Geekbang. 极客邦科技 InfoQ

QCon十周年特刊

全球软件开发大会

Floyd Marinescu, C4Media 创始人兼 CEO

Welcome to the 10th annual QCon Beijing!

The mission of QCon and InfoQ world wide is human progress: the software side. We do this by helping software developers like you adopt new technologies and practices by learning from each other. We believe you want to learn from your expert peers, which is why we go to great lengths to bring in top early adopter software teams world-wide to speak at our events about their successes and failures on their projects.

The leadership at QCon and InfoQ are also current and former software developers, which is why we have designed a high quality experience for developers in the way that a developer would want, putting content and community first.

We have a special love for China. Since the founding of C4Media (a Canadian company) in 2006, our very first hire after the founders was a staff member in China, who began internationalizing InfoQ.com. InfoQ China website launched on year after the English website in 2007, and is now under 100% Chinese ownership through our sister company Geekbang.

I personally spent a lot of time in Beijing tea shops back in 2007-2010 in training and discussions with the original team led by CEO of Geekbang Kevin Huo, and look back to these memories with great joy.

We new that China would rise as a major technological super power and we are grateful to have been of service to the software development community in China. Our goal was to provide the best ideas from the world wide software developer community on InfoQ & QCon and also to facilitate the best thought leaders in China to connect with each other and with you.

It is with great pleasure that I will deliver opening keynote remarks at this 10th annual QCon Beijing. See you soon and thank you for your patronage!

记录历史，是一种责任

极客邦科技创始人兼 CEO、InfoQ 中国创始人 霍泰稳

2019 年是 QCon 进入中国的第十个年头，有好多朋友问我，十年前为什么想着要做这么一件事情？其实和多数伟大事件的开始一样，初心很简单，就是想给国内的技术人员呈现一个高品质的大会。十年前，国内付费会议的概念还远未流行，大家崇尚的更多是免费思维，而与之相关联的就是在会议上的投入度不够，质量参差不齐。而我和团队坚信的一点是，对于高级工程师、架构师这个级别的同学，相比于参会的费用，他们的时间更为宝贵，我们有必要加大投入，为他们做高水平的学习活动。

事实证明这条路是正确的。站在当下回头看，在这十年里，我们一共交付了 42 场大会，策划了 632 个专题，邀请了 3,037 位行业专家来分享，有 8,915 家企业参加，服务了 36,501 名参会者。不客气地说，InfoQ 极客传媒举办的会议，专家来源囊括了所有中国领先的互联网公司，他们也代表公司分享了最有价值、最具有实践性的分享。

过去的十年，也是互联网技术迅猛发展的黄金十年，目前我们看到的主流技术几乎都诞生于其间。比如移动互联网。2007 年 1 月乔布斯发布第一款 iPhone，2008 年 9 月 Android 系统 1.0 正式发布。云计算亦是如此。2006 年，Amazon S3 正式发布。2008 年，微软发布 Windows Azure 预览版，Google 发布 Google App Engine 预览版。2010 年 OpenStack 发布。2012 年，UCloud、青云等国内云计算公司成立，阿里云开始对外大规模为用户提供云计算服务。

同理，人工智能、区块链、容器这些技术也都经历了这样的发展。综上所述，这其中每一门技术，从一颗小小的种子开始，到长成苍天大树，这中间有太多精彩绝伦的故事，关于人、关于公司、关于时代。而 QCon 和其背后的 InfoQ 极客传媒，一直在记录这一切，我们在不知不觉中已经为技术的历史留痕。

历史的车轮总是滚滚向前，过去的十年，我们经历了 PC 互联网，移动互联网。从前我们说“ 软件正在改变世界 ”，现在我认为 “ 软件正在定义世界 ”，特别是在这个即将到来的产业互联网时代，这一切将会表现的淋漓尽致。这不仅是我的观点，比尔·盖茨也曾经说过，未来任何一个工业都会变成软件工业，因为任何工业任何行业自动化的程度会越来越高，而自动化的过程本身就是软件化的过程。

“ 网上得来终觉浅，绝知此事在现场 ”，我想极客邦科技还是会像之前一样，借助会议这种看似很笨重实则最有效的方式，继续参与其中，同时，记录这一切正在悄然发生的改变。回归到极客邦科技的使命，就是要通过我们每一个小伙伴的努力，筛选出全球最优质的学习资源，为用户交付前沿的技术趋势和最佳实践，从而让创新技术推动社会进步。

最后，我想说，感谢十年来结伴而行的你。能为技术人服务，我很骄傲，也很自豪，我和团队会一直在这条路上走下去。

目录 | Contents

05 运维十年回顾，当前很多新技术的本质都是在解决运维问题

10 云计算十年回顾（上）：风雨兼程

14 云计算十年回顾（下）：势不可挡

20 移动开发十周年：开源节流，创造新生态

39 容器混合云，Kubernetes 助力基因分析

42 阿里云 PB 级 Kubernetes 日志平台建设实践

46 做混沌工程是什么样的体验？阿里：有点刺激

50 “容器”十年，关于软件交付的编年史

运维十年回顾
当前很多新技术的本质都是在解决运维问题

作者 赵成

在技术发展 10 年这个特殊的时间节点上，我们邀请了蘑菇街技术总监赵成来谈谈他在过去十年间的感受。一起回顾一下运维行业十年来的发展变化和趋势，以及这中间的演进逻辑，以期给更多的运维同行一个参考。

2008 年到 2019 年这 10 年多的时间里爆发了很多重要的技术和技术浪潮，运维技术也在这十年间发展到了深水区。随着云计算技术的普及以及容器技术的兴起，运维效率大大提升，运维平台得以将运维人员从繁重的人工操作中解救出来；而人工智能的发展也使得 AIOps 成为可能，让运维人员能够先用户发现故障，更好得保障业务运行。

很高兴能在 QCon 10 年之际接到邀请，写一写运维行业发展的这十年，非常感谢 InfoQ 社区的邀请和信任。

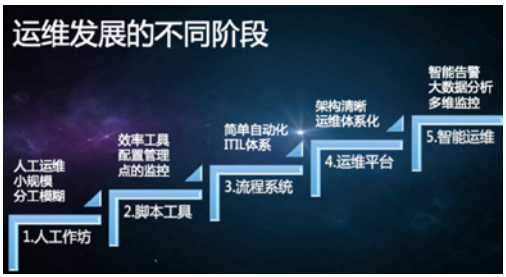
在我正式写文章之前，我仔细回顾了一下我个人经历的运维的过程，也去翻了很多其他公司公开能看到一些分享材料，也找很多业界的同事做了交流，让他们也一起回忆一下过往的经历，因为十年很长，还是有很多东西值得回味和探讨。

最终，我总结出 5 个结论，也是规律，分享给大家，期望带给各位读者和所在的企业一些思考和启发：

- 第一，运维行业的发展，是有规律可循的，是一个逐步演进的过程。这也说明，其实我们有很多经验可以向先行者们学习。
- 第二，运维行业的发展，不是孤立的，它与业界的整个技术趋势发展是相辅相成的。这就要求，关注运维的同时，我们也要关注整个技术趋势和背景。

- 第三，运维行业真正高速地发展，真正地被重视，其实就在最近 5、6 年。运维这个行业还很年轻，仍有非常大的发展空间。
- 第四，运维行业当前的痛点，本质上更多的是企业层面的痛点，而不是运维个体的痛点。所以，要尝试自上而下的解决问题，而不是自下而上。
- 第五，未来，一定是云计算的时代，未来已来，只是分布不均。所以，云计算时代下的运维转型升级，将是一个非常明确的方向。

如果用一张图表示这 10 年运维发展的过程，下图再合适不过：



接下来，我们分过去、现在和未来三部分来分享一下我对运维发展过程及未来趋势的理解。

过去（2009-2013）人工运维

第一个阶段，人工作坊阶段，也就是我们遇到的所有运维问题，基本靠人工操作完成。这种情况下，系统规模不大，遇到的问题相对简单，大多集中在硬件、网络和系统层面，所以有一定操作系统或网络维护经验的人就可以搞定。

这种场景下的运维，也就是我们常说的 SA，系统管理员，而且一般身兼多职，人数也不太多。

第二个阶段，脚本工具阶段，一般绝大多数企业都会很快从第一阶段过渡到第二阶段，因为上一阶段的大量重复繁琐的操作，完全可以转化为脚本来实现，而不是每次都去敲一堆类似的命令。

早期的 SA 主要以各种 shell 为主，所以很多 SA 如果会 shell 编写一些批处理脚本，就会很有竞争力了。再往后，我们大家所熟知的 Perl、Ruby、Python 等动态语言也被广泛应用于脚本工具的实现，特别是一些逻辑和场景相对复杂的自动化实现。

第三个阶段，流程和工具阶段，当我们把一些复杂的操作封装成一个个的脚本后，效率确实会提升很多，但是我们所面对的业务场景和体量也在变得更复杂。比如，对于运维同学，以前就是负责安装和配置一下操作系统，如果是几十台或百台的规模，脚本批量执行完全可以搞定。

但是，再往后，运维还要负责软件的频繁发布，每周要多次，甚至是每天都会有，这是由业务特点决定的，特别是互联网类型的业务，与原来传统的每个月、甚至几个月发布一次的场景要求完全不一样了。而且随着用户体量的增加，服务器数量可能已经到了几百上千台，而且部署的业务也不尽相同，所以单纯靠脚本执行，已经完全不能满足要求。

这时候，就要面临更加复杂化的场景实现，比如做一次业务部署，运维同学可能要安装服务器，做系统配置变更，安装软件包、启停进程，然后再负载均衡上配置服务 , 等等。这时，就需要有一个流程将一个个的脚本功能串联起来，同时还要有一些脚本执行结果校验及判断的过程。

所以，这就对流程和工具平台有了更大的诉求。同时，在一些 IT 化比较早的行业，如电信运营商和金融行业，由于对变更过程的严格控制，这就需要更加科学和规范的管理措施，所以会引入 ITIL 这样 IT 服务管理体系，对整个 IT 系统及其变更进行管控。

其实，第一到第三阶段，在 2009 年到 2013 年期间仍是绝大部分公司主流的运维模式。如果能够做到工具化，或者有一些工具化平台，那应该算是比较先进的运维模式了。

现在（2014-2019）自动化运维

但是，对于一些大厂，步伐会更快一些。2013-2014 年，就已经有国内大厂进入到了第四个阶段，运维已经体系化，完全的自动化。

直到目前为止，从笔者交流和了解到的实际情况看，绝大多数企业基本都在第四阶段的建设过程中，所以我们把 2014-2019 这个阶段定义为

现在。

到了这个阶段，就凸显出几个明显的特点，也是我们上面提到的其中几个规律，我们分别说一下。

第一，国内的运维行业的爆发，运维岗位真正地被重视，其实就是近 4、5 年左右的事情，也就是 2014 开始到现在。

为什么这么讲？其实我只要关注下，运维行业有自己垂直的技术大会，类似 QCon 以及 ArchSummit 这样的顶级技术峰会，开始专门设置运维专题，基本就是在 14 年左右开始的。这个现象说明，运维的技术复杂度已经上升到了一定程度，各大企业也开始意识到运维对于企业的效率、稳定和成本有着决定性作用，对运维的诉求和要求也越来越高。

从这个角度讲，新兴的运维行业其实才算是刚刚起步，未来仍然会有很大的潜力和空间。

第二，运维的发展不是孤立的，它与整个技术趋势发展是相辅相成的。

大厂之所以在这方面会走在前列，一方面是因为业务复杂度和体量所决定，另一方面，是因为这样的场景倒逼着整个业务和技术架构发生了很大的变化。比如我们现在早已耳熟能详的服务化和各类分布式技术，就是在这种场景下倒逼着技术体系演进出来的。

也正是在这样新的技术体系下，运维所面临的场景复杂度也急剧上升，原有的运维技能如操作系统维护、系统配置、脚本编写已经完全满足不了要求。同时，由于软件系统复杂度的提升，也需要运维投入更多的精力去关注业务软件架构和应用服务上。

所以，这种场景下，我们所熟知的 DevOps，SRE、PE、应用运维、技术运营这些新的名字、岗位或理念，开始如雨后春笋般浮现出来。

其实这里很多概念早在 10 多年前就已经出现了。比如 SRE 最早是在 2003 由 Google 提出；DevOps 理念的影子在 2007 年左右就开始浮现出来，2009 年的 DevOpsDays 大会上正式提出了这种叫法；而像 PE 这样的角色，最早是在 Yahoo! 设置的。

这些优秀的运维或者稳定性的理念，在国内兴起前，其实已经在国外被广泛实践了很多年。究其原因，还是因为国外的技术发展是超前于国内的，比如 Google 在分布式领域的“ 三驾马车 ”，直接开创了一个新的技术时代，让业界有机会充分实践分布式的技术。

这里，我想表达的一个观点是，这些理念在国内真正的落地，还是因为有实际的场景驱动。业务体量和复杂度到了那个程度，技术体系必然会找朝着分布式的方向发展。而配套的，必然会有 SRE、DevOps 以及 PE 这样相辅相成的体系出现。

国内大厂有机会提前走到这一步，从绝大部分公司发展的过程看，也必然会遵循这样的规律，只是早跟晚，快跟慢的问题。这里的决定性因素其实是业务复杂度和体量所决定的。

目前很多企业和公司之所以能走到运维的第四阶段，其实很大程度上也是因为广泛采用分布式技术，引入了服务化和各类分布式组件，在这种情况下，业务架构越来越清晰，对应的对运维的诉求和要求也就逐步提升了上来。

典型的技术和发展特点

从技术角度，我们关注下这 4、5 年来技术的发展，说两个最典型的：

一个是容器技术，以及以容器为核心的编排系统，现在基本是以 K8S 为标准了。

首先，Docker 的创始人 Solomon 其实是运维出身，并不是做开发的。当时他的想法也很简单，就是希望能够屏蔽一些跟应用无关的底层细节，Run any application,anywhere，提升部署和发布的效率。

后来一经推出，大受关注，特别是在 14 年左右，可谓是大红大紫，且围绕着容器的一整套生态也在逐步成长起来。到目前为止基于容器和 K8S 的基础平台，已经成为 PaaS 体系建设的标准，如果哪项技术不适配 K8S，那基本是没有发展空间的，也基本不会被认可。

从运维的角度看，容器解决的最大的问题就是运维的问题，特别是运维的效率问题。从目前业界的实践来看，容器确实发挥了极大的作用。

现在更为极致的一种理念是**无服务器技术**，也就是我们熟知的 Serverless，也叫函数服务器。这种理念极致的地方在于，以后纯粹就是 NoOps 的时代，开发写完代码直接部署发布到云上，完全不用考虑服务器和资源的问题。这种场景无疑是将整个迭代周期压缩到了极致，理想状态下，让整个技术团队无需考虑运维的事情。

但是，实际场景下，新技术发展仍需要一定周期和周边配套体系完善。同时，新技术能够发展完善，也需要找到适合自己发挥的业务场景，有时新技术出现并不是要为了完全替代早期的技术。

简单总结一下，我们会发现，当前非常多的新技术和新趋势的产生，从本质上都是在解决运维问题，未来也一定是这样一个趋势。

从最佳时间角度，到了这个阶段，从我个人认为，现在业界运维问题，更多的是企业层面的运维问题，而不是个体运维的问题。这一点跟开发者社区特别强调个人能力极为不同，很多运维的问题解决不了，有时候很大程度上是受限于企业体制、组织架构、文化等方面的非技术层面的因素，而不单纯是个人能力所能解决的。

所以，要解决企业的运维问题，有时是需要自上而下的推进，整个技术团队共同执行落地才可以。从运维的角度单方面发起，是不会有效果的。

未来（2019-Future）：智能运维和云计算

关于智能运维

智能运维或 AIOps，我之所以把它定位在未来阶段，主要是我认为目前能在这个领域有成果的，还是集中在大厂。对于绝大多数企业来说，特别是中小企业，时机仍然未到。

从 AI 的角度，AIOps 有三个方面的充要条件：机器学习算法、计算能力如 GPU、海量数据。

从上面三个条件看，也就不难理解，为什么 AIOps 做的比较超前的都是国内外的大厂，因为有技术实力、有足够的资源和数据，最关键的是有足够复杂和变态的业务场景以及运维场景，在倒逼着 Ops 往这个方向上走。

但是，对于一家企业来说，实施 AIOps 最重要的前提条件是数据，海量数据。目前来说，能够具备这个条件的只有大厂，因为只有大厂有这个业务和资源体量，能够产生海量数据。

同时，对于 AIOps 来说，还有很重要的一个前提条件，那就是高度完善的运维自动化，也就是 Ops 的部分。自动化都没做好前，AIOps 是没有任何意义的，千万不要本末倒置。

我的理解，AI 和 Ops 要解决的还是两个层面的问题。可以类比到人，AI 相当于人的大脑，我们手脚和躯干是执行系统，大脑负责决策判断，手脚躯干负责完成大脑下发的动作指令。对应到运维上面，AI 要解决的是怎么快速发现问题和判断根因，而问题一旦找到，就需要靠我们高度完善的自动化体系去执行对应的运维操作，比如容量不够就扩容、流量过大就应该触发限流和降级等等。

最后，AIOps 的发展一定是一个长期演进的过程。AI 是 Ops 的有力补充，进一步降低运维的工作强度和压力，但是 AIOps 一定建设在高度自动化和完善的运维体系之上的，是一个演进的过程，不会是一个跳跃性的过程，也不会产生一个完全颠覆性的 AIOps 模式，将现有的 Ops 体系替代掉。

云计算：未来已来，势不可挡

其实仔细关注下技术趋势的发展，我们会发现，现在很火的一些概念，比如 Serverless、FaaS、边缘计算、弹性计算、云原生、IoT 等，甚至是我们耳熟能详的 Docker 容器、K8S、机器学习、AI 等等，基本都跟云计算相关。很多都是在云计算这个趋势下衍生出来的新技术，而且又因为云计算提供的基础设施，相互之间又有紧密的联系。

说地严格一点，这些技术只有在云上，甚至是公有云上才会发挥作用和价值。脱离了云计算，这些技术没有任何意义。因为，云计算带来的最大的好处就是“按需索取”，也就是我们说的弹性，进而带来成本上的最优化。如果我们自己机

房里还维护着上千台设备，都是我们自己的成本，说实话，再弹性也没多大意义，因为不解决实际的成本问题。

再就是，到了机器学习领域，特点是周期性地需要大量 CPU 和 GPU 资源，并不是持续需要，所以如果还是延续老的思路自己采购，这个成本就大了去了，对于一般企业根本不现实。况且有时候还要考虑资源在不同区域分布的问题，比如边缘计算，一个普通企业搞一个机房还可以，但是要管理和维护很多机房，就不太现实了。

所以不难理解，未来的技术趋势，一定是跟云计算相关的。这个是大势，不可逆。

因此从个人成长的角度，我觉得，如果想要更好的发展、更大的空间，就朝着云计算这个行业走，做跟这个行业相关的岗位。一些岗位参考，比如，公有云平台的运维，至少在规模和体量上足够大，挑战也足够大，还能接触到很多新技术。其他由云计算衍生出来的解决方案架构师、技术运营、CRE 这样的岗位也都是不错的选择。

对于企业来说，尽快拥抱云计算，将更多的精力放到自己的核心业务能力上，通过云的能力，为自身的业务带来更多可能性，或许是一个更好的选择。

写在最后

这些年经历下来，特别是近几年，最大的感受就是变化之快，让人目不暇接。新事物、新技术、新产品、新平台层出不穷，有时不知从何下手。

面对这样充满了不确定性的场景，运维人员，包括其他技术人员，唯一能做的就是坚持学习，拥抱变化，脚踏实地的解决问题。对于个人要不断提升能力，对于企业要审时度势，选择好未来的技术方向，我想未来一定会更有挑战，更有乐趣，也必将更加精彩。

作者简介

赵成，资深 DevOps 和运维专家，现任蘑菇街平台技术总监，腾讯云 TVP，极客时间运维专栏作家，多届 ArchSummit 运维专题明星讲师和优秀出品人，SRECon19 Asia/Pacific Speaker，个人专注于云计算、SRE 和 AIOps 领域。

云计算十年回顾（上）：风雨兼程

作者 何恺铎

本文旨在通过回顾技术发展总结最佳实践、为开发者启发技术新思路。本篇为 InfoQ 特邀北京国双科技有限公司（以下简称：国双）技术总经理何恺铎撰文，对云计算发展历程进行深入分析和探讨。

写在前面

科学技术的革新始终在推动时代巨轮轰鸣向前。云计算，已经走过十余年的风雨历程，从 AWS 初创立时的牛刀小试，到如今成长为一个巨大的行业和生态，堪称是新世纪以来最伟大的技术进步之一。“云计算”这个术语，也早已从一个新鲜词汇，成为了妇孺皆知的流行语。十年荏苒，风云变幻，值此 InfoQ 中国筹划发表十周年系列回顾文章之际，我们正可忆昔抚今，回顾和感受云计算领域的发展与变革。

任何事物的诞生和发展一定有其前提条件和土壤，云计算亦是如此。记得在世纪初的大学课堂上，教授们颇为推崇网格计算理论，该理论事实上已经充分体现了计算资源分布式协作和统一管理的先进思想。可惜网格计算过于学术化，最终是更接地气也更宏大的云计算横空出世，震动了整个 IT 业界。

那么，云计算诞生及蓬勃发展的原因是什么呢？在笔者看来，主要有三大因素，分别是相关**软硬件技术的成熟**、巨大的社会价值和伟大的商业模式。所谓软硬件技术的成熟，指的是在技术和工程层面，构建云计算平台的条件开始陆续具备，主要包括超大规模数据中心建设、高速互联网络，以及计算资源虚拟化（Hypervisor）和软件定义网络（SDN）技术的不断发展和成熟——

这些基础能力构成了云计算发展的技术前提；所谓**巨大的社会价值**，指的是从用户角度出发，云计算的采用使任意组织和个人得以站在巨人的肩膀上开展业务，避免重复造轮，极大提高了软件与服务构建各环节效率，加速了各类应用的架构和落地，而云端按需启用和随意扩展的资源弹性，也能够为企业节省巨大成本；所谓**伟大的商业模式**，指的是云计算的产品和服务形态非常适合新时代的 B 端需要，订阅制和 Pay-as-you-go 的计费方式大幅降低了客户的进入门槛，而技术基础设施架构方面的稳定性需要又带来了较高的客户粘性，再加上多租户高密度数据中心所能带来的规模效应，这些因素使得云计算能够成为一门好的生意，对应着一个极佳的 B 端商业模式。这三者缺一不可，共同促成了云计算的兴起与繁荣，也吸引了不计其数的业界精英投入其中，是为云计算取之不竭的源动力。

当然，同任何新生事物一样，云计算行业的发展也并非一帆风顺。从早期被指责为“新瓶装旧酒”的概念炒作，到对云上数据隐私问题的担忧，再到对各类公有云线上偶发事故的讥讽和嘲笑，云计算的成长亦伴随着各种挑战和质疑。其中部分负面反馈实质上还是由于使用不当或偏离最佳实践造成，也让云计算背负了不少“冤屈”和骂名。所幸瑕不掩瑜，云计算的先进性终究让发展的主旋律盖过了干扰与杂音，配合其本身持续的改进，越来越多地得到客户的认可，市场规模也不断扩大。

本文为该系列的第一篇文章，会试图从普通开发者及实践者的视角来回顾云计算发展的上半程：萌芽时代和探索时代。因篇幅所限，所讨论的范畴将聚焦于公有云，以 IaaS 和 PaaS 层面的技术演进为主。

萌芽时代 2008-2011

事实上，云计算行业的开端较难精准定义。一般认为，亚马逊 AWS 在 2006 年公开发布 S3 存储服务、SQS 消息队列及 EC2 虚拟机服务，正式宣告了现代云计算的到来。而如果从行业视角来看，我们也不妨视 2008 年为另一个意义上的云计算元年。因为在这一年，当 AWS 证明了云是可行业务之后，越来越多的行业巨头和玩家注意到这块市场并开始入局：微软在 PDC2008 上宣布 Windows Azure 的技术社区预览版，正式开始微软众多技术与服务托管化和线上化的尝试；Google 恰好也在 2008 年推出了 Google App Engine 预览版本，通过专有 Web 框架允许开发者开发 Web 应用并部署在 Google 的基础设施之上，这是一种更偏向 PaaS 层面的云计算进入方式；而众所周知，国内的云计算标杆阿里云也是从 2008 年开始筹办和起步——可见是从 2008 年起，云计算的时代大幕逐步拉开，开始形成一个真正的多元化市场，并随着众多巨头的加入开始良性竞争。

在云计算兴起之前，对于大多数企业而言，硬件的自行采购和 IDC 机房租用是主流的 IT 基础设施构建方式。除了服务器本身，机柜、带宽、交换机、网络配置、软件安装、虚拟化等底层诸多事项总体上需要相当专业的人士来负责，作调整时的反应周期也比较长——相信许多研发负责人都有过等待服务器到位的经历。云的到来，突然给出了另一种高效许多的方式：只需轻点指尖或通过脚本即可让需求方自助搭建应用所需的软硬件环境，并且根据业务变化可随时按需扩展和按量计费，再加上云上许多开箱即用的组件级服务，这对许多企业来说有着莫大的吸引力。Netflix 就是早期云计算的拥抱者和受益者，该公司在 2010 年成功地全面迁移到 AWS，堪称是云计算史上最著名的案例之一。

技术产品上看，早期的云上产品组合虽然还比较单薄，也存在一些限制，但**计算和存储分离**的核心理念已经得到初步确立，并深刻影响了基于云上应用程序的架构模式。具体来说，该理念一方面体现在云厂商纷纷将存储服务开辟为独立的产品类别，通过如 AWS S3、Azure Storage、阿里云 OSS 等服务清晰剥离了二进制对象与文件的负载与管理，并且提供了丰富的接口和 API 以供应用程序进行集成；另一方面，在虚拟机层面基于网络存储的托管磁盘服务也得到了大力发展和推广，如 AWS 的 EBS、Azure 的 Page Blob（后封装为 Managed Disk）以及阿里云的块存储（云盘），此类托管磁盘既很好地保障了数据可靠性，

又提供了丰富的容量和性能级别选择，使得云上虚拟机的计算和存储充分解耦，在这两方面都能够独立扩展和调节。

在云计算的萌芽时期，另一个有趣现象是当时“云计算”和“大数据”纠缠不清的关系。可能是由于发展历程上几乎同期兴起，以及在大规模数据存储与计算上的确存在能力交集，两者的概念和定义一度容易互相混淆。早年笔者曾购买过一本很不错的 Hadoop 技术书籍，其副标题却是“开启通向云计算的捷径”，可见早期云计算的定义曾有较为模糊的阶段。当然，随着后续的时代发展，这样的歧义越来越少见了，云计算已多特指提供各类云端服务与组件的软硬一体化技术资源平台，是一个带有明确商业模式的综合性载体，而大数据则是技术上处理大体量数据的方法论和实现，主要是一种技术体系——所以两者各自独立又可互相依存，比如各云计算厂商都陆续推出了云上大数据分析服务，如 AWS 的 EMR、Azure 的 HDInsight、阿里云的 E-MapReduce，本质上正是开源大数据技术在云上的实现和适配。

探索时代 2011-2014

当云计算玩家们纷纷入场并确认大举投入的战略后，行业进入了精彩的探索时代。这一时期的各朵云在产品技术层面进行了许多有益尝试，虽然免不了在个别方向上走些弯路乃至经受挫折，但总体而言云端服务的能力与质量取得了相当大的进步和提升，也为云计算赢得了越来越多的关注和喝彩。

首先，IaaS 层面继续围绕虚拟机为核心得到稳扎稳打的推进和增强。更强更新的 CPU 带来了云上虚拟机计算能力的提升和换代自不必说，早期机型内存相对偏小的问题也随着新机型的推出逐步得到解决，新上云端的 SSD 磁盘更是让机器性能如虎添翼。厂商们不约而同地形成了通用型、计算型、内存型等多个虚拟机系列，通过将不同 CPU/ 内存比例搭配的机型摆上货架，给予不同应用程序负载以更多选择。当笔者浏览这些琳琅满目的机型列表和参数时，恍惚间犹如来到了当年热闹的中关村 IT 卖场，颇有在云端“攒机”的奇妙感觉。

同属基础设施的存储类服务在初期得到了市场欢迎和认可之后，也同样迎来了大发展。原有功能得以细化，通过引入冷、热乃至存档的各级分层，进一步凸显成本优势。为了弥补托管磁盘在跨机器文件共享方面的不足，类似 NFS 的文件存储类服务也逐步成为了云上标配，如 AWS 的 EFS、Azure 的 Azure Files 及阿里云 NAS 等，进一步把计算存储分离架构发挥到极致，大大方便了某些场景下的架构与实现。

与 IaaS 的高歌猛进相比，PaaS 的发展在这一时期则显得多少有些落寞，尤其是从希望从顶层应用框架入手来推广云的方式一直不温不火，没有得到太多的关注。例如前面我们提到的 Google App Engine，经过几年发展后技术上其实颇为扎实，除 Python 外也添加了 Java/Go 等更多语言的支持，亦可与 Google 其他基础设施无缝集成，但就市场而言总是处于叫好不叫座的状态；国内率先实践此模式的厂商是新浪 SAE，上线之初颇受关注，亮点在于支持 PHP 语言及集成 MySQL，可惜多年运营亦未达到火爆状态；阿里云也曾推出相似的 ACE 服务对此类产品形态进行尝试，后也于 2016 年下线——究其原因，恐怕还是因为 App Engine 类服务本质上是相对受限的环境，平台的技术约束和绑定较强，出问题时也较难进行深入排查所致。这揭示了**云的用户固然喜欢技术实现上的便利和平台抽象，但也同样渴求技术灵活性、可移植性和自我掌控**。在后来的 Web 类 PaaS 服务中，我们看到业界调整了策略：更多地定位为通用运行平台并着力于自动扩容和负载监控等专业增值服务，尽可能解除技术、语言和框架层面的限制，如 AWS 的 Elastic Beanstalk、Azure 的 App Service 等；另一种发展思路则彻底轻量化，主要面向中小客户推广较为经济的建站服务，以满足入门级托管需求。

所幸，PaaS 中的另一重要分支——泛数据库类服务（亦称 DBaaS）则得到了快速的发展。由于数据库服务较为标准化，又是应用程序中不可或缺的重要组成，因此很快得到了广泛支持和采用。这里的典型代表有 Amazon RDS、阿里云 RDS、Azure SQL Databases 和 Azure Database for

MySQL/PostgreSQL 等。

与自行使用虚拟机搭建相比，云上数据库一键式的创建过程，自带的高可用性和自动备份，可谓省心省力；丰富的性能等级选项更是可根据实际负载选择和调节，实现了成本的最优化控制。除此之外，由于云上数据库按时间及容量计费的轻量特点，也给予了一些优质企业级数据库以重要的展示和售卖渠道，降低了企业的评估和尝试成本：例如，原本只能以昂贵的一体机方式售卖的 MPP 数据库 SQL Server Parallel Data Warehouse (PDW)，微软将其彻底云化之后以 Azure SQL Data Warehouse 的产品形态发布，改变了其养在深闺人未识的状态，使得相关技术让普通开发人员触手可及，也无疑大大增加了在整体数据架构中的采用可能。

让我们回到宏观视角。在这一百家争鸣的探索时期，令人可喜的是中国云计算真正如火如荼地发展了起来。除了早期入场的阿里云和盛大云，腾讯、百度及三大运营商等各路巨头也都先后布局试水，并纷纷把“云”的品牌从一度红火的个人网盘服务让位于企业级云计算；微软 Azure 也于 2014 年在中国正式商用，标志着外资厂商开始参与国内市场竞争。值得一提的是，这段时期独立云计算企业 UCloud、七牛云、青云等都相继创立，分别以极具特色的产品服务和强大的自主研发能力，为中国云计算发展书写了浓墨重彩的篇章，使得国内云计算市场更加精彩纷呈。

云计算十年回顾（下）：势不可挡

作者 何恺铎

云计算从 AWS 初创时的牛刀小试到如今巨大的行业和生态，从新鲜词汇变为流行语，它的十年发展经历了哪些阶段？每个阶段分别创造和发展了什么？未来云计算又将朝着哪些方向继续前行？

写在前面

本文上篇回顾了云计算领域早期的发展与变革，从普通开发者及实践者视角梳理了十年来云计算发展的上半程：萌芽时代和探索时代。本文下篇是对该选题的延续，着重介绍云计算发展的下半程：发展时代和繁荣时代，并将探讨未来云计算领域的若干发展趋势。因篇幅所限，本文所讨论的范畴同样聚焦于公有云，以 IaaS 和 PaaS 层面的技术演进为主。

发展时代 2014-2018

当整个云计算行业一定程度走过蹒跚探索时期之后，开创者们积累了越来越多的经验，对市场反馈和客户需求有了更清晰的了解与洞察，业务模式与商业运营也驾轻就熟起来——云计算行业终于进入高速发展时代。在这一时期，不论是总体市场规模，还是云计算的产品与服务，都得到了极大的增长和丰富。

首先，IaaS 方面的继续进步体现在服务的特异性和多样性上，**不断推出细分领域和特定场景下适用的虚拟机实例**：如基于物理隔离的专属实例、可运行 SAP HANA 等大型负载的超高配实例、采用 CPU 积分制的性能突增实例、适用于机器学习与 AI 的 GPU 实例等。此类细分服务在云端出现的背后，是不少厂商针对性地研发和定

制了适用于云的专属配套硬件。这些新一代的服务器不仅是简单的配置升级，而且在设计之初就是为云端负载而生，与云计算产品理念及底层软件技术栈高度融合匹配。另外在虚拟机计费规则方面，除了经典的按使用时长计费方式，各厂商也相继推出更为灵活的计费模式：包年包月、预留实例、竞价实例等，可有效帮助降低使用成本。这些 IaaS 方面的诸多进展，进一步满足了各场景的细分需求，减少了许多客户的上云阻碍。

在存储服务方面，云计算的步伐也在加快，除立足于核心对象存储服务的增强外，开始以一体化方式进攻部分垂直市场，挤占传统厂商的市场空间。最典型的例子莫过于 CDN：阿里云自 2015 年起在 CDN 领域大举扩张，多次主动大幅降价抢占市场，其他云计算厂商也纷纷跟进，这是原本波澜不惊的 CDN 领域的标志性事件。面对云计算厂商的竞争，网宿、蓝汛等老牌 CDN 厂商顿时立感压力，市场份额也开始发生相应变化。在国外，AWS CloudFront 和 Azure CDN 等云服务同样与 Akamai 这样的传统巨头进行着激烈角逐。当然，得益于 CDN 的架构融合特性，传统厂商也可选择与云厂商合作，从上游的基础设施方面作为云的供应商存在，不失为合作共赢之道。但无论如何，事实说明云的参与和挤压是相关市场发展的必然趋势。而且从技术层面来看，服务之间的**一体化协同效应**是云计算厂商进入 CDN 这样的单一领域的独到优势。例如，云厂商的 CDN 可以与自家的对象存储服务联动，用户只需要轻点鼠标，即可将对象存储中的文件映射至云 CDN 的边缘节点网络来对外服务，免去了搭建传统回源站点的麻烦。

PaaS 方面在这个大发展时代也找到了崛起之道，**不再寻求大一统的应用程序框架，而是更多提供标准的可复用中间件，并与其他 IaaS/PaaS 设施进行组合与联动**——这一思路迅速得到开发者和架构师们的欢迎，随着特性完善其采用率开始稳步上升。典型的例子包括 API 网关、负载均衡器、消息队列等。更进一步地，这一阶段的 PaaS 服务在与 IaaS 虚拟网络集成方面有了较大进步——这是以往 PaaS 服务常为人诟病的短板，即只提供面向互联网的公开端口，游离于 IaaS 的虚拟网络和架构体系之外——这一现状也通过两种方式得到了相当程度的解决：一种可称之为 PaaS-in-VNet，即允许 PaaS 服务直接部署到现成的虚拟网络之上，例如 Azure 上的 App Service Environment (ASE)，就是将 App Service 部署到私有虚拟网络的服务形式；另一种方式则是所谓 PaaS-to-VNet，为 PaaS 与虚拟网络之间提供私有通道，在不开放公开访问的情况下允许双向网络互通，同样以 Azure App Service 为例，用户可通过开启服务内置的 VNet Integration 特性来配置与私有网络的安全通信。可以看到，无论是 PaaS-in-VNet 还是 PaaS-to-VNet，其本质都是试图解决 PaaS 和 IaaS 的融合问题，这无疑增加了 PaaS 在整体架构中被采纳的可能。

势头颇佳的 DBaaS 方面的进展，主要体现为**从经典关系型数据库拓展到新兴的各类 NoSQL 数据库及大数据领域的云服务**。MongoDB、Redis、Kafka、ElasticSearch——这些耳熟能详的开源数据库，我们几乎能在每一个云上找到其对应的托管服务，轻松地一键搭建所需集群。云上数据库的开箱即用是如此的便捷与诱人，不断获得市场，甚至引起了开源厂商的不满：MongoDB 近期将开源协议从 AGPL 调整为新推出的 SSPL，是一时关注度颇高的社区新闻，其矛头正是直指开源软件被直接包装为云服务牟利的状况（部分网友戏称此为“插管吸血”）。在此，我们对各方抱有商业目的的行为不做倾向性评价，但从产品技术层面来讲，实力雄厚的云厂商并不乏应对之道：一是基于开源版本作分叉并开始自行维护和迭代，二是完全自行实现数据内核和引擎，仅在客户端协议方面与主流数据库进行兼容。目前，后者这类自研云数据库越来越成为一种新趋势：厂商可以放开手脚，充分利用云的特点进行重新设计，同时又兼容流行协议，这一方式迅速取得了市场和开发者认同，此类数据库的杰出代表是 AWS Aurora 和 Azure CosmosDB。其中 AWS Aurora 完全兼容最流行的 MySQL，同时实现了计算存储的高度分离和近乎无限的扩展，而 Azure CosmosDB 则是一款多模式数据库服务，提供 SQL、MongoDB、Cassandra、Gremlin 等多种开放协议或查询语言的兼容，同时实现了全球

分布、按需扩容、一致性保障等特性。所以，无论是 Aurora 还是 CosmosDB，一经推出都迅速攻城略地，取得了不俗的战果。在国内，以阿里、腾讯为代表的大厂商也同样在自研数据库方面不断加码，陆续推出了阿里云 PolarDB、腾讯云 CynosDB 等重磅服务。

在这样的局面下，也许开源软件厂商可以参考同为创业公司的 DataBricks 的做法。DataBricks 作为大数据处理领域事实标准 Spark 框架的实际掌舵者，一方面主导把控 Spark 开源版本不断进步，另一方面也开始推出性能更佳、交付节奏更快的商业版本 Databricks Runtime。更重要的是，Databricks 积极地同云厂商合作，将自身的解决方案融入到公有云平台之中，成为了平台上原生的 PaaS 服务，例如与微软深度合作推出的 Azure Databricks。尽管云上已有基于纯开源方案的大数据服务如 AWS EMR 和 Azure HDInsight，但基于 Databricks 商业级解决方案的 PaaS 服务有自己独到的优势和特点，已初步获得了不错的发展势头。我们由衷希望，出色的开源软件公司能够像 DataBricks 这样寻找到一种与云合作共赢的商业模式，毕竟业界既需要一站式的整合平台，又应当保护开源与创新的良性环境。

容器与微服务，可以说是近年最重要的技术趋势之一。作为新技术的拥抱者和试验田，公有云自然不会袖手旁观。事实上各大云在容器服务方面的尝试由来已久，在早期厂商们就各自推出了如 AWS ECS、Azure ACS、阿里云容器服务等基础设施，提供基于底层 IaaS 的容器运行环境，同时包含开源或自研的编排引擎。而当 Kubernetes 在编排大战中逐步胜出并成为事实标准后，各大厂商又不约而同地对 Kubernetes 提供更加定向和深度的支持，纷纷推出 AWS Elastic Kubernetes Service (EKS)、Azure Kubernetes Service (AKS) 等新一代容器服务。另一类容器托管服务则进一步屏蔽了底层细节和调度麻烦，让容器作为独立计算单元直接在共享基础设施上运行，如 AWS Fargate、Azure Container Instance、阿里云 ECI 等，颇得无服务器计算思想之精髓。当然，经典的无服务器计算一般指 FaaS，例如也在同步发展的 AWS Lambda 和 Azure Functions，它们虽在编程框架和范式方面有所限制，但得益于更高层抽象，可让开发者聚焦业务逻辑，在合适的场景中使用得当可大幅提高研发效率。

让我们的视角再从技术回到商业。随着云计算行业体量越来越巨大，市场竞争也愈发激烈，价格战屡见不鲜。虽然说云计算有着相当不错的商业模式，但这毕竟是一个重投入长周期的行业，因此陆续有中小玩家力不从心、陷入困境。例如国外的 Rackspace，原本入场颇早也有相当积累，一度曾在市场中占先，但当巨头纷纷入场后就显得后劲不足，现已跌出市场占有率前五名。也许当云计算进入巨头角力时代后，中小玩家还是需要着力发展自己特色，深耕细分市场，或是寻求联姻以共享能力与资源。

客户方面，云计算在这一时期开始明显地从互联网企业向传统行业进行渗透。为了拿下更多传统行业客户，组织架构和流程的匹配也是必做的功课。走在前面的云厂商相应地完善了云上的多账号管理、组织架构映射、资源分组、细粒度权限管控等企业级功能。例如阿里云就在不断地更新升级其企业控制台，帮助用户更好地管理人员、资源、权限及互相之间的关系。

至此，经历了大发展的云计算已然成长为几乎承载一切、包容一切的巨大平台，是一艘提供企业信息化和数字化整体解决方案的航空母舰。云计算无疑已经全面走向成熟，成为了参与和推动 IT 业界向前发展的重要力量。

繁荣时代 2019-

时间终于进入 2019 年。基于过去十年发展的良好态势，我们没有理由不相信云计算将进入繁荣热潮。来自 Gartner 的分析报告显示，2019 年的全球公有云市场规模将超越 2 千亿美元，并将继续保持稳定增速。而国内由于起步相对较晚，市场渗透率还不高，将拥有更高的增速。“上云”将成为各类企业加快数字化转型、鼓励技术创新和促进业务增长的第一选择甚至前提条件。对于企业而言，更多的不会是上不上云的问题，而是要考虑上哪家云、怎么上云的问题，是如何迁移重构以适配云端的问题，是如何让云更好地服务生产的问题。因此，我们对云的未来理应充满信

心，同时也抱有更高的期待。接下来，我们不妨结合企业的需求和云厂商的投入方向，大胆预测未来云计算发展的若干趋势。

趋势之一：**云计算将进一步成为创新技术和最佳工程实践的重要载体和试验场**，走在时代进步的前沿。这是得益于云产品本身的 SaaS 属性，非常适合快速交付与迭代，能够较快地把新产品、新技术推向业界。可以看到，当下的热点技术，从 AI 与机器学习、IoT 与边缘计算、区块链到工程实践领域的 DevOps、云原生和 Service Mesh，甚至未来感十足的量子计算，都有云计算厂商积极参与、投入和推广的身影。以人工智能为例，不论是前面提到的 IaaS 中 GPU 计算资源的提供，还是面向特定领域成熟模型能力开放（如各类自然语言处理、图像识别、语音合成的 API），再到帮助打造定制化 AI 模型的机器学习平台（如 AWS SageMaker、Azure Machine Learning Service、阿里云 PAI 等），云事实上从各个层面都有力地支持和参与了 AI 相关技术的发展。就最终效果而言，云上的资源和产品让人工智能等新兴技术变得触手可及，大大降低了客户的探索成本，也加快了新技术的验证和实际交付，具有极高的社会价值。另外值得一提的是，云在新技术的发展过程中还保持了某种程度的中立性，对于技术趋势持有普遍包容和适应的态度——最典型的例子莫过于容器化和开源框架（如 Spring Cloud）支持下的云原生架构，它们事实上同部分云端 PaaS 服务存在竞争关系，甚至有助于用户解除厂商锁定，但云厂商并不会厚此薄彼，而是进行不遗余力的支持与适配，更多地把选择权留给客户。

趋势之二：**云计算将顺应产业互联网大潮，下沉行业场景，向垂直化产业化纵深发展**。随着通用类架构与功能的不断完善和对行业客户的不断深耕，云计算自然地渗透进入更多垂直领域，提供更贴近行业业务与典型场景的基础能力。典型的垂直云代表有视频云、金融云、游戏云、政务云、工业云等。以视频云为例，它是将视频采集、存储、编码转换、推流、视频识别等一系列以视频为核心的技术能力整合为一站式垂直云服务，不仅适用于消费互联网视频类应用的构建，更重要的是配合摄像头硬件和边缘计算节点进军广阔的线下安防监控市场。再如金融云，可针对金融保险机构特殊的合规和安全需要，提供物理隔离的基础设施，还可提供支付、结算、风控、审计等业务组件。可以预计，随着消费互联网红利耗尽，产业互联网将逐步受到重视并兴起，其规模之大、场景之多，将给予云计算厂商极大的发展空间；而云计算作为赋能业务的技术平台和引擎，也非常适合承载产业互联网的愿景，加快其落地与实现。

趋势之三：**多云与混合云将成为大中企业刚需，得到更多重视与发展**。当企业大量的工作负载部署在云端、对于云的应用进入深水区之后，新的问题则会显现：虽然云端已经能提供相当高的可用性，但为了避免单一供应商出现故障时的风险，关键应用仍须架设必要的技术冗余；另一方面，当业务规模较大时，从商业策略上说也需要避免过于紧密的厂商绑定，以寻求某种层面的商业制衡和主动权。因此，越来越多的企业会考虑同时采购多个云厂商的服务并将它们结合起来使用——这将催生多云架构和解决方案的兴起，以帮助企业集中管理协调多个异构环境，实现跨云容灾和统一监控运维等需要。例如华为云不久前发布了商用级的多云容器平台 MCP，可对跨云跨区域的多个容器集群进行统一资源与应用管理，提供一站式的接入、管控和调度能力；在网络基础设施层面，也有如犀思云这样专注于云交换服务的企业，提供云与云、网与网之间的快速互联，帮助多云互联在稳定性延迟等方面达到生产要求。除同时使用多个公有云之外，合规和隔离性要求更高时的另一选择是私有部署云基础设施，并与相应公有云专线连接形成混合云架构。从目前市场态势看，主要有公有云厂商主导的混合云方案和私有云厂商主导的方案两类。笔者个人更看好前者的发展，是因为公有云厂商方案让混合云的私有部分成为了公有云在自有数据中心的自然延伸，提供了与公有云端高度一致的能力和使用体验。此类服务的代表有微软的 Azure Stack，以及阿里云 Apsara Stack，包括之前只专注公有云的 AWS 终于在 re:Invent 2018 大会上推出了 AWS Outposts，也加入了混合架构的行列。

趋势之四：**云的生态建设重要性不断凸显，成为影响云间竞争的关键因素**。当某个云发展到了一定规模和阶段后，恐怕不能仅仅考虑技术和产品，同样重要的是建立和培育具有生命力的繁荣生态和社区，此为长久发展之道。因为一朵云再大再丰富，也必有覆盖不了的场景和完成不了的事情。这就需要大量的第三方服务提供商，以合作伙伴的身份基于云平台提供各类解决方案。此举既方便了用户，又增加了云的粘性，也可保证应用提供商的市场空间，可谓三方共赢。所以在当下各大云平台上，我们都能够找到应用市场和合作伙伴计划，这正是厂商们着力建设的第三方解决方案平台。例如，国内大数据领域的明星创业公司 Kyligence 拥有以 Apache Kylin 为核心的企业级大数据 OLAP 解决方案，通过其 Kyligence Cloud 套件深度适配了多个云端，先后登陆了包括 Azure、AWS 和阿里云在内的多个云市场与平台。

云生态的另一个重要方面是面向广大开发者、架构师和运维工程师的持续输出、培养和影响。**只有赢得广大技术人员的关注和喜爱，才能赢得未来的云计算之仗**。我们之所以敢下这个判断，是因为以下几点原因：其一，云的采购具有弹性特征，不论是 Pay-as-you-go 还是年单方式，都可随实际项目效果和生产运行情况进行调整，此时一线研发人员和架构师会颇具发言权，可将使用端的实际情况反推至商务决策层面并影响续约；其二，从历史上来看，部分较为失败的云上功能，往往是失之于理想化和简单化，或是过多地从管理或宣传视角考虑而忽略了落地细节，最终导致了实操效果受限、口碑下滑甚至无人问津，因此从开发者的角度思考产品设计对于云而言至关重要；其三，如若能培养庞大的技术爱好者和粉丝群体，形成传播效应，相信对应的云服务自然不愁业务的增长，还能进一步收集到更多产品反馈，形成良性循环。由上种种，所以当下各大厂商，都开始空前重视开发者关系，并视之为核心竞争力。云厂商们不但努力地建设丰富的文档体系和在专业媒体频繁发声，还会积极举办各类论坛和参与业界开发者会议，并新增如 Developer Advocate 这样的布道师职位，专注于在开发者群体中扩大影响力。这里我们不妨简单分析一个例子：IBM Cloud(原 BlueMix)。总体上 IBM 云历来相当注重和依赖企业端的庞大销售体系和客户资源，但在赢得开发者和社区方面投入相对不足，所以在大多数人的印象中，IBM 云总显得有些遥远和陌生。一旦“脱离了人民群众”，久而久之就难免在市场竞争中处于颓势。恐怕这也能够解释为什么 IBM 去年斥 340 亿美元巨资收购红帽：IBM 不仅仅是看重 Red Hat 深厚的开源技术积累及其 OpenShift 云平台，也一定包含了对于其开发者人气和社区基因的考量，可以很好地弥补自身短板。

综上所述，“**创新、垂直、混合、生态**”这四大趋势，将伴随云计算走向繁荣。对于云计算的美好未来，我们已迫不及待。最后再作一个小小的预判：随着云的高度复杂化和差异化，企业会愈发需要面向云端各个层面的解读、判断与帮助，除了第一方厂商支持团队的助力之外，独立的云计算咨询与托管服务会成为新的需求热点——这将催生一个不小的云增值业务市场，即 Cloud MSP (Managed Service Provider)。在国外，Cloud MSP 已有一定的关注度，Gartner 也开始为此领域发布观察报告及绘制魔力象限，目前埃森哲处于行业领先地位；在国内，我们也欣喜地看到如云角信息（已被神州数码收购）、云宿科技、新钛云服等厂商中立的云 MSP 不断出现，且发展势头颇为良好。能否为客户持续创造价值，并形成合理的商业模式，将是决定 Cloud MSP 这一云衍生行业未来的关键。

结语

云计算走过了激荡十年，可谓势不可挡，风雨兼程。它如此巨大和丰富，虽万字不足以道其一二。限于篇幅，本系列文章无法覆盖云计算的更多角度与话题，还请读者见谅。另外，本系列文章对于云计算发展的分代方式仅供参考，事实上技术的发展更迭总是互相交织和持续进行的。

云计算历史性地对 IT 硬件资源与软件组件进行了标准化、抽象化和规模化，某种意义上颠覆和重构了 IT 业界的供应链，这是一个巨大的革新与进步。如今，云计算正迎来最好的时代，

在中国这片广阔热土更是如此。我们由衷希望，云计算行业不仅取得商业上的成功，更能扎实服务各行各业，为社会经济发展提供数字化引擎和强大动力。让我们继续与云计算同行，与伟大的数字时代同行。

作者简介

何恺铎，国双（Nasdaq：GSUM）技术总经理，QCon 讲师，公众号“云间拾遗”作者。毕业于清华大学，曾供职于摩根士丹利基础架构部门，2011 年加入国双工作至今。多年来参与架构和设计了国双数个面向数字营销和社交聆听的大数据解决方案。个人关注的技术领域包括云计算、Hadoop 生态系统、数据库技术等。

移动开发十周年：开源节流，创造新生态

作者 臧成威 王志宇

无论是小程序还是 Flutter，都为行业增加了更加体系化的新生态。

前言

现今的生活已经离不开手机了，早上利用手机看新闻，白天打开微信交流，中午用美团叫个外卖，夜深刷刷视频、或者手机吃鸡娱乐一局。生活中处处可见移动互联网给千家万户带来的便利。回首十多年来移动互联网的发展，我们既可以回顾历史，抓住那些耀眼的瞬间，也可以以史为鉴，为我们指明未来的道路。

技术对于商业来说是生产力的提升，商业的本质是逐利的。因此，技术也需要满足两点基本诉求，开源和节流。开源即是促使开发团队具备更快的迭代，也就是效率的部分。节流即是降低开发团队的开发成本并且降低风险，是效率和质量的结合。

相对于持续几百年工业革命，移动互联网的发展是短暂的。在这十几年的发展中，为了满足开源和节流的涌现出很多技术。接下来我们将会以开发方式的演进、基建与软件架构变化、移动新技术三个方面展开，带领大家回顾这十年。

开发方式的演进

1. iOS / Android，移动端的爆发

2007 年 1 月 9 日，史蒂夫·乔布斯在 MOScone Center 发布了第一款 iPhone。早期的 iPhone 并不支持个人开发者进行开发，直到 2008 年的 3 月，才支持了软件开发套件。同年 6 月，iPhone 发布了跨时代的 iPhone 产品 iPhone 3G，同时推出了 App Store，给移动开发市场带来了

一场史无前例的变革。

2007 年 1 月 9 日乔布斯发布第一款 iPhone

无独有偶，2007 年 11 月 5 日，Android 发布了 1.0 beta 版本。2008 年 9 月 23 日 Android 系统正式发布，同年 Google 发布了基于 Android 系统的首款手机。2011 年，Android 在全球市场份额首次超过塞班系统，成为了全世界使用设备最多的移动端操作系统。

2008 年 9 月发布的 T-Mobile G1

有趣的是，Android 最早只是为了数码相机而进行设计的，由于其用户需求量不够大，才开始向移动端设备方向进行发展。

Android 的发展，得益于开源带来的开放性。手机生产商可以根据自己的需要进行扩展和裁剪，这些使得 Android 的生态迅速壮大。这也是和同一时期的 iOS 和 Windows Phone 差异最大的一个特点。

诺基亚发布的 Lumia 手机

当时的时代，除了 iOS 和 Android，势头还可以的就属 Window Phone 了。微软于 2010 年 10 月 21 日正式发布 Windows Phone 7.0。前身是已经在手持设备耕耘多年的 Windows mobile 系统。当时主要的生产商诺基亚、三星、HTC 也都算行业巨头。可惜由于种种原因，最终成为移动热潮中的先烈。

2. 移动端开发的兴起

随着移动端用户的增长，移动端 App 受到越来越多的人的喜爱。在大屏幕上进行所见即所得的操作，享受比网页更好的交互体验。转眼到了 2012 年，从这一年开始移动 App 数量井喷了几年。移动端开发从业人数也逐渐增长，App 的包大小也迅速增加。包大小的增加意味着代码行数的增加，众多的代码也给移动端开发架构提出了新的挑战。

早期的移动端开发体验并不完美。在 iOS 中，技术人员往往受困在手动内存管理的困境中。那时候，崩溃率在 1% 以上是一件不那么可笑的事情。而 Android 系统也因为垃圾回收策略不完善，加之系统权限管理的不完善，导致卡顿的现象时有发生。但也是在这样一个时代，移动端用户数量开始攀升，移动端开发开始蓬勃发展。从开始的手动内存管理，到白胡子老头的 MVC 教程，移动端从业技术人员开始搭建现代化的移动端架构之路。

时至今日，原生系统的开发仍然占据着相当的地位，而更多更好的第三方工具和 IDE 使得原生的开发更为优异。

3. Hybrid：混合开发模式提升开发效率

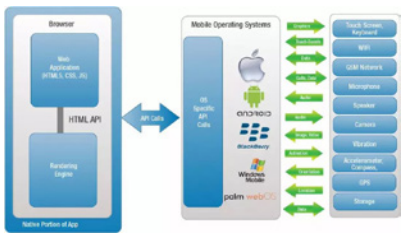
软件领域里面有一句经典的话，“Don't repeat yourself”。但商业从来都不是一个单人游戏，当时 iOS、Android 两大系统就已经横扫天下，我们不得不针对两个系统开发同样的功能或者是界面。然而由于 iOS 和 Android 的设计语言并不尽相同，某些公司甚至需要两套 UI 设计稿。当时还需要加上对 PC Web 端的支持，我们通常需要更多的人力去完成“同一件”事情。这样的重复带来的是就业，但也带来了麻烦。

对于企业来讲，效率是一个经久不衰的话题。如果两个人能做的事情能压缩到一个人完成，这样高效的开发给企业带来的是直接的经济利益。对于移动端开发来讲，如果能实现“一套代码，多端运行”，这势必会给企业带来人力资源上的节省。

2010 年左右，HTML5 如火如荼的冲入市场。顿时，所有的书、网络充斥着 HTML5 拯救世界的说法。也就是在此时，不少的公司直接跳过了 Native，全量拥抱 HTML5，这里面就包括我们熟知的 Facebook。

但好景不长，HTML5 并没有大家期待的那么万能。受限制与移动端性能和刚开始普及的 3G 网络情况，网页加载和渲染始终是一件十分耗费资源的事情。因此，Hybrid 技术应运而生。

早期的 Hybrid 技术几乎特指将 HTML5 嵌入到 App 中进行混合开发，这种开发模式的好处十分明显。对于产品经理，由于开发时间短，可以帮助其快速验证产品效果。对于研发工程师而言，前端同学也可以直接参与到了 App 的开发中来，扩宽了前端的技术栈。对于企业，前面讲到直接直接节省人力资源。其综合了网页开发上线周期快和部分页面性能考虑的特点，杜绝了之前一套 UI、三套实现的开发资源浪费。仅需要开发一套 HTML5 代码，就可以在多端运行。



Hybrid 架构图

提到 Hybrid，最重要的概念就是桥了。混合开发的关键是让 HTML5 能实现大部分的逻辑，但并不是所有的逻辑。首先，某些基础的业务逻辑，比如登录、地图已经在 Native 有了更好的实现。其次，HTML5 对硬件调用的支持并不是十分完善，定位、加速度计等硬件调用调用等还是需要 Native 层进行配合。Hybrid 的最终目的是让 App 同时具有快速迭代能力和优秀的用户体验。

桥主要解决的是两端的通信问题。实现桥的方案有很多种，常见的方案有两种。其一是向 JavaScript 运行时中注入 API，第二种是拦截 JavaScript 中的 Scheme。相对于函数注入的方式，URL Schema 的拦截实现起来可能比较简单，但，但上相对函数注入的方式会有一些损耗消耗。所以当时的实现里面很多都采用了运行时注入的方案。

Hybrid 是一种经典的跨平台实践模式，至今还被广泛的应用。现有的很多动态化框架与 Hybrid 技术都有不解之缘，其通信方式与 Hybrid 中的通信方式大同小异。

4. 跨平台技术兴起：Cordova (PhoneGap)、React Native、Weex

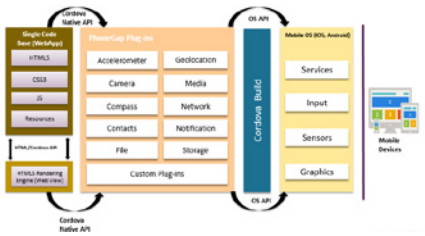
PhoneGap -> Cordova

PhoneGap 起步于 2009 年在旧金山召开的 iPhoneDevCamp 大会，Nitobi 的工程师 Brock Whitten、Rob Ellis 和 Andre Charland 在 iOS 系统内架设起 Web 接口和 Objective-C 之间的桥梁，让开发人员得以使用 HTML5、JavaScript、CSS 等 web 标准技术便捷开发原生程序，实现一次编译到处运行。其“桥接 Web 与 iPhone SDK 之间缝隙”的理念得到欢迎。

2011 年 10 月 4 日，Adobe 正式宣布收购 Nitobi 软件。PhoneGap 的代码贡献给了 Apache 软件基金会，但保留了 PhoneGap 的商标所有权，并命名为 Apache Callback。1.4 版发布后，Apache Callback 的名称变更为 Apache Cordova。Cordova 是街道的名字，在开发团队附近。

Cordova 是一个商业跨平台的应用架构。Hybrid 技术需要解决的重点是快速迭代问题，对于某些小的 App，人力资源成为了一个重要的指标。在 iOS 和 Android 发布的早期，纯 iOS 或者 Android 的开发者数量较少，在之前的发展中，Web 技术在 PC 端已经达到了一定的成熟度，开发者数量也是遥遥领先 iOS 和 Android 开发者的。因此，基于 Web 的跨平台技术成为了一个需求点。

Cordova 的重点是定义了一套用 HTML、CSS 以及 JavaScript 技术来编写应用程序的规则，”一次开发，多端使用“的目的。与 Hybrid 技术类似的是，Cordova 也实现了一套自己的桥，方便开发者可以调用起硬件来获取信息。其技术架构如下所示：



Cordova 架构图

Cordova 解决了当时的一个大难题：”一次开发，多端使用“。当时美团的很多商家端就是使用 Cordova 来开发的。从热更新的角度来讲，Cordova 有着良好的热更新能力。但是 Cordova 的动态更新体验并不够好，因为整个 App 都是使用 HTML5 编写的，更新可能需要暂停用户使用来进行，加上 HTML 对于触控事件的支持并没有原生体验好，性能也比原生差很多。所以，纯 HTML5 的 App 并没有在我们的视野中流行起来。

Cordova 同一时代的产品还有 2012 年的 IONIC，甚至现在还有 Platform 7 这样的方案可以应用于 Cordova。

React Native & Weex

从本质上来讲，React Native 和 Weex 可以算作 Hybrid 的方案。把它们和 Cordova 纯 HTML 开发的 App 进行对比的原因，是因为 React Native 和 Weex 都支持将资源文件（例如 HTML、CSS 或 JS）打包到 App 进行发布。

React.js 大会 2015 介绍 React Native

React Native 是由 Facebook 创建的开源移动应用程序框架。2012 年，马克·扎克伯格评论说：“我们作为一家公司犯下的最大错误就是过多地投入 HTML 而非 Native”。他承诺 Facebook 将很快提供更好的移动体验。

随后在 Facebook 内部，Jordan Walke 找到了一种从后台 JavaScript 线程为 iOS 生成 UI 元素的方法。他们决定组织一个内部黑客马拉松来完善这个原型，以便能够用这种技术构建原生应用程序。

经过几个月的开发，Facebook 在 2015 年发布了 React JavaScript Configuration 的第一个版本。

Facebook 在推出 React Native 的时候，一举高喊”Learn once, write anywhere“。React Native 的首要目的是让 React 框架在移动端开发中开花。其技术重点和难点在于使用 JavaScript 调用 Native 的组件，从而利用 JavaScript 实现原生的体验。

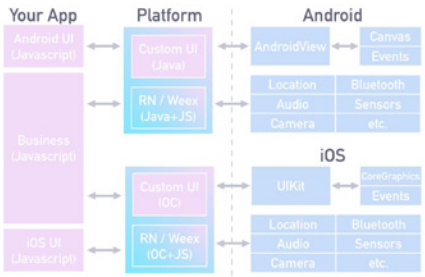
QCon 2016 南天宣布开放 Weex 内测邀请

Weex 的主要维护者是阿里巴巴，2016 年的 4 月 21 日 QCon 大会上阿里巴巴宣布开放了 Weex 的内测邀请。同年 6 月 30 日在 Github 上开源，并且在同年 11 月 30 日正式捐献给 Apache 基金会进行孵化，成为国内第一个捐献给 Apache 基金会的移动项目。

Weex 的口号是“Write Once, Run Everywhere”。和 React Native 类似，Weex 也是使用 Native 的 API 进行组件渲染。不过 Weex 更加重视让多端能够有一致的呈现效果。另外针对国内的使用习惯，对于 Weex 与原生应用的互相嵌套比 React Native 有比较明显的优势。

RN 和 Weex 的原理都是在 iOS 或者 Android 操作系统上利用 JavaScript 引擎进行 Bundle 的解

析，利用操作系统原生的 UI 框架进行渲染的。这种方案解决了 Hybrid 性能的不足，也提供了动态化和网页开发的体验。一时间引来很多大小公司的青睐。其架构形式如下：



RN / Weex 的混合栈架构示意图

RN 和 Weex 给行业带来的活力，从开源和节流的角度带来了生产力的提升。但是 RN 和 Weex 还并没有对一致性和构建生态给出完美答案，所以时代又促使新的技术产生。

5. 新时代跨平台技术：小程序 + Flutter

小程序

相对于上文提到的跨平台技术，小程序的技术功底看起来并没有那么深厚。从表象来看，小程序是 Hybrid 技术的加强版本。但是从产品的角度来解读，小程序是一个伟大的发明。

张小龙在 2017 微信公开课 Pro 上发布小程序

2017 年 1 月 9 日，微信创始人张小龙在 2017 微信公开课 Pro 上发布的小程序正式上线。同年 12 月 28 日，微信更新的 6.6.1 版本开放以小程序开发游戏，微信启动页面强制开屏小游戏“跳一跳”。2018 年 11 月，在第五届世界互联网大会“世界互联网领先科技成果发布活动”上，微信小程序入选“年度 15 项代表性领先科技成果”。

张小龙在小程序诞生之日就表示：小程序的核心理念是“用完即走”，后面又补充了“走了还会回来“，它是基于小程序现有状态的一个最佳总结。“用完即走”是指用户通过小程序，能够高效率地解决事情；而“走了还会回来”则是小程序良好用户体验的一个反馈与赠礼，二者相互影响，是决定一个小程序能否长远生存发展的两个重要因素。

小程序在生态的建设上非常出色，它定义了规则和能力，提供了集成开发环境，并且配套了详细的审核机制，让小程序和小游戏并行助力企业和自媒体以及个人发展。

小程序的成功还在于它形成了行业的标准，微信小程序后，相继推出了支付宝小程序、头条小程序。美团也根据自己的需要开发了美团小程序，目前还在积极建设丰富中。

Flutter

Google Flutter 团队高级工程师于潇老师在 GMTC 2018 宣布 Flutter Preview 1 发布

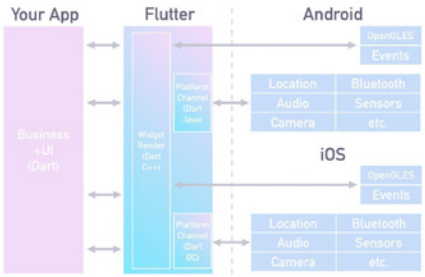
在 17 年的谷歌 I / O 大会上，Google 推出了 Flutter 这样一款新的用于创建移动应用的开源库。在 18 年 2 月 26 日世界移动大会上发布了 Flutter 的第一个 Beta 版本，5 月的 I / O 大会上更新到了 Beta 3 版本，向正式版又迈进了一步。值得一提的是 Google 对于中国市场非常的重视，2018 年 6 月 21 日北京 GTMC 大会上正式宣布了 Flutter Preview 1 这个预览版本。2018 年 12 月 4 日的 Flutter Live '18 大会上 Flutter 1.0 的正式版终于揭开面纱。

国内的头部开发商阿里巴巴、腾讯、美团、今日头条等公司都进行了不同程度的尝试。闲鱼的 Flutter 是目前为止使用最为广泛并且向社区回馈最多的一员。Flutter 之所以受到如此多的关注是因为它与之前的 Hybrid、React Native、Weex 有着很大的不同。

首先第一点，也是最本质的区别：Flutter 使用系统级 API 进行渲染，客户端仅需要提供一块画布就可以支持应用展示。这一点使其脱离了原生的渲染，让我们可以获得高度一致的渲染体现。除了一致性以外，Flutter 大量利用了 GPU 绘制的 Skia 图形引擎，在渲染速度和动画效果上都给出了更有体验。

其次是开发的效率和开发的生态环境，Flutter 从一开始就很关注开发的效率，利用 Hot-Reload 的亚秒级刷新能力开发人员可以非常快速的进行调试和开发，也能随时查看编写代码呈现的效果。另外 Flutter 采用 Dart 语言进行开发，Dart 语言兼有 JIT 和 AOT 两种运行模式，既可以保证开发模式下快速的部署和调试，也可以保证产品模式下高速的运行。Dart 语言易学易用，可以很方便的从 Java、Objective-C、JavaScript 等语言迁移过来。

Flutter 下层使用 C / C++ 编写的 Framework，上层用 Dart 进行视图和功能组件的搭建，对比 RN 和 Weex 架构分层示意如下：



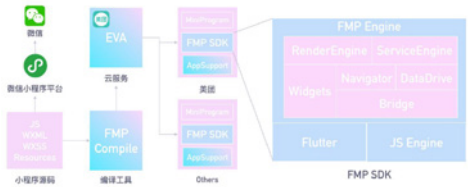
Flutter 架构示意图

无论是小程序还是 Flutter，都为行业增加了更加体系化的新生态。移动开发从趋势上谋求生态化的发展。生态化也是能面向更快的迭代发布以谋求商业的开源，更安全和高效的跨平台方案以谋求商业的节流。

渲染引擎 + 动态化

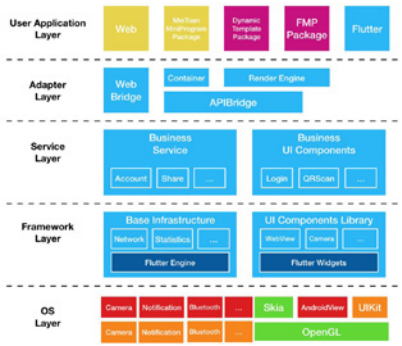
有了 Flutter 这样的跨平台渲染引擎可以最大化的完成体验一致性的挑战，也可以提升开发的效率。但是学习 Flutter 仍然是一个新的学习成本，迄今为止很多的方案都解决了“增量”的问题，但是都很难解决“存量”的问题。对于很多的大型公司而言，全面向 Flutter 转型是个谨慎而又困难的抉择。尤其是对于已经转向小程序或 React Native 的公司。

美团终端技术研发组的内部发起了一个代号为 FMP 的项目，目的在于解决“存量”问题。FMP 指 Flutter Mini Program，目的是用 Flutter 作为跨平台的渲染引擎，完整解析小程序包中 XML、CSS 和 JS 的一个计划。此举可以兼顾 Flutter 的高效与多平台一致，又可以支撑原有的小程序项目，让业务开发不需要多学习 Dart 语言与 Flutter 平台。FMP 目前的设计结构如下：



FMP 项目设计

这其中背后的逻辑就是用一个跨平台的渲染加处理引擎，对接目前的上层业务应用层，从而更快更一致的支撑原有应用。方式既可以用语法树分析和转译的方式，也可以直接利用引擎满足原来所有接口。其架构如下：



Flutter 动态化平台方案

更加复合的设计方案既是对技术追求卓越的表现，也是回馈商业上的开源和节流，我们相信在未来会有更多更好的生态诞生出来。

基础建设的提升

移动开发的发展，一方面是软件的提升，而另一方面则是基础建设的提升。软件的架构升级离不开这些基础建设的提升。这十年来，硬件的提升虽然已经不再满足摩尔定律，但是还是可以说是翻天覆地的。正是由于这些基础建设的提升，才有了移动开发的蓬勃发展。

存储的提升

2009 年 6 月 9 日诺基亚在美发售了 Nokia N97，这款搭载单核 434 MHz 的 ARM11 CPU、板载容量高达 32GB 的机皇当时的内存也不过 128M。同时间的 iPhone 3GS 其内存容量也不过 256M。iOS 系统的开发选用引用计数的语言 Objective-C，仍然对内存有着很良好的控制，当然手动引用计数也变成了当时 iPhone 开发人员在调试上和开发上最大的痛点。直到后期的自动引用计数才解决了这一问题。

后续的内存提升的很快，2010 年的 iPhone 4 可谓是乔布斯时代的一代经典，它就提升到 512M 的内存，当时的其他 Android 手机也都对齐了这一标准。再后来 2012 年的 iPhone 5 就已经提升到 1G 内存，同一时期的三星旗舰手机 Samsung Galaxy Note II 就已经有 2G 的内存。

时至今日大内存的手机已经司空见惯了，华为 P20 等旗舰机型已经支持 6G 的高速内存。内存的不断提升促使了客户端开发架构的演进，更多的缓存技术，虚拟 DOM 技术都得益于大内存带来的基础。没有内存的时代早已一去不复返了。

处理速度的提升

硬件提升很重要的一环就是 CPU 和 GPU 的提升，这些提升做为了移动种种架构的基础。2009 年的 CPU 都是单核心主频在 400-600 MHz 左右，如 iPhone 3GS 的 S5L8900 主频只有 412 MHz。

2011 年 8 月发布的小米 M1 手机号称世界上首款双核 1.5 GHz 手机，采用高通骁龙 S3 双核 MSM8260 处理器，主频速度比一般单核手机快近 2 倍。它开启了手机的双核甚至多核的时代，时至今日手机的核心已经多至 8 核。因此多线程与异步成为了移动开发的一个重点，这也是后期种种异步渲染引擎，异步响应式框架得以发展的基础。

展示精度的提升

除了内存和处理器，不得不说屏幕的提升。2009 年的 iPhone 3GS 和同期的 HTC G3 分辨率都为 320x480。所以早期的移动界面都非常的简洁，因为屏幕的大小和分辨率不足以显示更多的内容。2010 年 iPhone 4 带来划时代的 retina 屏幕概念，分辨率提升为 640x960。移动开发也随着屏幕的像素和大小变化而变化。

有趣的是，早期的 iOS 开发相比 android 开发有一个得天独厚的优势就是像素的一致。iOS 无论是 320x480 的 iPhone 3G/3GS 还是 640x960 的 iPhone4/4S/SE，开发人员都一律按照 320x480 的逻辑像素点来开发。而 Android 的屏幕多样性成了 Android 开发同学的痛点，mdpi/hdpi/xhdpi 等概念让开发苦不堪言。而后期 iPhone 5 的 640x1136 和 iPhone 6 的 750x1334 / 1080x1920 让 iOS 开发人员也尝到了屏幕适配的滋味。此时种种的屏幕适配框架就日益增多了起来。

除了像素的提升，屏幕也越做越大了。iPhone 3GS 只有 3.5 英寸，现在的 iPhone Xs Max 增大到 6.5 英寸。如此大的显示空间给了产品经理更多的想象空间，也让开发变得日益复杂。这些硬件的提升促使了框架的迭代。

网络的提升

近十年来移动互联网的发展离不开基础网络的发展。2009 年成为移动互联网高速发展的起点，与 3G 网络的兴起有着极大的关联。

与 2G Edge 最大 473.6Kbps 的下行速度相比，HSPA+ 制式的 3G 网络达到了 63Mbps 的极限速度。这使得移动互联网从“ 小水管 ”迅速通向了“ 高速公路 ”，如同拨号网络提升到宽带一样，移动互联网也是从这里高速的发展起来。

2013 年的 12 月 4 日，中国移动获得 4G 拍照。虽然移动 4G 的峰值 100Mbps 比 3G 的 HSPA+ 的 60Mbps 看起来没有差多少，但是其连接速度获得了质的提升。更重要的是 4G 时代的来临不只是移动网络速度的提升，还有资费的大幅降低。

如果说 3G 的普及带来了移动互联网的春天，4G 的资费降低造就了移动互联网多元化的发展，5G 则给移动带来了无限的遐想空间。ITU IMT-2020 规范要求 5G 速度高达 20Gbps，是 4G 网络的数百倍。试想一下几秒钟我们就可以下载一部 4GB 的高清影片的时代，还有什么可以阻碍我们的创新。曾经由于流量的限制而不能很好实现的 VR、AR、端上 AI 是否可以在新的时代绽放光彩？我们拭目以待。

移动新技术

世界是一辆一直向前的火车，不会因为某些人的上下车而停止脚步。2018 年末，苹果发布了 A12X 芯片，这块搭载在 iPad Pro 的芯片可谓是性能怪兽。据用户评测，这颗芯片已经达到了 i5-8xxx 的性能，而这一切都发生在移动端设备上。在移动端性能逐步提升的今天，让移动端完成许多原本只能在 PC 端完成的工作成为了可能。这其中就包括了 AI 和 AR/MR 技术。在 PC 时代，我们需要体验好的 VR，我们可能需要购买一块更好的显卡。当下，在移动端 CPU 不断增强的 AI 计算能力上，一切都将成为可能。

AI 技术从服务端向移动端迁移之路

人工智能（Artificial Intelligence，缩写为 AI）其实一直是一个古老的话题，早在上世纪 40 年代和 50 年代，来自不同领域（数学，心理学，工程学，经济学和政治学）的一批科学家便开始探讨制造人工大脑的可能性。在 1956 年，人工智能还被确立为一门学科。

然而 AI 的蓬勃发展是从近年来开始的，乔布斯离开我们前的最后一次发布会上，伴随着 iphone4s 一款 Siri 语音助理首次被苹果公司所提及。北京时间 2012 年 9 月 20 日凌晨，苹果在 iPod、iPhone 和 iPad 设备上正式放出 iOS6 操作系统更新。据更新信息显示，该升级包将有超过 200 项新功能，其中内置了全新的苹果地图。在此次更新中，Siri 也将支持中文。

Siri 这样一个语音助手开启了 AI 技术新的时代，也让广大用户更多的了解 AI，使用 AI。从而带来一场 AI 技术的变革。在 2016 年开始，AI 的技术不断被人们所提及，AlphaGo 战胜了韩国九段围棋选手李世石的新闻传遍了街头巷尾。而且随着移动端芯片的升级，AI 技术也由原来的云上计算渐渐转向了端上计算。

利用端上 AI，我们有了更为高效和智能的美颜，甚至视频中进行美颜也成为了可能。端上 AI 也帮助我们进行照片分类和自动美化处理。端上 AI 也能通过自然语言处理（NLP）更好的理解我们说的话。

AI 也许更懂你，不单可以图像处理和语言识别，也可以在推荐领域更好的发展。AI 可以推测大家打开美团外卖最想吃什么，甚至根据大家的喜好改变每个人的首页。相信在不远的未来，端上 AI 一定会带来更多的产业升级，让用户得到更为智能的场景体验。

从 VR、AR、MR 看移动端开发者的未来

视频技术的发展更多的受制于基础建设的影响，其实早在上个世纪 50 年代之前斯坦利·G·温鲍姆（Stanley G. Weinbaum）的科幻小说《皮格马利翁的眼镜（Pygmalion's Spectacles）》就已经提到过虚拟现实（Virtual Reality，缩写 VR）的概念。但是直到设备存储大小逐年扩大，设备运算能力，图形处理能力的不断加强，2015 年三星的 Gear VR 眼镜才配合三星 Galaxy 系列将 VR 设备带到家中。在 WWDC 2017 大会上，依托于 Metal 2 技术的星球大战 VR 体验也吸引了大家的眼球。

增强现实（Augmented Reality，简称 AR）这项技术其实也早在 1990 年就被前波音公司研究员 Tom Caudell 提出。同样是因为存储和计算能力的提升，WWDC 2017 大会上 ARKit 才问世。WWDC 2018 大会上更是推出了 ARKit 2.0。游戏和生活领域都有了很多的应用，例如淘宝的世界

杯 AR 活动，和链家的 AR 看房都是不错的应用。

混合现实（英语：Mixed Reality，简称 MR）指的是结合真实和虚拟世界创造了新的环境和可视化，物理实体和数字对象共存并能实时相互作用，以用来模拟真实物体。MR 可以理解成一种 VR 加 AR 的合成品。微软的 MR 设备 HoloLens 是其典型代表。

随着时代的发展，我们坚信会有更多酷炫的视频技术应用带给大家生活上新的体验，5G 和更多基础设施的提升一定会让未来更加炫目。所以新颖的视频技术一直是值得大家去关注的。

可穿戴技术

可穿戴技术主要探索和创造能直接穿在身上、或是整合进用户的衣服或配件的设备的科学技术。可穿戴技术是 20 世纪 60 年代，美国麻省理工学院媒体实验室提出的创新技术，利用该技术可以把多媒体、传感器和无线通信等技术嵌入人们的衣着中，可支持手势和眼动操作等多种交互方式。

谷歌联合创始人谢尔盖·布林于 2012 年 Google I/O 大会发布谷歌眼镜

2012 年 6 月 28 日，谷歌发布了一款穿戴式 IT 产品——谷歌眼镜。该设备由一块右眼侧上方的微缩显示屏，一个右眼外侧平行放置的 720p 画质摄像头，一个位于太阳穴上放的触摸板，以及喇叭、麦克风、陀螺仪传感器和可以支撑 6 小时电力的内置电池构成，结合了声控、导航、照相与视频聊天等功能。

Apple Watch，是苹果公司开发的一款智能手表，由蒂姆·库克于 2014 年 9 月 9 日发布。Apple Watch 也是一款可穿戴的智能设备。这款手表内置了 iOS 系统，并且支持 Facetime、WiFi、蓝牙、Airplay 等功能。

猫眼电影传媒有限公司的行业产品和大数据业务副总裁徐晓曾表达过移动技术的发展目前受制于屏幕的大小和电池续航能力的差异，可穿戴设备如果能在未来给予大家更大的视野、更丰富的交互形式、更长时间的续航。借助人体供电能力、视网膜投射等尖端技术，未来的移动将更加丰富多彩，这一天也许并不遥远。

后记

洋洋洒洒的带领大家回顾了十年的移动发展，然而这些文字并不足以涵盖这十年的风风雨雨。很多的技术沉淀了下来，被大家所记住。很多的技术没能跟得上历史的脚步，慢慢被大家所遗忘。在缅怀历史的同时，我们也要准备好迎接新的挑战。我们始终坚信未来一定会更美好，Eat better, Live better。

作者介绍

臧成威，美团大前端资深技术专家，开发经历 10 年，先后从事过嵌入式开发、传统导航软件开发、移动互联网开发。2015 年加入美团，现任美团 App 大前端架构师。负责美团 App 架构以及美团集团终端基础架构的架构设计工作。2015 年开始创建了 Hyperloop 持续集成及构建系统，2018 年带领团队开源了 EasyReact 响应式框架，2018 年参与了美团小程序平台的架构设计。擅长多种编程范式，函数式编程响应式编程。目前带领团队进行 Flutter 平台化的建设。

王志宇，美团终端技术团队技术专家。2015 年 8 月加入美团，先后负责配合 React Native 的引入推广，美团 iOS 版用户体验优化等工作。其负责的崩溃率指标由之前的 7‰ 下降到了历史最优的 0.36‰，远低于行业均值。热爱逆向相关技术，对 iOS 底层技术研究有着浓厚的兴趣。主要负责美团 iOS 版的崩溃率指标维护以及 App 性能优化的相关工作。参与响应式框架 EasyReact 开源工作，KSCrash 协作维护者。目前负责终端故障监控平台建设，参与数据处理链路、前端需求整理、组内协同等工作。

大数据十年之路

作者 宋词

大数据是当前最热的技术之一，这十年它经历了哪些阶段？每个阶段分别创造和发展了什么？未来大数据又将朝着哪些方向继续前行？在这篇文章里，我们沿大数据发展时间线，从产品、行业、技术多角度讨论其发展脉络，究其发展承其脉络大家可以学习、借鉴、并最终推测未来大致走向。

我曾有幸作为一个不处在核心岗位仍在电商边缘工作的观察者在电商崛起的时代被有幸卷入电商浪潮。我是亲眼看到以淘宝、天猫为代表中国电商行业崛起，那是一个创造金钱、财富、神话的时代，你可以比之为美国淘金时代。接下来的时代，即我们当前的时代，可能是一个更加激动人心、更加“技术爆炸“的时代，它们可能是IOT时代、是大数据时代、是云计算时代、是人工智能时代。兵贵神速，任何的先发优势在行业井喷年代造就的发展加速度是任何后续资本、人力、技术投入都不可比拟的，差之毫厘谬以千里，形容发展加速度亦十分贴切。

接下来，我会以大数据“史前”、大数据“当代”两个节点，讨论整体大数据发展特点和脉络。

大数据史前

如果我把大数据开创的时代当做公元元年纪年，那么大数据开创时代之前的时间称之为“史前时代”。我从不认为大数据之前的数据处理特别包括数据库理论才是数据处理的“田园时代”，以至于大批数据库理论学派发檄文声讨MapReduce理论的种种缺憾。同时我亦不认为当前整体软件系统设计过于复杂，整个大数据

生态过于冗余。IT 圈时常有人感慨，当前系统设计之臃肿、运作之复杂，实在有违“ Simplicity is beauty ”的原则，时常追忆当年玩 Unix Shell、翻翻 X86 中断手册，无比简约无比快感。但市场永远是对的，如果当前企业级软件市场确实需要如此复杂的软件体系、如此多样的系统生态，说明这部分理论基础以及工程实践确实需要如此复杂化、多样化。试看当今任何学科之前沿领域，无不复杂绝顶，即便同一学科不同子领域之下的职业从业者亦可能相互之间无法洞悉各自精髓。社会发展已经到如此精细化地步，不能责怪与之服务的人类知识体系。

大数据史前时代的数据处理，笔者粗略地将其划为两个时代边界：程序时代以及后续的数据库时代。

程序时代

从时间上，从计算机诞生到专业数据库发迹，那个时候的程序员过着普遍“ 轮子帮 ”的编码生活，即一切都需要靠手解决。这个是程序员的蛮荒时代，人人都面临着所有最基础、最原始、最重复的劳动工作，人人需要面对硬件、面对驱动，人人需要处理算法、处理数据结构；同时，这也是程序员的黄金时代，由于重复构建轮子所练就的技术内功，人人皆可为大神：随随便便读下 RFC 就可以完整实现 TCP/IP 协议栈（Bill Joy）、做个学校大作业就可以完成一个 Lex（Eric Schmidt），或者在大二在校时间写一个现代操作系统（Linus Torvalds）。

私以为，“**一切程序都是对于信息（数据）的处理**”应该是我们计算机处理问题的底层构建逻辑，是计算机领域放之四海而皆准的公理性断论。因此，不管是上古时代 C/S 架构的代码，抑或当前风头正紧的微服务、还是大数据中有关数据处理 Pipeline 的抽象，均是对于数据的处理抽象。

不同的是视当前问题的复杂程度、抽象对象的复杂程度，我们提供了不同层次的抽象设计原则而已。程序对于数据处理的抽象最为直接最为裸奔，整个程序处理逻辑都在于对计算机硬件执行抽象，是为数据处理 Action 的抽象，是 Input ->Action->Output 的抽象。该抽象对于数据的存储、数据的结构、数据的传输、数据的压缩等等均无定义，交由应用开发程序员自行安排、自行处理。这里的抽象最为原始、最为暴力，其灵活度甚高，同样其普适性亦涵盖整个计算机应用领域，无人敢说我的程序不是处理数据 / 信息的，因为整个计算机的发明即为信息 / 数据处理所服务的。因此我可以说，**程序，是为（大）数据处理最为初级的抽象方式**，这是大数据史前时代最为野蛮也最为普适的一个抽象。随着计算机科学以及工程的发展，我们一定会发展出更加高阶的抽象层次。

计算机科学 / 工程无不遵守社会发展大规律，当有大量底层系统构建者构建出操作系统、编译器、数据库之后，上层应用开发者仅仅需要了解底层 API 功能原理即可拼凑完整其上层业务构建。大数据系统 / 框架在分布式计算资源之上抽象了对于数据处理的 Pattern，大数据应用开发者只能 Follow 这个抽象去做相应的大数据计算和处理。构建底层的操作系统、数据库、中间件、大数据、AI，是社会分工的制定者，是游戏规则的设计者，是产业上游的利益分配者，他们能够定义整个 IT 商业市场的规模、玩法以及利益分配。

数据库时代

令人称赞的数据库时代，让程序员终将摆脱直接面对硬件磁盘结构以及底层文件系统进行数据操作。程序员稍加解脱可基于一个成熟的文件系统而不用去直接面向磁盘操作，但终究仍需要面对设计诸如索引结构、存储结构、并发读写、失败恢复等诸多底层系统设计细节。可以想象，诸如此类基础底层的系统类软件设计和实现，绝逼难度异于常规，非大神级别参与实现不可，这将变相提升整个软件项目的复杂度以及投入成本，并最终带来项目落地以及市场商业的巨大风险。

困难往往意味着机会，越普遍越难解的困难往往蕴藏着巨大的市场机会。诸如操作系统、数据库、编译器之类的软件产业基础服务，投入巨大、风险巨高，非一般公司可以为之。但同理而言，一旦此类公司发布出可商用的软件版本、构

建出可繁衍的软件生态，即可成就一番软件霸业。操作系统如此，看看曾经“不可一世”的微软、当前如日中天的苹果；数据库如此，瞧瞧遍大街的 Oracle“霸占”多少银行电信行业，养活多大上下市场生态。

在此，我想仅仅想讨论数据库领域两个我认为涉及到本文主旨的论点。

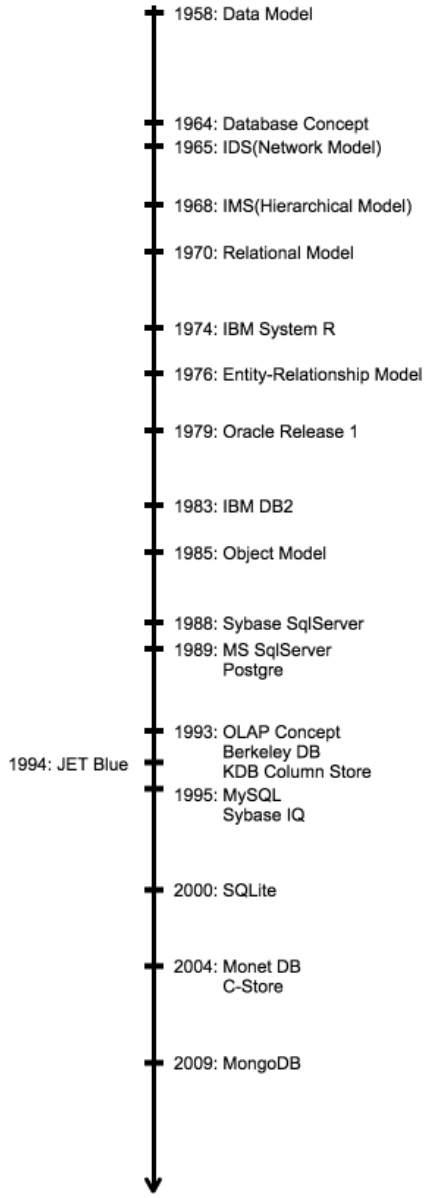
数据处理一次社会化的大分工

文件系统的抽象将存储进行了分工，大量开发人员不用在面对硬件和驱动读写存储，而是面向文件句柄、目录层次、读写权限，这是一个巨大的飞跃，恰到好处地将操纵硬件资源包装为一个面对有结构有层次利于操作的系列文件 API；再次，数据库将存储抽象从一个完全平面的、Bit 线性表、操作对象粒度为文件的层次再次提升到关系二维表（特指关系型数据库）、面向业务记录（数据行）、操作对象粒度细化到数据的层次。

抽象有利于业务表达，因为抽象会提供更高级别的 API（从文件 API 提升到数据 API），更有利于业务开发人员聚焦自身的业务表达。如果要让每个业务开发人员在文件系统层面操纵结构化数据存储，必定痛苦万分。数据库设计人员跨出这一步，提炼出高阶数据 API 同时丢失了文件系统普适性的广阔市场。于是乎，我们在云计算行业里面可以看到，阿里云的 OSS、AWS 的 S3 仍然可能承载了云上最广泛、最庞大的数据资源。因为其文件系统（当然这里是一个分布式文件系统）业务适配范围远远超出关系数据库的适配范围；同时，在阿里云 RDS 周边，我们看到了更多类似数据库的其他垂直领域数据处理系统，例如消息队列（提供流式数据读写的 API 抽象）、搜索引擎（提供全文检索服务的 API 抽象）等等，因为传统关系型数据库在聚焦关系代数二维模型情况下，让出其他数据存储和处理市场份额给到了其他数据系统。

SQL 与关系代数

我有充分的理由可以说，当前 IT 领域最火热的技术已经不是数据库，或者至少不是传统关系型的数据库。当前，包括大数据、人工智能、微服务、NoSQL 各类新型技术层出不穷、屡见不鲜，以至于我时常误以为 IT 技术到了刘慈欣《三体》所言的”技术爆炸“时代。身边人所言 IT 技术更新换代之快速，非常容易造成从业人员跟不上节奏，不无道理。从另外角度而言，不叫卖的技术要么早已技术基石、人人皆知，要么濒临淘汰、昨日黄花。诚然，关系型数据库绝非当



红炸子鸡，但其技术影响力绝非后者，而恰恰相反早已经是整个计算机科学和工程的基石。

数据库技术已成计算机基石，早已度过当年求曝光要宣传的流量小生时代。但不可否认，整个数据库技术为后续数据处理带来了大量基础科学理论以及工程实践。大量数据库理论论文、工程代码在后续的大数据、中间件领域中被大量借鉴、拷贝、甚至于 ” 抄袭 “ 。特别是关系型数据库有关数据处理模型的抽象：理论是关系代数、工程是 SQL 查询语言。

虽然说，数据库模型的历史虽然不是起源于关系型模型，但必须得说关系型数据库曾统一了所有的数据库模型，并一直统治至今。关系型模型进入人们视野的具体时间是 E.F.Codd 在 1970 年发表的论文 : ” A Relational Model of Data for Large Data Bank ” 。很多人对关系型数据模型的印象是表和字段，并还能想到的是表与表之间通过某些字段可以关联起来。似乎正因为这样，关系型数据库才被冠于这个名字。然而关系型数据模型中的“ 关系 ”在英文中对应的单词“ Relation ” 是一个抽象的数学概念，它既不是独指 2 个表之间的关系，也不等价于一个二维表（虽然习惯于以二维表来表示）。

上页图是有关数据库发展 Timeline，希望数据库的发展路线给大家看到整个数据库发展时间脉络。

大数据当代

理论

人人皆言，是 Google 最早提出了云计算的概念，但 Google 虽贵为云计算概念所创者，但只为云计算概念点赞却本身不落地推进执行，以至于被亚马逊抢占了市场。早期 Google 对业界另一大贡献即是大数据概念提出，同样不幸，Google 在大数据领域比云计算领域更是亲力亲为扮演活雷锋的角色。Google 与之 Hadoop 社区，好比一匆匆过客，眼见诸多理论被开源 “ 山寨 ” 进而被其他云计算公司使用，毫无作为令人叹惋。篇幅有限，在此我们仅从两个维度切入讨论 Google 的三驾马车，同时顺带聊聊 Google 在大数据领域的先发后至的思考。

大数据：退步还是进步？

讨论的第一个主题就是大数据相比于数据库在数据处理 “ 理论 ” 上是进步还是退化？从商业或市场上得出的结论是：大数据相比于数据库是市场进步，因为他们当前更加贴近市场对于大规模数据处理的诉求。

以 MapReduce 为例，当年 David J. DeWitt 以及 Michael Stonebraker 有关 MapReduce 的声讨檄文仍历历在目。2008 年，上述两位数据库大拿在 databasecolumn 网站发表《MapReduce: A major step backwards》（MapReduce: 一个巨大的倒退）基本上把互联网大数据派和数据库派之间的争吵推向一个高潮。任何一个稍懂数据库以及大数据的相关从业人员，都能够明确看到两者之间的严重分歧。于数据库人员而言：我派祖师爷数十年之心血积累，创建诸如关系模型、SQL 语言、ACID、存储优化等等理论精髓，方才以开山立派流芳百世。尔等把祖师爷数十年积累贬的一文不值，砍得七零八落，不是开历史倒车又是什么？数据库提了大致五点问题，摆出架势准备为数据处理的后生小辈谆谆教导一番：

- 在大规模的数据密集应用的编程领域，它是一个巨大的倒退。
- 它是一个非最优的实现，使用了蛮力而非索引。
- 它一点也不新颖——代表了一种 25 年前已经开发得非常完善的技术。
- 它缺乏当前 DBMS 基本都拥有的大多数特性。
- 它和 DBMS 用户已经依赖的所有工具都不兼容。

我认为上面问题将 MapReduce 当前设计实现的弊端描述得恰如其分，一点不冤。看 MapReduce 论文，其核心实现基本上推翻之前数据库几乎所有优秀研究成果，而采用了最原始最简单最暴力的实现方式，将就能用，但实属不雅。在互联网业务之局外人看来，特别在于数据库这帮学院派人士看来，类似处理方式无异于开历史之倒车。但身居互联网行业久矣，我深知互联网行事作风：快、糙、猛。互联网做事，能用就行，最好也能

快速占领市场。我管你们数据库之前如何设计精巧，今天要快速搞定我大 Google 大数据，为何不能做 trade-off。

从 MapReduce 之后，紧接着 2006 年 Google 再发大作《Bigtable: A Distributed Storage System for Structured Data》，BigTable 则是完全瞄准在线数据处理领域，讲述了用于存储和管理结构化数据的分布式存储系统，其建立在 GFS、MapReduce 等基础之上。该论文启发了后期的很多的 NoSQL 数据库，包括 Cassandra、HBase 等。如果说 MapReduce 完全专注于离线批量大数据处理 / 计算，则 BigTable 可以说和数据库完全在同一战场。可以想象适时诸多计算机学院派大牛当面对 BigTable 论文时必定摇头叹息：孺子不可教也。之后整个大数据行业借助 Hadoop 生态春风，蓬勃发展，至今十年有余，催生诸多云计算、大数据产品的市场。

在此，我想重申我的观点，**大数据是大数据时代之下系统演化结果，是更加贴近大数据场景下用户处理数据的诉求，而非开历史倒车**。大数据，我们讨论的就是一个“数据爆炸”时代下如何进行有效地大规模数据处理问题。这个问题是数据库之前未曾遇到、也未曾解决的特定问题，这些数据可能非结构化、非关系化，可能是半结构化的 Nginx 日志或者是用户上传的图片、再或者可能是整个城市大脑的交通探头高清视频数据。这些数据用传统的、狭义的关系型数据库无法解决，因此大数据方案舍弃了数据库模型中当前不适合上述数据处理的特性，牺牲某些功能从而换取大规模数据处理之能力。这是面向市场的、面向问题的、积极应对需求变化的技术做法，不教条也不做作。诚然，我相信大数据领域中某些领域，例如在处理关系数据事务型或者分析型场景下，可能仍然有大量数据库理论发挥作用，甚至看上去像一个数据库系统，如 Google Spanner；但在更大的数据处理与分析领域，我们将使用更多更分门别类的数据处理和存储方式，这类方式完全异于传统数据库，例如机器学习、例如图像识别。同时，我们可以预见，随着整个物理世界更多地数据化（上篇我们曾经讨论，凡是有利于加速信息生成、采集、传输、处理、反馈的技术都能够创造市场价值），而更多的物理社会数据化（IOT）、网络化（5G）势必造成更加复杂多样的数据处理需求类型，进而可以预见未来大数据处理会更加多样化，大数据分工于数据库系统，而接下来大数据同样内部面临巨大的分工：**更多更垂直更定制化的大数据系统将源源不断产生，以应对快速爆炸的数据时代**。

Google 大数据：机遇和失误

前文已述，Google 在大数据领域除了成就其技术影响力美名之外，基本毫无所获。其中一个原因，大概是缺乏对于开源社区的重视和投入。Google 以技术起家，十分重视技术影响力建设，以至于一直以来都是世界各大 IT 人员心中的技术灯塔。但从某种角度而言，技术影响力若无法变现，包括人才变现、营收变现，均是徒有虚名。Google 以三驾马车敲开大数据大门，虽打开一崭新行业，但概念虽新、落地很难，Google 显然缺乏让大数据在整个行业落地的动力和想法。同时，万万没想到开源社区竟然依样画葫芦”山寨“一把并最终形成 Hadoop 生态体系，并最终受众众多，用户甚广，时至今日 Hadoop 体系早已成为大数据行业事实标准。试想，如果当年 Jeff Dean 公开 MapReduce、GFS 论文同时，直接开放一套剥开 Google 内部系统依赖的完整开源软件。以 Google 自身强大的技术号召力，开源社区绝对不敢造次、多半服从 Google 技术生态。由此 Google 基本控制了大数据生态社区，后续云计算变现顺水推舟。但 Google 错失定义开源大数据软件机会，一失足成千古恨。

不过，Google 何等聪明伶俐，早已洞察一切。现在的 Google，从 TensorFlow、Kubernetes、Beam 开始，在技术开放之初，发表论文之时，就顺便开源一套软件技术内核，并投入重金支持开源社区构建。对于 Google 而言，社区即标准、社区即流量、社区即商业，一切都可以导向未来的商业化变现，长线投资、长期发展；而对于开源社区，如此巨头花重金支持生态发展，拍手称赞何乐不为。各取所需各获所利。

社区

Hadoop：开源大数据的基石

Hadoop 于 2005 年问世。之前，Doug Cutting 和 Mike Cafarella 已经拜读过 Google 的 GFS 论文，并且自己“手工造轮子”实现自己的 Google 分布式文件系统（最初称为 Nutch 分布式文件系统的 NDFS，后来改名为 HDFS 或 Hadoop 分布式文件系统）。在 2004 年时候，Google 发表神作《MapReduce: Simplified Data Processing on Large Clusters》，上述两位正在构架开源搜索引擎的大牛在考虑构建 Nutch webcrawler 的分布式版本正好需要这套分布式理论基础。因此，上述两位社区大牛基于 HDFS 之上添加 MapReduce 计算层。他们称 MapReduce 这一层为 Hadoop，由于 Hadoop 核心原理均是基于上述两篇论文，即 MapReduce 以及 GFS，其本身在技术理论上并无创新，更多是“山寨”实现。

Hadoop 技术相比于 Google 原作并无新意，甚至在 GFS 系统细节方面折扣实现不少。在笔者看来，Hadoop 体系能够成功，并在数据处理市场占据一席之地，其初期核心因素就在于以下几点。

- **时机**。彼时互联网 Web 2.0 风头正紧，大量用户与网站交互行为爆炸式增长，如何使用廉价的服务器去分析各类网站数据的业务需求已经迫在眉睫。此时，大量 Top 互联网公司有数据、有需求、有硬件，就缺一个廉价的数据分析系统。于是乎，开源、免费的 Hadoop 工具正好钻入此类大数据市场空档，迅速占领了核心种子客户群体，并为后续市场推广奠定了群众基础。
- **开源**。开源在开发者社区感染力不容小觑。Cutting 和 Cafarella 通过开源（以及 HDFS 的源代码）确保 Hadoop 的源代码与世界各地可以共享，最终成为 Apache Hadoop 项目的一部分。互联网时代下，大量软件被开发者以及背后的互联网商业公司作为开源系统贡献出来，整个互联网开发者行业已经被开源社区完全洗脑，仿佛开源就是人类灯塔，闭源就是万恶不赦。于是，此时，一个开源的、免费的、感觉挺符合互联网精髓的大数据处理软件出现在各大互联网公司圈中，迅速在互联网大数据处理领域触达了这部分市场群体。
- **商业**。早年开源软件皆靠诸位开源运动人士在业界做社区用户推广，这波人本身毫无金钱汇报全靠一腔精神热血。但本质上来说，**人类以及人类社会都是趋利性的**，没有利益驱动的市场行为终究无法持续。因此，早期没有找到合适盈利模式的开源软件一直发展缓慢，靠类似 Richard Stallman 类似开源黑客斗士去做市场推广，市场效率之低下。后期，在 Linux 商业公司红帽逐步摸索出开源软件变现模式后，其他开源软件也纷纷仿效。Hadoop 一时间背后迅速成立三家公司，包括 Cloudera、HotonWorks、MapR，这些公司盈利潜力完全都依赖于 Hadoop 开源生态的规模，因此，三家公司都会尽不遗余力地推进 Hadoop 生态发展，反过来促进了 Hadoop 整个生态用户的部署采用率。到大数据市场更后期的时代，其商业竞争更趋激烈。以 Kafka、Spark、Flink 等开源大数据软件为例，在各自软件提交到 Apache 基金会之时创始人立刻创办商业公司，依靠商业推进开源生态建设，同时通过收割生态最终反哺商业营收。

最终 Hadoop 在生态建设上获得了巨大的成功，其影响力在开源业界开创了一个崭新领域：大数据处理可见一斑。我们从如下几个维度来看看 Hadoop 生态成功的各类体现。

- Hadoop 的技术生态。不得不承认，Hadoop 有技术基座的先天优势，特别类似 HDFS 的存储系统。后续各大 Hadoop 生态圈中的大数据开源软件都多多少少基于 Hadoop 构建的技术底座。故而，大量大数据生态后起之秀基本均源于 Hadoop，或者利用 Hadoop 作为其基础设施，或者使用 Hadoop 作为上下游工具。此类依存共生关系在整个 Hadoop 社区生态已蔚然成风，越多大数据开源系统加入此生态既收割现有大数据生态客户流量，同时亦添加新功能进入 Hadoop 社区，以吸引更多用户使用 Hadoop 生态体系。就好比淘宝买家卖家

相互增长，形成商业互补，相辅相成。

- Hadoop 的用户生态。前文已述，优秀的开源（免费）系统确实非常容易吸收用户流量、提升用户基数，这个早已是不争事实。通过开源 (免费) 的系统软件铺开发者市场、培养开发者习惯、筹建开发者社区，早已是开源软件背后商业公司的公开市场打法，这就类似通过免费 APP 培养海量客户技术，最终通过收割头部客户实现营收。或者好比一款游戏，大部分可能均是免费玩家，但用户基数达可观规模之时，一定涌现出不少人民币玩家，并通过他们实现整体营收。当前风头正紧的开源大数据公司，包括 DataBricks（Spark）、Confluent（Kafka）、Ververica（Flink）莫不如此。在开源软件竞争激烈日趋激烈的环境下，其背后若无商业公司资金支撑，其背后若无市场商业团队运营支撑，当年写一个优秀的开源软件就凭 ”酒香不怕巷子深“ 的保守概念，现如今早已推不动其软件生态圈发展。试看当前大数据生态圈，那些日暮西山、愈发颓势的开源软件，其背后原因多多少少就是缺乏商业化公司的运作。
- Hadoop 的商业生态。大量商业公司基于 Hadoop 构建产品服务实现营收，云计算公司直接拉起 Hadoop 体系工具作为大数据云计算服务，软件集成商通过包装 Hadoop 引擎提供客户大数据处理能力，知识机构（包括书籍出版社、Hadoop 培训机构）通过培训 Hadoop 开发运维体系实现营收和利润，上述种种商业行为均基于 Hadoop 体系实现商业利润。整个 Hadoop 开创了开源大数据的新概念，并由此养活大数据行业数不胜数的参与者。这波参与者享受了开源 Hadoop 的收益，同时也在为 Hadoop 贡献知识。

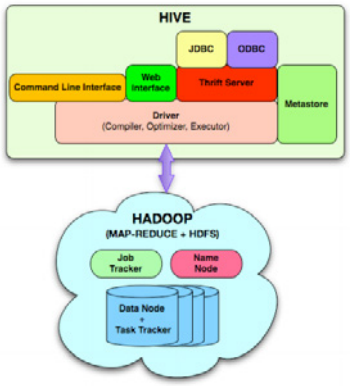
如果说 Google 三篇论文发表后敲开了大数据时代的理论大门，但论文异常高冷、不接地气、无法落地投产。真正人人皆用大数据的时代是直到开源社区提供了成熟的 Hadoop 软件生态体系之后，我们才可以说企业界方才逐步进入到大数据时代。可以说，当代 Hadoop 的诞生，为企业大数据应用推广起到了决定性作用。

生态：大数据的林林总总

Google 三家马车叩开了大数据理论大门，而 Hadoop 才真正开启了行业大数据时代。前文已述，Hadoop 已经早已超出 MapReduce（计算模型）和 HDFS（分布式存储）软件范畴。当前早已成为 Hadoop 大数据代名词，指代大数据社区生态，无数号称新一代、下一代的大数据技术无不构建在 Hadoop 生态基石之上。下文我们分维度重点讨论基于 Hadoop 生态之上的各式各样大数据组件抽象。

高阶抽象：让人人成为大数据用户

上篇道，数据库两位大拿 David J. DeWitt 以及 Michael Stonebraker 十分不待见 MapReduce 论文所诉理论，基本上是羽檄争执、口诛笔伐。其讨伐重点之一便是使用 MapReduce 而抛弃 SQL 抽象，将实际问题的解决难度转嫁用户非正确的系统设计方式。同样，这个批判确为 MapReduce 之缺陷，凡是正常人类绝逼感同身受。一个普普通通数据处理业务，用 SQL 表达多则百行、少则数行，熟练的数据工程师多则数小时少则数分钟即可完成业务开发，轻松简便。而一旦切换到 MapReduce，需要将 SQL 的直接业务表达子句换成底层各种 Map、Reduce 函数实现，少则数千行多达数万行，导致整个数据开发难度陡增，业务开发效率抖降。针对上述两个问题，Hive 给出解决方案。



开源社区受众更广，其使用者更多，因此实

际上开源社区对于开发效率提升诉求更高。但 Hadoop 社区似乎对于 SQL 情有独钟，更多将精力投入到 SQL-On-Hadoop 类的工具建设之中，最终，社区吸收 Facebook 提交给 Apache 基金会的 Hive，并形成了业界 SQL-On-Hadoop 的事实标准。

对于 Hive 而言，其官网特性说明充分阐释了 Hive 的作为一套 Hadoop MapReduce 之上的 DSL 抽象之价值和特性。

1. Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供完整的 sql 查询功能，可以将 sql 语句转换为 MapReduce 任务进行运行。其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析。
2. Hive 是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。

Hive 一个重要的意义是将 MapReduce 进一步进行抽象为业务高阶语言，让更多不善于 Java/C++ 编码的数据工程师能够快速上手使用大数据工具进行数据分析，让大数据业务开发成本更低、使用门槛更低、维护成本更低，让传统的、使用数据库的数据分析师能够基本无缝迁移到基于 Hadoop 的大数据平台，从而极大扩展了大数据用户群体，进一步拉动 Hadoop 社区的用户生态以及商业生态。从另外一个方面，Hive 作为一个 SQL-On-Hadoop 工具，为后续诸多大数据处理软件提供了很好的表率：即越高阶的业务抽象 API 能够极大降低用户开发门槛，拉动使用者基数。后续大量的开源闭源大数据系统都或多或少、有模有样地提供了各自 SQL 方言，方便用户更加快速、简单地上手各自的大数据软件。开源社区来看，从 Hive 开始，Presto、Impala、Spark、或者当前风头正紧的 Flink，无不提供 SQL 作为高阶数据分析语言。闭源产品而言，阿里云的 MaxCompute、AWS 的 Redshift、Google 的 BigQuery，均提供各自 SQL 抽象以争取更多云上开发人员的使用。

实时计算：让计算更快产生业务价值

大数据诞生之初给使用者最大的疑惑在于：为何计算如此之慢？彼时使用数据库多数查询在亚秒级别返回，使用数仓多数查询在数秒级别返回，到 Hadoop 的大数据时代，大部分查询在数分钟、数小时、甚至于数天级别方才能够查询出结果。大数据、大数据，在一旦解决计算可处理的问题之后，时效性问题便摆上台面。

大数据领域下，时效性分为两个方面：

1. 数据从用户请求到最终呈现的实时性，这条路径强调的是请求响应的及时性；
2. 数据从发生到处理、并最终产生业务价值的全链路时效性，这条路径强调的是数据链路及时性。

前者我们称之为交互式计算领域，后者我们称之为实时（流）计算领域。我一直认为交互式查询是技术方面的优化，是人人痛恨 MapReduce 计算模型太慢太落后的技术优化，和产品形态并无太大关联。而后者，则是整个大数据产品模型的变化，这是一种触发式计算，有点类似阿里云的 FunctionCompute，或者是 AWS 的 Lambda；或者更加准确地定义是：基于事件流的实时流计算。我们通常看到三个系统：

第一代：Storm

Storm 是 Nathan Marz 的心血结晶，Nathan Marz 后来在一篇题为《History of Apache Storm and lessons learned》的博客文章中记录了其创作历史。这篇冗长的博客讲述了 BackType 这家创业公司一直在自己通过消息队列和自定义代码去处理 Twitter 信息流。Nathan 和十几年前 Google 里面设计 MapReduce 相关工程师有相同的认识：实际的业务处理的代码仅仅是系统代码很小一部分，如果有个统一的流式实时处理框架负责处理

各类分布式系统底层问题，那么基于之上构建我们的实时大数据处理将会轻松得多。基于此，Nathan 团队完成了 Storm 的设计和开发。

将 Storm 类比为流式的 MapReduce 框架，我自认为特别贴切，因为这个概念类比更好向各位看官传达了 Storm API 的 low-level 以及开发效率低下，这类基础大数据的 API 让业务人员参与编写业务逻辑好比登天。同时，Storm 的设计原则和其他系统大相径庭，Storm 更多考虑到实时流计算的处理时延而非数据的一致性保证。后者是其他大数据系统必备基础产品特征之一。Storm 针对每条流式数据进行计算处理，并提供至多一次或者至少一次的语义保证。我们可以理解为 Storm 不保证处理结果的正确性。

第二代：Spark

Spark 在 2009 年左右诞生于加州大学伯克利分校的著名 AMPLab。最初推动 Spark 成名的原因是它能够经常在内存执行大量的计算工作，直到作业的最后一步才写入磁盘。工程师通过弹性分布式数据集（RDD）理念实现了这一目标，在底层 Pipeline 中能够获取每个阶段数据结果的所有派生关系，并且允许在机器故障时根据需要重新计算中间结果，当然，这些都基于一些假设 a）输入是总是可重放的，b）计算是确定性的。对于许多案例来说，这些先决条件是真实的，或者看上去足够真实，至少用户确实在 Spark 享受到了巨大的性能提升。从那时起，Spark 逐渐建立起其作为 Hadoop 事实上的继任产品定位。

在 Spark 创建几年后，当时 AMPLab 的研究生 Tathagata Das 开始意识到：嘿，我们有这个快速的批处理引擎，如果我们将多个批次的任务串接起来，用它能否来处理流数据？于是乎，Spark Streaming 就此诞生。相比于 Storm 的低阶 API 以及无法正确性语义保证，Spark 是流处理的分水岭：第一个广泛使用的大规模流处理引擎，既提供较为高阶的 API 抽象，同时提供流式处理正确性保证。

第三代：Flink

Flink 在 2015 年突然出现在大数据舞台，然后迅速成为实时流计算部分当红炸子鸡。从产品技术来看，Flink 作为一个最新的实时计算引擎，具备如下流计算技术特征。

1. 完全一次保证：故障后应正确恢复有状态运算符中的状态。
2. 低延迟：越低越好。许多应用程序需要亚秒级延迟。
3. 高吞吐量：随着数据速率的增长，通过管道推送大量数据至关重要。
4. 强大的计算模型：框架应该提供一种编程模型，该模型不限制用户并允许各种各样的应用程序在没有故障的情况下，容错机制的开销很低。
5. 流量控制：来自慢速算子的反压应该由系统和数据源自然吸收，以避免因消费者缓慢而导致崩溃或降低性能。
6. 乱序数据的支持：支持由于其他原因导致的数据乱序达到、延迟到达后，计算出正确的结果。
7. 完备的流式语义：支持窗口等现代流式处理语义抽象。

你可以理解为整个实时计算产品技术也是时间发展而逐步成熟发展而来，而上述各个维度就是当前称之为成熟、商用的实时计算引擎所需要具备的各类典型产品技术特征。Flink 是当前能够完整支持上述各类产品特征参数的开源实时流处理系统。

全家桶：一套引擎解决“所有”大数据问题

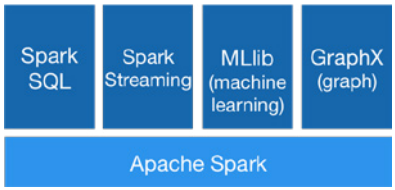
Flink 和 Spark：All-In-One

大数据全家桶其实是一个实打实的产品问题：从大数据社区反馈的情况来看，对于大部分大数据处理用户，实际上的大数据处理诉求分类有限，基本上在 Batch（60%+）、Stream（10%+）、Adhoc（10%+）、其他（包括 ML、Graph 等等）。对于任何一个大数据处理引擎深入做透一个领域后，势必会考虑下一步发展，是继续做深做专，抑或还是横向扩展。做又红又专？从商业来看，这个领域的市场规模增长可能有限，眼瞅着都到天花板了；但从横向角度来看，周边大数据引擎虎视眈眈，随时都有杀入我们现有市场之中。面对市场，各色需求可穷举；面对技术，引擎基础

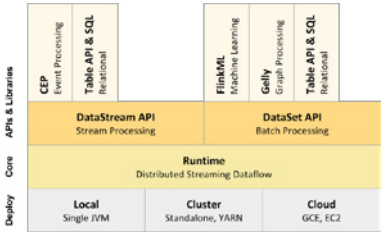
业已夯实，为何不周边突破扩展，开拓新的大数据领域。Spark 从批入手，针对 Hadoop 处理性能较差的问题，将 Spark 的 Batch 功能做成一个“爆款应用”，同时提供 Spark Streaming、Spark ML，前期靠 Spark Batch 为整个 Spark 社区用户倒流，并吸收为 Spark Streaming、Spark ML 的客户。而通过 Spark 大数据全家桶功能，Spark 产品构建一个粘性的护城河。大量中小用户一旦全功能上了 Spark，大家理解后续很难因为 Spark 某个功能点不太满足需求而抛弃使用 Spark。

Spark 从批入手，尝试在一个技术栈体系内统一基础的大数据处理；在另外一个方向，Flink 从 Stream 入手，在构建出 Flink Stream 强大生态后，也在考虑布局 Flink Batch，从 Stream 切入 Batch 战场。

下图是 Spark 的软件栈体系。



下图是 Flink 的软件栈体系。



两者在产品功能栈上早已十分相似，缺的是各自薄弱领域的精细化打磨。在此，我们不讨论两者功能强弱分布，毕竟本文不是一个产品功能介绍文章。未来两个系统鹿死谁手花落谁家，各位看官拭目以待吧。

Beam: One-Fits-All

前文已述，早期 Google 在大数据领域纯粹扮演了一个活雷锋的角色，以至于整个开源大数据生态蓬勃发展起来，并最终形成完整的大数据商业生态之时，Google 基本门外一看客，眼瞅着自己的技术理论在开源社区发扬光大，自己没捞半点好处。Google 痛定思痛，考虑建设开源社区并尝试力图控制社区。于是在此背景之下，Apache Beam“粉墨登场”。Google 考虑问题之核心不在于是否要开源一个大数据处理系统（当时社区 Spark 已经蔚然成风，Flink 的发展同样亦是扶摇直上，似乎社区并无缺少一个好的大数据引擎），而仅仅缺乏开源社区大数据处理接口之统一，包括将核心的批处理以及流处理接口统一。而之前 Google 内部的 FlumeJava 一直承担大数据 Data Pipeline 之 API 定义角色，于是想当然地从内部将之前的 FlumeJava 接口进行抽象改进，提供统一的批流处理 API 后在 2016 年贡献提交给 Apache 基金会。试图通过定义一套统一的 API 抽象层，说服各个厂商实现该套抽象，即可完成 API 统一的千秋大业，并为用户迁移 Google 云上埋下最大伏笔。但在 Apache Beam 发布之后，除 Flink 社区意图想联合 Beam 社区一起做做技术影响力之外，其他开源大数据产品和 Beam 一直若即若离。开源社区难以有真心和 Beam 社区结成歃血同盟者，社区合作者多半皆是善于投机的机会主义者。一时间，Beam 的前景黯淡。

结语

本文浏览了整个大数据发展历史，特别重点关注了数据计算处理部分的内容。我们从程序本质以及行业发展脉络分别描述了大数据之前的程序时代以及数据库时代，大数据当代的各类理论、系统、社区发展。我们观察历史，是让我们有信心面对当前；我们分析当代，是让我们有机会看清未来。未来整个云计算以及之上的大数据发展，已经超出本文的讨论范畴，但从上述历史、当代分析能对未来其发展推测一二，欢迎大家自行思考。文章行文较为仓促，定有纰漏之处，望各位看官不吝赐教。

容器混合云，Kubernetes 助力基因分析

作者 李鹏

一家大型基因测序功能公司每日会产生 10TB 到 100TB 的下机数据，大数据生信分析平台需要达到 PB 级别的数据处理能力。这背后是生物科技和计算机科技的双向支撑：测序应用从科研逐步走向临床应用，计算模式从离线向在线演进，交付效率越来越重要。

引言

James Watson 和 Francis Crick 于 1953 年发现了 DNA 的双螺旋结构，从此揭开了物种进化和遗传的神秘面纱，开启了人类对数字化遗传的认知，但是人类基因奥秘却是一点点被读懂的。

1956 年，一则癌症和染色体相关性的发现令整个癌症研究界震动：慢性骨髓性白血病（CML）患者的第 22 号染色体，比一般然明显短很多。二十余年后，学者们发现，9 号染色体的 Abl 基因，与 22 号染色体的 BCR 基因连到了一块，交错易位产生了一条 BCR-Abl 融合基因。BCR-Abl 蛋白一直处于活跃状态且不受控制，引发不受控的细胞分裂，从而导致癌症。

也就是说，只要细胞表达 BCR-Abl 蛋白，就有血癌风险。美国着手深入研究，并成功推出了治疗慢性骨髓性白血病的新药。这，就是格列卫，也是去年《我不是药神》中被我们熟知的‘高价药’。

在格列卫诞生前，只有 30% 的慢性骨髓性白血病患者能在确诊后活过 5 年。格列卫将这一数字从 30% 提高到了 89%，且在 5 年后，依旧有 98% 的患者取得了血液学上的完全缓解。为此，它也被列入了世界卫生组织的基本药物标准

清单，被认为是医疗系统中“ 最为有效、最为安全，满足最重大需求 ” 的基本药物之一。

容器混合云如何应对基因测序的 IT 挑战

基因测序在血液肿瘤领域应用的越来越广泛。根据病人的诊断结果，血液肿瘤专科医生会选择相应的检查，比如 PCR 结合实时荧光探针技术，来检测测 BCR-Abl 融合基因，以诊断慢性骨髓性白血病，也可以通过二代测序方式，SEGF（Single-end Gene Fusion）能够通过单端 NGS 测序数据检测复杂的基因融合类型。

在另一面，无创产检唐氏／爱德华式筛查，近年来以高准确率和对胎儿的低风险，越来越受到国内年轻产妇的欢迎。基因公司每年都完成几十万例的 NIPT 检查，每一例的 NIPT 涉及到数百 MB+ 的数据处理，存储和报告生成。一家大型基因测序功能公司每日会产生 10TB 到 100TB 的下机数据，大数据生信分析平台需要达到 PB 级别的数据处理能力。这背后是生物科技和计算机科技的双向支撑：测序应用从科研逐步走向临床应用，计算模式从离线向在线演进，交付效率越来越重要。

基因计算面临以下几方面挑战。

1. 数据存储：数据增长快，存储费用高，管理困难；长期保存数据可靠性难以保障；需要寻求低成本大数据量的数据压缩方式；元数据管理混乱，数据清理困难。
2. 分发共享：海量数据需要快速、安全的分发到国内多地及海外；传统硬盘寄送方式周期长，可靠性低；多地中心数据需要共享访问。
3. 计算分析：批量样本处理时间长，资源需求峰谷明显，难以规划；大规模样本的数据挖掘需要海量计算资源，本地集群难以满足；计算工作流流程迁移困难、线上线下调度困难、跨地域管理困难；线下弹性能力差，按需计算需求。
4. 安全合规：基因数据安全隐私要求极高；自建数据中心安全防护能力不足；数据合约（区块链）；RAM 子账号支持。

而这样看来一套完备架构方案则是必不可少的。与传统高性能计算相比，按需切分任务的需求，自动从云中申请资源，自动伸缩能力达到最小化资源持有成本，90% 以上的资源使用率，用完后自动返还计算资源。最大化资源的使用效率，最低单样本的处理成本，最快速的完成大批量样本的处理。随着基因测序业务增长，自动完成线下资源使用，和线上资源扩容。高速内网带宽，和高吞吐的存储，和几乎无限的存储空间。

基因计算不同于常规的计算，对海量数据计算和存储能力都提出了很高的要求。主要通过容器计算的自动伸缩特性和阿里云 ECS 的自动伸缩能力的打通，可以大规模弹性调度云上的计算资源。通过对基因数据的合理切分，实现大规模的并行计算同时处理 TB 级别的样本数据。通过按需获取的计算能力，以及高吞吐的对象存储的使用，大幅降低了计算资源持有的成本和单个样本的处理成本。

整体技术架构是云原生容器混合云，云上云下资源一体，跨地域集群统一管理。作为主要 Player，容器技术在数据分拆，数据质量控制，Call 变异提供了标准化流程化、加速、弹性、鉴权、观测、度量等能力，在另外一方面，高价值挖掘需要借助容器化的机器学习平台和并行框架对基因、蛋白质、医疗数据完成大规模线性代数计算来建立模型，从而使精准医疗能力成为现实。

基因工程中的关键问题及解决方案

数据迁移与传输

数据迁移、数据拆分阶段百万小文件的读取对底层的文件系统压力，通过避免不必要小文件的读写提高样本的处理效率。通过数据中心与阿里云的专线连接，实现高吞吐低延迟的数据上云以及与工作流结合的上云、校验、检测方式。而最终需要达成的目标是：在短时间内完成数十 TB 级数据的加密搬迁，确保数据传输客户端的高性能与安全性，实现并发传输、断点续传，且保有完善的访问授权控制。

基因计算典型任务：增强型工作流

基因计算的典型特征就是数据分批计算，需要按照特定步骤先后依次完成。将该问题抽象后，即需要申明式工作流定义 AGS(AlibabaCloud Genomics Service) workflow。

其工作流的特点是：多层次，有向无环图。科研大工作流 1000-5000+ 深度的 DAG，需要准确的流程状态监控和高度的流程稳定性。简单流程从任意步骤重现启动，失败步骤可以自动完成重试和继续，定时任务，通知，日志，审计，查询，统一操作入口 CLI/UI 。

我们采用的方案是：

- 简单 YAML 申明式定义，多层次，有向无环图，复杂依赖支持，任务自动分拆，自动并行化；
- 云原生，与社区 Argo 完全兼容的增强性 Workflow 定义；
- 失败步骤自动 Retry 增强，简单流程从任意步骤 Retry。
- 实时资源统计，监控集成云监控，云日志 SLS 集成，审计集成，定时任务；
- 统一操作入口 ags-cli 与 Kubectl 集成；
- 阿里云存储卷申明式支持，NAS，OSS，CloudDisk，缓存加速支持。

云上云下资源的统一调度

通过跨越 IDC 和云上可用区的混合云 ACK 集群实现计算资源的统一调度和数据的云端汇聚。自动化，流程化上云数据，和后续的数据处理流程，形成 24 小时内完成批次下机数据的本地，上云，云端处理和报告生成。按需弹性提供计算节点或者无服务化计算资源，形成按需计算能力，处理突发分析任务。我所带领的阿里云基因数据服务团队努力构建更具弹性的容器化集群，分钟级数百节点自动伸缩能力和分钟级数千轻量容器拉起的 Serverless 能力，通过提高并行度来提高内网带宽的利用率，最终提高整体数据吞吐率，通过 NAS 客户端和服务端的 TCP 优化来提高 IO 读写速度，通过为 OSS 增加缓存层和分布式的缓存来实现对象存储读取加速，等等。

还有很多问题，篇幅原因在此不一一展开：如何进行基因数据管理、最优化单位数据处理成本、采用批量计算的方式进行对样本分析、怎样使得基因数据处理安全及跨组织安全分享等等。

生命科学和精准医学应用，未来已来

NovaSeq 测序仪带来了低成本（100$/WGS）高产出（6TB 通量）的二代测序方案，大量 NovaSeq 的使用为基因测序公司每天产出的几十 TB 数据，这就要求大量的算力来分拆和发现变异，以及需要大量的存储来保存原始数据和变异数据。阿里云基因数据服务不断提升极致弹性的计算能力，和大规模并行处理能力，以及海量高速存储来帮助基因公司快速自动化处理每天几十上百 TB 的下机数据，并产通过 GATK 标准产出高质量的变异数据。

以 PacBio 和 Nanopore 为代表的三代测序的出现，超过 30K 到数百 K 的长读，和 20GB 到 15TB 的大通量产出，长读和数据量对数据比对，分拆，发现变异带来了更大的算力需要和高 IO 吞吐的需求，对基因计算过程中优化基因分析流程，拆分数据，按需调度大量计算资源，提供超高的 IO 吞吐带来了更大的挑战。

作者介绍

李鹏（Eric Li），阿里云资深架构师，数据科学家，美国 FDA2018 精准医疗大赛 Top2 Winner，金融／生物计算行业解决方案专家，专注于基于 Kubernetes 的容器产品开发和银行，生信行业的生产落地。在加入阿里云之前，曾在 IBM 担任 Watson 数据服务容器平台首席架构师，机器学习平台架构师，IBM 2015 Spark 全球大赛金奖获得者，带领多个大型开发项目，涵盖云计算，数据库性能工具、分布式架构、生物计算，大数据和机器学习。

阿里云 PB 级 Kubernetes 日志平台建设实践

作者 元乙

阿里云日志服务是阿里集团针对日志分析、处理的自研产品。Kubernetes 近两年来发展十分迅速，已经成为容器编排领域的事实标准，但是 Kubernetes 中日志采集相对困难，阿里云日志服务技术专家元乙将在 QCon 北京 2019 分享 Kubernetes 日志平台建设最佳实践，借此机会我们采访了元乙老师阿里云 Kubernetes 日志平台是如何建设的。

背景

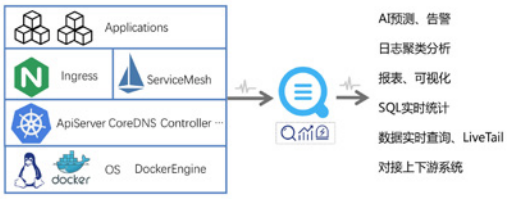
阿里云日志服务是阿里集团针对日志分析、处理的自研产品，最根本的目的是让用户专注在“分析”上，远离琐碎的工作。日志服务整体功能分为 3 个部分：日志采集、智能查询分析和数据分发。相比其他日志系统，阿里云日志服务有以下几个特点：

1. 采集范围广，支持 30+ 种的数据采集通道，包括服务器、交换机、容器、移动端、IOT 等各类设备；支持全球加速、断点续传等功能，使全球化数据高可靠采集成为可能。
2. 数据规模大，支持单用户日 PB 级数据写入，提供数据通道横向自动扩展能力，可根据数据流量进行自动扩容。
3. 查询能力强，提供 SQL92 标准的分析语法，秒级即可分析 10 亿条数据，同时提供丰富的数据图表，提供所见即所得的数据分析能力。
4. 下游渠道多，日志服务支持对接各类下游的数据处理、分析系统，包括 Flink、Storm、Spark 等各类流计算系统，同时也支持 Hadoop、MaxCompute 等离线分析系统。

Kubernetes 日志采集难点

Kubernetes 近两年来发展十分迅速，已经成为容器编排领域的事实标准。Kubernetes 是一个大的生态系统，围绕着 Kubernetes 我们需要去解决很多问题，例如 CI/CD、可扩展性、安全性、可观察性等等。日志是可观察性中必不可少的一部分，而在 Kubernetes 中日志采集相对更加困难，难点主要体现在以下 3 个方面。

1. 采集目标多，Kubernetes 平台运行着多种系统组件以及众多应用，这些日志由多种日志格式（分隔符、json、Java、Nginx 等）和多种日志形式（宿主机文件、容器 stdout、容器文件、syslog、journal 等）组成，通常主流的 Agent 很难支持各种数据的采集。
2. 环境动态性强，在 Kubernetes 中，各类服务都会进行自动的缩扩容，应用也会进行动态迁移，日志采集很难适应环境动态性强的系统，尤其是日志的完整性很难得到保证。
3. 使用负担大，随着集群规模、使用人数、应用种类的逐渐增长，日志采集的集中式管理、采集可靠性的监控等需求就显得尤其重要。更进一步，如何基于 Kubernetes 的扩展能力让日志采集也能和 Kubernetes 资源一样进行统一的管理？



阿里云 Kubernetes 日志平台的整体功能和核心技术

阿里云 Kubernetes 日志平台为 Kubernetes 日志提供接入、查询、分析、可视化、下游对接等日志分析整个生命周期的完整方案，并针对 Kubernetes 的组件日志提供通用的解决方案，例如审计日志、Ingress 日志、系统组件日志等。这里的核心技术主要有以下几点。

1. 全方位日志采集，能够支持 Kubernetes 各类日志的实时采集，并兼顾低资源消耗、高性能、高可靠性。同时基于 CRD 扩展，实现采集与 Kubernetes 的无缝集成。
2. 超大规模数据量，Kubernetes 可轻松管理数万台机器的集群，日数据量可能会达到数百 TB 甚至 PB 级，日志平台能够支撑海量的数据规模，同时保证可扩展性和可靠性。
3. 实时分析能力，日志最常用的场景是监控和问题调查，因此实时的查询 / 分析能力尤其重要，平台能够在秒级内实现对亿级数据任意条件、任意维度组合的分析。
4. 超高性价比，针对日志特点进行针对性系统优化，降低海量数据的存储、处理成本，最大化利用机器资源，在成本控制上对开源方案形成压倒性优势。
5. 通用方案打通，基于日志平台的通用能力，对 Kubernetes 的通用方案进行整体封装，并打通采集、存储、分析、可视化、告警等整个流程，实现通用方案的开箱即用。

平台设计难点

阿里云在通用日志平台的建设方面有着 10 年的经验，针对 Kubernetes 场景平台整体的复杂性增加很多，难点主要有一下几点。

1. 支持各类采集需求，需支持采集多种日志形式和日志格式，不同的使用场景对日志采集的需求不同，需要保证数据采集具备高可靠、高性能、低资源占用、可监控的能力。
2. Kubernetes 集成，日志的采集和管理需要和 Kubernetes 平台进行无缝兼容，因此需要提供 CRD 的扩展方式，尤其在多种方式同时操

作、集群不可用等复杂场景下，CRD 与服务端的同步与协调关系较难维护。

3. 多租户隔离，Kubernetes 日志平台的使用方较多，平台需保证从日志采集、处理、查询、消费等各个环节的多租户隔离，不能让部分用户的大量请求或非法使用而导致整个集群不可用。
4. 超大流量压力，在阿里内部，即使最大规格的 VIP 也无法承受所有日志的流量，双 11、春节红包等流量高峰瞬间可能会打爆集群，因此减少数据回路、削峰填谷、降级方案、系统兜底方案等尤其重要。

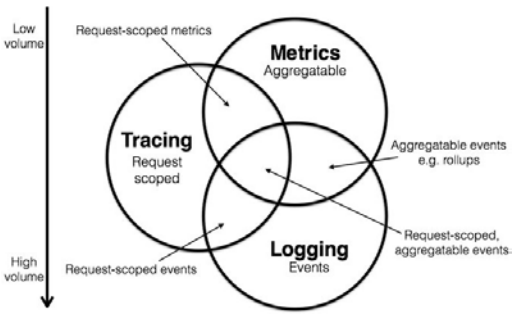
日志数据的使用

Kubernetes 中存在各种日志，包括内核日志、系统组件日志、Ingress、ServiceMesh、中间件、应用日志等，每种日志都会有不同人员在不同的场景中应用。例如 APIServer 的审计（Audit）日志，安全同学会用来做入侵检测、账号操作审计等，运维同学会基于审计日志做变更管理、核心组件监控、节点监控等，开发同学会使用审计日志检查变更是否生效；例如 Ingress 的访问日志，运营同学会用来做用户行为分析、业务走势分析、运营检测等；运维同学会用来做集群 / 服务监控；开发同学会基于 Ingress 访问日志进行发布前后的指标对比。

从日志平台角度来看，平台需要为不同的业务角色、不同的使用场景提供通用的数据处理 / 分析能力，包括但不限于：智能分析、链路跟踪、监控、数据清洗、流计算、数据仓库、安全分析、BI 分析等。

Kubernetes 日志平台与可观察性的关系

“可观察性”（Observability）从电气角度上的解释是：“若所有的内部状态都可以输出到输出信号，此系统即有可观察性”。CNCF-Landscape 首次将“可观察性”（Observability）引入到了 IT 领域。可观察性相关的工具主要包括 Logging、Metric、Tracing 三大类，这三者之间有很多重叠部分，从表现力上来看，Metrics 最弱、Tracing 其次、Logging 最强。Metric 主要记录了一些聚合的指标信息，例如 CPU/Mem 利用率、请求成功率、请求延迟等；Tracing 记录从请求发起到响应完毕的整个流程；而日志相对范畴最大，日志记录了系统运行期间所有的信息，而从日志的字段中可以聚合出 Metric、从日志的 RequestID 中可以提取出整个 Tracing 链路。



在 Kubernetes 中，通常通过 Metric 发现问题，然后通过 Tracing 定位问题模块，最后根据日志中的详细信息诊断错误。而在阿里云 Kubernetes 日志平台中可通过智能分析的功能直接基于日志发现、定位并诊断问题，大大减少问题调查时间。智能分析的能力主要有一下几类。

- 日志聚类，根据日志的相似性进行智能归类，秒级即可实现亿级数据自动归类，快速掌握日志整体状态。
- 异常检测，基于变点检测、折点检测、多周期检测、时序聚类等机器学习方法，自动检测时序中的异常。
- 日志模式对比，通过前后两个时间段 / 版本的日志模式对比，快速发现当前时间 / 版本

的日志差异。

- 知识库匹配，将问题调查经验以知识库形式保存下来，将日志与知识库内容进行匹配，快速得到具体日志对应的问题和解决方案。

阿里云 Kubernetes 日志平台的借鉴意义

阿里云 Kubernetes 日志平台建设过程中考虑的很多问题对于大家都有一定的借鉴意义，例如：

采集方案选择，Kubernetes 中采集通常会使用 DaemonSet 和 Sidecar 两种方式，日志也会分为 stdout 和文件两种形式，文件也分为容器内文件和宿主机挂载文件等不同方式，需根据业务场景特点选择合适的日志采集方案。

平台高可用建设，随着应用场景的逐渐扩展，对于日志平台的可用性要求也越来越高，我们在高可靠日志采集、日志平台自监控、异常自动屏蔽与恢复、高效运维等方面积累了很多宝贵的经验与教训。

生态对接，平台不可能实现日志生命周期中所需的所有系统和功能，很大一部分功能需要上下游的生态来完成（例如流计算、离线计算、Trace 系统、告警系统等），因此生态的对接成为了日志平台能够覆盖所有日志场景必不可少的一个部分。

性能与成本取舍，成本是每个公司都需要考虑的一点，通常日志的开销只占 IT 支出的 1-3% 左右，日志的采集、存储、查询等各个环节需尽可能的节省资源，同时还需保证整体性能在可接受范围内。

做混沌工程是什么样的体验？阿里：有点刺激

作者 王晓博

如果把系统架构比作一个房子，实施混沌工程就类似对房子进行验房的过程。通过模拟有损和无损的的场景来观察房子质量，目标是暴露结构性风险，不会去关注每个细节。

混沌未分天地乱，茫茫渺渺无人见。

因为对未知的好奇，盘古劈开了天地，身化做了天地。再后来，人类不满足于眼前的狭小世界，渴望天地之外，无人得见的另一番天地。

是以未知探寻未知，遍观未知，方知无知。

混沌工程，便因未知而生。

这一通过有意识地注入故障，从而发现系统的未知弱点的实验手段，现在还很年轻，在行业内的认知和实践累积也比较少，很多 IT 团队对它的理解还没有上升到一个领域的概念。但阿里很敢，率先付出了长久的关注与实践。

今天我就跟大家聊聊阿里的混沌工程。

其实，早在 2011 年，阿里电商就开始尝试通过故障注入技术去解决微服务的依赖问题，从注入实现、实验效率、业务影响等多个方面进行演进。2016 年开始，我们尝试在线上复现一些重大故障并验证措施是否有效。此时，阿里内部还称之为 ” 故障演练 ” 。

直到 2016 年去参加旧金山的 QCon 大会，我们发现很多人在讨论混沌工程，他们想要解决的问题与我们想要的不谋而合，于是，阿里开始关注这个技术领域。

越来越多的企业选择基于云原生技术构建系统架构，希望可以构建容错性好、易于管理和便于观察的松耦合系统。但我们需要意识到的是，在围绕新技术、新理念来升级系统架构和组织模式的同时，一定会遇到一些超出预期的不确定问题。

如何减小不确定性问题对系统稳态的影响，保持系统状态的一致性，是我们实施混沌工程的主要契机。

再说说我自己吧。

2011 年，我加入了阿里巴巴高可用架构团队，此后长期参与稳定性产品研发和集团架构演进工作，主导了强弱依赖、灰度环境、故障演练、智能对账等多款高可用产品的研发和建设，见证了阿里高可用产品体系从 1.0 到 3.0 的发展历程，积累了丰富的架构和稳定性经验。

2015 年，作为共享事业部的双 11 负责人，我负责大促和常态稳定性的保障工作。

一晃 8 年过去了，现在，我在阿里负责高可用技术的云化输出和技术演进工作，是阿里云产品 AHAS（应用高可用）的技术负责人。

如果你要问我，在推进混沌工程的过程中是否有一些比较刺激的经历，那我会说，还真有。

刚刚推进混沌工程的时候，我们优先选择的是在生产环境复现历史发生的故障，验证故障措施改进的有效性。之前的故障措施的验收往往是在线下环境验证或走一个流程。那次实验，我们选择了一个非常严重的故障做验收。虽然我们做了比较充分的评估，但是真正在生产环境实施的时候，还是比较紧张。

会是成功还是失败？答案是，成功。

最终故障注入后，相关系统完成了容灾切换，业务曲线几乎无影响。

成功是一针兴奋剂，增强了大家对系统稳定性的信心，我们都受到了鼓舞。

为什么阿里要选择混沌工程啊？

也许你会疑惑，既然现在已经有了故障注入和故障测试，比如单元测试和集成测试，那么，为什么阿里还要选择混沌工程？

我认为，它们虽有一定的重叠性，但混沌工程是发现新信息的实践。

单元测试和集成测试的核心目标是检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。

故障注入和故障测试的定义是通过引入故障，让程序走入一些不常经过的路径，是一种提高程序覆盖率的手段。

混沌工程虽然也使用短期的度量结果代表系统状态，但一般来讲度量结果都是一个监控的指标，而非具体的接口返回值。混沌工程是一个围绕稳态进行验证和探索的过程，每次实验可能都会产生新的数据。

如果把系统架构比作一个房子，实施混沌工程就类似对房子进行验房的过程。通过模拟有损和无损的的场景来观察房子质量，目标是暴露结构性风险，不会去关注每个细节。

看起来有点意思，对吧？

那你想不想知道，混沌工程是不是与你有关？

如果你是一名系统架构师、测试人员、SRE 人员，那你需要关注混沌工程。

那么，企业如何确定自己是否适合混沌工程？满足以下部分特征的企业，都适合实施混沌工程：

- 对于资损或 SLA 有极高要求的行业（金融、电商、医疗、游戏、公共等）
- 有较大的系统改造（比如：系统重构、微服务化、容器化、迁云）
- 业务发展快，稳定性积淀少
- 人员流动快，或 SRE 团队规模较小

对于使用混沌工程的团队，我建议：

- 要先从文化上让团队认可实施混沌工程的必要性和方法论。
- 重点关注实验方案的评估，明确定义系统稳态和终止条件。
- 尝试通过一些简单的场景开始尝试，增加大家对系统和过程的信心。
- 与其他团队进行合作，通过混沌工程配合其他稳定性手段，达成整体目标（如：监控发现率、故障改进覆盖率、系统自愈有效性等）。

下面我要说的很重要。

在使用混沌工程之前，我们需要准备三点：

- 思想上要准备好。要意识到系统故障是可以通过周期性的引入一些实验变量来提前暴露和解决的。不论是否实施混沌工程，系统的隐患或 Bug 都客观存在。
- 系统稳态和实验方案的仔细评估。对于实验

方案进行推演，如果已经可以预想到一些问题，那么修正后再进行新的实验。

- 提升系统的可观测性。对于系统稳态，要有配套的监控或观测工具，否则会影响混沌工程的实施效果。

你知道的，实践出真知。

作为第一批吃螃蟹的，我们也做了将近十年了，阿里在故障模拟、爆炸半径的控制、产品化方面出了一些成绩。业务方基本可以做到很低成本使用我们的产品，DevOps 同学基本已经可以实现自助演练。

从 2016 年到现在，看到大家对领域的认可度逐年变高，演练覆盖的应用规模和发现问题的数量已经翻了几倍，帮助业务方识别了很多潜在故障和改进点，很多高速发展的领域，比如新零售，也是通过实施混沌工程来快速的落地和改进稳定性，我还挺高兴的。

虽然混沌工程在行业内没有一个比较广的认知，但除去 Netflix 和阿里之外，国内外也有不少公司在做这个。

比如，国外有 Gremlin、ChaosIQ 这样专门实施 Resilience as a Service 的商业公司。像一些中、大型公司也都有实施混沌工程团队，比如 Linkedin、Uber、Google 等等。

国内的公司，据我所知，百度、华为、美团、京东、猫眼电影等，应该都有一些实施经验。

目前开源社区的工具主要关注在故障注入层面。商业化产品一般也是基于产品 + 专家服务的模式。

下面，我推荐一些好用的工具吧。

开源工具

kube-monkey、PowerfulSeal、ChaosIQ，提供了一些容器层面的故障注入能力。详细可以看：https://github.com/dastergon/awesome-chaos-engineering

近期阿里会开源一款混沌工程测试工具 ChaosBlade，提供基础资源、应用服务、容器等多维度的故障模拟能力。

商业化工具

Gremlin 提供一款商用的故障注入平台，部分功能免费，目前在公测中。

阿里云 - 应用高可用服务（AHAS）：AHAS 供了基于混沌工程原则的完整的实现，除了提供常见的故障注入能力，默认也打通了一些常见的云服务，提升系统的可观测性和自动化能力。目前免费公测中（支持非阿里云机器公网使用）。

随着企业云化，一定会有越来越多的公司开始关注和实施混沌工程。我希望可以有更多的公司分享思考和实践，并结合领域场景产出一些最佳的实践。

居安思危，思则有备，有备无患。这是《左传襄公十一年》说的。

此前，有人认为，混沌工程是一种在故障发生之前发现故障的技术，但也是一种心态。

我觉得这句话还是有道理的。混沌工程是一种验证系统对非预期情况防御有效性的实验思想，任何依照 ” 混沌工程原则 ” 进行的探索，都是有效实践。

系统架构可能会很复杂，比如采用了微服务、Docker、K8s，甚至函数计算类似的技术，需要实验项目也涉及很多门类。为了更有效地实施混沌工程，就需要借助一些场景丰富、操作简便、模型标准的工具或技术了。

混沌工程技术会随着其他技术发展而演进，混沌工程会与融入到每个领域的最佳实践中。

今年开始，我开始在一些场合将自己称呼为 ” 混沌工程布道师 ”。

布道师，听起来好像有点 “ 忽悠 ”。

不过，你知道皮埃罗·斯加鲁菲（Piero Scaruffi）吗？这位全球人工智能及认知科学专家，被誉为永远站在时代前面的硅谷精神布道者，我很敬佩他。推动技术传播，推动技术落地，推动行业变革，这就是布道者的意义之所在。

在阿里内部，其实并没有设置一个专门的布道师的岗位。作为技术 Leader 或架构师，完成既定的业务目标是最核心，也是最本分的任务，除此之外，他们还活跃在外界，扮演着产品宣导者和技术推广人的角色。

让更多的人知道他们的团队在做什么，从而发现更多可能。

我希望，能通过这种手段给自己一定的责任和压力，持续性地去关注和推进领域的发展，让

更多的人了解并加入这个领域。

愿为星星之火，这是我的选择。

作者介绍

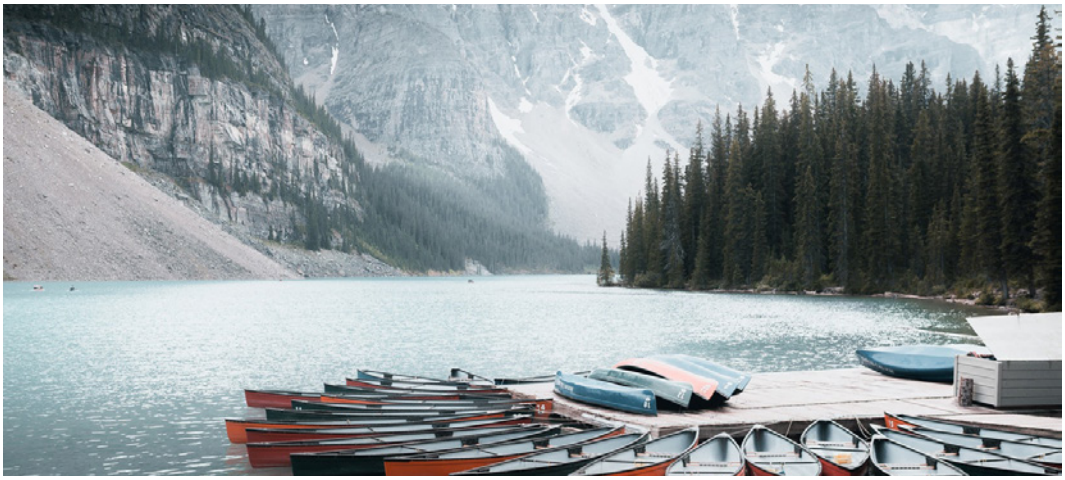
阿里巴巴高级技术专家，花名中亭。2011 年加入阿里巴巴高可用架构团队，长期参与稳定性产品研发和集团架构演进工作，主导了强弱依赖、灰度环境、故障演练、智能对账等多款高可用产品的研发和建设，见证了阿里高可用产品体系从 1.0 到 3.0 的发展历程，积累了丰富的架构和稳定性经验。

2015 年作为共享事业部的双 11 负责人，负责大促和常态稳定性的保障工作。目前负责高可用技术的云化输出和技术演进工作，阿里云产品 AHAS（应用高可用）的技术负责人。

2017 年 QCon 明星讲师，混沌工程布道师。

“容器”十年，关于软件交付的编年史

作者 张磊



伴随着容器技术普及到“千家万户”，我们在这两年期间所经历的，是现代软件交付形式的一次重要变革。

在这一年，全世界的开发人员都开始习惯用容器测试自己的软件，用容器做线上发布，开始对容器化的软件构建和交付流程习以为常。全世界的架构师们都在对“云原生”侃侃而谈，描绘多云时代的应用治理方式，不经意间就把“sidecar”这种容器组织方式当做了默认选项。在“云”已经成为了大众基础设施的今天，我们已经习惯了把“容器“当做现代软件基础设施的基本依赖。这就像我们每天打开 Eclipse 编写 Java 代码一样自然。

不过，不要忘了，只需要往回倒数两年，整个容器生态都还在围着 Docker 公司争得不可开交，看起来完全没有定数。当时的国内很多公有云厂商，甚至都没有正式的 Kubernetes 服务。在那个时候，要依托容器技术在云上托管完整的软件生命周期，可以说是相当前沿的探索。谁能想到短短两年之后，容器这个站在巨人肩膀上的设计，就真的成为技术人员日常工作的一部分呢？

事实上，伴随着容器技术普及到“千家万户”，我们在这两年期间所经历的，是现代软件交付形式的一次重要变革。

源起：属于 Jails 们的时代

虚拟化容器技术（virtualized container）的历史，其实可以一直追溯上世纪 70 年代末。感谢 Wikipedia 等活历史，我们现在至少还能从文字上了解到在很多读者出生前，一群来自贝尔实验

室（ Bell Laboratories ）的“ 黑客 ”研究员们正试图用“ 容器 ”的概念来解决什么样问题。

时间回到 1979 年，贝尔实验室正在为 Unix V7 （Version 7 Unix）操作系统的发布进行最后的开发和测试工作。在那个时候，Unix 操作系统还是贝尔实验室的内部项目，而运行 Unix 的机器则是长得像音响一样的、名叫 PDP 系列的巨型盒子。在那个“ 软件危机（The Software Crisis）”横行的末期，开发和维护 Unix 这样一套操作系统项目，即使对贝尔实验室来说也绝非易事。更何况，那里的程序员们还得一边开发 Unix ，一边开发 C 语言本身呢。

而在 Unix V7 的开发过程中，系统级别软件构建（Build）和测试（Test）的效率其实是其中一个最为棘手的难题。这里的原因也容易理解：当一个系统软件编译和安装完成后，整个测试环境其实就被“ 污染 ”了。如果要进行下一次构建、安装和测试，就必须重新搭建和配置整改测试环境。在有云计算能力的今天，我们或许可以通过虚拟机等方法来完整的复现一个集群。但在那个一块 64K 的内存条要卖 419 美元的年代，“ 快速销毁和重建基础设施 ”的想法还是有点“ 科幻 ”了。

所以，贝尔实验室的聪明脑袋们开始构思一种在现有操作系统环境下“ 隔离 ”出一个可供软件进行构建和测试的环境。更确切的说，就是我能否简单的执行一些指令，就改变一个程序的“ 视图 ”，让它把当前目录当做自己的根目录呢？这样，我每次只要在当前目录里放置一个完整操作系统文件系统部分，该软件运行所需的所有依赖就完备了。

更为重要的是，有了这个能力，开发者实际上就间接拥有了应用基础设施“ 快速销毁和重现 ”的能力，而不需要在环境搭好之后进入到环境里去进行应用所需的依赖安装和配置。这当然是因为，现在我的整个软件运行的依赖，是以一个操作系统文件目录的形式事先准备好的，而开发者只需要构建和测试应用的时候，切换应用“ 眼中 ”的根目录到这个文件目录即可。

于是，一个叫做 chroot（Change Root）的系统调用就此诞生了。

顾名思义，chroot 的作用是“ 重定向进程及其子进程的根目录到一个文件系统上的新位置 ”，使得该进程再也看不到也没法接触到这个位置上层的“ 世界 ”。所以这个被隔离出来的新环境就有一个非常形象的名字，叫做 Chroot Jail。

值得一提的是，这款孕育了 chroot 的 Unix V7 操作系统，成为了贝尔实验室 Unix 内部发行版的绝唱。在这一年末尾， Unix 操作系统正式被 AT&T 公司商业化，并被允许授权给外部使用，自此开启了一代经典操作系统的传奇之旅。

而Chroot的诞生，也第一次为世人打开了“ 进程隔离 ”的大门。

伴随着这种机被更广泛的用户接触到，chroot 也逐渐成为了开发测试环境配置和应用依赖管理的一个重要工具。而“ Jail ”这个用来描述进程被隔离环境的概念，也开始激发出这个技术领域更多的灵感。在 2000 年，同属 Unix 家族的 FreeBSD 操作系统发布了 ”jail“ 命令，宣布了 FreeBSD Jails 隔离环境的正式发布。相比于 Chroot Jail，FreeBSD Jails 把 ”隔离“ 这个概念扩展到了进程的完整视图，隔离出了独立进程环境和用户体系，并为 Jails 环境分配了独立的 IP 地址。所以确切的说，尽管 chroot 开创了进程环境隔离的思想，但 FreeBSD Jails，其实才真正实现了进程的沙箱化。而这里的关键在于，这种沙箱的实现，依靠的是操作系统级别的隔离与限制能力而非硬件虚拟化技术。

不过，无论是 FreeBSD Jails（Unix 平台），还是紧接着出现的 Oracle Solaris Containers（Solaris 平台），都没有能在更广泛的软件开发和交付场景中扮演到更重要的角色。在这段属于 Jails 们的时代，进程沙箱技术因为“ 云 ”的概念尚未普及，始终被局限在了小众而有限的世界里。

发展：云，与应用容器

事实上，在 Jails 大行其道的这几年间，同样在迅速发展的 Linux 阵营上也陆续出现多个类似的沙箱技术比如 Linux VServer 和 Open VZ（未进入内核主干）。但跟 Jails 的发展路径比较类似，这些沙箱技术最后也都沦为了小众功能。我们再次看到了，进程沙箱技术的发展过程中“ 云 ”的角色缺失所带来的影响，其实还是非常巨大的。

而既然说到“云”，我们就不得不提到基础设施领域的翘楚：Google。

Google 在云计算背后最核心的基础设施领域的强大影响力，是被业界公认的一个事实，无论是当初震惊世界的三大论文，还是像 Borg/Omega 等领先业界多年的内部基础设施项目，都在这个领域扮演者重要的启迪者角色。然而，放眼当今的云计算市场，仅仅比 Google 云计算发布早一年多的 AWS 却是“云”这个产业毫无争议的领导者。而每每谈到这里的原因，大家都会提到一个充满了争议的项目：GAE。

GAE 对于中国的某几代技术人员来说，可以说是一个挥之不去的记忆。然而，即使是这批 GAE 的忠实用户，恐怕也很难理解这个服务竟然就是 Google 当年决定用来对抗 AWS 的核心云产品了。事实上，GAE 本身与其说是 PaaS，倒不如说是简化版的 Serverless。在 2008 年，在绝大多数人还完全不知道云计算为何物的情况下，这样的产品要想取得成功拿下企业用户，确实有些困难。

不过，在这里我们想要讨论的并不是 Google 的云战略，而是为什么从技术的角度上，Google 也认为 GAE 这样的应用托管服务就是云计算呢？

这里的一个重要原因可能很多人都了解过，那就是 Google 的基础设施技术栈其实是一个标准容器技术栈，而不是一个虚拟机技术栈。更为重要的是，在 Google 的这套体系下，得益于 Borg 项目提供的独有的应用编排与管理能力，容器不再是一个简单的隔离进程的沙箱，而成为了对应用本身的一种封装方式。依托于这种轻量级的应用封装，Google 的基础设施可以说是一个天然以应用为中心的托管架构和编程框架，这也是很多前 Googler 调侃“离开了 Borg 都不知道怎么写代码”的真实含义。这样一种架构和形态，映射到外部的云服务成为 GAE 这样的一个 PaaS/Serverless 产品，也就容易理解了。而 Google 这套容器化基础设施的规模化应用与成熟，可以追溯到 2004~2007 年之间，而这其中一个最为关键的节点，正是一种名叫 Process Container 技术的发布。

Process Container 的目的非常直白，它希望能够像虚拟化技术那样给进程提供操作系统级别的资源限制、优先级控制、资源审计能力和进程控制能力，这与前面提到的沙箱技术的目标其实是一致的。这种能力，是前面提到的 Google 内部基础设施得以实现的基本诉求和基础依赖，同时也成为了 Google 眼中“容器”技术的雏形。带着这样的设思路，Process Container 在 2006 年由 Google 的工程师正式推出后，第二年就进入了 Linux 内核主干。不过，由于 Container 这个术语在 Linux 内核中另有它用，Process Container 在 Linux 中被正式改名叫作：Cgroups。

Cgroups 技术的出现和成熟，标志在 Linux 阵营中“容器”的概念开始被重新审视和实现。更重要的是，这一次“容器”这个概念的倡导者变成了 Google：一个正在大规模使用容器技术定义世界级基础设施的开拓者。

在 2008 年，通过将 Cgroups 的资源管理能力和 Linux Namespace 的视图隔离能力组合在一起，LXC（Linux Container）这样的完整的容器技术出现在了 Linux 内核当中。尽管 LXC 提供给用户的能力跟前面提到的各种 Jails 以及 OpenVZ 等早期 Linux 沙箱技术是非常相似的，但伴随着 Linux 操作系统开始迅速占领商用服务器市场的契机，LXC 的境遇比前面的这些前辈要好上不少。而更为重要的是，2008 年之后 AWS，Microsoft 等巨头们持续不断的开始在公有云市场上进行发力，很快就催生出了一个全新的、名叫 PaaS 的新兴产业。

老牌云计算厂商在 IaaS 层的先发优势以及这一部分的技术壁垒，使得越来越多受到公有云影响的技术厂商以及云计算的后来者，开始思考如何在 IaaS 之上构建新的技术与商业价值，同时避免走入 GAE 当年的歧途。在这样的背景之下，一批以开源和开放为主要特点的平台级项目应运而生，将“PaaS”这个原本虚无缥缈的概念第一次实现和落地。这些 PaaS 项目的定位是应用托管服务，而不同于 GAE 等公有云托管服务，这些开放 PaaS 项目希望构建的则是完全独立于 IaaS 层的一套应用管理生态，目标是借助 PaaS 离开发者足够近的优势锁定云乃至所有数据

中心的更上层入口。这样的定位，实际上就意味着 PaaS 项目必须能够不依赖 IaaS 层虚拟机技术，就能够将用户提交的应用进行封装，然后快速的部署到下层基础设施上。而这其中，开源、中立，同时又轻量、敏捷的 Linux 容器技术，自然就成为了 PaaS 进行托管和部署应用的最佳选择。

在 2009 年，VMware 公司在收购了 SpringSource 公司（Spring 框架的创始公司）之后，将 SpringSource 内部的一个 Java PaaS 项目的名字，套在了自己的一个内部 PaaS 头上，然后于 2011 年宣布了这个项目的开源。这个项目的名字，就叫做：Cloud Foundry。

Cloud Foundry 项目的诞生，第一次对 PaaS 的概念完成了清晰而完整的定义。这其中，“ PaaS 项目通过对应用的直接管理、编排和调度让开发者专注于业务逻辑而非基础设施 ”，以及“ PaaS 项目通过容器技术来封装和启动应用 ”等理念，也第一次出现在云计算产业当中并得到认可。值得一提的是，Cloud Foundry 用来操作和启动容器的项目叫做：Warden，它最开始是一个 LXC 的封装，后来重构成了直接对 Cgroups 以及 Linux Namespace 操作的架构。

实际上，Cloud Foundry 等 PaaS 项目的逐渐流行，与当初 Google 发布 GAE 的初衷是完全一样的。说到底，这些服务都认为应用的开发者不应该关注于虚拟机等底层基础设施，而应该专注在编写业务逻辑这件最有价值的事情上。这个理念，在越来越多的人得以通过云的普及开始感受到管理基础设施的复杂性和高成本之后，才变得越来越有说服力。在这幅蓝图中，Linux 容器已经跳出了进程沙箱的局限性，开始扮演的“ 应用容器 ”的角色。在这个新的舞台上，容器和应用终于画上了等号，这才最终使得平台层系统能够实现应用的全生命周期托管。

按照这个剧本，容器技术以及云计算的发展，理应向着 PaaS 的和以应用为中心的方向继续演进下去。

如果不是有一家叫做 Docker 的公司出现的话。

容器：改写的软件交付的历程

如果不是亲历者的话，你很难想象 PaaS 乃至云计算产业的发展，会如何因为 2013 年一个创业公司开源项目的发布而被彻底改变。但这件事情本身，确实是过去 5 年间整个云计算产业变革的真实缩影。

Docker 项目的发布，以及它与 PaaS 的关系，想必我们已经无需在做赘述。一个“ 降维打击 ”，

就足以给当初业界的争论不休画上一个干净利落的句号。

我们都知道， Docker 项目发布时，无非也是 LXC 的一个使用者，它创建和使用应用容器的逻辑跟 Warden 等没有本质不同。不过，我们现在也知道，真正让 PaaS 项目无所适从的，是 Docker 项目最厉害的杀手锏：容器镜像。

关于如何封装应用，这本身不是开发者所关心的事情，所以 PaaS 项目有着无数的发挥空间。但到这如何定义应用这个问题，就是跟每一位技术人员息息相关了。在那个时候，Cloud Foundry 给出的方法是 Buildpack，它是一个应用可运行文件（比如 WAR 包）的封装，然后在里面内置了 Cloud Foundry 可以识别的启动和停止脚本，以及配置信息。

然而，Docker 项目通过容器镜像，直接将一个应用运行所需的完整环境，即：整个操作系统的文件系统也打包了进去。这种思路，可算是解决了困扰 PaaS 用户已久的一致性问题，制作一个“ 一次发布、随处运行 ”的 Docker 镜像的意义，一下子就比制作一个连开发和测试环境都无法统一的 Buildpack 高明了太多。

更为重要的是，Docker 项目还在容器镜像的制作上引入了“ 层 ”的概念，这种基于“ 层 ”（也就是“ commit ”）进行 build，push，update 的思路，显然是借鉴了 Git 的思想。所以这个做法的好处也跟 Github 如出一辙：制作 Docker 镜像不再是一个枯燥而乏味的事情，因为通过 DockerHub 这样的镜像托管仓库，你和你的软件立刻就可以参与到全世界软件分发的流程当中。

至此，你就应该明白，Docker 项目实际上解决的确实是一个更高维度的问题：软件究竟应该通过什么样的方式进行交付？

更重要的是，一旦当软件的交付方式定义的如此清晰并且完备的时候，利用这个定义在去做一个托管软件的平台比如 PaaS，就变得非常简单而明了了。这也是为什么 Docker 项目会多次表示自己只是“ 站在巨人肩膀上 ”的根本原因：没有最近十年 Linux 容器等技术的提出与完善，要通过一个开源项目来定义并且统一软件的交付历程，恐怕如痴人说梦。

云，应用，与云原生

时至今日，容器镜像已经成为了现代软件交付与分发的事实标准。然而， Docker 公司却并没有在这个领域取得同样的领导地位。这里的原因相比大家已经了然于心：在容器技术取得巨大的成功之后，Docker 公司在接下来的“ 编排之争 ”中犯下了错误。事实上，Docker 公司凭借“ 容器镜像 ”这个巧妙的创新已经成功解决了“ 应用交

付”所面临的最关键的技术问题。但在如何定义和管理应用这个更为上层的问题上，容器技术并不是“银弹”。在“应用”这个与开发者息息相关的领域里，从来就少不了复杂性和灵活性的诉求，而容器技术又天然要求应用的“微服务化”和“单一职责化”，这对于绝大多数真实企业用户来说都是非常困难的。而这部分用户，又偏偏是云计算产业的关键所在。

而相比于 Docker 体系以“单一容器”为核心的应用定义方式，Kubernetes 项目则提出了一整套容器化设计模式和对应的控制模型，从而明确了如何真正以容器为核心构建能够真正跟开发者对接起来的应用交付和开发范式。而 Docker 公司、Mesosphere 公司以及 Kubernetes 项目在“应用”这一层上的不同理解和顶层设计，其实就是所谓“编排之争”的核心所在。

2017 年末，Google 在过去十年编织全世界最先进的容器化基础设施的经验，最终帮助 Kubernetes 项目取得到了关键的领导地位，并将 CNCF 这个以“云原生”为关键词的组织和生态推向了巅峰。

而最为有意思的是，Google 公司在 Kubernetes 项目倾其全力注入的“灵魂”，既不是 Borg/Omega 多年来积累下来的大规模调度与资源管理能力，也不是“三大论文”这样让当年业界望尘莫及的领先科技。Kubernetes 项目里最能体现 Google 容器理念的设计，是“源自 Borg/Omega 体系的应用编排与管理能力”。

我们知道，Kubernetes 是一个“重 API 层”的项目，但我们还应该理解的是，Kubernetes 是一个“以 API 为中心”的项目。围绕着这套声明式 API，Kubernetes 的容器设计模式，控制器模型，以及异常复杂的 apiserver 实现与扩展机制才有了意义。而这些看似繁杂的设计与实现背后，实际上只服务于一个目的，那就是：如何让用户在管理应用的时候能最大程度的发挥容器和云的价值。

本着这个目的，Kubernetes 才会把容器进行组合，用 Pod 的概念来模拟进程组的行为。才会坚持用声明式 API 加控制器模型来进行应用编排，用 API 资源对象的创建与更新（PATCH）来驱动整个系统的持续运转。更确切的说，有了 Pod 和容器设计模式，我们的应用基础设施才能够与应用（而不是容器）进行交互和响应的能力，实现了“云”与“应用”的直接对接。而有了声明式 API，我们的应用基础而设施才能真正同下层资源、调度、编排、网络、存储等云的细节与逻辑解耦。我们现在，可以把这些设计称为“云原生的应用管理思想”，这是我们“让开发者专注于业务逻辑”、“最大程度发挥云的价值”的关键路径。

所以说，Kubernetes 项目一直在做的，其实是在进一步清晰和明确“应用交付”这个亘古不变的话题。只不过，相比于交付一个容器和容器镜像，Kubernetes 项目正在尝试明确的定义云时代“应用”的概念。在这里，应用是一组容器的有机组合，同时也包括了应用运行所需的网络、存储的需求的描述。而像这样一个“描述”应用的 YAML 文件，放在 etcd 里存起来，然后通过控制器模型驱动整个基础设施的状态不断地向用户声明的状态逼近，就是 Kubernetes 的核心工作原理了。

未来：应用交付的革命不会停止

说到这里，我们已经回到了 2019 年这个软件交付已经被 Kubernetes 和容器重新定义的时间点。

在这个时间点上，Kubernetes 项目正在继续尝试将应用的定义、管理和交付推向一个全新的高度。我们其实已经看到了现有模型的一些问题与不足之处，尤其是声明式 API 如何更好的与用户的体验达成一致。在这个事情上，Kubernetes 项目还有不少路要走，但也在快速前行。

我们也能够看到，整个云计算生态正在尝试重新思考 PaaS 的故事。Google Cloud Next 2019 上发布的 Cloud Run，其实已经间接宣告了 GAE 正凭借 Kubernetes 和 Knative 的标准 API“浴火重生”。而另一个典型的例子，则是越来越多很多应用被更“极端”的抽象成了 Function，以便完全托管于与基础设施无关的环境（FaaS）中。如果说容器是完整的应用环境封装从而将应用交

付的自由交还给开发者，那么 Function 则是剥离了应用与环境的关系，将应用交付的权利交给了 FaaS 平台。我们不难看到，云计算在向 PaaS 的发展过程中被 Docker “ 搅局 ” 之后，又开始带着 “ 容器 ” 这个全新的思路向 “ PaaS ” 不断收敛。只不过这一次, PaaS 可能会换一个新的名字叫做：Serverless。

我们还能够看到，云的边界正在被技术和开源迅速的抹平。越来越多的软件和框架从设计上就不再会跟某云产生直接绑定。毕竟，你没办法抚平用户对商业竞争担忧和焦虑，也不可能阻止越来越多的用户把 Kubernetes 部署在全世界的所有云上和数据中心里。我们常常把云比作“ 水、电、煤 ”，并劝诫开发者不应该关心“ 发电 ”和“ 烧煤 ” 的事情。但实际上，开发者不仅不关心这些事情，他们恐怕连 “ 水、电、煤 ” 是哪来的都不想知道。在未来的云的世界里，开发者完全无差别的交付自己的应用到世界任何一个地方，很有可能会像现在我们会把电脑插头插在房间里任何一个插孔里那样自然。

这也是为什么，我们看到越来越多的开发者在讨论 “ 云原生 ” 。

我们无法预见未来，但代码与技术演进的正在告诉我们这样一个事实：未来的软件一定是生长于云上的。这将会是 “ 软件交付 ” 这个本质问题的不断自我革命的终极走向，也是 “ 云原生 ” 理念的最核心假设。而所谓 “ 云原生 ”，实际上就是在定义一条能够让应用最大程度利用云的能力、发挥云的价值的最佳路径。在这条路径上，脱离了“ 应用 ”这个载体，“ 云原生 ”就无从谈起；容器技术，则是将这个理念落地、将软件交付的革命持续进行下去的重要手段之一。

而至于 Kubernetes 项目，它的确是整个 “ 云原生 ” 理念落地的核心与关键所在。但更为重要的是，在这次关于 “ 软件 ” 的技术革命中，Kubernetes 并不需要尝试在 PaaS 领域里分到一杯羹：它会成为连通 “ 云 ” 与 “ 应用 ” 的高速公路，以标准、高效的方式将 “ 应用 ” 快速交付到世界上任何一个位置。而这里的交付目的地，既可以是最终用户，也可以是 PaaS/Serverless 从而催生出更加多样化的应用托管生态。

“ 云 ” 的价值，一定会回归到应用本身。

Geekbang. 极客邦科技 | TGO 鲲鹏會

TGO 鲲鹏会是极客邦科技旗下高端科技领导者聚集和交流的组织，以 CTO、CPO、COO、技术 VP 等科技领导者为服务对象，采用实名付费会员制，严格审核会员资格，旨在组建全球最具影响力的科技领导者社交网络，线上线下相结合，联结杰出的科技领导者学习和成长。

12 个城市成立分会

会员超 850 人

使命 Mission

为社会输送更多优秀的科技领导者

愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

2019极客邦科技会议推荐

5

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12 （月份）

ArchSummit 深圳
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

Geekbang. 极客邦科技 InfoQ

查看会议详情