促进软件开发领域知识与创新的传播

架构师

ARCHITECT

推荐文章 | Article

DevOps的前世今生

微信序列化生成器架构设计及演变

Nginx日志中的金矿

观点 | Opinion

OpsDev将至

热点 | Hot

东方航空技术选型为何花落MongoDB

Geekbang 极客邦科技 | InfoQ

2016年10月

卷首语

CIO 和 CTO：一字之差 差了什么？

作者 **张晓楠**

早些年互联网发展没有这么快的时候，CIO 应该就是技术岗位上最牛的角色了。虽然很多 CIO 并非技术出身，但是并不阻碍他们在企业信息化建设上拥有计划、采购、管理、运维的职权。

而当互联网拥有了颠覆一切的力量时，技术的能量变得巨大无边，它支撑着企业的商业模式、核心产品、经营决策，企业需要的这个 IT 管理者，必须是属于战斗部门，而并非支持部门。于是乎拥有技术管理、洞察能力，身在研发一线的 CTO 脱颖而出。

面对 CTO 声望和身价的迅速飙升，很多 CIO 们也开始探求转型 CTO 之路，从字面上来看 CIO 和 CTO 似乎只有一字之差，但是实际上二者的差别还是非常大的。

CIO 大多身处传统行业，他们跟人打交道的时间要远远超过埋头钻研技术的时间，CIO 人群并不那么喜欢互相交流和学习，最多有个小团体时不时聚会一下，聚会内容大多数是喝酒扯闲篇。对于新的技术趋势和新的技能，他们接受度有限、学习能力不强。

反过来看 CTO 则是一群永远有危机感的人，一个原因是他们效力的企业大多是容易快速被更替或者淘汰的行业，危机感强；另外一个原因在于，对新技术越倚重，就越有敬畏心，技术发展速度太快，自己只有不断学习才不至于掉队。

CIO 和 CTO 这两个人群，应该说各有需要补足的地方。对于 CIO 来说，技术的生产能力是他们的短板，而技术的应用能力则是他们的强项，更何况他们还有对 IT 项目规划和运营的经验，不过这些 IT 项目更多侧重在应用成熟技术，这是由他们的知识结构决定的，也是他们所处行业的特点决定的。

反观 CTO 人群，技术能力强，但是管理能力弱。很多 CTO 动手写起程序就像打了鸡血，但是要让他们管理团队或者跨部门协作就头疼。口才也是 CTO 的短板，写得一手好代码，但一要演讲就蔫了。

很多人关注 CIO 和 CTO 的职业发展前景，这也是为什么很多 CIO 想要转行 CTO 的原因所在。其实关注个人职业发展固然重要，但是对你深处的企业和行业的发展趋势有敏锐的洞察和理性的判断更加重要，你的个人规划能力再强，如果对供职企业判断不足，就只剩下频频跳槽了，这不是我们想要看到的结果。

CONTENTS / 目录

架构师

2016 年 10 月刊

本期主编 徐　川

流程编辑 丁晓昀

发行人　霍泰稳

提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com

专访黄翀：东方航空技术选型为何花落 MongoDB

作者 韩婷

在今年的 MongoDB World W016 大会上，来自中国东方航空公司黄翀分享了使用 MongoDB 的实践经验，介绍了东航如何将 Spark 和 MongoDB 配合使用来解决所面临的问题，以及具体的步骤，一时引起热议。

东航信息部 PSS 部门副经理童帅华表示，选择 MongoDB，是为了打造东方航空新一代 Shopping 系统，希望后台数据存储除了要有关系型数据库所有查询功能，还想支持多变的数据模型以及结构。

那么，东方航空具体如何进行的技术选型？效果如何？为此，InfoQ 对东方航空相关项目的技术负责人黄翀进行了专访，对这些问题一一进行了解答。

InfoQ：东方航空开始使用 MongoDB，您提到，目前只在一个项目中进行了使用。能否详细介绍下这个项目的背景呢？该项目的配置如何？其数据库有哪些特点，所接受的访问量是什么数量级？

黄翀：该项目是东航 Shopping 项目，为用户提供个性化的航班搜索服务，支持多目的地搜索、基于预算范围的搜索、城市主题的搜索、灵感语义的搜索、实时的低价日历搜索等。同时，灵活组合中转路径，提高

OD 航线覆盖率。

上述的项目背景所带来的技术挑战是：一次前端搜索，所带来的后端的库存和运价搜索复杂度是原来的几十倍或者上百倍。因此，该项目对系统的性能要求更高。

该项目的配置：采用 3 台 64 核 128G memory ssd 硬盘。

生产环境： 3 台服务器组 MongoDB Replica-set 集群。

访问量： 目前推广上线了部分渠道，旅客的查询量每日 600 万次 +，转换成数据库的查询量每日 4500 万次 +。

数据库总数据条数： 720 亿条，每日更新次数 2600 万次 +，99% 以上查询效率低于 200ms。

InfoQ：为什么选择在该项目中用 MongoDB ？为了保证提供正常或更好的服务，东航从哪些方面进行了评估？具体的评估过程、数据、结果如何？

黄翀：该项目的特点是：高负载高并发。因此，我们在选型数据库时，主要参考技术标准如下：

- 支持基于内存查询的数据库，减少磁盘IO交互；
- 支持复杂多变数据存储结构与类型；
- 支持集群架构保证高可用；
- 支持复杂的SQL查询，例如基于预算，来搜索满足条件的航班；或者基于航班时刻（上下午），来搜索满足条件的航班等。

综合以上标准进行评估，最后我们选择了 MongoDB。

项目开始之初就依次评估过 Greenplum，Oracle，MySQL，MongoDB 等技术，有 RDBMS（ACID），有 Nosql（cap）；既考虑 key-value 型，也考虑要支持范围复杂搜索。既考虑到数据总量对性能影响，也考虑到我们是

查询性能要求高，写入性能要求不是不高但是次要。既考虑到实际开发难度学习曲线，也考虑到运维成本。

最后，我们又比较了性能、开发难易程度等诸多因素，最后选择了MongoDB。

InfoQ：Spark 和 MongoDB 的结合，是否完美匹配了东航的需求？在实时性方面，这样的结合有什么优势？在目前的使用过程中表现如何，有没有体现出优势或者暴露出不足？

黄翀： Spark 作为一个基于内存计算的引擎目前已进入东航的视野，我们进行了 POC 测试及使用场景验证，证明其对提高我们性能是有帮助的，下一步我们会考虑在生产上应用 Spark+MongoDB 的技术。

我们在运价计算场景验证了这项技术的优势。本来每次请求都要计算运价，现在我们将预先计算好的结果都存入 MongoDB，将计算性能问题转化为查询性能问题。基于我们在 MongoDB 在优化查询方面的经验，简化了问题本身。

优势显而易见，不足在于 Spark 的 map 方法不适应一些自编程 map，造成一些复杂中转程运价的计算难于在 Spark 中实现。

InfoQ：该项目在迁移、部署和维护方面是如何进行考虑的？成本是如何进行估算的？该项目目前是否已正式上线，投入使用？

黄翀： 目前使用的项目不存在迁移的过程，因为是新的项目选型了 MongoDB 数据库。目前来说 MongoDB 对东航还属于新事物，因此运维这块还依赖于项目组自己和 MongoDB 的原厂商一起来保障。

使用 MongoDB 的成本主要是购买了原厂的运维保障，因为目前使用 MongoDB 的 Shopping 项目是东航电商项目的核心组件，运维级别非常高，因此采购原厂来支持保障数据库的运维是非常必要的。

该项目已正式上线投产使用。

InfoQ：该项目的技术方案具体是如何落地的？有没有遇到哪些阻力？对于想要采取相似技术方案的企业来说，有哪些经验可以分享？

黄翀：早在 2014 年初，我们已经使用 MongoDB Version 2.8 将以往使用 Oracle 作为数据库的功能都实现了一遍，这个切换是非常方便快速的，许多代码可以复用。由于 MongoDB 是无模式的，可以不遵守范式，所以在设计 document 结构时，以及 chunk key 的选择时是有争论的，MongoDB 的咨询师给了我们比较大的帮助。

随着项目进行，MongoDB 自己也在更新换代，我们一起成长。同时发现我们上一个新应用，比以往至少要快 5 个人天。我相信在 MongoDB 在不需要数据库支持强事务的场景，是非常值得采用一种数据库。

MongoDB 是操作性功能完备数据存储方案，能覆盖众多场景。在云中部署，结合 Docker 等一众运维工具，会提高运维效率，节约运维成本。

InfoQ：针对东航开始使用 MongoDB，网上也有很多的评论，也有很多的误解，有支持者也有质疑者，您如何看待这些观点呢？假如该项目的方案表现令东航十分满意，在未来会推广到其他项目中去么？

黄翀：对我们来说，我们使用 MongoDB，不是属于哗众取宠，为了使用开源技术而使用，而是针对使用的场景进行了多轮的技术验证和多个数据库 POC 测试比较，选定了满足我们要求的产品，同时成本也能为我们所接受。因此这样的选型是对东航有帮助的，同时在互联网化的今天，东航这样的传统性企业在技术上也在不断的开放自己，开始拥抱互联网技术，在民航 IT 生态圈也占据一席之地。

我们在今后的项目中，针对适合使用 MongoDB 的业务场景，也会继续考虑使用 MongoDB。

DevOps 的前世今生

作者 木环

目前在国外，互联网巨头如 Google、Facebook、Amazon、LinkedIn、Netflix、Airbnb，传统软件公司如 Adobe、IBM、Microsoft、SAP 等，亦或是网络业务非核心企业如苹果、沃尔玛、索尼影视娱乐、星巴克等都在采用 DevOps 或提供相关支持产品。那么 DevOps 究竟是怎样一回事？在 Puppet、RightScale 分别 DevOps 出版的调查报告基础上，整理本文，以期为读者理清思路。另外，中国正在开展了一份自己的调查问卷，由南京大学发起，欢迎大家投票参与。

DevOps 是什么？从哪里来？

DevOps 的概念

DevOps 一词的来自于 Development 和 Operations 的组合，突出重视软件开发人员和运维人员的沟通合作，通过自动化流程来使得软件构建、测试、发布更加快捷、频繁和可靠。

DevOps 概念早先升温于 2009 年的欧洲，因传统模式的运维之痛而生。

DevOps 是为了填补开发端和运维端之间的信息鸿沟，改善团队之间的协作关系。不过需要澄清的一点是，从开发到运维，中间还有测试环节。

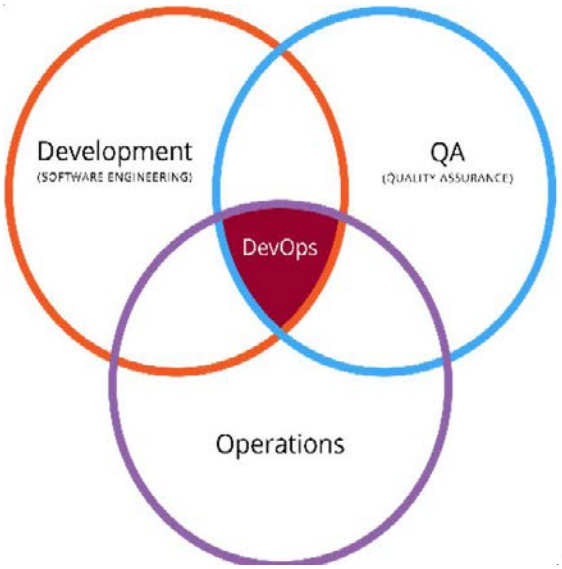


图 1

DevOps 其实包含了三个部分：开发、测试和运维（见图 1）。

换句话说，DevOps 希望做到的是软件产品交付过程中 IT 工具链的打通，使得各个团队减少时间损耗，更加高效地协同工作。专家们总结出了下面这个DevOps 能力图，良好的闭环可以大大增加整体的产出（见图 2）。

历史变革

由上所述，相信大家对 DevOps 有了一定的了解。但是除了触及工具链之外，作为文化和技术的方法论，DevOps 还需要公司在组织文化上的变革。回顾软件行业的研发模式，可以发现大致有三个阶段：瀑布式开发、敏捷开发、DevOps。

DevOps 早在九年前就有人提出来，但是，为什么这两年才开始受到越来越多的企业重视和实践呢？因为 DevOps 的发展是独木不成林的，现在有越来越多的技术支撑。微服务架构理念、容器技术使得 DevOps 的实施变得更加容易，计算能力提升和云环境的发展使得快速开发的产品可以立刻获得更广泛的使用（见图 3）。

DevOps 的几个关键问题

好处是什么？

DevOps 的一个巨大好处就是可以高效交付，这也正好是它的初衷。Puppet 和 DevOps Research and Assessment (DORA) 主办了 2016 年

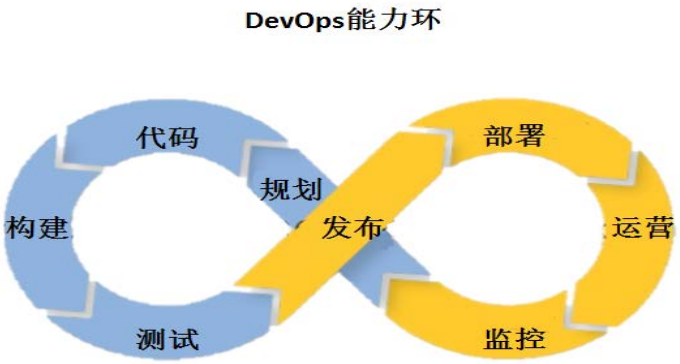


图 2

DevOps 调查报告，根据全球 4600 位各 IT 公司的技术工作者的提交数据统计，得出高效公司平均每年可以完成 1460 次部署。与低效组织相比，高效组织的部署频繁 200 倍，产品投入使用速度快 2555 倍，服务恢复速度快 24 倍。在工作内容的时间分配上，低效者要多花 22% 的时间用在为规划好或者重复工作上，而高效者却可以多花 29% 的时间用在新的工作上。所以这里的高效不仅仅指公司产出的效率提高，还指员工的工作质量得到提升。

DevOps 另外一个好处就是会改善公司组织文化、提高员工的参与感。员工们变得更高效，也更有满足和成就感；调查显示高效员工的雇员净推荐值（eNPS:employee Net Promoter Score）更高，即对公司更加认同。

快速部署同时提高IT稳定性。这难道不矛盾吗？

快速的部署其实可以帮助更快地发现问题，产品被更快地交付到用户手中，团队可以更快地得到用户的反馈，从而进行更快地响应。而且，DevOps 小步快跑的形式带来的变化是比较小的，出现问题的偏差每次都不会太大，修复起来也会相对容易一些（见图 4）。

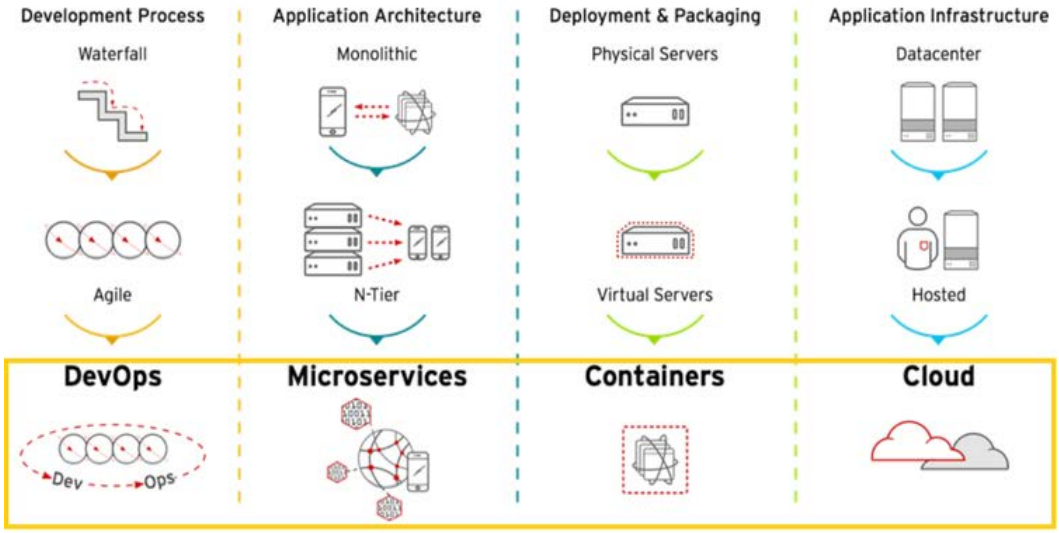


图 3

因此，认为速度就意味着危险是一种偏见。此外，滞后软件服务的发布也并不一定会完全地避免问题，在竞争日益激烈的 IT 行业，这反而可能错失了软件的发布时机。

为什么DevOps会兴起？为什么会继续火下去？

条件成熟：技术配套发展

技术的发展使得 DevOps 有了更多的配合。早期时，大家虽然意识到了这个问题的，但是苦于当时没有完善丰富的技术工具，是一种“理想很丰满，但是现实很骨感”的情况。DevOps 的实现可以基于新兴的容器技术；也可以在自动化运维工具 Puppet、SaltStack、Ansible 之后的延伸；还可以构建在传统的 Cloud Foundry、OpenShift 等 PaaS 厂商之上。

来自市场的外部需求：这世界变化太快

IT 行业已经越来越与市场的经济发展紧密挂钩，专家们认为 IT 将会有支持中心变成利润驱动中心。事实上，这个变化已经开始了，这不仅体现在 Google、苹果这些大企业中，而且也发生在传统行业中，比如出租车业务中的 Uber、酒店连锁行业中的 Airbnb、图书经销商 Amazon 等等。

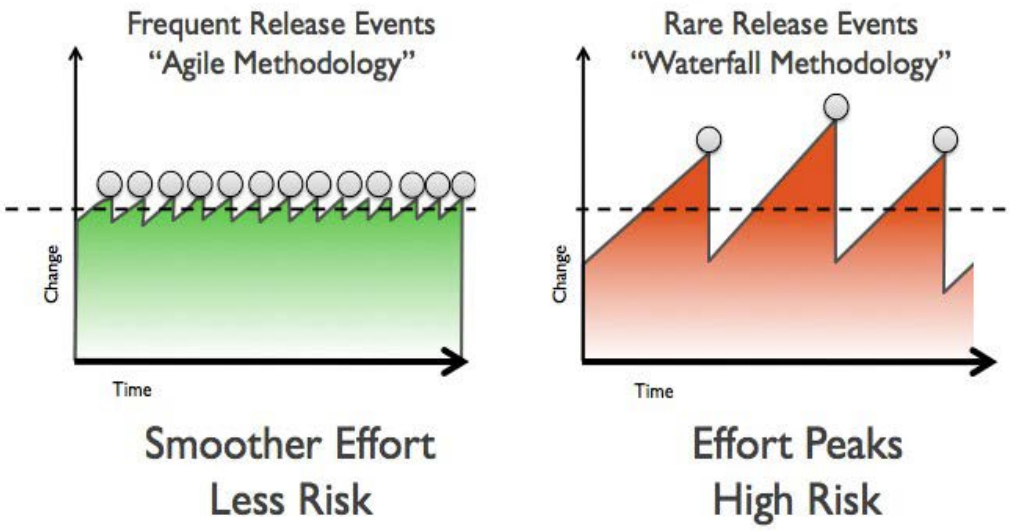


图 4

能否让公司的 IT 配套方案及时跟上市场需求的步伐，在今天显得至关重要。

DevOps 2016 年度报告给出了一个运维成本的计算公式：

停机费用成本 = 部署频率 * 版本迭代失败概率 * 平均修复时间 * 断电的金钱损失

来自团队的内在动力：工程师也需要

对于工程师而言，他们也是 DevOps 的受益者。微软资深工程师 Scott Hanselman 说过“对于开发者而言，最有力的工具就是自动化工具”（The most powerful tool we have as developers is automation）。工具链的打通使得开发者们在交付软件时可以完成生产环境的构建、测试和运行；正如 Amazon 的 VP 兼 CTO Werner Vogels 那句让人印象深刻的话：“谁开发谁运行”。（You build it, you run it）

实现DevOps需要什么?

硬性要求：工具上的准备

上文提到了工具链的打通，那么工具自然就需要做好准备。现将工具

类型及对应的不完全列举整理如下：

- 代码管理（SCM）：GitHub、GitLab、BitBucket、SubVersion
- 构建工具：Ant、Gradle、maven
- 自动部署：Capistrano、CodeDeploy
- 持续集成（CI）：Bamboo、Hudson、Jenkins
- 配置管理：Ansible、Chef、Puppet、SaltStack、ScriptRock GuardRail
- 容器：Docker、LXC、第三方厂商如AWS
- 编排：Kubernetes、Core、Apache Mesos、DC/OS
- 服务注册与发现：Zookeeper、etcd、Consul
- 脚本语言：python、ruby、shell
- 日志管理：ELK、Logentries
- 系统监控：Datadog、Graphite、Icinga、Nagios
- 性能监控：AppDynamics、New Relic、Splunk
- 压力测试：JMeter、Blaze Meter、loader.io
- 预警：PagerDuty、pingdom、厂商自带如AWS SNS
- HTTP加速器：Varnish
- 消息总线：ActiveMQ、SQS
- 应用服务器：Tomcat、JBoss
- Web服务器：Apache、Nginx、IIS
- 数据库：MySQL、Oracle、PostgreSQL等关系型数据库；cassandra、mongoDB、redis等NoSQL数据库
- 项目管理（PM）：Jira、Asana、Taiga、Trello、Basecamp、Pivotal Tracker

在工具的选择上，需要结合公司业务需求和技术团队情况而定。

（注：更多关于工具的详细介绍可以参见此文：51 Best DevOps Tools for #DevOps Engineers）

软性需求：文化和人

DevOps 成功与否，公司组织是否利于协作是关键。开发人员和运维人员可以良好沟通互相学习，从而拥有高生产力。并且协作也存在于业务人员与开发人员之间。出席了 2016 年伦敦企业级 DevOps 峰会的 ITV 公司在 2012 年就开始落地 DevOps，其通用平台主管 Clark 在接受了 InfoQ 的采访，在谈及成功时表示，业务人员非常清楚他们希望在最小化可行产品中实现什么，工程师们就按需交付，不做多余工作。这样，工程师们使用通用的平台（即打通的工具链）得到更好的一致性和更高的质量。此外，DevOps 对工程师个人的要求也提高了，很多专家也认为招募到优秀的人才也是一个挑战。

DevOps 的采用现状

哪些公司在用

DevOps 正在增长，尤其是在大企业中：调查发现，DevOps 的接受度有了显著提高。74% 的受访者已经接受了 DevOps，而去年这一比例为 66%。目前，在 81% 的大企业开始接受 DevOps，中小企业的接受度仅为 70%。

那么具体而言都有些公司在采用 DevOps 呢？ Adobe、Amazon、Apple、Airbnb、Ebay、Etsy、Facebook、LinkedIn、Netflix、NASA、Starbucks、Target（泛欧实时全额自动清算系统）、Walmart、Sony 等等。

他们怎么实施的

首先，大企业正在自下而上接受 DevOps，其中业务单位或部门（31%）

以及项目和团队（29%）已经实施 DevOps。不过，只有 21% 的大企业在整个公司范围内采用了 DevOps。

其次，在工具层面上，DevOps 工具的用量大幅激增。Chef 和 Puppet 依然是最常用的 DevOps 工具，使用率均为 32%。Docker 是年增长率最快的工具，用量增长一倍以上。Ansible 的用量也有显著增加，使用率从 10% 翻倍至 20%（见图 5）。

并且调查还发现不到半数（43%）的公司在使用诸如 Chef、Puppet、Ansible 或 Salt 等配置工具；然而使用配置工具的公司更有可能同时使用多个工具。25% 的受访者使用两种或更多配置工具，只使用一种工具的比例为 18%。其中 Chef 和 Puppet 是最常用的组合：使用 Chef 的组织中有 67% 同时也使用 Puppet，类似的，使用 Puppet 的组织中也有 67% 同时使用了 Chef（见图 6）。

中国也在准备一份 DevOps 的报告

文中的统计数据来自于国外的 DevOps 调研报告。其中由 Puppet 发起

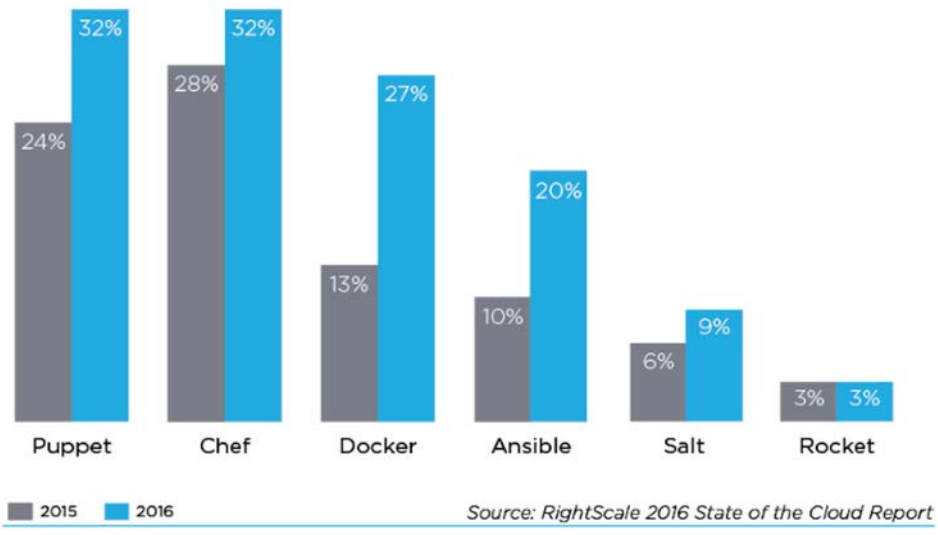


图 5

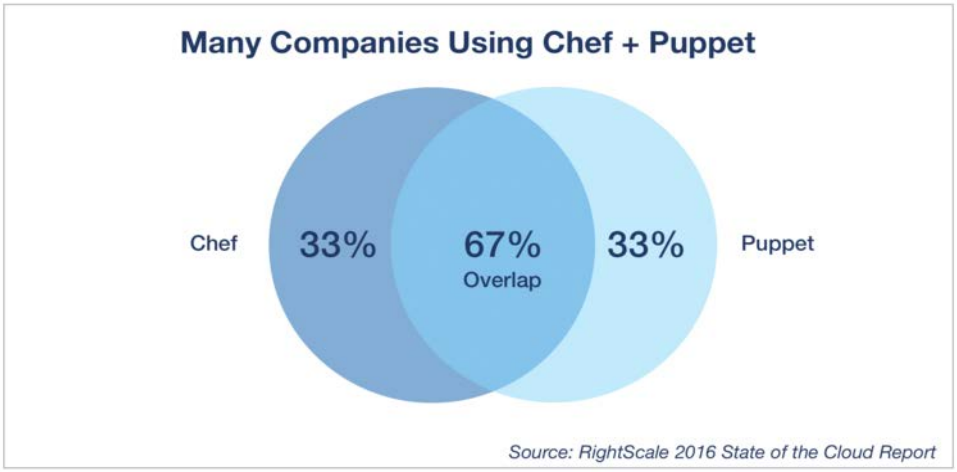


图 6

的 DevOps 年度国际调查报告已经连续出版五年，先后收集了 2.5 万技术人员的答卷；2016 年收集的有效答卷为 4600 份，不过仅有 10% 来自于亚洲。我们并不认为这样的采样率和采样数量可以充分地反映中国的 DevOps 行业现状。

目前，依托 DevOps 中国社区成员的积极参与支持，由南京大学发起开展《DevOps 中国 2016 年度调查》活动。期望通过本次问卷调查，收集 DevOps 的率先实践者们关于 DevOps 实践及经验的相关信息，从而了解 DevOps 在中国推广应用的状况，并汇总在 DevOps 实践中可能遇到的障碍、挑战和最佳实践，最终通过调查报告进一步促进 DevOps 在中国的认知和推广，同时为 DevOps 的每一位实践者提供有价值的参考信息和帮助。

调查问卷的设计主要参考了国际上 2014-2016 年度的《State of DevOps Report》及《State of Agile Report》的调查问卷设计，以实现信息数据在国内和国际之间的可比性，并根据业内专家意见经过多轮修改。目前问卷中英文版已上线，点此可进行中文版调查。本次问卷调查为学术公益性质，所形成的年度调查报告将免费公开。

期待大家能填写调查问卷，支持中国 DevOps 的发展！

微信序列号生成器架构设计及演变

作者 曾钦松

一、摘要

微信在立项之初，就已确立了利用数据版本号实现终端与后台的数据增量同步机制，确保发消息时消息可靠送达对方手机，避免了大量潜在的家庭纠纷。时至今日，微信已经走过第五个年头，这套同步机制仍然在消息收发、朋友圈通知、好友数据更新等需要数据同步的地方发挥着核心的作用。而在这同步机制的背后，需要一个高可用、高可靠的序列号生成器来产生同步数据用的版本号。这个序列号生成器我们称之为 seqsvr，目前已经发展为一个每天万亿级调用的重量级系统，其中每次申请序列号平时调用耗时 1ms，99.9% 的调用耗时小于 3ms，服务部署于数百台 4 核 CPU 服务器上。本文会重点介绍 seqsvr 的架构核心思想，以及 seqsvr 随着业务量快速上涨所做的架构演变。

二、背景

微信服务器端为每一份需要与客户端同步的数据（例如消息）都会赋予一个唯一的、递增的序列号（后文称为 sequence），作为这份数据的版本号。在客户端与服务器端同步的时候，客户端会带上已经同步下去数

据的最大版本号，后台会根据客户端最大版本号与服务器端的最大版本号，计算出需要同步的增量数据，返回给客户端。这样不仅保证了客户端与服务器端的数据同步的可靠性，同时也大幅减少了同步时的冗余数据。

这里不用乐观锁机制来生成版本号，而是使用了一个独立的 seqsvr 来处理序列号操作，一方面因为业务有大量的 sequence 查询需求——查询已经分配出去的最后一个 sequence，而基于 seqsvr 的查询操作可以做到非常轻量级，避免对存储层的大量 IO 查询操作；另一方面微信用户的不同种类的数据存在不同的 Key-Value 系统中，使用统一的序列号有助于避免重复开发，同时业务逻辑可以很方便地判断一个用户的各类数据是否有更新。

从 seqsvr 申请的、用作数据版本号的 sequence，具有两种基本的性质：

- 递增的64位整型变量；
- 每个用户都有自己独立的64位sequence空间。

举个例子，小明当前申请的 sequence 为 100，那么他下一次申请的 sequence，可能为 101，也可能是 110，总之一定大于之前申请的 100。而小红呢，她的 sequence 与小明的 sequence 是独立开的，假如她当前申请到的 sequence 为 50，然后期间不管小明申请多少次 sequence 怎么折腾，都不会影响到她下一次申请到的值（很可能是 51）。

这里用了每个用户独立的 64 位 sequence 的体系，而不是用一个全局的 64 位（或更高位）sequence，很大原因是全局唯一的 sequence 会有非常严重的申请互斥问题，不容易去实现一个高性能高可靠的架构。对微信业务来说，每个用户独立的 64 位 sequence 空间已经满足业务要求。

目前 sequence 用在终端与后台的数据同步外，同时也广泛用于微信后台逻辑层的基础数据一致性 cache 中，大幅减少逻辑层对存储层的

访问。虽然一个用于终端——后台数据同步，一个用于后台 cache 的一致性保证，场景大不相同。但我们仔细分析就会发现，两个场景都是利用 sequence 可靠递增的性质来实现数据的一致性保证，这就要求我们的 seqsvr 保证分配出去的 sequence 是稳定递增的，一旦出现回退必然导致各种数据错乱、消息消失；另外，这两个场景都非常普遍，我们在使用微信的时候会不知不觉地对应到这两个场景：小明给小红发消息、小红拉黑小明、小明发一条失恋状态的朋友圈，一次简单的分手背后可能申请了无数次 sequence。微信目前拥有数亿的活跃用户，每时每刻都会有海量 sequence 申请，这对 seqsvr 的设计也是个极大的挑战。那么，既要 sequence 可靠递增，又要能顶住海量的访问，要如何设计 seqsvr 的架构？我们先从 seqsvr 的架构原型说起。

三、架构原型

不考虑 seqsvr 的具体架构的话，它应该是一个巨大的 64 位数组，而我们每一个微信用户，都在这个大数组里独占一格 8bytes 的空间，这个格子就放着用户已经分配出去的最后一个 sequence：cur_seq。每个用户来申请 sequence 的时候，只需要将用户的 cur_seq+=1，保存回数组，并返回给用户（见图 1）。

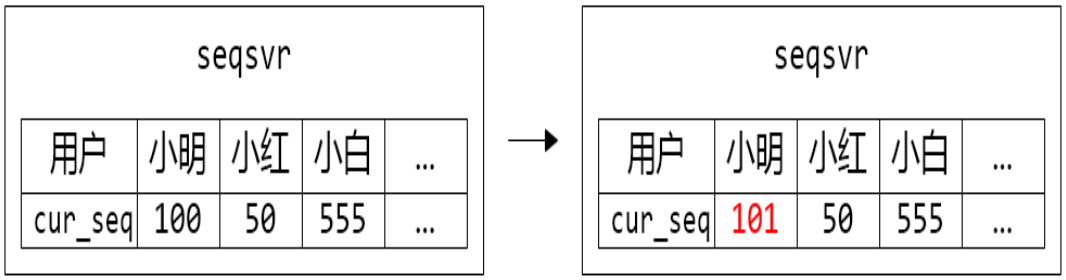


图 1. 小明申请了一个 sequence，返回 101

预分配中间层

任何一件看起来很简单的事，在海量的访问量下都会变得不简单。前文提到，seqsvr 需要保证分配出去的 sequence 递增（数据可靠），还需要满足海量的访问量（每天接近万亿级别的访问）。满足数据可靠的话，我们很容易想到把数据持久化到硬盘，但是按照目前每秒千万级的访问量（~10^7 QPS），基本没有任何硬盘系统能扛住。

后台架构设计很多时候是一门关于权衡的哲学，针对不同的场景去考虑能不能降低某方面的要求，以换取其它方面的提升。仔细考虑我们的需求，我们只要求递增，并没有要求连续，也就是说出现一大段跳跃是允许的（例如分配出的 sequence 序列：1, 2, 3, 10, 100, 101）。于是我们实现了一个简单优雅的策略：

内存中储存最近一个分配出去的 sequence：cur_seq，以及分配上限：max_seq。

分配 sequence 时，将 cur_seq++，同时与分配上限 max_seq 比较：如果 cur_seq > max_seq，将分配上限提升一个步长 max_seq += step，并持久化 max_seq

重启时，读出持久化的 max_seq，赋值给 cur_seq（见图 2）。

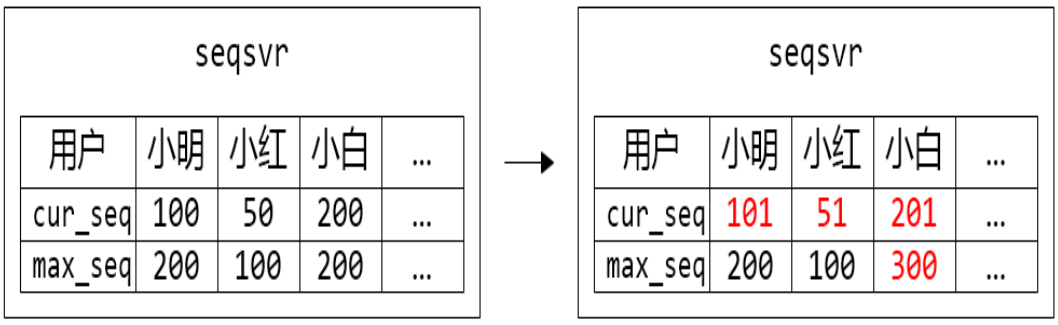


图 2. 小明、小红、小白都各自申请了一个 sequence，但只有小白的 max_seq 增加了步长 100

这样通过增加一个预分配 sequence 的中间层，在保证 sequence 不回退的前提下，大幅地提升了分配 sequence 的性能。实际应用中每次提升的步长为 10000，那么持久化的硬盘 IO 次数从之前 ~10^7 QPS 降低到 ~10^3 QPS，处于可接受范围。在正常运作时分配出去的 sequence 是顺序递增的，只有在机器重启后，第一次分配的 sequence 会产生一个比较大的跳跃，跳跃大小取决于步长大小。

分号段共享存储

请求带来的硬盘 IO 问题解决了，可以支持服务平稳运行，但该模型还是存在一个问题：重启时要读取大量的 max_seq 数据加载到内存中。

我们可以简单计算下，以目前 uid（用户唯一 ID）上限 2^32 个、一个 max_seq 8bytes 的空间，数据大小一共为 32GB，从硬盘加载需要不少时间。另一方面，出于数据可靠性的考虑，必然需要一个可靠存储系统来保存 max_seq 数据，重启时通过网络从该可靠存储系统加载数据。如果 max_seq 数据过大的话，会导致重启时在数据传输花费大量时间，造成一段时间不可服务。

为了解决这个问题，我们引入号段 Section 的概念，uid 相邻的一段用户属于一个号段，而同个号段内的用户共享一个 max_seq，这样大幅减少了 max_seq 数据的大小，同时也降低了 IO 次数（见图 3）。

目前 seqsvr 一个 Section 包含 10 万个 uid，max_seq 数据只有 300+KB，为我们实现从可靠存储系统读取 max_seq 数据重启打下基础。

工程实现

工程实现在上面两个策略上做了一些调整，主要是出于数据可靠性及灾难隔离考虑。

把存储层和缓存中间层分成两个模块 StoreSvr 及 AllocSvr。

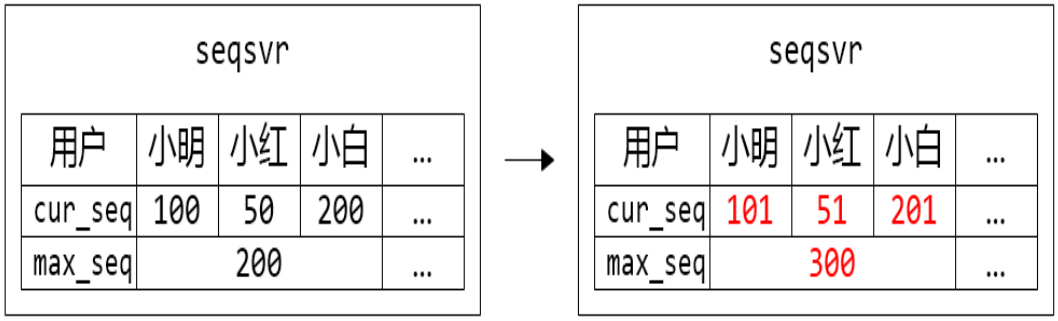


图 3. 小明、小红、小白属于同个 Section，他们共用一个 max_seq。在每个人都申请一个 sequence 的时候，只有小白突破了 max_seq 上限，需要更新 max_seq 并持久化

StoreSvr 为存储层，利用了多机 NRW 策略来保证数据持久化后不丢失；AllocSvr 则是缓存中间层，部署于多台机器，每台 AllocSvr 负责若干号段的 sequence 分配，分摊海量的 sequence 申请请求。

整个系统又按 uid 范围进行分 Set，每个 Set 都是一个完整的、独立的 StoreSvr+AllocSvr 子系统。分 Set 设计目的是为了做灾难隔离，一个 Set 出现故障只会影响该 Set 内的用户，而不会影响到其它用户（见图 4）。

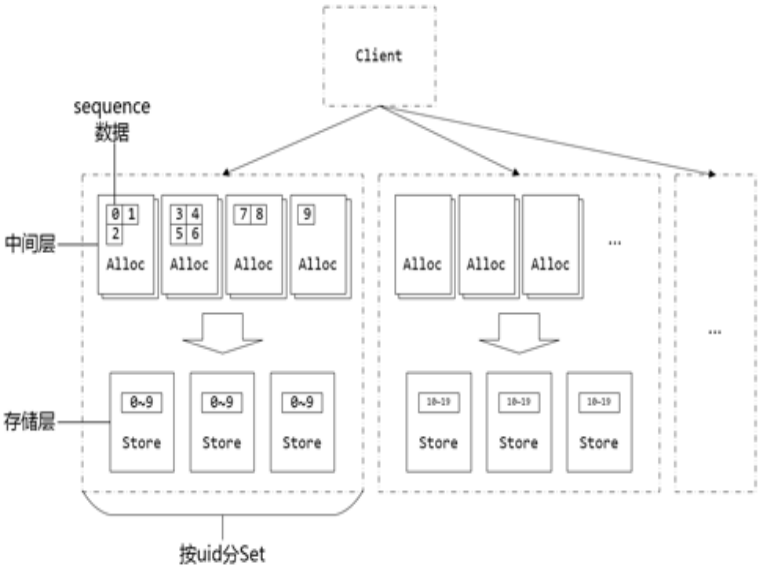


图 4. 原型架构图

四、容灾设计

接下来我们会介绍 seqsvr 的容灾架构。我们知道，后台系统绝大部分情况下并没有一种唯一的、完美的解决方案，同样的需求在不同的环境背景下甚至有可能演化出两种截然不同的架构。既然架构是多变的，那纯粹讲架构的意义并不是特别大，期间也会讲下 seqsvr 容灾设计时的一些思考和权衡，希望对大家有所帮助。

seqsvr 的容灾模型在五年中进行过一次比较大的重构，提升了可用性、机器利用率等方面。其中不管是重构前还是重构后的架构，seqsvr 一直遵循着两条架构设计原则：

- 保持自身架构简单；
- 避免对外部模块的强依赖。

这两点都是基于 seqsvr 可靠性考虑的，毕竟 seqsvr 是一个与整个微信服务端正常运行息息相关的模块。按照我们对这个世界的认识，系统的复杂度往往是跟可靠性成反比的，想得到一个可靠的系统一个关键点就是要把它做简单。相信大家身边都有一些这样的例子，设计方案里有很多高大上、复杂的东西，同时也总能看到他们在默默地填一些高大上的坑。当然简单的系统不意味着粗制滥造，我们要做的是理出最核心的点，然后在满足这些核心点的基础上，针对性地提出一个足够简单的解决方案。

那么，seqsvr 最核心的点是什么呢？每个 uid 的 sequence 申请要递增不回退。这里我们发现，如果 seqsvr 满足这么一个约束：任意时刻任意 uid 有且仅有一台 AllocSvr 提供服务，就可以比较容易地实现 sequence 递增不回退的要求（见图 5）。

但也由于这个约束，多台 AllocSvr 同时服务同一个号段的多主机模型在这里就不适用了。我们只能采用单点服务的模式，当某台 AllocSvr

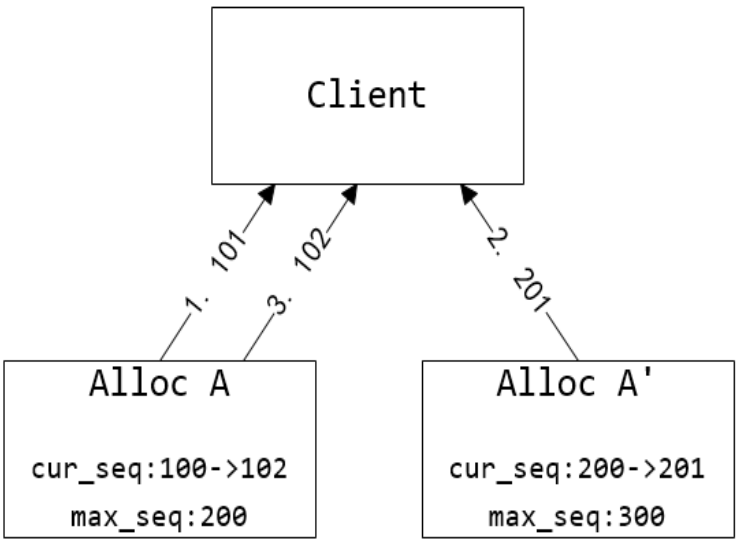


图 5. 两台 AllocSvr 服务同个 uid 造成 sequence 回退。Client 读取到的 sequence 序列为 101、201、102

发生服务不可用时，将该机服务的 uid 段切换到其它机器来实现容灾。这里需要引入一个仲裁服务，探测 AllocSvr 的服务状态，决定每个 uid 段由哪台 AllocSvr 加载。出于可靠性的考虑，仲裁模块并不直接操作 AllocSvr，而是将加载配置写到 StoreSvr 持久化，然后 AllocSvr 定期访问 StoreSvr 读取最新的加载配置，决定自己的加载状态（见图 6）。

同时，为了避免失联 AllocSvr 提供错误的服务，返回脏数据，AllocSvr 需要跟 StoreSvr 保持租约。这个租约机制由以下两个条件组成：

- 租约失效：AllocSvr N秒内无法从StoreSvr读取加载配置时，AllocSvr停止服务。
- 租约生效：AllocSvr读取到新的加载配置后，立即卸载需要卸载的号段，需要加载的新号段等待N秒后提供服务（见图7）。

这两个条件保证了切换时，新 AllocSvr 肯定在旧 AllocSvr 下线后才开始提供服务。但这种租约机制也会造成切换的号段存在小段时间的不可服务，不过由于微信后台逻辑层存在重试机制及异步重试队列，小段时间

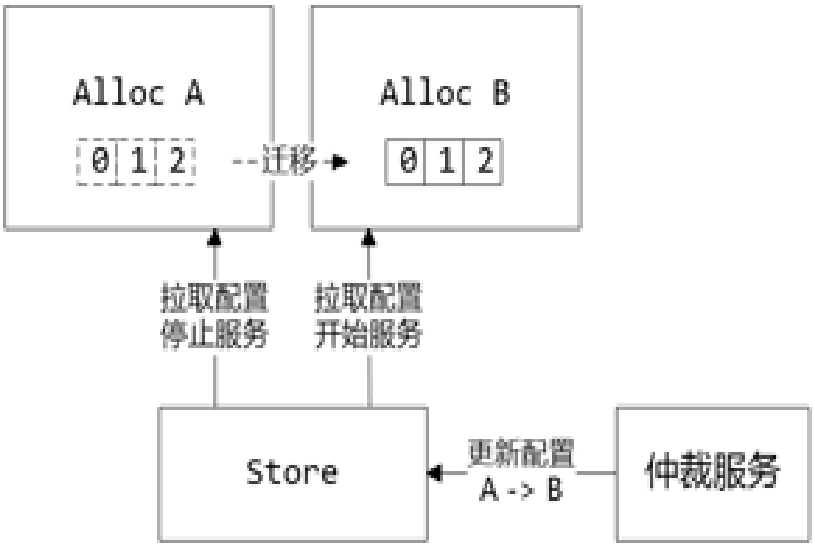


图 6. 号段迁移示意。通过更新加载配置把 0~2 号段从 AllocSvrA 迁移到 AllocSvrB

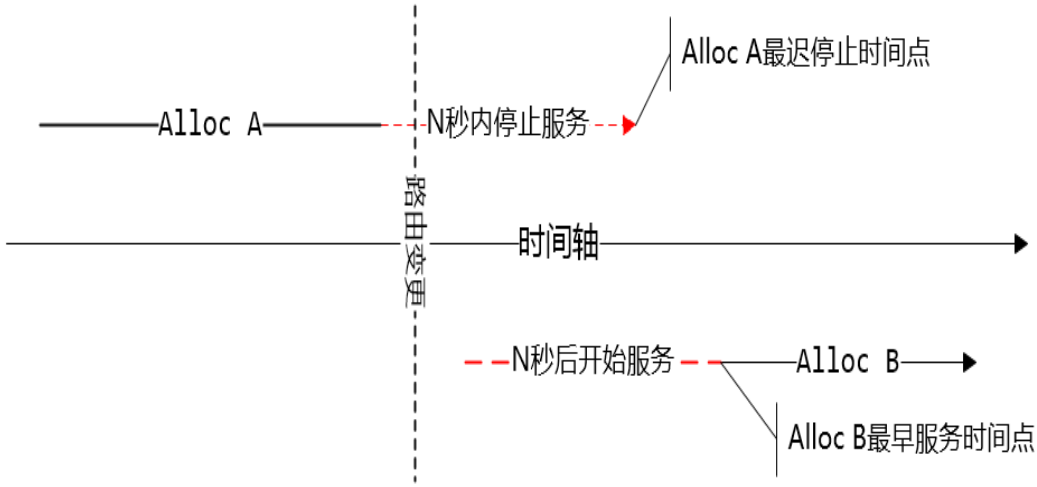


图 7. 租约机制。AllocSvrB 严格保证在 AllocSvrA 停止服务后提供服务

的不可服务是用户无感知的，而且出现租约失效、切换是小概率事件，整体上是可以接受的。

到此讲了 AllocSvr 容灾切换的基本原理，接下来会介绍整个 seqsvr 架构容灾架构的演变

五、容灾 1.0 架构：主备容灾

最初版本的 seqsvr 采用了主机 + 冷备机容灾模式：全量的 uid 空间均匀分成 N 个 Section，连续的若干个 Section 组成了一个 Set，每个 Set 都有一主一备两台 AllocSvr。正常情况下只有主机提供服务；在主机出故障时，仲裁服务切换主备，原来的主机下线变成备机，原备机变成主机后加载 uid 号段提供服务（见图 8）。

可能看到前文的叙述，有些同学已经想到这种容灾架构。一主机一备机的模型设计简单，并且具有不错的可用性——毕竟主备两台机器同时不可用的概率极低，相信很多后台系统也采用了类似的容灾策略。

设计权衡

主备容灾存在一些明显的缺陷，比如备机闲置导致有一半的空闲机器；比如主备切换的时候，备机在瞬间要接受主机所有的请求，容易导致备机

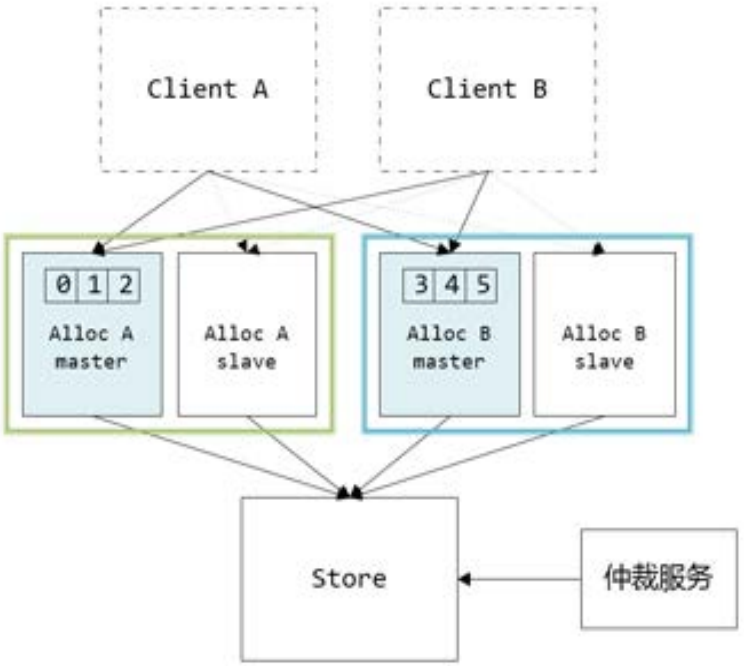


图 8. 容灾 1.0 架构：主备容灾

过载。既然一主一备容灾存在这样的问题，为什么一开始还要采用这种容灾模型？事实上，架构的选择往往跟当时的背景有关，seqsvr 诞生于微信发展初期，也正是微信快速扩张的时候，选择一主一备容灾模型是出于以下的考虑：

- 架构简单，可以快速开发；
- 机器数少，机器冗余不是主要问题；
- Client端更新AllocSvr的路由状态很容易实现。

前两点好懂，人力、机器都不如时间宝贵。而第三点比较有意思，下面展开讲下。

微信后台绝大部分模块使用了一个自研的 RPC 框架，seqsvr 也不例外。在这个 RPC 框架里，调用端读取本地机器的 client 配置文件，决定去哪台服务端调用。这种模型对于无状态的服务端，是很好用的，也很方便实现容灾。我们可以在 client 配置文件里面写“对于号段 x，可以去 SvrA、SvrB、SvrC 三台机器的任意一台访问”，实现三主机容灾。

但在 seqsvr 里，AllocSvr 是预分配中间层，并不是无状态的。而前面我们提到，AllocSvr 加载哪些 uid 号段，是由保存在 StoreSvr 的加载配置决定的。那么这时候就尴尬了，业务想要申请某个 uid 的 sequence，Client 端其实并不清楚具体去哪台 AllocSvr 访问，client 配置文件只会跟它说“AllocSvrA、AllocSvrB…这堆机器的某一台会有你想要的 sequence”。换句话讲，原来负责提供服务的 AllocSvrA 故障，仲裁服务决定由 AllocSvrC 来替代 AllocSvrA 提供服务，Client 要如何获知这个路由信息的变更？

这时候假如我们的 AllocSvr 采用了主备容灾模型的话，事情就变得简单多了。我们可以在 client 配置文件里写：对于某个 uid 号段，要

么是 AllocSvrA 加载，要么是 AllocSvrB 加载。Client 端发起请求时，尽管 Client 端并不清楚 AllocSvrA 和 AllocSvrB 哪一台真正加载了目标 uid 号段，但是 Client 端可以先尝试给其中任意一台 AllocSvr 发请求，就算这次请求了错误的 AllocSvr，那么就知道另外一台是正确的 AllocSvr，再发起一次请求即可。

也就是说，对于主备容灾模型，最多也只会浪费一次的试探请求来确定 AllocSvr 的服务状态，额外消耗少，编码也简单。可是，如果 Svr 端采用了其它复杂的容灾策略，那么基于静态配置的框架就很难去确定 Svr 端的服务状态：Svr 发生状态变更，Client 端无法确定应该向哪台 Svr 发起请求。这也是为什么一开始选择了主备容灾的原因之一。

主备容灾的缺陷

在我们的实际运营中，容灾 1.0 架构存在两个重大的不足：

- 扩容、缩容非常麻烦；
- 一个Set的主备机都过载，无法使用其他Set的机器进行容灾。

在主备容灾中，Client 和 AllocSvr 需要使用完全一致的配置文件。变更这个配置文件的时候，由于无法实现在同一时间更新给所有的 Client 和 AllocSvr，因此需要非常复杂的人工操作来保证变更的正确性（包括需要使用 iptables 来做请求转发，具体的详情这里不做展开）。

对于第二个问题，常见的方法是用一致性 Hash 算法替代主备，一个 Set 有多台机器，过载机器的请求被分摊到多台机器，容灾效果会更好。在 seqsvr 中使用类似一致性 Hash 的容灾策略也是可行的，只要 Client 端与仲裁服务都使用完全一样的一致性 Hash 算法，这样 Client 端可以启发式地去尝试，直到找到正确的 AllocSvr。例如对于某个 uid，仲裁服务会优先把它分配到 AllocSvrA，如果 AllocSvrA 挂掉则分配到

AllocSvrB，再不行分配到 AllocSvrC。那么 Client 在访问 AllocSvr 时，按照 AllocSvrA -> AllocSvrB -> AllocSvrC 的顺序去访问，也能实现容灾的目的。但这种方法仍然没有克服前面主备容灾面临的配置文件变更的问题，运营起来也很麻烦。

六、容灾 2.0 架构：嵌入式路由表容灾

最后我们另辟蹊径，采用了一种不同的思路：既然 Client 端与 AllocSvr 存在路由状态不一致的问题，那么让 AllocSvr 把当前的路由状态传递给 Client 端，打破之前只能根据本地 Client 配置文件做路由决策的限制，从根本上解决这个问题。

所以在 2.0 架构中，我们把 AllocSvr 的路由状态嵌入到 Client 请求 sequence 的响应包中，在不带来额外的资源消耗的情况下，实现了 Client 端与 AllocSvr 之间的路由状态一致。具体实现方案如下：

seqsvr 所有模块使用了统一的路由表，描述了 uid 号段到 AllocSvr 的全映射。这份路由表由仲裁服务根据 AllocSvr 的服务状态生成，写到 StoreSvr 中，由 AllocSvr 当作租约读出，最后在业务返回包里旁路给 Client 端（见图 9）。

把路由表嵌入到请求响应包看似很简单的架构变动，却是整个 seqsvr 容灾架构的技术奇点。利用它解决了路由状态不一致的问题后，可以实现一些以前不容易实现的特性。例如灵活的容灾策略，让所有机器都互为备机，在机器故障时，把故障机上的号段均匀地迁移到其它可用的 AllocSvr 上；还可以根据 AllocSvr 的负载情况，进行负载均衡，有效缓解 AllocSvr 请求不均的问题，大幅提升机器使用率。

另外在运营上也得到了大幅简化。之前对机器进行运维操作有着繁杂

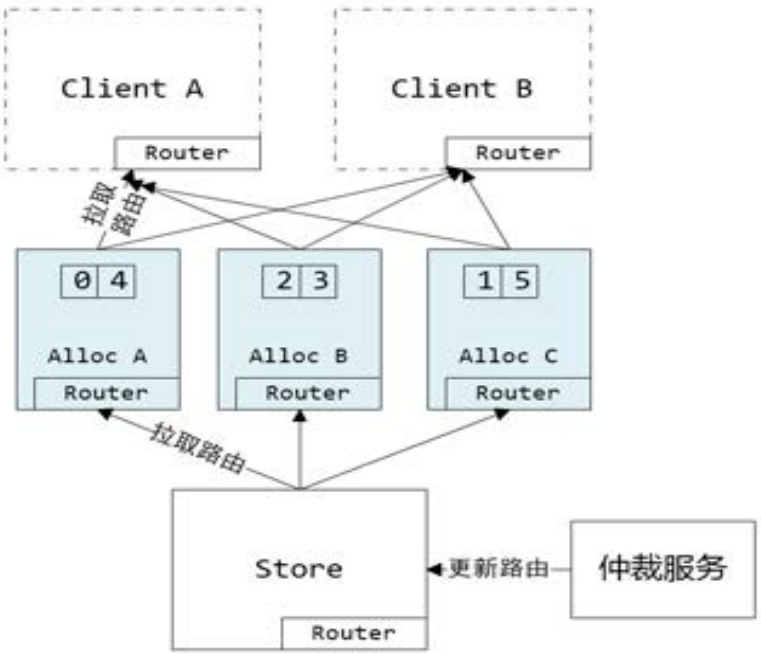


图 9. 容灾 2.0 架构：动态号段迁移容灾

的操作步骤，而新架构只需要更新路由即可轻松实现上线、下线、替换机器，不需要关心配置文件不一致的问题，避免了一些由于人工误操作引发的故障（见图 10）。

路由同步优化

把路由表嵌入到取 sequence 的请求响应包中，那么会引入一个类似“先有鸡还是先有蛋”的哲学命题：没有路由表，怎么知道去哪台 AllocSvr 取路由表？另外，取 sequence 是一个超高频的请求，如何避免嵌入路由表带来的带宽消耗？

这里通过在 Client 端内存缓存路由表以及路由版本号来解决，请求步骤如下：

1. Client根据本地共享内存缓存的路由表，选择对应的AllocSvr；如果路由表不存在，随机选择一台AllocSvr
2. 对选中的AllocSvr发起请求，请求带上本地路由表的版本号

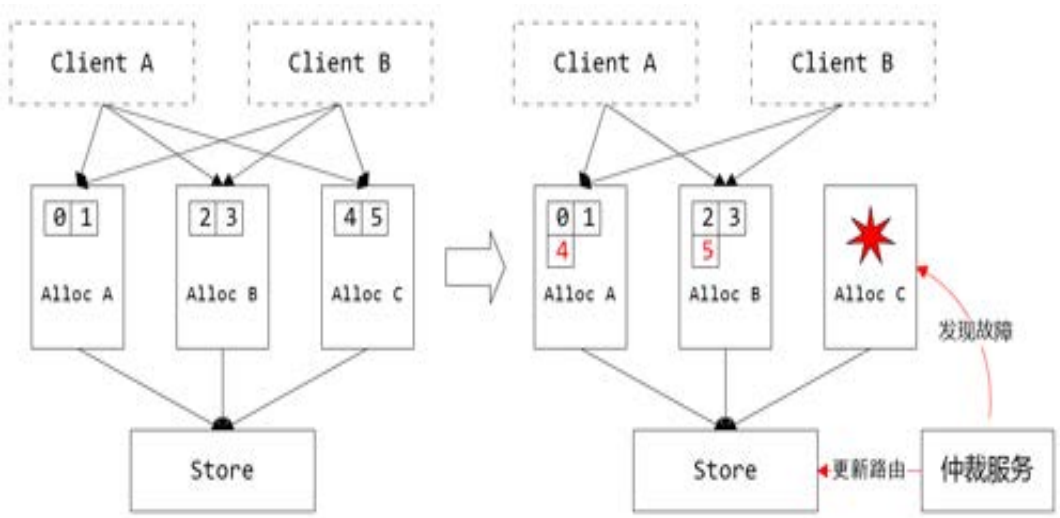


图 10. 机器故障号段迁移

3. AllocSvr收到请求，除了处理sequence逻辑外，判断Client带上版本号是否最新，如果是旧版则在响应包中附上最新的路由表
4. Client收到响应包，除了处理sequence逻辑外，判断响应包是否带有新路由表。如果有，更新本地路由表，并决策是否返回第1步重试

基于以上的请求步骤，在本地路由表失效的时候，使用少量的重试便可以拉到正确的路由，正常提供服务。

七、总结

到此把 seqsvr 的架构设计和演变基本讲完了，正是如此简单优雅的模型，为微信的其它模块提供了一种简单可靠的一致性解决方案，支撑着微信五年来的高速发展，相信在可预见的未来仍然会发挥着重要的作用。

Nginx 日志中的金矿

作者 张晓庆

Nginx（读作 Engine-X）是现在最流行的负载均衡和反向代理服务器之一。如果你是一名中小微型网站的开发运维人员，很可能像我们一样，仅 Nginx 每天就会产生上百 M 甚至数以十 G 的日志文件。如果没有出什么错误，在被 logrotate 定期分割并滚动删除以前，这些日志文件可能都不会被看上一眼。

实际上，Nginx 日志文件可以记录的信息相当丰富，而且格式可以定制，考虑到 `$time_local` 请求时间字段几乎必有，这是一个典型的基于文件的时间序列数据库。Nginx 日志被删除以前，或许我们可以想想，其中是否蕴含着未知的金矿等待挖掘？

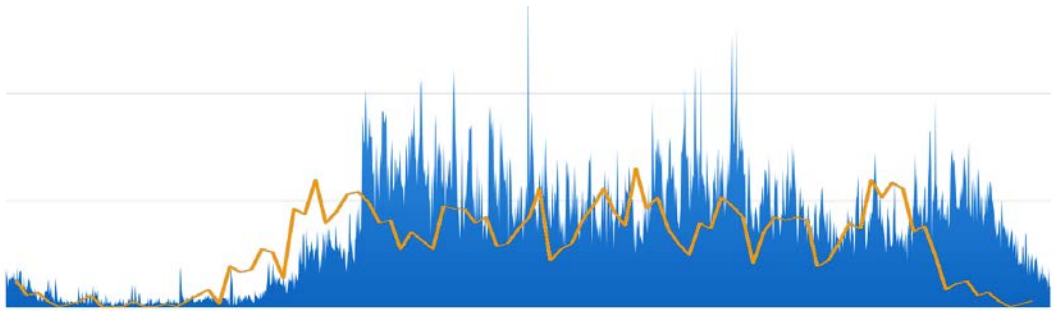
请求访问分析

Nginx 中的每条记录是一个单独的请求，可能是某个页面或静态资源的访问，也可能是某个 API 的调用。通过几条简单的命令，了解一下系统的访问压力。

请求总数、平均每秒请求数、峰值请求数，可以大体了解系统压力，

```
001    // 请求总数
002     less main.log | wc -l
003     1080577
004     // 平均每秒的请求数
005     less main.log | awk
      ‘{sec=substr($4,2,20);reqs++;reqsBySec[sec]++;} END{print reqs/
      length(reqsBySec)}’
006     14.0963
007     // 峰值每秒请求数
008     less main.log | awk ‘{sec=substr($4,2,20);requests[sec]++;}
      END{for(s in requests){printf(“%s %s\n”, requests[s],s)}}’ | sort
      -nr | head -n 3
009     Page Visits  Response Size  Time Spent/req  Moment
010     182 10/Apr/2016:12:53:20
011     161 10/Apr/2016:12:54:53
012     160 10/Apr/2016:10:47:23
```

作为系统扩容、性能及压力测试时的直接参考。查询特定的 URL，比如下单页面，了解每天的下单状况，导出 CSV 格式，或使用可视化工具，更直观地了解一段时间内的请求、下单数据。



备注：本文使用 awk 命令处理，与 Nginx 日志的格式有关，如果您格式不同，请酌情修改命令。本文所用的 Nginx 日志格式：

```
001 $remote_addr - $remote_user [$time_local] “$request”
002 $status  $body_bytes_sent $request_time $upstream_response_time
003 $upstream_addr “$http_referer” “$http_user_agent” “$http_x_
      forwarded_for”’;
004
005    示例：
006
007 42.100.52.XX - - [10/Apr/2016:07:29:58 +0800] “GET /index
008 HTTP/1.1” 200 7206 0.092 0.092 “-” “Mozilla/5.0 (iPhone; CPU iPhone
      OS
009  7_1_2 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko)
      Mobile/11D257” “
```

流量速率分析

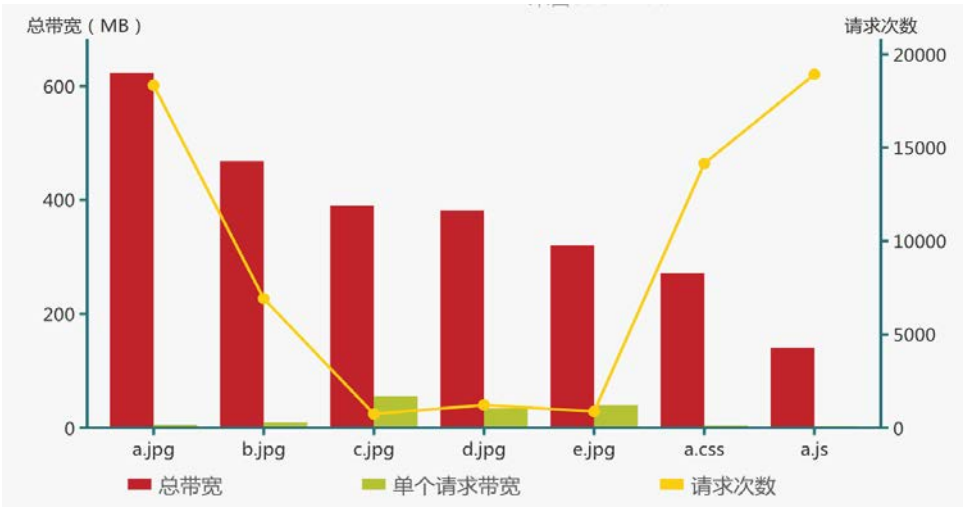
Nginx 日志如果开启，除了请求时间，一般会包含响应时间、页面尺寸等字段，据此很容易计算出网络流量、速率。

等等，你可能会有疑问，上面的请求访问分析，这里的流量速率分析，按时间轴画出来，不就是监控系统干的事儿吗，何苦这么麻烦查询 Nginx 日志？

的确如此，监控系统提供了更实时、更直观的方式。而 Nginx 日志文件的原始数据，可以从不同维度分析，使用得当，会如大浪淘沙般，发现属于我们的金子。

对一般网站来说，带宽是最珍贵的资源，可能一不小心，某些资源如文件、图片就占用了大量的带宽，执行命令检查如下：

```
001 less static.log | awk ‘url=$7; requests[url]++;bytes[url]+=$10}
002 END{for(url in requests){printf(“%sMB %sKB/req %s %s\n”, bytes[url]
      /
003 1024 / 1024, bytes[url] /requests[url] / 1024, requests[url],
      url)}}’ | sort -nr | head -n 15
```



备注：Nginx 配置文件中日志格式使用了 $body_sent_size，指 HTTP 响应体的大小，如果想查看整个响应的大小，应该使用变量 $sent_size。

不出意外，静态资源、图片类（如果还没有放 CDN）占据榜首，自然也是优化的重点：是否可以再压缩，某些页面中是否可以用缩略图片代替等。

与之相比，后台调用、API 接口等通常消耗更多的 CPU 资源，按照一贯“先衡量、再优化”的思路，可以根据响应时间大体了解某个 URL 占用的 CPU 时间。

```
001 less main.log | awk ‘{url=$7; times[url]++} END{for(url in
002 times){printf(“%s %s\n”, times[url], url)}}’ | sort -nr | more`
003     40404 /page/a?from=index
004     1074 /categories/food
005     572 /api/orders/1234.json
```

不对，发现一个问题：由于拥有服务号、App、PC 浏览器等多种前端，并且使用不规范，URL 的格式可能乱七八糟。比如 /page/a 页面，有的带有 .html 后缀，有的未带，有的请求路径则带有参数；分类页 /categories/food 带有 slug 等信息；订单、详情或个人中心的 URL 路径则有 ID 等标记。

借助 sed 命令，通过三个方法对 URL 格式进行归一化处理：去掉所有的参数；去掉 .html 及 .json 后缀；把数字替换为“*”。可以得到更准确的统计结果。

```
001 less main.log | awk ‘{print $7}’ |sed -re ‘s/(.*)\?.*/\1/g’ -e
002 ‘s/(.*)\..*/\1/g’ -e ‘s:/[0-9]+:/*:g’ | awk ‘{requests[$1]++;time[$1]
003 +=$2} END{for(url in requests){printf(“%smin %ss/req %s %s\n”, time
004 [url] / 60, time[url] /requests[url], requests[url], url)}}’ | sort
      -nr | head -n 50
```

备注：这里使用了扩展正则表达式，GNU sed 的参数为 -r，BSD sed 的参数为 -E。

那些累计占用了更多响应时间的请求，通常也耗用了更多的 CPU 时间，是性能优化重点照顾的对象。

慢查询分析

“服务号刚推送了文章，有用户反映点开很慢”，你刚端起桌子上的水杯，就听到产品经理的大嗓门从办公室角落呼啸而来。“用户用的什么网络”，你一边问着，一边打开服务号亲自尝试一下。是用户网络环境不好，还是后台系统有了访问压力？是这一个用户慢，还是很多用户都慢？你一边脑子里在翻腾，一边又打开命令行去查看日志。

与PC浏览器相比，微信服务号在网络环境、页面渲染上有较大的掣肘，在缓存策略上也不如APP自如，有时会遇到诡异的问题。如果手里恰好有Nginx日志，能做点什么呢？

考虑一下MySQL数据库，可以打开慢查询功能，定期查找并优化慢查询，与此类似，Nginx日志中的响应时间，不相当于自带慢查询功能嘛。利用这一特性，我们分步进行慢查询分析。

第一步：是不是用户的网络状况不好？根据既往的经验，如果只有少量的请求较慢，而前后其他IP的请求都较快，通常是用户手机或网络状况不佳引起的。最简单的方法，统计慢查询所占比例。

```
005 less main.log | awk -v limit=2 ‘{min=substr($4,2,17);reqs[min]
006 ++;if($11>limit){slowReqs[min]++}} END{for(m in slowReqs){printf(“%s
007  %s %s%s %s\n”, m, slowReqs[m]/reqs[m] * 100, “%”, slowReqs[m], reqs
008 [m])}}’ | more
009     10/Apr/2016:12:51 0.367% 7 1905
010     10/Apr/2016:12:52 0.638% 12 1882
011     10/Apr/2016:12:53 0.548% 14 2554
```

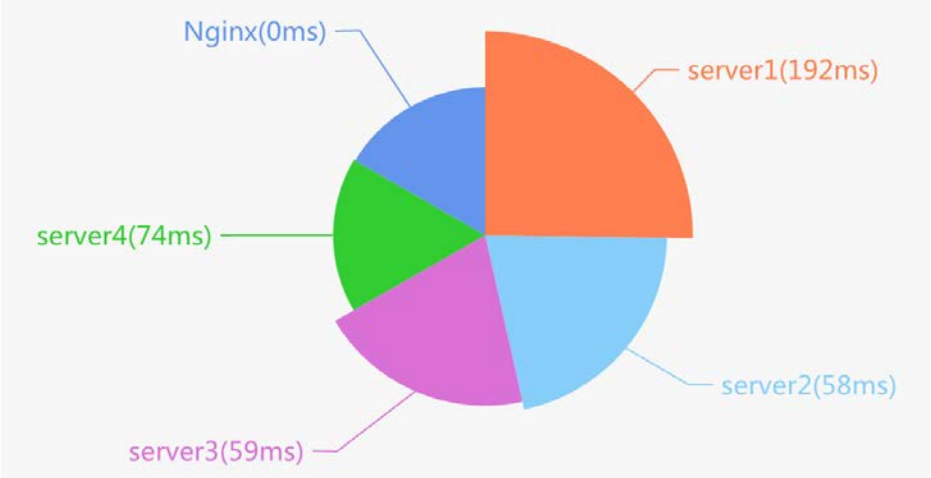
慢查询所占比例极低，再根据用户手机型号、访问时间、访问页面等信息看能否定位到指定的请求，结合前后不同用户的请求，就可以确定是否用户的网络状况不好了。

第二步：是不是应用系统的瓶颈？对比应用服务器的返回时间（$upstream_response_time字段），与Nginx服务器的处理时间（$request_

time 字段），先快速排查是否某一台服务器抽风。

我们遇到过类似问题，平均响应时间 90ms，还算正常，但某台服务器明显变慢，平均响应时间达到了 200ms，影响了部分用户的访问体验。

```
001 less main.log | awk ‘{upServer=$13;upTime=$12;if(upServer ==
002 “-”){upServer=”Nginx”};if(upTime == “-”){upTime=0};upTimes[upServer]
003 +=upTime;count[upServer]++;totalCount++;} END{for(server in upTimes)
004 {printf(“%s %s%s %ss %s\n”, count[server], count[server]/totalCount
      *
005 100, “%”, upTimes[server]/count[server], server)}}’ | sort -nr
```



不幸，市场部此次推广活动，访问压力增大，所有服务器都在变慢，更可能是应用系统的性能达到了瓶颈。如果此时带宽都没跑满，在硬件扩容之前，考虑优化重点 API、缓存、静态化策略吧，达到一个基本的要求：“优化系统，让瓶颈落到带宽上”。

第三步：应用系统没有瓶颈，是带宽的问题？快速查看一下每秒的流量：

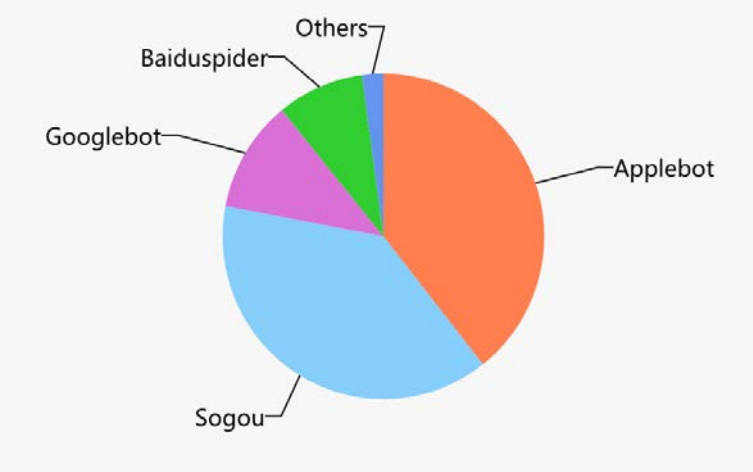
```
001 less main.log | awk ‘{second=substr($4,2,20);bytes[second]+=$10;}
002  END{for(s in bytes){printf(“%sKB %s\n”, bytes[s]/1024, s)}}’ |
      more`
003     1949.95KB 10/Apr/2016:12:53:15
004     2819.1KB 10/Apr/2016:12:53:16
005     3463.64KB 10/Apr/2016:12:53:17
006     3419.21KB 10/Apr/2016:12:53:18
007     2851.37KB 10/Apr/2016:12:53:19
```

峰值带宽接近出口带宽最大值了，幸福的烦恼，利用前面介绍的不同 URL 的带宽统计，做定向优化，或者加带宽吧。

还能做那些优化

SEO 团队抱怨优化了那么久，为什么页面索引量和排名上不去。打印出不同爬虫的请求频次（$http_user_agent），或者查看某个特定的页面，最近有没有被爬虫爬过。

```
012 less main.log | egrep 'spider|bot' | awk '{name=$17;if(index
013 ($15,"spider")>0){name=$15};spiders[name]++} END{for(name in
      spiders)
014 {printf("%s %s\n",spiders[name], name)}}' | sort -nr
```

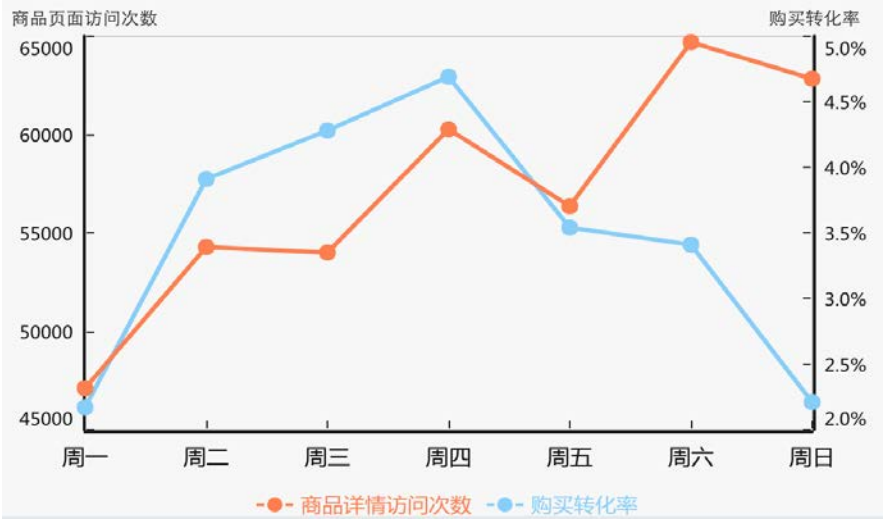


数据告诉我们，页面索引量上不去，不一定是某个爬虫未检索到页面，更多的是其他原因。

市场团队要上一个新品并且做促销活动，你建议避开周一周五，因为周三周四的转化率更高。

周三、周四的转换率比周末高不少，可能跟平台的发货周期有关，客户周三四下单，希望周末就能收到货，开始快乐的周末。你猜测到用户的心理和期望，连数据一起交市场品团队，期待更好地改善。

这样的例子可以有很多。事实上，上述分析限于 Nginx 日志，如果有



系统日志，并且日志格式定义良好，可以做的事情远不止于此：这是一个时间序列数据库，可以查询 IT 系统的运行情况，可以分析营销活动的效果，也可以预测业务数据的趋势；这是一个比较小但够用的大数据源，运用你学会的大数据分析方法，也可以像滴滴那样，分并预测不同天气、时间段下不同地区的车辆供需，并作出优化。

几点建议

1. 规范日志格式。这是很多团队容易忽略的地方，有时候多一个空格会让日志分析的复杂度大为增加。
2. 无论如何，使用时间戳字段。以时间序列的方式看待日志文件，这也是很多公司把系统日志直接写入到时间序列数据库的原因。
3. 如有可能，记录以下字段：用户（或者客户端）标识、单次请求标识、应用标识（如果单次请求会走到多个应用）。能够方便地查出用户链路、请求链路，是排查错误请求、分析用户行为的基础。
4. 关注写的操作。就像业务建模时，需要特别关注具有时标性、状

态会发生改变的模型一样，任何写的操作，都应记录到日志系统中。万一某个业务出错，不但可以通过业务模型复演，也可以通过日志系统复演。

5. 规范URL格式。这一点同样容易遭到忽略，商品详情页面要不要添加"?from=XXX"来源参数？支付页面采用路径标记“payment/alipay”，还是参数标记“/payment?type=alipay”更合适？区别细微但影响不可忽略。

技术团队应该像对待协议一样对待这些规范。仔细定义并严格遵守，相当于拿到了金矿的钥匙。

还需要寻找一个合适的日志分析工具，基于 Python、Go、Lua，都有免费的日志分析工具可供使用；想更轻量，准备几条常用的 shell 脚本，比如作者整理了一些到 GitHub 的这个项目上；或者基于 ELK 技术栈，把 Nginx 访问日志、业务日志统一存储，并通过 Kibana 进行不同维度的聚合分析，都是不错的办法。

或许你早就使用 Nginx 日志了，你是怎么使用的，有什么好的方法呢，欢迎一起交流。

OpsDev 将至

作者 金灵杰

DevOps 在 09 年被提出之后，热度一直不减。最近几年，随着容器化、微服务等概念的兴起，DevOps 又开始了大规模实践。但是，随着基础设施的增加，软件设计从 Ops 角度出发，可以更好的规避基础设施变更带来的风险。

从苹果公司开始

开发者、用户、投资者、分析师和竞争对手都渴望在 WWDC 上学习到苹果公司维持其领导地位的方式。虽然没有令人兴奋的新产品发布，但是贯穿其中很多话题的核心是：用户体验。

和其他手机厂商的发布会不同，苹果公司专注于客户的体验，而不是其产品的功能和特性。当竞争者在吹捧他们新品拥有的高像素相机和处理器核心数量的时候，苹果公司展示 iPhone 拍摄的优美、富有灵感的照片时，没有提及设备的任何技术细节。

现在，手机已经改变了我们的生活，日常的衣食住行都可以通过手机 app 来协助完成。但是苹果公司认为智能设备还能使我们的生活更加高效。不同于目前每个 app 提供独立的功能，苹果公司希望能够将这些服务整合到一起，对于用户来说，他们只需要使用苹果的服务，而无需打开多个 app。要实现这样的愿景，产品或者服务都需要有新的设计范式。任何期望通过接入苹果服务以对外提供个性化用户体验的公司，都应该考虑 OpsDev 而不是 DevOps。

进入 OpsDev 的世界

我们假设一家电器公司制造的智能冰箱，为用户提供以下体验。

当我坐在车里时，智能冰箱通过手机发出提醒：冰箱存货不足，该去买些食品啦。此时，手机自动生成并推荐了三个采购方案。第一个超市不需要绕路，但是没有我喜欢吃的冰淇淋；第二个超市比第一个远 10 分钟，但是有所有我喜欢吃的所有食物；第三个超市更远，需要多 15 分钟的路程，但是它除了有所有需要采购的东西，还提供额外优惠券。我点击了其中一个方案，车载导航开始自动规划行车路线，并显示在汽车多媒体系统中。

在不久的将来，很多公司会致力于提供集成的个性化用户体验，因此不同的数据源和服务必须被整合到一起。智能冰箱通过传感器判断冰箱内的存货，超市服务提供其库存和优惠信息，其他还有交通信息、地理位置信息等等。这些数据源由不同的提供商提供，并且保存在不同的数据中心。这些数据的访问，需要特定的授权方式、API 和数据处理流程。个性化服务的设计者，必须明确了解每个服务的规约，因为任务信息不能及时正确的获取，都会影响到用户体验。作为零售商，不会希望用户多花了 15 分钟时间过来，结果却因为需要购买的商品缺货，或者无法使用优惠券而造

成损失。

由此可见，个性化软件服务的交付影响了现有的软件设计模式。DevOps 克服了传统瀑布流所不能提供的快速交付能力，通过诸如代码审查、编码规范、持续集成等方式解决软件开发遇到的挑战，并且让开发直接参与到了产品上线流程。而 OpsDev 考虑的方面却不同。由于个性化服务依赖于底层一系列基础设施，一旦我们知道了每个独立的数据源的依赖关系和可用性，首先需要设计出将各个数据源结合在一起的组件。因此，设计者必须从运维的角度来考虑，对每个服务的依赖程度、故障修复、容错等；同时，由于每个服务相对独立由于每个服务相互独立，个性化软件服务的升级频率受其依赖的服务升级频率影响，这也需要从运维的角度考虑向下（向上）兼容。

什么是 OpsDev

OpsDev 是在开发行为开始前，就需要了解应用程序依赖的所有组件，并进行建模。此外，对基础设施稳定性的考量、环境建模、安全审计措施都是第一要务。其次，应用程序组件上线后的部署环境需要被建模。再次，将组件部署到生产环境的流程必须尽可能自动化。通过以上步骤，设计和研发团队可以在开发和测试阶段复制应用程序、环境模型和自动化部署流程。这样设计、研发和测试团队可以清楚的知道应用程序在生产环境的约束和参数。在传统的项目模式下，大量时间浪费在完成验证的应用程序部署到生产环境之后才发现的缺陷。

由于部署自动化在设计之初就被考虑到，应用程序可以随时被部署到开发、测试和生产环境。这种方式不仅可以通过自动化流程加快各个环节的部署速度，还可以减少因为手工部署导致的质量问题。部署程序应该可

以聚合多个独立开发的模块。每个模块都可以由独立的开发团队进行开发，并且定制适合自己的持续基础、持续交付流程。发布程序需要预先知道每个模块之间的依赖关系，这样可以快速的聚合每个通过持续集成、持续交付的流程的模块，将它们整合之后部署到集成环境或者生产环境。

通过 OpsDev 方法，发布方式变得更加灵活。整个部署流程被设计的可以插入验证流程。一个典型的持续交付流程可以是：发布程序首先将待发布应用程序部署到集成环境（UAT 测试环境、预发布环境）上，然后触发发布前的自动化、手工验证、性能基线验证等一系列验证措施，整个过程通过之后，部署程序再将应用程序发布到生产环境，发布成功之后记录基线信息和本次变更的状态。

总结

对于物联网或者基于用户体验的移动 App 厂商来说，传统的软件开发流程无法适应它们的产品，因为它们都会对很多 SaaS 服务有依赖，且应用程序包含很多组件（物联网设备中的 App、移动设备中的 App、数据中心的后台服务、网页服务等）。像苹果这样鼓励开发者优先思考用户体验，并提供个性化服务的公司，可能会加速思考方式从 DevOps 向 OpsDev 的转变。

通过 Cloud Data Services 打造新型认知计算数据分析云平台

作者 刘羽飞

在今年初 IBM 首席执行官 Ginni Rometty 公开表示将向认知计算与云计算平台方向转型之后，尽管外界一致认为这一过程并不会轻松，但经过近半年时间的努力，IBM 确实已取得了一些进展，尤其是在中国国内，比如 IBM 重新整合了以新开发平台、云服务以及开源数据工具相结合的云数据服务体系，以及通过与世纪互联合作的方式实现了云数据库产品 Cloudant 的落地。那么在这些举措的背后，IBM 的真正目标是什么？ IBM 转向云计算，又会对其用户以及整个行业带来哪些影响呢？为此 InfoQ 对 IBM 中国开发中心大数据及分析平台总经理吉燕勇进行了专访。

InfoQ：请先谈一谈您目前所负责的工作，以及之前的一些主要工作经历。

吉燕勇：在过去十几年当中，我一直在 IBM 中国开发中心工作，目前主要负责的工作是实现云上的大数据分析能力。IBM 中国开发中心是 1999

年成立的，专门负责开发 IBM 自己的核心产品。

2004 年之前，我主要负责电子商务及相关产品的开发工作。从 2004 年到 2010 年之间，在结构化数据方面，为了给国内众多银行用户提供更强的数据库方面的技术支持，我负责组建了 IBM 中国开发中心的数据库开发团队，另外在非结构化数据方面，企业内容管理开发团队同样也是我组建起来的。

到 2011 年的时候，因为 IBM 之前已经收购了 SPSS、Cognos 等公司，我又组建了业务分析团队，这样一来 IBM 中国开发中心大数据及分析平台的结构化数据团队、非结构化数据团队、以及分析团队就比较完整了。

近两年 IBM 的技术方向变化比较快，重点往认知计算、云平台方向发展。因此今年，我们就把负责云端开发的部门独立了出来，而我来负责所有大数据以及分析的云端开发工作。

另外因为市场对大数据人才的需求越来越强，我们从 2012 年开始就跟西安交大一起设立了一个 IBM 大数据分析专业，我同时也兼任着该系的主任一职。这就是我在 IBM 中国开发中心的主要工作经历。

InfoQ：云端的数据处理能力，可以让企业更快速的进行实时数据分析，更便捷地访问、分享、管理企业自己的数据。我们知道 IBM 目前在这个领域中的产品服务组合被称为 Cloud Data Services，其中包含了一系列基于云的企业级数据分析管理工具以及相关服务，虽然 IBM 提出 CDS 概念的时间并不长，但我们能看到实际上 CDS 中的服务并不是新开发的。那么能否请您对那些不太熟悉这些服务的网友们介绍一下 IBM CDS 的整体架构吗？其中集成了哪些比较重要的产品和解决方案？

吉燕勇： IBM Cloud Data Services 涵盖了几乎全部的 IBM 核心大数据及分析技术能力，它可以分为五个方面，也就是数据库、数据分析、

企业内容管理、数据集成、洞察服务。

IBM之前收购了The Whether Company，另外还与Twitter展开了合作，希望能够充分利用这些气象数据、社交数据方面的资源，更好地打造 IBM CDS 的洞察服务。所有这些技术能力，共同构成了 IBM 在云端的数据服务，也希望通过这些服务，能为用户带来更多的应用方式。

目前的 CDS 平台上的服务，更多是面向企业级用户的，这些服务以 24×7 的形式，不间断地为用户提供运维等方面的技术支持。同时，我们在打造 CDS 平台的过程中，也将整合大量开源的项目，把各种开源的资源充分利用起来，利用开源项目的优势更好地为用户提供技术服务。

InfoQ：IBM Cloud Data Services 中的大部分产品其实已经经过了很长时间的开发演进，已经成为了相对成熟的服务，那么能否请您介绍一下 CDS 的整个发展历程呢？对于 IBM 来说，CDS 的发展思路是怎样的？

吉燕勇：从整个 IT 行业发展的角度来看，在过去几年中已经非常明显地在向云计算、大数据、移动开发，甚至包括社交、安全等领域发展，这些其实都是行业转型的热点。

对于 IBM CDS 来说，我们更关注大数据和云计算两个方向。云计算提供了一种新的交付方式，它可以让企业把更多精力放在业务上。而大数据则可以挖掘出更多的商业价值。

IBM 从 2004 年开始就不断加大投入，花费了将近 200 亿美金，打造出了完善的、丰富的大数据分析能力。在过去，这些投资可能更多地会用于服务于传统的企业应用开发模式，而现在在这样一个开放转型的过程中，我们则希望能把 IBM 强大的结构化数据处理、非结构化处理、以及大数据分析的能力可以通过云服务来提供给企业用户，所以我们今年提出了被称

为 Analytics Platform Services 的战略，并且专门成立了这样的一个部门，以收购来的 Cloudant 等产品为依托，希望能够打造出更好的 PaaS 方面的能力，让更多用户通过云端就可以使用 IBM 丰富的服务。

在今年初宣布向认知计算与云平台转型时，整个 IBM 的大数据分析部门也在向相同的方向转型，今后所有产品线的开发都将围绕着 APS 战略而进行，力图在 CDS 中将 IBM 最具优势的数据处理能力展现出来，这其实对于 IBM 来说也是一次重要的组织架构调整。

InfoQ：在 IBM Cloud Data Services 的发展过程当中，遇到过哪些困难？应该如何去应对这些挑战？

吉燕勇：主要的挑战还是在于开发团队思维模式的转变上。过去十几年我们一直采用的是传统的本地开发模式，虽然同样的大数据分析，但毕竟不是基于云来进行开发。因此，在转型 CDS 的过程中，整个团队需要认真想清楚，我们的这些服务怎么迁移到云端，用户在云端是如何使用这些服务的，用户遇到问题时我们又该如何在云端提供技术支持等这些问题。所以在开发模式以及管理模式的转变上还是花费了相当多的时间的。

除此之外，在转型发展的过程当中，也涉及到了很多新技术的运用，也得到了新的积累，但相比较之下，这些技术上的困难还是比较容易解决的。

InfoQ：您认为像 IBM Cloud Data Services 这样的服务平台，可以使哪些受众获益最多呢？

吉燕勇：数据现在已经变成企业最宝贵的资产之一，无论是企业 IT 人员，还是业务人员，都想利用数据来发掘更多的商业价值，他们都希望通过对数据的分析，能有所回报和收益。而 CDS 其实可以让企业中的所有角色都能获益。

对于企业中的业务人员来说，他们可以直接通过 CDS 里面一些服务和工具来获得具有预测性和指导性的分析能力，同时并不需要他们拥有 IT 专业知识，比如 IBM 刚刚在中国市场发布的基于云的认知计算与数据分析解决方案 Watson Analytics 就是这样的工具，业务人员可以很容易地获得业务洞察力并提升业务运营效率。

对于传统开发人员来说，同样也可以通过 CDS 这种基于云的一站式服务模式，迅速地获取常用的开发工具集合，提升开发速度。

对于传统企业里的 IT 管理人员来说，则可以通过云服务，结合自身内部的一些现有的解决方案，很快打造出一种混合式的服务出来，并让整个企业因此而获益。

对于目前在全球范围内谈得比较多的数据科学家来说，同样很有帮助。CDS 不但提供了大量开源工具与传统工具，还提供了很多独有数据的访问权，数据科学家可以在这样的环境中较快地开展工作。

InfoQ：目前在这个领域当中，同样也有其他的企业推出了相关的产品和服务，那么相比之下，您认为 Cloud Data Services 的不同的之处在于哪些方面？另外在帮助企业上云方面，它又能为企业带来哪些竞争优势呢？

吉燕勇：首先第一点，IBM 正在专注于将传统的数据服务转型为云服务。IBM 在大数据分析领域中经过长时间的内部研发与外部收购，已经形成了非常成熟的大数据分析产品线，因此现在为了适应新的转型趋势，而将 IBM 在大数据领域的技术优势转移到云上，以便更好地服务用户。

其次，IBM 希望通过云计算打造出一种面向企业的平台级服务，企业用户将可以得到全天候的技术支持服务。目前 IBM 是通过美国、英国、中国，三个地区的团队来提供 24x7 的企业级的服务，并对企业用户需求实

现快速响应，为企业用户的业务连续性提供保障。

第三，IBM 一直非常支持开源，在 CDS 平台中，我们将 IBM 自行研发的大数据分析产品与开源项目整合到了一起，比如 Apache Spark cloud service。IBM 希望能将开源的特点，以及 IBM 在大数据领域的优势结合在一起，为用户打造一个比较完善的服务环境。

InfoQ：您刚才提到了平台级的服务，那我们可以看到在 IBM Bluemix 上也提供了一些 IBM CDS 中的服务，您是否可以谈谈 IBM CDS 与 IBM Bluemix 的关系？这两个平台又是怎样合作为用户提供服务的？

吉燕勇：Bluemix 与 CDS 都是 IBM 推出的服务平台，IBM 一开始就在思考如何让这两者进行合作，如何能够让用户更方便、更快捷、更有效地去运用 IBM 的大数据分析能力。

Bluemix 比较侧重于面向开发者来提供服务，在开发过程中可以调用很多现成的服务以实现不同的功能。而 CDS 则是将大数据分析相关服务放到了 Bluemix 中，这样开发者就可以在 Bluemix 平台上直接调用 IBM 的数据分析与处理服务。

InfoQ：您能谈一谈目前 IBM Cloud Data Services 的发展重心在于哪个方面吗？

吉燕勇：目前我们部门的重点工作还是开发和运维，实际上我们的团队正在负责为 IBM 全球用户提供运维支持，并实现不宕机的保障。这里运维工作可能跟传统的运维不太一样，它需要以一种创新、前瞻的思路来考虑可能出现的问题，比如如何能快速发现用户出现了技术问题，如何进行自动监控并处理好这些问题，同时还要和开发部门紧密合作，思考如何能把其他部门开发人员的服务快速通过 DevOps 上线，并解决遇到的问题。

IBM 今年整体要求所有部门都要把重心放在云上，IBM 中国区开发中心同样也不例外。

在重点研发工作上，IBM 中国其实正在扮演着非常重要角色，这其中也涉及到很多核心的专利技术，而这也是我的整个团队的核心价值所在。

InfoQ：想请您谈一谈用户们都非常关心的服务落地问题，目前 IBM Cloud Data Services 中的产品和服务落地情况怎么样？未来还有什么规划？

吉燕勇：我们希望打造的服务平台，首先是要能把之前 IBM 的所有技术能力都放进来，接着让这些能力相互配合并形成一种整体式的服务。我们还准备加速研发，在数据科学以及机器学习等方面加强投入，让这个平台能够更加完善，能为全球用户提供服务。

关于服务落地，我们需要适应并遵从国家相关的数据安全法规。在此基础上，我们会加速 CDS 相关服务的落地，加大与本地企业的合作力度，利用我们团队的技术研发能力尽快解决落地过程中遇到的各种问题，不断推动服务落地的进程。

今年 5 月份通过与 21 世纪互联的合作，Cloudant 已经正式落地中国。而其他服务则将基于中国市场具体的用户需求，排出落地部署优先级，然后再一步步实现落地。

主办方 Geekbang 极客邦科技 InfoQ

ArchSummit
全球架构师峰会 2016

[北京站] 2016年12月2日-3日 北京·国际会议中心

十七大专题，场场精彩.....

FinTech | 微服务与容器实践 | 新闻资讯与广告 | 资源共享新模式 | 高可用架构之道 .....


谢孟军
Go基金会主席


兰建刚
Go基金会主席


王晓波
同程旅游首席架构师


余沛
艺龙网CTO


赵勇
格灵深瞳联合创始人 & CTO


颜世光
百度搜索基础架构团队技术负责人


刘鑫
上海宽睿信息科技有限责任公司 创始人&CEO


魏小强
金火眼创始人


邹光先
爱拍联合创始人&CTO


吴泽明(范禹)
阿里巴巴集团天猫事业部研究员 .....

8折购票 立减千元/团购还享更多优惠（截至10月30日）

bj2016.archsummit.com | 热线咨询：010-89880682 / 18515221946

扫码进入 ArchSummit官网 了解更多大会信息

QCon
全球软件开发大会

主办方 Geekbang. 极客邦科技 InfoQ

解决方案专场

2016年10月20日 | 上海·宝华万豪酒店

宋净超
TalkingData 大数据工程师
《Yarn on Docker——容器技术在大数据场景下的应用》

张 虎
云巴 创始人兼CEO
《如何开发10万在线级别的直播弹幕技术》

郭 炜
易观 CTO
《如何通过APP+Wi-Fi数据给企业做大数据精准用户画像》

费良宏
AWS 首席云计算技术顾问
《程序员的深度学习入门指南》

郑泽宇
才云 首席大数据科学家
《大数据集群在Kubernetes上的实践》

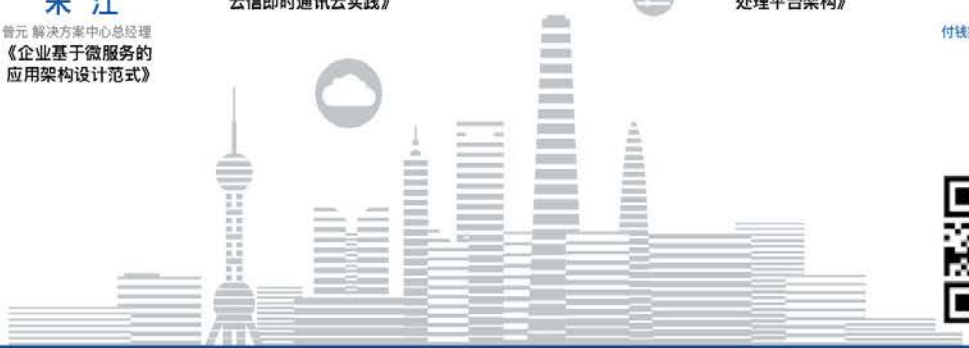
朱 江
普元 解决方案中心总经理
《企业基于微服务的应用架构设计范式》

阙杭宁
网易云信 CTO
《云服务之路，云信即时通讯云实践》

王 丰
极光推送 首席架构师
《实时消息推送架构心得分享》

卜 赫
七牛云 技术布道师
《大规模实时流处理平台架构》

史晓慕
付钱拉 技术研发部技术总监
《付钱拉的API架构演进》



限额免费，干货共享！

精彩一触即发，QCon上海见！

服务咨询 联系人：Rita 电话/微信：13691360511

InfoQ 中文站 2016迷你书

架构师 月刊 2016年9月

本期主要内容：怎样打造一个分布式数据库，10亿级流数据交互查询，为什么抛弃MySQL选择VoltDB，从单体架构迁移到微服务，8个关键的思考、实践和经验，50天10万行代码，一号专车系统重构细节回顾，京东前端：三级列表页持续架构优化，怎样才能叫高级程序员

云生态专刊 08

《云生态专刊》是InfoQ为大家推出的一个新产品，目标是“打造中国最优质的云生态媒体”。

顶尖技术团队访谈录 第七季

本次的《中国顶尖技术团队访谈录》·第七季挑选的六个团队虽然都来自互联网企业，却是风格各异。希望通过这样的记录，能够让一家家品牌背后的技术人员形象更加鲜活，让更多人感受到他们的可爱与坚持。

架构师特刊 大数据平台架构

本期技术特刊中我们总结了酷狗、美团、Airbnb的大数据平台架构实践范例，以及携程、IFTTT、卷皮等公司业务结合大数据平台的架构分析，希望读者能通过不同的角度从中收获到搭建大数据平台知识。