一家公司。它根据过去 10 年里的各种数据和当前实时的状况，通过专利算法来预测国内航班可能会延迟的概率，并能够早于航空公司 6 个小时通知你。这对于经常坐飞机而又饱受飞机延误之苦的人们来说，这个预告还真是有其现实意义的。

数据的可视化

数据可视化可以提供更为清晰直观的数据感官，将错综复杂的数据和数据之间的关系，通过图形的方式表达出来。俗话说：一图胜千言，这句话用来形容数据可视化真是再贴切不过了。从某种意义上说，数据可视化更像是一种艺术，它所传达的美感总是让人印象深刻。

MeLLmo 公司就是先行者之一，主要关注于企业移动应用领域的数据可视化技术。MeLLmo 推出的数据可视化平台 Roambi 可以通过网站和移动设备导入各种类型的数据，并将其图形化处理。关于 Roambi 的盈利模式，主要是为企业用户提供 Pro 的付费服务。MeLLmo 近期刚刚完成 A 轮融资，总融资额为 5000 万美元。

InfoGraphics 也是广义数据可视化的一种表现形式，通常用于信息的可视化，许多知名公司都为其拥有的信息制作过极富美学特质的 InfoGraphics。Visual.ly 号 称是目前互联网上最大的 InfoGraphics 收集平台，并且在研发在线的 InfoGraphics 制作工具。Visual.ly 希望可以建立起一个设 计师社区，让 InfoGraphics 的制作者从中收益，比如参与到付费的广告制作中，并借此来使 Visual.ly 获利。目前 Visual.ly 已经收 到了 50 万美元的种子投资。

总结

围绕着大数据的生态圈，我们参观了各个环节上杰出的创新者们。这些创新者顺应大数据时代的浪潮，敏锐地抓住了数据爆炸时代所产生的商业机会，他们或者已经成功，或者依旧在追求成功的路上。

诚然，受限于篇幅的原因，我们不可能将整个大数据生态环境中所有的创新者纳入其中，仅能选取一两个有代表性的公司，而且也没法进行更为深入的介绍。 但是，希望本文可以帮助读者了解整个大数据时代的商业全景，以及基于大数据的各种商业创新和技术创新，借此能够激发出更多的创新，并向这些先行者们致敬。

相关内容：

- Platform 创始人王敬文谈云计算和大数据
- Facebook 谈 Hadoop，Hive，HBase 和 A/B 测试
- 云时代的列式数据库——Sybase IQ15.3 新特性
- Twitter Storm：开源实时 Hadoop
- 一种开放的可互操作的云

特别专题

关系数据库还是 NoSQL 数据库

作者 孙立

上一篇简单的说明了为什么要使用 NoSQL。接下来我们看下如何把 NoSQL 引入到我们的项目中，我们到底要不要把 NoSQL 引入到项目中。

在过去，我们只需要学习和使用一种数据库技术，就能做几乎所有的数据库应用开发。因为成熟稳定的关系数据库产品并不是很多，而供你选择的免费版本就更加少了，所以互联网领域基本上都选择了免费的 MySQL 数据库。在高速发展的 WEB2.0 时代，我们发现关系数据库在性能、扩展性、数据的快速备份和恢复、满足需求的易用性上并不总是能很好的满足我们的需要，我们越来越趋向于根据业务场景选择合适的数据库，以及进行多种数据库的融合运用。几年前的一篇文章《One Size Fits All - An Idea Whose Time Has Come and Gone》就已经阐述了这个观点。

当我们在讨论是否要使用 NoSQL 的时候，你还需要理解 NoSQL 也是分很多种类的，在 NoSQL 百花齐放的今天，NoSQL 的正确选择比选择关系数据库还具有挑战性。虽然 NoSQL 的使用很简单，但是选择却是个麻烦事，这也正是很多人在观望的一个原因。

NoSQL 的分类

NoSQL 仅仅是一个概念，NoSQL 数据库根据数据的存储模型和特点分为很多种类。

类型	部分代表	特点
列存储	Hbase Cassandra Hypertable	顾名思义，是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的 IO 优势。
文档存储	MongoDB CouchDB	文档存储一般用类似 json 的格式存储，存储的内容是文档型的。这样也就有有机会对某些字段建立索引，实现关系数据库的某些功能。

key-value 存储	Tokyo Cabinet / Tyrant Berkeley DB MemcacheDB Redis	可以通过 key 快速查询到其 value。一般来说，存储不管 value 的格式，照单全收。（Redis 包含了其他功能）
图存储	Neo4J FlockDB	图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。
对象存储	db4o Versant	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。
xml 数据库	Berkeley DB XML BaseX	高效的存储 XML 数据，并支持 XML 的内部查询语法，比如 XQuery,Xpath。

以上 NoSQL 数据库类型的划分并不是绝对，只是从存储模型上来进行的大体划分。它们之间没有绝对的分界，也有交差的情况，比如 Tokyo Cabinet / Tyrant 的 Table 类型存储，就可以理解为是文档型存储，Berkeley DB XML 数据库是基于 Berkeley DB 之上开发的。

NoSQL 还是关系数据库

虽然 09 年出现了比较激进的文章《关系数据库已死》，但是我们心里都清楚，关系数据库其实还活得好好的，你还不能不用关系数据库。但是也说明了一个事实，关系数据库在处理 WEB2.0 数据的时候，的确已经出现了瓶颈。

那么我们到底是用 NoSQL 还是关系数据库呢？我想我们没有必要来进行一个绝对的回答。我们需要根据我们的应用场景来决定我们到底用什么。

如果关系数据库在你的应用场景中，完全能够很好的工作，而你又是非常善于使用和维护关系数据库的，那么我觉得你完全没有必要迁移到 NoSQL 上面，除非你是个喜欢折腾的人。如果你是在金融，电信等以数据为王的关键领域，目前使用的是 Oracle 数据库来提供高可靠性的，除非遇到特别大的瓶颈，不然也别贸然尝试 NoSQL。

然而，在 WEB2.0 的网站中，关系数据库大部分都出现了瓶颈。在磁盘 IO、数据库可扩展上都花费了开发人员相当多的精力来优化，比如做分表分库（database sharding）、主从复制、异构复制等等，然而，这些工作需要的技术能力越来越高，也越来越具有挑战性。如果你正在经历这些场合，那么我觉得你应该尝试一下 NoSQL 了。

选择合适的 NoSQL

如此多类型的 NoSQL，而每种类型的 NoSQL 又有很多，到底选择什么类型的 NoSQL 来作为我们的存储呢？这并不是一个很好回答的问题，影响我们选择的因素有很多，而选择也可能有多种，随着业务场景，需求的变更可能选择又会变化。我们常常需要根据如下情况考虑：

1. 数据结构特点。包括结构化、半结构化、字段是否可能变更、是否有大文本字段、数据字段是否可能变化。
2. 写入特点。包括 insert 比例、update 比例、是否经常更新数据的某一个小字段、原子更新需求。
3. 查询特点。包括查询的条件、查询热点的范围。比如用户信息的查询，可能就是随机的，而新闻的查询就是按照时间，越新的越频繁。

NoSQL 和关系数据库结合

其实 NoSQL 数据库仅仅是关系数据库在某些方面（性能，扩展）的一个弥补，单从功能上讲，NoSQL 的几乎所有的功能，在关系数据库上都能够满足，所以选择 NoSQL 的原因并不在功能上。

所以，我们一般会把 NoSQL 和关系数据库进行结合使用，各取所长，需要使用关系特性的时候我们使用关系数据库，需要使用 NoSQL 特性的时候我们使用 NoSQL 数据库，各得其所。

举个简单的例子吧，比如用户评论的存储，评论大概有主键 id、评论的对象 aid、评论内容 content、用户 uid 等字段。我们能确定的是评论内容 content 肯定不会在数据库中用 where content=’’ 查询，评论内容也是一个大文本字段。那么我们可以把 主键 id、评论对象 aid、用户 id 存储在数据库，评论内容存储在 NoSQL，这样数据库就节省了存储 content 占用的磁盘空间，从而节省大量 IO，对 content 也更容易做 Cache。

```
    //从MySQL中查询出评论主键id列表
    commentIds=DB.query("SELECT id FROM comments where aid='评论对象id' LIMIT 0,20");
    //根据主键id列表，从NoSQL取回评论实体数据
    CommentsList=NoSQL.get(commentIds);
```

NoSQL 代替 MySQL

在某些应用场合，比如一些配置的关系键值映射存储、用户名和密码的存储、Session 会话存储等等，用 NoSQL 完全可以替代 MySQL 存储。不但具有更高的性能，而且开发也更加方便。

NoSQL 作为缓存服务器

MySQL+Memcached 的架构中，我们处处都要精心设计我们的缓存，包括过期时间的设计、缓存的实时性设计、缓存内存大小评估、缓存命中率等等。

NoSQL 数据库一般都具有非常高的性能，在大多数场景下面，你不必再考虑在代码层为 NoSQL 构建一层 Memcached 缓存。NoSQL 数据本身在 Cache 上已经做了相当多的优化工作。

Memcached 这类内存缓存服务器缓存的数据大小受限于内存大小，如果用 NoSQL 来代替 Memcached 来缓存数据库的话，就可以不再受限于内存大小。虽然可能有少量的磁盘 IO 读写，可能比 Memcached 慢一点，但是完全可以用来缓存数据库的查询操作。

规避风险

由于 NoSQL 是一个比较新的东西，特别是我们选择的 NoSQL 数据库还不是非常成熟的产品，所以我们可能会遇到未知的风险。为了得到 NoSQL 的好处，又要考虑规避风险，鱼与熊掌如何兼得？

现在业内很多公司的做法就是数据的备份。在往 NoSQL 里面存储数据的时候还会往 MySQL 里面存储一份。NoSQL 数据库本身也需要进行备份（冷备和热备）。或者可以考虑使用两种 NoSQL 数据库，出现问题后可以进行切换（避免出现 digg 使用 Cassandra 的悲剧）。

总结

本文只是简单的从 MySQL 和 NoSQL 的角度分析如何选择，以及进行融合使用。其实在选择 NoSQL 的时候，你可能还会碰到关于 CAP 原则，最终一致性，BASE 思想的考虑。因为使用 MySQL 架构的时候，你也会碰到上面的问题，所以这里没有阐述。

关于作者

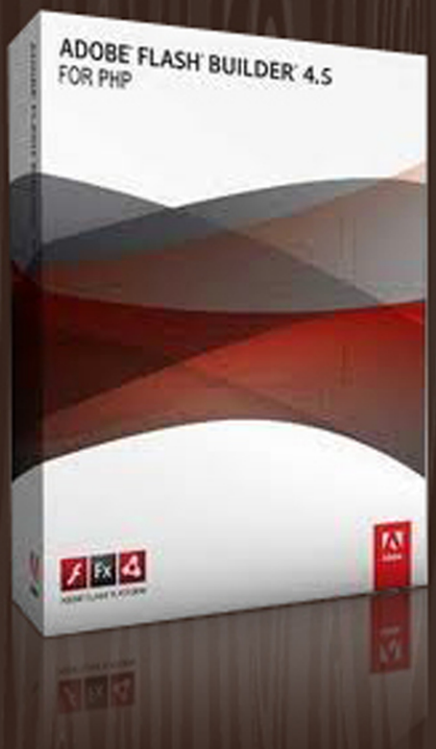
孙立，目前在凤凰网负责底层组的研发工作。曾就职于搜狐和 ku6。多年互联网从业经验和程序开发，对分布式搜索引擎的开发，高并发，大数据量网站系统架构 优化，高可用性，可伸缩性，分布式系统缓存,数据库分表分库（sharding）等有丰富的经验，并且对运维监控和自动化运维控制有经验。开源项目 phplock，phpbuffer 的作者。近期开发了一个 NOSQL 数据库存储 INetDB, 是 NoSQL 数据库爱好者。他的新浪微博是：http://t.sina.com.cn/sunli1223

原文链接：http://www.infoq.com/cn/news/2011/01/relation-db-nosql-db

相关内容：

- Google 开源 Key-Value 数据库 LevelDB
- QCon 全球企业开发大会在中国的三年发展总结
- MongoDB 开发应用实践（PHP/Perl)
- 视觉中国潘凡谈 MongoDB 应用实践
- Redis 复制与可扩展集群搭建

Adobe



Adobe Flash Builder 4.5 高级版

隆重发布

更强的功能
更多的设备
更少的编程

免费下载试用版

Adobe官方下载（本地服务器）

- Flash Builder 4.5高级版试用版（Windows，581.2M，点击免费高速下载）
- Flash Builder 4.5高级版试用版（Mac，679.8M，点击免费高速下载）
- Flash Builder 4.5 for PHP高级版试用版（Windows，956.5M，点击免费高速下载）
- Flash Builder 4.5 for PHP高级版试用版免费（Mac，911.3M，点击免费高速下载）

Adobe官方下载（国外服务器）

- Flash Builder 4.5高级版试用版（Windows和Mac版本，免费高速下载）
- Flash Builder 4.5 for PHP高级版试用版（Windows和Mac版本，免费高速下载）

InfoQ

更多下载

推荐文章 | Articles

向 Java 开发者介绍 Scala

作者 Thomas Alexandre 译者 王恒涛

Scala 结合了面向对象编程与函数编程思想，使用一种能够完全兼容 Java、可以运行在 Java 虚拟机上的、简洁的语法。对于函数编程风格的支持，尤其 是对于 Lambda 表达式的支持，能够有助于减少必须要编写的逻辑无关固定代码 ,也许让它可以更简单的关注要面对的任务本身，而相对的 Java 中对 Lamdba 表达式的支持要到预定于 2012 年发布的 JavaSE8 才会实现。本文就是对于 Scala 介绍。

作为第一步，先安装好最新的 Scala 发布包 Typesafe stack，打开命令行窗口，键入“scala”：这会启动 REPL(读入-运算 输出 循环)交互式编码环境。然后你就可以写下你的第一行 Scala 代码：

```
scala> val columbus : Int = 1492
columbus: Int = 1492
```

我们刚刚声明了一个类型为 Int 的变量，初始值为 1492，就像我们在 Java 里用语句 Int columbus = 1492;所做的一样。除了把类型放在变量之后这样一种反向的声明方式之外，Scala 在这里所表现出的不同是使用“val”来显性地把变量声明为不可变。如果我们想要修改这个变量:

```
scala> columbus = 1500
 <console>:8: error: reassignment to val
   columbus = 1500
         ^
```

请注意错误消息精确地指出了错误位于行的哪个位置。再尝试声明这个变量，但这一次用“var”，让其可变更。这样编译器拥有足够的智能来推断出 1492 是一个整数，你也就不需要指定类型了：

```
scala> var columbus = 1492
 columbus: Int = 1492

scala> columbus = 1500  columbus: Int = 1500
```

接下来，我们来定义一个类：

```
scala> case class Employee( name:String="guest",
                age:Int=30, company:String =  "DevCode" )
 defined class Employee
```

我们定义了一个类，名为 Employee，有三个不可变更的字段：name、age 和 company，各自有自己的缺省值。关键字“case”相当于 Java 里的 switch 语句，只不过要更为灵活。它说明该类具有模式匹配的额外机制，以及其他一些特性，包括用来创建实例的工厂方法（不需要使用“new”关键字来构造），同样的也不需要创建缺省的 getter 方法。与 Java 中不同的是，变量缺省下的访问控制是 public（而不是 protected），而 Scala 为公开变量创建一个 getter 方法，并命名为变量名。如果你愿意，你也可以把字段定义成可变且/或私有（private）的，只需要在参数之前使用“var”（例如：case class Person(private var name:String)）。

我们再来用不同方式创建一些实例，看看其他的特性，像是命名参数和缺省参数（从 Scala2.8 开始引入）：

```
scala> val guest = Employee()
           guest: Employee = Employee(guest,30,DevCode)
scala> val guestAge = guest.age // （age变量的缺省getter方法）
           guestAge: Int = 300
scala> val anna = Employee("Anna")
           anna: Employee = Employee(Anna,30,DevCode)
scala> val thomas = Employee("Thomas",41)
           thomas: Employee = Employee(Thomas,41,DevCode)
scala> val luke = Employee("Luke", company="LucasArt")
           luke: Employee = Employee(Luke,30,LucasArt)
scala> val yoda = luke.copy("Yoda", age=800)
           yoda: Employee = Employee(Yoda,800,LucasArt)
```

不过，下面的写法

```
scala> val darth = Employee("Darth", "DevCode")
 <console>:9: error: type mismatch;
 found : java.lang.String("DevCode")
 required: Int
 Error occurred in an application involving default arguments.
     val darth = Employee("Darth", "DevCode")
                   ^
```

是行不通的（可不是因为 Darth 不是 DevCode 的雇员！），这是由于构造函数在这个位置需要 age 作为参数，因为函数参数没有显性地进行命名。

现在我们再来看集合，这才是真正让人兴奋的地方。

有了泛型（Java5 以上），Java 可以遍历一个——比方说整数型列表，用下面这样的代码：

```
List<Integer> numbers = new arrayList<Integer>();
    numbers.add(1);
    numbers.add(2);
    numbers.add(3);
    for(Integer n:numbers) {
        System.out.println("Number "+n);
}
```

运行的结果是

```
Number 1
Number 2
Number 3
```

Scala 对于可变集合和不可变集合进行了系统性地区别处理，不过鼓励使用不可变集合，也因此在缺省情况下创建不可变集合。这些集合是通过模拟的方式实现添加、更新和删除操作，在这些操作中，不是修改集合，而是返回新的集合。

与前面的 Java 代码等价的 Scala 代码可能像下面这样：

```
scala> val numbers = List(1,2,3)
 numbers: List[Int] = List(1, 2, 3)

scala> for (n <- numbers) println("Number "+n)
            Number 1  Number 2  Number 3
```

这里的“for”循环语法结构非常接近于 Java 的命令式编程风格。在 Scala（以及 Java 虚拟机上其他很多语言如：Groovy、JRuby 或 JPython）里还有另外一种方式来实现上面的逻辑。这种方式使用一种更加偏向函数编程的风格，引入了 Lambda 表达式（有时也称为闭包——closure）。简单地说，Lambda 表达式就是你可以拿来当作参数传递的函数。这些函数使用参数作为输入（在我们的例子中就是“n”整型变量），返回语句作为函数体的最终语句。他们的形式如下

```
functionName { input =>
    body
```

```
}

scala> numbers.foreach { n:Int =>     // 按回车键继续下一行
    | println("Number "+n)
    | }  Number 1  Number 2  Number 3
```

上面的例子中，函数体只有一条语句（println……），返回的是单位（Unit，也就是“空结果”），也就是大致相当于 Java 中的 void，不过有一点不同的是——void 是不返回任何结果的。

除了只打印出我们的数值列表以外，应该说我们更想做的是处理和变换这些元素，这时我们需要调用方法来生成结果列表，以便后面接着使用。让我们尝试一些例子：

```
scala> val reversedList = numbers.reverse
           reversedList: List[Int] = List(3, 2, 1)

scala> val numbersLessThan3 = numbers.filter { n => n < 3 }
           numbersLessThan3: List[Int] = List(1, 2)
scala> val oddNumbers = numbers.filterNot { n => n % 2 == 0 }
           oddNumbers: List[Int] = List(1, 3)
scala> val higherNumbers = numbers.map { n => n + 10 }
           higherNumbers: List[Int] = List(11, 12, 13)
```

最后的这一个变换“map”非常有用，它对列表的每一个元素应用闭包，结果是一个同样大小的、包含了每个变换后元素的列表。

我们在这里还想介绍最后的一个方法，就是“foldLeft”方法，它把状态从一个元素传播到另一个元素。比如说，要算出一个列表里所有元素的和，你需要累加它们，并在切换元素的时候保存中间的计数：

```
scala> val sumOfNumbers = numbers.foldLeft(0) { (total,element) =>
    | total + element
    | }
 sumOfNumbers: Int = 6
```

作为第一个变量传递给 foldLeft 的值 0 是初始值（也就是说在把函数用到第一个列表元素的时候 total=0）。(total,element)代表了一个 Tuple2，在 Scala 里这是一个二元组（就像要表示三维空间坐标，经常要用到 Tuple3(x,y,z)等等）。注意在合计时，Scala 的编程接口实际上提供了一个“sum”方法，这样上一条语句就可以写成：

```
scala> val sumOfNumbers = numbers.sum
```

```
    sumOfNumbers: Int = 6
```

还有许多其他的类似的集合变换方法，你可以参照 scaladoc API。你也可以把这些方法组合起来（例如：numbers.reverse.filter…… ），让代码更加简洁，不过这样会影响可读性。

最后，{ n => n + 10 }还可以简单地写成(_ + 10)，也就是说如果输入参数只是用于你调用的方法，则不需要声明它；在我们的例子里，“n” 被称为匿名变量，因为你可以把它用任何形式来代替，比如说 “x” 或者 “number”，而下划线则表示一处需要用你的列表的每个元素来填补的空白。（与 “_” 的功能类似，Groovy 保留了关键字 “it”，而 Python 则使用的是 “self”）。

```
scala> val higherNumbers = numbers.map(_+10)
           higherNumbers: List[Int] = List(11, 12, 13)
```

在介绍了对整数的基本处理后，我们可以迈入下一个阶段，看看复杂对象集合的变换，例如使用我们上面所定义的 Employee 类：

```
scala> val allEmployees = List(luke,anna,guest,yoda,thomas)
 allEmployees: List[Employee] = List(Employee(Luke,30,LucasArt),
                Employee(Anna,30,DevCode),
                Employee(guest,30,DevCode),
                Employee(Yoda,800,LucasArt),
                Employee(Thomas,41,DevCode))
```

从这个五个元素的列表里，我们可以应用一个条件来过滤出应用匿名方法后返回值为 True 的雇员，这样就得到了——比方说属于 DevCode 的雇员：

```
scala> val devcodeEmployees = allEmployees.filter { _.company == "DevCode" }
 devcodeEmployees: List[Employee] = List(Employee(Anna,30,DevCode),
             Employee(guest,30,DevCode),
             Employee(Thomas,41,DevCode))

scala> val oldEmployees = allEmployees.filter(_.age > 100).map(_.name)
           oldEmployees: List[String] = List(Yoda)
```

假设我们手头的 allEmployees 集合是我们使用 SQL 查询获得的结果集，查询语句可能类似于 “SELECT * FROM employees WHERE company = ‘DevCode’ ”。现在我们可以把 List[Employee] 变换到以 company 名称作为键、属于该公司的所有员工的列表作为值的 Map 类型，这样就可以把雇员按 company 来排序：

```
scala> val sortedEmployees = allEmployees.groupBy(_.company)
           sortedEmployees:
                scala.collection.immutable.Map[String,List[Employee]] =
                        Map(DevCode - > List(Employee(Anna,30,DevCode),
                                   Employee(guest,30,DevCode),
                                   Employee(Thomas,41,DevCode)),
                                   LucasArt ->
                                      List(Employee(Luke,30,LucasArt),
```

```
                                                Employee(Yoda,800,LucasArt)))
```

每一个列表已经作为一个值存入了（键——值）哈希表，为了示范如何进一步处理这些列表，可以设想我们需要计算每个公司的雇员平均年龄。

这具体意味着我们必须要计算每个列表的每个雇员的的“age”字段的和，然后除以该列表中雇员的数量。让我们先计算一下 DevCode：

```
    scala> devcodeEmployees
     res4: List[Employee] = List(Employee(Anna,30,DevCode),
                Employee(guest,30,DevCode),  Employee(Thomas,41,DevCode))

    scala> val devcodeAges =
                   devcodeEmployees.map(_.age)  devcodeAges: List[Int] =
                          List(30, 30, 41)
    scala> val devcodeAverageAge =
               devcodeAges.sum / devcodeAges.size  devcodeAverageAge: Int =
                        33
```

回到我们的 Map (key:String ->value:List[Employee])，下面是个更加一般性的例子。我们现在可以归并并计算每个公司的平均年龄，要做的只是写几行代码：

```
    scala> val averageAgeByCompany = sortedEmployees.map{ case(key,value)=>
        |
value(0).copy(name="average",age=(value.map(_.age).sum)/value.size)}
     averageAgeByCompany: scala.collection.immutable.Iterable[Employee] =
          List(Employee(average,33,DevCode), Employee(average,415,LucasArt))
```

这里的“case(key,value)”说明了 Scala 提供的模式匹配机制是多么强大。请参考 Scala 的文档来获取更多的信息。

到这里我们的任务就完成了。我们实现的是一个简单的 Map-Reduce 算法。由于每个公司雇员的归并是完全独立于其他公司，这个算法非常直观地实现了并行计算。

在后面的附录里给出了此算法的等价的实现，分为 Java 版本和 Scala 版本。

参考

The typesafe stack.

附录

```
    Map Reduce: Java

    public class Employee {
        final String name;
        final Integer age;
```

```
    final String company;

    public Employee(String name, Integer age, String company) {
        this.name = name == null ? "guest" : name;
        this.age = age == null ? 30 : age;
        this.company = company == null ? "DevCode" : company;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getCompany() {
        return company;
    }

    @Override
    public String toString() {
        return "Employee [name=" + name + ", age=" + age + ",
              company="
              + company + "]";
    }
}

class Builder {
    String name, company;
    Integer age;

    Builder(String name) {
        this.name = name;

    }
    Employee build() {
        return new Employee(name, age, company);
    }
    Builder age(Integer age) {
        this.age = age;
        return this;
    }
    Builder company(String company) {
        this.company = company;
        return this;
    }
}

import java.util.ArrayList;
```

```
import java.util.Collection;
import java.util.List;
import com.google.common.base.Function;
import com.google.common.collect.ImmutableListMultimap;
import com.google.common.collect.ImmutableSet;
import com.google.common.collect.Multimaps;

public class MapReduce {

    public static final void main(String[] args) {
        Employee guest = new Builder("Guest").build();
        Employee anna = new Builder("Anna").build();
        Employee thomas = new Builder("Thomas").age(41).build();
        Employee luke = new
            Builder("Luke").company("LucasArt").build();
        Employee yoda = new
            Builder("Yoda").age(800).company("LucasArt").build();

        Collection<Employee> employees = new ArrayList<Employee>();
        employees.add(guest);
        employees.add(anna);
        employees.add(thomas);
        employees.add(luke);
        employees.add(yoda);

        ImmutableListMultimap<String, Employee>
            personsGroupByCompany = Multimaps.index(employees,
                        new Function<Employee,String>() {

               public String apply(Employee person) {
                  return person.getCompany();
                }

             });

        ImmutableSet<String> companyNamesFromMap =
            personsGroupByCompany.keySet();

        List<Employee> averageAgeByCompany = new
            ArrayList<Employee>();

        for(String company: companyNamesFromMap) {
             List<Employee> employeesForThisCompany =
                personsGroupByCompany.get(company);
             int sum = 0;
             for(Employee employee: employeesForThisCompany) {
                sum+= employee.getAge();
```

```
                }
                averageAgeByCompany.add(new

Employee("average",sum/employeesForThisCompany.size(),company));
        }
        System.out.println("Result: "+averageAgeByCompany);
       }
    }

    MapReduce.scala:

    case class Employee(name: String = "guest", age: Int = 30, company: String
= "DevCode")

       object MapReduce {
           def main(args: Array[String]): Unit = {

           val guest = Employee()
           val anna = Employee("Anna")
           val thomas = Employee("Thomas", 41)
           val luke = Employee("Luke", company = "LucasArt")
           val yoda = luke.copy("Yoda", age = 800)

           val allEmployees = List(luke, anna, guest, yoda, thomas)
           val sortedEmployees = allEmployees.groupBy(_.company)
           val averageAgeByCompany = sortedEmployees.map { case (key, value) =>
               value(0).copy(name = "average", age = (value.map(_.age).sum) /
value.size)
         }
           println("Result: "+averageAgeByCompany)
       }
    }
```

关于作者

Thomas Alexandre 是 DevCode 的高级咨询顾问，专注于 Java 和 Scala 软件开发。他热爱技 术，热衷于分享知识，永远在寻求方法、采用新的开源软件和标准来实现更加有效的编程。在十四年的 Java 开发经验之外，过去几年他集中精力在新的编程语言 和 Web 框架上，例如 Groovy/Grails 和 Scala/Lift。Thomas 从法国里尔大学获得了计算机科学博士学位，在卡耐基梅隆大学度过了 两年的博士后研究生涯，研究方向是安全和电子商务。

原文链接：http://www.infoq.com/cn/articles/scala-for-java-devs

推荐文章 | Articles

HTML 5 or Silverlight?

作者 Daniel Jebaraj 译者 高翌翔

阿尔伯特•爱因斯坦在处于罕见的消沉时期时曾经说过，手段的完善和目标的混乱似乎刻画了这个时代。有人可能认为此话出自软件开发者之口，而非物理学家之口。

开发平台持续演变和改进，这常常导致我们只见树木不见森林。从正在进行的关于 Microsoft Silverlight 和 HTML5 的争论中就能够看到这种困惑。

有人认为由于 HTML5 持续增长的势头，微软将会放弃 Silverlight。但这似乎不太可能。尽管微软对于 Silverlight 的战略已经转变，不再吹捧将 Silverlight 作为提供跨平台运行时的承载工具，但是微软在继续推动 Silverlight 成为 Windows Phone 以及一些媒体和业务线（line-of-business）应用程序的开发平台。Silverlight 并未消失。事实上，最终形式的 Silverlight 5 将于今年推出，而且那些用于维持微软传奇的工具也将证实这个观点。

虽然 HTML5 标准仍处于草案阶段，但是它在将来肯定会成为主导的跨平台解决方案，甚至连微软也承认，称 HTML 是“唯一适用于所有应用的、真正的跨平台解决方案。”

基本上，Silverlight 和 HTML5 都有各自的位置和用途，我们可以对两种工具之间的相似性和差异进行仔细观察，这样就会找到答案。

表面的相似性

乍看上去，在许多方面 HTML5 和 Silverlight 都很相似。此类相似性与易于部署、丰富的用户界面、以及交互模型等联系在一起。

桌面程序担忧的问题之一是部署，特别是那些工作在 Windows 环境下的 Windows 开发者尤为关注。部署对于那些工作在中大型企业的人们来说也是一个问题。桌面应用程序的部署足迹对于他们而言是相当麻烦的，因为他们必须确保某个确定版本的运行库可以正常工作，而且他们必须将更新安装到每台机器上。因此部署对于功能丰富的桌面应用程序而言已成为痛点。

在某些情况下，因为业务原因，企业还将坚持使用桌面应用程序，它们具有更好的性能，并

且能够更好地利用本地硬件——只是以更加简单、无缝的方式来做事。此外，对于桌面程序平台的开发场景而言，有着杰出程序员的知识积累。这些原因能够胜过对于简单部署的需求。

对于那些将部署置于核心优先级位置上的客户而言，完全的 Web 解决方案可能更可取，尽管所交付的丰富功能相对于在桌面应用程序中等同的功能而言还不能让所有客户感到满意。虽然我们正在寻找替代解决方案，但是必须平衡此类问题。

一般情况下，Adobe Air 和 Silverlight 都是不错的解决方案。在微软的工具库中，Silverlight 是非常好的解决方案，因为尽管存在运行库，但是在构造之初就考虑到无缝部署，而且便于用户更新。基本上，微软很注重更新体验。你不必为将应用程序下载到最终用户的机器上而担心。

HTML5 提供了类似的部署方案；然而由于它依赖浏览器，因此存在一些隐藏的陷阱。使用 HTML5 解决方案的团队必须可以相当肯定，他们的客户可以访问并安装最新版本的浏览器。此外，HTML5 有着广泛的浏览器支持：IE9、Chrome 以及 Firefox；在 IE10 上 HTML 5 会更有效地工作。未来所有的移动客户端都会支持 HTML5，尽管现在它们还不支持。

简而言之，虽然 Silverlight 和 HTML5 之间存在细微的部署差异，但是二者的部署模型基本上是无缝的。

就用户界面（UI）的丰富程度而言，Silverlight 则具备一些优势。如果是为了更快地完成丰富的用户界面，那么 Silverlight 可能是更好的解决方案。不过，HTML5 在这方面正迎头赶上 Silverlight。用不了不久，HTML5 就将拥有更多预先封装的内容，从而使我们可以更便捷地构建丰富的用户界面环境。

这两款工具在交互模型方面也是类似的。二者都不要求用户等待页面刷新，而且二者的使用方式都与桌面应用程序类似。

更仔细地研究

更仔细地研究一下，我们就会发现，HTML5 和 Silverlight 之间强烈的功能相似性正在趋于消退。首先，与真正基于 Web 的部署比较而言，Silverlight 更适合于拥有部署环境相对控制权的企业内部网应用程序。

如果你更深入地了解 Silverlight 的部署方案，那么你会发现 Silverlight 还不是真正的最终用户解决方案。因此，如果开发者的目的是让应用程序的用户下载 Silverlight 并在用户的机器上运行 Silverlight 的话，那么开发者需要知道对于客户的清晰描述。**客户是否将拥有可以运行**

Silverlight 的系统？某些操作是否被允许？

例如，如果用户访问 Amazon.com 的时候，弹出了下载 Silverlight 客户端程序的提示，这可能并不是良好的体验。对于网站使用者而言，无缝体验越多越好。

然而，对于企业内部网解决方案的情况——开发者对于机器拥有更多控制权，并且知道都是装有 Windows 操作系统的机器——尽管开发者可能并不需要对桌面应用程序拥有相当程度的控制权，但是他们知道那些机器有能力运行 Silverlight。这就为开发者提供了极大的灵活性。

现在我们可以选择追随 Silverlight；当与 HTML5 比较时，Silverlight 无疑是一种更富生产力的开发体验。

微软有些能够让创建和部署 Silverlight 应用程序更轻松的优秀工具。虽然 Silverlight 更有条理——相关工具位于他们自己的分类之下，而 HTML5 还需要多做一些工作。如果你确信你对自己的部署环境比较熟悉的话，或许你可以对那些关系亲密的客户这样说，“这些是最低要求”；那么 Silverlight 则更适合。而且将进一步得到高质量 Silverlight 工具的支持，那些工具使开发者能以一种快速拖拽的方式来创建 Silverlight 应用程序。通过使用内建的控制抽象模型以及微软已经提供的本地控件，Silverlight 还使用户界面开发和大多数其他开发具有更高的生产力。

HTML5 另一方面需要得到你当前使用的 Web 应用程序开发工具的支持。如果你正在 ASP.NET 环境下开发，那么工具就是 Visual Studio .NET，而 VS 并未提供任何对 HTML5 的工具支持，目前许多开发平台可能也存在着类似情况。

选择通用语言还是尽力理解？

编程语言是另一个考虑因素。C#（Silverlight）比 JavaScript（HTML5）更易于使用和调试。回归到工具和语言的本质。JavaScript 使用起来真的是很不一样；即便是经验丰富的 JavaScript 程序员也知道，由于 JavaScript 生来就是过程式的、类型不安全的语言，因此 JavaScript 理解起来确实有点儿困难，然而 C#则是一种面向对象、类型安全的语言。这基本上是说，与 JavaScript 相比，使用 C#可以更好地编写和维护大量代码。

Silverlight 的局限性和其他考虑因素

如果需要移动部署，那么 Silverlight 是有限制的。目前，只有 Windows Phone 支持 Silverlight。其他平台未来可能支持 Silverlight，但是这还不一定。而且在短期内不可能发生任何改变。

目前，要想开发移动客户端可使用的 Silverlight 应用程序，那么相应 Windows Phone 设备必须先获得授权才行。

如果开发者无法控制移动客户端，而又希望支持那些设备，那么 HTML5 就是个切实可行的选择。由于 iOS、Android 3 已经支持 HTML5，并且 Windows 承诺在 IE10 下支持 HTML5，因此 HTML5 现已成为明确之选。

然而，Silverlight 可以提供比 HTML5 更好的性能。在过去的几个月里，微软实现了针对 Silverlight 5 的硬件解决方案，因此在一些新型号的机器上 Silverlight 5 相对 HTML5 具有轻微的性能优势。但是 HTML5 肯定会在将来赶上来。例如，IE9 对于 HTML5 做了许多改进，因此这个性能差异只是暂时性的，而非决定性的。这算不上多大的差别。

很显然，HTML5 是一种基于标准的环境，这是一些开发者极为关注的事情，而其他人却不大关注。如果是使用微软工具库的开发者，那么他就会理解并欣赏那些标准；然而，通过理解微软将在一段时间后也会遵循这一标准，这种态度会有所缓解，因此他们并不是不愿采用微软的专利技术。

在许多情况下都存在遵循标准的推动力，所以如果情况发生改变，那么迁移路径也很容易。HTML5 是基于标准的，并且随着 HTML5 的到来，它正在被传播到全世界。使用 HTML5 的前途是很光明的——使用 Silverlight 却并非如此。

相对于 HTML5，Silverlight 有一个相当大的优势，即 Silverlight 程序中 90%至 95%的代码可以与桌面应用程序共享。如果你拥有一款成熟的桌面应用程序以及相应的 Web 移植解决方案，那么使用 Silverlight 模型则会更容易实现。

使用 HTML5，开发者可以保持用户界面的独立，并且拥有一个业务层；然而，必须在两个平台上编写大量的用户界面代码，这么做会付出更多精力而且几乎无法共享任何代码。

想个有根据的目的

最后，在选择工具时心中必须有着明确的目的，从而避免爱因斯坦所发现的混乱。选择正确道路是基于人们希望尽可能减小发生重大错误的可能性。例如，开发者可能熟悉 WPF 和 Silverlight，并因此选择了这条道路。但是在对想做的应用程序进行探索后，他们发现可能希望在六个月到一年内推出某种移动客户端。更重要的是，他们想让人们能通过多种设备进行访问。

此时，只要他们准备最终转到 HTML5 或者为每个移动平台编写本地应用程序，即使之前选定了 Silverlight 解决方案也没有关系。如果并非如此，那么这将是一个非常昂贵的选择。

从另一方面来说：由于每个人都在说，“这是未来的趋势。”，那么这可能成为一种追随 HTML5 的推动力；因此，即使客户的所有开发者都拥有 Visual Studio，而且被授权可使用那些最新和最好的微软工具，只要做出了决定，就要坚持朝着确定的方向走下去。

如果他们可以很好地控制部署环境，并且在接来下的六个月内他们将不引入任何 Linux 或 Mac 设备，那么他们也许正在犯下昂贵的错误。当然他们可以追随 Silverlight，这将使他们能够更快地进入市场，也可以更有效地利用他们当前的技能和工具集。

归根结底：只要开发者能做出明智的选择，无论 Silverlight 还是 HTML5 都不错。

应用最佳选择

既然微软必须强烈支持 HTML5，那么现在微软已经无法控制移动市场。他们不再发号施令，而值得怀疑的是如果他们能做到的话，是否会那么做呢。微软的策略是尽可能地应用最好的工具和平台，无论它是否是专有的。

根据微软的 声明，“在网络上，Silverlight 的用途从来都不是要替代 HTML；它做的是一些 HTML（以及其他技术）不能以方便开发者的方式去深入了解的事 情。微软仍然致力于使用 Silverlight 通过使 HTML 尚未涉及的解决方案成为可能来扩展 Web。从 HTML 页面中简单丰富的数据孤岛，到浏览器中 完全类似于桌面程序的应用程序，甚至更强大，Silverlight 使得应用程序可以提供各种用户所需的丰富体验。”

显而易见，在可预见的未来 Silverlight 和 HTML5 将共同繁荣。

关于作者

Daniel Jebaraj 在 Syncfusion 公司领导产品开发，对于在每个微软平台上的开发者而言，Syncfusion 是一家企业级技术合作伙伴，提供贯穿整个应用程序生命周期，与面向服务方式相结合的最广泛的.NET 组件和控件。

他监督整体产品开发，并为特定版本发布制定计划。通过积极地与客户接洽，Daniel 确保新产品中的每一处改进都基于客户反馈。此前，Daniel 作为开发副总裁，专注于推动 Syncfusion 的产品开发。

在 2001 年加入 Syncfusion 以前，Daniel 在 Rogue Wave 软件公司管理开发团队。

Daniel 拥有克莱姆森大学的工业工程硕士学位。

原文链接：http://www.infoq.com/cn/articles/Html5-or-Silverlight

推荐文章 | Articles

解析 JDK 7 的 Garbage-First 收集器

作者 周志明

Garbage-First（后文简称 G1）收集器是当今收集器技术发展的最前沿成果，在 Sun 公司给出的 JDK RoadMap 里面，它被视作 JDK 7 的 HotSpot VM 的一项重要进化特征。从 JDK 6u14 中开始就有 Early Access 版本的 G1 收集器供开发人员实验、试用，虽然在 JDK 7 正式版发布时，G1 收集器仍然没有摆脱"Experimental"的标签，但是相信不久后将会有一个成熟的商用版本跟随某个 JDK 7 的更新包发布出来。

因版面篇幅限制，笔者行文过程中假设读者对 HotSpot 其他收集器（例如 CMS）及相关 JVM 内存模型已有基本的了解，涉及到基础概念时，没有再延伸介绍，读者可参考相关资料。

G1 收集器的特点

G1 是一款面向服务端应用的垃圾收集器，Sun（Oracle）赋予它的使命是（在比较长期的）未来可以替换掉 JDK 5 中发布的 CMS（Concurrent Mark Sweep）收集器，与其他 GC 收集器相比，G1 具备如下特点：

- 并行与并发：G1 能充分利用多 CPU、多核环境下的硬件优势，使用多个 CPU（CPU 或者 CPU 核心）来缩短 Stop-The-World 停顿的时间，部分其他收集器原本需要停顿 Java 线程执行的 GC 动作，G1 收集器仍然可以通过并发的方式让 Java 程序继续执行。
- 分代收集：与其他收集器一样，分代概念在 G1 中依然得以保留。虽然 G1 可以不需其他收集器配合就能独立管理整个 GC 堆，但它能够采用不同的方式去处理新创建的对象和已经存活了一段时间、熬过多次 GC 的旧对象以获取更好的收集效果。
- 空间整合：与 CMS 的"标记-清理"算法不同，G1 从整体看来是基于"标记-整理"算法实现的收集器，从局部（两个 Region 之间）上看是基于"复制"算法实现，无论如何，这两种算法都意味着 G1 运作期间不会产生内存空间碎片，收集后能提供规整的可用内存。这种特性有利于程序长时间运行，分配大对象时不会因为无法找到连续内存空间而提前触发下一次 GC。

- 可预测的停顿：这是 G1 相对于 CMS 的另外一大优势，降低停顿时间是 G1 和 CMS 共同的关注点，但 G1 除了追求低停顿外，还能建立可预测的停顿时间模型，能让使用者明确指定在一个长度为 M 毫秒的时间片段内，消耗在垃圾收集上的时间不得超过 N 毫秒，这几乎已经是实时 Java（RTSJ）的垃圾收集器特征了。

实现思路

在 G1 之前的其他收集器进行收集的范围都是整个新生代或者老年代，而 G1 不再是这样。使用 G1 收集器时，Java 堆的内存布局与就与其他收集器有很大差别，它将整个 Java 堆划分为多个大小相等的独立区域（Region），虽然还保留有新生代和老年代的概念，但新生代和老年代不再是物理隔离的了，它们都是一部分 Region（不需要连续）的集合。

G1 收集器之所以能建立可预测的停顿时间模型，是因为它可以有计划地避免在整个 Java 堆中进行全区域的垃圾收集。G1 跟踪各个 Region 里面的垃圾堆积的价值大小（回收所获得的空间大小以及回收所需时间的经验值），在后台维护一个优先列表，每次根据允许的收集时间，优先回价值最大的 Region（这也就是 Garbage-First 名称的来由）。这种使用 Region 划分内存空间以及有优先级的区域回收方式，保证了 G1 收集器在有限的时间内获可以获取尽可能高的收集效率。

G1 把内存“化整为零”的思路，理解起来似乎很容易理解，但其中的实现细节却远远没有现象中简单，否则也不会从 04 年 Sun 实验室发表第一篇 G1 的论文拖至今将近 8 年时间都还没有开发出 G1 的商用版。笔者举个一个细节为例：把 Java 堆分为多个 Region 后，垃圾收集是否就真的能以 Region 为单位进行了？听起来顺理成章，再仔细想想就很容易发现问题所在：Region 不可能是孤立的。一个对象分配在某个 Region 中，它并非只能被本 Region 中的其他对象引用，而是可以与整个 Java 堆任意的对象发生引用关系。那在做可达性判定确定对象是否存活的时候，岂不是还得扫描整个 Java 堆才能保障准确性？这个问题其实并非在 G1 中才有，只是在 G1 中更加突出了而已。在以前的分代收集中，新生代的规模一般都比老年代要小许多，新生代的收集也比老年代要频繁许多，那回收新生代中的对象也面临过相同的问题，如果回收新生代时也不得不同时扫描老年代的话，Minor GC 的效率可能下降不少。

在 G1 收集器中 Region 之间的对象引用以及其他收集器中的新生代与老年代之间的对象引用，虚拟机都是使用 Remembered Set 来避免全堆扫描的。G1 中每个 Region 都有一个与之对应的 Remembered Set，虚拟机发现程序在对 Reference 类型的数据进行写操作时，会产生一个 Write Barrier 暂时中断写操作，检查 Reference 引用的对象是否处于不同的 Region 之中

（在分代的例子中就是检查引是否老年代中的对象引用了新生代中的对象），如果是，便通过 CardTable 把相关引用信息记录到被引用对象所属的 Region 的 Remembered Set 之中。当进行内存回收时，GC 根节点的枚举范围中加入 Remembered Set 即可保证不对全堆扫描也不会有遗漏。

运作过程

如果不计算维护 Remembered Set 的操作，G1 收集器的运作大致可划分为以下几个步骤：

- 初始标记（Initial Marking）
- 并发标记（Concurrent Marking）
- 最终标记（Final Marking）
- 筛选回收（Live Data Counting and Evacuation）

对 CMS 收集器运作过程熟悉的读者，一定已经发现 G1 的前几个步骤的运作过程和 CMS 有很多相似之处。初始标记阶段仅仅只是标记一下 GC Roots 能直接关联到的对象，并且修改 TAMS（Next Top at Mark Start）的值，让下一阶段用户程序并发运行时，能在正确可用的 Region 中创建新对象，这阶段需要停顿线程，但耗时很短。并发标记阶段是从 GC Root 开始对堆中对象进行可达性分析，找出存活的对象，这阶段耗时较长，但可与用户程序并发执行。而最终标记阶段则是为了修正并发标记期间，因用户程序继续运作而导致标记产生变动的那一部分标记记录，虚拟机将这段时间对象变化记录在线程 Remembered Set Logs 里面，最终标记阶段需要把 Remembered Set Logs 的数据合并到 Remembered Set 中，这阶段需要停顿线程，但是可并行执行。最后筛选回收阶段首先对各个 Region 的回收价值和成本进行排序，根据用户所期望的 GC 停顿时间来制定回收计划，从 Sun 透露出来的信息来看，这个阶段其实也可以做到与用户程序一起并发执行，但是因为只回收一部分 Region，时间是用户可控制的，而且停顿用户线程将大幅提高收集效率。通过图 1 可以比较清楚地看到 G1 收集器的运作步骤中并发和需要停顿的阶段。

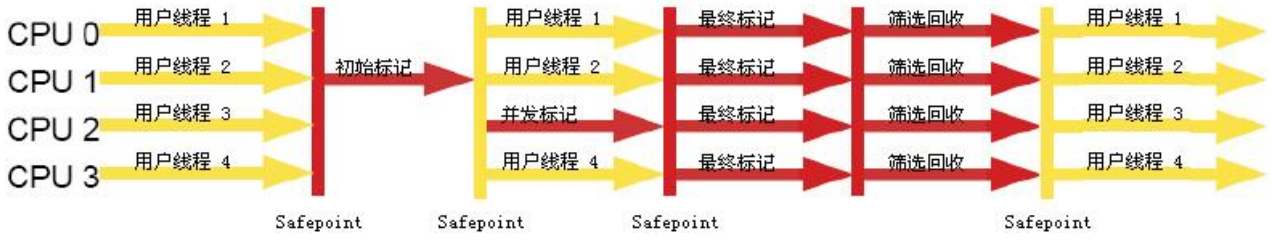


图 1 G1 收集器运行示意图

G1 收集器的实际性能

由于目前还没有成熟的版本，G1 收集器几乎可以说还没有经过实际应用的考验，网上关于 G1 收集器的性能测试非常贫乏，笔者没有 Google 到有关的生产环境下的性能测试报告。强调“生产环境下的测试报告”是因为对于垃圾收集器来说，仅仅通过简单的 Java 代码写个 Microbenchmark 程序来创建、移除 Java 对象，再用-XX:+PrintGCDetails 等参数来查看 GC 日志是很难做到准衡量其性能的（为何 Microbenchmark 的测试结果不准确可参见笔者这篇博客：http://icyfenix.iteye.com/blog/1110279）。因此关于 G1 收集器的性能部分，笔者引用了 Sun 实验室的论文《Garbage-First Garbage Collection》其中一段测试数据，以及一段在 StackOverfall.com 上同行们对 G1 在真实生产环境下的性能分享讨论。

Sun 给出的 Benchmark 的执行硬件为 Sun V880 服务器（8×750MHz UltraSPARC III CPU、32G 内存、Solaris 10 操作系统）。执行软件有两个，分别为 SPECjbb（模拟商业数据库应用，堆中存活对象约为 165MB，结果反映吐量和最长事务处理时间）和 telco（模拟电话应答服务应用，堆中存活对象约为 100MB，结果反映系统能支持的最大吞吐量）。为了便于对比，还收集了一组使用 ParNew+CMS 收集器的测试数据。所有测试都配置为与 CPU 数量相同的 8 条 GC 线程。

在反应停顿时间的软实时目标（Soft Real-Time Goal）测试中，横向是两个测试软件的时间片段配置，单位是毫秒，以（X/Y）的形式表示，代表在 Y 毫秒内最大允许 GC 时间为 X 毫秒（对于 CMS 收集器，无法直接指定这个目标，通过调整分代大小的方式大致模拟）。纵向是两个软件在对应配置和不同的 Java 堆容量下的测试结果，V%、avgV%和 wV%分别代表的含义为：

- V%：表示测试过程中，软实时目标失败的概率，软实时目标失败即某个时间片段中实际 GC 时间超过了允许的最大 GC 时间。
- avgV%：表示在所有实际 GC 时间超标的时间片段里，实际 GC 时间超过最大 GC 时间的平均百分比（实际 GC 时间减去允许最大 GC 时间，再除以总时间片段）。
- wV%：表示在测试结果最差的时间片段里，实际 GC 时间占用执行时间的百分比。

测试结果如下表所示：

表 1：软实时目标测试结果

Benchmark / confguration	Soft real-time goal compliance statistics by Heap Size								
	V%	avgV%	wV%	V%	avgV%	wV%	V%	avgV%	wV%

SPECjbb		512M			640M			768M		
G1	(100/200)	4.29%	36.40%	100.00%	1.73%	12.83%	63.31%	1.68%	10.94%	69.67%
G1	(150/300)	1.20%	5.95%	15.29%	1.51%	4.01%	20.80%	1.78%	3.38%	8.96%
G1	(150/450)	1.63%	4.40%	14.32%	3.14%	2.34%	6.53%	1.23%	1.53%	3.28%
G1	(150/600)	2.63%	2.90%	5.38%	3.66%	2.45%	8.39%	2.09%	2.54%	8.65%
G1	(200/800)	0.00%	0.00%	0.00%	0.34%	0.72%	0.72%	0.00%	0.00%	0.00%
CMS	(150/450)	23.93%	82.14%	100.00%	13.44%	67.72%	100.00%	5.72%	28.19%	100.00%
Telco		384M			512M			640M		
G1	(50/100)	0.34%	8.92%	35.48%	0.16%	9.09%	48.08%	0.11%	12.10%	38.57%
G1	(75/150)	0.08%	11.90%	19.99%	0.08%	5.60%	7.47%	0.19%	3.81%	9.15%
G1	(75/225)	0.44%	2.90%	10.45%	0.15%	3.31%	3.74%	0.50%	1.04%	2.07%
G1	(75/300)	0.65%	2.55%	8.76%	0.42%	0.57%	1.07%	0.63%	1.07%	2.91%
G1	(100/400)	0.57%	1.79%	6.04%	0.29%	0.37%	0.54%	0.44%	1.52%	2.73%
CMS	(75/225)	0.78%	35.05%	100.00%	0.54%	32.83%	100.00%	0.60%	26.39%	100.00%

从上面结果可见，对于 telco 来说，软实时目标失败的概率控制在 0.5%~0.7%之间，SPECjbb 就要差一些，但也控制在 2%~5%之间，概率随着（X/Y）的比值减小而增加。另一方面，失败时超出允许 GC 时间的比值随着总时间片段增加而变小（分母变大了嘛），在（100/200）、512MB 的配置下，G1 收集器出现了某些时间片段下 100%时间在进行 GC 的最坏情况。而相比之下，CMS 收集器的测试结果对比之下就要差很多，3 种 Java 堆容量下都出现了 100%时间进行 GC 的情况，

在吞吐量测试中，测试数据取 3 次 SPECjbb 和 15 次 telco 的平均结果。在 SPECjbb 的应用下，各种配置下的 G1 收集器表现出了一致的行为，吞吐量看起来只与允许最大 GC 时间成正比关系，而在 telco 的应用中，不同配置对吞吐量的影响则显得很微弱。与 CMS 收集器的吞吐量对比可以看到，在 SPECjbb 测试中，在堆容量超过 768M 时，CMS 收集器有 5%~10%的优势，而在 telco 测试中 CMS 的优势则要小一些，只有 3%~4%左右。

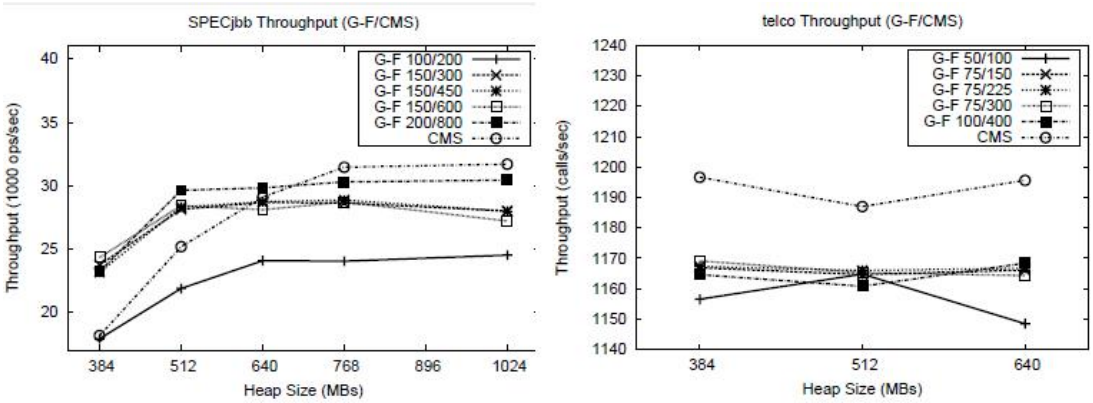


图 2：吞吐量测试结果

在更大规模的生产环境下，笔者引用一段在 StackOverfall.com 上看到的经验分享："我在一个真实的、较大规模的应用程序中使用过 G1：大约分配有 60~70GB 内存，存活对象大约在 20~50GB 之间。服务器运行 Linux 操作系统，JDK 版本为 6u22。G1 与 PS/PS Old 相比，最大的好处是停顿时间更加可控、可预测，如果我在 PS 中设置一个很低的最大允许 GC 时间，譬如期望 50 毫秒内完成 GC（-XX:MaxGCPauseMillis=50），但在 65GB 的 Java 堆下有可能得到的直接结果是一次长达 30 秒至 2 分钟的漫长的 Stop-The-World 过程；而 G1 与 CMS 相比，它们都立足于低停顿时间，CMS 仍然是我现在的选择，但是随着 Oracle 对 G1 的持续改进，我相信 G1 会是最终的胜利者。如果你现在采用的收集器没有出现问题，那就没有任何理由现在去选择 G1，如果你的应用追求低停顿，那 G1 现在已经可以作为一个可尝试的选择，如果你的应用追求吞吐量，那 G1 并不会为你带来什么特别的好处。"

在这节笔者引了两段别人的测试结果、经验后，对于 G1 给出一个自己的建议：直到现在为止还没有一款"最好的"收集器出现，更加没有"万能的"收集器，所以我们选择的只是对具体应用最合适的收集器。对于不同的硬件环境、不同的软件应用、不同的参数配置、不同的调优目标都会对调优时的收集器选择产生影响，选择适合的收集器，除了理论和别人的数据经验作为指导外，最终还是应当建立在自己应用的实际测试之上，别人的测试，大可抱着"至于你信不信，反正我自己没测之前是不信的"的态度。

参考资料

- Sun 实验室的论文《Garbage-First Garbage Collection》，作者为：David Detlefs、Christine Flood、Steve Heller、Tony Printezis
- 《The Garbage-First Garbage Collector》
- 《G1: Java's Garbage First Garbage Collector》

推荐文章 | Articles

了解云计算的漏洞

作者 Bernd Grobauer, Tobias Walloschek and Elmar Stöcker 译者 王恒涛

这篇文章最早发表在 Security & Privacy IEEE 杂志，由 InfoQ 联合 IEEE 计算机学会为您呈现。

关于云计算安全性的讨论往往失于对一般问题和云计算特定问题未加区分。为了让关于安全漏洞的讨论更加明了，根据风险要素与云计算的可靠定义，作者制定了一些指标。

每一天，每一条刚刚出炉的新闻、博客文章或其他的一些发行物都在提醒我们云计算的安全风险和威胁。多数情况下，安全问题都被认为是采用云计算的道路上最大的障碍。但这种关于云计算安全问题的论调反而让找到一个完善的方法来评估实际的安全后果变得更加困难，原因有如下两点：首先，在有关风险的这些讨论中，很大一部分都对一些基本的术语词汇——包括**风险**，**威胁**和**漏洞**——不加区分地交替使用，而不考虑各自实际的含义；其次，并不是每一个被提出来的问题，都是特别与云计算的背景对应。

为了更好地理解云计算在安全问题上所带来的新课题，我们必须分析云计算是如何影响了既有的安全问题。这里的一个关键因素是安全**漏洞**：云计算使得一些大家已经耳熟能详的漏洞变得更加突出，并且贡献了一些新成员。然而，在我们仔细分析特定于云计算语境的漏洞之前，我们必须先确定到底什么才算得上是一个真正的“漏洞”。

漏洞：概述

漏洞是一个突出的风险因素。ISO 27005 中把风险定义为“一种潜在的可能性，即某种特定的威胁利用一项或一组设施的漏洞来造成组织的破坏，”对其的度量应包括发生的概率以及事件的后果[1]。Open Group 的风险分类法提供了一个有用的危险因素总览（见图 1）。

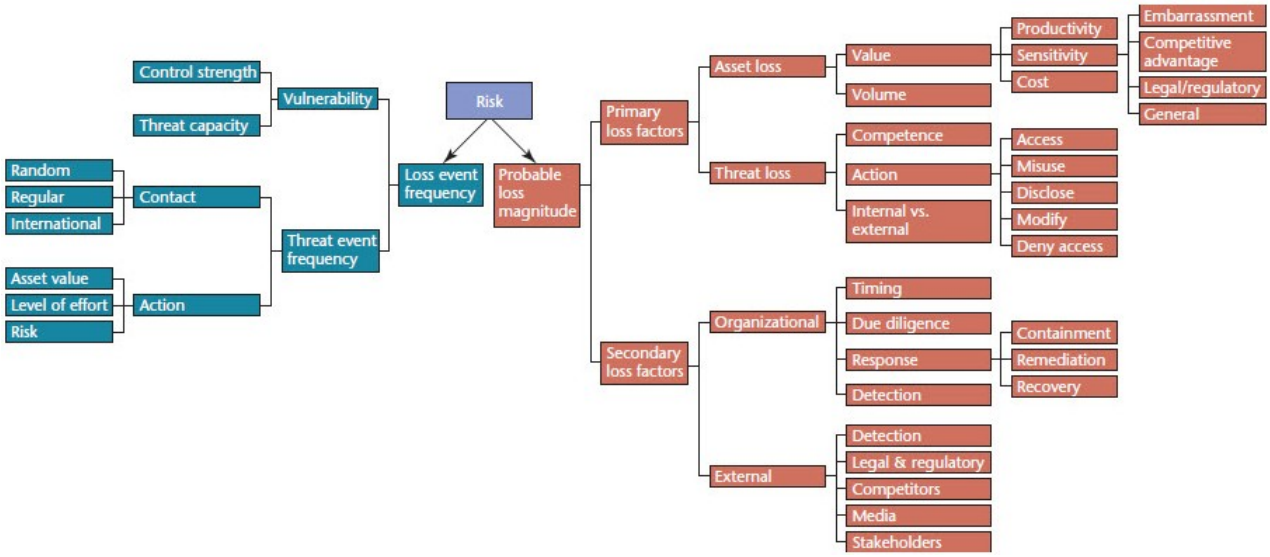


图 1.在 Open Group 的风险分类法所总结的会造成风险的因素。风险等于造成损害的事件的发生频率（左）和可能的损失幅度（右）的乘积。而漏洞则对造成损害的事件的发生频率具有影响。

Open Group 的分类法中使用了与 ISO 27005 一样的两条顶级风险因素：有害事件的发生概率（在这里称为造成损害的事件的发生频率）和其后果（在这里称为**可能的损失幅度**）。可能的损失幅度的子要素（如图 1 右侧所示）影响一个有害事件的最终代价。而（在图 1 左侧的）造成损害的事件的发生频率的子因素相对而言较为复杂。当一个威胁载体（例如说黑客）成功地利用了一个漏洞的时候就会发生造成损害的事件，这种情况发生的频率取决于两个因素：

- 威胁载体试图利用漏洞的频率。这个频率取决于载体的动机（他们从攻击中获得什么好处？他们需要付出多少努力？对于攻击者来说的风险是什么？），同时也取决于载体可以在多大程度上访问（“接触”）到攻击目标。
- 威胁载体的攻击能力与系统抵御攻击的坚固性之间的差距。

这第二个因素让我们接近得到一个有用的对于漏洞的定义。

定义漏洞

据 Open Group 的风险分类法，

> 漏洞就是设施无法抵御威胁载体行动的可能性。当威胁载体所施展的力量与目标抵抗能力之间存在差距的时候，就会产生漏洞。

因而，在描述漏洞时必须放在防止某种特定类型的攻击的背景下来考虑。这里可以举一个真实世界的例子，如果一辆汽车不能够保护它的驾驶者在正面被一辆以六十英里行驶的卡车撞击时免受伤害，这就是一个漏洞；这辆汽车的溃缩吸能区的强度相对于卡车的冲力来说太弱了。而相对于来自一辆自行车、甚或一辆中速行驶小型轿车的攻击而言，这款车的抵抗力则完全足够了。

我们可以还把计算机的漏洞描述成——也就是你可以用供应商所提供的补丁来弥补的安全相关错误——某种抵抗能力的弱化或消失。比方说，一个缓冲区溢出漏洞削弱了系统的防止任意代码执行的强度，而攻击者是否会利用此漏洞则完全取决于他们的能力。

漏洞和云计算的风险

现在，我们将从右侧的风险因素树开始，着手探讨云计算如何影响图 1 中的风险因素。

从云客户的角度来看，右侧所描述的可能的对未来损失的幅度的处理完全没有受到云计算的影响：事件的后果和最终的代价——就假定是来自于机密的泄露——完全一样，不论这种数据的泄露是发生在云平台还是传统的 IT 基础设施。而从云服务供应商的角度来看，事情就有一点不一样了：因为云计算系统以前是相互隔离在相同的基础设施上的，一个造成损害的事件可能带来相当大的影响。但是这个实际情况是很容易把控和纳入风险评估的：看起来不需要在概念上做出什么变化来适应云计算环境下的影响分析。

因此，我们必须在图 1 的左侧——造成损害的事件的发生频率——来找找是否有些什么变化。云计算可以改变一个有害事件的发生概率。正如我们后面还会指出的那样，云计算会导致漏洞要素发生相当大的变化。当然，移植到云基础设施可能会改变攻击者的访问级别和动机，以及工作量和风险——这是在以后的工作中必须考虑的一个事实。但是，对支持特定于云计算的风险评估而言，从考察特定于云计算的漏洞的严格本质入手似乎最有效益。

云计算

真的有所谓“特定于云计算”的漏洞吗？如果是这样的话，在云计算的本质中必然存在某些因素让一个漏洞成为特定于云计算的漏洞。

从本质上讲，云计算把已知的技术（如虚拟化）用巧妙的方法结合起来，“从流水线”上提供 IT 服务，产生了规模经济的效果。下面我们将更加详细地讨论什么是核心技术，以及这些技术在云计算的应用中有哪些关键的特性。

核心云计算技术

云计算非常依赖于现有的几样核心技术能力：

- **Web 应用程序和服务**：如果没有 Web 应用程序和 Web 服务技术，要发展软件即服务（SaaS）和平台即服务（PaaS）是无法想象的：SaaS 实例通常作为 Web 应用程序实施，而 PaaS 实例提供了 Web 应用和服务的开发和运行环境。在基础设施即服务（IaaS）实例中，管理员通常使用 Web 应用程序/服务技术来实施相关服务和 API，例如客户的管理访问。
- **虚拟化的 IaaS 实例**：在这些技术中虚拟化技巧都占据了核心地位。由于 PaaS 和 SaaS 服务往往建立在支持性 IaaS 基础设施之上，虚拟化的重要性也就延伸到了这些服务模型中。在未来，我们希望虚拟化可以从虚拟化服务器发展到可以直接用于 SaaS 服务的计算资源。
- **加密**：许多云计算安全的要求只有通过使用加密技术才能够得到解决。

随着云计算的发展，这份核心技术的清单很有可能扩大。

基本特征

在其基本的云特性的描述[2]中，美国国家标准与技术研究所（NIST）敏锐地指出了从流水线上提供 IT 服务意味着什么：

- **按需自助服务**:用户可以使用——比方说一个门户网站和管理界面,来订购和管理服务，而不再需要与来自服务供应商的真实的人打交道。服务及其相关资源的上线准备和下线准备都在供应商端自动解决。
- **无处不在的网络接入**：云服务是通过网络（通常是互联网）访问的，应用标准的机制和协议。
- **资源池**：用于提供云服务的计算资源是通过使用一个在所有服务用户间共享的同质基础设施实现的。
- **敏捷的弹性**：资源可以快速并且具备弹性的扩充或缩减。
- **可度量的服务**：资源/服务的使用率随时进行测算，支持资源使用率优化、使用率用户报告以及用多少收多少的商业模式。

NIST 的云计算定义框架，包括其关键特性列表，现在已经演化成为事实上的定义云计算的标准。

特定于云计算的漏洞

根据我们前面介绍的云计算的抽象视图，我们现在可以下手定义什么构成了特定于云的漏洞。我们可以说一个漏洞是特定于云的，条件是其：

- 对于某个核心云计算技术来说是不可分割的或是广泛存在的，
- 根本诱因是 NIST 的关键特性列表中的一项，
- 其发生可以归咎于云计算创新让尝试和测试安全控制难以、甚至根本无法实施，或者
- 在成功的最新云计算服务中很常见。

我们现在来研究这四项指标。

核心技术漏洞

云计算的核心技术——Web 应用程序和服务、虚拟化和加密——存在一些漏洞，有些是固有于技术本身，而另一些则是普遍存在于该技术的流行实现方式中。这里举这些漏洞的三个例子，包括虚拟机逃逸、会话控制和劫持以及不安全或过时的加密。

首先，虚拟化的本质就决定了存在攻击者从一个虚拟环境中成功逃脱的可能性。因此，我们必须把这个漏洞归类于固有于虚拟化、与云计算高度相关的那一类漏洞。

其次，Web 应用技术必须克服这样一个问题，即从设计的初衷来说，HTTP 协议是无状态协议，而 Web 应用程序则需要一些会话状态的概念。有许多技术能够实现会话处理，而许多会话处理的实现都容易遭受会话控制和劫持，这一点随便一个具有丰富 Web 应用安全经验的安全专业人士都可以作证。会话控制/劫持漏洞是 Web 应用技术所固有的呢，抑或“只是”常见于许多当前实现？这一点是值得商榷的。不过，在任何情况下，这样的漏洞当然和云计算有关系。

最后，密码分析学的进步可以使任何加密机制或算法变得不再安全，因为总是有新奇的破解方法被找出来。而更为普遍的情况是，加密算法实现被发现具有关键的缺陷，可以让原本的强加密退化成弱加密（有时甚至相当于完全不加密）。在没有加密来保护云里的数据保密性和完整性的情况下，无法想象云计算能够获得广泛的应用，因而可以说不安全或过时的加密漏洞与云计算有着非常密切的关系。

关键的云特性的漏洞

正如我们前面提到的，NIST 描述了五个关键的云计算特性：按需自助服务，无处不在的网络接入，资源池，敏捷的弹性和可度量的服务。

下面是一些源自上述一种或以上特性的漏洞的例子：

- **未经授权的管理界面访问**：按需自助服务云计算特性需要一个管理界面，可以向云服务的用户开放访问。这样，未经授权的管理界面访问对于云计算系统来说就算得上是一个具有特别相关性的漏洞，可能发生未经授权的访问的概率要远远高于传统的系统，在那些系统中管理功能只有少数管理员能够访问。
- **互联网协议漏洞**：无处不在的网络接入云计算特性意味着云服务是通过使用标准协议的网络获得访问。在大多数情况下，这个网络即互联网，必须被看作是不可信的。这样一来，互联网协议漏洞也就和云计算发生了关系，像是导致中间人攻击的漏洞。
- **数据恢复漏洞**：关于资源池和弹性的云特性意味着分配给一个用户的资源将有可能在稍后的时间被重新分配到不同的用户。从而，对于内存或存储资源来说，有可能恢复出前面用户写入的数据。
- **逃避计量和计费**：可度量的服务云特性意味着，任何云服务都在某一个适合服务类型的抽象层次（如存储，处理能力以及活跃帐户）上具备计量能力。计量数据被用来优化服务交付以及计费。有关漏洞包括操纵计量和计费数据，以及逃避计费。

接下来我们可以利用 NIST 的完善的云计算定义来思考云计算问题。

已知安全控制的缺陷

如果云计算创新直接导致在实施控制上的困难，标准安全控制漏洞就应该认为是特定于云计算的。这种漏洞也被称为**控制的挑战** 。

在这里，我们剖析这种控制的挑战的三个例子。第一个挑战是虚拟网络提供的基于网络的控制不足。由于云服务自身的性质的限制，对 IaaS 的网络基础设施的管理访问和量身定制网络基础设施的能力通常是有限的，因此无法应用标准控制，如基于 IP 网络的分区。此外，标准的技术，如基于网络的漏洞扫描通常被 IaaS 提供商所禁止，原因之一是无法把友好的扫描从攻击者的活动区别开来。最后，像虚拟化这样的技术意味着网络流量同时产生在真实和虚拟的网络中，比如，当托管在同样服务器上的两个虚拟机环境（VMEs）通信的时候。

这些问题构成了一个控制的挑战，因为基于尝试和测试的网络级安全控制在一个给定的云环境中可能无法正常工作。

第二个挑战是在差劲的密钥管理程序。正如在一项最近的欧洲网络和信息安全局的研究所表明的一样[3]，云计算基础设施需要管理和存储许多不同种类的密钥。由于虚拟机不会有一个固定的硬件基础设施，并且基于云的内容往往是地理上分散的，更难以对云计算基础设施的密钥实施标准控制——如硬件安全模块（HSM）存储。

最后，安全指标没有根据云基础设施进行调整。目前，还没有这样一种标准化的特定于云计算的安全指标，让云客户可以使用它来监视云资源的安全状态。在这样的安全指标被制定和实施之前，对安全评估、审计和问责的控制会更加困难和昂贵，甚至可能是不可能开展的。

最新的云计算实例中常见的漏洞

虽然云计算相对年轻，但在市场上已经存在无数的云计算实例。因此，我们为前述的三项特定于云计算的漏洞指标补充第四个实证指标：如果一个漏洞在最新的云计算实例中很常见，就必须认为它是特定于云计算的。这些漏洞的例子包括注入漏洞和薄弱的身份验证方案。

针对注入漏洞的攻击是指操纵服务或应用输入来以开发者意想之外的方式来解释或执行输入的片段。注入漏洞的例子包括：

- SQL 注入：指在输入包含 SQL 代码，诱发错误的数据库后端执行；
- 命令注入：在输入中包含通过操作系统被错误执行的命令；
- 跨站点脚本：在输入中包含 JavaScript 代码，在受害者的浏览器中执行。

此外，许多广泛使用的身份验证机制是很薄弱的。例如，用于验证的用户名和密码是薄弱的，原因如下：

- 不安全的用户行为（选择弱密码，重复使用的密码，等等）；
- 单因素认证机制固有的局限性。

身份验证机制的实现也可能有弱点并导致被攻击，例如凭证拦截和重播。当前最新的云计算服务中的大多数都采用了用户名和密码的身份验证机制。

架构组件和漏洞

云服务模式通常分为 SaaS，PaaS 和 IaaS，在给定云基础设施时，每一种模式都会影响暴露

出来的漏洞。增加更多的结构性到服务模式堆栈会有所帮助：图 2 中给出了一种云计算参考架构，明确了最重要的安全相关的云计算组件，而且为了分析安全问题提供了一个云计算的抽象概述。

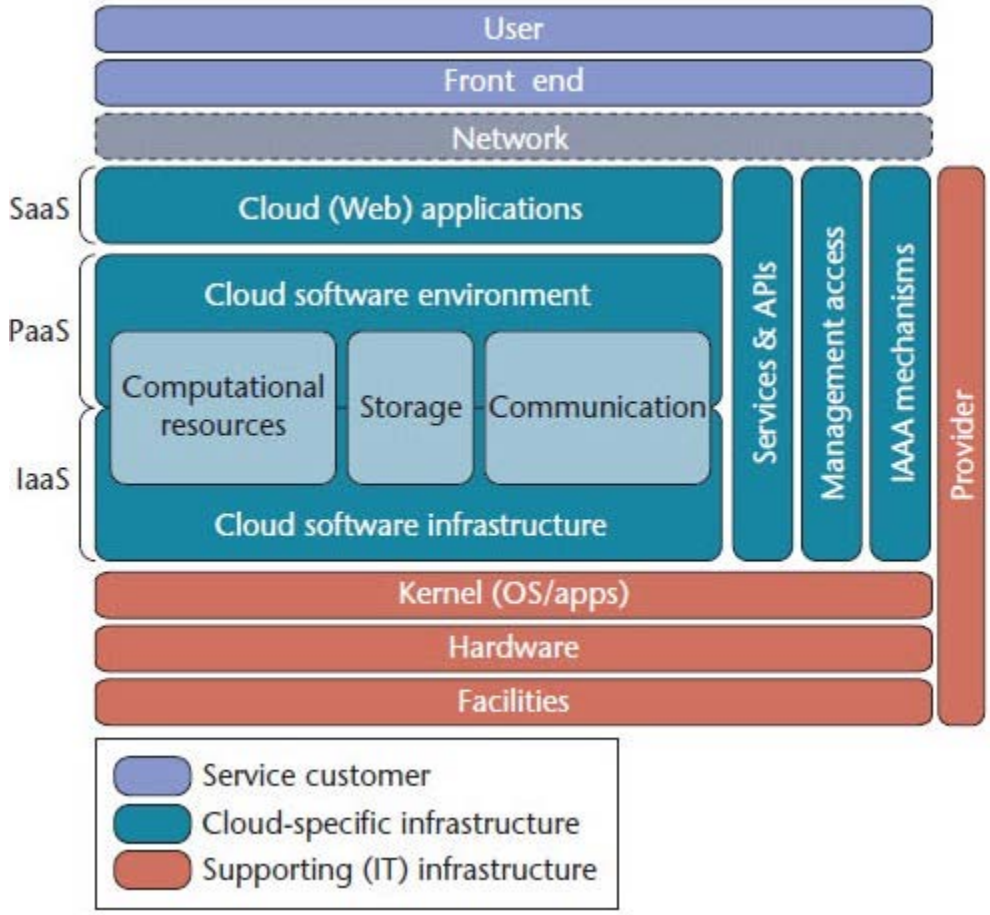


图 2.云计算参考架构。我们建立了特定于云计算的漏洞到这个参考架构的组件之间的映射关系，这样让我们大致了解有哪些漏洞可能会与一个给定的云计算服务相关。

这个参考架构是基于在洛杉矶的加州大学和 IBM 开展的工作[4]之上的。它继承了分层的方法，因为层可以囊括一个或多个服务组件。在这里，我们使用“服务”在广义上的概念，既可能包括物质（如建筑、电力和硬件），也可能包括非物质（如运行时环境）。对于云计算的软件环境和云软件基础设施这两层，这个模型明确了层中的三个主要的服务组件——计算、存储以及通信。顶层服务可以由堆栈更下层来实现，事实上跳过中间层。例如，云端的 Web 应用程序可以用传统的方式实施和操作——也就是说，在一个标准的操作系统上运行，而无需使用专用的云计算软件的基础设施和环境组件。从这样的分层和组合性可以知道，在模型的任何层之间，都可能从场地内服务或功能供应转换到服务和功能外包。

除了原有的模式，我们还确定了若干层服务的相关支持功能，并将它们添加到模型中，垂直

覆盖几个水平层。

我们的云计算参考架构有三个主要部分：

- **支持（IT）基础设施**：这些对于任何 IT 服务、云计算或其他方式来说都是常见的设施和服务。我们之所以把它们包括在架构之中，是因为我们希望提供一个全景——要完整描述 IT 安全性就必须也照顾到云服务的非特定于云计算的组成部分。
- **特定于云计算的基础设施**：这些组件构成云服务的核心，特定于云计算的漏洞和相应的控制通常映射到这些组成部分。
- **云服务的消费者**：同样，我们之所以把云服务的客户包括到架构中是因为它对于全面的安全性讨论很重要。

此外，我们明确地把将云服务消费者和云计算基础设施分开的网络表示出来，因为云资源是通过（通常是不可信的）网络进行访问的这一个事实是云计算的主要特点之一。

使用云计算参考架构的结构，我们现在可以挨个讨论架构的组成部分，并给出每个组成部分特定于云计算的漏洞的例子。

云计算软件基础设施和环境

云计算软件基础设施层把作为服务提供给上层的基本 IT 资源抽象成为一个抽象层次，这些资源包括：计算资源（通常是 VME——虚拟机环境）、存储以及（网络）通讯。这些服务可单独使用，典型场景是用于存储服务中，但他们也经常捆绑在一起，这时服务器就会附有网络连接，（通常）还提供对存储的访问能力。这种捆绑在一起的服务通常简称为 IaaS，无论是否带有存储能力。

云计算的软件环境层在应用平台层面提供服务：

- 一个开发和运行时环境，支持用一种或多种语言所开发的服务和应用程序；
- 存储服务（是数据库接口，而不是文件共享）；
- 通信基础设施，如微软的 Azure 服务总线。

在基础设施和环境层存在的漏洞通常都与这两层提供的三种资源类型的某一种密切相关。然而，跨租户访问漏洞就与所有三种类型的资源都有关系。我们前面描述的虚拟机逃逸漏洞的就是一个典型的例子。我们用它作为一个核心虚拟化技术所固有的漏洞的例子，但也可以认

为它根本上是来自于资源池的本质特征：当使用资源池时，跨资源的未经授权的访问将成为一个问题。因此对 PaaS 而言，就算用于隔离不同的租户（和租户服务）的技术未必基于虚拟化（不过这其实是一个越来越流行的趋势）的时候，跨租户访问漏洞也还是会产生影响。同样，云存储容易导致交叉租户存储访问，云通信——以虚拟网络的形式——容易导致交叉租户网络访问。

计算资源

一组高度重要的计算资源漏洞是与如何处理虚拟机映像有关：提供几乎相同的服务器映像的唯一可行的办法——从而为虚拟服务器提供按需服务——是通过克隆模板镜像来实现。

有漏洞的虚拟机模板镜像会导致许多操作系统或应用程序上的漏洞传播到更多系统。攻击者可能伪装成服务客户租用一个虚拟服务器以获得管理员权限，这样就能够分析系统的构成方式、补丁版本，甚至是具体代码，从而获得在攻击其他客户的镜像时有用??的信息。另外一个有关的问题是，镜像有可能是来自于不可信来源，这种现象随着 IaaS 服务的虚拟镜像交易市场的出现更为突出。在这种情况下会存在一些风险，比如镜像可能被动过手脚，从而为攻击者提供后门。

虚拟机复制造成的数据泄漏也是一个类似的漏洞，原因同样在于为了提供随需服务而进行镜像克隆。克隆会导致虚拟机机密数据的泄漏问题：一个操作系统的某些元素——如主机密钥和加密字符串——本来应该完全属于一台主机，可是克隆却可能破坏这个关于隐私的隐含前提。这次同样是虚拟机镜像的新兴交易市场——比如说亚马逊 EC2——会引出一个相关的问题：用户可以把运行中的镜像转换成模板，并向其他用户来提供模板镜像。根据在创建模板前该镜像被使用的情况，此镜像有可能会包含用户并不愿意公开的内容。

这里还有一些控制上的问题，其中一些与应用加密有关。如果介于硬件与操作系统间的虚拟化抽象层在为虚拟机运行环境生成随机数的时候发生问题，这种薄弱的随机数生成机制可能会导致加密上的漏洞，因为要生成随机数往往需要硬件级别的信息源。虚拟化可能在利用这样的信息源上存在缺陷，或者说在同一台主机上容纳多个虚拟机运行环境可能会穷尽可用的信息源，导致薄弱的随机数生成机制。我们前面也提到了，这个抽象层还让先进的安全控制——像是硬件安全模块——的使用更加复杂，结果就可能是蹩脚的密钥管理程序。

存储

除了由于资源池和弹性的特性所引起的数据恢复上的漏洞，还有一个与介质擦除相关的控制

问题，在云计算环境中这往往是很难或不可能实现的。例如，在一个生命周期的末尾，如果一个磁盘仍被另一租客使用，就不能执行要销毁物理硬盘的数据销毁政策。

由于加密技术经常被用来克服与存储相关的漏洞，这一核心技术的漏洞——不安全或过时的加密和蹩脚的密钥管理——在云存储中有着特殊地位。

通信

云通信服务最突出的例子是为 IaaS 环境中的虚拟机运行环境提供网络支持。由于资源池，几个客户可能拥有同样的网络基础设施组件：共享网络基础设施组件的漏洞——像是 DNS 服务器或动态主机配置协议中的漏洞，或 IP 协议的漏洞——可能在 IaaS 的基础设施中引发基于网络的跨租户的攻击。

虚拟化网络还提出了一个控制上的问题：在云服务中，与上面其他问题一样，IaaS 的网络基础设施的管理权限访问和剪裁网络基础设施的可能性通常是受限的。此外，诸如虚拟化等技术的使用会导致网络流量不仅发生在“真正的”网络上，同时也发生在虚拟网络中（如在同一服务器上托管的两个虚拟机运行环境之间的通信），大多数虚拟网络的实现只提供了集成基于网络安全的有限可能性。总而言之，这形成了一个控制上的问题，即不充分的基于网络的控制，因为基于尝试与测试的网络级别安全控制可能在某些云计算环境下无法工作。

云计算 Web 应用程序

Web 应用使用浏览器技术来在前段执行用户交互。随着基于浏览器的技术如 JavaScript、Java、Flash 和 Silverlight 被更多地采用，Web 云计算应用可以被分为两种：

- 被维护在云端的应用程序组件；
- 在用户的浏览器内运行的浏览器组件。

今后，对于一些不需要频繁访问远端数据的情况，开发商将越来越多地使用一些技术，像是谷歌 Gears 等，来允许脱机使用 Web 应用的浏览器组件。我们已经描述了两个典型的 Web 应用程序技术漏洞：会话控制和劫持漏洞，以及注入漏洞。

其他特定于 Web 应用程序的漏洞与浏览器的前端组件有关。其中包括客户端的数据操作的漏洞，用户可利用其操纵应用程序组件发送到服务器的应用程序组件的数据，从而攻击 Web 应用。换句话说，服务器组件收到的输入是不是“预期”的客户端组件发送的输入，而是变更过的或是完全由用户生成的输入。此外，Web 应用程序还依赖于浏览器的机制以隔离嵌

入到应用程序（如广告、Mashup 组件等）的第三方内容。因此，浏览器的隔离漏洞可能允许第三方内容来操作的 Web 应用程序。

服务和 API

虽说云计算基础设施的所有层次看起来都明显提供服务，但要讨论云计算基础设施的安全性，还是值得特别地考虑所有基础设施的服务和应用的编程接口。大多数服务都可能是 Web 服务，从而也具有许多 Web 应用程序漏洞。事实上，Web 应用程序层可能完全由一个或多个 Web 服务实现，这样应用的 URL 只会把浏览器组件暴露给用户。因此，配套服务和 API 函数也具有 Web 应用程序层的许多漏洞。

管理访问

NIST 的云计算定义指出云服务的核心特征之一是：可以快速准备并发布，只需要最小的管理工作或服务提供商的配合。因此，每个云服务的一个共同点是管理接口，这会直接导致非法访问管理接口的漏洞。此外，因为经常使用 Web 应用或服务实现管理访问，它通常也会带来与 Web 应用层和服务/API 组件同样的漏洞。

标识、身份验证、授权和审计机制

所有的云服务（以及每个云服务的管理界面）都需要身份管理、认证、授权和审计（IAAA）的机制。在一定程度上，这些机制的某些部分可能被分离出来，作为一个独立的 IAAA 服务以供其他服务使用。足够的授权检查（这必然会用到身份验证和/或从 IAA 服务收到的授权信息）和云基础设施的审计这两个 IAAA 要素是每个服务实现的不可分割的部分。

IAAA 组件相关联的大部分漏洞必须被视为特定于云计算，因为它们在当前最新的云计算实例中很常见。前面我们已经举了薄弱的用户认证机制的例子，其他的例子还包括：

- **由于帐户锁定而拒绝服务**：一个经常使用的安全控制——尤其是对于用户名和密码验证方式来说——就是锁定那些在很短时间内连续发生失败验证请求的帐号。攻击者可以利用这样的方法来发动针对某个用户的 DoS 攻击。
- **薄弱的凭证重置机制**：当云计算供应商自己来管理用户凭证，而不是使用联邦式身份验证，他们必须提供一个机制来重置凭证，以防凭证被忘记或丢失的情况。在过去，密码恢复机制已被证明非常薄弱。

- **不足或错误的授权检查**：最新的 Web 应用程序和服务的云计算实例往往容易受困于不足或错误的授权检查，会暴露未经授权的信息或行为给用户。比方说缺少授权检查就是 URL 猜测攻击的根源。在这种攻击中，用户可以修改 URL 来显示其他用户帐户的信息。
- **粗粒度的授权控制**：云服务的管理界面特别倾向于提供过粗粒度的授权控制模型。因此，像职责分离这样的标准安全措施得不到实施，因为没办法只提供给用户那些仅够他们展开工作的权限。
- **日志和监控不足的可能性**：目前，还没有一个标准或机制来让云计算的客户来记录和监测云计算资源内的设施。这就产生了一个尖锐的问题，即日志文件记录所有租户事件，无法容易地把单个租户的信息剪裁出来。此外，监测能力的不足往往阻碍了供应商进行安全监控。直到我们制定出可用的关于记录和监测的标准并实施成为工具之前，是很难——甚至说是不可能——实现需要记录和监测的安全控制的。

以云服务提供商的经验而言，在所有这些 IAAA 漏洞中，目前验证问题是主要的漏洞，因为它让用户放在云服务里的数据蒙受风险[5]。

提供商

所有云计算组件的漏洞，或者更确切地说，无法让用户像控制自己的基础设施一样控制云计算基础设施，通常都让提供商感到担心。控制方面的课题有：安全审计不足的可能，以及认证方式和安全度量在云计算中没有得到采用这样一个事实。此外，审计、认证与持续安全检测这样的标准安全控制没有被有效地实施。

云计算还处于不断发展的阶段，随着该领域的成熟，更多的特定于云的漏洞肯定还会出现，而旧的威胁也会减弱。从 Open Group 的风险分类，加上我们在此识别出的特定于云计算的漏洞的四项指标，我们得到了精确的关于漏洞的定义，提高了准确度和清晰度，而这正是目前为止的对云计算安全性的讨论中所缺乏的。

一些本来成功的安全控制在云计算的背景下变得无效，这些情况在安全控制的问题中往往很突出。因此，这些问题对于进一步的云计算安全研究具有特殊的意义。事实上，目前有许多努力——如安全度量和认证方式的开发，以及全功能虚拟网络组件发展——都试图直接解决这些控制方面的问题，方法是在云计算中使用基于尝试和测试的控制。

作者简介

Bernd Grobauer 是在信息安全方面的高级顾问，领导着西门子计算机紧急响应小组（CERT）

的研究活动，包括安全事件检测和处理、恶意软件防御和云计算安全。Grobauer 拥有丹麦奥胡斯大学的计算机科学博士学位。他是国际信息完整性机构的资格咨询委员会成员。要想联络他请发邮件至 bernd.grobauer@siemens.com 。

Tobias Walloschek 是西门子 IT 解决方案和服务公司的高级管理顾问。他的研究兴趣是云计算安全性和业务导入战略。Walloschek 拥有德国因戈尔施塔特应用科学大学的工商管理学士学位。他是信息系统安全认证专家。要想联络他请发邮件至 tobias.walloschek@siemens.com。

Elmar St?cker 是西门子 IT 解决方案和服务公司的一位经理，在那里他负责投资组合策略和专业服务组合监管。他还领导云计算的安全性和 PaaS 活动。Elmar St?cker 拥有德国亚琛工业大学的计算机科学硕士学位。要想联络他请发邮件至 elmar.stoecker@siemens.com。

IEEE Security & Privacy 的首要目标是激发和追踪在安全性、隐私和可靠性上的进步，并把这些进步以一种对广泛的跨界专业人群——从学界到业界——有所裨益的形式反映出来。

[1] ISO/IEC 27005:2007 Information Technology—Security Techniques—Information Security Risk Management, Int'l Org. Standardization, 2007.

[2] P. Mell and T. Grance, "Effectively and Securely Using the Cloud Computing Paradigm (v0.25)," presentation, US Nat'l Inst. Standards and Technology, 2009;

[3] European Network and Information Security Agency (ENISA), Cloud Computing: Benefits, Risks and Recom-mendations for Information Security, Nov. 2009;

[4] L. Youseff, M. Butrico, and D. Da Silva, "Towards a Unified Ontology of Cloud Computing," Proc. Grid Computing Environments Workshop (GCE), IEEE Press, 2008; doi: 10.1109/GCE.2008.4738443.

[5] E. Grosse, "Security at Scale," invited talk, ACM Cloud Security Workshop (CCSW), 2010;

原文链接：http://www.infoq.com/cn/articles/ieee-cloud-computing-vulnerabilities

相关内容：

- 一种开放的可互操作的云、构建企业私有云架构
- 新的云在地平线上出现——Oracle Public Cloud
- 设计一种云级别身份认证结构、私有云综述

围绕InfoQ中文站关注的领域设计话题

——线下定期举行的技术讨论沙龙

形式： 1个主题，1+个嘉宾，N个参会人员。

参会人员： 有决策权的中高端技术人员。

人数： 限制人数，保证每个人的发言权利。

频率： 每月一次。

城市： 以北京为主，现已在北京、杭州、深圳、西安举办过多场。

InfoQ Enterprise Software Development Community QClub

InfoQ官方微博：http://weibo.com/infoqchina

InfoQ豆瓣小组：http://www.douban.com/group/infoq/

讨论组：groups.google.com/group/infoqchina

编辑电邮：editors@cn.infoq.com

新品推荐 | New Products

Windows 8 将替换 Win32 API

作者 Jonathan Allen **译者** 詹涛

Windows 8 新引入了称为 WinRT 的核心 API。支持使用 C/C++、.NET 或 JavaScript 来开发 Metro 风格的应用。这些应用自 动获得硬件加速和高级电源管理的功能。现有的 Silverlight 和 WPF 应用程序可以以最小的代价移植到新的“Native XAML”库。

原文链接：http://www.infoq.com/cn/news/2011/09/WinRT

Twitter Storm：开源实时 Hadoop

作者 Bienvenido David III **译者** 丁雪丰

Twitter 将 Storm 正式开源了，这是一个分布式的、容错的实时计算系统，它被托管在 GitHub 上，遵循 Eclipse Public License 1.0。Storm 是由 BackType 开发的实时处理系统，BackType 现在已在 Twitter 麾下。GitHub 上的最新版本是 Storm 0.5.2，基本是用 Clojure 写的。

原文链接：http://www.infoq.com/cn/news/2011/09/twitter-storm-real-time-hadoop

Eclipse 3.7.1 开始支持 Java 7

作者 Alex Blewitt **译者** 贾国清

Eclipse 基金会近日发布了 Eclipse Indigo 3.7.1 版本，此为 Indigo（3.7）的一次小版本发布。通常，这种小版本的发布不值一提，与以往不同的是，此版本的 Eclipse JDT（Java Development Tools）中增加了对 Java 7 的支持。

原文链接：http://www.infoq.com/cn/news/2011/09/eclipse-indigo-371

Sync Framework 打破了平台之间的屏障

作者 Jenni Konrad, Roopesh Shenoy **译者** 侯伯薇

Sync Framework Toolkit 构建在 Sync Framework 2.1 之上，使用 OData 在所有平台或客户端——包括 Windows Phone 7、Silverlight、Windows Mobile、iPhone、iPad、黑莓、Android 设备以及使用 HTML5 的浏览器——之间实现同步。

原文链接：http://www.infoq.com/cn/news/2011/09/Sync-Framework-Toolkit

Sencha Touch 2：令人期待的新特性

作者 Dionysios G. Synodinos **译者** 郑柯

Sencha 宣布：Sencha Touch2 的 beta 版本将于 10 月发布，并且带有重要新特性，包括原生打包和性能改进。

原文链接：http://www.infoq.com/cn/news/2011/09/sencha-touch-2-preview

Spring Social 给 Java 带来 Social Connectivity

作者 Alex Blewitt **译者** 金明

最近发布的 SpringSocial 1.0 给人们带来了一个通过一致的 API 连接社交服务的标准方法。该初始版本支持对 Twitter、Facebook、GitHub 与 Trippit 等社交服务的连接，并针对添加其他服务提供了 SPI。

原文链接：http://www.infoq.com/cn/news/2011/09/spring-social

F# 3.0 -- LINQ + 类型提供程序 = 富信息编程

作者 Roopesh Shenoy 译者 李永伦

微软最近宣布了 F# 3.0 的开发者预览版——新特性包括通过查询表达式支持 LINQ，以及 类型提供程序系统和一组内置的提供程序，使得对各种数据源的编程变得简洁。

原文链接：http://www.infoq.com/cn/news/2011/09/Fsharp-3.0

Amazon 的全新浏览器 Silk 使用分离式架构

作者 Abel Avram 译者 郑柯

Amazon 开发了基于 WebKit 的浏览器——Silk，它使用 SPDY 维持与 AWS 托管服务的单一连接，在 AWS 上，web 页面可以提前加载和准备，得以推送到设备上。效果就是：浏览速度更快，设备功耗更低，更安全。

原文链接：http://www.infoq.com/cn/news/2011/09/Amazon-Silk

MaintainJ 3.2 业已发布，多项功能得到增强

作者 Dionysios G. Synodinos 译者 侯伯薇

MaintainJ 是一种反向工程工具，它能够为 Java 代码库生成运行时序列图和类图，最近它发布了 3.2 版本，在其中多项功能得到了增强。MaintainJ 基于 Eclipse 构建，因此能够在所有基于 Eclipse 平台构建的 IDE 上运行。

原文链接：http://www.infoq.com/cn/news/2011/09/maintainj-3.2

推荐编辑 | 翻译团队编辑 高翌翔

大家好，我是高翌翔，很荣幸能与各位《架构师》读者认识。

【我与 InfoQ】

我从 2005 年至今一直从事基于.NET 平台的 Web 应用程序开发，虽始终在参与或主管中小型项目的开发，但至今距离成为一名合格的软件架构师的目标还有非常大的差距。由于 InfoQ 始终坚持发布企业软件开发领域的最新技术资讯、以及个人兴趣和工作需要，因此最近两年经常访问 InfoQ 主站和中文站，并且在机缘巧合之下于今年 8 月正式加入了 InfoQ 中文站翻译团队。

【我眼中的 InfoQ 团队】

在加入 InfoQ 中文站大家庭后，得到了众多编辑同仁的热心帮助和鼓励。各位编辑同仁都是各自领域内的高手，能与他们一起共事感到非常荣幸，而且从他们身上看到了六种值得学习的非技术特质，即冷静、诚实、求知、反思、协作、分享：

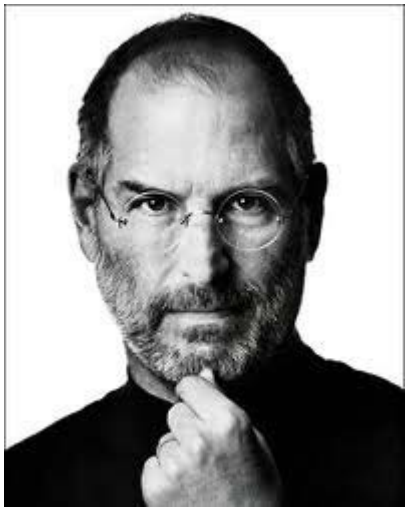
- 冷静是缜密思考、恰当决策的前提，大家在讨论中总能保持冷静而客观的态度，各种问题自然迎刃而解。
- 诚实地对事、对人、对己，知错就改，大家在与 InfoQ 共同进步。
- 求知体现在大家对翻译质量的严格要求，绝不放过任何一处可能的错误。
- 反思体现在团队会定期召开线上讨论和线下聚会，收集、整理大家的意见和建议，进而不断改进。
- 协作是我感受最深的一点，团队会为新编辑指派资深编辑作为翻译教练，进行逐字逐句地悉心帮助指导，这里要特别感谢我的翻译教练伯薇。此外，当遇到疑难的段落翻译时，只要在编辑组里群发邮件，大家都会积极伸出援手、各抒己见。
- 分享是一种气氛，当置身于 InfoQ 中文站大家庭中时，就会被这种气氛深深感染，乐于分享你的所感所悟！

以上是加入 InfoQ 中文站翻译团队以来的一些个人感悟，欢迎大家通过邮件（yixianggao@126.com）与我交流。

最后发挥一下主人翁精神，您有意给 InfoQ 中文站投稿或是加入翻译团队么？请邮件至 editors@cn.infoq.com 。

封面人物

史蒂夫·乔布斯



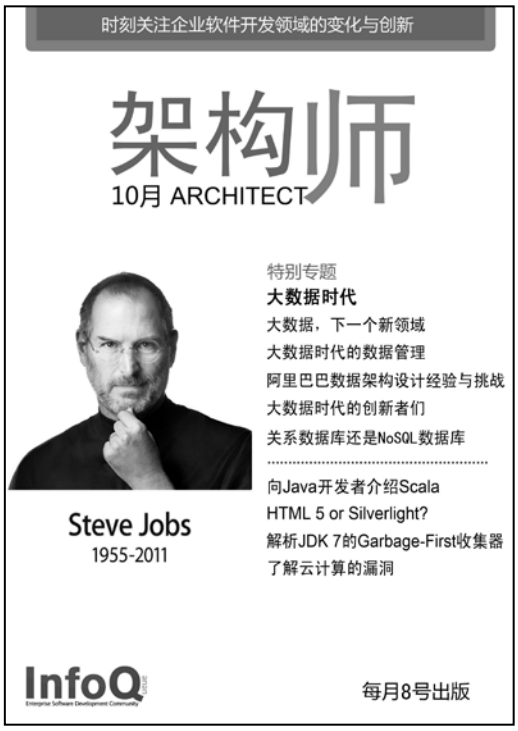
史蒂夫·乔布斯（Steve Jobs）是苹果公司的前任首席运行官兼创办人之一，同时也是前皮克斯动画工作室（Pixar Animation Studios，香港译作彼思动画制作室）的董事长及行政总裁（Pixar 已在 2006 年被迪士尼收购）。乔布斯还是迪士尼公司的董事会成员和最大个人股东。乔布斯被认为是计算机业界与娱乐业界的标志性人物，同时人们也把他视作麦金塔计算机、ipad、iPod、iTunes Store、iPhone 等知名数字产品的缔造者。

1976 年乔布斯和 26 岁的沃兹尼艾克成立苹果电脑公司。1985 年获得里根总统授予的国家级技术勋章；1997 年成为时代周刊封面人物；2007 年被《财富》杂志评为年度最伟大商人。2009 年被财富杂志评选为这十年美国最佳 CEO，同年当选时代周刊年度风云人物。

2011 年 8 月 24 日，苹果 CEO 史蒂夫·乔布斯(Steve Jobs)正式向苹果董事会提交辞职申请。他还在辞职信中建议由首席营运长 Tim Cook 接替他的职位。乔布斯在辞职信中表示，自己无法继续担任 CEO，不过自己愿意担任公司董事长、董事或普通职员。苹果公司股票暂停盘后交易。乔布斯在信中并没有指明辞职原因，但他一直都在与胰腺癌作斗争。

北京时间 2011 年 10 月 6 日，苹果董事会宣布前 CEO 乔布斯于当地时间 10 月 5 日逝世，终年 56 岁。

1kg.org
多背一公斤
爱自然 | 更爱孩子



架构师 10 月刊

每月 8 日出版

本期主编：李明

总编辑：霍泰稳

美术编辑：胡伟红

读者反馈：editors@cn.infoq.com

投稿：editors@cn.infoq.com

InfoQ 中文站新浪微博：
http://t.sina.com.cn/infoqchina

商务合作：sales@cn.infoq.com 13581658359

本期主编：李明，InfoQ 中文站原创团队编辑

李明（nasi），毕业于东北大学，曾供职于百度网页搜索部，从事分布式网络爬虫及其国际化的研发工作。目前在某通信公司任系统架构师，进行高性能大规模分布式系统的设计与开发。擅长搜索引擎技术，关注开源社区发展，注重敏捷开发实践。同时他还是一位吉他手，兴趣广泛，乐于分享。

InfoQ neue
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