

云生态

Cloud Ecosystem

专刊

第3期
Vo.03

Docker专栏

InfoQ
CLOUD

生态圈新闻

观点&趋势

技术热点

对话大咖



打造中国最优质的云生态媒体

扫我，码上开启新世界



Geekbang, 有温度的技术社区。Geek是一种精神, 也是一种态度; Geekbang是一个圈子, 也是一种习惯。在这儿, 你要么是Geek, 要么走在成为Geek的路上。

ArchSummit

International Architect Summit

全球架构师峰会 2015

[深圳站]
2015年7月17-18日

中国·深圳·大梅沙京基海湾大酒店



李高峰

华为商城首席架构师

专题演讲:
垂直电商技术的艰辛之路



余庆

易到用车首席架构师

专题演讲:
移动互联网下基于地理位置变化
场景实时搜索引擎



张跃

批改网CEO

专题演讲:
批改网: 基于语言大数据的英语作文
自动批改服务系统建设和运营



陈明

微信高级工程师、朋友圈负责人

专题演讲:
微信朋友圈技术之道



方林

华大基因深圳研发部副院长

专题演讲:
生命科学中的大数据



徐梦云

饿了么数据技术部总监

专题演讲:
数据仓库治理



邓澍军

猿题库研究部总监

专题演讲:
猿题库: 大数据时代的在线教育



曹彬彬

中信证券首席数据应用专家

专题演讲:
证券行业大数据应用探讨

联系方式:

购票热线: 010-89880682

会务咨询: arch@cn.infoq.com

在线咨询(QQ): 2332883546

在线交流(QQ群): 425975960

更多精彩内容, 敬请登陆: sz2015.archsummit.com

Brought by **InfoQ**

QCon上海站

2015年10月15-17日

上海光大会展中心
国际大酒店

www.qconshanghai.com

Brought by **InfoQ**

QCon

全球软件开发大会

International Software Development Conference



QCon是由InfoQ主办的全球顶级技术盛会，每年在伦敦、北京、东京、纽约、圣保罗、上海、旧金山，里约热内卢召开。自2007年3月份首次举办以来，已经有超万名高级技术人员参加过QCon大会。QCon内容源于实践并面向社区，演讲嘉宾依据热点话题，面向5年以上的技术团队负责人、架构师、工程总监、高级开发人员分享技术创新和最佳实践。



www.qconferences.com

云上的基础设施与运维

探讨企业云计算基础设施和架构思路以及高效安全运维解决方案

2015年7月17日(周五)



深圳

大梅沙京基海湾大酒店



报名请扫描二维码
名额有限, 报名从速

数据分析 与企业架构

聚焦IT企业架构设计与研发体系,
构建最优内部数据分析平台

2015年7月18日(周六)

云生态专刊

CloudEcoSystem

本刊由 **InfoQ** 中文站
制作出品

出品人：霍泰稳

总策划：崔康

本期编辑：魏星

流程编辑：丁晓昀

读者反馈/投稿

editors@cn.infoq.com

商务合作

sales@cn.infoq.com

目录

卷首语

热点回顾

雷击引发青云服务中断故障始末..... 5

Facebook 如何向十亿人推荐东西 11

Red Hat Linux 严重 Bug 将影响基于 Haswell 架构的服务器.. 14

对话大咖

方国伟：企业服务入云还有很长的路要走 17

技术热点

从 CIO 视角出发审视云环境下的安全议题 21

Java NIO 通信框架在电信领域的实践 26

Docker 专栏

京东 618：Docker 扛大旗，弹性伸缩成重点..... 46

Docker 的安全基准 50

观点&趋势

为私有云结庐而做“隆中对”（下） 53

分布式系统的特点以及设计理念..... 57

生态圈新闻

Microsoft Azure 位居 Nasuni 存储基准测试之首..... 62

Twitter 已经用 Heron 替换了 Storm 65

Windows 10：微软的十亿设备野心 66

卷首语

开放的云生态是创新的基础

每年 IaaS 领域的 OpenStack 和 PaaS 领域 Cloud Foundry 的全球峰会，都成为云计算领域发展的风向标。最近这两个领域的峰会今年五月分别在加拿大的 Vancouver 和硅谷 Santa Clara 先后举行。从参会人员的快速增长以及会上大量的成功案例的分享，我们惊喜地看到开源的云技术 OpenStack 和 Cloud Foundry 经过这些年的发展，已真正地开花结果了。虽然 OpenStack 和 Cloud Foundry 侧重于云计算技术的不同层面，但开放和生态则是这两个云服务技术发展和成熟的共同特点。

云计算技术是企业持续创新和保持竞争力的基础。在目前这样一个快速变化的时代，企业的竞争优势会瞬间即逝，只有持续创新才是解决之道。企业必须能够敏捷地适应变化，快速地开发出适应市场的新的技术和新的应用，以提供新的服务。传统企业需要转变来适应新的市场挑战，而开放的云技术正是这种转型的助力。正如美国最大的保险公司 AllState 的高级副总裁 Andy Zitney 在最近的 Cloud Foundry 全球峰会上强调的那样：“我们需要变化，因为任何人都想颠覆我们（保险行业）。没有变化就很难应付这种挑战。我们需要云计算平台来实现和驱动这种转变，使得产品投放市场加快，并通过持续改进产品来更好地应对市场变化。”

开放和生态是云计算技术和服务的发展基石。封闭在云计算产业已没有生存空间。云计算通过开放来赋予企业挑战旧体系的创新动力，通过生态则催生了专业化有效分工和互联网经济的形成。大数据、移动优先、互联网+、物联网（IOT）和工业 4.0 的发展，无一不是以云计算为技术支撑为基础，并很大程度上得益于云计算的开放性和完善的生态。

开放和生态有助于云计算技术的标准化和落地，特别是移动优先（Mobile First）和云原生（Cloud Native）的需求使得云计算平台成为帮助企业转型实现持续创新的平台。移动优先是企业业务交付的需求，而云原生是技术架构的需求。云原生应用的关键点是没有状态约束。为了让应用有效地部署运行在云服务平台上，应用不能保持状态，或者满足互联网应用设计模式的“12 要素”。而云原生应用的关键是平台实现微服务（Microservices）架构。微服务的基本要求包括快捷服务提供机制、基本的监控、敏捷地应用开发和 DevOps 文化。而 Twitter、LinkedIn 和 Netflix 云原生微服务架构成功实践包括：实现平台部署和运维微服务，提供自我服务的部署，可以持续交付应用软件，

以及可以避免故障。特别是通过业务营销、产品开发和运维团队的配合，实现业务规模化和快捷交付，使得基于云原生应用的技术和商业价值的持续交付相匹配。云原生应用架构可以真正实现应用持续部署和交付，进而实现敏捷地持续创新。

开放和生态也产生云计算技术的融合。开放云技术的融合可以打消人为的技术界限，更好地满足和整合用户多种不同业务需求。目前的趋势是基础平台的需求从工作负载的整合在向应用的投放转变，更好地和用户交互，从而更有效地驱动企业业务的增长。企业 IT 更多地从基础设施转变为体现在应用上。一般认为 IaaS 负责基础设施及其自动化，PaaS 管理工作负载并快捷的将其投放市场，进而直接影响业务。相对来说 PaaS 是较具颠覆性的云服务模式，PaaS 是一个本质上不同的应用设计、部署、运行方式。但这种模式的采用会给企业带来前所未有的价值。特别是作为“最后一英里”的云技术，PaaS 已不断地成为企业云计算战略的一个关键部分。同时，IaaS 和 PaaS 等技术的相互渗透和融合可以更为灵活地为用户动态提供多元计算资源抽象：包括 VMs 和容器化资源；基础资源和应用资源。同时这种融合可以简化应用生命周期管理，提供高效的应用和服务开发、部署和运维。云计算的开放性所产生的融合也模糊技术和业务的界限，或者技术不断地成为企业业务的重要部分。

让我们拥抱开放，一起打造一个健康的云服务生态系统。

——鲁为民

热点回顾



云生态专刊

雷击引发青云服务中断故障始末

作者 魏星

2015 年 6 月 6 日下午，因服务商“[睿江科技](#)”机房遭遇雷暴天气引发电力故障，[青云](#)广东 1 区全部硬件设备意外关机重启，造成青云官网及控制台短时无法访问、部署于 GD1 的用户业务暂时不可用。与此同时，另一家云服务商 [LeanCloud](#) 也发生了长达 4 小时的服务中断情况。

事后青云通过[官方微博](#)发布了事故报告，除了审视自身需要改进的地方，对 IDC 防雷能力也提出了疑问。[LeanCloud](#) 通过[官方微博](#)和[官方报告](#)也对自身事故原因进行了说明。

笔者对云服务故障上一次的记忆还停留在《[公有云故障案例分析——Microsoft Azure 的飞来人祸](#)》和微软《[Azure 的官方博客的说明](#)》。针对此次事故，一位 IDC 的从业老兵认为：

云计算出问题一般都是两方面，一个是数据中心稳定否，一个是云平台自身系统稳定否。在选择第三方数据中心方面，一般云服务商会在全中国选定几个区域，每个区域选两家以上的 IDC，中间用裸光打通。当然，这是比较理想的状态，其成本较高。

那么，能否用自建数据中心的方式来解决稳定性的问题呢？该老兵的回答是：

目前第三方云服务商还没有实力自建 IDC。例如一个 T3 级别 1200 个机柜的数据中心，投入超过 2 亿并且需要 1 年以上的建设周期，一个柜子的运营成本每年要至少 10 万元人民币。因此，国内第三方云服务商目前以租赁 IDC 的服务为主。而选择一个可靠的数据中心尤为重要。

据了解，IDC 建设有严格的[国标规范](#)，其对不可抗力的要求远大于普通基础设施。以地震为例，IDC 的要求是抗里氏 8 级地震。除了国标，运营商对机房建设还有电信研究院把关。第三方机房的建设更多地会参考国际标准。因此在如何选择 IDC 服务方面，[首都在线](#)工程部经理刘铮介绍说：

IDC 机房的基础设施从地理位置、建筑承重、电力保障、空调效能、消防、安防等各个方面都有要求。选择 IDC 需要考虑机房位置、资源、建筑、空间、动力系统、制冷、安保、运维等方面综合考量。

国内 IDC 机房是这几年才发展起来的，大致分为三种。

第一种，是运营商机房。这类机房会严格遵守 IDC 的建设标准。电信运营有近百年的历史，积累了大量的实践经验。

第二种，是运营商合作机房，这类是指运营商和社会资金合作，按照运营商的标准设计、选型构建的机房。除了所有权外，基本等同运营商机房。

第三种，是中立机房，这类机房良莠不齐。有很好的机房，例如世纪互联 M5 机房，是完全按照先进 IDC 机房规范进行设计的。也有很差的。

目前看来，在中国市场由于运营商对网络的垄断地位，个人觉得采购成熟的运营商机房是个不错的选择。而自建机房会明显增加云厂商大量的基础设施建设成本，同时也会增加采购运营商带宽成本。

但现有国内的数据中心都是上一代数据中心，为服务器托管所设计的，空间密度低，能提供的单机柜功率低，配套空调系统效能低。就我所知大部分国内机房单机柜支撑 10~20A 机柜也就是 2200w 至 4400w，这对现在的云计算系统而言，密度太低了。单机柜密度低会造成密度降低，增加云计算大量的光纤耗材成本，再设计云计算资源池时就会不得不迁就机房现有条件，分散布放计算节点，增大了 tor 与 eor 之间距离；存储方面也造成了分散，影响了效率，使得云计算资源设计难以模块化。而非模块化设计、非标准化设计会增加运维难度，增加故障隐患。国外的机房大多单机柜可以提供 48A、64A 甚至于更高。

至于青云此次事故中提到的 UPS 问题，刘铮解释说：

UPS 系统非常重要，是现有数据中心最容易出现故障的一环。目前国内大多采用的都还是电池方案；国际上除了电池外还有飞轮方案等 UPS 代替方案。理论上要求 N+1 甚至 N+N 的容量冗余、定期的电力巡检及维护、UPS 定期代载供电、电池的维护保养及更换。

机房动力系统是最关键的，对 IDC 选址要求至少两路供电，要有完备的柴油发电机组，现有油量储备至少支撑数小时供电，并有不间断的柴油供给。

来自 InfoQ 高效运维群的讨论主要观点如下：

- ❑ 机房整体崩溃的容灾切换方案和演练有必要加强。通讯基站都会做防雷，一个 IDC 不能因雷击就影响用户，雷击只是一个诱因。
- ❑ 现阶段国内云服务商还是以开发为主，并没有对网络、IDC、运维看的特别重要。
- ❑ 网络与 IDC 已成为运维的重要话题。运营商机房限制很多，多线融合是第三方 IDC 的优势，但网络是其瓶颈。
- ❑ 由于 IDC 互联成本太高，可以根据需求做 DPN + DRaaS 对部分或整个 DC 做实时异地容灾。但互联互通将成为趋势，也是市场发展的必然。
- ❑ 异地容灾的成本也不容小觑。目前看来，冷备有问题、双活有门槛，云服务商需要看自己的业务需要，看 SLA。中断服务影响最大，所以稳定和高可用是架构设计第一要素。
- ❑ 做运维就是把更多的不可控变成可控。没做到高可用、实时异地容灾，设计上的单点故障也就难以解决。
- ❑ 现在的云产品只能谈可用性，还谈不上易用性。

随即，我们采访了 [UCloud](#) 联合创始人兼 CTO 莫显峰，老莫从选择 IDC 租赁的标准方面给出的建议是：

选择 IDC 的标准非常多，包括物业使用权、租约、变电站、油罐、柴发、空调、承重、链路拓扑/带宽等等，但这也不足以确保 100% 的可靠性。美国标准 TIA-942 《数据中心的通信基础设施标准》，主要根据数据中心基础设施的可用性、稳定性和安全性分为四个等级：T1，可用性为 99.67%；T2，可用性 99.749%；T3，可用性 99.982%；T4，可用性 99.995%。年平均故障时间也从 0.4 小时到 28.8 小时不等，这意味着每年都可能存在各种原因的不可用。

对于青云本次 IDC 故障的警示以及第三方云服务商应该怎么规避此类问题，莫显峰认为应该从一下几个方面加强防范：

1. 选择合适的机房。
2. 建立同城多中心：同城多 IDC 之间用光纤直连，将客户的业务部署在多个机房，避免在某个机房出故障的时候，影响用户的业务可用性。
3. 协助客户做好规划：完全规避 IDC 的问题是不可能的，因此理论上的 7*24 小时的业务必须在软件架构、应用部署上进行优化，例如：
 - a. 针对外网线路故障采用多线路灾备，故障发生时实现自动切换；

b.针对机房故障包括电力故障和空调故障等，采用跨 IDC 数据库灾备，接入层冗余的模式进行容灾；

c.针对城域网故障采用异地有损容灾、IDC 双活或多活等方案。

4.定期巡检、演习：提前发现并消除隐患，大多数市电突然中断是无法避免的，但是 UPS 和柴发确保能工作很重要。

至于大家讨论的各云厂商之间互联互通的理想，UCloud 认为网络互联是可以先行的第一步。“只要各家的 VPC 支持光纤/专线互通，即可实现灾备的能力。目前 UCloud 是支持光纤接入公有云的。”莫显峰如是说。

综上所述，首先事故本身是一个非常小概率的事件，人力所及的是尽量降低其发生的概率，而无法绝对避免。IDC 行业有着严格的建设标准，判断一个机房的可用性要在较长的时间范围内看其可用时间的长短。发生事故并不等同于机房不靠谱或者云不靠谱。当然，事故发生后要彻底排查发生的原因，采取整改措施。

因此，从设备和管理两个层面去降低事故概率是十分必要的，云服务厂商也要不断提升云平台的灾难恢复速度。最后，我们对青云联合创始人兼 CEO 黄允松（Richard）进行了简短的采访，他对此事故的回复如下。

InfoQ：以前 IDC 遭遇雷暴天气引发故障的案例也很多，在你看来能造成大面积影响的主要原因是什么？或者说，青云在 IDC 布局方面是否因受限于供应商而存在一些隐患？

Richard：机房遭遇雷击后，承载 QingCloud 设备所在区的两组 UPS 输出均出现了 2 秒钟的瞬时波动，从而导致了机柜出现瞬时断电再加电。UPS 厂家给出的书面判断是，雷击楼体后，对 UPS 主机造成干扰，UPS 浪涌保护器未生效，引起 UPS 并机线通信故障，电压回流造成电压高报警，逆变器过载关机，短时短路。

目前青云在全国共运行着 8 个区域，其中公有云的 4 个区是青云自营租用的，我们在每个省份都是选择当地最好 IDC 供应商，比如这次出问题的机房隶属广东睿江科技，这家 IDC 运营商在华南是绝对的领先者。这次事故机房是当地电信的枢纽机房，T3 等级，即便如此，也遇见了此次雷击引发电力闪断的巨大灾难。我们非常清楚云平台的服务能力一方面体现在软件技术实力，另一方面也取决于物理层面的稳定和可靠。我们已经下定决心自行投资运营数据中心，以将物理层面事故概率降到最低。

需要强调的一点是，无论是云计算还是传统 IT，都依赖于数据中心基础设施的稳定运行，而作为云服务商，我们的系统承载着众多用户的业务，因此要在数据中心的可靠性方面再下大力气。

InfoQ：从发现故障告警到完全恢复服务青云大概用了不到 3 个小时，用户的受影响程度以及反馈如何？IDC 方面有没有新的报告出来？

Richard：其实总时间是 2 小时 31 分钟，其中大约有 1 小时 40 分钟花在了物理硬件加电、全云系统的自检过程（检查所有设备、所有用户资源的状态、数据完整性检验等），用于恢复全部用户云资源的时间很短，因为此处皆自动化了。但是前面的硬件加电、检查、云系统自检这些过程还依赖于人、属于半手动的过程，这个需要进一步的开发，使之皆能自动化进行。

因为是全体断电，所以对用户的影响是大的，这也是数据中心行业能有的最严重的故障了，所有在广东一区部署的应用皆受到了影响。那些新型应用架构、吻合云模式的应用（我们称之为 Cloud Native App）随着云平台恢复而立刻恢复，那些偏传统的应用通常需要用户登录进去启动一些服务、或做些操作。总体上来说，因为没有出现数据丢失，用户的反馈大多比较冷静和理性。特别值得一提的是，有一些很技术专业的用户甚至将这次事故视作一个机会来检视自己应用架构的合理与改良，真的很酷。极客邦转发的一篇就是例子。

IDC 在 6 月 6 日提供了官方报告，我们已经发出来了，后来也提供了一份纯技术分析文章，其实是对前份报告的电力细节描述，我们也已经通过邮件和社交平台发给了用户。

InfoQ：经过本次事故，青云会在哪些方面进行改进，能简单说说你们的改进措施吗？

Richard：首先我们真的非常抱歉，数据中心出了这种非常小概率的大故障，让那么多用户受到严重影响。我们已经决定要自己运营数据中心（第一个力争年内在北京启用），欢迎有意投资云计算产业的企业机构与我们洽谈。另外，我们的自营网络建设一期工程已进入尾声，会在 Q3 之前完成北京各区之间的光纤直连，实现同城环网，这将极大程度提高云平台的容灾能力；随后会逐步实现其他各区环网。

另外从平台技术角度，我们计划加入全面的硬件自动化，这将使得我们 100% 杜绝人力工作慢的问题，使得灾难恢复工作更加快速；还有对于平台自检过程需要优化，我们将在北京的机房设置 evil monkey 测试场景，将全机房断电列为日常测试项目，确保此类灾难能在 30 分钟完成全部恢复。

Facebook 如何向十亿人推荐东西

作者 张天雷

为了保证用户体验和使用效果，推荐系统中的机器学习算法一般都是针对完整的数据集进行的。然而，随着推荐系统输入数据量的飞速增长，传统的集中式机器学习算法越来越难以满足应用需求。因此，分布式机器学习算法被提出用来大规模数据集的分析。作为全球排名第一的社交网站，Facebook 就需要利用分布式推荐系统来帮助用户找到他们可能感兴趣的页面、组、事件或者游戏等。近日，[Facebook 就在其官网公布了其推荐系统的原理、性能及使用情况](#)。

目前，Facebook 中推荐系统所要面对的数据集包含了约 1000 亿个评分、超过 10 亿的用户以及数百万的物品。相比于著名的 [Netflix Prize](#)，Facebook 的数据规模已经超过了它两个数据级。如何在大数据规模情况下仍然保持良好性能已经成为世界级的难题。为此，Facebook 设计了一个全新的推荐系统。幸运的是，Facebook 团队之前已经在使用一个分布式迭代和图像处理平台——[Apache Giraph](#)。因其能够很好的支持大规模数据，Giraph 就成为了 Facebook 推荐系统的基础平台。

在工作原理方面，Facebook 推荐系统采用的是流行的协同过滤（Collaborative filtering, CF）技术。CF 技术的基本思路就是根据相同人群所关注事物的评分来预测某个人对该事物的评分或喜爱程度。从数学角度而言，该问题就是根据用户-物品的评分矩阵中已知的值来预测未知的值。其求解过程通常采用矩阵分解（[Matrix Factorization](#), MF）方法。MF 方法把用户评分矩阵表达为用户矩阵和物品的乘积，用这些矩阵相乘的结果 R' 来拟合原来的评分矩阵 R ，使得二者尽量接近。如果把 R 和 R' 之间的距离作为优化目标，那么矩阵分解就变成了求最小值问题。

对大规模数据而言，求解过程将会十分耗时。为了降低时间和空间复杂度，一些从随机特征向量开始的迭代式算法被提出。这些迭代式算法渐渐收敛，可以在合理的时间内找到一个最优解。随机梯度下降（[Stochastic Gradient Descent](#), SGD）算法就是其中之一，其已经成功的用于多个问题的求解。SGD 基本思路是以随机方式遍历训练集中的数据，并给出每个已知评分的预测评分值。用户和物品特征向量的调整就沿着评分误差越来越小的方向迭代进行，直到误差到达设计要求。因此，SGD 方法可以不需要遍历所有的样本即可完成特征向量的求解。交替最小二乘法（[Alternating Least Square](#), ALS）是另外一

个迭代算法。其基本思路为交替固定用户特征向量和物品特征向量的值，不断的寻找局部最优解直到满足求解条件。

为了利用上述算法解决 Facebook 推荐系统的问题，原本 Giraph 中的标准方法就需要进行改变。之前，Giraph 的标准方法是把用户和物品都当作为图中的顶点、已知的评分当作边。那么，SGD 或 ALS 的迭代过程就是遍历图中所有的边，发送用户和物品的特征向量并进行局部更新。该方法存在若干重大问题。首先，迭代过程会带来巨大的网络通信负载。由于迭代过程需要遍历所有的边，一次迭代所发送的数据量就为边与特征向量个数的乘积。假设评分数为 1000 亿、特征向量为 100 对，每次迭代的通信数据量就为 80TB。其次，物品流行程度的不同会导致图中节点度的分布不均匀。该问题可能会导致内存不够或者引起处理瓶颈。假设一个物品有 1000 亿个评分、特征向量同样为 100 对，该物品对应的一个点在一次迭代中就需要接收 80GB 的数据。最后，Giraph 中并没有完全按照公式中的要求实现 SGD 算法。真正实现中，每个点都是利用迭代开始时实际收到的特征向量进行工作，而并非全局最新的特征向量。

综合以上可以看出，Giraph 中最大的问题就在于每次迭代中都需要把更新信息发送到每一个顶点。为了解决这个问题，Facebook 发明了一种利用 work-to-work 信息传递的高效、便捷方法。该方法把原有的图划分为了由若干 work 构成的一个圆。每个 worker 都包含了一个物品集合和若干用户。在每一步，相邻的 worker 沿顺时针方法把包含物品更新的信息发送到下游的 worker。这样，每一步都只处理了各个 worker 内部的评分，而经过与 worker 个数相同的步骤后，所有的评分也全部都被处理。该方法实现了通信量与评分数无关，可以明显减少图中数据的通信量。而且，标准方法中节点度分布不均匀的问题也因为物品不再用顶点来表示而不复存在。为了进一步提高算法性能，Facebook 把 SGD 和 ALS 两个算法进行了揉合，提出了旋转混合式求解方法。

接下来，Facebook 在运行实际的 A/B 测试之间对推荐系统的性能进行了测量。首先，通过输入一直的训练集，推荐系统对算法的参数进行微调来提高预测精度。然后，系统针对测试集给出评分并与已知的结果进行比较。Facebook 团队从物品平均评分、前 1/10/100 物品的评分精度、所有测试物品的平均精度等来评估推荐系统。此外，均方根误差（Root Mean Squared Error，RMSE）也被用来记录单个误差所带来的影响。

此外，即使是采用了分布式计算方法，Facebook 仍然不可能检查每一个用户/物品对的评分。团队需要寻找更快的方法来获得每个用户排名前 K 的推荐物品，然后再利用推荐系统计算用户对其的评分。其中一种可能的解决方案是采用 ball tree 数据结构来存储物品向量。all tree 结构可以实现搜索过程 10-100 倍的加速，使得物品推荐工作能够在合理时

间内完成。另外一个能够近似解决问题的方法是根据物品特征向量对物品进行分类。这样，寻找推荐评分就划分为寻找最推荐的物品群和在物品群中再提取评分最高的物品两个过程。该方法在一定程度上会降低推荐系统的可信度，却能够加速计算过程。

最后，Facebook 给出了一些实验的结果。在 2014 年 7 月，[Databricks 公布了在 Spark 上实现 ALS 的性能结果](#)。Facebook 针对 [Amazon 的数据集](#)，基于 [Spark MLlib](#) 进行标准实验，与自己的旋转混合式方法的结果进行了比较。实验结果表明，Facebook 的系统比标准系统要快 10 倍左右。而且，前者可以轻松处理超过 1000 亿个评分。

目前，该方法已经用了 Facebook 的多个应用中，包括页面或者组的推荐等。为了能够减小系统负担，Facebook 只是把度超过 100 的页面和组考虑为候选对象。而且，在初始迭代中，Facebook 推荐系统把用户喜欢的页面/加入的组以及用户不喜欢或者拒绝加入的组都作为输入。此外，Facebook 还利用基于 ALS 的算法，从用户获得间接的反馈。未来，Facebook 会继续对推荐系统进行改进，包括利用社交图和用户连接改善推荐集合、自动化参数调整以及尝试比较好的划分机器等。

Red Hat Linux 严重 Bug

将影响基于 Haswell 架构的服务器

作者 Jeff Martin 译者 魏星

最近，Azul Systems 公司的 CTO 与联合创始人 Gil Tene 在 Google Groups 报告了一个十分重要，但鲜为人知的 Linux 内核补丁，采用英特尔 Haswell 架构的 Linux 系统用户和管理员尤其应该关注该问题。特别是基于 Red Hat 发行版的用户（包括 CentOS 6.6 以及 Scientific Linux 6.6），应该立即更新这个补丁。即便是运行在虚拟机中的 Linux，如果这个虚拟机是在流行的云平台上（如 Azure、Amazon 等），它也可能跑在 Haswell 机器上，打补丁应该是有好处的。

Tene 是对该缺陷的描述如下：

“这个内核漏洞的影响非常简单：在一些看似不可能的情况下，用户进程会死锁并被挂起。任何一个 futex 调用等待（即使被正确地唤醒）都有可能永远被阻止执行。就像 Java 里的 Thread.park() 可能会一直阻塞那样，等等。如果足够幸运，你会在 dmesg 日志中发现 soft lockup 消息；如果没那么幸运（比如跟我们这样），你将不得不花几个月的人工成本去排查代码中的问题，还有可能一无所获。”

Tene 继续解释了这个缺陷代码是如何执行的（最终可以归结到一个遗漏了 default 情况的 switch 块）。现在最大的问题是，尽管问题代码已经在 2014 年 1 月修复，但是在 2014 年 10 月左右，该缺陷又被移回了 Red Hat 6.6 家族系统中。其他系统包括 SLES、Ubuntu、Debian 等有可能也被影响了。

这些系统的修复情况现在并不一致，并且有可能被忽略。Red Hat 用户应该采用 RHEL 6.6.z 或更新的版本。Tene 还指出另一个关键点在于，对于要将哪些东西放入内核，不同的发行版会有不同的选择，这也导致问题的修复情况并不一致。

例如，对于 RHEL 7.1 而言，“其实上游的 3.10 内核是没有这个 bug 的，但 RHEL 7 的内核又不是纯粹的上游版本。不幸的是，RHEL 7.1（就像 RHEL 6.6 那样）在移植的时候把（基于 RHEL 7 版本）这个 bug 包含了进去……我认为其他发行版可能也是这么做的。”

对基于 RHEL 的发行版，Tene 提供了一个快速参考列表：

RHEL 5 (包括 CentOS 5 和 Scientific Linux 5): 所有版本 (包括 5.11 版) 都没有问题。

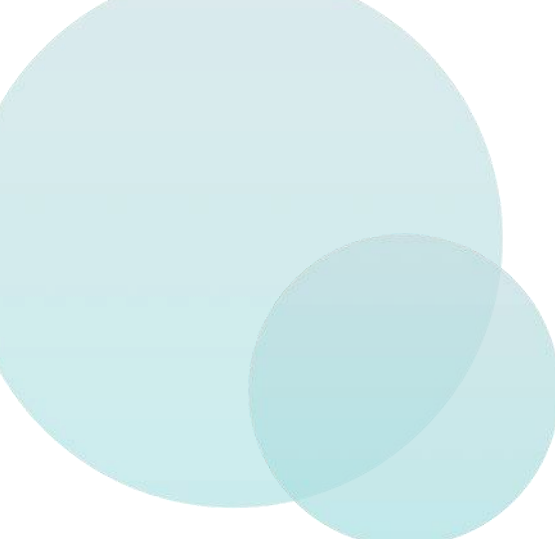
RHEL 6 (包括 CentOS 6 和 Scientific Linux 6): 从 6.0 ~ 6.5 版都没问题。 但 6.6 版存在缺陷，而 6.6.z 版本没有问题。

RHEL 7 (包括 CentOS 7 和 Scientific Linux 7): 7.1 是有缺陷的。并且截至 2015 年 5 月 13 日也没有一个 7.x 的修复。

尽管在 [Hacker News](#) 上对受影响系统的数量存在一些争议，但它提供了一些环境来检查你的系统是否需要修复。

对话大咖

云生态专刊



方国伟：企业服务入云 还有很长的路要走

作者 魏星

编者按：随着云计算的发展与落地，越来越多的企业开始制定自己的云计算发展战略。但对于如何构建企业云服务目前业界存在着不同的声音。本届 ArchSummit2015 深圳，InfoQ 特别邀请到平安科技基础架构部的副总工程师方国伟来讲述“大型企业的基础架构演变”的话题。

InfoQ：最早您在 IBM 从事 Java 开发，如今 Java 已经诞生 20 年了，您对这门编程语言有哪些感触可以分享一下吗？

方国伟：在过去十几年，Java 和 C 语言一直是最流行程编程语言排行榜的前两位，应该说这是目前应用最为广泛的两种语言。我是在 1998 年最早接触 Java 语言，那个时候我还在读研究生。当时我们在做一个基于 Web 的项目，那个时候做 Web 应用最主要的一种技术实现手段就是通过 CGI 方式。

那时对 Java 最主要的印象有三点：第一就是跨平台，第二个就是性能比较差，第三个就是图形化编程比较麻烦并且效果也不理想。跨平台是当时我们比较看重的一点，因为我们后台有一些小机，而 Java 的性能问题应该是后面持续几年都是被人诟病的一个话题。由于 Java 的主要应用场景逐渐变为服务器后台，所以图形化编程的问题也不算是个大问题。我记得 Java 真正的快速发展是在 2000 年后，那个时候 Servlet、JSP 等规范慢慢成熟，并且市面上开始出现一些中间件来帮助用户做应用平台，从而简化了后台应用的编程，同时结合 Java 的跨平台特性迅速在企业级应用中风靡。

应该说 Java 语言旺盛的生命力与其语言本身的与时俱进是分不开的，如果大家去看业界的各种编程语言，像 Java 语言这样不断有新的版本、不断吸收其他语言优点的语言真是不多。比如 Java 与 C#语言就在发展过程中相互借鉴和竞争，Java 也学习了不少面向函数的一些编程语言特性。

所以，小到一门语言、一个人，大到一个公司、一个国家都有“与时俱进”的要求。当然，不同的语言都有各自独特的地方，Java 语言解决了许多企业后台应用的需求，但目前的硬件发展需要更多的并行处理能力，像 Go 语言这样的新生代是专门为并发问题来设计的，后面我们也可以拭目以待 Java 后续是如何来更好的处理这些新的挑战。

InfoQ：以您从业这么多年的经历来看，传统企业在业务入云方面面临的主要问题有哪些？

方国伟：企业首先需要根据自身情况来决定自己的云计算战略是什么样的，是自己构建私有云平台为主，还是采用外部公有云服务，或者是采用混合云的方式。这几种方式在实际操作的过程中差别非常大。

其次是对自己的业务应用有明确的区分，不是所有现有应用都适合入云。

第三是要充分利用云平台的各种特性，应用程序本身的架构也需要做些相应的调整。如果不做调整，实际上很多时候的入云就会变成纯粹的虚拟化而已，而不能充分发挥云的优势。

InfoQ：随着云计算的普及，企业入云在数据迁移方面面临的问题有哪些？造成这些问题的原因是什么？

方国伟：我想这个问题应该是从公有云的角度来看企业入云的数据迁移问题的，因为在私有云的场景中这个问题一般不存在。

企业入云在数据迁移一般存在几个问题：

首先是数据安全和隐私的问题。云服务商能否充分保证企业用户的数据安全性。多个调查表明，大部分企业对公有云的第一担心就是数据安全问题。这个问题有传统观念上的原因，也有实际企业对云服务商数据管理和保护能力的担心，因为毕竟这些数据是保存在一个资源物理共享的平台上。

其次就是数据存放方式以及接口的问题。在云平台上数据的存储一般会提供多种方式，比如对象存储、NoSQL、文件系统等，企业在把数据进行迁移的时候有可能需要对应用进行改造。

第三就是数据的迁入和迁出问题。用户是数据真正的拥有者，所以希望能够自由的对数据进行控制，包括迁入和迁出。这需要云服务商在技术上进行迁出支持。

InfoQ：对于正在转型和即将转型到云服务的企业，在技术架构选型方面，您有哪些建议？

方国伟：这取决于这是家什么样的企业，如果是一家快速发展的小型企业，一个比较好的方式是直接采用公有云的服务。当然在选取云服务商的时候，要从技术、服务和品牌等多方面进行考察。

如果是选择自己构建一个云平台来支撑企业的业务发展，那么首先要明确云计算服务的层次，再决定从那一层入手，比如是 IaaS、PaaS 还是 SaaS 等。

另外，在技术路线上无非是自己开发、基于开源框架构建和基于商业产品构建几种选择。至于企业选择哪种路线主要是要结合企业自己团队的技术力量以及企业对云服务的时间要求或云服务发展路线要求来决定的。

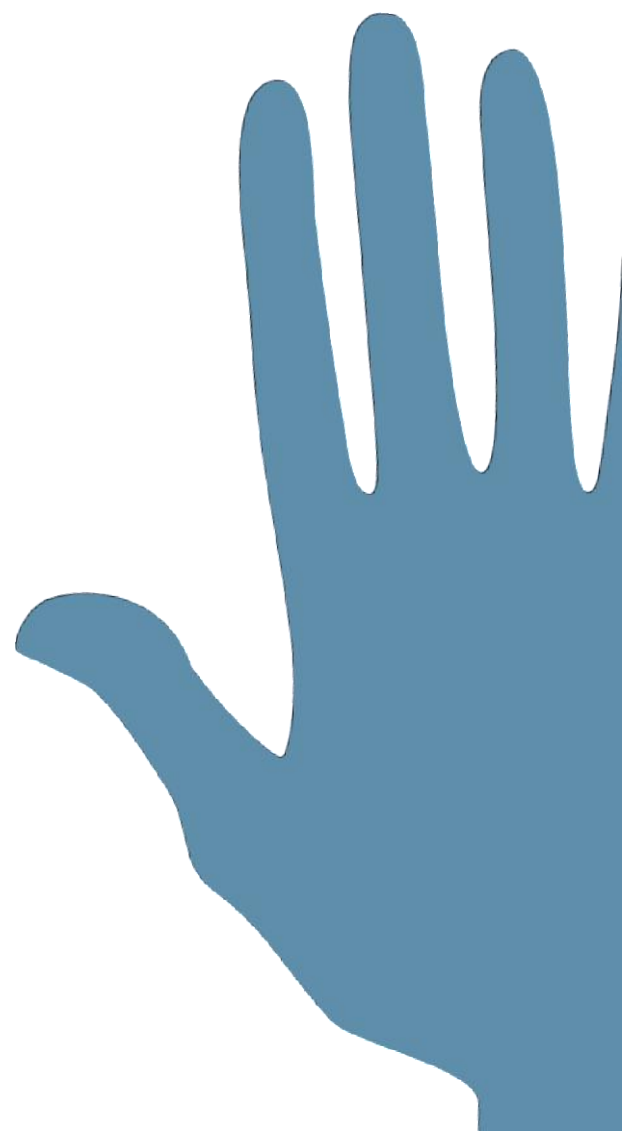
InfoQ：对于广大的传统客户，在向云计算转型过程中面临的问题，您有哪些建议，可以举例加以详细说明一下吗？

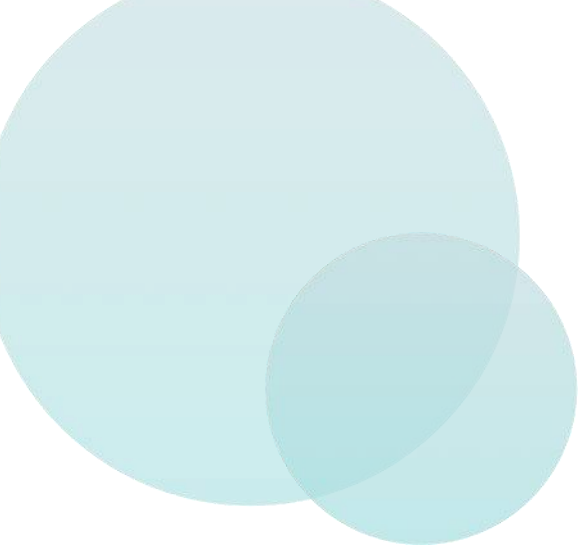
方国伟：传统企业在向云计算转型过程中会有许多挑战，因为毕竟云平台服务与传统的 IT 服务方式有挺大的差异。

传统企业在向云计算转型的过程中存在几个方面的挑战，这也是我将在 [ArchSummit 大会](#) 上跟大家探讨的主要内容，其中包括企业云平台的定位问题、传统应用的迁移问题、新应用的架构设计问题、服务流程的改变问题、云平台与非云平台团队的组织架构和协作问题等。这些问题如何处理都会影像企业向云计算的转型。

技术热点

云生态专刊





从 CIO 视角出发审视 云环境下的安全议题

作者 **Stephen Orban**

“很多人都在过度强调安全而非机遇，这就像是一个人更害怕生存而非死亡。”

——James F. Bymes

安全如今已经成为一个广泛的议题，并且渗透到了 IT 事务中的方方面面。纵观我在技术业界打拼的这么多个年头，我发现“安全”已经成了一个能够迅速扼杀任何创新型努力的词汇。在云计算发展的早期阶段，那些对云技术了解不深的人们总会就其安全性水平问东问西。相较于更为重要的、如何利用这项新技术帮助企业自身实现商业价值，他们往往首先把安全性作为技术演进的最大障碍。举例为说，2012 年我就开始利用 AWS 支持企业级项目，当时我所能依靠的只有自己的同事、团队以及他们所拥有的解决问题的经验。然而我们的研究结果大大坚定了自己的信心，事实上 AWS 为我们提供了一种远优于孤军奋战的系统安全保护途径。

作为一位前任 CIO 兼 AWS 客户，下面我将结合自身经历聊聊自己眼中的云环境安全性问题。

我知道，安全性目前是、未来也将继续是 AWS 所关注的核心议题之一。因为只有这样，AWS 才能为来自众多行业及政府机构的如此广泛且多样化的客户提供服务。我们拥有 PCI 系统、PII 数据、SOX 要求以及需要保护的知识产权信息。在了解到其它企业有能力在云环境下开发出足以顺利满足上述控制框架要求的、令人满意的解决方案之后，我们既受到启发、又对云技术的未来充满信心。

可以确定的是，AWS 方面投入了大量资源进行其服务平台的保护工作，相关资源总量远远超过我们用于运营支持的资源总和——而这还仅仅是在安全层面。与其它运行着自有数据中心的企业类似，我们也一直不断对成本、上市时间、质量以及安全性作出权衡。在这方面作出决策绝非易事，而且我们往往很难弄清自己是否做出了正确的选择。没能认真评估防火墙变更、线缆配置错误或者过于躁进地进行操作系统配置有可能影响我们的安

全水平。如果大家在企业环境中拥有长期工作经历，我相信各位绝对能明白这两者之间的联系。光是想想单纯因为我们自身因素而造成的安全风险，就已经足够把我们吓得魂不附体。而且如果把安全性当成企业运营中的头等大事，大家几乎没办法继续完成其它业务工作。

我知道，缩小受攻击面能够让我们将精力集中在自身的差异化特性当中。充分运用 AWS 所提供的安全机制能够帮助我们将一部分原本用于实施裸机保护的资源解放出来，转而用于保护应用程序。已知安全漏洞的逐步增加以及黑帽社区的日益壮大意味着我们必须进一步强化针对托管基础设施的应用程序安全保护力度。在 AWS 的帮助下，我们能够在增添新型资源的同时、又不至于让其它方面的配额过于捉襟见肘，这样一来我们对于安全保障工作的心态也更加平和。当然，这种共同分担的责任模式并不代表客户方面可以完全卸下包袱，但由此带来的助益却是不容否定的。至少就个人而言，我是乐于运用一切可资利用的帮助。

根据我掌握的情况，AWS 在全球安全发展前景方面的前瞻性要远远强于我们这单位一家企业的水平。我当然也希望能够充分享受由此带来的规模经济收益。事实上，我们也希望能够将 AWS 服务改进所带来的收益分享给自己的每一位客户。

在我看来，自动化机制能够大大降低出现人为错误的可能性，而且这一点在安全性以及应用程序开发领域也同样适用。我们希望尽可能多地将自动化方案引入那些需要反复进行的技术任务。根据我了解到的情况，AWS 高度依赖于自动化技术以实现规模化提升，同时降低人为错误的发生空间，并借此改进自己的安全性模型。能够拥有这样一位出色的合作伙伴，将鼓励并引导我们同样利用自动化手段实现收益增长。

CIO & LEADER 网站最近采访了 AWS CISO Stephen Schmidt，并就一系列安全议题展开探讨。在此次采访中，Stephen 谈到了 AWS 在安全性领域所采取的规模化、投资以及自动化机制等举措。我认为此次采访进一步增强了自己的信心，并坚定了我建议合作伙伴采用或者考虑采用 AWS 的决心。本次采访内容发布在 2014 年 12 月的 CIO & LEADER 实体杂志当中，以下转述内容全部得到了 CIO & LEADER 网站的授权。

CIO & LEADER：大多数企业信息安全负责人都没能成功顶住 DDoS 以及 APT 等下一代安全威胁带来的压力。您面临的此类安全威胁是否更大？您又是如何加以化解的？

Stephen Schmidt : 我们见证着一切在互联网上的发生。我希望分享一些有趣的数据来给大家提供更为直观的量化印象。在每 500 个 IP 地址当中，就有 1 个通过互联网被路由至 Amazon 网络当中，而且 700 个 IP 地址当中约有 1 个被映射至 EC2 实例处。大家可以把我们看作一套规模极为庞大的望远镜阵列，旨在发现一个极小的目标。这套设施允许我们识别出针对客户的安全威胁，并建立起自己的服务以帮助这些客户抵御此类威胁。举例来说，很多 APT 攻击者试图收集大量合法的用户名与密码内容。正是出于这个理由，我们不允许在网络中传输的用户名与密码中包含客户数据。我们还推出了多种智能验证令牌，这是因为利用物理设备进行验证更为安全、而且其更难被攻击者们所盗取。

CIO & LEADER : 第三方风险同样受到安全从业人员的高度关注。作为一名 CISO，您如何处理这些风险呢？

Schmidt : 为了最大程度降低此类风险，最重要的是确保我们的各合作第三方与我们遵循同样的安全标准。我们需要严格确保此类规则贯彻到位，并通过审计实现约束。举例来说，如果我们在某国建立了一套 CloudFront 主机代管设施，那么我们就要求该代管服务供应商提供与自身完全等同的安全水平。这部分内容在双方合作协议当中明确标定，而且我们会定期对其进行检查。

因此，我一个专项团队，其任务在于每年多次到访世界各地的每一座代管设施。我们采取突击检查的管理方式，以确保合作方能够根据既定规则完成自己的份内任务。我们的要求非常严格，而且在检查过程中要求对方员工全部撤离现场。举例来说，他们是否使用经过认证的固定件、螺钉或者螺母，这样我们才能保证自己无法从外部将其拧下。我们还会检查墙上的检修孔尺寸，确保其符合规定的规格要求，这样恶意人士就无法将手伸入实施破坏。我们也在检查中确保所有延伸出设施的线缆都包裹有保温导管。凭借着这一系列标准，我们才能通过检查来确保供应商满足我们的所有特殊要求。

CIO & LEADER : 看起来 AWS 确实拥有一套可靠的第三方风险应对策略。但您如何应对来自企业内部的安全威胁？

Schmidt : 应对内部威胁的最佳途径就是限制指向数据的人为访问。因此，我们在内部采取的措施之一在于主动降低有能力访问信息的人员数量。

尽管如此，我们的业界规模一直处于疯狂增长当中，我们每一周都需要削减能够访问客户信息的内部员工数量。我们能够以自动化方式实现这一调整工作。举例来说，如果某人需要多次——超过一次或两次——重复同样的工作，那么我们就会将其纳入自动化流程。我们会有针对性地建立起一套能够自动完成该项任务的工具。此类方案拥有两大优势。首先，工具基本不会犯错，它们不仅能够顺利完成任务、而且可以保证每次都同样顺利完成任务。相比之下，员工则可能带来多种意外因素，并因此造成问题。其次，工具能够显著提高可用性。因此，自动化对安全性及可用性的贡献确实值得肯定。

CIO & LEADER：那么，AWS 在 2015 年中设定了怎样的发展方向？贵公司将关注哪些技术成果及解决方案？

Schmidt：对 AWS 而言，我们下一步将高度关注的就是加密机制。加密机制将无处不在，也就是说加密技术将覆盖到每一个领域。我们的工程技术人员将高度重视的另一个领域在于向客户提供针对加密机制的控制能力，这样他们就能实现密钥内容管理。第三则是确保我们为客户提供一系列工具，帮助他们做出更理想的安全性决策。

过去供应商往往会告知客户，一旦有问题出现，他们将很快到场并加以修复。但在 AWS，我们选择了完全不同的解决思路。相比之下，AWS 给出的方案是：目前的这种状况还有提升的空间，而这里的这个按钮能够切实带来提升或者修正。总结来讲，我们为客户提供他们所需要的工具，此类工具成本极低甚至完全免费，这样他们完全可以自行解决问题。

CIO & LEADER：那么作为一位 CISO，您下一步打算在哪些领域投入资金？

Schmidt：我们在自动化技术领域投入了大量资源，因此我们打造的主要成果之一就是工具。而我们还将大量自动化要素引入常见的安全测试、渗透测试以及配置管理测试等日常事务当中，旨在确保一切以计划中的方式顺畅运转。在这些领域，我们每一年都会投入大量资金。

采取上述举措的原因有二，其一当然在于安全效益，其二则单纯是因为我们无法在不借助自动化机制的情况下运营如此庞大的设施体系。随着规模的不断提升，如果我无法快速推广自动化机制，我们根本不可能雇佣到那么多水平出色的安全工程师来保护全部 AWS 服务。我们投入了大量资源以实现自动化技术。

为企业创建安全系统是每一位 IT 高管人士的核心信条。为什么不使用目前市面上最出色的工具来实现同样的效果？大多数专业的安全从业者都会告诉大家，一切都取决于客户所具备的实际设备。出色的方案并不足以替代人才、实践以及艰苦的工作，但如果采用更强大的设备则能够带来良好的性能表现，并吸引客户加以使用。云服务虽然无法取代业务系统中的杰出人才、专业知识以及管理机制，但确实能够显著提高企业获得成功的可能性。

革命尚未成功，同志仍需努力。



Java NIO 通信框架 在电信领域的实践

作者 李林锋

1. 华为电信软件技术架构演进

1.1. 电信软件

从广义上看电信软件的范围非常广，细分实际可以分为两大类：系统软件和业务应用软件。

系统软件包括路由器底层的信令机软件、手机操作系统等，业务应用软件主要包括客户关系管理 CRM、网上营业厅、融合计费 OCS 和各类消息网关，例如短信网关、彩信网关等。

本文重点介绍电信业务应用软件的技术变迁历史，以及华为电信软件架构演进和 Java NIO 框架在技术变迁中起到的关键作用。

1.2. 华为电信软件的技术演进史

1.2.1. C 和 C++主导的第一代架构

在 2005 年之前，华为软件公司的核心系统主要以 C 和 C++ 进行开发，由于 C 和 C++ 开源框架非常少，加之那个时代开源社区并不成熟，大部分的系统都采用自研开发，包括协议栈、系统调度、数据访问层和日志。

大多数的软件都运行在服务端，对外提供高性能、低时延和高并发的系统调用，协议栈大多数都采用电信私有协议栈，对于部分有前台管理 Portal 的系统，往往基于原生的 HTML 或者 Struts 等 WEB 框架开发，通过 HTTP 协议与后端进行交互，它的逻辑架构图如下：

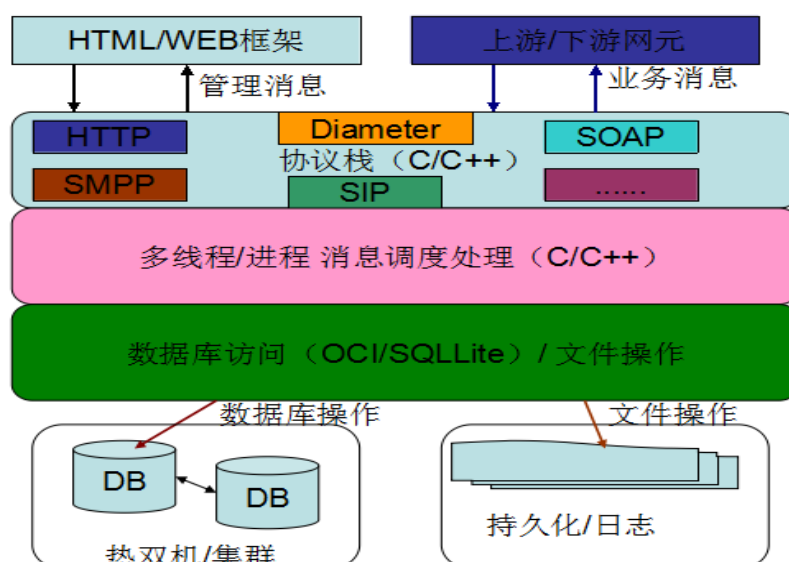


图 1-1 华为电信软件 V1 版逻辑架构图

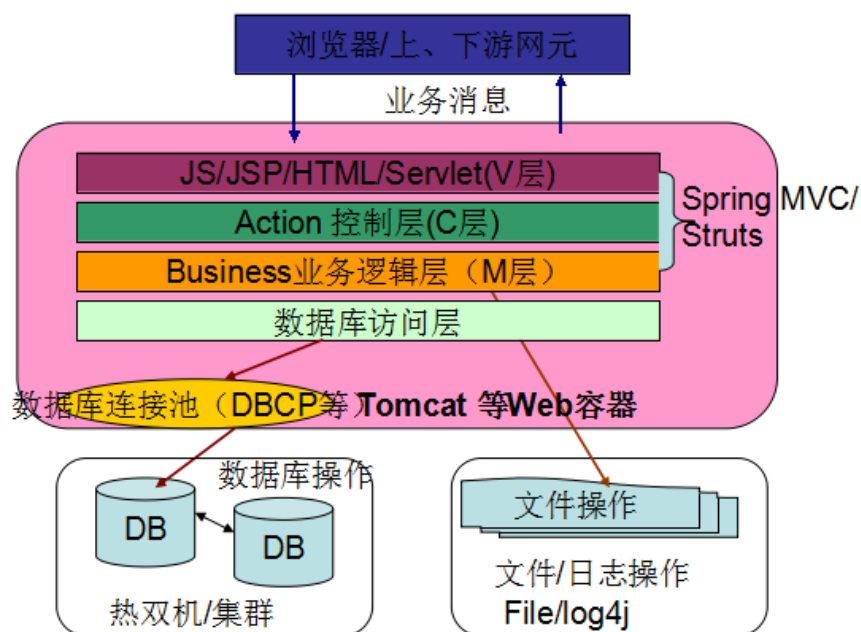
在那个时代，电信软件绝大多数都部署在高性能的小机中，处理各种信令、电信私有协议的接入和解析、复杂业务逻辑处理，系统对处理性能、时延、多核处理的要求非常高。当时 Java 主流版本还是 JDK 1.4.2 (1.4.X)，它在传统的 Web 应用、电子商务网站和政企系统中得到了比较广泛的应用，但是在电信领域并没有大的应用，主要原因如下：

- 1) 在 JDK1.5 之前的早期版本中，Java 在多线程编程、并行处理等方面能力很差，无法在电信软件服务器端使用；
- 2) JDK 1.4.X 对非阻塞 I/O 的支持并不好，相关 NIO 编程的可参考资料和开源框架很少，传统的阻塞 I/O 模型在电信高性能、高可靠场景中力不从心；
- 3) 业界很少有 Java 高性能服务端处理成功的案例，大家普遍对 Java 支持电信级应用场景持怀疑态度；
- 4) 那个时代电信领域的开发者都是 C/C++ 出身，大家对新技术和语言有种天生的排斥。

2005 年之后，随着 Java 在各领域的快速普及和应用，以及基于 Java 的各种开源框架井喷式增长，华为越来越多的产品开始尝试切换到 Java 进行开发，主流架构随即演进到了以 Java 为主的 V2 版本。

1.2.2. Spring + Struts + Tomcat 的第二代架构

2005 年-2008 年间，华为电信软件大多数产品线都切换到 Java 语言进行新产品的设计和开发，当时随着 Struts 的 MVC 模式以及 Spring 对 J2EE 复杂企业应用对象生命周期的配置式管理的流行，华为电信软件绝大多数产品采用基于 Spring + Struts + Tomcat 模式进行开发，数据访问中间件主要采用 iBatis 和 Hibernate, 它的逻辑架构如下所示：



切换到以 Spring + J2EE 容器为基础技术框架之后，应用开发的难度迅速降低，开发效率获得了极大提升。短短 1-2 年时间，公司大多数以 C/C++ 的项目切换到了 Java 语言和 V2 架构上。

1.2.3. 以 SOA 为中心的第三代架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。

随着电信业务的快速发展，电信原有系统和新建设系统之间存在语言、协议、运行环境等诸多差异。如何整合异构系统，实现高效企业集成，也是一个巨大的挑战，此时，企业服务总线（ESB）是个不错的选择。

为了满足电信业务的需求，华为软件研发了 SOA 中间件，它的逻辑架构图如下：

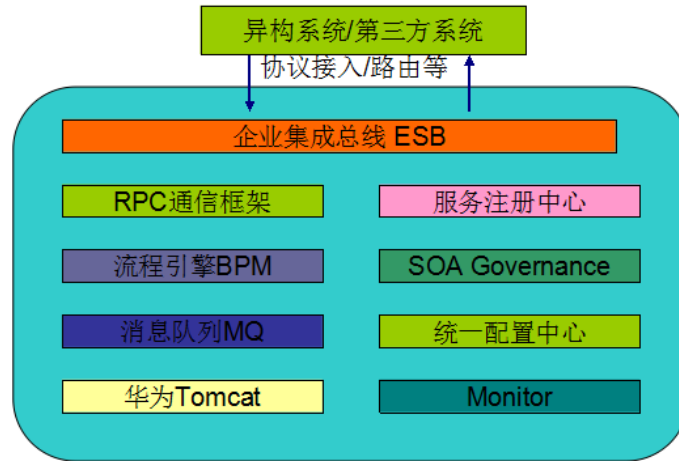


图 1-3 以 SOA 服务化为核心的 V3 架构

SOA 是一种粗粒度、松耦合的以服务为中心的架构，接口之间通过定义明确的协议和接口进行通信。SOA 帮助工程师们站在一个新的高度理解企业级架构中各种组件的开发和部署形式，它可以帮助企业系统架构师以更迅速、可靠和可重用的形式规划整个业务系统。相比于传统的垂直架构，SOA 能够更加从容的应对复杂企业系统集成和需求的快速变化。

1.2.4. 以分布式、云化为核心的第四代架构

随着业务的不断发展，硬件成本的下降，基于 X86 架构的廉价硬件 + 分布式软件的模式在互联网行业得到了大规模应用，分布式架构日趋成熟。

从运营商业务看，尽管高性能的小机仍然是标配，但是运营商业务向数字化转型和云化降成本逐渐成为一种趋势。

传统 SOA 架构中的一些缺陷逐步暴露，例如企业集成总线 ESB 是实体总线，性能线性扩展能力有限；硬件负载均衡器的压力越来越大，不断扩容导致硬件成本增加；随着业务规模的不断增长，传统的数据库、配置中心等逐渐成为单点瓶颈等。

我们需要通过新的分布式架构来解决电信软件面临的成本高、性能无法线性增长等问题，以分布式技术为核心构建的华为分布式中间件应用而生，它主要包括如下组件：

- 1) 高性能、低时延的分布式服务框架；
- 2) 分布式消息队列 MQ；
- 3) 分布式缓存；
- 4) 分布式数据库访问中间件，支持跨库操作，支持异构数据库；

- 5) 软负载 SLB ；
- 6) 分布式日志采集和检索 (Flume + ELK)；
- 7) 分布式实时流式计算框架 ；
- 8) 分布式消息跟踪系统 ；
- 9) 其它.....

自从亚马逊的云计算服务面世以来，云计算技术作为应对笨重的传统 IT 架构的战略，已经成为越来越多的政府和企业的选择，云已经成为 ICT 技术和服务领域的常态。

运营商基础设施云化的主要原因如下：

- 1) IT 资源规模比较大，如何高效的使用这些设备，提升效率，虚拟化是个不错的选择；
- 2) 资源的孤岛现象是比较严重，大部分 IT 系统，依然采用传统的竖井式的建设模式，IT 系统的资源无法在跨系统间进行共享，同时因为各系统建各系统的特点，使得资源的利用率非常低，各个业务有峰值的时候，虽然业务之间峰值不在一块，但是依然起不到消峰的作用，占用的资源比较大；
- 3) 系统的压力也是不均衡，资源由于没法共享，只能采用被动的采购，使得更大容量的设备采购，来应付电信业务增长所需要的扩容；
- 4) 系统部署的周期长、运维也比较难。

为了满足运营商云化的需求，华为相继研发了 IaaS、PaaS 等用于支撑运营商 IT 和基础设施云化，下面让我们一起看下华为软件云化后的逻辑架构：

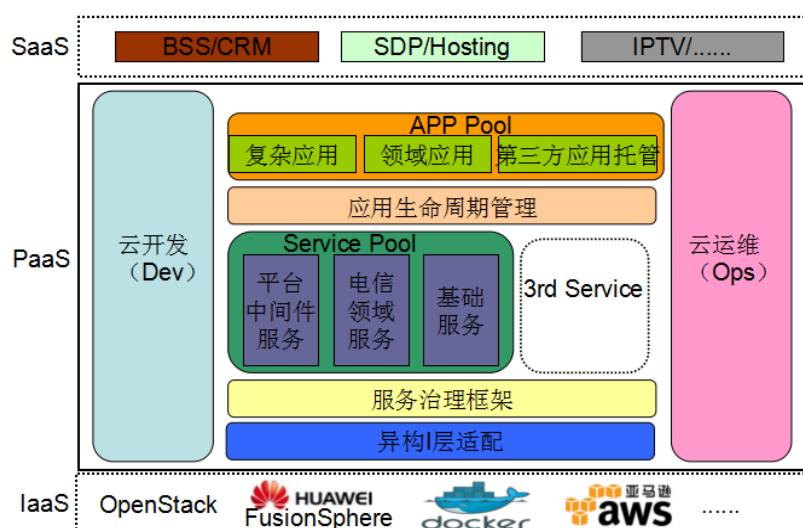


图 1-4 以分布式、云化为核心的 V4 架构

第四代技术架构以分布式、云化为核心，相比于前三代架构，它的核心特性如下：

- 1) 采用分布式技术构建，所有的中间件都没有单点，支持线性增长和弹性伸缩；
- 2) 以微服务架构为核心，打造电信领域的 DevOps(结合华为 PaaS 平台);
- 3) 由传统的 SOA Governance 向微服务治理和自治演进，提升服务治理效能；
- 4) 分布式日志采集 + 实时流式计算框架，更快的故障定界，提升大规模、分布式系统中的运维效率；
- 5) 业务和数据的拆分，分而治之，通过分布式中间件服务向业务屏蔽拆分细节；
- 6) 架构云化带来的巨大优势：资源池提升硬件利用率、DevOps 提升开发和运维效率、应用和服务的自动弹性伸缩、应用和服务故障自动恢复、高 HA、自动化运维等。

1.3. 架构演进中的技术

随着架构的演进，Java 的版本也在不断升级，技术堆栈不断更新，EJB、Spring、RMI、MQ、Node.js、NIO、Hadoop 等。在众多技术堆栈中，我印象最深的就是 Java 的 NIO 类库以及业界成熟 NIO 框架的使用，它在华为软件架构演进中发挥了重大作用，曾立下了汗马功劳。现在，以 Netty 为代表的 NIO 框架已经在华为平台产品和业务产品中得到了广泛的应用。

作为华为软件公司最早使用 Java NIO 技术进行平台开发、2009 年即在全球商用成功的亲历者和实践者，我想跟大家分享下 Java NIO 框架在华为软件以及电信领域的应用和实践。

2. Java NIO 技术的引入

2.1. BIO 带给我们深深伤痛

在 2008 年的时候，我参与设计和开发的一个电信系统在月初出帐期，总是发生大量的连接超时和读写超时异常，业务的失败率相比于平时高了很多，报表中的很多指标都差强人意。后来经过排查，发现问题的主要原因出现在下游网元的处理性能上，月初的时候 BSS 出帐，在出帐期间 BSS 系统运行缓慢，由于双方采用了同步阻塞式的 HTTP+XML 进行通信，导致任何一方处理缓慢都会影响对方的处理性能。按照故障隔离的设计原则，对方处理速度慢或者不回应答，不应该影响系统的其他功能模块或者协议栈，但是在同步阻塞 I/O 通信模型下，这种故障传播和相互影响是不可避免的，很难通过业务层面解决。

受限于当时 Tomcat 和 Servlet 的同步阻塞 I/O 模型，以及在 Java 领域异步 HTTP 协议栈的技术积累不足，当时我们并没有办法完全解决这个问题，只能通过调整线程池策略和 HTTP 超时时间来从业务层面做规避。由于我们的系统是一个全国级的一级系统，需要对接周边各个网元，同时服务器资源十分有限，即便采用了高峰期间动态修改超时时间、优化线程池模型等多种措施，效果依然差强人意。

每当跟客户开会的时候，客户总会提起这个话题：别人响应慢，为啥会导致你的系统阻塞呢，可以返回处理其它消息啊？！我无法跟客户解释技术细节，因为同步阻塞 I/O 仅仅是 Java I/O 的一种实现，操作系统支持非阻塞 I/O 和异步 I/O。

站在技术的角度，客户的需求是合理并且也是可以实现的，当时受限于经验以及其它技术原因，我们无法从根本上解决客户提出的问题，团队有种深深的挫败感，Java BIO 同步阻塞通信导致的各种问题给我留下了一些心理阴影，一直挥之不去。

2.2. BIO 模型存在的问题

传统同步阻塞通信面临的主要问题如下：

- 1) 性能问题：一连接一线程模型导致服务端的并发接入数和系统吞吐量受到极大限制；
- 2) 可靠性问题：由于 I/O 操作采用同步阻塞模式，当网络拥塞或者通信对端处理缓慢会导致 I/O 线程被挂住，阻塞时间无法预测；
- 3) 可维护性问题：I/O 线程数无法有效控制、资源无法有效共享（多线程并发问题），系统可维护性差

传统同步阻塞通信的处理模型图如下：

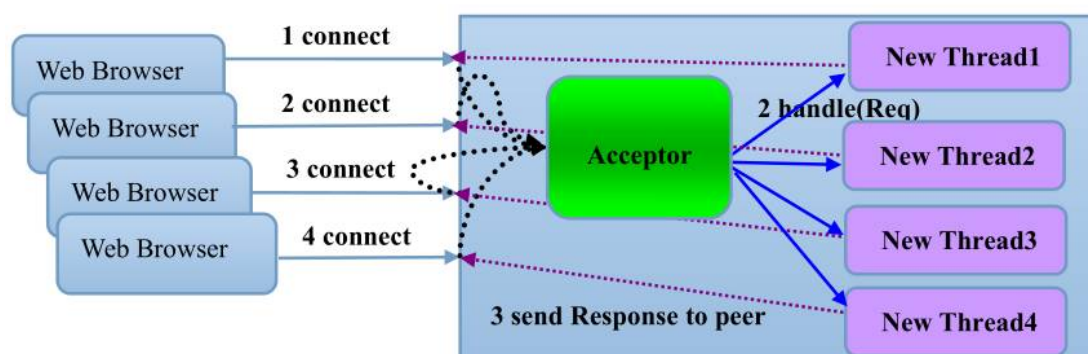


图 2-1 同步阻塞通信模型处理模型图

从上图我们可以看出，每当有一个新的客户端接入，服务端就需要创建一个新的线程（或者重用线程池中的可用线程），每个客户端链路对应一个线程。当客户端处理缓慢或

者网络有拥塞时，服务端的链路线程就会被同步阻塞，也就是说所有的 I/O 操作都可能被挂住，这会导致线程利用率非常低，同时随着客户端接入数的不断增加，服务端的 I/O 线程不断膨胀，直到无法创建新的线程。

同步阻塞 I/O 导致的问题无法在业务层规避，必须改变 I/O 模型，才能从根本上解决这个问题。

2.3. 历史性的引入 Java NIO

2.3.1. Java NIO 被冷落的原因

从 2004 年 JDK1.4 首次提供 NIO 1.0 类库到现在，已经过去了整整 10 年。JSR 51 的设计初衷就是让 Java 能够提供非阻塞、具有弹性伸缩能力的异步 I/O 类库，从而结束 Java 在高性能服务器领域的不利地位。然而，在相当长的一段时间里，Java 的 NIO 编程并没有流行起来，究其原因如下。

大多数高性能服务器，被 C 和 C++ 语言盘踞，由于它们可以直接使用操作系统的异步 I/O 能力，所以对 JDK 的 NIO 并不关心；

移动互联网尚未兴起，基于 Java 的大规模分布式系统极少，很多中小型应用服务对于异步 I/O 的诉求不是很强烈；

高性能、高可靠性领域，例如银行、证券、电信等依然以 C++ 为主导，Java 充当打杂的角色，NIO 暂时没有用武之地；

当时主流的 J2EE 服务器，几乎全部基于同步阻塞 I/O 构建，例如 Servlet、Tomcat 等，由于它们应用广泛，如果这些容器不支持 NIO，用户很难具备独立构建异步协议栈的能力；

异步 NIO 编程门槛比较高，开发和维护一款基于 NIO 的协议栈对很多中小型公司来说像是一场噩梦；

业界 NIO 框架不成熟，很难商用；

国内研发界对 NIO 的陌生和认识不足，没有充分重视。

基于上述几种原因，NIO 编程的推广和发展长期滞后，特别是国内，在 2009 年的时候，几乎无法搜到国内企业成功使用 NIO 技术的案例。

2.3.2. 华为软件引入 Java NIO 的原因

从 2008 年开始，华为软件研发了 Java 版的业务网关，并迅速占领国内外市场。随着产品的推广，在一些高并发、大业务量的局点相继出现了几起事故，质量回溯的结果都指向了 Java BIO 通信模型，包括 Servlet 2.X 的同步阻塞 I/O、Tomcat 5.X（当时没使用 5.5）的同步 I/O、以及其它的同步 I/O 协议栈。

问题根因已经很清楚，如果不改变同步 I/O 通信模型，问题会继续发生，对于运营商而言，这是不可能接受的事情。自古华山一条路，即然业界没有成熟的异步 I/O 协议栈，那我们就自研。

2009 年初，由于对技术的热爱，我作为业务骨干被领导派去参加异步高性能网关平台的研发工作，与两位资深的架构师（其中一位工作 20 年，做华为交换机出身）一起合作。这是我第一次全面接触异步 I/O 编程和高性能电信级协议栈的开发，眼界大开——异步高性能内部协议栈、异步 HTTP、异步 SOAP、异步 SMPP……所有的协议栈都是异步非阻塞模式。

后来的性能测试表明：基于 Reactor 模型统一调度的长连接和短连接协议栈，无论是性能、可靠性还是可维护性，都可以“秒杀”传统基于 BIO 开发的应用服务器和各种协议栈，这种指标差异本质上是一种技术代差。

2009 年底，基于异步网关平台研发的 XX 业务产品在海外某运营商成功上线，它的高性能、低时延和高 HA 令局方惊叹不已，原来准备的 20 多台小机最后只使用了 3 台，为客户节省了一大把\$。

2.3.3. 那些年我们踩过的 NIO “坑”

在我从事异步 NIO 编程的 2009 年，业界还没有成熟的 NIO 框架，那个时候 Mina 刚刚开始起步，功能和性能都达不到商用标准。最困难的是，国内 Java 领域的异步通信还没有流行，整个业界的积累都非常少。那个时候资料匮乏，能够交流和探讨的圈内人很少，一旦踩住“地雷”，就需要夜以继日地维护。在随后 2 年多的时间里，经历了 10 多次的在通宵、凌晨被一线的运维人员电话吵醒等种种磨难之后，我们自研的 NIO 框架才逐渐稳定和成熟。期间，解决的 BUG 总计 20~30 个。

为了解决这些 Bug，2 年中我经历了 10 几个通宵，现在回想起来仍历历在目，特别是 JDK epoll 空轮询导致的 CPU 100%，更是坑中之坑（JDK NIO 类库的 Bug），曾令多少产品中招，包括 Mina、Netty、Jetty 等著名开源框架。

2.4. 从 Java 原生 NIO 到 NIO 框架

从 2011 年开始，华为软件主要使用 NIO 框架 Netty 进行通信软件的开发，为什么继续使用原声的 Java NIO 类库，下面给出了我们切换的原因。

2.4.1. JAVA 原生 NIO 类库的复杂性

在分析 Java 原生 NIO 类库复杂性之前，我们首先看下最简单的 NIO 服务端和客户端创建流程。

最简单的 NIO 服务端创建程序流程：

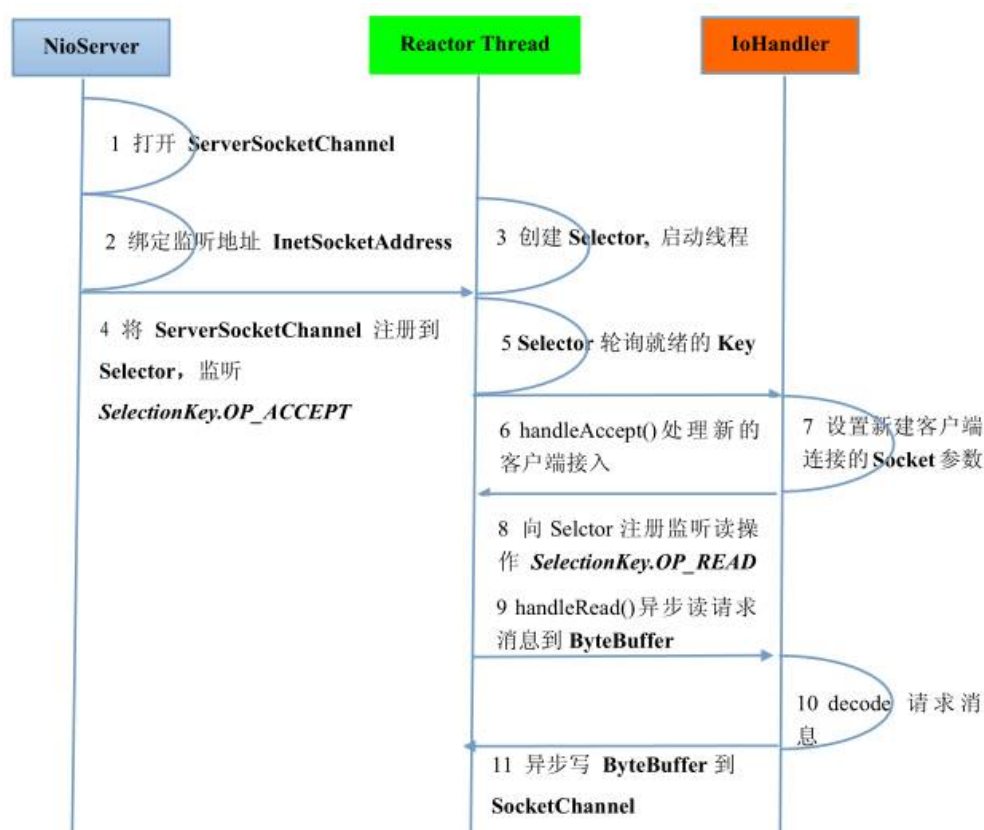


图 2-2 Java NIO 服务端创建流程

最简单的 Java NIO 客户端创建流程如下：

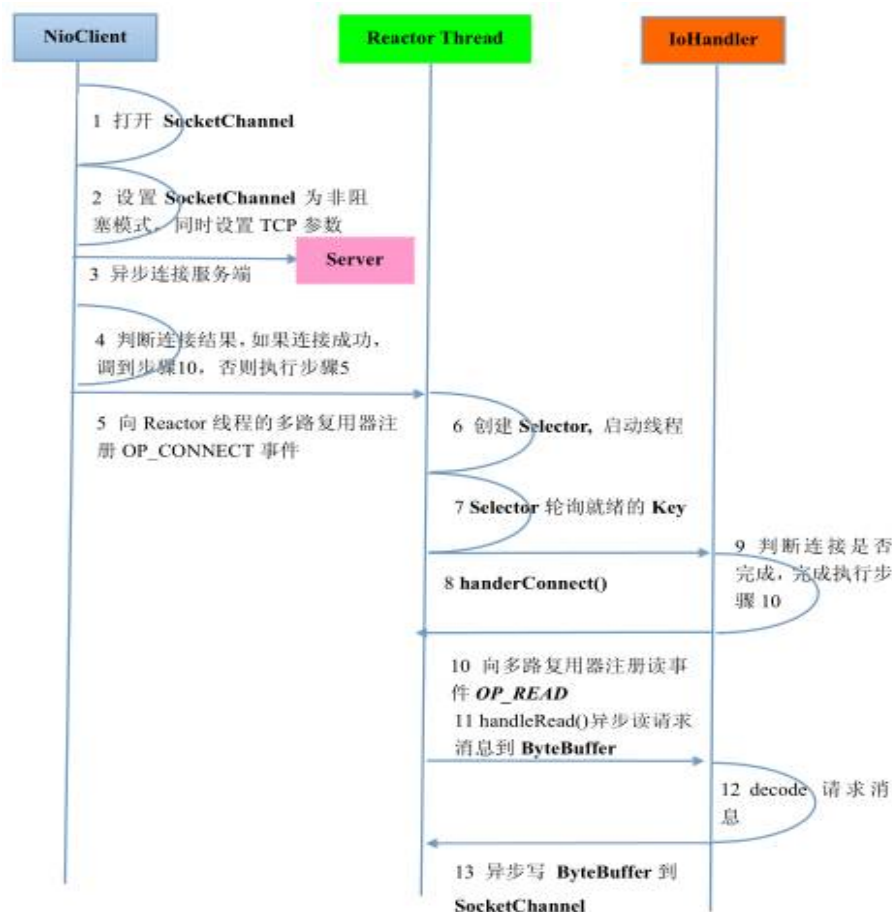


图 2-3 Java NIO 客户端创建流程

现在我们总结一下为什么不建议开发者直接使用 JDK 的 NIO 类库进行开发，具体原因如下：

- (1) NIO 的类库和 API 繁杂，使用麻烦，你需要熟练掌握 Selector、ServerSocketChannel、SocketChannel、ByteBuffer 等；
- (2) 需要具备其他的额外技能做铺垫，例如熟悉 Java 多线程编程。这是因为 NIO 编程涉及到 Reactor 模式，你必须对多线程和网络编程非常熟悉，才能编写出高质量的 NIO 程序；
- (3) 可靠性能力补齐，工作量和难度都非常大。例如客户端面临断连重连、网络闪断、半包读写、失败缓存、网络拥塞和异常码流的处理等问题，NIO 编程的特点是功能开发相对容易，但是可靠性能力补齐的工作量和难度都非常大；
- (4) JDK NIO 的 BUG，例如臭名昭著的 epoll bug，它会导致 Selector 空轮询，最终导致 CPU 100%。官方声称在 JDK1.6 版本的 update18 修复了该问题，但是直到

JDK1.7 版本该问题仍旧存在，只不过该 BUG 发生概率降低了一些而已，它并没有被根本解决。该 BUG 以及与该 BUG 相关的问题单可以参见以下链接内容：

◎ http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6403933

◎ http://bugs.java.com/bugdatabase/view_bug.do?bug_id=2147719

异常堆栈如下：

```
java.lang.Thread.State: RUNNABLE
    at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
    at sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:210)
    at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:65)
    at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:69)
    - locked <0x0000000750928190> (a sun.nio.ch.Util$2)
    - locked <0x00000007509281a8> (a
java.util.Collections$ UnmodifiableSet)
    - locked <0x0000000750946098> (a sun.nio.ch.EPollSelectorImpl)
    at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:80)
    at net.spy.memcached.MemcachedConnection.handleIO(Memcached
Connection.java:217)
    at net.spy.memcached.MemcachedConnection.run(MemcachedConnection.
java:836)
```

2.4.2. 以 Netty 为代表的 NIO 框架已经成熟

Netty 是业界最流行的 NIO 框架之一，它的健壮性、功能、性能、可定制性和可扩展性在同类框架中都是首屈一指的，它已经得到成百上千的商用项目验证，例如 Hadoop 的 RPC 框架 avro 使用 Netty 作为底层通信框架；很多其他业界主流的 RPC 框架，也使用 Netty 来构建高性能的异步通信能力。

通过对 Netty 的分析，我们将它的优点总结如下：

- 1) API 使用简单，开发门槛低；
- 2) 功能强大，预置了多种编解码功能，支持多种主流协议；
- 3) 定制能力强，可以通过 ChannelHandler 对通信框架进行灵活地扩展；
- 4) 性能高，通过与其他业界主流的 NIO 框架对比，Netty 的综合性能最优；
- 5) 成熟、稳定，Netty 修复了已经发现的所有 JDK NIO BUG，业务开发人员不需要再为 NIO 的 BUG 而烦恼；

6) 社区活跃，版本迭代周期短，发现的 BUG 可以被及时修复，同时，更多的新功能会加入；

7) 经历了大规模的商业应用考验，质量得到验证。在互联网、大数据、网络游戏、企业应用、电信软件等众多行业得到成功商用，证明了它已经完全能够满足不同行业的商业应用了。

正是因为这些优点，Netty 逐渐成为 Java NIO 编程的首选框架，它也是华为公司首选的 Java NIO 通信框架，公司已经将其纳入到公司级的优选开源第三方软件库中。

3. Netty 在电信领域的实践

电信行业软件的几个特点：

1) 高可靠性：5 个 9；

2) 高性能、低时延；

3) 大规模组网：例如中国移动、Telfonica 拉美十三国、沃达丰等，业务组网规模都非常大；

4) 复杂的网络形态：对接不同设备提供商的网元和系统。

3.1. 高性能、低时延

3.1.1. 非阻塞 I/O 模型

在 I/O 编程过程中，当需要同时处理多个客户端接入请求时，可以利用多线程或者 I/O 多路复用技术进行处理。I/O 多路复用技术通过把多个 I/O 的阻塞复用到同一个 select 的阻塞上，从而使得系统在单线程的情况下可以同时处理多个客户端请求。与传统的多线程/多进程模型比，I/O 多路复用的最大优势是系统开销小，系统不需要创建新的额外进程或者线程，也不需要维护这些进程和线程的运行，降低了系统的维护工作量，节省了系统资源。

我们采用 Netty 的 NIO 传输模式来提升 I/O 操作的效率，节省线程等其它资源开销，它的模型如下所示：

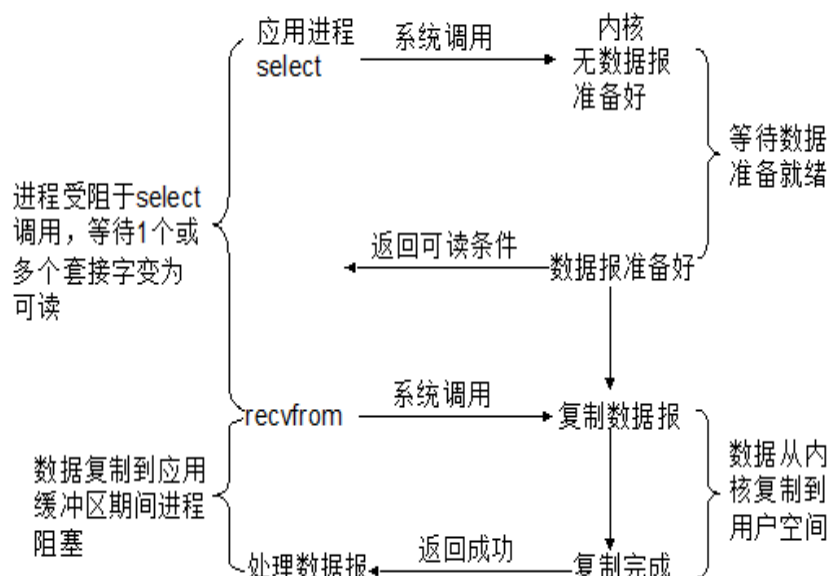


图 3-1 Netty 的非阻塞 I/O 调度模型

3.1.2. 高性能的序列化框架

在华为软件，对于序列化框架的选择，我们遵循如下几个原则：

- 1) 序列化后的码流大小（网络带宽的占用）；
- 2) 序列化&反序列化的性能（CPU、内存等资源占用）；
- 3) 是否支持跨语言（异构系统的对接和开发语言切换）；
- 4) 高并发调用时的性能，是否随着线程并发数线性增长。

基于上述的指标，目前最常用的选择是：Google 的 ProtoBuf 和 Apache 的 Thrift。

Netty 原生提供了对 ProtoBuf 序列化框架的支持，它的优点如下：

- 1) 在谷歌内部长期使用，产品成熟度高；
- 2) 跨语言、支持多种语言，包括 C++、Java 和 Python；
- 3) 编码后的消息更小，更加有利于存储和传输；
- 4) 编解码的性能非常高；
- 5) 支持不同协议版本的前向兼容；
- 6) 支持定义可选和必选字段。

Netty ProtoBuf 服务端开发示例如下：

```
// 配置服务端的 NIO 线程组
EventLoopGroup bossGroup = new NioEventLoopGroup();
EventLoopGroup workerGroup = new NioEventLoopGroup();
try {
    ServerBootstrap b = new ServerBootstrap();
    b.group(bossGroup, workerGroup)
        .channel(NioServerSocketChannel.class)
        .option(ChannelOption.SO_BACKLOG, 100)
        .handler(new LoggingHandler(LogLevel.INFO))
        .childHandler(new ChannelInitializer<SocketChannel>() {
            @Override
            public void initChannel(SocketChannel ch) {
                ch.pipeline().addLast(
                    new ProtobufVarint32FrameDecoder());
                ch.pipeline().addLast(
                    new ProtobufDecoder(
                        SubscribeReqProto.SubscribeReq
                            .getDefaultInstance()));
                ch.pipeline().addLast(
                    new ProtobufVarint32LengthFieldPrepender());
                ch.pipeline().addLast(new ProtobufEncoder());
                ch.pipeline().addLast(new SubReqServerHandler());
            }
        });
}
```

Thrift 相对复杂一些，需要将编解码框架从 Thrift 中剥离出来，然后利用 Netty 编解码框架的扩展性定制实现，在此不再赘述。

3.1.3. 收敛的 Reactor 线程模型

Java 线程采用抢占的方式争夺 CPU 等资源，当系统线程数增大到一定量级之后，性能不仅没有提升，反而下降。

对于大型的电信应用，如果使用 Tomcat 等做 Web 容器，为了保证吞吐量和性能，HTTP 线程池的最大线程数往往配置为 1024。在系统运行期间我们 Dump 线程堆栈，发现大量的线程竞争，这不仅导致 HTTP 协议栈的性能下降，更影响其它业务处理线程的执行效率。

使用 Netty 之后，我们通过控制 `NioEventLoopGroup` 的 `NioEventLoop` 个数来收敛线程，防止线程膨胀。`NioEventLoop` 聚合了一个多路复用器 `Selector`，可以高效的处理 `N` 个 `Channel`，它的线程模型如下：

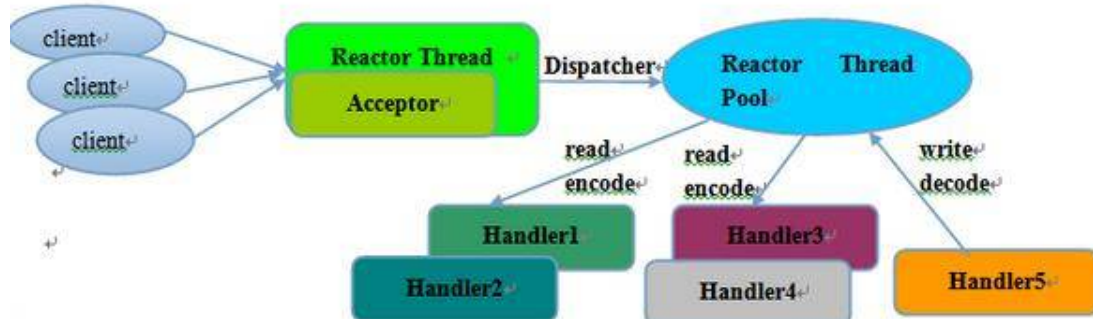


图 3-1 Netty Reactor 线程模型

3.1.4. 其它优化

为了进一步提升性能，降低时延，我们还采用了其它一些优化措施，总结如下：

- 1) 使用 Netty 4 的内存池，减少业务高峰期 `ByteBuf` 频繁创建和销毁导致的 GC 频率和时间；
- 2) 在程序中充分利用 Netty 提供的“零拷贝”特性，减少额外的内存拷贝，例如使用 `CompositeByteBuf` 而不是分别为 `Head` 和 `Body` 各创建一个 `ByteBuf` 对象；
- 3) TCP 参数的优化，设置合理的 `Send` 和 `Receive Buffer`，通常建议值为 64K - 128K；
- 4) 软中断：如果 Linux 内核版本支持 RPS (2.6.35 以上版本)，开启 RPS 后可以实现软中断，提升网络吞吐量；
- 5) 无锁化串行开发理念：使用 Netty 4.X 版本，天生支持串行化处理；业务开发过程中，遵循 Netty 4 的线程模型优化理念，防止人为增加线程竞争。

3.2. 高 HA

3.2.1. 内存保护

为了提升内存的利用率，Netty 提供了内存池和对象池。但是，基于缓存池实现以后需要对内存的申请和释放进行严格的管理，否则很容易导致内存泄漏。

如果不采用内存池技术实现，每次对象都是以方法的局部变量形式被创建，使用完成之后，只要不再继续引用它，JVM 会自动释放。但是，一旦引入内存池机制，对象的生命

周期将由内存池负责管理，这通常是个全局引用，如果不显式释放 JVM 是不会回收这部分内存的。

对于 Netty 的用户而言，使用者的技术水平差异很大，一些对 JVM 内存模型和内存泄漏机制不了解的用户，可能只记得申请内存，忘记主动释放内存，特别是 JAVA 程序员。

为了防止因为用户遗漏导致内存泄漏，Netty 在 Pipe line 的尾 Handler 中自动对内存进行释放。

缓冲区内存溢出保护：做过协议栈的读者都知道，当我们对消息进行解码的时候，需要创建缓冲区。缓冲区的创建方式通常有两种：

- 1) 容量预分配，在实际读写过程中如果不够再扩展；
- 2) 根据协议消息长度创建缓冲区。

在实际的商用环境中，如果遇到畸形码流攻击、协议消息编码异常、消息丢包等问题时，可能会解析到一个超长的长度字段。笔者曾经遇到过类似问题，报文长度字段值竟然是 2G 多，由于代码的一个分支没有对长度上限做有效保护，结果导致内存溢出。系统重启后几秒内再次内存溢出，幸好及时定位出问题根因，险些酿成严重的事故。

Netty 提供了编解码框架，因此对于解码缓冲区的上限保护就显得非常重要。下面，我们看下 Netty 是如何对缓冲区进行上限保护的：

- 1) 在内存分配的时候指定缓冲区长度上限；
- 2) 在对缓冲区进行写入操作的时候，如果缓冲区容量不足需要扩展，首先对最大容量进行判断，如果扩展后的容量超过上限，则拒绝扩展；
- 3) 在解码的时候，对消息长度进行判断，如果超过最大容量上限，则抛出解码异常，拒绝分配内存。

3.2.2. 流量整形

电信系统一般都有多个网元组成，例如参与短信互动，会涉及到手机、基站、短信中心、短信网关、SP/CP 等网元。不同网元或者部件的处理性能不同。为了防止因为浪涌业务或者下游网元性能低导致下游网元被压垮，有时候需要系统提供流量整形功能。

流量整形（Traffic Shaping）是一种主动调整流量输出速率的措施。一个典型应用是基于下游网络结点的 TP 指标来控制本地流量的输出。流量整形与流量监管的主要区别在于，流量整形对流量监管中需要丢弃的报文进行缓存——通常是将它们放入缓冲区或队列

内，也称流量整形（Traffic Shaping，简称 TS）。当令牌桶有足够的令牌时，再均匀的向外发送这些被缓存的报文。流量整形与流量监管的另一区别是，整形可能会增加延迟，而监管几乎不引入额外的延迟。

流量整形的原理示意图如下：

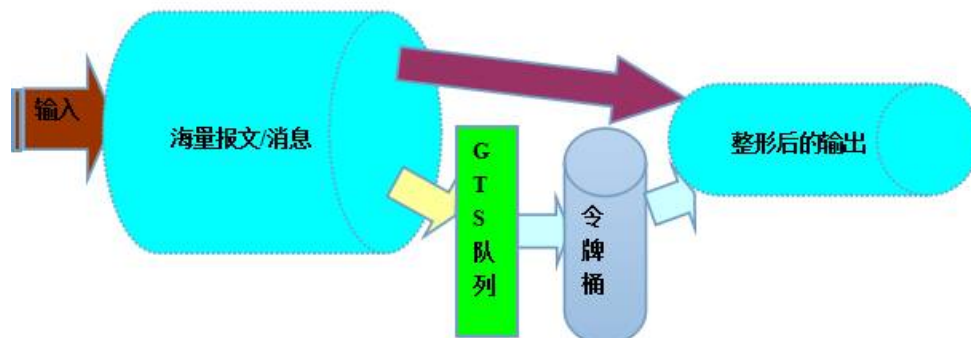


图 3-2 Netty 流量整形原理图

Netty 内置两种流量整形策略，可以方便的被用户添加和使用：

- 1) 全局流量整形的作用范围是进程级的，无论你创建了多少个 Channel，它的作用域针对所有的 Channel。用户可以通过参数设置：报文的接收速率、报文的发送速率、整形周期；
- 2) 单链路流量整形与全局流量整形的最大区别就是它以单个链路为作用域，可以对不同的链路设置不同的整形策略，整形参数与全局流量整形相同。

3.2.3. 其它可靠性措施

其它比较重要的可靠性措施如下：

- 1) 客户端连接超时控制策略；
- 2) 链路断连重连策略；
- 3) 链路异常关闭资源释放；
- 4) 解码失败的异常处理策略；
- 5) 链路异常的捕获和处理；
- 6) I/O 线程的释放。

3.3. 华为软件对 Netty 的优化

针对电信软件的特点，结合华为软件的实际业务需求，我们对 Netty 进行了优化，优化的策略如下：

- 1) 能够通过 Netty 提供的扩展点实现的，通过扩展点实现，不自己造轮子；
- 2) 不允许修改 Netty 源码，基于 Netty 提供的接口，开发华为自己的优化实现类；
- 3) 华为优化实现类独立打包，对原 Netty 类库是二进制依赖，不修改 Netty 原类库；
- 4) 服务端和客户端创建时，传递华为自己的实现类参数。

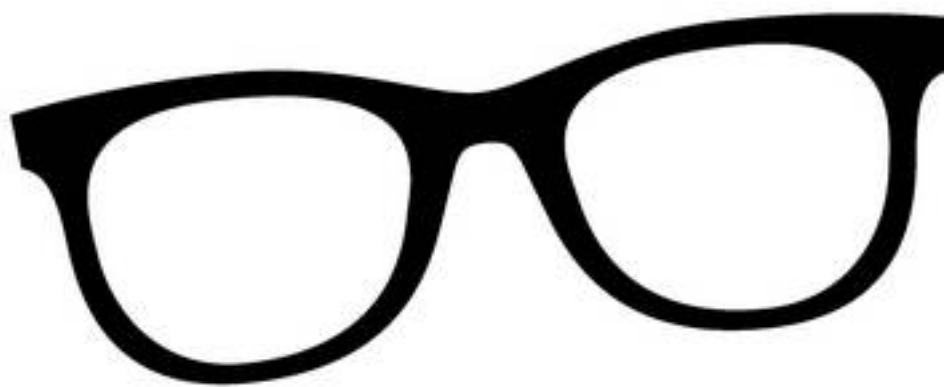
华为的主要优化点总结如下：

- 1) 安全性改造：满足华为公司安全红线、电信运营商的安全需求相关改造；
- 2) 可靠性增强：消息发送队列的上限保护、链路中断时缓存中待发送消息回调通知业务、增加错误码、异常日志打印抑制、I/O 线程健康度检测等；
- 3) 可定位性增强：单链路的网络吞吐量、接收发送的速度、接收\发送的总字节数、畸形码流检测机制、解码时延超大消息日志打印等。

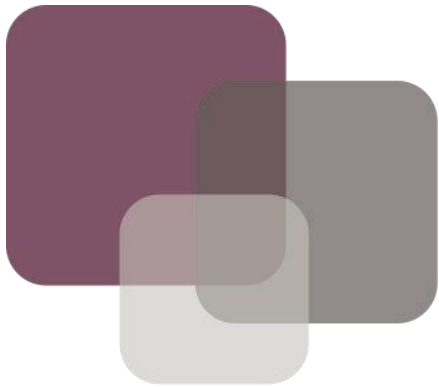
4. 作者简介

李林锋，2007 年毕业于东北大学，2008 年进入华为公司从事高性能通信软件的设计和开发工作，有 7 年 NIO 设计和开发经验，精通 Netty、Mina 等 NIO 框架和平台中间件，现任华为软件平台架构部架构师，《Netty 权威指南》作者。目前从事华为下一代中间件和 PaaS 平台的架构设计工作。

Docker 专栏



云生态专刊



京东 618：Docker 扛大旗，弹性伸缩成重点

作者 郭蕾

不知不觉中，年中的 618 和年终的 11.11 已经成为中国电商的两大促销日，当然，这两天也是一年中系统访问压力最大的两天。对于京东而言，618 更是这一年中最大的一次考试，考点是系统的扩展性、稳定性、容灾能力、运维能力、紧急故障处理能力。弹性计算云是京东 2015 年研发部战略项目，它基于 Docker 简化了应用的部署和扩容，提高了系统的伸缩能力。目前京东的图片系统、单品页、频道页、风控系统、缓存、登录、团购、O2O、无线、拍拍等业务都已经运行在弹性计算云系统中。

过去的一段时间里，弹性计算云项目在京东内部获得了广泛应用，并且日趋稳定成熟。一方面，这个项目可以更有效地管理机器资源，提高资源利用率；另外还能大幅提高生产效率，让原来的申请机器上线扩容逐渐过渡到全自动维护。京东弹性计算云项目将深刻影响京东未来几年的基础架构。

受访嘉宾介绍

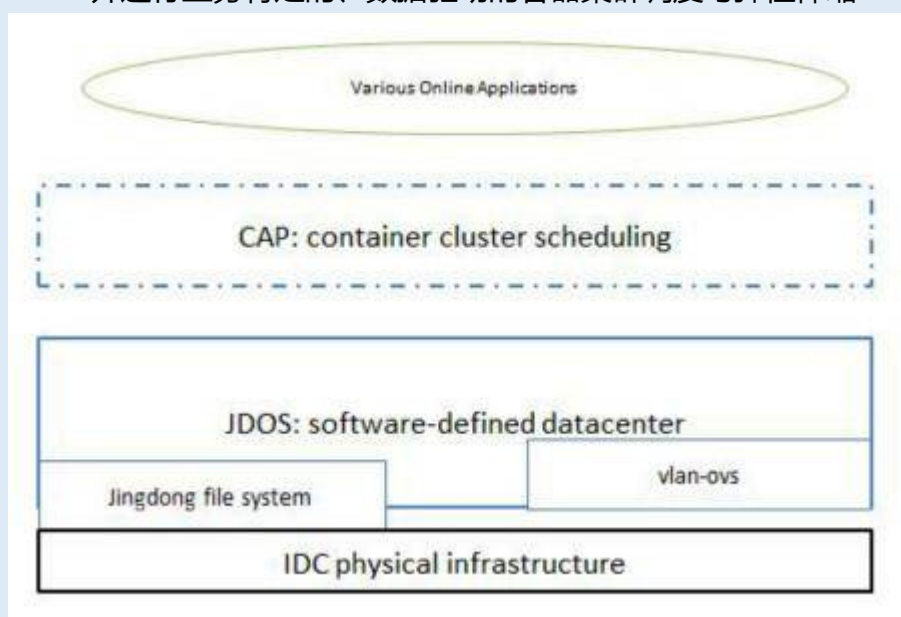
刘海锋，京东云平台首席架构师、系统技术部负责人。系统技术部专注于基础服务的自主研发与持续建设，包括分布式存储、以内存为中心的 NoSQL 服务、图片源站、内容分发网络、消息队列、内部 SOA 化、弹性计算云等核心系统，均大规模部署以支撑京东集团的众多业务。

InfoQ：能否介绍下京东弹性计算云项目的情况，你们什么时候开始使用 Docker 的？目前有多大的规模？

刘海锋：弹性计算云项目在去年第四季度开始研发，今年春节后正式启动推广应用。经过半年多的发展，逐渐做到了一定规模。截至 6 月 17 日，我们线上运行了 9853 个 Docker 实例（注：无任何夸大）以及几百个 KVM 虚拟机。京东主要的一些核心应用比如商品详情页、图片展现、秒杀、配送员订单详情等等都部署在弹性云中。

弹性计算云项目也作为今年 618 的扩容与灾备资源池，这估计是国内甚至世界上最大规模的 Docker 应用之一。随着业务的发展以及 IDC 的增加，预计今年年底规模会翻两番，京东大部分应用程序都会通过容器技术来发布和管理。

系统架构可以这样简洁定义：弹性计算云 = 软件定义数据中心 + 容器集群调度。整个项目分成两层架构，底层为基础平台，系统名 JDOS，通过『OpenStack married with Docker』来实现基础设施资源的软件管理，Docker 取代 VM 成为一等公民，但这个系统目标是统一生产物理机、虚拟机与轻量容器；上层为应用平台，系统名 CAP，集成部署监控日志等工具链，实现『无需申请服务器，直接上线』，并进行业务特定的、数据驱动的容器集群调度与弹性伸缩



InfoQ：能否谈谈你们的 Docker 使用场景？在 618 这样的大促中，Docker 这样的容器有什么优势？618 中有哪些业务跑到 Docker 中？

刘海锋：目前主要有两类场景：无状态的应用程序，和缓存实例。这两类场景规模最大也最有收益。不同的场景具体需求不同，因此技术方案也不相同。内部我们称呼为“胖容器”与“瘦容器”技术。从资源抽象角度，前者带独立 IP 以及基础工具链如同一台主机，后者可以理解为物理机上面直接启动 cgroup 做资源控制加上镜像机制。

618 这样的大促备战，弹性计算云具备很多优势：非常便捷的上线部署、半自动或全自动的扩容。Docker 这样的操作系统级虚拟化技术，启动速度快，资源消耗低，非常适合私有云建设。

今年 618，是京东弹性计算云第一次大促亮相，支持了有很多业务的流量。比如图片展现 80%流量、单品页 50%流量、秒杀风控 85%流量、虚拟风控 50%流量，还有三级列表页、频道页、团购页、手机订单详情、配送员主页等等，还有全球购、O2O 等新业务。特别是，今年 618 作战指挥室大屏监控系统都是部署在弹性云上的。

InfoQ：你们是如何结合 Docker 和 OpenStack 的？网络问题是如何解决的？

刘海锋：我们深度定制 OpenStack，持续维护自己的分支，称之为 JDOS (the Jingdong Datacenter Operating System)。JDOS 目标很明确：统一管理和分配 Docker、VM、Bare Metal，保证稳定高性能。网络方面不玩复杂的，线上生产环境划分 VLANs + Open vSwitch。SDN 目前没有显著需求所以暂不投入应用。我们以『研以致用』为原则来指导技术选择和开发投入。

InfoQ：能否谈谈你们目前基于 Docker 的 workflow ？

刘海锋：弹性计算平台集成了京东研发的统一工作平台（编译测试打包上线审批等）、自动部署、统一监控、统一日志、负载均衡、数据库授权，实现了应用一键部署，并且全流程处理应用接入，扩容、缩容、下线等操作。支持半自动与全自动。

InfoQ：这么多的容器，你们是如何调度的？

刘海锋：容器的调度由自主研发的 CAP (Cloud Application Platform) 来控制，并会根据应用配置的策略来进行调度；在创建容器的时候，会根据规格、镜像、机房和交换机等策略来进行创建；创建完容器后，又会根据数据库策略、负载策略、监控策略等来进行注册；在弹性调度中，除了根据容器的资源情况，如 CPU 和连接数，还会接合应用的 TPS 性能等等来综合考虑，进行弹性伸缩。

目前已经针对两大类在线应用实现自动弹性调度，一是 Web 类应用，二是接入内部 SOA 框架的服务程序。大规模容器的自动化智能调度，我们仍在进一步做研究与开发。

InfoQ：目前主要有哪些业务使用了 Docker？业务的选择方面有什么建议？

刘海锋：目前有 1000 个应用已经接入弹性云，涵盖京东各个业务线，包括很多核心应用。目前我们主要支持计算类业务，存储类应用主要应用到了缓存。数据库云服务也将通过 Docker 进行部署和管理。

特别强调的是，业务场景不同，技术方案就有差别。另外，有些对隔离和安全比较敏感的业务就分配 VM。技术无所谓优劣和新旧，技术以解决问题和创造业务价值为目的。

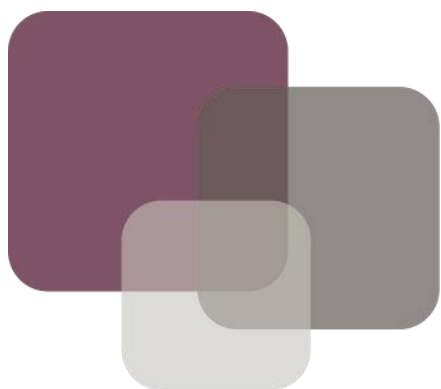
InfoQ：你们的缓存组件也跑在 Docker 中，这样做有什么好处？IO 什么的没有问题吗？有什么好的经验可以分享？

刘海锋：我们团队负责一个系统叫 JIMDB，京东统一的缓存与高速 NoSQL 服务，兼容 Redis 协议，后台保证高可用与横向扩展。系统规模增长到现在的三千多台大内存机器，日常的部署操作、版本管理成为最大痛点。通过引入 Docker，一键完成容器环境的缓存集群的全自动化搭建，大幅提升了系统运维效率。

技术上，缓存容器化的平台并不基于 OpenStack，而是基于 JIMDB 自身逻辑来开发。具体说来，系统会根据需求所描述的容量、副本数、机房、机架、权限等约束创建缓存容器集群，并同时在配置中心注册集群相关元数据描述信息，通过邮件形式向运维人员发出构建流水详单，并通知用户集群环境构建完成。调度方面，不仅会考虑容器内进程，容器所在机器以及容器本身当前的实时状况，还会对它们的历史状况进行考察。一旦缓存实例触发内存过大流量过高等扩容条件，系统会立即执行扩容任务创建新的容器分摊容量和流量，为保证服务质量，缓存实例只有在过去一段时间指标要求持续保持低位的情况下才会缩容。在弹性伸缩的过程中，会采用 Linux TC 相关技术保证缓存数据迁移速度。

InfoQ：使用过程中有哪些坑？你们有做哪些重点改进？

刘海锋：坑太多了，包括软件、硬件、操作系统内核、业务使用方式等等。底层关键改进印象中有两个方面：第一，Docker 本地存储结构，抛弃 Device Mapper、AUTFS 等选项，自行定制；第二，优化 Open vSwitch 性能。比如，优化 Docker 镜像结构，加入多层合并、压缩、分层 tag 等技术，并采用镜像预分发技术，可以做到秒级创建容器实例；优化 Open vSwitch 转发层，显著提升网络小包延迟。



Docker 的安全基准

作者 Chris Swan ，译者 朱明

Docker 公司与 Center for Internet Security (CIS) 合作，制作了一份[基准文档](#)，包含很多针对部署 Docker 的安全性的建议。Diogo Mónica 在一篇博客[“理解 Docker 的安全性和最佳实践”](#)中公布了该基准，最近她和 Nathan McCauley 一起受雇来领导 Docker 的安全团队。该团队也发布了[“容器安全性入门”](#)白皮书。

基准文档涵盖了运行 Docker 的宿主（系统）的配置、Docker 本身的配置，以及在 Docker 管理下运行的容器的配置。该基准针对 Docker 1.6.0，（这是）本文写作时的最新版本，并且基于（使用）红帽企业 Linux (RHEL) 第 7 版或 Debian 第 8 版作为宿主操作系统（OS）。基准的附录中有包含每项建议的检查清单。

建议分为两个级别，第一级涉及的措施“实用而谨慎，提供明确的安全方面的好处，而又不妨碍运用的技术超出可接受的范围。”第二级的建议更加具有侵入性，被描述为“针对把安全性视为头等大事的环境或情况，作为深入全面的防御措施，或者会消极地抑制技术的运用或性能。”第一级的建议适用于宿主操作系统和 Docker，而第二级的 3 项关于强制访问控制（mandatory access control，MAC）和端点保护平台（endpoint protection platform，EPP）的建议，只适用于 Docker。

很多建议在其审核一节有脚本片段，可以用来判断配置是否处于所需的状态，这表明，大部分基准可以组织成脚本，用来检查（及重新配置）宿主是否符合基准。补救措施在适当的情况下也用脚本片段描述，但是这些脚本不太有用，因为在大多数情况下，必须编辑启动配置。鉴于所有主要的 Linux 发行版本现在都切换到 systemd 作为其默认的启动系统，CIS 可能会选择显示与此相应的配置步骤，但这也有可能造成许多仍在旧发行版本上运行 Docker 的用户的困惑。

尽管有些建议是相当通用的，比如“不要在生产环境使用开发工具”，大多数建议还是很具体和可操作的，比如“不要使用 aufs”。因此该基准可以用来剖析某一 Docker 环境，决

定可以采取以提高其安全性的实际步骤。当由于底层宿主操作系统的不同而存在多种选择时，就会给出由 Docker 核心团队和其他人撰写的很多外部指导性文档。

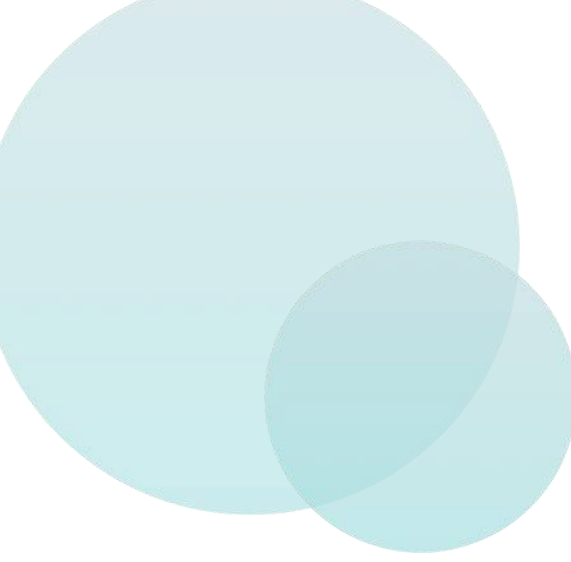
一些可能以前被认为是容器的最佳实践，比如，“一个容器只包含一个进程”，以及，“不要在容器内运行 SSH”，在基准中作为安全性的建议。把这认为是安全性的问题，将有可能进一步削弱把容器作为小型虚拟机的使用模式。

基准中有一些特别之处希望能在将来的版本中解决。其中之一是在 Docker 将来的版本中对用户命名空间的支持，表明这有可能出现在 1.6 版本中（即使该基准就是关于 1.6 版本的）。这也许说明用户命名空间的整合的确要比预想的更成问题。基准还建议使用 [nsenter](#)，尽管其使用已经在很大程度上被 Docker 1.3 引入的 `exec` 命令所取代。

伴随基准推出的白皮书表明，Docker 公司正努力把容器定位为改进安全性的方法，但就象任何新技术一样，公司面临艰难的时刻说服对安全性有顾虑的客户，在生产环境运行是安全的。CIS 基准的发布为那些想在生产环境运行 Docker 的（用户）提供了可测量的手段来评估其安全状况，这很可能有助于缓解任何顾虑。对于拥抱 Docker 来打包应用程序、加速持续交付管道、以及促进微服务架构的开发者，这应当使（部署）到生产环境中的最后步骤在一定程度上更加容易。

观点&趋势

云生态专刊



为私有云结庐而做 “隆中对”（下）

作者 张鑫

编者按：本文系 InfoQ 中文站向 ZStack 项目创始人张鑫的约稿。2015 年 4 月正式对外开源的 [ZStack](#) 项目宣称要解决在 OpenStack 中得不到解决的问题，并明确将项目目标指向目前前景似乎越来越不明朗的私有云市场。作为 ZStack 项目的架构师、CloudStack 的前开发人员，张鑫对于私有云——或者说 on-premise 的企业 IT——到底遇到了什么问题、此类服务到底应该怎么做，是如何构想的？本文将分享他对这个话题的思考。

写在前面：由于企业私有云市场迟迟未打开，近两年来已有多家 IaaS 企业被廉价收购甚至倒闭，业界已经开始出现一种质疑私有云是伪命题的声音。在此，作者想借 ZStack 发布的机会，梳理一下私有云的过去和现状，并展望一下它的未来。

私有云为什么还没有成功

我们知道私有云市场的四位先行者——Eucalyptus、CloudStack、OpenNebula 已渐渐淡出市场，OpenStack 虽然比较火爆但也迟迟打不开企业市场。作者认为造成这个结果的原因是，起步时过度依赖 Amazon EC2 模式，没有正视市场需求，以及没有按用户场景合理使用新技术。

上述几个 IaaS 软件在起步的初期，都没有什么前人的经验可以借鉴。没有人知道设计这么一个分布式的集成系统，面临的挑战在哪里，都有哪些坑要填。于是所有 IaaS 软件都处于一个在功能上模仿 Amazon EC2、在架构上自然生长的状态。由于当时市场空白较大，各家都急于上马 IaaS 项目，开发的推动力在于快速实现 EC2 的功能，在架构方面思考较少，为未来不同需求留下的架构冗余也不多。当整个 IaaS 软件发展到中后期，开发者们试图根据传统企业的真实需求进行架构变更时，才发现架构重构已经难以进行。

Eucalyptus 因为全面模仿 Amazon EC2 模式，其方案一直难以定制；而 CloudStack 虽然针对传统企业的差异需求进行了变更的尝试，但结果却是新版本软件越来越不稳定。OpenStack 则干脆主动忽视这一市场需求；OpenStack 社区本身就有强烈的声音拒绝传统应用带来的需求。例如《Keep OpenStack Weird》一文就呼吁 OpenStack 拒绝为传统应用改变，而要求企业必须进化以开发出适应 OpenStack 的应用。Gartner 在文章《Why vendors can't sell OpenStack to enterprises》就明确指出拒绝拥抱传统应用生态是 OpenStack 无法打开企业市场的一个重要原因。

IaaS 在公有云的蓬勃发展带动了相关领域的创新，SDN（软件定义网络）、SDS（软件定义存储）应运而生。的确，这些新技术的出现都是为了解决传统技术所面临的一些问题，但这些问题往往在公有云和大规模数据中心比较明显。例如公有云对网络隔离的需求，要求隔离技术能突破传统 VLAN 最多 4096 个的限制；存储技术要求能够实现分布式、多写备份以及动态扩容等。对于传统企业来说，这些技术往往既没必要也不稳定。当前一些私有云厂商，在向客户推荐解决方案时过分推销新技术，例如明明客户只需要传统的扁平网络，却硬要客户部署 SDN；明明客户对存储动态扩容没有需求，使用传统的 NFS + RAID 备份就已足够，却推荐客户部署分布式存储。客户项目上马后，在日常使用中长期遇到不稳定、难维护的问题，渐渐对私有云失去信心。OpenStack 社区蓬勃发展后涌入的大量硬件厂商，从自身利益出发提交了很多兼容性不好的代码，造成 OpenStack 某些核心组件不稳定（例如 Neutron），也在一定程度上加剧了客户对私有云的不信任。

私有云未来应该怎么做

对于未来私有云的发展，作者认为关键在于：开发出能拥抱传统应用生态的商业模式，专注于用户场景，控制软件投入风险，向上融合走整体解决方案路线。

1、开发出能拥抱传统应用生态的商业模式

要想打开传统企业市场，私有云厂商要正视已是既成事实的传统应用生态。虽然未来的企业应用必将进化成亲和现代 IaaS 架构的云应用，但在此之前，我们不得不兼顾企业已在传统应用上的投入。兼顾的方法是在虚拟化方面学习 VMWare，在网络模式和上层服务向 Amazon 靠拢。VMWare 凭借多年来在企业虚拟化方面的积累，开发出了很多满足传统应用需求的功能，例如在管理上以虚拟机为中心、虚拟机高可靠、专有资源配置等。私有云提供商的 IaaS 软件要么开发出类似功能，要么直接集成 VMWare 作为虚拟化解决方案。在网络方面，Amazon 的很多网络功能已经成为云计算领域的事实标准，例如 VPC、EIP、Security Group、ELB 等都广受用户欢迎。私有云 IaaS 软件要将这些服务进行整合，

以实现传统应用的各个网络场景。此外，由于全世界的公有云几乎都遵循 Amazon EC2 模式，在网络上靠拢 Amazon 能够很容易帮助客户搭建混合云（hybrid cloud），打通用户在公有云和私有云中的业务。最后，Amazon 很多 IaaS 之上的功能例如 Auto scaling、CloudFormation、CloudWatch 等都是解决用户痛点的创新，私有云 IaaS 需要能够提供类似的功能。

2、专注于用户场景

专注于用户场景要求私有云提供商从用户的应用场景出发，推荐符合用户需求的解决方案，一切以稳定优先。以网络为例，无论是 SDN 还是传统技术，最后面对的还是 OSI 的 7 层模型。在 IaaS 层面就是 L2 隔离，L3 子网加路由，L4~L7 应用层协议，应用场景是非常明确的。对于企业用户，可能只需要数目有限的私有网络，那我们就可以推荐用传统的 VLAN 来进行隔离；对于一些服务提供商，要求网络拓扑能够灵活按需变化，那我们就可以推荐使用 SDN 来实现路由。又例如存储，企业用户的场景往往是容量可以预估，稳定性可靠性要求高，那我们就可以推荐成熟的、基于传统协议的商业存储；如果面对的是服务提供商，数据量可能会爆炸增长的，那我们就应该推出类似于 Ceph 这样的分布式存储。

3、控制软件投入风险

IaaS 软件是集成技术，必然会引来各种第三方厂商要求进行集成。私有云提供商要对自家 IaaS 软件在这方面的投入进行控制，不能沦为第三方厂商的集成器，把主要精力浪费在下层系统的集成中。愿意部署私有云的企业，往往都是有预算购买新硬件产品的。私有云提供商应该优先专注于集成最稳定、应用广泛的成熟产品，在向客户提供解决方案时，只推荐已验证过的第三方产品。并且要借助客户的力量，反推第三方厂商与自身集成。例如 CloudStack 中的很多硬件驱动就是其客户要求网络和存储厂商为 CloudStack 提供的。此外，要认识到在虚拟化、网络以及存储方面已经存在非常多的标准技术，而对标准技术的稳定集成已能够满足绝大多数企业客户的需求。

4、向上融合走整体解决方案路线

私有云的未来一定是向上层发展，实现 IaaS 和 PaaS 的融合。私有云的目的是为了将用户从基础架构中解脱出来，专注于业务创新。但在基础架构（IaaS）跟最终业务（application 或 SaaS）之间还有一层隔离，即软件的部署交付，目前人们称之为平台即服务（PaaS）。未来的私有云展现给用户的一定是一个最终平台，即不仅能够管理用户的基础架构，还能为用户部署和管理上层业务。我们不妨将 IaaS 和 PaaS 融合的平台看作是

私有云的一个未来形态，如果说操作系统是传统软件的入口的话，那么未来企业软件的入口就是私有云。用户部署软件的方式将不再是到一个个操作系统里手动安装配置，而是软件商提交给用户的就是包含软件本身的操作系统镜像，通过支持私有云定义的接口，可以在镜像启动时导入下层基础设施信息，例如网络地址、网络磁盘路径以及软件配置信息，并且能在运行的过程中根据自身状态将数据反馈给 IaaS，实现基础设施层面的调整（例如启动更多的虚拟机）。目前已经有一些产品和项目在向这个方向努力，但都停留在跟 IaaS 第三方集成的层面上。由于 IaaS 本身不提供原生支持以及成熟度的问题，目前还没有好的解决方案。

总结

私有云要打开企业市场，不能再走早期单纯复制公有云的模式，而是要拥抱传统应用生态，走一条兼顾现实、着眼未来的道路，并最终向上发展，为企业 IT 架构提供一套完整的解决方案。

参考文章

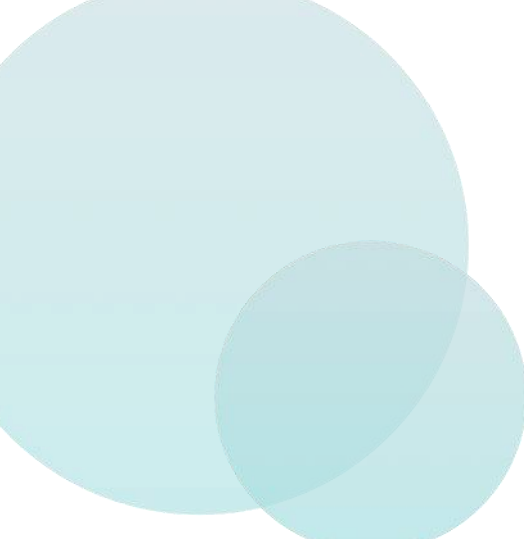
《Why OpenStack is different from other open source projects》

《Keep OpenStack Weird》

《Why vendors can't sell OpenStack to enterprises》

作者简介

张鑫，2006 年加入 Intel 上海开源技术中心（OTC），从事开源虚拟机项目 XEN 的开发，为社区共享了多个功能，例如 XEN 中 E100 网卡模拟器，XEN/IA64 虚拟 BIOS 对 Windows 的支持等。同时也共享了大量 bug 修复的补丁。2010 年赴硅谷加入 Cloud.com（后被 Citrix 收购），从事 CloudStack 的开发工作，其间多次作为 CloudStack 代表参与客户私有云项目的设计和部署。在从 Citrix 退出后，和搭档一起创立 ZStack。



分布式系统的特点 以及设计理念

作者 王璞

分布式系统并不是什么新鲜词，在上个世纪七八十年代就已经有各种分布式系统出现。只是在互联网时代，分布式系统才大放异彩，尤其是 Google 更是把分布式系统运用到了极致。Google 整个的软件构架都是基于各种各样的分布式系统，诸如 Borg、MapReduce、BigTable 等。正是这些分布式系统，使得 Google 可以处理高并发请求响应以及海量数据处理等。Apache 旗下的 Hadoop、Spark、Mesos 等分布式系统，把大数据处理相关技术变得非常亲民，让更多企业客户体会到了分布式系统的便利。

一、分布式系统的特点

分布式系统最大的特点是可扩展性，它能够适应需求变化而扩展。企业级应用需求经常随时间而不断变化，这也对企业级应用平台提出了很高的要求。企业级应用平台必须要能适应需求的变化，即具有可扩展性。比如移动互联网 2C 应用，随着互联网企业的业务规模不断增大，业务变得越来越复杂，并发用户请求越来越多，要处理的数据也越来越多，这个时候企业级应用平台必须能够适应这些变化，支持高并发访问和海量数据处理。分布式系统有良好的可扩展性，可以通过增加服务器数量来增强分布式系统整体的处理能力，以应对企业的业务增长带来的计算需求。

分布式系统的核心理念是让多台服务器协同工作，完成单台服务器无法处理的任务，尤其是高并发或者大数据量的任务。**分布式系统由独立的服务器通过网络松散耦合组成的。**每个服务器都是一台独立的 PC 机，服务器之间通过内部网络连接，内部网络速度一般比较快。因为分布式集群里的服务器是通过内部网络松散耦合，各节点之间的通讯有一定的网络开销，因此分布式系统在设计上尽可能减少节点间通讯。此外，因为网络传输瓶颈，单个节点的性能高低对分布式系统整体性能影响不大。比如，对分布式应用来说，采用不同编程语言开发带来的单个应用服务的性能差异，跟网络开销比起来都可以忽略不计。因此，分布式系统每个节点一般不采用高性能的服务器，而是性能相对一般的普通 PC 服务器。

提升分布式系统的**整体性能是要通过横向扩展**（增加更多的服务器），而不是纵向扩展（提升每个节点的服务器性能）。

分布式系统最大的特点是廉价高效：由成本低廉的 PC 服务器组成的集群，在性能方面能够达到或超越大型机的处理性能，在成本上远低于大型机。这也是分布式系统最吸引人之处。成本低廉的 PC 服务器在硬件可靠性方面比大型机相去甚远，于是分布式系统由软件来对硬件进行容错，通过软件来保证整体系统的高可靠性。

分布式系统最大的好处是实现企业应用服务层面的弹性扩展。应用服务层面的弹性扩展是相对计算资源层面的弹性扩展而言的。一般公有云服务（IaaS）厂商都会提供计算资源层面的弹性扩展，比如可以很方便地增加或删除虚拟主机、提升或降低虚拟主机的性能配置等等。但是企业客户真正需要的是应用服务层面的弹性扩展，即随着业务量的涨落，后台应用服务的实例能动态变化，这是 IaaS 厂商还做不到的。比如，某移动互联网短视频分享应用，在晚间 11 点到凌晨 1 点是访问高峰，同时在线人数高达几十万，这时后台应用服务要扩张到数千个实例才能应付这么高并发的访问请求；过了高峰时段，后台应用服务可以收缩到几十个实例。有了分布式系统，就可以很方便地调度应用服务实例，从几十个到几百个甚至上千个，真正实现应用服务的弹性扩展。

二、分布式系统设计理念

上面简单介绍了分布式系统的基本情况，下面详细阐述笔者理解的几个分布式系统设计理念：

1. 分布式系统对服务器硬件要求很低

这一点主要现在如下两个方面：

对服务器硬件可靠性不做要求，允许服务器硬件发生故障，硬件的故障由软件来容错。所以分布式系统的高可靠性是由软件来保证。

对服务器的性能不做要求，不要求使用高频 CPU、大容量内存、高性能存储等等。因为分布式系统的性能瓶颈在于节点间通讯带来的网络开销，单台服务器硬件性能再好，也要等待网络 IO。

一般而言，互联网公司的大型数据中心都是选用大量廉价的 PC 服务器而不是用几台高性能服务器搭建分布式集群，以此来降低数据中心成本。比如，Google 对于数据中心的成本控制做到了极致：所有服务器一律不要机箱；主板完全定制，只要最基本的组件，

早期的定制主板连电源开关和 USB 接口都不要；在主板上加装隔离带把 CPU 单独隔出来，让冷风只吹 CPU，不吹内存、硬盘等不需要降温的组件，最大限度降低冷却电力消耗。

2. 分布式系统强调横向可扩展性

横向可扩展性 (Scale Out) 是指通过增加服务器数量来提升集群整体性能。纵向可扩展性 (Scale Up) 是指提升每台服务器性能进而提升集群整体性能。纵向可扩展性的上限非常明显，单台服务器的性能不可能无限提升，而且跟服务器性能相比，网络开销才是分布式系统最大的瓶颈。横向可扩展性的上限空间比较大，集群总能很方便地增加服务器。而且分布式系统会尽可能保证横向扩展带来集群整体性能的（准）线性提升。比如有 10 台服务器组成的集群，横向扩展为 100 台同样服务器的集群，那么整体分布式系统性能会提升为接近原来的 10 倍。

互联网公司的数据中心，一般一个分布式系统横向扩展的上限在万台服务器左右。Google 数据中心的基本单元，CELL，由两万台左右服务器组成，每个 CELL 由一套分布式管理系统，BORG，统一管理，每个数据中心都由多个 CELL 组成。

3. 分布式系统不允许单点失效

单点失效 (Single Point Failure) 是指，某个应用服务只有一份实例运行在某一台服务器上，这台服务器一旦挂掉，那么这个应用服务必然也受影响而挂掉，导致整个服务不可用。例如，某网站后台如果只在某一台服务器上运行一份，那这台服务器一旦宕机，该网站服务必然受影响而不可用。再比如，如果所有数据都存在某一台服务器上，那一旦这台服务器坏了，所有数据都不可访问。

因为分布式系统的服务器都是廉价的 PC 服务器，硬件不能保证 100%可靠，所以分布式系统默认每台服务器随时都可能发生故障挂掉。同时分布式系统必须要提供高可靠服务，不允许出现单点失效，因此分布式系统里运行的每个应用服务都有多个运行实例跑在多个节点上，每个数据点都有多个备份存在不同的节点上。这样一来，多个节点同时发生故障，导致某个应用服务的所有实例都挂掉、或某个数据点的多个备份都不可读的概率大大降低，进而有效防止单点失效。

通常情况，不要让服务器满负荷运行，服务器长时间满负荷运行的话，出故障的概率显著升高。所以分布式系统采用一大堆中低性能的 PC 服务器，尽可能把负载均摊到所有服务器上，让每台服务器的负载都不高，保证集群整体稳定性。

4. 分布式系统尽可能减少节点间通讯开销

如前所述，分布式系统的整体性能瓶颈在于内部网络开销。目前网络传输的速度还赶不上 CPU 读取内存或硬盘的速度，所以减少网络通讯开销，让 CPU 尽可能处理内存的数据或本地硬盘的数据，能显著提高分布式系统的性能。典型的例子就是 Hadoop MapReduce，把计算任务分配到要处理的数据所在的节点上运行，从而避免在网络上传输数据。

5. 分布式系统应用服务最好做成无状态的

应用服务的状态是指运行时程序因为处理服务请求而存在内存的数据。**分布式应用服务最好是设计成无状态**。因为如果应用程序是有状态的，那么一旦服务器宕机就会使得应用服务程序受影响而挂掉，那存在内存的数据也就丢失了，这显然不是高可靠的服务。把应用服务设计成无状态的，让程序把需要保存的数据都保存在专门的存储上，这样应用服务程序可以任意重启而不丢失数据，方便分布式系统在服务器宕机后恢复应用服务。

比如，在设计网站后台的时候，对于用户登陆请求，可以把登陆用户的 session 相关信息保存在 Redis 或 Memcache 等缓存服务中，这样每个网站的后台实例不保存用户登录状态，这样即使重启网站后台程序也不丢失用户的登录状态信息；如果把用户的 session 相关信息保存在网站后台程序的内存里，那一旦受理用户登录的网站后台程序实例挂掉，必然有用户的登录状态信息会丢失。

总而言之，分布式系统是大数据时代企业级应用的首选平台，它有良好的可扩展性，尤其是横向可扩展性（Scale Out），使得分布式系统非常灵活，能应对千变万化的企业级需求，而且降低了企业客户对服务器硬件的要求，真正能做到应用服务层面的弹性扩展（auto-scaling）。

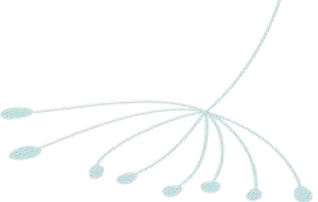
作者介绍

王璞，2002 年北京航空航天大学获得力学学士学位，2007 年北京大学获得计算机硕士，2011 年美国 George Mason University 获得计算机博士学位，研究方向机器学习，发表十余篇机器学习以及数据挖掘相关论文。毕业后在硅谷先后供职 StumbleUpon，Groupon 和 Google 三家公司。专长海量数据处理，分布式计算，以及大规模机器学习。于 2014 年回国创办数人科技，专注于为企业客户提供大数据分析处理一站式解决方案。



生态圈新闻

云生态专刊



Microsoft Azure 位居 Nasuni 存储基准测试之首

作者 Janakiram MSV , 译者 谢丽

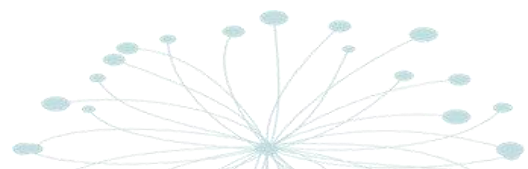
云 NAS 与存储公司 Nasuni 发布了年度云存储基准测试的结果。Microsoft Azure Storage 在速度、可用性和扩展性方面胜出。基准测试涵盖的其它服务还包括 Amazon S3 和 Google Cloud Storage。

Nasuni 年度报告名为[云存储现状](#)，包含了该公司针对一组指标对主流云存储服务进行基准测试的结果。今年，Nasuni 将测试范围限制在三大领先的服务提供商——Microsoft Azure Storage、Amazon Simple Storage Service (S3) 和 Google Cloud Storage。虽然该公司也考虑过 IBM SoftLayer 和 HP Cloud Storage，但由于计划停机和供应商战略方向变化，他们决定放弃对两者的测试。

该基准测试测量每个云存储服务的吞吐量、可用性和可扩展性指标。速度测试测量每个存储服务提供商处理大量写入、读取和删除操作的原始能力。Nasuni 使用不同大小的文件以不同水平的并发测试了每项服务。可用性测试持续超过 30 天，测量每个服务在 60 秒的间隔内单次读取、写入和删除操作的响应时间。可扩展性测试测量每个提供商随着管理对象数量增加提供稳定服务的能力。

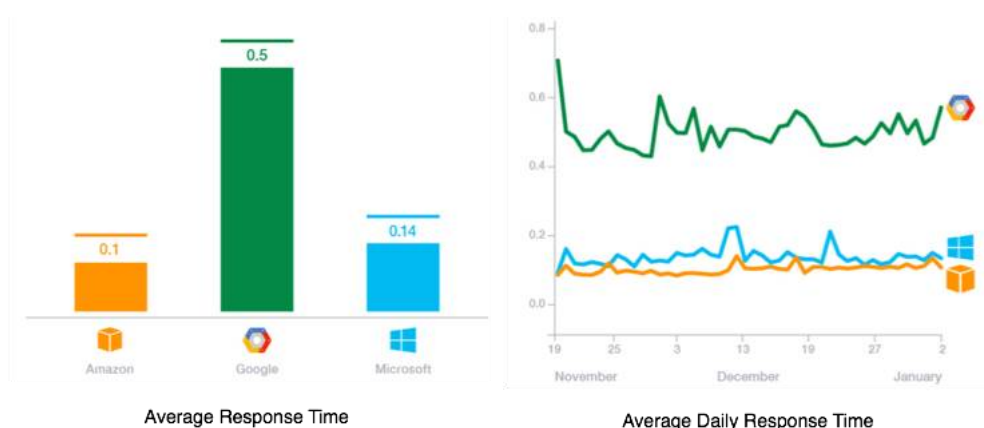
Nasuni 的方法包括使用配置相同的 VM 在多个区域运行基准测试。按照设计，该基准测试用于评估存储服务提供商在文件服务器数据负载下的性能。每个文件数据集中文件大小的分布与 Nasuni 企业用户实际使用的文件大小分布一致。可用性测试运行在 Rackspace 中配置的单个 VM 上。“响应时间”指标用于测量每个提供商在 60 秒的间隔内执行单次写入、读取和删除操作的响应时间。扩展性测试的测试方式是使用内部机器在最短的时间内达到最高可扩展数量。

在性能基准测试中，Microsoft Azure Storage 胜出。不过，Amazon S3 发起了激烈的竞争，并在包含大文件测试中超过了 Azure。

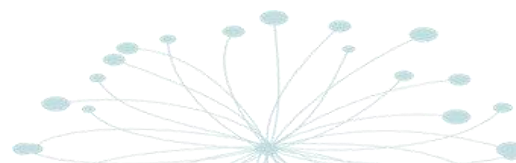
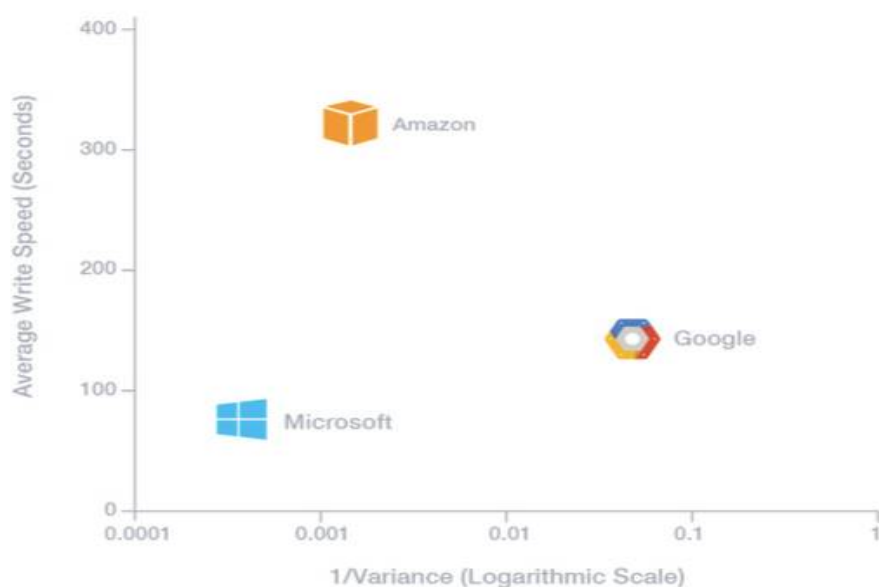




第二项测试与可用性有关，Amazon 和 Microsoft 并列。Google Cloud Storage 垫底，平均响应时间为 0.5 秒，S3 和 Azure Storage 的速度几乎是其 5 倍。

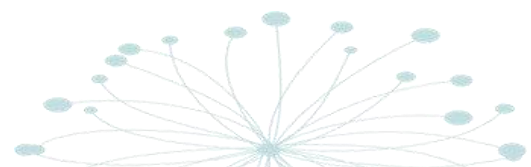


第三项测试测量可扩展性。该项测试测量每个服务的写入速度变化、写入丢失数量、读取丢失数量。测试结果表明，三大提供商之间的差别并不大，Google 和 Microsoft 写入错误数为 0，而 Amazon 有 5 个写入错误，平均错误率为百分之 0.000005。Microsoft Azure Storage 速度方差最大，但它展示了第二高的速度和零读取错误与零写入错误的完美记录。



根据 Nasuni 基准测试的最终结果，Microsoft 是公共云存储领域的顶级云提供商。对于中小文件，它提供了最快的速度，在某些情况下，超过 Amazon 近两倍。对于大文件，Amazon 显示出了更好的读取和写入性能，但 Microsoft 的删除操作速度超过 Amazon。

这是 Microsoft Azure 连续第二次位居 Nasuni 基准测试之首。





Twitter 已经用 Heron 替换了 Storm

作者 **Abel Avram**，译者 **谢丽**

Twitter 已经用 Heron 替换了 Storm。此举将吞吐量最高提升了 14 倍，单词计数拓扑时间延迟最低降到了原来的 1/10，所需的硬件减少了 2/3。

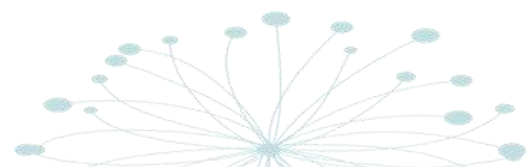
Twitter 使用 [Storm](#) 实时分析海量数据已经有好几年了，并在 2011 年将其开源。该项目稍后开始在 Apache 基金会孵化，并在去年秋天成为顶级项目。Storm 以季度为发布周期，现在已经达到了 0.9.5 版本，并且正在向着人们期望的 1.0 稳定版前进。但一直以来，Twitter 都在致力于开发替代方案 Heron，因为 Storm 无法满足他们的实时处理需求。

[Twitter 的新实时处理需求](#)包括：“每分钟数十亿的事件；大规模处理具有次秒级延迟和可预见的行为；在故障情况下，具有很高的数据准确性；具有很好的弹性，可以应对临时流量峰值和管道阻塞；易于调试；易于在共享基础设施中部署。” [Karthik Ramasamy](#) 是 Twitter Storm/Heron 团队的负责人。据他介绍，为满足这些需求，他们已经考虑了多个选项：增强 Storm、使用一种不同的开源解决方案或者创建一个新的解决方案。增强 Storm 需要花费很长时间，也没有其它的系统能够满足他们在扩展性、吞吐量和延迟方面的需求。而且，其它系统也不兼容 Storm 的 API，需要重写所有拓扑。所以，最终的决定是创建 Heron，但保持其外部接口与 Storm 的接口兼容。

拓扑部署在一个 [Aurora](#) 调度器上，而后者将它们作为一个由多个容器（cgroups）组成的任务来执行：一个 Topology Master、一个 Stream Manager、一个 Metrics Manager（用于性能监控）和多个 Heron 实例（spouts 和 bolts）。拓扑的元数据保存在 ZooKeeper 中。处理流程通过一种反压机制实现调整，从而控制流经拓扑的数据量。除 Aurora 外，Heron 还可以使用其它服务调度器，如 YARN 或 Mesos。实例运行用户编写的 Java 代码，每个实例一个 JVM。Heron 通过协议缓冲处理彼此间的通信，一台机器上可以有多个容器。（要了解更多关于 Heron 内部架构的细节信息，请阅读论文“[Twitter Heron：大规模流处理](#)”。）

Twitter 已经用 Heron 完全替换了 Storm。前者现在每天处理“数 10TB 的数据，生成数 10 亿输出元组”，在一个标准的单词计数测试中，“吞吐量提升了 6 到 14 倍，元组延迟降低到了原来的五到十分之一”，硬件减少了 2/3。

当被问到 Twitter 是否会开源 Heron 时，Ramasamy 说“在短时间内不会，但长期来看可能。”





Windows 10：微软的十亿设备野心

作者 徐川

6月5日，微软在北京举办了 [Build Tour Beijing](#) 活动，这是微软在全球 23 场宣讲活动之一，此活动面向全球开发者，以更接地气的方式进一步推广微软的新技术和理念。在活动中，微软总部 TED 工程部门资深产品经理 Shen Chauhan 向开发者展示了 UWP 平台应用等技术，会后和微软大中华区开发体验和平台合作事业部总经理 Srikanth Raju 等一起接受了记者的采访。微软全球副总裁兼首席布道师 Steven Guggs 也在 9 号参加了媒体的 FAQ 环节。

在活动与采访中，有一个数字反复出现，那就是 1 Billion——十亿。这个数字是微软希望搭载 Windows 10 的设备在两三年内达到的数量。为了达到这个目标，微软从战略和策略上都做了准备：

PC 免费升级策略：前段时间有消息称，微软将为盗版 Windows 系统提供免费升级，对于正版升级用户会提供一年的免费使用时间。全球的 PC 用户有数十亿之多，如果其中 50% 能在 3 年内升级 Win10，那么要达成目标并不困难。根据[数据统计](#)，Windows 7 占有率达到 50% 花了约 7 年时间，要想 Windows 10 只花 3 年达到相同程度，必须采用更加激进的策略。

单一平台战略：搭载 Windows 10 的设备不仅仅只有 PC，Windows 10 将是横跨 IOT、移动、PC、Xbox、Hololens 等各种设备的共同操作系统，它们的内核相同，API 也相同，不同的是 UI 和非必要组件，如 Windows IOT 版本是没有 UI 的。将这些其它设备算进去的话，Windows 10 设备数量将会进一步上升。

Windows as a Service：这是微软在今年一月份的一次活动上提出的口号，它的其中一个隐藏含义则是 Windows 10 将是 Windows 的“最终版”，作为服务，Windows 10 将更容易被人们接触和获取，这也将加快 Windows 更新的速度。

Shen 和 Srikanth 认为，Windows 10 将在各种各样的设备上出现，对开发者和用户都将具有很大的吸引力。因此，10 亿是个非常现实的数字，他们对达到这一目标充满信心。



不过，要打造一个完整的生态体系，仅仅只有设备和操作系统是不够的，还需要应用和开发者的参与，微软也从种种方面采取手段来拉近开发者的距离：

拥抱开源社区：自去年以来，微软采用了相当大的力量拥抱开源，包括将 .NET 编译平台 Roslyn 以及开发环境、MSBuild 引擎、rDSN 分布式系统开发框架等纷纷开源，并且使用了流行的代码托管平台 Github。“开发者在哪里，我们就去哪里”，毫无疑问，这样的态度获得了开发者的认可。

支持跨平台开发：Visual Studio 自去年更新以来，开始支持 Android 和 iOS 应用的开发，对于开发者来说，多一个好用的 IDE 来进行开发调试无疑是令人高兴的，另一方面，微软发布了跨平台的开发工具 VS Code，微软这样开放的态度有助于将开发者吸引回 Windows 平台进行开发。

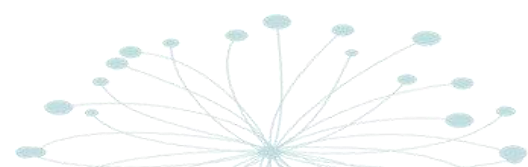
Universal Windows Platform (UWP)：UWP 是单一平台战略的产物，采用了 Adaptive UX 来适应不同的终端，开发者只需开发一次即可适配不同的设备，大大降低了开发成本，配合 10 亿设备的远景，足以对开发者产生强大的吸引力。

UWP Bridge 项目：目前移动领域由 Android 和 iOS 称雄，Windows Phone 则一直饱受缺乏应用的诟病，UWP Bridge 项目可以帮助开发者简单的将它们 Android、iOS、Win32、Web 项目转换到 Windows 10 平台上，如果这个项目成功的话，Windows 10 将不再缺乏应用。

“总之，我们要做的就是，先建立尽可能广泛的基础用户，然后帮助他们尽可能容易的在 Windows 10 系统上开展工作。我们在这个过程中提供了很多可用的代码和 API 套件，以此为开发者们带去更好的体验。”Steven Guggs 说道。

不过，虽然手段很多，但效果如何，还是要看开发者怎么说。Shen Chauhan 表示，微软非常重视开发者的意见，这次活动也邀请了一些合作伙伴现场演示他们为 Windows 10 开发的应用。新浪微博产品总监康金山、HiWiFi 联合创始人李恺、AE Mobile CEO 沈磊也向媒体分享了他们为 Windows 10 开发应用的感受。

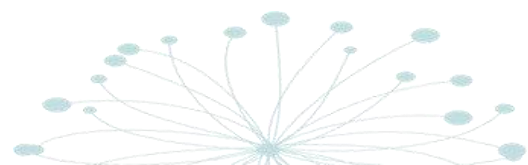
康金山分享了开发新浪微博 Windows 10 版的经历。他们主要使用了两个 Windows 10 的新特性：Responsive Design 和 App Services。利用 Responsive Design，原来新浪微博的桌面开发团队和 WP 客户端开发团队能够合并，而且新团队不到 10 人就能完成开发，大大节约了人力成本。



李恺表示，他们之前只开发了 Android 和 iOS 版本，这次为 Windows 10 开发 UWP 版本，最大的感受就是开发过程非常简单，只要一次开发，就可以快速把想要的效果做出来。

沈磊的团队专门面向国外 WP 用户开发手游，他们的应用下载次数已经超过 1 亿次，他表示 WP 市场并没有人们想象的那么小。这次他分享的对 Azure 的使用体验，Azure 的一些 SQL Server 每分钟自动备份特性以及较低的价格给他留下了深刻印象。

通过对 Windows 10 了解的越来越深入，可以确认这是微软近几年来最激进的举动，对于它能否赢得用户和开发者的心，且让我们拭目以待。





IBM Bluemix

下一代数字创新平台

用你想要的方式开发应用

