

T O P T E C H T E A M

中国顶尖 CHINA 顶尖

2017年 / 第八季

技术团队

|访|谈|录|



扫一扫，了解更多

InfoQ

Broadview®
www.broadview.com.cn

良好的研发团队文化是 怎样“长”成的

这是一个有关团队文化的话题。

京东从OpenStack切换到 Kubernetes的经验之谈

这个过程中，有这些经验可供业界借
鉴。



来自微信团队的6个开源项目

微信团队2016年在开源上动作不断，InfoQ带你一起看看，他们的开源项目与开源态度。

阿里9年，我总结的前端架 构演进3大阶段及团队管理 心法

听听别人的故事没准也能帮助自己的
成长。

腾讯HTTPS性能优化实践

计算性能的分析和优化；无密钥加
载；证书优化。

从普通程序员到三百人团 队CTO，技术人成长的易 与不易

今天不灌鸡汤，只有最朴实的文字与
回答。

中国顶尖技术团队访谈录 第八季

本期主编 郭 蕾

流程编辑 丁晓昀

发行人 霍泰稳

联系我们

提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com



极客邦科技

InfoQ
ueue

EGO EXTRA GEEKS' ORGANIZATION
NETWORKS

StuQ
斯达克学院

INTRODUCTION | 极客邦科技简介 »

极客邦科技 是一家 IT 技术与学习服务综合提供商，旗下运营 EGO 社交网络、InfoQ 技术媒体、StuQ 斯达克学院职业教育三大业务品牌。致力于整合全球优质学习资源，帮助技术人和企业成长。

企业使命

整合全球优质学习资源，帮助技术人和企业成长

企业愿景

打造全球领先的技术人学习和交流的平台

企业价值观

公开透明、诚实正直、每日精进
乐于服务、负责守诺、创新敢为



极客邦科技十年：

- ◎ 2007 年 3 月 极客邦科技创始人兼 CEO 霍泰稳将 InfoQ 引入中国。
- ◎ 2017 年 3 月 极客邦科技十周年，已拥有 EGO（社交网络）、InfoQ（技术媒体）、StuQ 斯达克学院（职业教育）3 条业务线。累计服务国内超过 100 万的技术人，与超过 300 家企业合作，主办超过 50 场技术大会，走进日美，业务覆盖两岸三地。
- 每天，超过 40 万技术人通过 InfoQ 中国微信公众号了解最新技术趋势与最佳技术实践。
- 每天，超过 100 万技术人与极客邦科技旗下垂直矩阵化的新媒体平台交流互动。
- 每月，超过 1000 人在斯达克学院上学习新课程，掌握实用性 IT 新技能。
- 每年，超过 1 万名中高端 IT 人通过 QCon、ArchSummit 等国际性大会学习最新技术实践，了解最前沿技术趋势，结识业内技术大咖。
- 每年，超过千万的独立账号访问 InfoQ 中国的网站了解国际前沿技术与资讯。
- 每年，EGO 有超过 30 场的学习活动，超过 300 位的国内 CTO、架构师、技术 VP 等高端技术管理者通过 EGO 组织建立紧密联系，分享经验，共同成长。



ABOUT | 关于 InfoQ »

InfoQ 是一家全球性在线新闻 / 社区网站,创立于 2006 年,目前在全球拥有英、法、中、葡、日 5 种语言的站点。

InfoQ 中国于 2007 年由极客邦科技创始人兼 CEO 霍泰稳先生引入中国,同年 3 月 28 日,InfoQ 中文站 InfoQ.com.cn 正式上线。每年独立访问用户超过 1000 万人次。

InfoQ 中国以促进软件开发领域知识与创新的传播为使命,面向 5-8 年工作经验的研发团队领导者、CTO、架构师、项目经理、工程总监和高级软件开发者等中高端技术人群,提供中立的、由技术实践主导的技术资讯及技术会议,搭建连接中国技术高端社区与国际主流技术社区的桥梁。

TECHNOSPHERE | InfoQ 覆盖的技术领域 »



大数据



移动



前端



云计算



架构



研发



AI



运维



容器

PRODUCT | InfoQ 产品 »

垂直社群 聊聊架构、移动开发前线、大数据杂谈、细说云计算、前端之巅、高效开发运维

新媒体 InfoQ 微信矩阵公众号、今日头条、百度百家、微博、直播、短视频

直 播

短 视 频

技 术 大 会 全球软件开发大会

全球架构师峰会

全球移动技术大会

全球容器技术大会



软件正在改变世界
SOFTWARE IS CHANGING
THE WORLD

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年以上工作经验的技术团队负责人、架构师、工程总监、高级开发人员分享技术创新和最佳实践。

卷首语 | QCon大会主编 黄丹 Kitty



明媚的春光下，在QCon北京2017的会场，这期《中国顶尖技术团队访谈录》又与您相遇了。

2007年3月28日，InfoQ中文站正式上线运营。时至今日，InfoQ中文站已经走过了十个年头。十年间，我们一直致力于促进软件开发领域知识与创新的传播，推出了很多内容产品，和读者一起成长。

随着互联网的快速发展，整个技术生态和人才圈都比以前更加活跃，技术

人的价值更加凸显。技术团队的规模也是越来越大。然而，技术和工具每时每刻都在变化，软件系统又是如此复杂，要使团队随时掌握每一种技能几乎是不可能的。一个高效的技术管理者，要找到一种平衡，让团队能够在交付软件系统的同时不断成长；也要想方设法地将知识、思想分享给团队成员。这样的管理者是怎么做的呢？

架构是取舍，管理亦是。每个决定背后都有很多取舍和纠结，都是经过无

升级你的软件开发思维

QCon 北京·2017

数次讨论、推演、决定、优化的过程。在这期访谈录中，有赞联合创始人、CTO 崔玉松谈到了从普通程序员到CTO 的成长之路。而他也来到了 QCon 的现场，和大家聊聊他的心路历程。创业 4 年来，团队从最早的几个人发展到超过 300 人，历经数不清的艰辛和抉择，无论从团队组建、人才招募，还是重大的技术决定、人才引入，亦或是过程管理、战略达成，踩过很多坑，也积累了一些经验。读读文章，听听演讲，或者是当场做一些交流，说不定能获得一些灵感和启发。

提到技术团队成长，自然也离不开技术人个人的成长。不知道大家发现没有，若干年前，很多人说程序员是吃青春饭的，那时候大家经常吐槽程序员干

不到 30 岁；而随着近几年互联网技术与应用的蓬勃发展，大家调侃的阈值也放大到 35 岁到 40 岁。工作 5 ~ 8 年之后，技术人有了一定程度的积累。未来的路应该如何走呢？是选择深耕技术领域，成为资深的技术专家；还是选择成为技术管理者；或是转型，创业？不管选择哪个方向，前人的经验应该对你有所帮助。如果在成长的路上，遇到了技术挑战，或是碰到了发展瓶颈，不妨多找有经验的同行深入交流一下。QCon 北京 2017 的现场，200 位专家，几千位参会者，或许有你的良师益友。

诗仙李白曾云：阳春召我以烟景，大块假我以文章。美好的季节，欣赏风景，阅读文章，启发思想，快哉！

良好的研发团队文化是怎样“长”成的？



作者 张辉清

从死气沉沉到激情活力，从固步自封到好学分享，这是一个有关团队文化 的主题。寺庙文化传承千百年，舌尖上的美食流传至今，它是如何形成和生长的？是参考大公司或从管理书籍上挑选几个词语，还是脚踏实地、土里吧唧，自己一步一步埋头干？本文与你一起探讨！

写在前面

平时经常听到 Google 公司讲企业文化，阿里、华为讲价值观，总觉得牛 B，高大上的东西。几个字就成为公司的核心竞争力，我们能不能自己也搞一个呢？其实文化只是一种思想观念、行为习惯、处事风格，家有家风，国有

国法，一个民族有一个民族的文化。寺庙文化传承千百年，舌尖上的美食流传至今，它应该有一个自己的发展规律。今天，我们一起来探讨一下，研发团队文化是怎么形成的，下面以我个人的经历来抛砖引玉。

我 35 岁前主要钻研技术，近几年逐步转向管理，带过几十人至一两百人

的研发队伍。在最近的管理工作中，总结并归纳了几个可以贴到墙上的大字，即「共治分享自视一起拼，简单有效快」，接下来讲讲这 14 个字的来源。

遇到的问题

我一年多以前入职了现在的单位，以技术负责人的身份。到岗以后发现了不少的问题：

- 1. 技术差：**架构臃肿，分层混乱，不利于维护和扩展；机房建设落后，有两个自建机房，故障处理没流程；
- 2. 气氛差：**团队没有活力，没有争论，没有学习和分享的氛围。准时上下班，人员懒散，没有激情，像一潭死水一样。不接受新事物，封闭自己，遇到问题相互推诿；
- 3. 组织结构不合理：**人员没有充分利用起来，造成资源浪费，团队角色职责定义模糊，整个团队没有一个清晰的目标和落地的步骤，成员相互的合作不协调。

解决之道

有问题当然是解决问题，所有的问题本质上都是人的问题，空降的领导必然会导致一部分人员的流失，只要控制在一定的范围内即可。不是因为你不够好，只是因为他对过去领导的依恋，以及对新事物的不正面。改变一个人的成本，比招聘一个人的成本还要大。所以如果有人要离职，同意即可，让他优雅

地离开，这是于公司于个人都是有利的事情。

解决了人的问题，接着是解决技术问题，组建架构部，引进一些优秀的人才，自己带几个项目，打两场硬仗，培养兄弟感情的同时，也是让大家认同你的最好方式。做完了这些，然后调整组织架构，实行弹性工作制，接下来就是日常的管理工作了。部门共治，搭建统一工作平台，树立新工作作风，搞氛围，激活团队，只能一步一步地来。

部门共治

部门共治就是部门要共同治理，我一直认为，每个人只要管好自己，部门的管理就简单得多。先在文件服务器上建一个文件夹，取名为部门共治，然后在里面新建如周报、周会纪要、工时统计、故障报告等目录（见图 1）。

- 1. 周报**，我个人一直有写周报的习惯，它是自我管理的方式。上一周总结、下一周计划，让自己有一个清晰的工作目标，理清事情的轻重缓急，同时透明化自己的工作，增加彼此的信任。先是自己写，然后分级逐步推广到全员。
- 2. 工时统计**，工时统计是每个月对团队工作时长的统计，由人事协助出报表。它统计到每个人、小组、部门的上班时长情况，主要通过疲劳指数、超出率、小组平均超出率、总平均超出率 4 个指数来体现。
一切以数据说话，增加人员要看小组平均超出率，员工累不累看疲劳

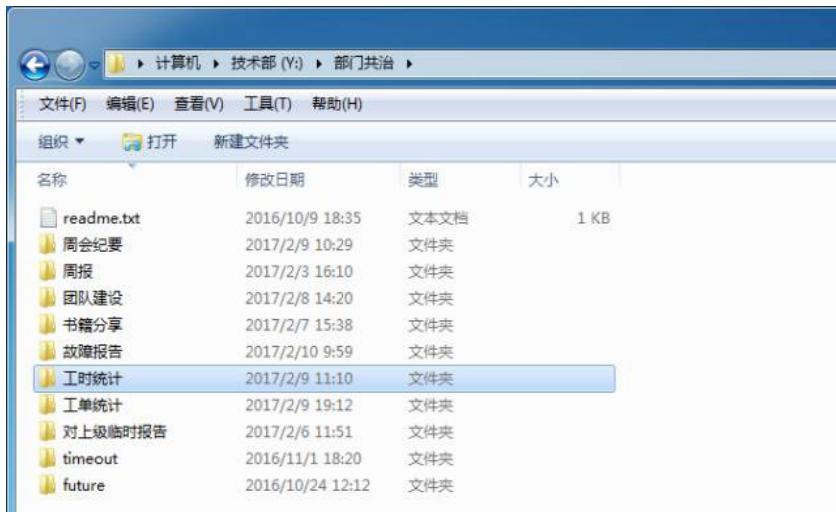


图 1

指数，总平均超出率体现了整个部门的负荷情况，了解整个部门的工作节奏，也为增员或者裁员提供了参考指标，工时统计还可以与工单统计相结合。在具体操作时，需要注意可能产生的负面影响，如造成团队成员的反感、过度鼓励加班等，其实超出率过高或过低都不是很好的现象。

3. 周会，管理小组每周碰一下，汇总一下工作和问题，同时提交会议纪要，全员都可以查看，并邮件方式同步给领导。我们的周会在每周一下午上班的第一时间，并要求开启午休关掉的灯，管理层从自己做起，严格约束自己，给大家立一个榜样，顺带还解决午休延迟开灯的问题。

早期的周会由我主持，后面轮流主持和记录。主持人站在部门负责人

的角度，对汇报人员提出质疑或建议，这样就迫使站在整个研发中心的角度去思考，培养了每个人的全局观，减少本位主义和平时工作中不必要的冲突，同时还提高了管理人员的组织协调力。

4. 故障分析报告，将故障分级管理，然后尽量透明化，重要故障全员通告，此工作由运维部门负责。故障报告不仅可追溯，也是一个总结分析的过程，可协助查找到根本问题。透明化可以提高大家的警惕，造成的损失也间接地告诉了代码与商业价值的关系。

5. 书籍分享和书斋建设，如果要构建一个学习型团队，书籍的分享和书斋的建设是很好的办法。同时它相当于一面文化宣传墙，利于人员的招聘。书籍可以公司出钱购买，也可以鼓励大家捐赠、分享。书籍它

本身很便宜，但真的要捐赠时也有不舍，所以鼓励大家离开时可赎回。捐赠和购书、以及分享书籍的过程，也是学习氛围养成的过程。

搭平台，立作风

搭工作平台，立新工作作风。没有度量就没有提升，管理要数据化、工具化。搭建统一的工作平台，提高大家的工作效率，同时还要树立新的工作作风，如站立式会议、周五分享日、项目管理工具和知识管理工具。

- 1. 项目管理工具JIRA**，引入JIRA项目管理工具，可以管理需求、工单、缺陷等各种任务。可以实时地监控、跟踪任务的状态，还可以在每月底出工单统计报告，评估开发人员的工作量和测试人员的工作量。
- 2. 知识管理工具WIKI**，类似于维基百科，主要用于知识的分享。可以创建部门的空间、自己个人的空间，写自己的工作心得、新的技术知识等。我个人的一些心得和重要邮件，我也会截图分享到自己的个人空间，这有利于新成员的了解和合作，也利于团队文化的形成。以前企业文化搞海报、杂志等方式，现在采用这种方式更容易接受，也更容易推广。
- 3. 站立式会议**，研发是一个需要创意，大脑密集性的工作。所以技术讨论是必不可少的部分，而站立式会议则是非常好的形式。小组周会或者几个人讨论一个问题，非常适

合站立式短会。20分钟左右，大家站在白板前，不需要做会议记录，直接用手机拍照，会后发邮件或QQ群。简单高效快，大家保持专注，也不会懒洋洋的样子，没人会玩手机。因为站着，所以会议不会太长，高效、方便、随时随地，不一定需要会议室，所以研发中心要多一些白板，多一些小的会议室。

周五分享日，技术的分享现在已成为研发团队的标配，我们每周五是技术分享日，时间定在下午 17 点到 19 点，这样避免影响正常的工作。同时让大家知晓，成长不仅是团队的事情，也是个人的事情。分享有利于主讲人知识的巩固、总结，成果的展示，也有利于大家快速的学习。架构部的分享是工作的一部分，其它的则是自愿报名，自由组织。我们的分享也要采用简单有效快的方式，鼓励直接 show 代码，或使用 wiki 来讲，这样也利于二次传播，不一定要高大上，也不一定要 PPT，达到目的和起到效果即好。

搞氛围，激活团队

工作和生活要平衡，认真工作，快乐生活。如果管理是鞭策或大棒，那么领导就是鼓励或葫芦卜。季度会议、乐捐与周四下午茶、户外兴趣小组是搞氛围、激活团队的很好方式。

- 1. 季度会议**，周会说事，季度会议谈情怀。每3个月左右开一次季度会，套路是新人介绍，上季度的总结，下季度的计划，其他如新书推

荐、特长展示等。新人介绍可以快速地融入整个部门，感受团队的热情。上季度的工作总结和下季度的计划，是每个部门长向下级汇报，我们鼓励上级向下级定期汇报，总理都要每年向全国人民汇报，我们也要这样，让每个人都知道各小组最近在干些什么事，下阶段要做什么，设置一个共同的目标，然后一起拼搏。新书推荐，主持人会向大家介绍近期自己读过的书，以及读书的感悟与心得，一起成长。

季度会议如同年度会议，年会不开会，只是吃吃喝喝，以及节目表演，所以季度会议的形式也要比较轻松，以乐观向上的氛围为主，参与的人数比较多。我们的季度会议采用轮值，每次的风格都不一样，这样挺好的，也非常锻炼主持人的组织协调能力，会议时间一般占用周五分享日的时间。

- 2. 乐捐与周四下午茶**，乐捐是一个自罚约定、一个问责制度，让员工敢于担当。对于开会迟到、系统小故障、乐捐10元能够起到很好的作用。惩罚只是手段，改进才是目的，培养承担责任的习惯，每次出现问题，都有人站出来承担责任。可别小瞧10元，10元只是最低额，额度自定，乐捐它是一个很好的追责方式，故障更容易明确主要责任人。明确责任后，个人自愿，如果你不愿意10至50元，则由他的直接

领导来承担，这是一种社区式的潜在力量。

乐捐不可强制，如会议迟到了你不愿意承担，那便需要由会议主持人承担。乐捐的钱会汇入到部门基金，用于部门的下午茶。我们每周举办一次下午茶，周四进行，因为周一例会日、周二发布日、周三无会议日、周五又是分享会。下午茶给大家一个轻松的时间，促进交流，营造气氛，费用由个人乐捐、项目经费共同组成。

- 3. 户外活动兴趣小组**，工作生活Balance，兴趣小组能够促进大家的交流，非正式的沟通能改善大家的关系，增强团队的凝聚力。我们组建了羽毛球兴趣小组、户外徒步兴趣小组，以及参与公司的年会表演，组织形式是开放的，是弱化了公司和部门信息的QQ群。兴趣小组有利于业务与技术的融合、跨部门沟通，大家劳逸结合，一起爬山一起拼。

更多管理工具

当然不仅仅是以上，我们每一个管理工具都是针对当前的问题，推行2至4周，一个阶段仅推广一个管理工具。我们尝试并留下来的管理工具总共有20多个，具体有：“周会周报、弹性工作制、技术分享会、季度会议、下午茶、10元自罚约定、白板前站立式会议、技术评审制度、会议轮流主持制度、书斋建设、项目管理制度的推行、项目奖金



申报制度、招聘比自己更优秀人的约定、末尾淘汰约定、鼓励争论、跨部门沟通约定、源代码统一管理、Y 盘文档管理、JIRA 和 WIKI 研发管理工具推广、羽毛球兴趣小组、户外爬山兴趣小组、个体绽放，周一例会日、周二发布日、周三无会议日、周四下午茶、周五分享日”。

总结与提升

太阳底下没有新鲜事，这些工具或措施，或多或少都知道些，我们只是把它落地贯彻，形成了一定的体系。我们的团队一步一步往前走，确实越来越好。这时不断有新人加入，如何将这过去一年多的想法和做法都留下来，并传达出去呢？我们需要把它浓缩成几个字才好，这就是前面提到的“共治分享自视一起拼，简单有效快”，具体内容如下：

共治：共治就是部门要共同治理，自我管理和部门管理一起进行，每个人要管好自己，部门的共治我们希望能转

化为个人的自治。具体形式有部门共治文件夹、轮值会议等；

分享：因为专业所以自信，因为自信所以开放，因为开放所以分享。具体形式有周五分享日、书斋建设；

自视：自视就是自我察觉，双眼反向注视着自己。对公司和他人的要求转化为对自己的要求，少一些抱怨，多找一些方法，如乐捐；

一起拼：一起拼就是团队要一起拼搏，一起奋斗，一起喝酒、一起爬山，如户外、项目管理；

简单有效快：把复杂的事情变简单，追求事物的本质，这就是简单有效。“快”是指快速地交付，快速响应业务的需求，如站立式会议等。

以上是我们近一年的总结和实践，也许它并不完美，但它还是能表达团队之前及现阶段的一些行为。当我们确定了这个阶段的部门文化后，团队氛围有了较大的改善，从死气沉沉到激情活力，

“我们不能简单地参考大公司的做法，或者管理书籍上挑选几个词语，然后领导喊两声或挂在墙上。”

从固步自封到好学分享，从互相推诿到勇于担当。在后期的几个大项目中，项目的成功显得自然得多，特别在新员工的融合上，成本也低得多。

“长”出来的团队文化

可能现在你已发现，原来文化的构建并没有想象地那么难，也没有多少高大上。它也需要脚踏实地、土里吧唧，一步一步埋头干。先有管理工具、制度和行为措施，然后予以贯彻，形成一种习惯，最后才是总结提升。

我们不能简单地参考大公司的做法，或者管理书籍上挑选几个词语，然后领导喊两声或挂在墙上。这个过程就如同花朵或生物一般，需要播种、栽培，然后才能收获。这样“长”出来的文化，才能管人做事，深入骨髓、改变思想，才能成为公司或团队的核心竞争力。以

上浅见，希望能给你一些参考，当然，你也可以尝试一把，Just do it!

作者介绍

张辉清，中青易游 CTO，曾就职于携程架构、古大集团，在古大集团任首席架构师和高级技术总监。现阶段关注技术与业务的融合、工程效率和团队管理。

京东从OpenStack切换到Kubernetes的经验之谈



作者 鲍永成

京东从 2016 年底启动从 OpenStack 切换到 Kubernetes 的工作，截止目前已迁移完成 20%，预计 Q2 可以完成全部切换工作。Kubernetes 方案与 OpenStack 方案相比，架构更为简洁。在这个过程中，有这些经验可供业界借鉴。

背景介绍

2016 年底，京东新一代容器引擎平台 JDOS2.0 上线，京东从 OpenStack 切换到 Kubernetes。到目前为止，JDOS2.0 集群 2w+Pod 稳定运行，业务按 IDC 分布分批迁移到新平台，目前已迁移 20%，计划 Q2 全部切换到 Kubernetes 上，业务研发人员逐渐适应从基于自动

部署上线切换到以镜像为中心的上线方式。JDOS2.0 统一提供京东业务，大数据实时离线，机器学习（GPU）计算集群。从 OpenStack 切换到 Kubernetes，这中间又有哪些经验值得借鉴呢？

本文将为读者介绍京东商城研发基础平台部如何从 0 到 JDOS1.0 再到 JDOS2.0 的发展历程和经验总结，主要

包括：

- 如何找准痛点作为基础平台系统业务切入点；
- 如何一边实践一边保持技术视野；
- 如何运维大规模容器平台；
- 如何把容器技术与软件定义数据中心结合。

集群建设历史

物理机时代（2004-2014）

在 2014 年之前，公司的应用直接部署在物理机上。在物理机时代，应用上线从申请资源到最终分配物理机时间平均为一周。应用混合部署在一起，没有隔离的应用混部难免互相影响。为减少负面影响，在混部的比例平均每台物理机低于 9 个不同应用的 Tomcat 实例，因此造成了物理机资源浪费严重，而且调度极不灵活。物理机失效导致的应用实例迁移时间以小时计，自动化的弹性伸缩也难于实现。为提升应用部署效率，公司开发了诸如编译打包、自动部署、日志收集、资源监控等多个配套工具系统。

容器化时代（2014-2016）

2014 年第三季度，公司首席架构师刘海锋带领基础平台团队对于集群建设进行重新设计规划，Docker 容器是主要的选型方案。当时 Docker 虽然已经逐渐兴起，但是功能略显单薄，而且缺乏生产环境，特别是大规模生产环境的实践。团队对于 Docker 进行了反复

测试，特别是进行了大规模长时间的压力和稳定性测试。根据测试结果，对于 Docker 进行了定制开发，修复了 Device Mapper 导致 crash、Linux 内核等问题，并增加了外挂盘限速、容量管理、镜像构建层级合并等功能。

对于容器的集群管理，团队选择了 OpenStack + nova-docker 的架构，用管理虚拟机的方式管理容器，并定义为京东第一代容器引擎平台 JDOS1.0（JD DataCenter OS）。JDOS1.0 的主要工作是实现了基础设施容器化，应用上线统一使用容器代替原来的物理机。

在应用的运维方面，兼用了之前的配套工具系统。研发上线申请计算资源由之前的一周缩短到分钟级，不管是 1 台容器还是 1 千台容器，在经过计算资源池化后可实现秒级供应。同时，应用容器之间的资源使用也得到了有效的隔离，平均部署应用密度提升 3 倍，物理机使用率提升 3 倍，带来极大的经济收益。

我们采用多 IDC 部署方式，使用统一的全局 API 开放对接到上线系统，支撑业务跨 IDC 部署。单个 OpenStack 集群最大是 1 万台物理计算节点，最小是 4K 台计算节点，第一代容器引擎平台成功地支撑了 2015 和 2016 年的 618 和双十一的促销活动。至 2016 年 11 月，已经有 15W+ 的容器在稳定运行。

在完成的第一代容器引擎落地实践中，团队推动了业务从物理机上迁移到容器中来。在 JDOS1.0 中，我们使用的

IaaS 的方式，即使用管理虚拟机的方式来管理容器，因此应用的部署仍然严重依赖于物理机时代的编译打包、自动部署等工具系统。但是 JDOS1.0 的实践是非常有意义的，其意义在于完成了业务应用的容器化，将容器的网络、存储都逐渐磨合成熟，而这些都为我们后面基于 1.0 的经验，开发一个全新的应用容器引擎打下了坚实的基础。

新一代应用容器引擎（JDOS 2.0）

1.0的痛点

JDOS1.0 解决了应用容器化的问题，但是依然存在很多不足。

首先是编译打包、自动部署等工具脱胎于物理机时代，与容器的开箱即用理念格格不入，容器启动之后仍然需要配套工具系统为其分发配置、部署应用等等，应用启动的速度受到了制约。

其次，线上线下环境仍然存在不一致的情况，应用运行的操作环境，依赖的软件栈在线下自测时仍然需要进行单独搭建。线上线下环境不一致也造成了一些线上问题难于在线下复现，更无法达到镜像的“一次构建，随处运行”的理想状态。

再次，容器的体量太重，应用需要依赖工具系统进行部署，导致业务的迁移仍然需要工具系统人工运维去实现，难以在通用的平台层实现灵活的扩容缩容与高可用。

另外，容器的调度方式较为单一，

只能简单根据物理机剩余资源是否满足要求来进行筛选调度，在提升应用的性能和平台的使用率方面存在天花板，无法做更进一步提升。

平台架构

鉴于以上不足，在当 JDOS1.0 从一千、两千的容器规模，逐渐增长到六万、十万的规模时，我们就已经启动了新一代容器引擎平台（JDOS 2.0）研发。JDOS 2.0 的目标不仅仅是一个基础设施的管理平台，更是一个直面应用的容器引擎。JDOS 2.0 在原 1.0 的基础上，围绕 Kubernetes，整合了 JDOS 1.0 的存储、网络，打通了从源码到镜像，再到上线部署的 CI/CD 全流程，提供从日志、监控、排障、终端、编排等一站式的功能。JDOS 2.0 的平台架构如图 1 所示。

在 JDOS 2.0 中，我们定义了系统与应用两个级别。一个系统包含若干个应用，一个应用包含若干个提供相同服务的容器实例。一般来说，一个大的部门可以申请一个或者多个系统，系统级别直接对应于 Kubernetes 中的 namespace，同一个系统下的所有容器实例会在同一个 Kubernetes 的 namespace 中。应用不仅仅提供了容器实例数量的管理，还包括版本管理、域名解析、负载均衡、配置文件等服务。

不仅仅是公司各个业务的应用，大部分的 JDOS 2.0 组件（Gitlab/Jenkins/Harbor/Logstash/Elastic Search/Prometheus）也实现了容器化，



图 1

在 Kubernetes 平台上进行部署。

开发者一站式解决方案

JDOS 2.0 实现了以镜像为核心的持续集成和持续部署（见图 2）。

1. 开发者提交代码到源码管理库
2. 触发 Jenkins Master 生成构建任务
3. Jenkins Master 使用 Kubernetes 生成 Jenkins Slave Pod。
4. Jenkins Slave 拉取源码进行编译打包
5. 将打包好的文件和 Dockerfile 发送到构建节点
6. 在构建节点中构建生成镜像

7. 将镜像推送到镜像中心 Harbor
8. 根据需要在不同环境生产/更新应用容器

在 JDOS 1.0，容器的镜像主要包含了操作系统和应用的运行时软件栈。APP 的部署仍然依赖于以往运维的自动部署等工具。在 2.0 中，我们将应用的部署在镜像的构建过程中完成，镜像包含了 APP 在内的完整软件栈，真正实现了开箱即用（见图 3）。

网络与外部服务负载均衡

JDOS 2.0 继承了 JDOS 1.0 的方案，采用 OpenStack-Neutron 的 VLAN 模式，

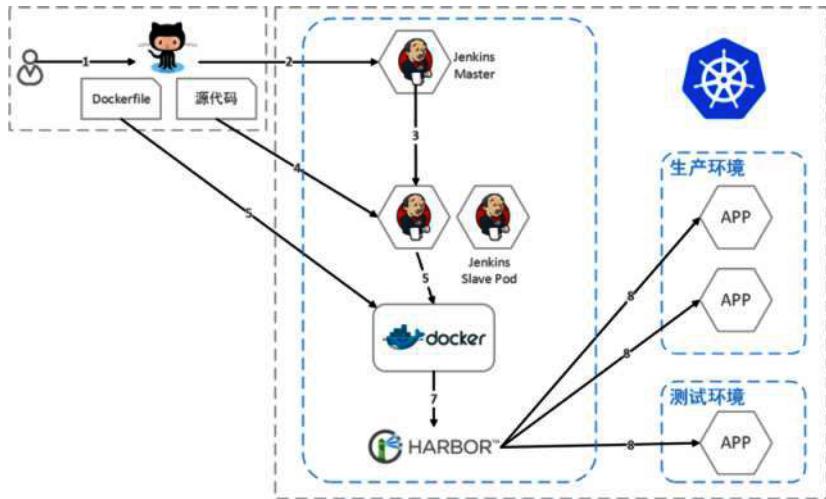


图2

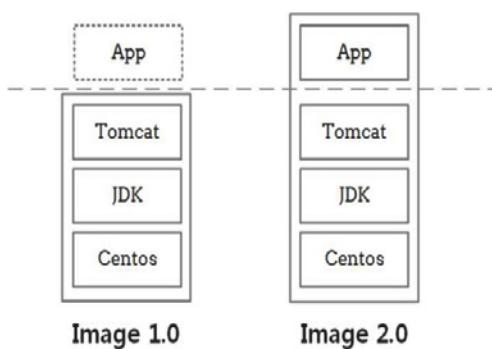


图3

该方案实现了容器之间的高效通信，非常适合公司内部的集群环境。每个 Pod 占用 Neutron 中的一个 port，拥有独立的 IP。基于 CNI 标准，我们开发了新的项目 Cane，用于将 Kubelet 和 Neutron 集成起来。

同时，Cane 负责 Kubernetes 中 service 中的 LoadBalancer 的创建。

当有 LoadBalancer 类型的 service 创建 / 删除 / 修改时，Cane 将对应的调用 Neutron 中创建 / 删除 / 修改 LBaaS 的服务接口，从而实现外部服务负载均衡的管理。另外，Cane 项目中的 Hades (<https://github.com/ipdcode/hades> 京东开源在 GitHub 上) 组件为容器提供了内部的 DNS 解析服务。

灵活调度

JDOS 2.0 接入了包括大数据、Web 应用、深度学习等多种类型的应用，并为每种应用根据类型采用了不同的资源限制方式，并打上了 Kubernetes 的不同标签。基于多样的标签，我们实现了更为多样和灵活的调度方式，并在部分 IDC 实验性地混合部署了在线任务和离线任务。相较于 1.0，整体资源利用率提升了约 30%（见图 4）。



图4

推广与展望

有了 1.0 的大规模稳定运营作为基础，业务对于使用容器已经给予了相当的信任和支持，但是平台化的容器和基础设施化的容器对于应用的要求也不尽相同。比如，平台化的应用容器 IP 并不是固定的，因为当一个容器失效，平台会自动启动另一个容器来替代，新的容器 IP 可能与原 IP 不同。这就要求服务发现不能再以容器 IP 作为主要标识，而是需要采用域名，负载均衡或者服务自注册等方式。

因此，在 JDOS2.0 推广过程中，我们也推动了业务方主要关注应用服务，减少对单个容器等细节的操作，以此自研了全新智能域名解析服务和基于 DPDK 高性能负载均衡服务，与 Kubernetes 有效地配合支持。

近两年，随着大数据、人工智能

等研发规模的扩大，消耗的计算资源也随之增大。因此，我们将大数据、深度学习等离线计算服务也迁移进入 JDOS2.0。目前是主要采用单独划分区域的方式，各自的服务仍然使用相对独立的计算资源，但是已经纳入 JDOS2.0 平台进行统一管理，并通过机器学习方法，提升计算资源使用效率。

灵活的标签给予了集群调度无限的可能。未来我们将丰富调度算法，并配以节能的相关技术，提高集群整体的 ROI，从而为打造一个低能耗、高性能的绿色数据中心打下基础。

回望与总结

Kubernetes 方案与 OpenStack 方案相比，架构更为简洁。OpenStack 整体运营成本较高，因为牵涉多个项目，每个项目各自有多个不同的组件，组件之

间通过 RPC（一般使用 MQ）进行通讯。为提高可用性和性能，还需要考虑各个组件的扩展和备份等。这些都加剧了整体方案的复杂性，问题的排查和定位难度也相应提升，对于运维人员的要求也相应提高。

与之相比，Kubernetes 的组件较少，功能清晰。其核心理念（对于资源和任务的理解）、灵活的设计（标签）和声明式的 API 是对 Google 多年来 Borg 系统的最好总结，而其提供的丰富的功能，使得我们可以投入更多精力在平台的整体生态上，比如网络性能的提升、容器的精准调度上，而不是平台本身。尤其是，副本控制的功能受到了业务线上应用运维工程师的追捧，应用的扩容缩容

和高可用实现了秒级完成。JDOS 2.0 目前已经接入了约 20% 的应用，部署有 2 个集群，目前日常运行的容器有 20000 个，仍在逐步推广中。

真诚感谢 Kubernetes 社区和相关开源项目的贡献者，目前京东已经加入 CNCF 组织，并在社区排名达到 TOP30。

作者介绍

鲍永成，京东基础平台部技术总监，带领基础平台部集群技术团队从 2014 年上线京东容器引擎平台 JDOS1.0 到现在的 JDOS2.0，作为坚实的统一计算运行平台承载京东全部业务稳定运行。目前主要工作方向是 JDOS2.0 研发和京东第一代软件定义数据中心建设。



阿里9年，我总结的前端架构演进3大阶段及团队管理心法



作者 赵振宇

技术人生就是在不断地修行，每个人都有每个人的功课，每个人也有每个人的精彩。你也许刚上路，又或许踽踽独行了很久，听听别人的故事没准也能帮助自己的成长。在阿里修行的9年，他学会了这些。

少年励志，初入技术圈

我生在一个文化气息浓厚的家庭，这让我从小就对艺术有了一种懵懂的向往。第一次接触到计算机时，我就明白自己会在这个领域玩下去；第一次接触到互联网时，我就坚定了将其作为事业，把自己的黄金年龄投入其中的信念。文化气息的熏陶和坚定的信念，使得我踏

上了寻找将美好的设计感和互联网技术相结合的长路上。

2004年，我在上海毕业后，加入了COSCO。当时的环境有很好的应用场景和很牛的前辈，我在这个时期接触到了一种可以被叫做前端的职业，接触到了这个把“人的情感设计”和“技术的实现”连接在一起的令人兴奋的事情，而这恰恰就是我当初最希望做的事情，于是毫

不犹豫的将其作为自己立足的根本。除了找到自己的定位，我在这里收益最大的是**快速学习并建立了自己的职场价值观**。

后来，我赶上了第二波互联网浪潮。我不是那种希望很早就进入退休状态的人，所以在 2007 年，我毅然决然地投身到 Web 2.0 的创业过程中。半年的时间，我们创业的产品就在当时的社区中具备了一定的知名度。这短短的半年时间，让我在**技术、设计、产品、用户**等方面都有了不错的知识积累。

2008 年，我们三个技术人的创业进入到了瓶颈期，我也明白了自己需要进一步学习的有哪些。此时，机缘巧合，我以一个普通的前端工程师的身份加入了阿里巴巴。

阿里九年，我所到达的四个站点

时光如梭，一晃眼，我加入阿里巴巴已快 9 年，这 9 年中，我经历了 4 个重要的阶段。

第一站：UED 团队前端工程师

开始 2 年，我在 UED 团队。作为一线的前端工程师，我参与了 UED 深度进入产品，强烈追求业务结果的发展时期。这为我深深地打下了用数据从用户行为路径的角度优化产品的思路。这使得我从进入前端正规军的第一站开始，就有了一个需要站在比前端职责更大的角度去考虑如何工作的环境。这是我职业顺利发展最为关键的因素。这个阶段，我

专注的领域在性能和体验优化上。

第二站：UED 团队前端团队 Leader

2010 年，由于所在 UED 团队的调整，我成为了前端团队的 Leader。在此阶段，我参加了 AliExpress 的初创团队建设，又回归到国际事业部。在负责了整个 UED 团队管理一年后，我又回归到前端团队，同时也把前端团队带进了技术团队。这个阶段，让我明白了前端这个角色的本心和这个阶段所需要的环境，使得我心中的前端开始与技术团队形成连线。

第三站：阿里巴巴企业事业群 (B2B) 的前端团队 Leader

2014 年，随着团队的规模进一步扩大，我开始负责阿里巴巴企业事业群 (B2B) 的前端团队，主要的职责是事业群下的前端团队的整合和技术的收拢。这使得我必须在更大、更复杂的环境下，来看待前端团队的定位。这个阶段，让我有机会思考企业架构下前端的价值和定位。

第四站：B 类电商体验技术中台团队 Leader

2016 年，在承接集团中台建设的战略后，我思考清楚并开始实践研发中台的建设，并略有成效。在中台急切的落地的期待下，我加入到 B 类电商中台团队，进入到了当前的第四站，负责体验技术中台团队。现在，我更多关注的课题是：如何在成熟的系统中去做收敛和

统一，并为将来留下灵活性。当然，我依旧是从前端的角度切入。

我所经历的B类电商平台前端架构演进

我不敢妄言整个互联网电商平台的前端架构，因为没有全部经历过。但我在阿里巴巴B2B电商平台的这段经历，使得我有立场说一说B类电商平台的前端架构演进。

架构和业务的发展有密不可分的关系，下面我将结合业务发展阶段对应的谈前端架构在当下环境的特点和时间线上的演进。

“信息透出，促成双方会面”阶段

在电商平台还在「信息透出，促成双方会面」阶段的时候，业务的主要特征是以搜索 / 导购作为主线，用户链路以在线沟通意向为终点。业务模式很简单，抽象起来就是用户提供信息，电商平台展示信息，协助买卖方在线沟通。在这个阶段，前端的架构视角的关键词是：**继承式代码复用，加载期性能治理**。

继承式代码复用一般遵循着：基础框架 / 运行时；组件基类；基础组件父类；业务组件实例这样的一个技术架构模型上。而在工程架构上的特点是：覆盖式发布（含中美同步）；基本依赖管理；性能治理（压缩，去重合并，监控）。

大家会发现，这个阶段考虑的都是

通用性问题。抛去电商的业务因素，可以发现这是个放之哪里都能用的架构，**解决前端自身在研发过程中的问题占了绝对比重**。

“在线交易达成”的阶段

在电商平台进入到「在线交易达成」的阶段时，业务的主要特征就是：开始期望用户把围绕着交易的所有事务工作在线化；业务场景在之前的基础上新纳入各种复杂的用户端信息管理（交易流程、纠纷流程、生意关系管理等）；用户在平台上需要完成的操作开始变得更多，使得人机交互场景变得更频繁且有更高体验要求。另外，多人协同研发的局势也越来越明显。在这个阶段，前端的架构视角的关键词是：模块化管理，前后端分层，执行期性能治理。

这个阶段，应用场景出现了频繁的数据交换需求，从而开始注重前后端的通信管理以及工作解耦，从而探索前端的分层架构；模块化管理的进入引入了模块管理器（离线 / 在线）；发布前构建，从而引入工程链的体系。因为开发环节的进一步复杂，对于质量治理，线上线下监控告警等都开始有意识的进一步完善。

大家还是会发现，该阶段考虑的虽然还是属于通用性问题，而解决的问题已经逐渐进入到大型复杂协同模式下需要面对的问题了，慢慢从框架之争进入到到了如何更好的组织代码以及探索前端职责边界。

电商平台进入真正的平台化阶段

在电商平台进入真正的平台化，需要在目前的基础上快速接入第三方服务（海关，税务，银行），快速建设垂直市场（垂直行业市场，封闭市场，大企业采购市场等），需要考虑的是如何解决大量不可预知的差异化的承接问题，这个问题已经需要站在整体企业架构中才能去尝试解决的了。在这个阶段，前端的架构视角的关键词是：**设计解构、逻辑调度、应用模型、分层标准化**。

随着业务的多样性发展，整体架构的服务治理工作的开展以及人力紧张状态的持续，我们需要进一步的去识别研发过程中能够被进一步抽象和标准化的部分，并将变化的部分通过可配置、可编排的方式提供灵活的服务，并赋予业务实施的过程；需要进一步强化和明确在系统架构和信息架构间前端的转化工作。

该阶段考虑的部分逐渐破开前端的固有视角，开始从业务特征的视角，从数据消费者的视角，从用科学方法来解构专业的视角，来建设前端的能力。同时，也开始从前端自身的开发架构慢慢转变到有一定业务特征的应用架构。

我看大型企业级架构前端分层

分层目的

分层的目的从根本上说有两点：

- 解耦前后端开发工作；
- 通过分层降低单层的复杂度。

这两点最终都能够反应到效率上：

一个效率点是瀑布式开发转变成并行基于接口的开发，缩短开发任务路径；另一个效率点是在面对业务调整时，降低复杂度的分层系统具备更强的灵活性，从而提升整个系统的响应效率。

另外，分层对稳定性也有一定意义的帮助，基于层的测试能够做的更加纯粹和有针对性。

同时，接口调用性能监控和全链路的性能监控在其中非常关键，用来做因为分层带来的额外性能开销可能存在风险的监控。

主流模型

主流的模型有两种：

- 客户端渲染 + 应用模型数据聚合
- 客户端展现 + 服务端渲染 + 应用模型数据聚合

那么，应用模型数据聚合（为 UI 提供数据）和服务端渲染用什么实现呢？

客户端在承担越来越多职责的情况下会进一步的引起思考，一些计算任务放置到服务端会更合适；而业务领域模型的数据无法直接有效的对 UI 服务，需要有一层来处理数据消费者视角的数据加工工作。

结合“同构”的意义，我们会发现 JavaScript 有了更大的应用场景。而我的观点是 JavaScript 的运行时有了更大的应用场景，故不论是 Node.js、Nashorn（甚至以前的 Rhino），亦或是直接使用 V8 都是可以做到我们想要做



的事情，而让 JavaScript 跑在服务端之外，整件事更应该关注的是稳定性，容灾容错，弹性，监控告警这些我们现在还有些陌生的领域。选型这件事不是语言偏好和角色之争，而是系统应该做成什么状态的思考。

分层通用原则

分层的通用原则有：

- 每一层完成独立的功能，每层能够独立演进，独立部署，独立测试。
- 每一层的功能可依赖与处在同一层或下一层的功能，避免系统复杂度过高。
- 每一层功能的接口定义与接口实现要分离，对该层的访问只能通过接口。

分层架构通过将事务处理的分层来分化系统的复杂性，提高系统的可扩展和可维护性。但同时因为分层事务处理导致需要在多个层间传递能力，会导致性能的损耗。

目前谈论的前端分层主要是集中在 presentation layer 和 business

layer 中的一部分。

“每一层功能的接口定义与接口实现分离，对该层的访问只能通过接口”，这个原则对前端的分层同样有很强的指导意义，太多的不兼容底层框架升级导致业务实施有太多的额外成本开销。这个恰恰是分层架构给我们前端在本职工作上需要的更多思考。

国际化站点的前端业务的特点

国际化站点的业务对于前端而言，最显而易见的第一个问题是国际化部署；然后是内容的国际化管理；再往后是目前还在尝试中的本地化。

国际化部署对于前端而言不仅仅是把资源文件同步分发到全球各地机房就结束了的，源站和全球 CDN 中就有非常多的治理工作，比如 CDN 预热、热区规划、缓存版本管理等；而且资源文件和应用的发布在全球化发布过程中会被无限放大发布时间差的问题，应用的跨版本平滑发布，配置和文件幂等检测等。

内容的国际化管理，首先是语种的

国际化管理、用集中式键值对管理、热部署、页面内容识别等方法解决常用的应用中 XML 资源文件式的管理存在的管理困难，分散在应用中而带来的冗余，发布困难，多语种应用定位等问题；其次是类似“单位 / 日期 / 货币 / 姓名格式”等更精细化的国际化差异系统级管理。当然这里还有些挺特殊的例子，比如阿拉伯语、希伯来语的从右到左书写方式。

本地化区别于国际化，更注重在本土文化的差异性上，目前还在探索。

我看大型企业中前端团队的管理

关于团队管理的问题，没有正确答案。组织结构是在承接战略而灵活变动的，而其中的优劣也是相对而言。作为管理者，需要解决的恰恰是**享受了优势之后随之而来的问题，因为任何问题都会成对出现**。

例如，是否应当将前端的同学合拢在一个团队呢？我们可以从专业建设、视角和组织灵活性来分析一下。

专业建设

合拢成一个团队时，前端专业氛围，对于前端个体而言，是一个较好的环境，但是在整体技术体系的建设上有一定局限性。

相反，分散到业务中，应用技术体系的专业建设，跨领域的知识储备，一些组织问题带来的隔阂会天然消失，对

于业务开发者而言是一个较好的环境。但是专业发展可能会成为问题。

视角

合拢成一个团队时，能够将所有前端参与的业务信息集中在一起，能够让这个团队（核心人员）有好的业务全局视角，但是对单个领域的了解和理解深度会有一定影响。

分散到业务中，对当前业务领域能够有更为深入的了解和理解，能够在特定领域中创新出特定的解决方案，但是缺乏更大纬度全局视角时会有一定的判断限制（业务判断和技术判断）。

组织灵活性

- 合拢成一个团队时，同一职能在不同业务中的组织灵活性强，能够在业务的张弛中相互协调，但是容易被频繁的资源协调工作占满自己的日程。
- 分散到业务中，根据业务域进行专门的支撑，整体的目标感，沟通效率等跨职能的协同角度会更具优势，但是职能角色的资源灵活性就会受限。

不论以哪种方式来组织前端的同学，都需要让组织更加灵活，相应的关键因素有两个：

- 前端在遵循统一的技术基础之上一起建设支撑业务的统一基础能力。
- 前端一致的角色价值认知。

这里统一的技术基础和一致的角色价值认知，一实一虚，对应着技术的基

础和角色的文化：

统一的技术基础，能够让一个组织中的前端在一个基础上展开工作，也能让前端的同学进入不同业务时，开发的基础是一致的，也有讨论技术时相对一致的谈话基础。统一的技术基础可以让业务领域的实施过程可以不过多关注底层技术实现，而更多的聚焦在业务解决方案的设计和实现上，可以不断的沉淀出更有效的业务的基础能力出来。

一致的角色价值认知是一个团队一个角色统一的文化，比如在我的团队，

“链接商业，设计，计算能力，为用户提供专业的人机交互体验”。这是让前端们能够坚持前端的初心，而不会因为身处不同团队，在做不同业务的工作时出现一些发展的迷茫。

前端群体发展方向：“云”和“端”

“泛前端”或“大前端”的概念喊了很多年了，我认同这两个词，但有一些我自己的逻辑。前端当初出现的原因是“人机交互体验”，用什么技术用什么语言去实现这个人机交互过程并不关键，但理念和目标应该是一致的，甚至在整个知识树中，可能除了不同的端、不同语言、不同端交互的特征之外，基本知识结构上不会相差太多，甚至有差异性的地方还有很好的互相借鉴意义。

在企业中，前端的合作伙伴有非常多。理论上，作为前端个体而言，转型成任何一种角色都挺正常。但对于一个群体而言，我认为在大纬度上会有两种

方向——“云”和“端”。

“端”容易理解，在各个端，利用各种技术，完成业务产品中的数据消费，信息架构，人机交互的工作（PC、无线、VR、AR等）。

“云”不是通常意义的云计算，而是借云计算和云开发者之间的关系类比一下，这里指“端”开发者在完成业务产品时所需要的基础研发能力以及产品的能力的提供者。

前端工程师发展建议：思辨、容纳和好奇心

我们的前端生态很活跃，这让我的心理挺矛盾的。我一方面高兴，是因为活跃的社区才能充满创造力，而前端又极端的需要创造力。另一方面我又担心，是因为在活跃的社区中，明天就有可能天翻地覆，谁都不知道我今天选择的是不是代表未来，有那么点赌博的味道；而且频繁的调整业务实现方式，困扰的不只是前端自己。

上文曾提到，我们通过分层架构，尝试从技术上解决前端技术体系的频繁变动而导致临近技术体系需要被动调整的问题。除了技术上的应对，前端人或者说技术人想在这样的环境中成长，应以什么样的心态来应对呢？我有三个建议：思辨、容纳和好奇心。

思辨

社区的活跃中，有语言的进步，有工具的进步。语言是你决定成为前端之

后的首先需要牢牢关注的基石，如果有精力，应该关注语言的进步背后的推动力，这是能够让你一定意义上拥有以不变应万变的能力。工具是你在目前的社区上，在工作中最常谈论最常用上，它们具备一定的通用性，看上去能解决所有人的某些问题，但也更容易被革新。这部分需要关注，因为这是你能更好解决问题的方法，但不要过于沉迷于工具，随着技术的发展，面对问题域的进一步复杂化，甚至其他工具的发展，新的工具会源源不断的被发明，旧的工具一样会周而复始的被抛弃，这个生命周期是一定存在的。

容纳

所有在社区中出现的东西，都是为了解决它所在的场景中的问题而诞生的，存在即合理在这个语境下挺合用的，先开放的接受之后，明白自己想要用这个东西做什么：学习用法？学习解决问题的思路？学习代码的组织方式？学习编程的技巧？明确这个问题后，会发现自己的目的性就明确多了，也会发现学习的脉络就会慢慢浮现上来。

好奇心

做前端需要有充足的好奇心，这个好奇心是指：对所有不熟悉的东西都有兴趣去了解一下；了解之后有兴趣上手实践一下；实践之后有兴趣思考一下；在合适的场景合适的时机应用一下，或者优化一下。

综合在一起，可以这样来描述。

- 对新技术，新交互形式等新鲜玩意充满好奇和探索的欲望。
- 从对用户的认知，对业务的认识，对产品的理解，找到最合适的方式解决问题，比一直追在技术浪潮尖端学习来的帮助更大。
- 不要过于满足于任何时刻的成果，学会时刻的自省，以及有克制的精益求精。

很庆幸，在我踏入职场不久就让我接触到了前端这个角色，让我有机会把对美好设计感的追求和技术的力量叠在一起，在B类业务经历的这段时间，让我有更多的机会和责任去思考成为前端的追求，总算略有些成果，和大家分享。

作者介绍

赵振宇，2004年毕业于上海海事大学，2008年加入阿里巴巴国际事业部，从一线工程师开始阿里的前端之旅。先后负责阿里巴巴国际事业部前端团队，阿里巴巴企业事业群前端团队。目前担任阿里巴巴企业事业群（BBC）体验技术中台负责人。10多年大型互联网企业及自己创业的经验。在阿里期间，主要是从事国际化站点的前端业务解决方案和应用架构的技术架构和技术管理工作。主要专长和兴趣包括：UI领域标准化建设、互联网电商平台与应用架构演进、大型企业级架构中前端的分层治理、前端人员的发展与培养。

从普通程序员到三百人团队CTO，技术人成长的易与不易



作者 崔玉松

成功学者们总是在灌着各种鸡汤，好像按他们说的做就能走上人生巅峰。普通程序员们也经常幻想着成长可以一蹴而就，三五年做到架构师、CTO 好像触手可及。平凡的人总是相似，不凡的人各有各的不凡。今天不灌鸡汤，只有最朴实的文字与回答。本文不一定能马上帮到你，但起码可以带给你一些思考与方向。

写在前面

起初，他也曾是一名普通小程序员，循规蹈矩地上班下班；后来，他突然发觉这并不是自己想要的生活，于是从阿里辞职，跳槽有赞；现在，他是带领三百人技术团队的CTO。

和许多程序员一样，有赞CTO崔玉松自爆也曾不善言语、不喜与人沟通。曾经的梦想是做一名优秀的架构师，带

领一个不超过10个人的精英架构师团队。如今，他侃侃而谈，做事沉稳干练，并管理有超过300人的研发团队。InfoQ为此专访了崔玉松，希望他能够为大家分享一下，在不进则退的技术人生里，如何仰望星空、脚踏实地。

左手阿里，右手初创，选择？

2013年，他放弃阿里投向有赞，

一边是成熟稳定的互联网大公司，一边是充满不确定性的初创小公司，是什么促使他做出了这种决定？有赞哪些地方吸引了他？

标准的回答应该是为了理想，实际上我也不知道算不算。2012年底和团队一起基本上做完了淘宝搜索业务端的重构，我在思考下一步做什么。当时得到的信息很碎片，大家都不是很清晰明年搜索业务做点什么，只是说垂直化，个性化。我自己思考了一段时间，搜索业务从诞生以来到当时其实没什么变化。

总的来讲就是一个搜索框，输入一个词看到结果，搜索结果呈现之前有很多很复杂的逻辑和算法。但是这个业务已经到了除非发生革命性变化，否则用户不会有太多感知的地步。我觉得自己的投入和用户感受到的价值不成正比，于是我开始关注外界的机会。

加入有赞是个偶然，2012年年底我看白鸦的微博在上招人，到2013年初还在招人，我就在微博上联系了一下他，最后大家约个时间见面，聊了两三个小时，感觉挺靠谱的，其实当时已经决定加入了。保守起见，我回去想了几天，就主动提出来要加入。当时只是觉得团队挺靠谱，事情也值得做，然后主动给白鸦打了一个电话，告诉他我要加入，电话完就写了辞职信。

我记得当时他还没准备好给我的薪资，说商量好告诉我。晚上的时候，他告诉了给我的薪资数值，实际上这个薪资是我当时在阿里的一半都不到。只

是我在联系他之前已经考虑到了这个问题，所以就算没有工资我也想去试一下，对于我而言做点自己认可的事情比任何其他额外因素都重要。

程序员到CTO，如何快速成长？

从一个普通程序员，到team leader，再到管理300多人的CTO，管理上是如何做到快速提升与成长的？有无特别的经验可供分享？

有赞现在有超过300个工程师的研发团队，从一个工程师到一个leader的阶段是非常简单的，很多人都可以很快做到。leader就是很多匹拉车的马中，走在最前面，带头做榜样和控制方向的那匹马。通常来讲，团队的成员不超过20个人时，整个管理成本很低，基本上不会有太多的管理动作。只要细心一些，人员招募到位，做事情的时候身先士卒，日子基本上都会很好过。

但是，当研发团队规模达到100多个人的时候，leader所起的作用就比较弱了，之前作为leader的各种本能的管理手段都失效了。当团队大到无法感知你的能量，这个时候就要考虑更多不一样的策略了，比如把团队分层。当然这个也取决于在团队治理的时候选择什么样的策略。

有赞选择了高度扁平化的策略。所以在团队规模达到100人的时候我们都没有正式的manager角色，我们只是把团队按照角色和职责做一些简单的划

分，确认核心人员能够在各个战线上充当 leader，能够身先士卒，勇于担当，自我激励往前走。如果你的团队有 20 个这样的人，基本上 100 人的团队还是不用太多的管理手段介入。

当团队规模超过 200 人，这个时候就真的需要真正的 manager 了，因为这个时候会遇到很多噪音，需要大量的时间纠偏。团队可能有一半人甚至更多对最高技术管理者根本不了解，甚至没有那么强的信任感，因为大家每天接触的时间实在太有限。大部分人可能一年说不上几句话，让人家信任你是不太可能的，所以我们在这个节点上正式组建了 team leader 的团队。我只负责 team leader 团队和少数需要攻坚的直属团队。

新晋的 leader 至少你是充分信任的。虽然他们肯定也会犯很多的错，而且绝大部分还是我之前曾经犯过的错误。但是人就是这样，你和他讲的时候他不以为然，非要自己犯一次才会觉得你说的是对的。不过，在我看来这就是一种成本，从公司层面看是一种必须付出的成本。即使你招的是一个成熟的管理者，也要适应你的公司独特的需要，他也是要做很多调频和“犯错”。

组建管理团队我觉得重要的地方是在于信任和放权，道理大家都懂，但绝大多数从底层成长起来的技术负责人都做不到。我刚开始也不适应，因为事情不是你亲自分配下去的，执行的人是谁也不清楚，每件事情都想要问一下结果，

总会担心事情发展是不是会走偏，是不是有隐患，等等。这确实需要一段时间的适应和练习。

核心团队的组建与管理

核心团队是如何组建的？日常如何进行团队管理？

我们经过三年的时间才逐步沉淀了一些有做 team leader 潜力的人，定向挖过来的也有，非常少。

日常我过问的比较少，一般只参与少数重点项目，少数重点的人转正或者谈话我一般会参加。对于团队，我还是把很多时间放在找人和招人上，特别是重要岗位的人。2016 年上半年，我 70% 以上时间都是在招聘，下半年各个 leader 逐渐成熟了，大家招人的理念和评判的标准逐渐一致之后，日常招聘的工作也都放出去了。

我只负责一些重要岗位重要人员的面试，这个时候我可以把更多的时间投入到每个团队我熟悉的人或者重点人员的沟通上，一有时间就聊几句。在沟通过程中，不但能发现很多存在的问题，同时也维持了和最前线同学的感情。

一个优秀的工程师团队非常重要，如何组建这样的团队，有赞是如何做到的？鼓励做什么，不鼓励做什么？

组建任何一个优秀团队都是不容易的，这是一个漫长的寂寞的过程。大部分公司在寂寞的过程里放松了对于候选人的要求，很多时候会把完成业务看的更重要。有赞在过去四年里坚持的了该

坚持的东西，甚至在早期使用了非常极端的手段。

比如，把试用期的薪资做的非常低，故意让周六上班。实际上，在转正的时候我们会加很高的比例，周六也大多以分享和团建活动为主，并非是真的在工。这样就能把一些对于短期利益非常非常看重的人过滤掉，因为一开始我们核心团队的共识就是我们要做一个事业，并非短期的生意。公司可以挂了，事情可以做不成，但绝不能违背内心追求。

不过，这个策略实际上也拒绝了一些真的不错的人，他们觉得自己的价值被低估了，不愿意加入团队。我们也怀疑过这个策略是不是对的，不过最终还是坚持了。任何规则都是有利有弊，我们需要的是更齐心的团队，所谓人心齐，泰山移。后面随着有赞业务和团队的规模化，步入到一个新阶段后，我们也及时调整了我们的策略。

有赞招人的三个最核心的特质：聪明、要性、皮实，缺一不可，有赞现在600人团队，技术超过300人，实际上全职的manager角色在公司没有超过10个人，我也算是其中的一个。高度扁平化的组织要求大家更多的是自我驱动，自我运转，对于信息的传递效率要求非常的高，对于噪音的忍耐限度很低。

我们非常鼓励大家相互补位，有问题及时寻找资源，及时获取有效的信息，鼓励大家面对面把事情说清楚，所以我们专门给所有人配备笔记本。我们杜绝一切办公室政治，杜绝人前一套人后一

套，也杜绝没事捕风捉影，造谣生事。

技术、管理、领导力的权衡

技术能力、管理能力、领导力哪个更重要？如何权衡？

很难讲哪个更重要，我也希望我是个完美的人，技术能力、管理能力、领导力都很强，可惜我做不到。我觉得正确的认识自己是很重要的技能，这是我经历了很多事情之后开始想明白的。如果像有赞这种成长方式，团队从开始几个人然后发展到上百人，我觉得前期技术能力是非常重要的。

因为早期业务发展非常迅速，根本没有时间做架构重构，对于技术管理者而言，做最有利的架构才能驱动公司的业务。毕竟一个团队可能除了技术管理者之外大家能力都很平均，如果连技术管理者也不懂或者能力很一般，那基本上业务会受损很大。评判这个的标准是CEO对于技术团队的关注次数乘以批评数量。

管理能力和领导力，我觉得领导力可能更重要，没有领导力基本上连核心管理层都建不起来或者管理不好。管理能力差可以通过核心管理层帮助你去做很多事情，甚至HRBP也可以帮助做很多事情。这个也和我的经历有关，我就是典型的管理能力不怎么样的人，全靠我们优秀的Team Leader团队。

进阶管理后，遇坑怎么填？

进入有赞后，管理或者技术上遇到

“ 创业对我来讲就是完成一件我想完成的事情，就是一个经历，无所谓好坏。 ”

过哪些特别的坑？最后是怎么解决的？

管理上没什么好说的，就是要正确认识自己的管理能力，扬长避短，摆正自己的角色和位置。不该说的要坚决忍住，该说的要仔细想想是不是应该 leader 去说，给他们解决问题和定位角色的机会。

技术上有赞走过的路和大多数从小到大的创业公司差不多，都是前期专注于解决业务问题，最后架构问题在某个时间点集中爆发，导致很多的不稳定。这一点不管是阿里也好，京东也罢，还是最近交流的一些其他公司，基本上都是一样的，只是大家问题的严重程度和解决问题的速度不一样。有赞的解决方法和大家也没什么太大区别，就是组建一流底层核心架构和核心运维团队，这

个团队必须得好，不然解决问题的速度非常非常慢，每天都可能宕机，会严重影响公司业务及效益。

公司做大，内心有无膨胀与彷徨？

作为公司的创始人及 CTO，当看到公司规模做来越大，发展越来越好，内心有没有膨胀过、彷徨过？内心想法是？

我是一个相对保守和冷静的人，以前看过很多关于死亡和哲学的书，所以从来没有膨胀过，反而会经常思考更多的危机和问题。不过，这个也有不好的地方，就是很少表扬团队，因为我看到的都是问题和危机。

彷徨倒是有过一段时间，可能差不

多有三四个月。我从未想过我有一天要带几百人的团队或者将来要带上千人甚至更多人的团队，从我在大学里决定开始玩编程的时候，我的理想就是做一个优秀的架构师，绝大部分问题能在我这里能得到解决。我希望我带领的是一个不超过 10 个人的非常精英的架构师团队。

我自认为自己不是一个很好的管理者，一个优秀的管理者要去触及人性的各个方面，而这个是我之前觉得我干不了的事情。我退缩过，但是最后还是得站出来，职责所在，责任所在，我的角色必须去承担这样的事情，遇到的困难和痛苦都必须我自己解决和承受。想明白这些事情之后就突然觉得释放了，那就重新给自己一个新的定位和目标，去打造中国最好的技术团队。也许没有几个人像我这样幸运，有这样的机会，在创业中杀出一条路。所以，我很知足。

创业维艰，最大收获是什么？

好像很难讲具体的收获是什么，将心比心吧。看很多事情的时候变得更多面了。现在看以前的自己，就是个愤青，不过现在也会喷。比如前几天的大雾霾，但是喷之前大多先做很多的细致研究，从数据上，各种观点上，还有从很多其他方面查漏补缺，最终再形成一个结论。

除此之外，可能最大的收获就是见识了很多没有见识过的东西，认识了很多跟自己趣味相投的人，感觉自己的人生变得更加丰满更加真实了。还有，以

前我是一个不怎么说话的人，现在能够和三五个人坐在酒吧里就一个话题聊十几个小时。

还有很多很多的变化，我也不知道哪个是最大的收获，创业对我来讲就是完成一件我想完成的事情，就是一个经历，无所谓好坏。我只是选择了这样的生活方式，和很多人选择了朝九晚五并没有什么大的区别。不同的人有不同的生活方式，但每种方式遗憾和快乐都会并存，想清楚什么是自己想要的就好。

作者介绍

崔玉松，有赞联合创始人、CTO。前阿里巴巴技术专家，资深码农。崔玉松也将在 2017 年 4 月 QCon 大会 · 北京站上的“工程团队建设”专题中做分享！想要听取更多经验的小伙伴不可错过。

腾讯HTTPS性能优化实践



作者 罗成

本文根据罗成在 2016ArchSummit 全球架构师（北京）峰会上的演讲整理而成。
主要内容分以下三部分：计算性能的分析和优化；无密钥加载；证书优化。

为什么66%的网站不支持 HTTPS？

谈优化之前我们先看背景和趋势，大家也很清楚 HTTPS 是大势所趋，Google、Facebook 和国内诸多大型互联网公司也已经支持 HTTPS，然而这里有两点大家需要注意：

- iOS10的ATS政策（App Transport

Security）要求2017年1月1日后所有在iOS App Store上架的App都需要支持HTTPS，否则无法上架；

- Google的Chrome浏览器54版本已经将HTTP的域名输入框前增加“！”的提示，如下图，所有的HTTP站点都会有这个标识。同样在2017年1月1日后开始，Chrome浏览器会在用户点击“！”的提示符后将该网站不安

- 慢
 - 移动端慢500ms以上

- 贵
 - 增加服务器成本
 - ◆ HTTPS性能不到HTTP 1/10
 - 证书成本
 - ◆ 申请繁琐
 - ◆ 价格不一
 - ◆ 容易过期、失效

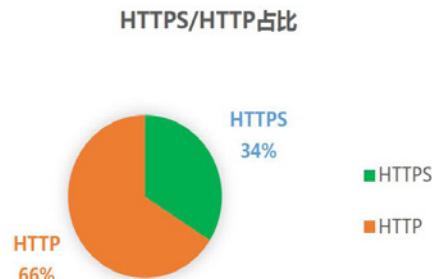


图 1

全的信息显示出来，只要涉及到登录和搜集用户数据的页面，只要是 HTTP 的都会标注不安全，相信这也会加速 HTTPS 的推进。

HTTPS 很安全，很古老也很成熟，为什么一直到今天我们还有 66% 的网站不支持 HTTPS 呢？原因有两点（见图 1）：

1. **慢**，HTTPS 未经任何优化的情况下要比 HTTP 慢几百毫秒以上，特别在移动端可能要慢 500 毫秒以上，关于 HTTPS 慢和如何优化已经是一个非常系统和复杂的话题，由于时间的关系，本次分享就不做介绍了。但有一点可以肯定的是，HTTPS 的访问速度在经过优化之后是不会比 HTTP 慢；

2. **贵**，特别在计算性能和服务器成本方面。HTTPS 为什么会增加服务器的成本？相信大家也都清楚 HTTPS 要额外计算，要频繁地做加密和解密操作，几乎每一个字节都需要做加解密，这就产生了服务器成本，但也有两点大家可能并不清楚：

- HTTPS 有哪些主要的计算环节，是不

是每个计算环节计算量都一样？

- 知道这些计算环节对 CPU 的影响，我们如何优化这些计算环节？
- 接下来我将介绍我们在这两个问题上的探讨。

HTTPS 主要的计算环节

首先看 HTTPS 主要的计算环节，下图是一个协议交互的简要介绍图，它的四种颜色分别代表 4 种不同的主要计算环节：

红色环节是非对称密钥交换，通过客户端和服务端不一致的信息协商出对称的密钥；

蓝色环节是证书校验，对证书的签名进行校验，确认网站的身份；

深绿色环节是对称加解密，通过非对称密钥交换协商出对称密钥来进行加解密；

浅绿色环节是完整性校验，不仅要加密还要防止内容被篡改，所以要进行自身的完整性校验（见图 2）。

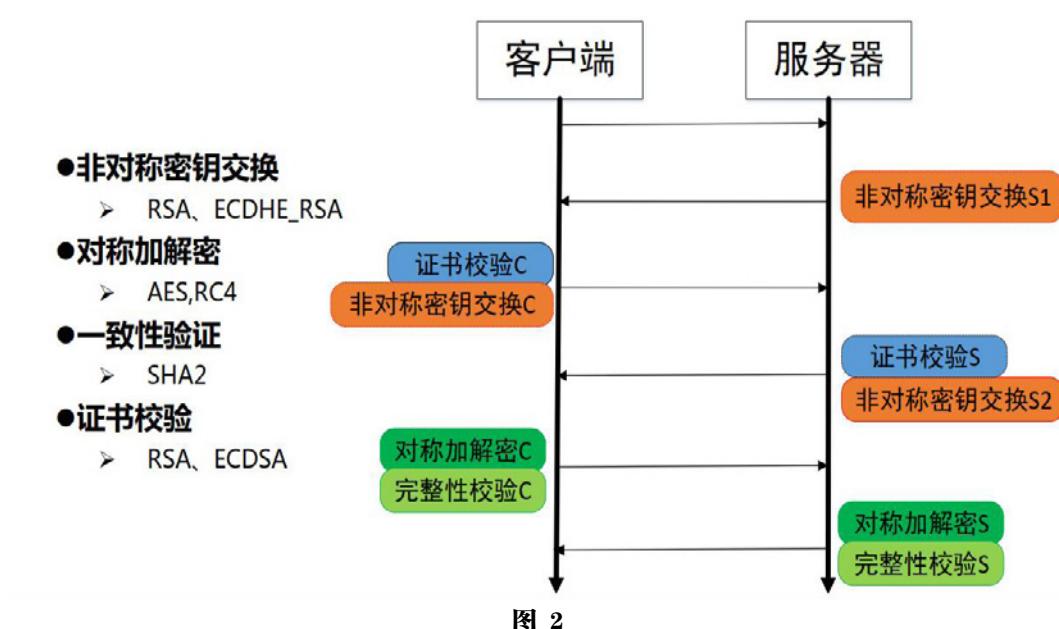


图 2

知道这些主要的计算环节之后，每一个计算环节对计算性能的影响分别是多少以及如何分析？这里和大家分享我们计算性能的分析维度，主要分为三部分：算法、协议和系统。

1. 算法，所谓的算法其实是HTTPS所用到密码学里最基本的算法，包括对称加密、非对称密钥交换、签名算法、一致性校验算法等，对应的分析手段也很简单：openssl speed；
2. 协议，因为不同的协议版本和消息所对应使用的算法是不一样的，虽然算法的性能很确定，但是和协议关联起来它就不确定了。由于性能和协议相关，我们重点分析的是协议里完全握手的阶段，我们会对完全握手的每个消息和每个函数进行时间的分析；
3. 系统，比如我们使用Nginx和

OpenSSL，我们会对它进行压力测试，然后在高并发压力环境下对热点事件进行分析和优化。

接下来详细介绍以上的分析维度，首先是对称加密和一致性校验算法的测试分析，这个手段和工具（openssl speed）很简单，我就不多介绍了。这里总结一下：下图中柱状图越高表示性能越好，可以看出性能最好的是AES-128-GCM，性能最差的是AES-256-CBC，但即使它性能最差，它也需要47微秒就能处理4000个字节，性能相比来说也还能接受（见图3）。

接下来我们看密钥交换和签名算法的测试（见图4），下表中Sign代表服务端进行的签名，Verify指的是客户端对签名进行的校验。我们关注一下红色数字809，这代表使用RSA-2048位时，我们服务端1秒钟只能处理809次，这

●算法

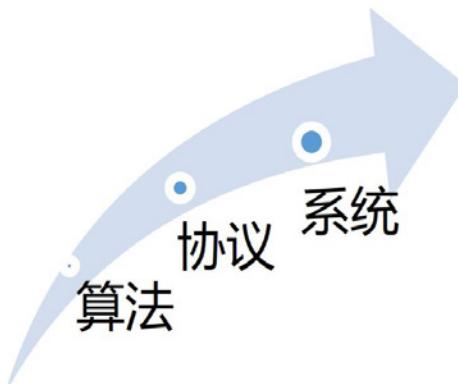
- openssl speed
- 对称加密，非对称密钥交换，签名算法，一致性校验算法

●协议

- 完全握手
- 函数级耗时

●系统

- 热点事件
- 工程实现



●openssl speed -elapsed evp

算法名	每秒处理的字节数(块大小为1K)	处理4K字节需要的时间
AES-128 CBC	117499.22k	0.00003s
AES-192 CBC	97594.71k	0.00004s
AES-256 CBC	83456.68k	0.000047s
SHA1	488445.95k	0.000008s
SHA256	193084.22k	0.00002s
RC4	623545.69k	0.0000064s
AES-128 GCM	1120621.23k	0.0000035s
AES-256 GCM	981585.24k	0.000004s
CHACHA20 POLY1305	205781.33k	0.000019s



图 3-图 4

已经是我们使用线上非常好的一款 CPU 进行测试的结果，事实上大部分机器 1 秒钟只能处理三四百次，可以说性能非常差。

接下来是 Verify 校验，能看出来我们使用 Ecdsa(nistp256) 时一秒钟能处理 7000 多次，同样这也是我们使用线上比较好的服务器所测试的结果，由于 Verify 发生在客户端，考虑到移动端手机的 CPU 是非常弱的，因此这里一秒钟可能只能处理几百次（见图 5）。

接下来看协议耗时的分析，这里用

ECDHE_RSA 非对称密钥交换握手进行举例，大家注意红色 ServerKeyExchange 部分，它用了 2400 微秒（2.4 毫秒），这是一个非常恐怖的概念，如果我们 HTTPS 请求每一次都需要进行完全握手处理，这意味着我们 CPU 一个核每秒最多只能处理 400 次多一点（见图 6）。

最后我们看热点事件的分析，它也比较简单，我们对系统进行压力测试，用 perf record 对事件进行记录，然后使用 flame graph 将它们可视化出来，最后看到一些相关数据和结果。

●openssl speed RSA

➤ RSA签名计算一秒钟最多809次

算法名	Sign	Verify	Sign/s	Verify/s
RSA 2048	0.001235s	0.000037s	809.4	27339.7
DSA 2048	0.000435s	0.000463s	2297.0	2161.6
Ecdsa(nistp256)	0.0001s	0.0001s	16576.9	7012.4

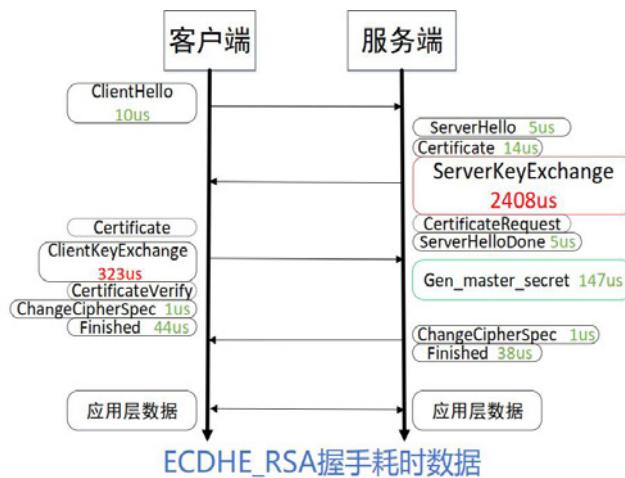


图 5-图 6

总结以上计算性能分析：

- 完全握手的性能不到普通HTTP性能的10%，如果说HTTP的性能是QPS 1万，HTTPS可能只有几百；
- 为什么会这么低呢？主要是RSA算法，它对性能的影响占了75%左右；
- ECC椭圆曲线如果使用最常用的ECDHE算法，这部分约占整体计算量的7%；
- 对称加解密和MAC计算，它们对性能影响比较小，是微秒级别的。

有了这些分析结论，如何优化呢？我们总结了三个步骤：

首先第一步也是最简单的一个优化策略，就是减少完全握手的发生，因为完全握手它非常消耗时间；

对于不能减少的完全握手，对于必须要发生的完全握手，对于需要直接消耗CPU进行的握手，我们使用代理计算；

对称加密的优化评论。

简化握手的原理以及实现

我们首先来看完全握手和简化握手，这是TLS层的概念，我简单说下简化握手的两个好处：

1. 首先简化握手相比完全握手要少一个RTT（网络交互），从完全握手大家可以看出，它需要两个握手交互才能进行第三步应用层的传输，而简化握手只需要一个RTT就能进行应用层的数据传输；

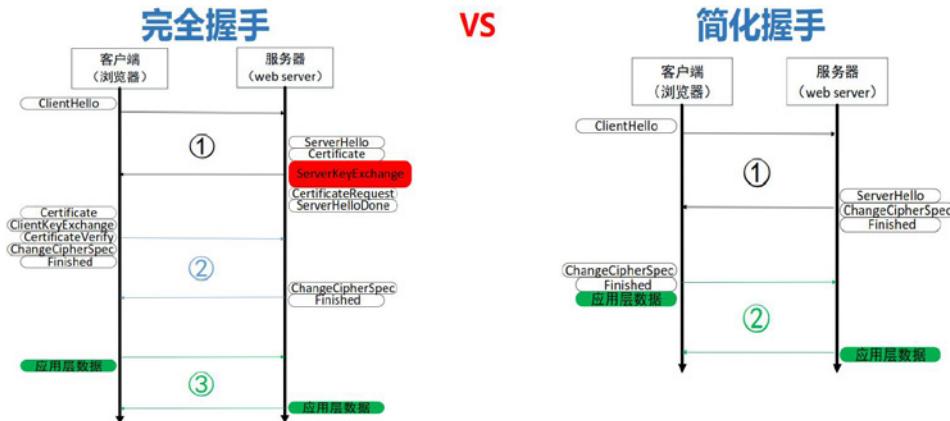


图 7

2. 完全握手有ServerKeyExchange的消息（红色框部分），这个消息之前提过需要2.4毫秒，另外完全握手有Certificate证书的消息，而简化握手并不需要。这也就是简化握手第二个好处，它减少了计算量，它不需要CPU消耗太多时间（见图7）。

既然简化握手这么好，我们如何实现？首先看协议层如何支持。TLS 协议层有两个策略可以实现，第一个是Session ID，Session ID由服务器生成并返回给客户端，客户端再次发起SSL握手时会携带上Session ID，服务端拿到后会从自己的内存查找，如果找到便意味着客户端之前已经发生过完全握手，是可以信任的，然后可以直接进行简化握手。

第二个策略是Session Ticket，同样它也是客户端发起握手时会携带上的扩展，服务器拿到Session Ticket后会对它进行解密，如果解密成功了就意味着它是值得信任的，从而可以进行简

化握手，直接传输应用层数据。

工程实现上会有什么问题呢？现在最常用的 Nginx+OpenSSL 有两个局限：

Nginx 只支持单机多进程间共享的 Session Cache，假如我们所有接入用的是一台服务器、一台 Nginx 的话，那 ID 生成和查找都在一起，肯定是可以命中的，但是我们大部分特别是流量比较大的接入环境都是多台机器接入。比如我们同一个 TGW 或者说 LVS 下面有多台 Nginx，那么第一台 Nginx 产生的 ID 返回给用户，用户可能隔了一个小时之后再发起 SSL 握手，它携带上的 Session ID 肯定会随机地落到某一台 Nginx 上（比如落在第三台 Nginx 上），这样肯定无法查找到之前的 Session ID，无法进行简化握手，这是第一个局限，即命中率会比较低；

OpenSSL 提供了一个 Session Cache 的 callback 可以回调，但是这个回调函数是同步的，而 Nginx 是完全异步事件驱动的框架，如果 Nginx 调用这个

callback 进行网络查找，假如这个网络查找需要 1 毫秒，这意味着整体性能不会超过一千次。

我们如何进行改进？我们看第一个问题(Session Cache)的两个改进方案：

1. IP Hash，这是最简单的根据 IP 做 Hash 的负载均衡策略，相信大家对此都很清楚，这方案的好处是可以保证相同的 IP 用户永远都在同一台 Nginx 上面，Session Cache 的命中率会提升，但是它有两个缺点：

- 它容易导致热点，我们有很多 Net 网关出口 IP 用户的访问量非常大，也就是说有一些 IP 请求非常大导致某一台机器负载不均衡；
- 用户 IP 可能会经常变化，特别在移动端上，在 Wi-Fi 和 4G 环境下切换导致的 IP 变化同样会使 Session Cache 的命中率降低。

2. 分布式缓存，这是更优的方案，假如用户开始发起握手，我们第一台 Nginx 生成 ID 会写入到一个全局的比如

redis 缓存里，然后返回给用户。用户下一次发起握手的时候，假如他落到第三台 Nginx 上面，由于我们都是全局的 Session Cache 查找，这命中率一下就提升上来了。我们实现了这个方案，但暂时还没有开源，在这里可以给大家推荐两个开源方案，大家有兴趣可以了解一下。

- OpenResty，它提供了 SSL Cache 全局查找的指令；
- BoringSSL，这是 Google fork OpenSSL 的版本，它也在 SSL 层面上实现了异步的 Session Cache 查找。

接下来我们看 Session Ticket（见图 8），由于 Session Cache 有个缺点是必须在服务端做缓存，会浪费很大内存，而 Session Ticket 有个好处是它不需要服务端做缓存，但同样它也有个缺点：默认情况下比如三台 Nginx 各自的 Session Ticket 加解密密钥是不同（这里的密钥是指 Session Ticket 的对称加解密的密钥而不是指证书对应的

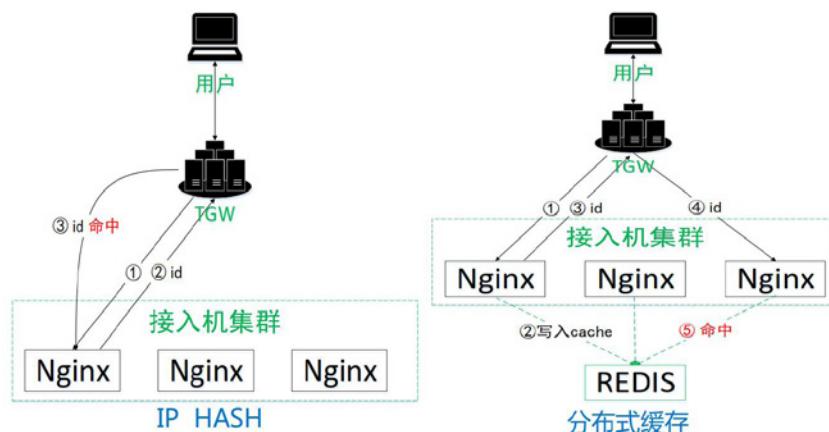


图 8

私钥）。

举个例子，比如第一台 Nginx 的 Session Ticket 用密钥加密返回给用户，用户下一次再访问落到第三台机器，你用第一台机器加密产生的密钥用第三台的密钥去解密肯定会失败。这个问题很好解决，我们将所有的 Nginx 机器配置成同一个加解密的密钥就可以了，这样也能实现简化握手（见图 9）。

我们看一下第三个方案：Self Session Ticket。Session ID 和 Session Cache 都有一个共同点：Session 基于内存，如果我们的 App、浏览器或操作系统如果第一次启动或重启（或者浏览器的 Tab 关闭后又打开）都有可能导致 Session ID 和 Session Ticket 丢失，出于安全角度考虑，这种情况下就必须要发起完全握手，怎么解决呢？

如果这个 App 是我们完全自主、独立自主开发，我们可以实现 Self Session Ticket，我们将 Ticket 存储

在硬盘里面，而不是在内存里。这显然会带来一定的安全风险，但是我们会做一些限制：

- Ticket 存储在 App 的私有路径里，对非 Root 的手机是很难读取到私有路径的数据；
- 我们可以选择对一些安全系数要求不是很高的业务开启这个功能；
- 这个密钥开关我们做到可以随时控制。

综上，即使这方案出现最危险的情况，其实是和 HTTP 方案是一样的，它不会比 HTTP 方案安全性要差（见图 10）。

异步代理计算的原理和实现

刚才提到的都是关于如何减少完全握手，提升简化握手，但对很多的请求，比如说浏览器第一次启动必须要经过完全握手，而且不是我们能够自主控制的。对于这部分的内容，完全握手比例占了至少 30% 以上，也就是说我们还有 30%

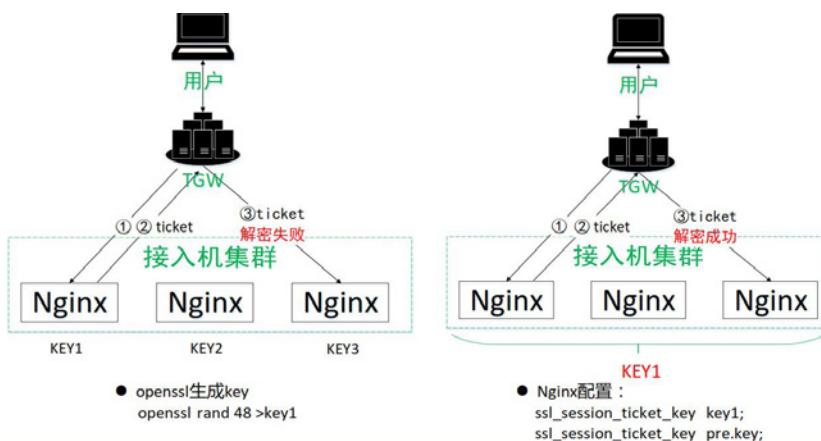


图 9

●完全握手的场景

- App, 浏览器, OS重启
- 基于内存



●安全性分析

- 私有路径

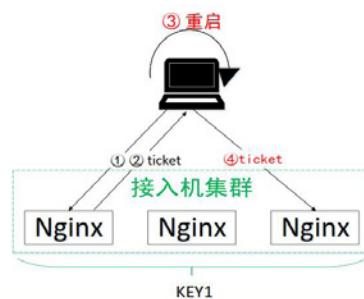


图 10

以上的请求必须要触发 CPU 进行大量计算, 对这部分怎么解决呢?

我们的方案是异步代理计算, 主要是分三个步骤:

- 1. 算法分离**, 把最消耗CPU资源的算法剥离出来, 不让它消耗本机的CPU资源;
- 2. 代理计算**, 既然不消耗本机的CPU资源, 我们可以使用硬件加速卡或者空闲的CPU资源来完成计算;
- 3. 异步执行**, 我把算法分离出来交给计算集群去计算的时候, 这个过程是异步的, 我不需要同步等待计算结果返回, 这样对我们性能提升也是非常有帮助的。

算法分离

要分离哪些算法? 最主要是密钥交换算法(非对称密钥交换算法), 密钥交换算法最常用的是三类:

- RSA
- ECDHE_RSA
- DHE_RSA

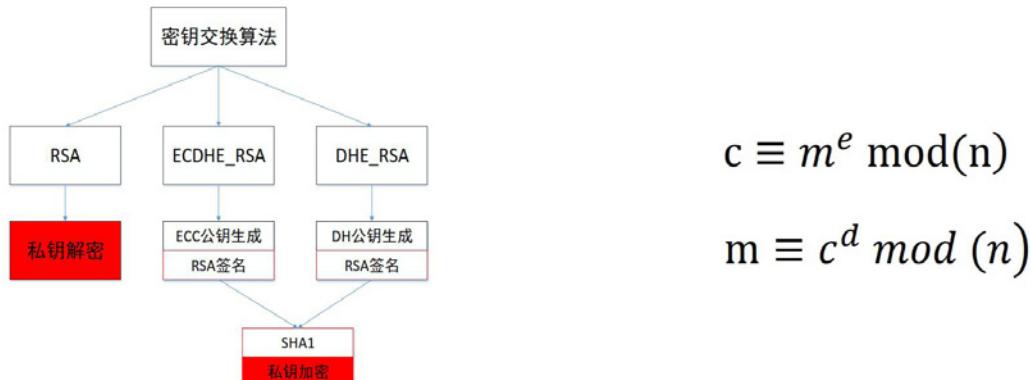
由于 DHE_RSA 是非常消耗性能的, 这个方法也不安全, 所以我们线上并没有采用。目前使用最多的是 ECDHE_RSA, 考虑到兼容性的问题时使用了 RSA。

RSA 和 ECDHE_RSA 为什么会消耗 CPU 资源? RSA 主要是对客户端发回来的 pre_master_secret 进行解密, 它消耗 CPU 资源的过程是私钥解密的计算; 而 ECDHE_RSA 则有两个步骤(见图 11):

1. 生成ECC椭圆曲线的公钥和几个重要的参数;
2. 对这几个参数进行签名, 客户端要确保参数是我服务端发过来的, 就是通过RSA的签名来保证。

RSA 签名为什么消耗 CPU 呢? RSA 签名同样有两个步骤:

1. 首先它通过SHA1进行Hash计算;
2. 对Hash结果进行私钥加密, 也就是最终消耗CPU的过程是私钥解密和私钥加密的计算。这两个计算为什么消耗CPU? 看下图公式, 如果e或者d



$$c \equiv m^e \pmod{n}$$

$$m \equiv c^d \pmod{n}$$

图 11

这个指数是一个接近2的2048次方的天文数字，那就非常消耗CPU。这就是RSA算法为什么消耗CPU的最直接的数学解释。

我们再看一下协议层面我们该如何实现分离。同样以ECDHE_RSA为例，由于使用了ECC参数和RSA签名，之前提到的ServerKeyExchange这个消息用了2.4毫秒，我们需要对这个消息进行分离，将一步操作拆成多个步骤。

我们再看一下RSA密钥交换的分离，RSA非对称密钥交换算法不需要ServerKeyExchange的消息，但需要对Client发过来的ClientKeyExchange进行解密的操作，也就是最消耗算法的过程是RSA解密的地方，我们需要对这个步骤进行分离。

异步代理计算——架构

以上是算法层面包括协议层面进行的分离。接下来解释工程实现上的架构，下图左部分是最常用最简单的配置，比

如我们配置好Nginx+OpenSSL，然后把证书和私钥放上去，把443端口打开，就能用上HTTPS了，但这里消耗的都是本机的CPU资源，这里面就会性能很差。

我们看异步硬件加速卡代理计算的模型，用户发起HTTPS握手，涉及到私钥计算（比如ServerKeyExchange和对ClientKeyExchange解密）的时候，我们会把ECC的参数剥离出来发送给我们的计算集群，发送出去同时立马异步返回，又可以接受其他用户的请求，不需要等待（见图12）。

计算集群计算完了以后，我们会把计算结果返回给比如Nginx，Nginx又触发最初的用户场景，从而完成HTTPS的握手，这是一个大概的交互流程。

这里面需要注意的是，我们不仅可以使用SSL硬件加速卡，还可以使用线上的空闲CPU资源，比如CPU比较空闲的存储集群，如果硬件加速卡出现故障，我们可以直接把卡拔掉，可以用硬件加速卡集群上的CPU资源来做计算。

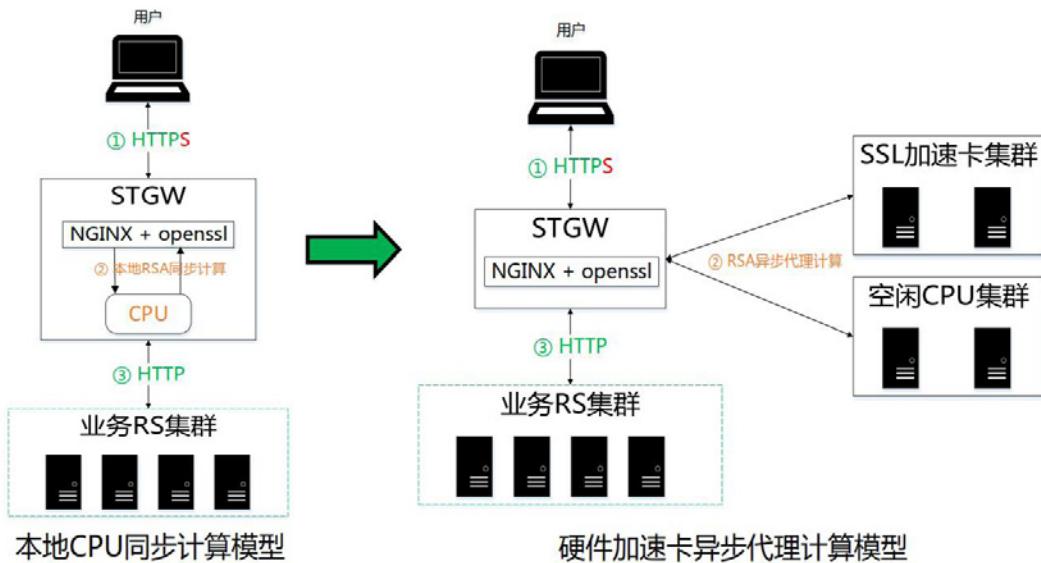


图 12

我们对算法的解耦是非常纯粹的，和协议、证书没有任何关系，我们只做 RSA 的计算，不管哪一边进行升级，只要 RSA 算法还是安全的，我们的协议就十分稳定，我们的维护成本也非常低。

异步代理计算——工程实现

我们看一下工程实现，像 Nginx 需要事件框架进行修改来实现 OpenSSL。Nginx 计算集群交互，通过模块是无法实现的，尽管 Nginx 模块机制非常强大丰富，但是它在 HTTP 头部数据解析完成之后才能介入处理，但 SSL 握手只有进行 SSL 握手完之后才能进行应用层的数据传输，也就是说在进行 SSL 握手的时候，它没有任何 HTTP 的数据，这时候模块是没办法介入处理的，必须对事件框架进行修改。

关于 OpenSSL，需要对 OpenSSL 的协议栈进行修改，主要涉及到 s3_

srvr.c 这个文件，这个文件主要是实现 OpenSSL 握手的协议栈，这里介绍一个开源方案：OpenSSL1.1.0，它已经支持了异步事件，同样还是需要修改 Nginx 才能实现异步。性能也很强大，纯 ECDHE_RSA 性能相比本机能够提升 3.5 倍，而且解耦做得非常好。

ECC椭圆曲线的优化

刚才提到的一系列都是针对完全握手、简化握手的介绍。接下来看对 ECC 椭圆曲线的优化，这里说的优化不是针对算法本身进行的优化，而是使用 OpenSSL 的官方版本过程中需要注意的地方。

尽量使用 NIST P-256 这条曲线，这条曲线是 Intel 几个工程师在 2013 年进行的优化，性能提升了 4 倍。通常来讲密钥越大，性能肯定越差，比如 RSA-2048 肯定比 1024 要慢 4 倍左右，

P-256 曲线看上去应该比 P-224 要慢，但是实际上 P-256 曲线比 P-224 性能高了 4 倍，正是因为经过了一系列的优化。

另外需要注意的是 OpenSSL 的版本，这个优化特性只是在 1.0.1L 之后才加入的，下图表格中 OpenSSL 的 1.0.1e 版本 ecdh(nistp256) 的性能只有 2548，而 OpenSSL 的 1.1.0b 版本 ecdh(nistp256) 性能能达到 10271，提高了 4 倍（见图 13）。

对称加密算法的优化

我们再看一下对称加密算法的优化，对称加密主要分两块：

1. 块式对称加密算法，根据刚才 OpenSpeed 跑的结果也能发现 AES-GCM 性能最高，建议大家使用。AES-NI 同样是 Intel CPU 提供的硬件加速指令，

这个指令相比不开启的 CPU 性能要提升 5 倍左右，现在市面上 2010 年之后的 Intel CPU 都是支持的，但需要注意如果要直接使用 AES 对称加解密，一定要使用 OpenSSL 封装的 EVP_EncryptInit_ex 函数，而不是用最底层的 AES_encrypt（尽管它很好记也很好用），默认的 AES_encrypt 函数是不会用硬件加速指令的，因此性能会很差；

2. 流式对称加密算法。Chacha20-Poly1305 是由 Google 专门针对移动端 CPU 优化的流式对称加密算法，它的性能相比普通算法要提高 3 倍。但 Chacha20 算法只适用于稍微低端、且不支持 AES-NI 指令的手机才能提高 3 倍，举个例子，例如 iPhone 支持 AES-NI，但它的性能还是会比 AES-GCM 要差的（见图 14）。

- 优先使用 NIST p256
 - P224 以上安全

- OpenSSL 版本
 - 1.0.1l

● OpenSSL 1.1.0b

算法名	OP/s	OP
ecdh (nistp192)	0.0003s	3805.3
ecdh (nistp224)	0.0004s	2808.8
ecdh (nistp256)	0.0001s	10271.9
ecdh (nistp384)	0.0009s	1176.0

● OpenSSL 1.0.1e

算法名	OP/s	OP
ecdh (nistp256)	0.0004s	2548.8
ecdh (nistp384)	0.0008s	1192.8

图 13

●AES-GCM

- 性能最高

●AES-NI

- 性能提升5倍左右
- EVP_EncryptInit_ex vs AES_encrypt
- OPENSSL_ia32cap="~0x200000200000000"
- openssl speed -elapsed -evp aes-128-gcm

●高性能CPU

- TCO

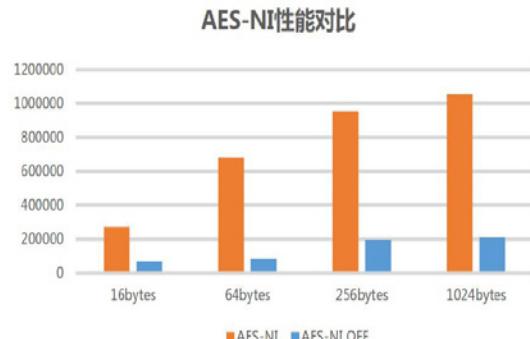


图 14

RC4、SSL3.0 已经不安全了，也彻底退出了历史舞台，但我们还有很多客户端只支持 SSL3.0，比如 IE6，比如 WindowsXP 的一些客户端和浏览器，我们的业务方仍需要支持这些业务，因此我们就必须支持 SSL3.0，那如何支持呢？

首先 SSL3.0 为什么不安全？主要是两点：

- SSL3.0 存在 AES-CBC 的缓存 Poodle 漏洞；
- RC4 存在 Bios 偏移的漏洞。

这两个漏洞哪个更严重？Poodle 漏洞会更严重一些，于是开启 SSL3.0 的话我们只支持 RC4，这是第一点安全性的考虑，其次 RC4 的算法性能要比 AES-CBC 要高，比如业务方一定要支持 WindowsXP 和 IE6 的话，优先使用 RC4。

无密钥加载

我们再看无密钥加载，无密钥加载的背景是这样的，HTTPS 证书和私钥，

私钥是 HTTPS 安全的根本，如果私钥被泄露了，那么意味着你的 HTTPS 是没有安全性可言，如果泄露了，只能撤销证书，重新生成私钥。

而且我们大部分使用 HTTPS 的场景都是将私钥和证书同机部署在比如 Nginx 上面，然后私钥保存在硬盘，这其实有安全风险的。比如说我们有很多兼容的大客户，他使用了证书和私钥，但是他的业务可能分散在几个云，或者几个 CDN 上，他就会担心私钥如果泄露了怎么办？

我们因此设计了一个无密钥加载的方案，我们先看普通的 HTTPS 流程，用户发起 HTTPS，到达腾讯云，到达 STGW，我们直接调用私钥完成计算，完成 HTTPS 的握手，然后卸载，然后将 HTTPS 传递给业务。这是最普通的流程，会有一定的安全风险（见图 15）。

再看一下无密钥加载流程，什么是无密钥呢？比如腾讯云，我们接收的服务器不需要部署私钥，我们私钥是完全放在客户的物理服务器上面，为了保证

- 私钥是安全的根本
- 同机部署
 - 接入服务器
- 泄露风险大
 - CDN
 - 金融客户

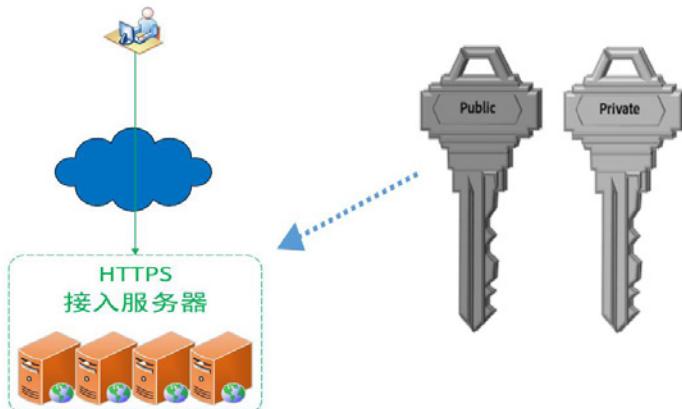


图 15

私钥的安全，客户甚至可以把这个物理服务器放在家里。

正常的接收流程是这样的，我们用用户到达 HTTPS 握手，到达 STGW，然后涉及到私钥计算的时候，我们会把请求以及和私钥相关的参数，封装成一个异步请求发给腾讯云客户的物理服务器，然后物理服务器会调用私钥进行相关计算，再把计算结果返回给腾讯云服务器，完成 HTTPS 卸载。

整个交互流程中，STGW 或者说腾讯云和私钥没有任何的接触，私钥是客户自己才拥有。这就是我们无密钥加载的一个简单的介绍。当然具体的流程或者实现，其实跟我刚才提到的异步代理计算有点类似，包括协议方面（见图 16）。

证书优化

接下来看证书的优化。如果是个人用户，向大家推荐 Let's Encrypt，它是免费开源的 CA 证书颁发机构，它的

优点就是免费开源，开源最大的好处就是可以对协议对整个交互流程都很清楚，支持 Debug。它还有一个优点是能支持自动部署，它提供几个工具运行几分钟就可以把证书申请下来。

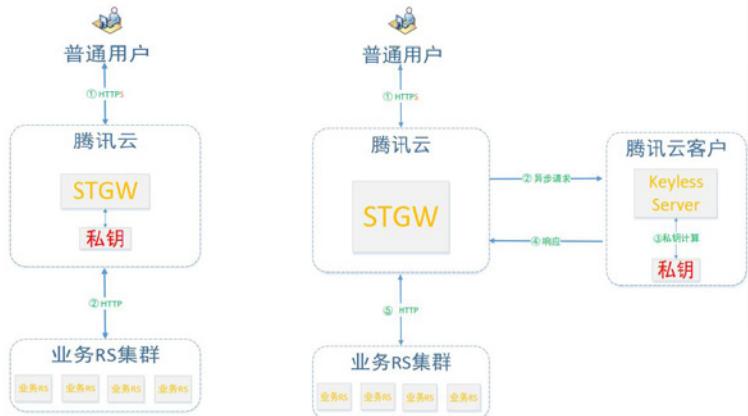
但它也有缺点，第一，它只是低级别即域名级别的证书申请，它只对域名的有效性进行校验，你能拥有这个域名就可以给你颁发证书，这就有安全风险了，假如我对域名进行劫持，我就可以冒名申请别人的证书。

另外兼容性也会比较差，像 PC 端 Chrome 访问没有任何问题，但我们用 Android 7.0 去访问就有可能出现问题，因为 CA 支持和更新并不及时，因此就有兼容性的风险，所以推荐个人用户使用。

对于企业用户，建议大家使用 EV 和 OV 级别的，因为它会需要你证书的机构、你的地址、法人信息、营业执照、在工商局的认证等信息，才确保只有你才能申请这个证书。

这是云的优势，申请很简单，一键式申请，不需要你自己生成私钥，也不需要生成证书的申请文件。这是腾讯云证书简单的说明。我们还有一个动作，在跟最大的证书厂商进行合作接触，我们会实现自己更低成本的自主品牌证书的颁发，这样的话证书会更加便宜，下图是腾讯云上面的介绍。

我们看一下证书签名的选择，我们最后使用了 RSA，还使用 ECDSA，这两个是最主流最常用的签名类型。



● RSA

- 兼容性好
- 服务端性能差

● ECDSA

- 兼容性差
 - ◆ XP不支持
 - ◆ 支持ECDHE，但系统缺少root ca
- 服务端性能好，客户端性能差

● 同时支持

- 成本增加

RSA的好处是兼容性好、历史悠久，所有的客户端都支持，因为它算法已经存在40年了，缺点就是之前提到的代理计算，RSA需要私钥进行计算，服务端需要加密解密性能会比较差。

ECDSA，优点就是服务端的性能好一点，P-256只需要256位的输出域就能实现和2048位长度同样的安全性，它的安全性更高，服务端计算性能要好，但是客户端的性能要差。使用ECDSA P-256客户端的计算量要比RSA要大，

浏览器	最低版本
Apple Safari	4 (On ECC Compatible OS)
Google Chrome	1.0 (On ECC Compatible OS)
Microsoft Internet Explorer	7 (On ECC Compatible OS)
Mozilla Firefox*	2.0

操作系统	最低版本
Apple OS X	OS X 10.6
Google Android	4.0
Microsoft Windows	Windows Vista
Red Hat Enterprise Linux	6.5

图 16-图 17

- **SHA1 or SHA256**
 - SHA1不安全
 - SHA2兼容性差
- **不支持SNI = 不支持SHA2 ?**
- **Nginx配置**
 - 证书一 server_name空

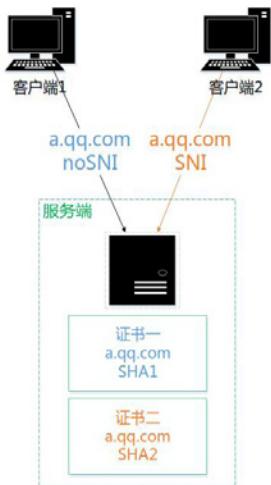


图 18

RSA 的公钥的数字非常小，只有 65537，而 ECDSA 的客户端公钥长度很大，计算性要差很多，特别是手机计算性能要弱。

ECDSA 的缺点就是兼容性差，比如说 WindowsXP 不支持。具体可以看下图的系统支持情况，这里面其实由于 RSA 和 ECDSA 这两个完全不同类型的证书，服务端其实可以同时支持，当然成本可能会稍微增加一些。目前使用 ECDSA 的还是多一些的（见图 17）。

最后看一下 SHA1 和 SHA256，SHA1 不安全，像 Google 和 Microsoft 已经宣布放弃 SHA1，如果你使用 SHA1 证书就访问不了。

SHA2 兼容性比较差，比如 WindowsXP 不支持 SHA2 证书，但对于需要支持 WindowsXP 的业务来说如何兼容呢？我们观察到一个特性：不支持 SNI 的客户端也不支持 SHA2。

SNI 是 TLS 的一个特性和扩展，发起 client hello 的时候，它会携带上

一个域名的信息，在握手的时候告诉服务端要访问哪一个域名（Server Name Indicator），虽然从协议或者从原理的角度来讲没有任何的必然的关系，但这个现象给我们一个启发：我们通过服务端配置能够解决这个兼容性的问题。

同样以 Nginx 和 OpenSSL 举例，假如说客户端 1 不支持 SNI，只支持 SHA1，客户端 2 是新的系统支持 SNI。我们配置配两张证书，第一张证书我们同样都是接 403 端口和两个不同的 Server，证书 1 放在前面，证书 2 支持 SHA2 放在后面。

当客户端 1 发起请求的时候，由于它不支持 SNI，所以默认会返回第一张证书（第一张证书是支持 SHA1 的），这样不支持 SNI 也不支持 SHA2 的客户端 1 兼容性没有问题。

客户端 2 发起请求的时候，携带了 SNI，由于有 SNI 和域名信息，服务端会挑选有域名的信息 SHA2 返回给客户

●更广 ●更快

- http2主流实现强制使用https
- ATS 强制使用HTTPS
- Chrome mark http unsecure
- TLS1.3
- QUIC

●更强 ●更开放

- RSA 2048 -> 4096
- RSA -> ECC
- Let's encrypt

图 19

端，实现 SHA2 的兼容（见图 18）。

最后是我对于 HTTPS 发展的简单的概括和理解（见图 19）：

更广。它会越来越流行，以后会成为互联网的标配。HTTP2 作为下一代的协议，主流实现像 Chrome\FireFox 都是强制使用 HTTPS 的。ATS 也强制使用 HTTPS，Chrome2017 年也会将 HTTP 标识为不安全；

更快。这里涉及到访问速度的优化，TLS1.3 是革命性飞跃式的 TLS 协议，可以认为是 TLS2.0，它相比 1.2 有非常大的变化，最大的变化有完全握手，它只需要一个 RTT，而简化握手不需要 RTT，也就是应用层数据不需要握手直接可以完成数据的传输，另外 QUIC 解决 HTTPS 协议的性能问题，TLS 协议以后也会越来越快，大家不用担心访问速度的问题；

更强。密钥长度越来越大，加密强度会越来越强，随着客户端和硬件的发展，CPU 加解密的性能也会越来越强；

更开放。之前提到类似 Let's Encrypt 的免费开源方案，以后也会成为越来越流行的方案。

以上是 HTTPS 的性能优化，谢谢大家！

作者介绍

罗成，腾讯资深研发工程师。目前主要负责腾讯 stgw（腾讯安全云网关）的相关工作，整体推进腾讯内部及腾讯公有云，混合云的七层负载均衡及全站 HTTPS 接入。对 HTTPS，SPDY，HTTP2，QUIC 等应用层协议、高性能服务器技术、云网络技术、用户访问速度、分布式文件传输等有较深的理解。

来自微信团队的6个开源项目

作者 郭蕾

开源改变世界，这不仅是一种态度，更是一种能力。微信团队 2016 年在开源上动作不断，InfoQ 带你一起看看，他们的开源项目与开源态度。

写在前面

从 GitHub 上可以看出，到目前为止，由腾讯微信团队发起的开源项目已经有 6 个，并且这其中大部分都是在 2016 年开源的，领域涉及移动、数据库、基础类库、框架。中国大公司的开源曾经给社区留下不好的印象，比如有人就这么说，大公司的开源，开源也就意味着结束。也有人说，大公司的开源大部分都是 KPI 项目，开源与业务不能相得益彰，所以根本无法持续投入。

相比来说，Facebook 就是互联网公司里的开源大户，目前他们的开源项目已经有近 300 个。对于为什么要发布开源项目，Facebook 开源项目负责人也曾经解释过，总结起来说有三点，一是开

源能够帮助他人更快地开发软件，促进世界创新，主要是社会价值层面的考虑。二是开源能够倒逼 Facebook 的工程师写出更好的代码。三是开源能够更有效利用社区的力量，帮助 Facebook 一起解决难题。

那微信是如何理解开源这件事，以及他们未来准备通过哪种机制保证开源项目的健康发展，带着这些问题，InfoQ 记者采访了微信终端团队的负责人赵原。

开源不仅是一种态度，更是一种能力，微信希望通过开源打通内部团队和外部社区，一方面可以把微信的顶级技术输出给了社区，另一方面也可以把外部的优秀的思想传递到内部团队。对于开源的理解，赵原这样解释。



第一个关键词**价值**。微信团队通过将内部的研究成功优化、整合，并输出，以帮助更多的开发者更方便地构建他们的软件。工程师之间最好的交流媒介就是代码，通过代码，微信团队可以将他们的技术思想传递到社区，并影响更多的人。

第二个关键词**优势**。微信最大的优势是有海量的用户，很多外界开发者根本没有发现或者遇到过的复杂问题，微信团队都早已经解决掉了，比如在移动开发领域的系统兼容性、用户多样化的网络环境等问题。通过开源，微信不仅可以授之以鱼，还可以授之以渔。

第三个关键词**活力**，微信开源的项目必须来自微信，最后必须在微信落地。开源项目最怕没人维护，通过产品迭代，这些开源的项目持续的更新，给他们一

个心跳的机制，使他们依然有活力。

第四个关键词**易用**。一个优秀程序员可以花上很多的时间研究微信开放的技术源码的技术思想和设计。但对于不是很有技术实力的开发人员，是否能从微信的开源项目中依然获益呢？答案是肯定的。他可以将微信开源的项目引入在自己的APP中，将微信很多的能力运用在自己的APP中。

而对于项目开源之后的运营问题，赵原也做了思考。

活力，不得不提起一个尖锐的问题，在大公司里面做开源项目，开源通常意味着这个项目的结束。优秀的开源项目意味着开始，和大公司里的开源项目做法天生有一些冲突。

第一大公司里面做开源项目其实是一些短期KPI导向的短期项目，开源

项目需要持续的人力投入在其中。第二个问题，每一个团队里面员工开发的精力是有限的，做技术研究的时候，需要花很多时间在供应商的开发，这是从员工方面看开发精力的冲突。第三大公司内通常有完备的开发体系，而这些开发是封闭的，很难说和外部的开源体系对接起来。

今天看这个问题，微信也想过很暴力、很简单的想法，比如说团队 10 个人，这 10 个人可以做功能开发，如果为了解决这个问题，再招 10 个员工砸在开源项目上，大公司可能最不缺的就是人力，问题是在大公司里面看上去是可以解决的，但放在微信上面却觉得行不通，微信讲究小团队作战，讲究精兵策略，不浪费一兵一卒。

为了解决这个问题，微信在开发 Tinker 项目的过程中，大概用了快一年的时间解决，其实解决问题的方法很简单。简而言之，就是将微信的开发团队改造成一个开源化的开发模式。

微信通过对内部系统的改造，使内部的开发和外部的开发模式基本没什么不同。一份代码，既可以在微信中使用，也可以在外部的开发者中使用，这科技减少额外的人力投入。

另外微信还收获了很多价值。比如外部的开发者还可以提供一些微信团队没有发现的问题，对于微信来说是一个很好的优化 bug 的渠道。通过与外部社区的深入交流，团队的学习能力也能得到很大提升。

下面是 InfoQ 编辑整理的微信现有的比较活跃的开源项目列表，欢迎交流讨论。

C/C++协程库Libco

Libco 是微信后台大规模使用的 C/C++ 协程库，2013 年至今稳定运行在微信后台的数万台机器上。Libco 提供了完善的协程编程接口、常用的 Socket 族函数 Hook 等，使得业务可用同步编程模型快速迭代开发。

早期微信后台因为业务需求复杂多变、产品要求快速迭代等需求，大部分模块都采用了半同步半异步模型。接入层为异步模型，业务逻辑层则是同步的多进程或多线程模型，业务逻辑的并发能力只有几十到几百。随着微信业务的增长，系统规模变得越来越庞大，每个模块很容易受到后端服务 / 网络抖动的影响。基于这样的背景，微信开发了 Libco，实现了对业务逻辑非侵入的异步化改造。

GitHub 地址：<https://github.com/tencent/libco>

Star 数量：1043

生产级paxos类库PhxPaxos

PhxPaxos 是微信后台团队自主研发的一套基于 Paxos 协议的多机状态拷贝类库。它以库函数的方式嵌入到开发者的代码当中，使得一些单机状态服务可以扩展到多机器，从而获得强一致性的多副本以及自动容灾的特性。PhxPaxos

在微信服务里面经过一系列的工程验证和大量的恶劣环境下的测试，在一致性的保证上极为健壮。

PhxPaxos 的特性包括使用基于消息传递机制的纯异步工程架构、每次写盘使用 fsync 严格保证正确性、支持 Checkpoint 以及对 PaxosLog 的自动清理、使用点对点流式协议进行快速学习、支持跨机器的 Checkpoint 自动拉取、内置 Master 选举功能、自适应的过载保护等。

GitHub 地址: <https://github.com/tencent-wechat/phxpaxos>

Star 数量: 970

高可用、强一致的MySQL集群

PhxSQL

PhxSQL 是一个兼容 MySQL、服务高可用、数据强一致的关系型数据库集群。PhxSQL 以单 Master 多 Slave 方式部署，在集群内超过一半机器存活的情况下、即可提供服务，并且自身实现自动 Master 切换、保证数据一致性。PhxSQL 不依赖于 ZooKeeper 等任何第三方做存活检测及选主。PhxSQL 基于 MySQL 的一个分支 Percona 5.6 开发，功能和实现与 MySQL 基本一致。

MySQL 主备在主机上支持完整 SQL、全局事务、以 repeatable read 和 serializable 级别的事务隔离，在金融、帐号等关键业务中有巨大的价值。但是 MySQL 传统主备方案也有其缺点。最明显的就是主机故障后的自动换主和

新旧主数据一致性，即所谓的一致性和可用性。为了解决这个问题，并同时完全兼容 MySQL，微信在 MySQL 的基础上应用 Paxos，设计和开发了 PhxSQL。

GitHub 地址: <https://github.com/tencent-wechat/phxsql>

Star 数量: 1485

RPC 框架 PhxRPC

PhxRPC 是微信后台团队推出的一个简洁小巧的 RPC 框架，编译生成的库只有 450K（编译只依赖第三方库 Protobuf）。PhxRPC 的特性如下：

使用 Protobuf 作为 IDL 用于描述 RPC 接口以及通信数据结构。

基于 Protobuf 文件自动生成 Client 以及 Server 接口，用于 Client 的构建，以及 Server 的实现。

半同步半异步模式，采用独立多 IO 线程，通过 Epoll 管理请求的接入以及读写，工作线程采用固定线程池。IO 线程与工作线程通过内存队列进行交互。

提供完善的过载保护，无需配置阈值，支持动态自适应拒绝请求。

提供简易的 Client/Server 配置读入方式。

基于 lambda 函数实现并发访问 Server，可以非常方便地实现 Google 提出的 Backup Requests 模式。

GitHub 地址: <https://github.com/tencent-wechat/phxrpc>

Star 数量: 467

终端跨平台网络组件：Mars

Mars 是微信官方的终端基础组件，是一个结合移动应用所设计的基于 Socket 层的解决方案，在网络调优方面有更好的可控性，采用 C++ 开发。目前已接入微信 Android、iOS、Mac、Windows、WP 等客户端。

在微信中，任何网络实现的 bug 都可能导致重大事故。例如微信的容灾实现，如果因为版本的实现差异，导致某些版本上无法进行容灾恢复，将会严重的影响用户体验。微信研发了统一的跨平台的网络基础库 Mars 来满足发展的需要，一方面，基础组件可以提高研发效率，另外一方面，也可以提高系统的稳健性。

在设计上，Mars 以跨平台、跨业务为前提，遵从高可用，高性能，负载均衡的设计原则。以网络的可用性为例，移动互联网有着丢包率高、带宽受限、延迟波动、第三方影响等特点，使得网络的可用性，尤其是弱网络下的可用性变得尤为关键。Mars 的 STN 组件作为基于 socket 层的网络解决方案，在很多细节设计上会充分考虑弱网络下的可用性。

GitHub 地址：<https://github.com/Tencent/mars>

Star 数量：5895

热补丁技术Tinker

Tinker 是微信官方的 Android 热补

丁解决方案，它支持动态下发代码、So 库以及资源，让应用能够在不需要重新安装的情况下实现更新。

当前市面的热补丁方案有很多，其中比较出名的有阿里的 AndFix、美团的 Robust 以及 QZone 的超级补丁方案，但它们都存在无法解决的问题，所以微信研发了自己的解决方案。总的来说，AndFix 作为 native 解决方案，首先面临的是稳定性与兼容性问题，更重要的是它无法实现类替换，它是需要大量额外的开发成本的。而 Robust 兼容性与成功率较高，但是它与 AndFix 一样，无法新增变量与类只能用做的 bugFix 方案。Qzone 方案可以做到发布产品功能，但是它主要问题是插桩带来 Dalvik 的性能问题，以及为了解决 Art 下内存地址问题而导致补丁包急速增大的。

Tinker 的具体设计目标如下：

开发透明：开发者无需关心是否在补丁版本，他可以随意修改，不由框架限制。

- 性能无影响：**补丁框架不能对应用带来性能损耗。
- 完整支持：**支持代码，So 库以及资源的修复，可以发布功能。
- 补丁大小较小：**补丁大小应该尽量的小，提高升级率。
- 稳定，兼容性好：**保证微信的数亿用户的使用，尽量减少反射。

GitHub 地址：<https://github.com/Tencent/tinker>

Star 数量：6707



ABOUT | 关于 EGO »

EGO 是极客邦科技旗下高端技术人聚集和交流的组织，以 CTO、技术 VP 等技术领导者为服务对象，旨在组建全球最具影响力的技术领导者社交网络，线上线下相结合，联结杰出的技术领导者学习和成长。

EGO 采用会员推荐制，每位会员需通过 EGO 现有会员以及业内领袖推荐，并经过 EGO 各区域面试官面试之后，付费加入。最大限度保证加入的会员属于同等级别，让各位管理者在相互理解信任的环境下分享交流。EGO 目前已经成立北京、上海、深圳、广州、杭州、成都六大分会，也将陆续拓展国内外互联网技术聚集区域，建立当地的技术管理者社交组织。

MEMBER RIGHTS | 会员权益 »

学习交流

私密独享的线下小组交流学习活动；顶尖技术专家、商业领袖经验分享；跨领域、跨地区的闭门会议；软技能提升

拓展人脉

全球技术领导力峰会、企业互访、技术管理者出海等活动，结识志同道合的朋友，搭建国内外技术管理者社交的平台，搭建会员企业间沟通合作的平台

解决问题

会员遇到的问题通过 EGO 线上社区获得帮助，EGO 成为会员身边的智囊团

获取信息

及时获取最新行业资讯，独家原创专家观点，热点问题独到解析

会员福利

优先参与极客邦科技举办的各类技术会议，优先申请讲师，并享有会员优惠购票政策，联结会员企业资源，打造会员专享资源生态圈



GTLC全球技术领导力峰会 »

GTLC 是由 EGO 主办的高端技术领导人盛会。2016 年，在中国首次实践，大会以“重新定义技术领导力”为主题，汇聚 400 多位优秀技术管理人才，联合 10 多位业内顶尖 CTO 组成大会讲师团，旨在分享、探讨技术管理过程中的最佳实践。EGO 每年都会举办全球技术领导力峰会，为全球优秀的技术领导者们打造高质量的技术管理交流、学习平台。



ABOUT | 关于斯达克学院 (StuQ) »

极客邦科技旗下实践驱动的 IT 教育平台，通过线上和线下多种形式的综合学习解决方案，帮助 IT 从业者和研发团队提升技能水平。

从 2016 年 1 月份开始，斯达克学院（以下简称 StuQ）正式对外提供学习服务，目前已经有来自 10 个领域的 216 名一线技术专家，作为讲师在 StuQ 分享了自己的研发实践；有超过 6 万名学员提升了自己的职业技能，包含 9000 名付费学员；有 267 家企业通过 StuQ 提供的企业内训、工作坊和轻咨询等服务提升了研发团队的能力。

COURSES | 十大技术领域课程 »



数据科学



移动开发



构架设计



云计算



运维开发



前端开发



后端开发



技术管理



产品设计



产品测试

个人学习服务 › 通过一线技术专家的传道授业解惑，全面提升个人的研发能力和软实力。

公开课 › 借助知乎 Live 等轻便的方式，StuQ 每周邀请专家就某一个话题进行探讨，强调对某一个技术的介绍或者如何提升自己的软技能，比如职业生涯规划、团队沟通等。

直播课 › StuQ 和一线专家共同策划系统课程，借助 Zoom 等先进教育直播工具，持续就某一个技术进行解析，旨在通过实践性的内容，帮助学员掌握如何在实际项目中应用该技术。

企业学习服务 › 通过资深技术专家带来的实战课程，帮助研发团队提升综合技能水平。

企业内训 › 针对企业内部研发团队学习计划和技能需要，一线技术专家在线沟通诊断，为企业定制专业的技术培训服务，提供多样化的研发体系课程，帮助企业解决研发过程中的问题和困难，促进团队快速成长。

工作坊 › 针对同领域、不同企业的技术人员开展线下精品小班课，携手一线资深技术专家，为技术人成长提供必修精选课程，课程设计系统、全面、注重实践分享，帮助技术人提升职业技能。

轻咨询 › 研发团队接受过培训之后，在生产过程中依然遇到许多具体的挑战和疑惑，来自一线的资深技术专家通过线上或线下的答疑解惑，帮助研发团队快速度过难关，大幅度提升研发效率

EXPERTS RECOMMEND

专家推荐

▶ 程立 / CHENG LI

持续关注InfoQ好多年了。由于工作繁忙没有很多时间泡技术社区，我一直选择坚持精品与原创路线的InfoQ作为获得业界信息的主要来源。当遇到难题时也会到InfoQ上寻找灵感并常常有所收获，可以说InfoQ是我的老师、智囊和朋友，借此机会向InfoQ说声谢谢！

▶ 王文彬 / WANG WENBIN

InfoQ办的QCon大会是一个高质量的盛宴，对于最新的互联网技术和最佳实践一直在做探讨。除了邀请国内的牛人，也会有国外的大牛来做分享，对技术人员是一个不可错过的大会。

▶ 冯大辉 / FENG DAHUI

InfoQ，技术人都喜欢。几年下来，通过InfoQ网站获得了许多有价值的资讯，通过InfoQ的电子杂志借鉴到很多技术思路，而通过InfoQ举办的数次QCon大会，又结识了不少业界朋友。期待InfoQ坚持自己的特色，期待越办越好！

▶ 杨卫华 / YANG WEIHUA

InfoQ每年遍布全球的QCon大会是技术界的盛会，给业界很多研发方向上的启发，新浪微博的技术架构也从往届QCon大会演讲中获取了不少宝贵经验。

▶ 洪强宁 / HONG QIANGNING

InfoQ是我获取业内最先进的技术和理念的重要渠道。在InfoQ的帮助下，我也得以与国内外众多技术高手交流切磋，获益匪浅。感谢InfoQ！

▶ 吴永强 / WU YONGQIANG

接触InfoQ，包括QCon，已经有好几年了，我非常喜欢它的风格，灵动、快速、实用，Moq网站、QCon、《架构师》杂志都能够紧贴互联网技术的发展前沿，带来大量的最佳实践，对我们这样发展中的公司的帮助非常大。希望InfoQ能够越做越好！

▶ 卢旭东 / LU XUDONG

我很早就是InfoQ的注册用户了（哈哈，有好几年了吧，持续保持潜水状态），它一直是我们了解业界研发趋势，学习先进技术和方法的最好平台！在这里还能认识很多志同道合的朋友，InfoQ有潜质成为国内最专业、最大、最有影响力的研发社区！InfoQ的电子杂志更是必看，深浅结合，对实践很有指导性。

▶ 毛新生 / MAO XINSHENG

InfoQ社区是架构师的一流资讯来源，也是大家交流的桥梁。