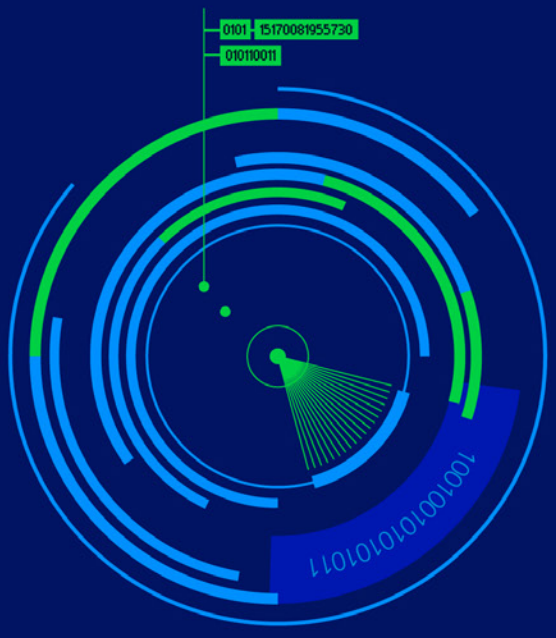
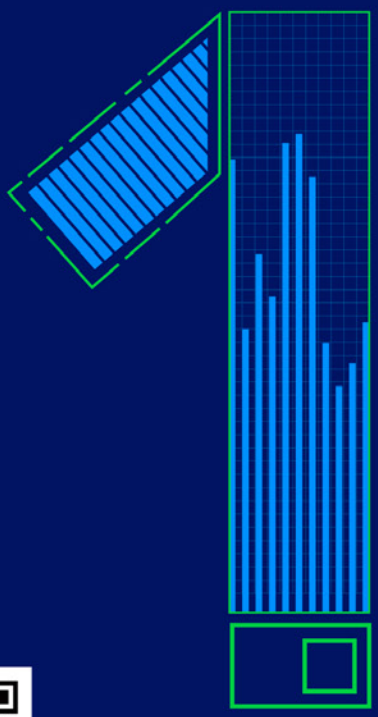
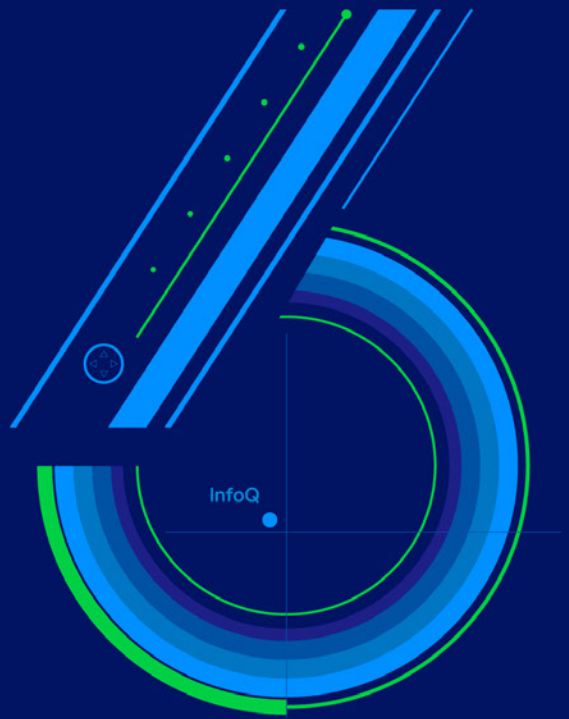
解读

REVIEW 2016

DAT/
2017.01







<扫码关注InfoQ公众号>

Geekbang. 极客邦科技 | InfoQ

卷首语

许多年后，如果我们回过头来评点，也许 2016 年是非常重要的一个时间节点。

2016 年的互联网，人工智能大放异彩，年初 AlphaGo 4:1 战胜李世石，年底 60 连胜横扫网络围棋，沉寂了数十年的人工智能再次走上前台，颠覆了人们的既有观念。

同时，随着 Google DayDream、微软 HoloLens 等产品的发布，以及阿里 Buy+、PokemonGo 等应用的出现，VR/AR 技术离我们越来越近，虽然目前体验仍有改进空间，但没人可以否认那是最接近我们幻想中未来交互的样子。

这些前沿领域的技术人是幸运的，他们的努力，将会推动时代前进的脚步。

还有一些领域，虽然变化没那么引人注目，但也都各自在发生着深刻的变化。

在容器领域，Docker 掀起了编排之争，同时集群管理越来越受重视，Kubernetes 受到重视。

开始逐渐普及。与此同时，容器还成为催化剂，催动着很多领域发生变化。

在大数据领域，越来越多的公司开始搭建大数据平台，数据资产管理和流数据处理受到重视，同时，大数据与人工智能的结合是 2016 年最受瞩目的技术之一，并且在一些企业得到成功应用。

在运维领域，容器技术为运维带来了全新的挑战，DevOps 开始落地，同时各种云架构让运维情况更加复杂，Google SRE 的实践则为运维确定了全新的标准。

在移动领域，2016 年可谓是国内移动技术的爆发之年，插件化、热修复、组件化等技术的开源和讨论丰富了移动开发的技术栈，React Native、Weex、微信小程序的强势来袭，让移动开发技术面临变革。

更别提火爆的前端，对于这个一年一更新的技术领域，2016 年发生了很多有意思的事件，React/Angular/Vue 三大框架开始进入三国鼎立时代，创新层出不穷。

变革对于技术人来说并不是坏事，它意味着永远有新的机会。时代交替，会淘汰的只有不思进取的人，而合格的技术人，永远不会停下学习的脚步。

在这新春之际，我们为所有技术人献上过去一年各个领域的盘点和展望，愿你的学习之路走得更加平坦。

InfoQ 中文站编辑 徐川

目录

05 解读 2016 之容器篇：“已死”和“永生”

20 解读 2016 之深度学习篇：开源深度学习框架发展展望

30 解读 2016 之大数据篇：跨越巅峰，迈向成熟

40 2016 移动开发技术巡礼

59 2016 前端开发技术巡礼

79 解读 2016，眺望 2017：运维的风口在哪？

解读 2016 之容器篇：“已死”和“永生”

张磊

也说不上什么时候起，“XXX Is Dead. Long Live XXX”的句式突然成为了技术会议上演讲题目的一个标准套路。然而不管已经被引用的多么烂俗，用这套悖论来总结 2016 年容器技术圈子发生的凡事种种，却实在有种说不出来的恰到好处。

无需多言，稍微回顾一下 2016 年容器技术圈子的时间线，我们很容易就能回想起容器技术如何在这一年迅速登上云计算舞台的中心。这股热潮，从年初 Docker 公司闪电收购 Unikernel Systems 提前扼杀各种“被颠覆”的苗头，蔓延到 Kubernetes，Mesos，Docker 三家项目在年中掀起的“编排”之争，再到年末阿里云一举震撼国内创业市场。“编排”，“fork docker”，“OCI runtime”，“镜像标准”，一个又一个令人目不暇接

的关键词带着背后的技术爆点填满了 2016 一整年的时间线。只不过，惊喜不断的同时，嘈杂的容器圈子也难免给我们带来了些许无所适从的挫败感。回顾这一年的容器技术发展历程，相信大家都有这样的疑惑：当这个圈子平复下来之后，我们该如何去理性地思考和解读呢？

Docker

在 2016 年，当我们再次说出这个关键词的时候，已经很难用一句话解释清楚我们到底在说的是什么。是 Docker 公司？是 Docker 容器？是 Docker 镜像？还是 Docker 集群？

在创业初期通过一连串极其成功的开源战略迅速蹿红之后，Docker 项目几经重构，最终选择了“大一统”的战略模式，一系列普通认知中应该是独立项目的功能模块都被编译进了 Docker 项目的二进制文件中，这其中最引人注目的，当属 SwarmKit 项目。

2016 年 6 月，Docker 公司宣布将在接下来版本的 Docker 项目中将会提供内置的“编排”功能，这个功能的实现将主要由一个名叫“SwarmKit”的依赖库来负责。此新闻一出立刻成为容器圈子一时的舆论热点，尤其在国内。其中，看涨者不少，有言“生态闭环”、“Docker 正统”，唱衰者也不缺，直呼“公然越界”、“野心昭然”。时至今日，该项目本身也日趋稳定，我们不妨再回头来重新解读一下曾经在风口浪尖上的 SwarmKit。

SwarmKit 的核心功能乃是“编排”，不过它对这个编排的定义还是比较模糊的，在初期主要指的是“多容器副本”和“副本负载均衡”两个核心能力，后面逐渐加入的是更多应用管理功能。说起这个项目发布的初衷，当时国内的诸多讨论之中，曾有一种误区是认为 SwarmKit 是 Docker

Swarm 项目的继承、是 Swarm 项目的“内置版”（当然，这也要部分归功于 Docker 公司老辣的命名技巧）。但现在回头来看，这些“编排”能力在 Docker Swarm 项目中，一直都是不存在的，继承自然无从谈起。SwarmKit 唯一跟 Docker Swarm 项目重叠的功能乃是“调度”，但实际上 SwarmKit 的调度也是从头做起，它维护了一个 NodeHeap（堆），然后通过堆算法配合过滤条件来筛选最符合要求的节点来运行任务。这套调度机制在 Swarm 中也是不存在的。而在 API 层面，Docker Swarm 项目提供的是一套简洁的单容器的 API 来让用户操作容器集群（这个能力非常受欢迎），而 SwarmKit 却从一开始就引入了 Service，Task 等一系列面向容器集群的、平台级别的概念，并且从底层实现上就不兼容 Swarm 风格的单容器 API。两者的关系正如同“Java”和“JavaScript”一样风马牛不相及。

那么 Docker 公司内置“编排”能力并且起一个这样的有混淆意味的名字，到底意义何在呢？

答案很简单：“平台（Platform）”。或者说成是“应用 / 容器集群管理”，或者说成是“PaaS”甚至“CaaS”，都可以，一个意思。

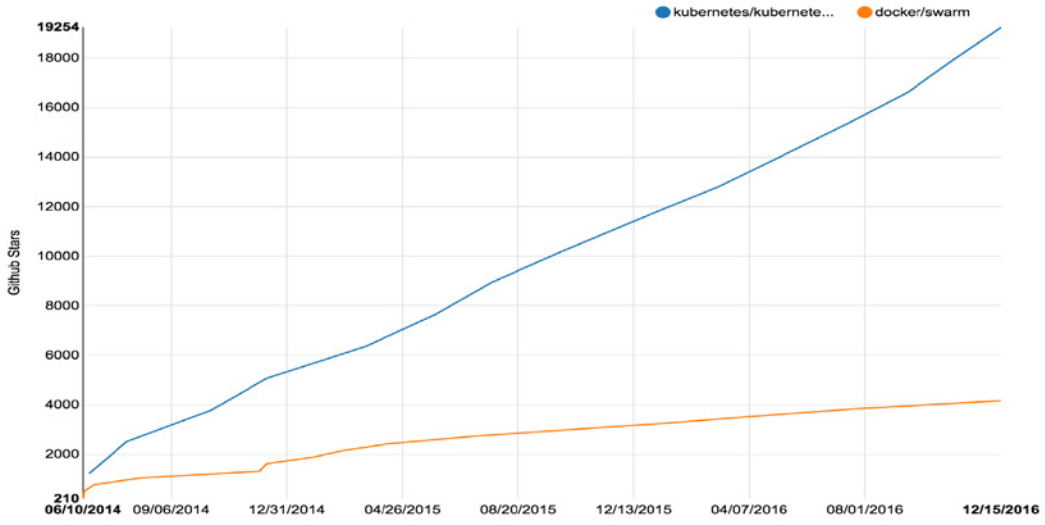
这个改变的关键就在于，从今以后 Docker 项目就变成了一个“平台”项目而不再是一个单纯的“容器”项目了。它要站在 Kubernetes，DC/OS，Cloud Foundry 一样的位置上直面云的终端用户，而不是继续做这些平台背后的容器技术（甚至只是容器技术中的一种）。

这种平台级别的能力对于 Docker 公司来说是至关重要。容器的热度终究会冷却，用户很快就不会关心底层的容器技术为何物，他们只会记得 Kubernetes API，Service， Replication Controller，DC/OS，顶多在编写 Dockerfile 的时候，才回忆起 Docker 公司的存在。很多人批评 Docker 公司野心太大，其实对于一家拒绝了微软 40 亿美金收购的后端技

术创业公司来说，有怎样的进取心都不为过。Docker 公司的目标是下一个 VMware，下一个 Intel，一个实实在在能盈利能上市的商业公司，在这个巨头如林的云计算行业里，这是令人钦佩的。

Docker 公司在 2016 年在集群领域所做的努力离不开“平台”二字，不过国内曾经一度涌现出来的过分解读着实让这个项目背负了太多的压力。其实打开 SwarmKit 项目的代码看看，从编排到调度，模块设计，代码实现，跟其他项目并无二异，ipvs 也是普通的 NAT 模式，Master 节点一样维护着 Apiserver, Scheduler, Orchestrator，同宿主机上的 Agent 通过 gRPC 交互，就连 Agent 也跟其他“第三方”项目一样也要通过 client 去调用 Docker Engine 的 API。所谓的各种“颠覆性”、“大道至简”、“容器 OS 化”、“从下往上改变容器云形态”等观点，实在无从谈起。

还有一种可能性是 Docker 公司依靠 Docker Swarm 这样一个独立的、以单容器操作 API 为核心的项目来完成 PaaS 的使命（业界基于 Swarm 构建 PaaS 的用户不在少数）。但现实是，几乎同时发布的 Google Kubernetes 项目却出人意料地狙击了 Swarm 的发展势头。下图是自发布起，Swarm 项目和 Kubernetes 项目 GitHub Star 数目的变化统计。



事实上，Docker Swarm 项目“使用单容器 API 来操作集群”的理念是相当有吸引力的。但是这个能力在接下来 Docker 公司想要重点发展的企业私有云市场却陷入了窘境：单容器 API 纵然简单友好，但企业级用户却没办法直接用它来实现哪怕一个最简单的“容器集群负载均衡”的需求。所以很多企业私有云用户更倾向于把 Docker Swarm 项目作为容器云的一个环节，然后自己来实现各种平台级别功能（往往还要参考 Kubernetes 的各种理念和设计）。照这个趋势发展，如果不推翻重来提供类似的面向集群的 API，Docker 公司的平台之梦恐怕就很难实现了。这也正是为什么我们前面简单一对比就不难发现，SwarmKit 相比 Swarm 项目其实是翻天覆地的自我革命，而非继承或者内置，所谓向后兼容的问题自然也无从谈起。

其实，很多人可能已经忘记了早在 Docker Swarm 项目发布之前，Docker 公司已经发布过一个集群管理的项目叫“libswarm”（访问该项目地址有彩蛋：https://github.com/docker/libswarm）。这个项目的初衷非常单纯，就如同 Docker 项目最开始发布时一样“冰雪通透”，“libswarm 项目的初衷是提供一套不依赖与现有分布式系统的集群管理 API，并使得其他项目可以使用它来方便的构建容器集群……”

彼时的 Docker 公司，还希望 Mesos，Fleet 们使用 libswarm 来管理 Docker 容器云呢。

所谓“Swarm 已死，Swarm 永生”，Docker 项目又何尝不是呢。

在经历了 OCI 成立、贡献出了核心组件 libcontainer 之后，Docker 公司坚定地走向了独立发展的道路。在巨头们的围剿之中，这是一家创业公司从默默无闻到炙手可热，再到痛定思痛之后的必然选择，SwarmKit，以及其他所有外界看来“野心太大”的项目和举措，都只是这个信念的间

接产物而已。在未来，Docker 公司依然会不遗余力的构建自己的平台世界，网络，存储，Infra Layer，CI/CD，一个全功能平台级项目所欠缺的版块都会被一一补全，各种各样内置于 Docker Daemon 中的 Kit 库和收购还会层出不穷，Docker 公司还会以此为基础重点推广可以盈利的 Docker Cloud 和 Docker Datacenter。这样的选择很正确，并且唯一。

另一方面，在补全平台级别功能的过程中，Docker 项目会依然选择将这些管理组件跟原先的 Docker Daemon 耦合在一起。从技术角度来看，这并不是个明智之选，过高的耦合度所带来的不稳定性、Data Race 和维护的问题会愈加凸显。但从推广的角度来看，这个做法非常厉害：Docker 项目需要努力争取现有用户和粉丝的青睐，引导他们放弃单容器 API，转而接纳新的、平台级别的 API。这个转变是站上“Platform”这个舞台在所难免的，也是从 Swarm 项目上所得来的经验教训。

2016 年末，Docker 项目将容器运行时相关的最后一个组件 containerd 也正式剥离，“Docker”这个名字离“容器”渐行渐远，离“PaaS”越来越近。可能有人会疑惑：像 Kubernetes，DC/OS，或者未来的 Docker 项目，它们跟 PaaS 不是还有所不同吗？实际上，在这股正是由 Docker 掀起的容器浪潮下，PaaS 的定义恐怕早已悄然变化。

正所谓“PaaS 已死，PaaS 永生”。

在容器集群管理和企业级需求的支持上，Docker 公司还是个新生儿，但 Docker 项目成功的哲学乃是“simple but powerful”，它一直坚持提供尽量简单的命令行界面，并不惜为此选择更复杂的实现方式，这将是它在未来会继续火热的杀手锏之一：对于任何希望快速寻求一个“可用”的容器工具的开发者和运维者来说，这个吸引力是巨大的。

Kubernetes

尽管 Docker 公司整整一年都在“平台”领域发力，Kubernetes 依然是这个领域最瞩目的项目。这并非意外，在开源的世界里，一旦在某个领域树立了标杆，就很容易跟竞争对手拉开质的差距。Kubernetes 幸运的成为了“容器集群管理“领域的开创者，其他的后进项目，无论是 Marathon 还是 SwarmKit，都只能主动或者被动地 follow 开创者的提出的理念。这正如同如果让 Google 再做一个容器，它也会十有八九 follow Docker 一样。“如果大家只是再造一个 Kubernetes，那有什么理由会比 Kubernetes 团队做的更好？”

当然，含着“千呼万唤始出来”的 Borg 论文出生，又是 Google 公司在 Big Data 领域错失机会之后着重推出的平台级开源项目，Kubernetes 本身自然有其过人之处。

Kubernetes 的发布给整个容器圈子带来了一系列前所未闻的概念：Service，Replication Controller，Pod，Labels & Label Selector，DaemonSet，Cron Job 等等等等。当时，不少用户还在评论 Kubernetes 的理念太超前了。而现在回头来看，这些特性不仅被用户所接受，而且很多还被其他平台项目比如 Docker、Marathon、DC/OS 所采纳，变成了它们的内置功能：正如同做容器不支持 Docker 就显得“落伍”一样，搞平台不谈“Service”、”Replica“，你的编排就不够 fancy。

Kubernetes 这种相对超前的技术视野与它背后原 Google Borg 和 Omega 团队成员的经验和努力密关系巨大。Borg/Omega 系统在 Google 基础设施体系中的声誉和地位无需多言，而用户在 Kubernetes 中接触到的很多概念，其实在 Borg/Omega 系统中都有等价的特性。从这个角度来说，

把 Kubernetes 项目描述为 Borg 系统在开源领域的重生并不为过。

在这一年里，Kubernetes 不断地强化自己在容器集群管理领域的优势，密集发布了一系列让人称道的设计。不难预料，这些概念很快也会在其他项目中被借鉴和采纳。

就比如 Secret，它允许用户将加密过的 Credentials 信息保存在 Etcd 中，然后在容器中通过环境变量或者挂载文件的方式访问它们，从而避免明文密码被随意写在环境变量、配置文件或者 Dockerfile 中的问题。

类似的还有 ConfigMap，只不过保存在 Etcd 的内容，是应用所需的配置信息。

再比如 Deployment，它使得用户可以直接编辑容器化任务的属性，然后直接触发 Rolling Updae，还允许用户随意回滚任务到以前的版本。

再比如 DaementSet，它允许用户一键部署运行在所有节点上的守护进程任务。

而最近推出的 StatefulSet，则提供了原生支持有状态的应用的强大能力，并且既包括了拓扑结构状态，也包括了存储状态。

还有备受欢迎的 ScheduledJob，它允许用户用标准的 Cron Job 格式来定义从镜像启动的定时任务，并保证这个任务执行的正确性和唯一性。

如此种种，都是 Kubernetes 在容器编排和管理领域树立标杆的手段，而这些设计背后的思想又十分朴素：如果在 Borg 里，或者在没有容器的传统环境下，我们能够通过脚本或者其他手段自动化地做某些事情，那么 Kubernetes 同样应该帮你做到。这个过程，正是 Kubernetes 所定义的（也是我们传统运维意义上的）“编排”，也是 DevOps 理念中所追求的“NoOPs”的主要手段。

在 2016 年，Kubernetes 另一个重点发力的领域则是 CRI（Container Runtime Interface）。值得一提的是，CRI 的提出者和推广者正是 2015 年 InfoQ CNUT 容器大会讲师、华人女工程师 Dawn Chen，她也是 Kubernetes 的几位元老级（Elder）成员之一。早在 Docker 公司在集群领域发力之前，Dawn 所领导的 Node Team 就已经颇有远见地开始制定解耦容器运行时的方案并持续在社区推动。通过制定这样固定的访问接口，Kubernetes 不再对任何容器 API 产生多余的依赖，也避免了锁定在某种容器运行时之上的尴尬。目前，Kubernetes CRI 主要由来自 Google 公司、Hyper 国内团队、以及 CoreOS 的工程师负责维护。

得益于 CRI 的逐渐成熟，Kubernetes 项目在容器运行时领域的支持能力得到了巨大的提升，除了默认的 Docker 容器之外，基于虚拟化技术的 HyperContainer，CoreOS 公司的 rkt 都已经能够通过 CRI 原生接入了 Kubernetes。而 runC 团队也不甘寂寞，来自 SUSE 和 RedHat 的 runC Maintainer 提交了了一个叫做 CRI-O 的孵化项目用来直接将 runC 接到 Kubernetes 中。此外，来自微软的 Windows 容器也已经进入 Kubernetes 体系。尽管还未完全 release，Kubernetes CRI 的受欢迎程度已经略见一斑。

在接下来的进展中，Kubernetes 会继续加强自己的已有优势，更多新的容器编排和管理能力会接踵而至，其中有状态应用的支持、更加完善的网络规则、GPU 和 NUMA 的支持、批处理和 Big Data 任务支持都是值得关注的特性。值得注意的是，Kubernetes 在定义容器编排和容器管理方面有一个先天的优势：由于 OpenShift（Redhat 基于 Kubernetes 的 PaaS）的存在，Redhat 团队一直在非常谨慎地确保 Kubernetes 不会功能泛滥。这正是 Kubernetes 保证自己所提供的平台能力恰到好处的一个重

要手段。

与此同时，在捐献给 Linux Foundation 之后，Kubernetes 所属的 CNCF 基金会已经构建出了一个围绕 Kubernetes 的平台生态系统，CNCF 的成员项目不仅包括了 Prometheus 这样的明星，还包括了 gRPC 这样的底层依赖。这些项目之间会尽力兼容 API 和数据格式，减轻用户构建云平台的负担。一个最典型的例子，就是 Kubernetes 可以直接使用来自 Prometheus 的监控数据来做容器自动扩展，无需做任何格式转换和数据过滤。而 Kubernetes 正在全面基于 gRPC 进行的全通路上的性能优化（包括 Etcd v3）也得益于该社区的有力支持。日前，国内的 PingCAP 也得到了 CNCF 的大力赞助，有望在存储状态管理和基于本地磁盘的持久化卷方面为社区注入新的血液。

未来的时间里，Kubernetes 面对的挑战依然严峻，其中最大的问题在于作为一个试图真正解决问题工业级项目，如何能保证自己功能足够完善的同时，始终给用户提供简单和友好的交互体验。毕竟在未来一段时间内，争取用户依然是所有玩家的重中之重。我们已经看到 Kubernetes 正在对 UI/UX 进行优化，最典型的例子是 kubeadm 工具，一举解决了之前 Kubernetes 部署不方便的顽疾，用户终于可以用 kubeadm init 和 kubeadm join 这样的指令来启动完整的集群了。

在 2016 年的 Kubernetes 生态中，还有一个不能被忽视的因素就是自家兄弟 Tensorflow 的迅速蹿红（甚至短时间内就超越了 Docker 项目，速度令人咋舌），并且直接推动了人工智能元年的诞生。在社区层面 Tensorflow 目前仍无实质性的竞争对手，无论是大小公司还是机构，基于该项目的人工智能项目如雨后春笋般涌现，创业公司也层出不穷，而 Kubernetes+Tensorflow 的搭配则成为了默认的“基础设施 + 深度学习平

台”的组合。这个机遇恐怕没人能事先预料到。

回想当初 Kubernetes 刚发布时的，不少人还对没有拿到一个真正的开源版的 Borg 表示失望。两年后的今天，再回顾这个项目的发展历程，我们不得不说 Kubernetes 已经走出了一条独立的、健康的发展路线。在这条以开源容器技术兴起为源头的道路上，越来越少人会再去谈论 Borg，去做无谓的比较。Kubernetes 项目正继承着这些优秀的思想，开启了一个关注容器和应用本身、专注编排和集群管理的新领域。

在这个前所未有的、容器为王的世界里：“Borg 已死，Borg 永生”。

Mesos 生态

当今容器圈子，除了以 Docker 为主角的容器运行时和以 Kubernetes 为主角的容器集群管理，还有一方“势力”绝不能被忽视，这就是 Mesos 生态。众所周知，Mesos 本身是一个“老”项目，它诞生于伯克利，崛起于 Big Data 的爆发。在 Docker 刚开始成名的头两年，Swarm 项目羽翼未丰，Mesos 独有的支持多种 Framework 的设计使它得以轻松接入了 Docker 生态，并在当时成为了管理 Docker 容器集群的不二之选，占满了 DockerCon 的重要位置。毕竟天生就具备管理万级别节点规模的水准，上层的 Marathon 框架又能提供一套完善的 PaaS 功能，还有喜人的 UI，还能提供生产级别 Big Data 业务的支撑，Mesos 没有理由不受欢迎。不过紧接着，Marathon+Mesos 的组合开始显现出疲态，在 Docker 和 Kubernetes 各自亮出杀手锏不断刷新用户三观的时候，Apache 社区固有的响应迟钝拖慢了 Mesos 进化的速度，而由创业公司维护的 Marathon 又一直没办法构建出更加繁荣和活跃的社区。

“社区”两个字，竟成了 Mesos 生态的命门。在 Docker 公司煞费苦心在

社区争取每一个用户和粉丝、Google 公司放下身段把 Github 作为一线阵地，用 Kubernetes 全力输出技术理念的时候，一旦错失了先机，哪怕有一身本事如 Mesos 项目，也只能望用户而心叹。这正是目前 Mesos 生态系统在容器圈子表现的不够强势的重要原因。当然，既然实力强劲，Mesos 生态在工业界中的案例还是数不甚数，除了 Marathon 框架，Mesosphere 公司重点维护的 DC/OS 项目其实能够提供并不逊于 Kubernetes 的各项容器编排和管理能力，不可谓不强大。但是社区表现不力，使得 Mesos 生态错失了成为容器圈的“buzz word”（热词）的机会。SwarmKit 发布后，Mesos 生态同 Docker 项目的蜜月期也宣告结束，随后的 Mesos 1.0 版本也正式 release 了 Unified Containerizer 并通过默认的 MesosContainerizer 取代 Docker Daemon，且原生支持多种镜像格式，这个改变比 Kubernetes CRI 要彻底的多。与此同时，CNI 网络（Kubernetes 社区采用的网络插件标准）也已经在 Mesos 项目上得以支持。

在未来，Mesos 生态会重点完善自己在 15-16 年反应不及时所欠下的功能缺口，包括网络、存储、多租户、甚至 Pod 支持等重要的功能模块都已经有方案提交到了 Roadmap。别忘了，Mesos 生态（包括 DC/OS）是目前（包括未来一段时间）容器圈子唯一支持大数据任务的基础设施依赖，是唯一有生产级别超大规模集群管理能力的资源管理框架，也是唯一原生提供界面的应用管理平台。这三个“唯一”在很多用户心目中（尤其是传统企业里），占有十足的分量。

曾经以 Marathon+Mesos 为代表的 Mesos 社区已经逐渐淡去，但围绕着 DC/OS 的新的 Mesos 生态终将亮剑，正所谓：“Mesos 已死，Mesos 永生”

国内外创业生态圈

如果说 2016 年的容器圈子仍然十分热闹，那必然少不了 startup 的繁荣。围绕容器运行时、编排、网络、存储、镜像管理、CI/CD、PaaS 方

案等的一系列生态环节所创立的 startup，是推动这个大环境蓬勃发展的直接动力。

除了本身就是创业公司的 Docker，容器圈的另一个主角当然是 CoreOS。虽然创新能力十足，但 CoreOS 公司的 rkt 容器仍然没能在 Docker 的强势下占据容器市场的主流。由于单点突破受阻，2016 年的 CoreOS 公司也做出了跟 Docker 公司类似的转型，开始向一个涵盖范围更广的“容器云”公司靠拢。除了一系列改名、扩展 rkt 的职能之外，还有一个重要的手段是：抛弃 Fleet，拥抱 Kubernetes。

Fleet 曾经是 CoreOS 体系中重要的容器编排和调度项目，也曾同 Mesos 等项目一样在容器云领域占有一席之地。但现实证明，它并不足以让 CoreOS 在“云”的市场上争取到竞争优势。凭借 Etcd 在 Kubernetes 中的重要作用，CoreOS 的工程师从性能调优作切入口进入了 Kubernetes 生态，并做出了显著的成绩，rkt 也在 CRI 发布之前就成为了 Kubernetes 的可选容器运行时之一，kubelet 则被内置到 CoreOS Linux 中作为默认的编排和调度框架。2016 年，CoreOS 又围绕 Kubernetes 发布了一系列工具如 Operator 来完善生态中的“有状态应用管理”，“存储管理”等能力，已经可以说是最成功的结合 Kubernetes 来创业的 startup。

与 CoreOS 不同，国内的创业公司的发力点主要集中于提供容器服务的 PaaS 产品（也有人称之为 CaaS 以突出自己的容器特性）。相比于创业初期主要集中于容器管理平台的建设，2016 年的国内容器创业公司则主要在围绕自己的平台构建生态类产品，涵盖了监控、存储、镜像监管、客服等一系列工具，其产品能力明显强于同类型的国外 startup。另一个显著的变化是，2016 年国内创业公司开始更多关注和宣传线下企业私有云市场的生意，创业初期着重推广的公有云服务的更新和维护力度明显降低。

毕竟，在国内创业公有云盈利困难是一个不争的事实。

产品能力的异常强劲，侧面反衬出了国内创业公司在上游社区层面的影响力的弱势，在社区中推动和提出项目特性的能力依然欠缺。当然，创业维艰，尤其是国内大环境下，恐怕也只有华为能够在一边维护和推进 runC 等 OCI 项目的同时，一边在 Kubernetes 上开展完整的“联邦集群”特性（甚至将 Google 的负责人招致麾下）。还有 Hyper，这只团队已经是 Kubernetes CRI 的重要维护者之一（CRI 中很大部分代码正出自他们之手），同时也是 runC 运行时 cri-o 项目的维护者，已经在 Kubernetes 容器管理部分争取到了一席之地。而其本身维护的虚拟化容器 HyperContainer 项目和以此为基础的 hyper.sh 容器托管服务，则创下了占据 HackerNews 首页长达 24 小时的惊人记录。不过放眼全球市场，这些工作只能说是 Hyper 的本职，并不能掩盖国内团体在整个容器开源社区里弱势的现状。而这个现状的改变，以目前国内大多数 startup 的运营方式和核心能力点来看，恐怕还尚需时日。

2016 年国内另一件新闻则是阿里云同 Docker 公司的牵手。这并非临时起意，早在 2015 年在国内容器创业氛围正值巅峰时，阿里云并没有直接进场，“而是在谨慎考察国内环境对容器的接纳程度”（此信息来自阿里云官方）。时至 2016 年，容器技术已经在国内红透半边天，作为后来者，体量又如阿里云这般的巨头要想再进场，必须要拿出最强势的姿态来踏出这第一步。同 Docker 公司展开官方合作，可以说是最佳选择。而对于 Docker 公司来说，Google 和 AWS 已经成为直接竞争对手，放眼全球，阿里云即是拿得出手的一线厂商，又对 Docker 公司无毒无害，这次合作可谓一拍即合。只是从此国内其他公司再想拿“Docker 官方”、“Docker Native”来做宣传，在这个排他性的合作面前，恐怕就需三思而后行了。

总结

2016 年，容器技术圈子依然热闹非凡，容器社区里的弄潮儿们在“is dead”和“long live”的悖论里不断地自我否定和进化已成为这个社区所独有的常态。Docker 公司“萌萌哒”的鲸鱼背后，一只野心勃勃的海洋霸主正蓄势待发；而作为容器编排管理领域的领导者，纵使有 Google、Redhat 等巨头的撑腰，Kubernetes 恐怕也不会放慢脚步，通过 CRI 联合 CNCF、OCI 两大开源社区举措也在情理之中。容器技术的圈子里容不得懈怠，Mesos 生态已经开始励精图治，意在凭借独有的能力拿回昔日的地位。我们不难发现，在 Docker 公司向平台领域发展的同时，Kubernetes、Mesos 也同样渗透进了容器运行时的范畴。实际上，正如同那些支付宝与微信之争，几个大佬原生的核心能力恐怕才是它们取得今天成绩的关键所在。作为终端用户，理清自身真实需求之后，做出适合自己的选择其实并不太难。

容器技术圈子的繁荣，得益于现代开源软件社区的成功。Docker 自不必说，Kubernetes、Tensorflow 亦如是。连 Google、Microsoft 这样的巨头都一改对开发者傲慢的态度和轻视开源社区运营的作风而扎堆到 GitHub 上施展浑身解数，有幸同处一个时代而成为参与者和亲历者的我们又有何理由作壁上观呢。

作者介绍

张磊，浙江大学计算机博士生，微软云计算与数据中心管理领域最有价值专家，HyperHQ 项目成员、CNCF Kubernetes 子项目成员，曾策划并出版《Docker 容器与容器云》一书。

解读 2016 之深度学习篇：开源深度学习框架发展展望

周志湖，赵永标

引言

深度学习（Deep Learning）的概念由加拿大多伦多大学教授 Geoffrey Hinton 等人于 2006 年提出，它本质上是一种神经网络算法，其原理是通过模拟人脑进行分析学习，算法训练时可以不用人工干预，因此它也属于一种无监督式机器学习算法。从深度学习的概念提出到今天，已历经十年时间，在这十年当中，无论是国际互联网巨头 Google、微软、百度，还是全世界各大学术研究机构，都投入了巨大的人力、财力用于理论和工业级应用的探索，并在字符识别（OCR）、图像分类、语音识别、无人自动驾驭、自然语言处理等众多领域内取得了突破性的进展，极大地推动了人工智能（AI）的发展。

图 1. 深度学习的典型应用场景示例

2012 年斯坦福大学的 Andrew Ng 和 Jeff Dean 教授共同主导的 Google Brain 项目通过使用深度学习让系统能够自动学习并识别猫，这项目研究引起了学术界和工业界极大的轰动，激起了全世界范围研究深度学习的热潮，《纽约时报》披露了 Google Brain 项目，让普通大众对深度学习有了最初的认知。2016 年 3 月，Google 旗下 DeepMind 公司开发的 AlphaGo 以 4:1 的总比分战胜了世界围棋冠军、职业九段选手李世石，这让人们对人工智能（AI）的认知跨越到一个新的阶段。

深度学习在众多领域的成功应用，离不开学术界及工业界对该技术的开放态度，在深度学习发展初始，相关代码就被开源，使得深度学习“飞入寻常百姓家”，大大降低了学术界研究及工业界使用的门槛。本文并不打算对深度学习算法的发展趋势进行分析，而是跳出算法层面，对当前主流的开源深度学习框架的发展趋势进行分析。

开源深度学习框架

在计算机视觉领域内，神经网络算法一直被用来解决图像分类识别等问题，学术界采用的算法研究工具通常是 Matlab、Python，有很多深度

学习工具包都是基于 Matlab、Python 语言，在使用海量图像训练深度学习算法模型时，由于单台机器 CPU 的计算能力非常有限，通常会使用专门的图形处理器（Graphics Processing Unit，GPU）来提高算法模型速度。但在工业级应用当中，由于 Matlab、Python、R 语言的性能问题，大部分算法都会使用 C++ 语言实现，而且随着深度学习在自然语言处理、语音识别等领域的广泛应用，专用的 GPU 也慢慢演变为通用图像处理器（General Purpose GPU，GPGPU），处理器不仅仅被用于渲染处理图像，还可以用于需要处理大数据量的科学计算的场景，这种提高单台机器的处理能力的方式属于纵向扩展。

随着大数据时代的来临，大数据处理技术的日趋成熟为解决深度学习模型训练耗时问题提供了重要发展方向，因此如何通过大数据训练深度学习模型在当前引起了广泛关注，大量的研究表明：增加训练样本数或模型参数的数量，或同时增加训练样本数和模型参数的数量，都能够极大地提升最终分类的准确性。由于 Hadoop 已经成为构建企业级大数据基础设施的事实标准，有许多的分布式深度学习算法框架被构建在 Hadoop 生态体系内，这种通过分布式集群提高处理能力的扩展方式被称为横向扩展。

虽然利用 Spark 等平台训练深度学习算法可以极大地提高训练速度，但近年来，存储设备和网络在性能方面的提升远超 CPU，如图 2 所示。

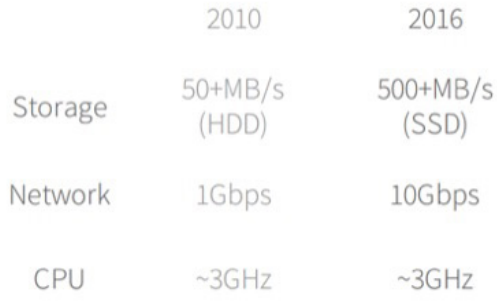
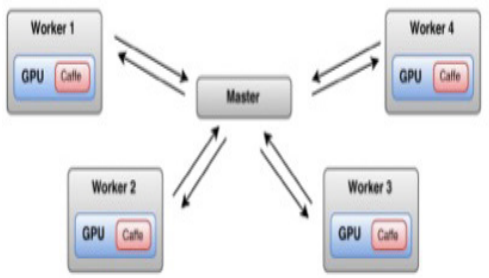


图 2. 存储系统、网络及 CPU 的性能变化比较

CPU 性能瓶颈极大地限制了分布式环境下，单台节点的处理速度。为解决这一问题，许多优秀的开源深度学习框架都在尝试将开源大数据处理技术框架如 Spark 和 GPGPU 结合起来，通过提高集群中每台机器的处理性能来加速深度学习算法模型的训练，图 3 给出的是 SparkNet 的架构原理图。这种通过结合横向扩展及纵向扩展提高处理能力的方式被称为融合扩展。图 4 对深度学习提升性能的方式进行了总结，图中分别给出了横向扩展、纵向扩展及融合扩展的典型开源深度学习框架。



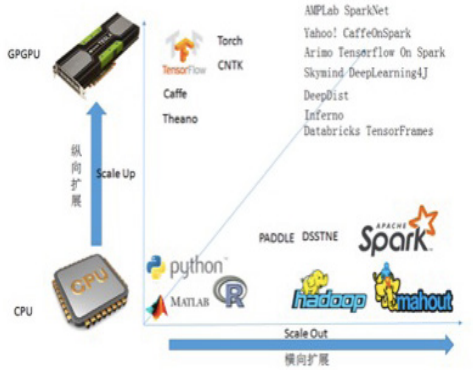


图 3.SparkNet 中的 Scale Up 和 Scale Out　图 4. 深度学习提升性能的方式

下面将对目前一些优秀的开源深度学习框架进行介绍，包括基于 GPU 的单机开源深度学习框架和融合了 GPU 的开源分布式深度学习框架。让大家熟悉目前开源深度学习主流框架的同时，又能够进一步窥探开源分布式深度学习框架的发展方向。

单机开源深度学习框架

目前拥有众多的学术机构如国际顶级名校加州大学伯克利分校，以及互联网巨头如 Google、微软等开源的深度学习工具，比较成熟的基于 GPU 的单机开源深度学习框架有：

Theano：深度学习开源工具的鼻祖，由蒙特利尔理工学院时间开发于 2008 年并将其开源，框架使用 Python 语言开发。有许多在学术界和工业

界有影响力的深度学习框架都构建在 Theano 之上，并逐步形成了自身的生态系统，这其中就包含了著名的 Keras，Lasagne 和 Blocks。

Torch：Facebook 和 Twitter 主推的一款开源深度学习框架，Google 和多个大学研究机构也在使用 Torch。出于性能的考虑，Torch 使用一种比较小众的语言（Lua）作为其开发语言，目前在音频、图像及视频处理方面有着大量的应用。大名鼎鼎的 Alpha Go 便是基于 Torch 开发的，只不过在 Google 开源 TensorFlow 之后，Alpha Go 将迁移到 TensorFlow 上。

TensorFlow：Google 开源的一款深度学习工具，使用 C++ 语言开发，上层提供 Python API。在开源之后，在工业界和学术界引起了极大的震动，因为 TensorFlow 曾经是著名的 Google Brain 计划中的一部分，Google Brain 项目的成功曾经吸引了众多科学家和研究人员往深度学习这个“坑”里面跳，这也是当今深度学习如此繁荣的重要原因，足见 TensorFlow 的影响力。TensorFlow 一直在往分布式方向发展，特别是往 Spark 平台上迁移，这点在后面会有介绍。

Caffe：Caffe 是加州大学伯克利分校视觉与学习中心（Berkeley Vision and Learning Center ，BVLC）贡献出来的一套深度学习工具，使用 C/C++ 开发，上层提供 Python API。Caffe 同样也在走分布式路线，例如著名的 Caffe On Spark 项目。

CNTK：CNTK （Computational Network Toolkit）是微软开源的深度学习工具，现已经更名为 Microsoft Cognitive Toolkit，同样也使用 C++ 语言开发并提供 Python API。目前在计算视觉、金融领域及自然处理领域已经被广泛使用。

DeepLearning4J 官网有篇文章《DL4J vs. Torch vs. Theano vs.

Caffe vs. TensorFlow》，对这些主流的深度学习框架的优劣势进行了详细的分析比较，感兴趣的读者可以点击查看。

分布式开源深度学习框架

Google 研究员 Jeffy Dean 在 2012 发表了一篇《Large Scale Distributed Deep Networks》对分布式环境下的深度学习算法设计原理进行了阐述，给出了深度学习在分布式环境下的两种不同的实现思路：模型并行化（Model parallelism）和数据并行化（Model Parallelism）。模型并行化将训练的模型分割并发送到各 Worker 节点上；数据并行化将数据进行切分，然后将模型复本发送到各 Worker 节点，通过参数服务器（Parameter Server）对训练的参数进行更新。具体原理如图 5 所示。

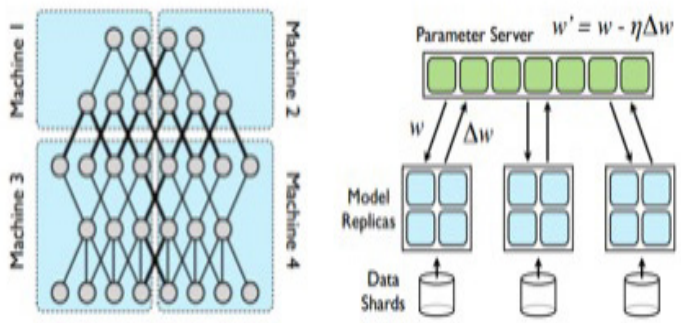


图 5. 深度学习并行化的两种方式：模型并行化（左）和数据并行化（右）

目前开源分布式深度学习框架大多数采用的是数据并行化的方式进行设计。目前有两大类：框架自身具备分布式模型训练能力；构建在 Hadoop 生态体系内，通过分布式文件系统（HDFS）、资源调度系统 (Yarn) 及 Spark 计算平台进行深度学习模型训练。其中框架自身具备分布式模型训练能力的开源深度学习框架有：

- DSSTNE：亚马逊开源的一套深度学习工具，英文全名为Deep Scalable Sparse Tensor Network Engine（DSSTNE），由C++语言

实现，解决稀疏数据场景下的深度学习问题。值得一提的是，亚马逊在是否走开源的道路上一直扭扭捏捏，开源DSSTNE，似乎也是在Google、Facebook等巨头抢占开源深度学习领域高地之后的无奈之举。

- Paddle：百度开源的并行分布式深度学习框架（PArallel Distributed Deep Learning，PADDLE），由C++语言实现并提供Python API。Paddle框架已经经过百度内部多个产品线检验，包括搜索广告中的点击率预估（CTR）、图像分类、光学字符识别（OCR）、搜索排序、计算机病毒检测等。

由于Hadoop生态体系已经占据了企业级大数据市场的大部分份额，因此目前许多开源分布式都在往Hadoop生态体系迁移，这其中有Caffe、TensorFlow，也有百度的Paddle。构建在Hadoop/Spark生态体系下的深度学习框架实现原理图如下：

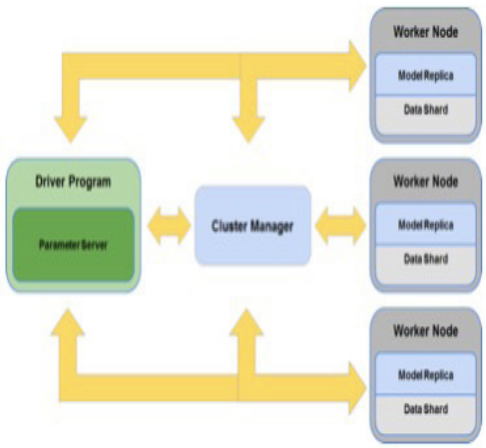


图 6. Hadoop 生态体系分布式深度学习算法实现原理

目前比较有影响力的基于Hadoop/Spark的开源分布式深度学习框架有：

1. SparkNet：由AMPLab于2015年开源，底层封装Caffe和

Tensorflow，采用集中式的参数服务器进行实现，具体实现原理及架构参见论文《SPARKNET: TRAINING DEEP NETWORKS IN SPARK》。

2. Deeplearning4J：由Skymind公司于2014年开发并开源的分布式深度学习项目，采用Java语言实现，同时也支持Scala语言。它使用的参数服务器模式为IterativeReduce。
3. Caffe On Spark：Yahoo于2015年开源的分布式深度学习框架，采用Java语言和Scala语言混合实现，使用Spark+MPI架构以保障性能，其参数服务器采用点对点（Peer-to-Peer）的实现方式。通过将Caffe纳入到Hadoop/Spark生态系统，解决大规模分布式环境下的深度学习问题，意图将Caffe On Spark成为继Spark SQL、Spark ML/MLlib、Spark Graphx及Spark Streaming之后的第五个组件。
4. Tensorflow on Spark：2014年由Arimo公司创建，将TensorFlow移植到Spark平台之上。
5. TensorFrames（TensorFlow on Spark Dataframes）：Databricks开源的分布式深度学习框架，目的是将Google TensorFlow移植到Spark的DataFrames，使用DataFrame的优良特性，未来可能会结合Spark的Structure Streaming对深度学习模型进行实时训练。
6. Inferno：由墨尔本La Trobe大学博士生Matthias Langer开发的基于Spark平台的深度学习框架，作者正在整理发表关于Inferno的实现原理论文，同时也承诺在论文发表后会将代码开源到GitHub上。
7. DeepDist：由Facebook的资深数据科学家Dirk Neumann博士开源的一套基于Spark平台的深度学习框架，用于加速深度信念网络（Deep Belief Networks）模型的训练，其实现方式可以看作是论

文《Large Scale Distributed Deep Networks》中 Downpour随机梯度下降（Stochastic Gradient Descent，SGD）算法的开源实现。

8. Angel：腾讯计划于2017年开源的分布式机器学习计算平台，Angel可以支持Caffe、TensorFlow和Torch等开源分布式学习框架，为各框架提供计算加速。Angel属于腾讯第三代机器学习计算平台，第一代基于Hadoop，只支持离线计算场景，第二代基于Spark/Storm，使用自主研发的Gala调度平台替换YARN，能够同时支持在线分析和实时计算场景，第三代属于腾讯自研的平台，除底层文件系统使用HDFS外，资源调度和计算引擎全部为腾讯自研产品。

SparkNet、Deeplearning4J 及 Caffe On Spark 等构建在 Spark 平台上的深度学习框架在性能、易用性、功能等方面的详细比较，参见《Which Is Deeper Comparison of Deep Learning Frameworks Atop Spark》、《Inferno Scalable Deep Learning on Spark》。由于近几年在大数据技术的日趋成熟和流行，特别是基于 JVM 的语言（主要是 Java 和 Scala）构建的大数据处理框架如 Hadoop、Spark、Kafka 及 Hive 等几乎引领着大数据技术的发展方向，相信以后将会有越来越多的开源深度学习框架构建在 Hadoop 生态体系内，并且基于 Java 或 Scala 语言实现。

总结与展望

本文首先介绍了深度学习提升性能的三种方式：纵向扩展（给机器加 GPU）、横向扩展（通过集群训练模型）及融合扩展（在分布式的基础上，给每个集群中的 Worker 节点加 GPU），然后对主流的开源深度学习框架进行了介绍。通过对这些开源深度学习框架的了解，可以看到当前开源深

度学习框架的发展有以下几个趋势：

- 分布式深度学习框架特别是构建在Hadoop生态体系内的分布式深度学习框架（基于Java或Scala语言实现）会越来越流行，并通过融合扩展的方式加速深度学习算法模型的训练。
- 在分布式深度学习方面，大数据的本质除了常说的4V特性之外，还有一个重要的本质那就是Online，数据随时可更新可应用，而且数据本质上具备天然的流式特征，因此具备实时在线、模型可更新算法的深度学习框架也是未来发展的方向。
- 当待训练的深度学习算法模型参数较多时，Spark与开源分布式内存文件系统Tachyon结合使用是提升性能的有效手段。

深度学习作为 AI 领域的一个重要分支，俨然已经成为 AI 的代名词，人们提起 AI 必定会想到深度学习，相信随着以后大数据和深度学习技术的不断发展，更多现象级的应用将不断刷新人们对 AI 的认知，让我们尽情期待“奇点”临近。

作者简介

周志湖，电子科技大学计算机硕士，研究方向为计算机视觉、机器学习，曾在国内外核心期刊发表机器学习相关注水论文若干，现为绿城中国资深数据平台架构师、专业经理，主要从事房产行业大数据应用。

赵永标，宁波大学应用数学硕士，11 年互联网从业经验，现为杭州以数科技有限公司 CTO、联合创始人，主攻工业、公安及房产行业大数据应用。

解读 2016 之大数据篇：跨越巅峰，迈向成熟

阎志涛

即将过去的 2016 年，大数据技术在持续火热发展的同时，也在各细分领域取得了不同的创新。回顾大数据的 2016，我们都得到了什么？2017 年，会是大数据技术与人工智能融合迸发的时代吗？

大数据管理日趋重要

随着大数据在不同的领域越来越多的应用场景的发现，如何对数据资产进行管理就变得越来越重要。由此也产生了很多的创业公司和开源项目。

WhereHows

WhereHows 是 LinkedIn 在 2016 年开源的一套数据目录发现和数据世系管理的平台。可以当作企业的中心元数据管理系统，对接不同的数据存储和数据处理系统，从而能够全面的管理企业数据目录、数据结构以及数据世系。

Alation

Alation 是一套企业级的数据管理和数据发现的平台，与 WhereHows 不同的是 Alation 并不是一个开源的平台，而是一套商用的平台。除了基础的数据管理、数据发现，这个平台还支持多角色的协作，因为对于数据相关的工作，更好的协作才能提高生产的效率。Alation 公司是成立于 2012 年的一家创业公司，2015 年获得了 900 万美金的 A 轮融资。

大数据应用平台化

随着大数据处理技术的进一步发展，如何整合大数据不同的底层大数据处理技术，将数据集管理、数据加工流水线、数据应用管理融合在一个统一的平台无疑能够大大降低大数据从数据引入到数据变成有价值的产品的复杂度。

CDAP

CDAP 是 CASK 公司开源的大数据应用平台。通过将数据接入、数据管理、数据处理流水线和数据应用开发管理集成在一个统一的平台，CDAP 可以使得企业象开发普通的应用一样开发大数据的应用产品，降低开发的复杂度。如果做一个类比，CDAP 的整体思路类似于在 J2EE 时代的 WebLogic，是一个针对数据应用的中间件平台产品。

StreamSets

StreamSets 是一个侧重数据集成、数据加工流程构建的平台，也是一个开源的产品。通过 StreamSets，用户可以方便的接入不同的数据源，并且完成数据加工流程的构建。SteamSets 有可视化的数据流构建工具，并且能够对运行态的数据应用进行监控。相对于 CDAP，StreamSets 更侧重于数据的接入和数据流的构建、监控和管理。

大数据流式处理成为趋势

在 2016 年，大数据流式处理技术取得了飞速的发展，并且逐渐的变成了大数据处理的新的趋势。在这个大数据流式处理大潮中，几个关键的开源项目逐渐的取得了更多人的注意。

Flink

Apache Flink 并不是一个新的开源项目，但是随着大数据流式处理的日益重要，Flink 因为其对流式处理的支持能力，得到了越来越多的人的重视。在 2016 年，几乎所有的大数据技术大会上，都能够看到 Flink 的身影。

在 Flink 的设计理念中，数据流是一等公民，而批量操作仅仅是流式处理的一种特殊形式。Flink 的开发接口的设计和 Spark 非常的相像，支持 Java,Scala 等编程语言，并且也有支持 SQL 的 Table API，因此有非常好的易用性。另外 Flink 支持将已经存在的 MapReduce 任务直接运行在 Flink 的运行环境上。

同 Spark 一样，Flink 也是期望基于它的核心打造一个大数据的生态系统，它的核心是支持流式的 DataStream API 和支持批量计算的 DataSet API。在上层则是应用层的 API，包括：

CEP

在 Flink 上提供了支持 CEP（复杂事件处理）的库，从而使用者可以非常方便的构造基于 CEP 的应用。

FlinkML

在 Flink 上提供了机器学习算法库，类似于 Spark 的 MLLib。当前的 Flink 1.1 版本的机器学习算法库包含了一些主流的机器学习算法的实

现，比如 SVM,KNN,ALS 等等。

Gelly

Gelly 是在 Flink 上支持图计算的 API 库，类似于 Spark 上的 GraphX。在大数据时代，通过图算法和图分析能够在很多业务场景产生巨大的应用价值，比如在金融领域用图发现羊毛党。我相信 Flink 正式看中了这一点，在自己的核心之上，发展出来进行图计算的 Gelly。

2016 年 Flink 在国内也逐渐的引起了大数据同仁们的重视，阿里巴巴针对 Flink 对 Yarn 支持的不足做了很多的优化和修改，开发了 Blink，并且积极的与 Flink 社区进行沟通，希望能够将一些核心的修改 merge 回社区。而 TalkingData 也在对 Flink 进行尝试，相信在 Flink 社区，会有越来越多的中国人的身影和贡献。

Beam

提到流式处理，不得不提的一个项目是 Apache Beam。这是一个仍旧在孵化器中的项目，但是其出发点和背景使得我们不在早期就对它保持持续的关注。Beam 本身不是一个流式处理平台，而是一个统一的编程框架。

在大数据处理和计算平台百花齐放的今天，开发者不得不面对 Spark, Flink, Storm, Apex 等等不同的计算框架，而这些计算框架各自有不同的开发 API，如何能够屏蔽底层的差异，使得上层有一个统一的表达，对于大数据应用开发者来讲就变得非常有意义了。

TalkingData 在构造自己的 Data Cloud 的时候就面临这个问题，而这个时候我们发现 Beam 就给了我们这个答案。Beam 系出名门，是由 Google 开源出来的，并且得到了 Spark, Flink 等等社区的大力的支持。在 Beam 中，主要包含两个关键的部分：

Beam SDK

Beam SDK 提供一个统一的编程接口给到上层应用的开发者，开发者不需要了解底层的具体的大数据平台的开发接口是什么，直接通过 Beam SDK 的接口，就可以开发数据处理的加工流程。Beam SDK 会有不同的语言的实现，目前提供 Java，python 的 SDK 正在开发过程中，相信未来会有更的的不同的语言的 SDK 会发布出来。

Beam Pipeline Runner

Beam Pipeline Runner 是将用户开发的 pipeline 翻译成底层的数据平台支持的运行时环境的一层。针对不同的大数据平台，会有不同的 Runner。目前 Flink, Spark, Apex 以及 google 的 Cloud DataFlow 都有支持 Beam 的 Runner。

在 Strata+Hadoop 纽约的大会上，通过与 Beam 团队的沟通我了解到，尽管 Beam 现在仍旧是在孵化器中，但是已经足够的成熟和稳定，Spotify 公司就在用 Beam 构造自己的大数据 pipeline。

大数据分析和计算技术方兴未艾

提到大数据技术，最基础和核心的仍旧是大数据的分析和计算。在 2016 年，大数据分析和计算技术仍旧在飞速的发展，无论老势力 Hadoop 还是当红小生 Spark，乃至新兴中间力量 Druid，都在 2016 年继续自己的快速的发展和迭代。

Hadoop

近两年 Spark 的火爆使得 Hadoop 犹如昨日黄花，其实 Hadoop 并没有停止自己的发展的脚步。在 2016 年，Hadoop 3.0 的 alpha1 版本终于面世。而伴随着 Hadoop 3.0 正式版本发布的日益临近，Hadoop 3.0 能够给我们

带来些什么呢？

Erasure Coding 的支持

这个特性真是千呼万唤始出来。在当前这个时代，Hadoop 在一个大数据平台中最核心的部分就是 HDFS。而 HDFS 为了保证数据的可靠性，一直采用的是多副本的方式来存储数据。但是这几年数据规模的增加远远大于人的想象，而这些产生的数据，必然会存在冷热数据的区分。

无论冷热，数据对于一个公司都是核心的资产，谁都不希望数据丢失。可是对于冷数据，如果采用多副本方式，会浪费大量的存储空间。通过 Erasure Coding，则可以大大的降低数据存储空间的占用。对于冷数据，可以采用 EC 来保存，这样能够降低存储数据的花销，而需要时，还可以通过 CPU 计算来读取这些数据。

Yarn Timeline Service V.2

在 Hadoop 3.0 中，引入了 Yarn 时间轴服务 v.2 版本，用于应对两大挑战：

1. 改善时间轴服务的可伸缩性和可靠性；
2. 通过引入流和聚合增强可用性。

MapReduce 任务本地优化

通过 map 输出本地收集的支持，可以大幅优化一些对 shuffle 比较敏感的任务的性能，能够有超过 30% 的性能的提升。

支持超过两个 NameNode

在以前的版本中，NameNode 只能有两个来实现高可靠性，其中一个 namenode 是活跃的，另外一个则是 standby。但是有些场景需要更高的可靠性，在 Hadoop 3.0 中可以配置超过一个的 Standby 的 name node，从而保证更高的可靠性。

跨 Datanode 的 balancer

在旧的版本中，一个 datanode 管理一个物理机上的所有的磁盘，正常情况下是平均分配的写入，但是如果有磁盘的增减，就会造成数据的倾斜。在 Hadoop 3.0 上引入了新的跨 DataNode 的 balancer，可以更好的解决磁盘数据倾斜的问题。

Spark

在 2016 年，Spark 迎来了最近两年的一个最大的版本的发布，Spark 2.0 的发布。从年初开始，Spark 就在对 Spark 2.0 进行预热，可是 Spark 2.0 的发布并不如预期来的顺利。5 月份 Spark 2.0 Preview Release 发布，时隔两个月到 2016 年 7 月份，Spark 2.0 的正式版本发布。

不过 Spark 2.0 的正式版本也并没有完全达到预期，仍旧有很多的 bug，而结构化流式仍旧处于实验性阶段，一直到十一月发布的 2.0.2，还是 2.0 的 bug fix。在这一年中，Spark 主要的发展如下：

提升性能

从钨丝计划开始，Spark 就开始进行架构性的调整。无论开始的堆外内存的管理，到后边 2.0 逐渐引入的本地代码生成，都是希望能够使得自己能够变得更快。而很多 Spark 的用户也正式因为 Spark 的速度优势，逐渐从传统的 MapReduce 切换到了 Spark。

易用性

最初的一批 Spark 用户都需要花费一定的时间去理解 Spark 的 RDD 模型，对应的去了解 Spark 的开发的方法。虽然 Spark 应用开发起来简洁，但是相对普通程序员来讲，还是有一定的门槛。

随着 Spark 的日益普及，降低开发难度，提高易用性变成了 Spark 社

区的很重要的事情。摒弃掉 Shark，引入自己的 SQL 引擎，借鉴其他的数据平台抽象出 DataFrame 进而抽象出 DataSet，Spark 无疑变得对于普通程序员越来越友好，对于新晋 Spark 开发者来讲，会 SQL 就可以非常方便的开发大数据应用了。

流处理

在前面我们提到了大数据流式处理是新的趋势，Spark 无疑也感受到了这个趋势，并且期望能够跟随着这个趋势演进。Spark 从一产生就生成自己是将流式和批式处理统一的一个计算框架，可是 RDD 的特点决定了 Spark 的流式只是微批次，而不是纯粹的流式。而新的时代的挑战者 Flink 则称流式是第一等公民，并且在不同的 benchmark 上与 Spark Streaming 进行比对。

由于基础设计的不同，Spark Streaming 在延迟方面被 Flink 乃至 Apex 一直吊打，痛定思痛，Spark 社区决定引入结构化流式处理来应对。这也是 Spark 2.0 当中非常核心的一块儿增强，比较遗憾的是，Spark 的结构化流式在 2016 年发布到现在，仍旧是一个实验性的特性，让我们期待它尽快的成熟。

Druid

Druid 作为一个大数据的 OLAP 系统在 2016 年取得了巨大的成功，尤其在中国。在中国有越来越多的互联网公司采用 Druid 来构造自己的大数据分析平台，而 Druid 社区在中国也变得非常的活跃。几次 Druid Meetup 都取得了非常大的成功，Druid 的核心研发，华人工程师杨仿今也开始独立创业，并且获得了资本的青睐。

2015 年的时候当时在国内还只有很少的公司在采用 Druid。在 2016

年，阿里巴巴、迅雷、小米等等公司都开始采用 Druid 来构建自己的大数据平台。阿里巴巴基于 Druid 做了非常深度的定制开发来支撑自己的业务，而 TalkingData 也针对 Druid 在多维度精准排重统计的不足，将自己的 AtomCube 与 Druid 以插件的方式做了集成，使得 Druid 作为一个大数据的 OLAP 平台，具有了更强的能力。有理由相信，随着 Druid 在中国这个全球数据规模最大的市场的不同应用场景的落地，这个开源项目必定会产生越来越大的影响力。

展望 2017

回顾完 2016 年，让我们再对 2017 年做个展望，看看 2017 年在大数据领域会发生些什么：

- 流式数据处理成为主流，会有越来越多的企业采用流式数据来支撑自己分析、预测，从而能够更快速的做出决策；
- 人工智能和大数据技术融合，大数据技术的发展驱动了2016年人工智能的火热，而将人工智能与大数据处理相融合，构造智慧的大数据平台将会是一个新的趋势。人的智慧和机器的智能相互配合，可以大大的降低大数据处理的开销，从而显著提高大数据的投入产出比；
- 数据资产管理受到越来越多企业的重视，随着大数据加工和处理技术的日趋成熟，如何管理企业的数据资产变得越来越重要。相信会有越来越多的企业将会成立专门的大数据部门，来管理企业的数据资产，而对应的数据管理技术产品将会在2017年变得更为普及。

作者介绍

阎志涛，现任 TalkingData 研发副总裁，本科毕业于北京大学大气物

理专业，硕士毕业于华北计算计算技术研究所，研究方向为分布式计算系统。在加入 TalkingData 之前，历任 IBM CDL 资深架构师，Oracle 亚太区首席中间件技术顾问，BEA 亚太区首席中间件技术顾问等职务。超过 15 年的 IT 领域从业经验，一直从事大规模分布式计算系统、中间件、BI 等相关工作。

2016 移动开发技术巡礼

徐川

前言

2017 年就要来了，过去的一年里你是否疏于学习，欠下技术债呢？如果答案是肯定的，希望本文能让你在学习上少走一点弯路。

2016 年对于移动开发领域来说是颇受冲击的一年，Native 开发面临着 React Native、微信小程序等的冲击，再加上资本寒冬，想必想找移动开发初级工作的同学感受到了一丝凉意。

2016 年对于移动开发又是颇为精彩的一年，很多公司都把自己最核心的技术开源分享出来，呈现出百花齐放的局面。在本文里，我将会一一盘点这些技术。

另外，本文还试图对 2017 年的移动技术做一个预测，以及评点某些受争议的技术，需要强调的是，这些只是试图为读者提供看事情的一个角度，不必尽信。

平台篇

iOS 平台

苹果今年在 WWDC 上发布了 iOS 10、watchOS 3、macOS Sierra、tvOS 四大操作系统，并且在每个操作系统上都有创新，其中 iOS 无疑已成为最重要的操作系统，苹果在大会上宣布开放了三大框架，包括向建筑商开放智能家居平台 HomeKit、向开发者开放 SiriKit 和 CallKit，将更多能力开放给开发者。前两者由于一些限制，开发者使用的较少，而 CallKit 则将我们向纯网络电话时代推进了一大步，同时开放了用户期待已久的来电识别功能，目前已有很多 App 都开始使用该框架推出新功能。

另一个值得一提的是 iMessages，苹果在 WWDC 花很大篇幅对它进行了介绍，同时推出了 iMessages 内置的 App Store，关于它的意义，后面谈微信小程序的时候再说。

在秋季发布会上，苹果发布了新款 MacBook Pro，新加的硬件模块 Touch Bar 引起争议，但也引起了开发者的兴趣，大家都在探索它的新奇玩法。而且在实际上手后，大家发现它没有想象中的那么不便。这件事也告诉我们，对于没有接触过的事物，不要过早下结论。

在政策方面今年苹果有三大改变：

1. 从16年7月1日起，开发者所提交的应用必须能够支持纯IPv6网络环境。IPv4地址告罄已是事实，可以说Apple又一次走在了推动新标准的前列。
2. 同样是7月1日，苹果规定在App Store中国区上架的手游需要通过国家新闻出版广电总局的审批，在审核信息中提交游戏版本才可上架。对于这条，想必大家想吐槽的都吐槽过了，只能说这条充满中

国特色的新政不是苹果的锅。

3. 最后一个是苹果要求应用必须在2017年1月1日前支持HTTPS（已延期），最近不少团队都在忙这事。全面推行HTTPS，有助于保护手机用户的隐私，可以说补全了苹果手机的安全短板。

除了第二个之外，另外两个都是进步的体现，虽然给开发者带来了一些麻烦，但这样做可以避免将来可能出现的安全隐患和一些问题。

与开发者有关的另一个重头戏就是 Swift，9 月 13 号 Swift 3.0 正式发布，Swift 3.0 中的一系列变化旨在依照 Swift API Design 指南，通过去除一部分可能被视作遗留下来的 Objective-C 的内容，来清理 API。不过并不包含之前传言的 ABI 稳定。

Swift 的发展可以说很稳健，同时也给人以惊喜。10 月 25 号 Swift 团队在官方博客上宣布成立服务端 API 工作组，彰显了 Swift 向其它平台扩展的野心，在未来 Swift 也许可以达成前后端通吃。

关于 Swift 的应用情况，国外在开发 iOS App 的时候早已是 Swift First，从各种网络教程和分享来看，基本都是基于 Swift 的。反观国内，Swift 只有在部分新业务和新团队才会考虑使用，对于国内的超级 App 来说，由于需要频繁发版，同时很多应用使用了基于 OC runtime 的“黑科技”，使用 Swift 重写一部分应用不太现实，这些我们也需要承认。但是学习 Swift 从现在就可以开始了。

你可以在这里回顾 WWDC 的演讲视频和技术 Session，对于学习苹果新技术，这是第一手资料： https://developer.apple.com/videos/wwdc2016/。

另外，对于中国市场，苹果宣布将在北京和深圳成立两处研发中心，用于聚合在中国的工程和运营团队，面向中国以及全球的用户持续为其产

品打造领先的技术和服务。未来我们将可以看到更多的为中国的本地化优化，同时我们也期待苹果和国内工程师有更多的交流。

苹果平台 2017 年预测

苹果每年都努力为我们带来一些惊喜，要准确的预测几乎是不可能的，这里我大胆的来预测一下。

过去一年，我们经常看到苹果软件质量受到诟病，这并奇怪，因为苹果目前要开发维护四大操作系统，一个大型 IDE，诸多内置 App（包括重量级的办公软件），甚至还有 Apple News、Apple Music 这样的新业务，任何一个公司面临这样的情况都有可能力不从心。在今年苹果将 OS X 重新命名为 macOS，与其它几个操作系统统一，而据媒体报道，苹果有意将 macOS 的开发由 iOS 团队负责，这意味着 macOS 有可能并入 iOS，成为该系统的一个分支。这样苹果的重担会减轻不少。在 2017 年的 WWDC 上很可能宣布进一步的情况。

在新平台和新技术上，苹果可能推出基于 VR/AR 的新设备，或者是一个新框架。有消息显示苹果今年在 VR 领域有所投入，并且获得了 VR 显示以及头显的专利。

另外，在人工智能方面，明年苹果可能会推出新的系统级 AI 框架。其实 iOS 10 里面已经包含了很多 AI 技术，如语音识别、图片识别等，Siri 的背后更是集人工智能应用之大成，Metal 和 Accelerate 框架也提供新的卷积神经网络 API，但我们还需要一个应用级的框架。按照人工智能现在的热潮，可以预计明年的 WWDC 苹果会重点介绍这方面。

最后，关于 Swift 4 已经确认会在明年发布，Swift 的消息其实相当开放，我们可以在 Swift Evolution 项目看到它的最新进展。前不久 Swift 创始人 Chris 分享了对于 Swift 4 和 Swift 5 的规划。Swift 4 会

着重解决开发者关心的问题，以及 ABI 稳定性。苹果会不会配合着来一波推广呢，我觉得很有可能。

Android 平台

今年 Android 新版本 7.0 的出现要比往年早一些，在 IO 大会前两个月就发布了开发者预览版，与此同时还推出了 Android 测试版计划，学习微软的 Windows Insider，让用户也能提前体验最新版系统。新系统最受欢迎的新功能包括 Doze 模式和增强的通知中心。

与 Android 相关的平台今年也获得更新，包括 Android Wear 2.0，能够独立运行 App；Android Auto，支持在手机上运行，将手机变成车载智能系统；Android Things，由之前的物联网平台 Brillo 更名而来，。

与开发者相关的新特性包括：

- 新的Interpreter+JIT+AOT编译器。该编译器减少了应用程序安装时间和存储空间，并在手机空闲的时候编译代码的热点部分，提高性能。
- Vulkan跨平台的图形和计算API。在2015年发布的Vulkan也终于植入到Android系统中。
- 多窗口支持。还包括分屏、画中画等等。
- 能免安装运行的Instant Apps，看上去是革命性的，据分析其原理可能和国内正在研究的插件化/沙盒技术类似，不过到现在也一直处于测试状态。
- 渐进式Web应用PWA。这是今年Google重点推广的一项Mobile Web技术，大幅提升了离线使用、通知等体验。

在 IO 大会上还同时发布了 Android Studio2.2，它带来的新特性包括：

- 将编译器替换成Google自己开发的Jack编译器，以及配套的工具链Jack and Jill。
- 部分支持Java 8特性，包括lamda表达式，使用这个需要Jack编译器。
- 新的布局方式ConstraintLayout和相应的可视化编辑器，可以看做是RelativeLayout的增强。它的目标是减少布局的层级，同时改善布局性能，还减少了使用RelativeLayout的复杂性 。

在中国市场方面，Google 前不久刚在国内举办了两场规模盛大的开发者大会。发布了新的中文版开发者官网和开发者博客。不过，他们对于 Google 搜索、Play 市场等何时返回国内仍然讳莫如深，从 2016 年年初就传言 Google 返华，然而整个 2016 年进展也并不大。

Android 平台 2017 年预测

从今年的更新可以看到，Android 的发展也逐渐步入平缓阶段，人们的目光更多的被 Google 发布的其它新奇有趣的产品所吸引，比如 VR 平台 DayDream、智能助手和智能 IM 应用。有人甚至认为 Android 逐渐被边缘化，有可能被新系统所取代，比如前阵子传言的 Andromeda。

不过我想说这种担心是多余的，Android 仍然是 Google 最重要的产品之一，并且是其衍生品 Android Wear、Android Auto 等系统的基石，所以在近期不太可能有很大的改变。但是我们可以期待底层和工具上的大的创新，如 ART 运行时、Jack 编译器。

另一个在明年可能带来很大改变的技术是 Instant Apps，但是因为它很可能必须依赖 GMS，在国内不太可能使用。但根据国内在这方面的技术积累，我相信打造相同技术规格、完全兼容的中国版 Instant Apps 不是难事，说不定明年国内手机厂商会将它作为重点功能进行宣讲呢。

最后，根据 Android 新版本的普及速度，明年 Android 7.0 才会逐步在国内推广开，于是开发者们终于可以开始学习 Android 7.0 的新技术了。

iOS 开发技术篇

这里只讨论纯 iOS 开发技术，看看今年大家都讨论了哪些东西。

组件化

组件化并不是什么新东西，事实上当业务、团队规模大到一定程度，必然会去寻求模块化和组件化的方案。特别在国内存在大量超级 App 的情况下，组件化也早已在实践中实施。今年 1 月份蘑菇街李忠老师在第四届线下沙龙 MDay 上分享了蘑菇街组件化的实践。我邀请他到移动开发前线社群做了一个群分享，从此开启了对于这个话题的讨论。

组件化重点是将不同业务组件化，使不同的业务团队能够独立开发、测试和维护。而讨论的重点在于 App 内部和外部调用以及页面跳转，在蘑菇街分享后，casa 分享了他的看法，部分公司也分享了自己的组件化实践。

相关的讨论和分享文章如下：

- 蘑菇街App的组件化之路
- iOS应用架构谈 组件化方案
- 蘑菇街App的组件化之路·续
- iOS 组件化方案探索
- 滴滴的组件化实践与优化
- 手机淘宝客户端架构探索实践
- iOS App组件化开发实践

热补丁

尽管今年苹果将 App Store 上架审核流程加速，但对于人们的需求来

说，审核时间还是显得略长，发现问题马上修复是一个刚需，这在 iOS 开发中的体现就是热补丁。

在之前 iOS 中的热补丁开源项目有 Wax，今年随着前端在移动开发中的大火，JS 热修复方案火了起来，其中的代表就是 JSPatch，它在国内已经取得广泛应用。另外在热修复中，还需要与启动保护配合使用，否则有些无法启动的应用无法修复。

而在年底，滴滴发布了 DynamicCocoa，可以将 Objective-C 代码转换到 JS，然后下发到 JSCope 执行，用它甚至可以实现完全动态化，热修复只是它的一个功能。该项目计划于 2017 年初开源。

- JSPatch成长之路PPT
- iOS启动连续闪退保护方案
- DynamicCocoa：滴滴 iOS 动态化方案的诞生与起航

Swift

Swift 的重要性已不用多说，2016 年大家对于学习 Swift 的热情也很高。我们看到除了一些创业团队采用 Swift 开发之外，大公司的一些创新业务也开始使用 Swift 开发。另外，Swift 在服务端开始吹响号角，得到官方加持，未来 iOS 开发者可以无痛全栈了。

Swift 在今年的主要变化是从 2.x 升级到 3.0，由于是破坏性更新，在迁移时不免遇到种种问题，但也不是不能解决，另外相对于迁移，IDE 的支持才是更大的问题，很多人吐槽 Xcode 写 Swift 写着写着没有代码高亮、没有函数提示、崩溃闪退、编译耗时等等，这个我们也没什么办法，只能期待明年苹果好好重视一下这个问题了。

Swift 是一整门语言，技术点众多，挑几篇有代表性的文章：

- 网易漫画Swift混编实践
- ENJOY 工程 Swift 3 适配
- 适配 Swift 3 的一点小经验和坑

另外今年也出了不少 Swift 的学习书籍，推荐几本：

- 《Swift进阶》Chris Eidhof Airspeed Velocity 著 王巍 译
- 《函数式 Swift》Chris Eidhof Florian Kugler Wouter Swierstra 著 陈聿菡 杜欣 王巍 译
- 《Swift面向协议编程》 陈刚 编著

响应式编程 / 函数式编程 /FRP

函数响应式编程在 2015 年就是一个很火的话题，当时主要讨论的是 ReactiveCocoa，随着 Swift 逐渐普及的大趋势，以及 Rx 概念在其它语言中开始流行，RxSwift 异军突起，受到了一定关注。

由于响应式和函数式编程在团队中推广使用有一定难度，因此关于两者的布道不能停，相信明年还会有更多的分享。

- 美团App iOS开发与FRP
- MVVM 与 FRP 编程实战演讲视频与PPT
- Swift的响应式编程革命
- 是时候学习RxSwift了

Realm 数据库

移动开发中有多种客户端数据库可选择，包括苹果官方的 Core Data，以及经典的 SQLite，两者的应用都很广泛。在今年，一个创业公司的数据库产品引起了大家的兴趣，就是 Realm，它专为移动端而设计，API 简洁方便，有一些友好的特性，因此被大家关注。

其实 Realm 是一个跨端的数据库，在 Android 平台也有很多人学习和使用。我将之放在 iOS 下是因为有些人认为 Core Data 难用，而 Realm 是一个很好的替代。

- iOS遗留系统重构实践 从Core Data到Realm
- 移动端数据库新王者：realm
- 手把手教你从Core Data迁移到Realm
- Realm数据库 从入门到“放弃”

2017 年展望

技术总是日新月异，我们不能断言明年 iOS 领域会流行什么技术，但有一些线索并不难寻：

1. 动态化/热修复。DynamicCocoa会在明年初开源，到时候又会是一轮讨论热潮。
2. Swift实践/黑科技。Swift的安全特性让之前Objective-C runtime里的特技失传，这也是很多大型App不愿意采用Swift的原因。一方面我们期待官方会有更多的高级特性放出，另一方面也期待社区的突破。
3. 其它领域技术的借鉴。在前端、Android开发中有些好的设计和理念，可以将之借鉴到iOS平台，比如Redux、Android中的Layout、资源设计。在今年其实有一些讨论和开源项目出现，但它们到底能带来多少收益，有没有最佳实践，这是明年我会关注的一个话题。
4. HTTP/2和Protobuf，今年苹果在网络上动作很大，并且都是强制性的，开发者必须跟进，明年苹果是否还有更多的动作，以及在网络上的新玩法HTTP/2和Protobuf，相信会有不少团队去探索和实践。

Android 开发技术篇

2016 对于 Android 开发来说是非常精彩的一年，黑科技频发，我们可以确定的说在 Android 应用层开发方面我们已经走在了世界前面。然而，其实很多黑科技都是形势所逼，并且其中一些和 Google 的官方政策相违背，这是目前热火的表象下的隐忧。

插件化

从去年下半年开始，Android 插件化开始进入人们的视野，到今年，开源和分享开始爆发，插件化可以说是 Android 开发高级技术的集大成者，要掌握它需要对 Android 系统框架、App 运行机制等足够了解，因此成为很多开发者追求的目标。

但是回过头来想一下，插件化的适用场景其实有限，多用于头部的超级应用，一般的中小型 App 没有必要适用，所以也没有必要盲目追求新技术。

插件化发展到后来，基本都开始追求免安装运行 App，也就是沙盒 / 双开，在这方面我们也有一些商业应用出现，以及开源项目。

这里就推荐一些开源项目：

- DynamicLoadApk 比较早的一款插件化项目，任玉刚等研发。
- DroidPlugin 360手机助手推出的插件化项目。
- VirtualApp 天才少年Lody推出的Android应用双开项目。
- Small wequick推出的开源项目，包括Android和iOS双平台（iOS平台仅限越狱）。
- DynamicAPK 携程出品的插件化框架，介绍见此。
- VirtualAPK，滴滴出行推出的插件化项目，尚未开源，介绍见此。

热补丁

插件化并不是所有应用都需要，但是热修复却基本是正式的项目都想要的。热补丁与插件化的不同点在于，热补丁的关注点在对应用进行方法级的替换以达成修复。2016 年 Android 的热修复取得了非常大的进展，不但有多家公司分享、开源了自己的热修复项目，而且还催生了商业服务。

这里就为大家盘点一下今年都出现了哪些热修复项目：

- AndFix，阿里推出的开源项目，并且在其基础之上衍生了商业服务。之前阿里还推出一个Dexposed，但由于一些问题已停止维护。
- Tinker，微信推出的开源项目，在年中宣布要开源时就引起了广泛关注。
- Amigo，饿了么推出的开源项目。
- Qzone超级补丁，暂未开源，演讲视频和PPT见此，社区有人根据其原理研发了Nuwa并开源，现已停止维护。
- QFix，手Q的热补丁项目，暂未开源，介绍见此。
- Robust，新美大的热补丁项目，暂未开源，介绍见此。

RxJava

RxJava 是 JVM 上的响应式编程框架，可以简化异步操作的代码，是 Rx 系列的一部分，去年年底国内社区开始有人布道，也有很多人分享，RxJava 与 Retrofit 结合可以大幅简化网络操作的复杂性，因此也被人们广泛使用。

推荐文章如下：

- 给 Android 开发者的 RxJava 详解
- 是时候学习RxJava了

- RxJava入门之实例解析
- 如何测试RxJava代码

Android 组件化

Android 里的组件化是相对于插件化来说的，插件化追求插件直接完全独立，甚至插件本身是可独立运行的 APK，组件化则是在组件独立开发，在编译时仍合成为完整 App。

Android 组件化的实践的代表是淘宝的 Atlas，该项目将于 2017 年初开源。

- 回归初心，从容器化到组件化
- Atlas：手淘Native容器化框架和思考
- Android业务组件化开发实践
- Android组件化开发实践

Kotlin

Kotlin 是老牌开发工具厂商 JetBrains 推出的一门 JVM 语言，也非常适合在 Android 项目中使用，今年推出了 1.0 版本，并在国内举办了一场线下开发者日活动。

Kotlin 被视为 Android 平台的 Swift，可以刷时髦值，从实际开发体验上来说，与 JetBrains 的 IDE 结合（Android Studio 亦可）也非常不错。国内有些团队和个人已经开始尝试。但由于 Java 本身很完备，更换语言目前来说只是基于个人喜好。因此要想 Kotlin 流行，就像苹果推广 Swift 一样，我们也需要谷歌的推动才会让更多人有兴趣学习和使用。

- Kotlin语言1.0Beta发布，JetBrain介绍其设计理念
- Kotlin如何成为我们Android开发的主要语言

- 使用Kotlin&Anko，扔掉XML开发Android应用

2017 年展望

数一数将于明年开源的一些项目，明年 Android 开发该讨论我们也心里有数。

- 插件化。2017年，Android开发领域关于插件化和热补丁的讨论仍将继续，但我希望国内开发者能更多的走出去和国外同行交流，让这些技术不只是一头热。
- 组件化。其重点在于工程期的辅助工具和编译流程，期待明年除了Atlas之外，有更多的类似项目出现。
- Kotlin，在这里推这个，也算是我的个人喜好吧，希望明年Google能官方表态支持Kotlin.
- AI技术。人工智能是目前最火的技术领域没有之一，如何与移动结合结合玩出花来是一个值得探讨的话题。我觉得Android上取得突破的可能性比iOS更大，毕竟Android更开放，玩法更多。

跨平台技术篇

不只是今年，跨平台技术一直在移动开发领域火热非常，毕竟 Android、iOS 都不是可以轻易舍弃的平台。今年的跨平台技术，相比往年的 Xamarin、Titanium 等，更实际，更接地气，这也导致了跨平台技术今年讨论的热度更高，都快刷屏了。

今年大家所讨论的跨平台技术，无论是 React Native、Weex 还是微信小程序，从技术实现来说都是处于 Native 和 Web 之间，还是使用 Web 技术开发，从 Hybrid 过渡过来是很自然的事情，因此国内不少公司在 React Native 推出不久就开始使用。

RN 及类似技术最光明的未来可能是：类 RN 首先取代当前 App 中的 WebView，之后会取代一部分 Native 界面，长期上来说，一些性能要求高的本地库也会提供对它们的支持，如 Realm 就提供了 RN 版本，原生开发会被进一步压缩。原生开发，未来可能专注于一些专项研发，如音视频，或公共组件接入和维护。

当然，这种未来不一定到来，但移动开发者要做好心理准备。

React Native

React Native 如今的地位毋庸置疑，Facebook 没有在国内专门宣讲过，但我已经见识到了它的一些国内铁粉，并且有些公司如携程更是不遗余力的对它进行支持和布道，这种情况下 RN 发展想不好都不行。

但这样也出现了一些问题，一个是本地化的支持，第二个是有些公司自己对 React Native 做了优化，但没有反馈给 RN 项目，这样最终只是做了一个分支版本，他踩的坑别人还是要踩一遍，这样不环保，长远来看还有维护难的问题。

今年在 RN 上有实践并分享出来的团队的分享如下：

- QQ空间React Native项目实战总结
- 携程是如何做React Native优化的
- 基于React Native的58同城App开发实践
- 宝宝树React Native增量升级解决方案PPT
- 旅行喵 React Native 技术实践

Weex

Weex 是阿里手淘团队推出的跨平台开发框架，于今年 6 月份开源。阿里对该框架非常重视，在全集团推广，并且在今年双十一会场大规模应

用，取得了成功。在 9 月份的 JSConf 上他们邀请 Vue.js 作者加入团队担任技术顾问，刚刚发布的 0.9 版也正式支持 Vue 2.0 语法特性了。最近它还加入了 Apache 基金会的孵化器，力争打造成功的开源社区。

相比于 React Native，Weex 略显年轻，不过由于后发优势，在一些地方优化做的比 RN 好。至于用哪个，就要开发者自己去探索哪个更适合自己了。

- Weex：关于移动端动态性的思考、实现和未来
- Weex详解：灵活的移动端高性能动态化方案
- awesome-weex

微信小程序

今年下半年，对移动开发最具冲击的可能要数微信小程序了，它的颠覆性不在于技术，而在于对整个移动互联网生态的影响。大部分的产品经理也许不知道 RN 或者 Weex，但他们一定知道微信小程序，也一定在考虑要不要做一个，这就是微信小程序的影响力。

其实，在 IM 里做平台微信不是第一个，在 6 月份的 WWDC 上苹果展示的 iMessages，就支持集成其它应用，发红包不在话下，甚至还能打车。可惜 iMessages 这个应用实在不接地气，在国内也没有什么影响力。而另一个国人同样不怎么了解的 IM 应用，Facebook Messenger 在向微信学习，在今年 3 月举行的 F8 开发者大会上，Messenger 平台化服务正式对外推出。扎克伯格宣布已有 40 多项服务接入到 Messenger 平台。最近它还推出了 Instant Games，你可以在 FB Messenger 里玩游戏了。

这些现象体现的趋势是基于 IM 的应用分发可能是下一件大事，而这个能参与的只有现有的超级 IM。

所以虽然现在微信小程序看上去能做的还很有限，对于移动 App 没有替代作用，但将来就说不好了，保持关注吧。

PWA

PWA 是 Google 今年力推的一项移动 Web 技术，在支持 Service Worker 和 Fetch API 的浏览器上可以得到完整体验，在不支持的浏览器可以得到降级体验。

我不是很看好 PWA，因为它的能力和性能还是有限，国内目前是超级 App 当道，对于性能都是精益求精，几乎不可能去采用这个方案，更何况它在 iOS 上的体验是降级的，不支持服务器推送，就这一条也会被 Pass 掉。而苹果何时支持 Service Worker 等新特性是说不好的。

对于页面元素少、交互简单的应用可以尝试一下。

热门行业技术篇

2016 年还有些“移动 +”的领域十分火热，这里指的就是直播和 VR/AR，它们除了技术，还有趋势上的问题。

直播

直播是 2016 年的年度标志事件，其火热程度不用多说，因此也有很多泡沫，在数字上弄虚作假已经是人人皆知的事实。但是要看到，直播的兴起的前提是我们软硬件达到了要求，同时直播是对某些陈旧事物的替代，因此即使泡沫破灭，也可以预见剩下的几家会活得很好。

同时，直播开始作为一个功能模块进入很多 App，和其它行业结合，如淘宝、天猫上的直播就给网络购物带来了很好的效果。因此，作为年度热点技术，每个移动开发者都有必要去了解和学习。

直播技术本身涉及到前后端，最有挑战的部分其实不在客户端而是在

后端，客户端的部分大多数有了最佳实践，同时在秒开优化上有了很多创新。

- 移动直播技术秒开优化经验
- 阿里直播平台面临的技术挑战
- 从0到1打造直播 App

VR/AR

与直播相比，VR/AR 则是从得意到失意的典型，年初 VR 还被认为是年度技术，到年底某些做 VR 设备的厂商传出裁员、倒闭，高端头显设备如 HTC vive 的销售量也并不大，一下子大家都不看好了，这个领域似乎被资本所抛弃。

但与此同时，使用 VR/AR 技术的产品屡屡成为互联网的爆点，如 Pokemon Go、阿里双十一 VR、支付宝 AR 红包，说明 VR/AR 在用户体验和交互模式上还是很有潜力，只是我们要找到正确的使用方法。目前来看，基于 LBS+AR 的游戏是可行的点，VR 则在营销上有用武之地。

另外，虽然 VR/AR 被资本不看好，但互联网巨头仍然重金投入，如 Google、Facebook、Microsoft 都将 VR/AR 作为自己的战略发展目标。Google 的 Daydream 更是非常接地气，期待明年当支持设备增多后，给我们更加完善的体验。

总之，VR/AR 目前来看并不会死，但现在投入风险很高，建议可以再等等。

还有哪些没有提到？

性能监控与分析。今年不少公司都研发了自己的客户端性能监控系统。

安装包瘦身。Android 有魔鬼瘦身，iOS 则以滴滴为代表，用 clang

插件瘦身。

Hybrid 开发，虽然已经是很成熟的技术，但仍有可创新之处，豆瓣和美团都有很好的分享。

技术不会停下脚步，学习永无止境。让我们期待 2017 年的移动技术吧。

2016 前端开发技术巡礼

殷勇

2016 年马上过去了，像过去六年中的每一年一样，Web 前端领域又产生了“面目全非”而又“耳目一新”的变化，不但旧事物持续不断地被淘汰，新事物也难保坐久江山，大有岌岌可危之势。开源界如群雄逐鹿，不断生产新的概念、新的框架、新的工具，去年中一些流行的技术今年大多得到了进一步的演进和升级，活跃度非常高，却仍然不能保证前端的未来属于它们。在今年整体资本市场冷却的大环境下，to B 业务的创业公司显现出了较强的生命力，这种类型的业务也给 Web 前端的工作带来了明显的差异性，工程师整体技能方向也展露出一丝不一样的分支。本文将从下至上、由低到高的维度盘点过去一年中 Web 前端领域发生的重要事件以及影响未来 2017 的关键性因素。视野所限，不尽完整。

一、更新的网络与软件环境

1.1 HTTP/2 的持续普及

今年中，几乎所有的现代桌面浏览器都已经支持了 HTTP/2 协议，移动端依靠降级为 SPDY 依旧可以覆盖几乎所有平台，这样使得从协议上优化页面的性能成为了可能。

同时，前端静态资源打包的必要性成为了一定程度上的争论焦点，打包合并作为传统的前端性能优化方案，它的存留对前端工程化影响极大，Facebook 公司著名的静态资源动态打包方案的优越性也会被弱化。社区上多篇文章纷纷发表对 HTTP/2 的性能实验数据，却不尽相同。

在 2017 年，我相信所有大型站点都会切换 HTTP/2，但依旧不会放弃对静态资源打包合并的依赖。而且，对于 Server Push 等高级特性，也不会有太多的应用。

1.2 Internet Explorer 8

三年前还在考虑兼容 IE6 的前端技术社区，在前不久天猫宣布不再支持 IE8 后又引起了一股躁动。IE8 是 Windows XP 操作系统支持的最高 IE 版本，放弃 IE8 意味着放弃了使用 IE 的所有 XP 用户。

其实在 2016 年的今天，前端社区中框架、工具的发展早已不允许 IE8 的存在，Angular 早在 1.3 版本就果断放弃了 IE8，React 也在年初的 v15 版本上宣布放弃。在 PC 领域，你依旧可以使用像 Backbone.js 一样的其他框架继续对 IE 进行支持，但无论是从研发效率上还是从运行时效率上，放弃它都是更好的选择。

由于对 HTML5 兼容性不佳，在 2017 年，相信 IE9 也会逐渐被社区放弃，以取得更好的性能、更少的代码体积。

二、如何编写 (Java)Script

2.1 ES2016？ ES2017？ Babel！

去年定稿的 ES2015（亦称 ES6）带来了大量令人激动的新语言特性，并快速被 V8 和 SpiderMonkey 所实现。但由于浏览器版本碎片化问题，目前编写生产环境代码仍然以 ES5 为主。今年年中发布的 ES2017 带来的新特性数量少的可怜，但这正好给了浏览器厂商消化 ES2015 的时间，在 ES2017 到来之前喘口气——是的，明年的 ES2017 势必又会带来一大波新特性。

JS 解释引擎对新特性的支持程度并不能阻碍狂热的开发者使用他们，在接下来的很长时间，业界对 Babel 的依赖必然有增无减。Babel 生态对下一代 ECMAScript 的影响会进一步加大，人们通过先增加新的 Babel-plugin，后向 ECMA 提案的方式成为了 ECMAScript 进化的常态。开发者编写的代码能直接运行在浏览器上的会越来越少。

但使用 Babel 导致的编译后代码体积增大的问题并没有被特别关注，由于 polyfill 可能被重复引入，部署到生产环境的代码带有相当一部分冗余。

2.2 TypeScript

作为 ECMAScript 语言的超集，TypeScript 在今年取得了优异的成绩，Angular 2 放弃了传说中的 AtScript，成为了 TypeScript 的最大客户。人们可以像编写 Java 一样编写 JavaScript，有效提升了代码的表述性和类型安全性。

但凡事有两面，TypeScript 的特性也在不断升级，在生产环境中，你可能需要一套规范来约束开发者，防止滥用导致的不兼容，这反而增加

了学习成本、应用复杂性和升级安全性。个中优劣，仍需有大量的工程实践去积累经验。

此外，TypeScript 也可以看做一种转译器，与 Babel 有着类似的新特性支持。在 2017 年，我们期待 TypeScript 与 Babel 会发展成怎样的一种微妙关系。

2.3 promise、generator 与 async/await

在回调地狱问题上，近两年我们不断被新的方案乱花了眼。过去我们会利用 async 来简化异步流的设计，直到“正房”Promise 的到来。但它们只是 callback 模式的语法糖，并没有完全消除 callback 的使用。

ES2015 带来的 generator/yield 似乎成为了解决异步编程的一大法宝，虽然它并非为解决异步编程所设计的。但 generaor 的运行是十分繁琐的，因此另一个工具 co 又成为了使用 generator 的必备之选。Node.js 社区的 Koa 框架初始就设计为使用 generator 编写洋葱皮一样的控制流。

但昙花一现，转眼间 async/await 的语法，配合 Promise 编写异步代码的方式立即席卷整个前端社区，虽然 async/await 仍然在 ES2017 的草案中，但在今天，不写 async/await 立刻显得你的设计落后社区平均水平一大截。

在 Node.js 上，v7 已经支持在 harmony 参数下的 async/await 直接解释，在明年 4 月份的 v8 中，将会正式支持，届时，Koa 2 的正式版也会发布，几乎完全摒弃了 generator。

2.4 fetch

受到回调问题的影响，传统的 XMLHttpRequest 有被 fetch API 取代之势。如今，成熟的 polyfill 如 whatwg-fetch、node-fetch、

isomorphic-fetch 在 npm 上的每日下载量都非常大，即便对于兼容性不好的移动端，开发者也不愿使用繁琐的 AJAX。借助 async/await 的语法，使用 fetch API 能让代码更简洁。

三、Node.js 服务与工具

3.1 Koa 2

Koa 与流行的 Express 属于“同根生”的关系，它们由同一团队打造。相比 Express，新的 Koa 框架更轻量、更灵活。但 Koa 的设计在短时间内曾经出现了较大的变动，这主要受到了 async/await 语法对异步编程的影响。在 v2 版本中，Koa 的 middleware 抛弃 generator 转而支持 async，所有第三方 middleware 实现，要么自行升级，要么使用 Koa-convert 进行包装转换。

目前 Koa 在 Node.js 社区的 HTTP 服务端框架中受到关注度比较高，不过其在 npm 上 latest 目前仍处于 1.x 阶段，预计在 2017 年 4 月份发布 Node.js v8 后，就会升级到 2.x。

Koa 的轻量级设计意味着你需要大量第三方中间件去实现一个完整的 Web 应用，目前鲜有看到对 Koa 的大规模重度使用，因此也就对其无从评价。相信在明年，越来越多的产品应该会尝试部署 Koa 2，届时，对第三方资源的依赖冲突也会尖锐起来，这需要一个过程才能让 Koa 的生态完备起来。预计在 2018 年，我们会得到一个足够健壮的 Koa 技术栈。这会促进 Node.js 在服务端领域的扩展，轻量级的 Web 服务将会逐渐成为市场上的主流。

四、框架纷争

4.1 jQuery 已死?

今年六月份 jQuery 发布了 3.0 版本，距离 2.0 发布已经有三年多

的时间，但重大的更新几乎没有。由于老旧浏览器的逐渐放弃和升级，jQuery 需要处理的浏览器兼容性问题越来越少，专注于 API 易用性和效率越来越多。

随着如 Angular、React、Ember、Vue.js 等大量具备视图数据单双向绑定能力的框架被普及，使用 jQuery 编写指令式的代码操作 DOM 的人越来越少。早在 2015 年便有人声称 jQuery 已死，社区中也进行了大量雷同的讨论，今天我们看到确实 jQuery 的地位已大不如前，著名的 sizzle 选择器在今天已完全可由 querySelector* 原生方法替代，操作 DOM 也可以由框架根据数据的变动自动完成。

明年 jQuery 在构建大型前端产品的过程中的依赖会被持续弱化，但其对浏览器特性的理解和积淀将对现有的和未来的类 Angular 的 MVVM 框架的开发依旧具有很大的借鉴意义。

4.2 Angular 2

好事多磨，Angular 2 的正式版终于在今年下半年发布，相比于 1.x，新的版本几乎是完全重新开发的框架，已经很难从设计中找到 1.x 的影子。陡峭的学习曲线也随之而来，npm、ES2015 Modules、Decorator、TypeScript、Zone.js、RxJS、JIT/AOT、E2E Test，几乎都是业界这两年中的最新概念，这着实给初学者带来了不小的困难。

Angular 2 也更面向于开发单页应用（SPA），这是对 ES2015 Modules 语法描述的模块进行打包（bundle）的必然结果，因此 Angular 2 也更依赖于 Webpack 等“bundler”工具。

虽然 Angular 声称支持 TypeScript、ECMAScript 和 Dart 三种语言，不过显然业界对 Dart 没什么太大兴趣，而对于 ECMAScript 和

TypeScript，两种语言模式下 Angular 2 在 API 和构建流程上都有着隐式的（文档标注不明的）差异化，这必然会给开发者以困扰。加上业界第三方工具和组件的支持有限，TypeScript 几乎是现在开发者唯一的选择。

此外，Angular 团队已声明并没有完全放弃对 1.x 组件的支持，通过特有的兼容 API，你可以在 2.x 中使用针对 1.x 开发的组件。鉴于不明确的风险，相信很少有团队愿意这样折腾。

现在在产品中使用 Angular 2，在架构上，你需要考虑生产环境和开发环境下两种完全不同的构建模式，也就是 JIT 和 AOT，这需要你有两套不一样的编译流程和配置文件。在不同环境下模块是否符合期望，可以用 E2E、spec 等方式来进行自动化测试，好的，那么 Angular 2 的测试 API 又可能成了技术壁垒，它的复杂度可能更甚 Angular 本身。可以确信，在业务压力的迫使下，绝大部分团队都会放弃编写测试。

总之，Angular 2 是一个非常具有竞争力的框架，其设计非常具有前瞻性，但也由于太过复杂，很多特性都会成为鸡肋，被开发者所无视。由于 React 和 Vue.js 的竞争，Angular 2 对社区的影响肯定不如其前辈 1.x 版本，且其更高级的特性如 Server Render 还没有被工程化实践，因此相信业界还会持续观望，甚至要等到下一个 4.x 版本的发布。

4.3 Vue.js 2.0

Vue.js 绝对是类 MVVM 框架中的一匹黑马，由作者一人打造，更可贵的是作者还是华人。Vue.js 在社区内的影响非常之大，特别是 2.0 的发布，社区快速生产出了无数基于 Vue.js 的解决方案，这主要还是受益于其简单的接口 API 和友好的文档。可见作为提供商，产品的简单易用性显得尤为重要。在性能上，Vue.js 基于 ES5 Setter，得到了比 Angular 1.x 脏

检查机制成倍的性能提升。而 2.0 在模块化上又更进一步，开发难度更低，维护性更好。可以说 Vue.js 准确地戳中了普通 Web 开发者的痛点。在国内，Vue.js 与 Weex 达成了合作，期待能给社区带来怎样的惊喜。

4.4 React

目前看来，React 似乎仍是今年最流行的数据视图层解决方案，并且几乎已经成为了每名前端工程师的标配技能。今年 React 除了版本从 0.14 直接跃升至 15，放弃了 IE8 以外，并没有更多爆发式的发展。人们对于使用 JSX 语法编写 Web 应用已经习以为常，就像过去十年间写 jQuery 一样。

React 的代码在维护性能上显而易见，如果 JSX 编写得当，在重渲染性能上也具备优势，但如果只部署在浏览器环境中，那么首屏性能将会受到负面影响，毕竟在现阶段，纯前端渲染仍然快不过后端渲染，况且后端具备天生的 chunked 分段输出优势。我们在业界中可以看到一些负面的案例，比如某新闻应用利用 React 全部改写的 case，就是对 React 的一种误用，完全不顾其场景劣势。

围绕着 React 发展的替代品和配套工具依旧很活跃，preact 以完全兼容的 API 和小巧的体积为卖点，inferno 以更快的速度为卖点，等等。每个框架都想在 Virtual DOM 上有所创新，但它们的提升都不是革命性的，由此而带来的第三方插件不兼容性，这种风险是开发者不愿承担的，笔者认为它们最大的意义在于能为 React 的内部实现提供另外的思路。就像在自然界，生物多样性是十分必要的，杂交能带来珍贵的进化优势。

4.5 React-native

今年是 React-native（一下简称 RN）支持双端开发的第一年，不断有团队分享了自己在 RN 上的实践成果，似乎前途一片大好，RN 确实有效

解决了传统客户端受限于发版周期、H5 受限于性能的难题，做到了鱼和熊掌兼得的理想目标。

但我们仍然需要质疑：首先，RN 目前以两周为周期发布新版本，没有 LTS，每个版本向前不兼容。也就是说，你使用 0.39.0 的版本编写 bundle 代码，想运行在 0.35.0 的 runtime 上，这几乎会 100% 出问题。在这种情况下，如何制定客户端上 RN 的升级策略？如果升级，那么业务上如何针对一个以上的 runtime 版本编写代码？如果不升级，那么这意味着你需要自己维护一个 LTS。要知道目前每个 RN 的版本都会有针对前版本的 bug fix，相信没有团队有精力可以在一个老版本上同步这些，如果不能，那业务端面对的将是一个始终存在 bug 的 runtime，其开发心理压力可想而知。

其次，虽然 RN 声称支持 Android 与 iOS 双端，但在实践中却存在了极多系统差异性，有些体现在了 RN 文档中，有一些则体现在了 issue 中，包括其他一些问题，GitHub 上 RN 的近 700 个 issue 足以让人望而却步。如果不能高效处理开发中遇到的各种匪夷所思的问题，那么工期就会出现严重风险。此外，RN 在 Android 和 iOS 上的性能也不尽相同，Android 上更差一些，即便你完成了整个业务功能，却还要在性能优化上消耗精力。并且无论如何优化，单线程模型既要实现流畅的转场动画，又要操作一系列数据，需要很高的技巧才能保证可观的性能表现。在具体的实践中，对于 H5，往往由于时间关系，业务上先会上一个还算过得去的版本，过后再启动性能优化。然而对于 RN，很有可能达到“过得去”的标准都需要大量的重构工作。

再次，RN 虽然以 Native 渲染元素，但毕竟是运行在 JavaScript Core 内核之上，依旧是单线程，相对于 H5 这并没有对性能有革命性质的

提升。Animated 动画、大 ListView 滚动都是老生常谈的性能瓶颈，为了解决一些复杂组件所引起的性能和兼容性问题，诸多团队纷纷发挥主动能动性，自己建设基于 Native 的高性能组件，这有两方面问题，一是不利于分发共享，因为它严重依赖特定的客户端环境，二是它仍依赖客户端发版，仍需要客户端的开发，违背了 RN 最最重要的初衷。可以想象，在大量频繁引用 Native 组件后，RN 又退化成了 H5+Hybrid 模式，其 UI 的高性能优势将会在设备性能不断升级下被削弱，同时其无 stable 版本反而给开发带来了更多不可预测的风险变量。

最后，RN 仍然难以调试和测试，特别是依赖了特定端上组件之后，本地的自动化测试几乎成为了不可能，而绝大多数客户端根本不支持自动化测试。而调试只能利用 remote debugger 有限的能力，在性能分析上都十分不便。

可以说 RN 的出现带给了移动开发以独特的新视角，使得利用 JavaScript 开发 Native 成为了可能，NativeScript、Weex 等类似的解决方案也发展开来。显然 RN 目前最大的问题仍然是不够成熟和稳定，利用 RN 替代 Native 依然存在着诸多风险，这对于重量级的、长期维护的客户端产品可能并不是特别适合，比如 Facebook 自己。RN 的优势显而易见，但其问题也是严重的，需要决策者对个方面利弊都有所了解，毕竟这种试错的成本不算小。

由于时间关系，市场上并没有一个产品在 RN 的应用上有着足够久的实践经验，大部分依然属于“我们把 RN 部署到客户端了”的阶段，我们也无法预测这门技术的长久表现，现在评价 RN 的最终价值还为时尚早。在 2017 年，期待 RN 团队能做出更长足的进步，但不要太乐观，以目前的状态来看，想达到 stable 状态还是有着相当大的难度。

4.6 Redux 与 Mobx

Redux 成功成为了 React 技术栈中的最重要成员之一。与 Vue.js 一样，Redux 也是凭借着比其他 Flux 框架更简单易懂的 API 才能脱颖而出。不过已经很快有人开始厌烦它每写一个应用都要定义 action、reducer、store 以及一大堆函数式调用的繁琐做法了。

Mobx 也是基于 ES5 setter，让开发者可以不用主动调用 action 函数就可以触发视图刷新，它只需要一个 store 对象以及几个 decorator 就能完成配置，确实比 Redux 简单得多。

在数据到视图同步上，无论使用什么样的框架，都有一个至关重要的问题是需要开发者自己操心，那就是在众多数据变动的情形下，如何保证视图以最少的但合理的频率去刷新，以节省极其敏感的性能消耗。在 Redux 或 Mobx 上都会出现这个问题，而 Mobx 尤甚。为了配合提升视图的性能，你依然需要引入 action、transaction 等高级概念。在控制流与视图分离的架构中，这是开发者无可避免的关注点，而对于 Angular、Vue.js，框架会帮你做很多事情，开发者需要考虑的自然少了许多。

4.7 Bootstrap 4

Bootstrap 4 处于 alpha 阶段已经非常久了，即使现在 3.x 已经停止了维护，它似乎受到了 Twitter 公司业务不景气的影响，GitHub 上的 issue 还非常多。Bootstrap 是建设内部平台最佳的 CSS 框架，特别是对于那些对前端不甚了解的后端工程师。我们不清楚 Bootstrap 还能坚持多久，如果 Twitter 不得不放弃它，最好的归宿可能是把它交给第三方开源社区去维护。

五、工程化与架构

5.1 Rollup 与 Webpack 2

Rollup 是近一年兴起的又一打包工具，其最大卖点是可以对 ES2015

Modules 的模块直接打包，以及引入了 Tree-Shaking 算法。通过引入 Babel-loader，Webpack 一样可以对 ES2015 Modules 进行打包，于是 Rollup 的亮点仅在于 Tree-Shaking，这是一种能够去除冗余，减少代码体积的技术。通过分析 AST（抽象语法树），Rollup 可以发现那些不会被使用的代码，并去除它。

不过 Tree-Shaking 即将不是 Rollup 的专利了，Webpack 2 也将支持，并也原生支持 ES6 Modules。这可以说是“旁门左道”对主流派系进行贡献的一个典型案例。

Webpack 是去年大热的打包工具，俨然已经成为了替代 grunt/gulp 的最新构建工具，但显然并不是这样。笔者一直认为 Webpack 作为一个 module bundler，做了太多与其无关的事情，从而表象上看来这就是一个工程构建工具。经典的构建需要有任务的概念，然后控制任务的执行顺序，这正是 Ant、Grunt、Gulp 做的事情。Webpack 不是这样，它最重要的概念是 entry，一个或者多个，它必须是类 JavaScript 语言编写的磁盘文件，所有其他如 CSS、HTML 都是围绕着 entry 被处理的。估计你很难一眼从配置文件中看出 Webpack 对当前项目进行了怎样的“构建”，不过似乎社区中并没有人提出过异议，一切都运行得很好。

题外话：如何使用 Webpack 构建一个没有任何 JavaScript 代码的工程？

新的 Angular 2 使用 Webpack 2 编译效果更加，不过，已经提了一年的 Webpack 2，至今仍处于 beta 阶段，好在现在已经 rc，相信离 release 不远了。

5.2 npm、jspm、Bower 与 Yarn

在模块管理器这里，npm 依旧是王者，但要说明的是，npm 的全称是

node package mamager，主要用来管理运行在 Node 上的模块，但现在却托管了大量只能运行在浏览器上的模块。造成这种现象的几个原因：

- Webpack的大量使用，使得前端也可以并习惯于使用CommonJS类型的模块；
- 没有更合适的替代者，Bower以前不是，以后更不会是。

前端的模块化规范过去一直处于战国纷争的年代。在 Node 上 CommonJS 没什么意见。在浏览器上，虽然现在有了 ES2015 Modules，却缺少了模块加载器，未来可能是 SystemJS，但现在仍处于草案阶段。无论哪种，都仍处于 JavaScript 语言层面，而完整的前端模块化还要包括 CSS 与 HTML，以及一些二进制资源。目前最贴近的方案也就只能是 JSX+CSS in JS 的模式了，这在 Webpack 环境下大行其道。这种现象甚至影响了 Angular 2、Ember 2 等框架的设计。从这点看来，jspm 只是一个加了层包装的壳子，完全没有任何优势。

npm 本身也存在着各种问题，这在实践中总会影响效率、安全以及一致性，Facebook 果断地出品了 Yarn——npm 的替代升级版，支持离线模式、严格的依赖版本管理等在工程中非常实用的特性。

至于前端模块化，JavaScript 有 CommonJS 和 ES2015 Modules 就够了，但工程中的组件，可能还需要在不同的框架环境中重复被开发，它们依旧不兼容。未来的话，WebComponents 可能是一个比较优越的方案。

5.3 同构

同构的设计在软件行业早就被提出，不过在 Web 前端，还是在 Node.js、特别是 React 的出现后，才真正成为了可能，因为 React 内核的运行并不依赖于浏览器 DOM 环境。

React 的同构是一个比较低成本的方案，只要注意代码的执行环境，

前后端确实可以共享很大一部分代码，随之带来的一大收益是有效克服了 SPA 这种前端渲染的页面在首屏性能上的瓶颈，这是所有具备视图能力的框架 Angular、Vue.js、React 等的共性问题，而现在，它们都在一种程度上支持 server render。

可以想到的做前后端同构面临的几个问题：

1. 静态资源如何引入，CSS in JS模式需要考虑在Node.js上的兼容性；
2. 数据接口如何fetch，在浏览器上是AJAX，在Node.js上是什么；
3. 如何做路由同构，浏览器无刷新切换页面，新路由在服务端可用；
4. 大量DOM渲染如何避免阻塞Node.js的执行进程。

目前 GitHub 上 star 较多的同构框架包括 Vue.js 的 nuxt 和 React 的 next.js，以及数据存储全包的 meteor。可以肯定的是，不论它们是否能部署在生产环境中，都不可能满足你的所有需求，适当的重新架构是必要的，在这个新的领域，没有太多的经验可以借鉴。

六、未来技术与职业培养

6.1 大数据方向

越来越多做 toB 业务的公司对前端的需求都是在数据可视化上，或者更通俗一些——报表。这个部分在从前通常都是前端工程师嗤之以鼻的方向，认为无聊、没技术。不过在移动端时代，特别是大数据时代，对此类技能的需求增多，技术的含金量也持续提升。根据“面向工资编程”的原则，一定会有大量工程师加入进来。

对这个方向的技术技能要求是 Canvas、WebGL，但其实绝大多数需求都不需要你直接与底层 API 打交道，已经有大量第三方工具帮你做

到了，不乏非常优秀的框架。如百度的 ECharts，国外的 Chart.js、Highcharts、D3.js 等等，特别是 D3.js，几乎是大数据前端方向的神器，非常值得学习。

话说回来，作为工程师，心存忧患意识，一定不能以学会这几款工具就满足，在实际的业务场景中，更多的需要你扩展框架，生产自己的组件，这需要你具备一定的数学、图形和 OpenGL 底层知识，可以说是非常大的技术壁垒和入门门槛。

6.2 WebVR

今年可以说是 VR 技术爆发式的一年，市场上推出了多款 VR 游戏设备，而淘宝更是开发出了平民的 buy+ 购物体验，等普及开来，几乎可以颠覆传统的网上购物方式。

VR 的开发离不开对 3D 环境的构建，WebVR 标准还在草案阶段，A-Frame 可以用来体验，另一个 three.js 框架是一个比较成熟的构建 3D 场景的工具，除了能在未来的 VR 应用中大显身手，同样也在构建极大丰富的 3D 交互移动端页面中显得必不可少，淘宝就是国内这方面的先驱。

6.3 WebAssembly

asm.js 已发展成 WebAssembly，由谷歌、微软、苹果和 Mozilla 四家共同推动，似乎是非常喜人乐见的事情，毕竟主要浏览器内核厂商都在这里了。不过合作的一大问题就是低效，今年终于有了可以演示的 demo，允许编写 C++ 代码来运行在浏览器上了，你需要下载一大堆依赖库，以及一次非常耗时的编译，不过好歹是个进步。

短时间内，我们都不太可能改变使用 JavaScript 编写前端代码的现状，Dart 失败了，只能期望于未来的 WebAssembly。有了它，前端在运行

时效率、安全性都会上一个台阶，其他随之而来的问题，就只能等到那一天再说了。

6.4 WebComponents

WebComponents 能带给我们什么呢？HTML Template、Shadow DOM、Custom Element 和 HTML Import？是的，非常完美的组件化系统。Angular、React 的组件化系统中，都是以 Custom Element 的方式组合 HTML，但这都是假象，它们最终都会被编译成 JavaScript 才会执行。但 WebComponents 不一样，Custom Element 原生就可以被浏览器解析，DOM 元素本身的方法都可以自定义，而且元素内部的子元素、样式，由于 Shadow DOM 的存在，不会污染全局空间，真正成为了一个沙箱，组件化就应该是这个样子，外部只关心接口，不关心也不能操纵内部的实现。

当前的组件化，无不依赖于某一特定的框架环境，或者是 Angular，或者是 React，想移植就需要翻盘推倒重来，也就是说他们是不兼容的。有了 WebComponents，作为浏览器厂商共同遵循和支持的标准，这一现状将极有可能被改写。

未来的前端组件化分发将不会是 npm 那么简单，可能只是引用一个 HTML 文件，更有可能的是包含 CSS、HTML、JavaScript 和其他二进制资源的一个目录。

目前只有最新的 Chrome 完全支持 WebComponents 的所有特性，所以距离真正应用它还尚需时日。由于技术上的限制，WebComponents polyfill 的能力都非常受限，Shadow DOM 不可能实现，而其他三者则更多需要离线编译实现，可以参考 Vue.js 2 的实现，非常类似于 WebComponents。

6.5 关于微信小程序

微信小程序对于今年不得不说，却也无话可说。依托于庞大的用户量，微信官方出品了自有的一套开发技术栈，只能说给繁杂的前端开发又填了一个角色——微信前端工程师。

七、总结

最后还有几点需要说明。

7.1 工程化

首先，现在业界都在大谈前端工程化，人人学构建，个个会打包。鄙人认为，工程化的要点在于“平衡诸方案利弊，取各指标的加权收益最大化”。仅仅加入了项目构建是远远不够的，在实践中，我们经常需要考虑的方向大可以分为两种：一是研发效率，这直接应该响应业务需求的能力；二是运行时性能，这直接影响用户的使用体验，同时也是产品经理所关心的。这两点都直接影响了公司的收入和业绩。

具体到细节的问题上来，比如说：

- 静态资源如果组织和打包，对于具备众多页面的产品，考虑到不断的迭代更新，如何打包能让用户的代码下载量最少（性能）？不要说使用Webpack打成一个包，也不要说编译common chunk就万事大吉了，难道还需要不断地调整编译脚本（效率）？改错了怎么办？有测试方案么？
- 利用Angular特别是React构建纯前端渲染页面，首屏性能如何保证（性能）？引入服务端同构渲染，好的，那么服务端由谁来编写？想来必是由前端工程师来编写，那么服务端的数据层架构是怎么样的？运维角度看，前端如何保证服务的稳定（效率）？

- 组件化方案如何制定（效率）？如果保证组件的分发和引用的便捷性？如何保证组件在用户端的即插即用（性能）？

对于工程师来说，首先需要量化每个指标的权重，然后对于备选方案，逐个计算加权值，取最大值者，这才是科学的技术选型方法论。

然而在业界，很少能看到针对工程化的更深入分享和讨论，大多停留在“哪个框架好”，“使用 XXX 实现 XXX”的阶段，往往是某一特定方向的优与劣，很少有科学的全局观。甚至只看到了某一方案的优势，对其弊端和可持续性避而不谈。造成这种现状的原因是多方面的，一是技术上，工程师能力的原因并没有考虑得到，二是政治上，工程师需要快速实现某一目标，以取得可见的 KPI 收益，完成团队的绩效目标，但更多的可能是，国内绝大多数产品的复杂性都还不够高，根本无需考虑长久的可持续发展和大规模的团队合作对于技术方案的影响。

因此，你必须接受的现状是，无论你是否使用 CSS 预处理器、使用 Webpack 还是 grunt、使用 React 还是 Angular，使用 RN 还是 Hybrid，对于产品极有可能都不是那么地敏感和重要，往往更取决于决策者的个人喜好。

7.2 角色定位

确实，近两年，Web 前端工程师开始不够老实，要么用 Node.js 插手服务端开发，要么用 RN 插手客户端开发。如何看待这些行为呢？

鄙人以为，涉足服务端开发是没问题的，因为只涉及到渲染层面，还是属于“前端”的范畴的。况且，在实际的工程实践中，已经可以证明，优秀的前端研发体系确实离不开服务端的参与，想想 Facebook 的 BigPipe。不过，这需要服务端良好的分层架构，数据与渲染完全解耦分离，

后端工程师只负责业务数据的 CRUD，并提供接口，前端工程师从接口中获取数据，并推送到浏览器上。数据解耦是比接口解耦更加优越的方案。因此现在只要你的服务端架构允许，Node.js 作为 Web 服务已经比较成熟，前端负责服务端渲染是完全没有问题的。

前端涉足客户端开发也是合理的，毕竟都运行在用户端，也属于前端的范畴。抛开阿里系的 Weex 鄙人不甚了解，NativeScript、RN 都还缺乏大规模持续使用的先例，这是与涉足服务端领域的不同，客户端上的方案都还不够成熟，工具的限制阻碍了前端向客户端的转型，仍然需要时间的考验。不过时间可能不会很多，未来的 Web 技术依托高性能硬件以及普及的 WebGL、WebRTC、Payment API 等能力，在性能和功能上都会挑战 Native 的地位。最差的情况，还可以基于 Hybrid，利用 Native 适当扩展能力，这就是合作而非竞争关系了。

总之前端工程师的本仍然在浏览器上，就这一点，范围就足够广使得没人有敢言自己真正“精通”前端。如果条件允许的话，特别是技术成熟之后，涉猎其他领域也是鼓励的。

7.3 写在最后

在各种研发角色中，前端注定是一个比较心累的一个。每一年的年末，我们都能看到几乎完全不一样的世界，这背后是无数前端人烧脑思考、激情迸发的结果。无论最终产品的流行与否，都推动着前端技术领域的高速更新换代。正是印证了那一句“唯有变化为不变”。作为业务线的研发工程师，我们的职责是甄选最佳组合方案，取得公司利益最大化。这个“最佳”的涉猎面非常广，取决于设计者的技术视野广度，也有关于决策者的管理经验，从来都不是一件简单的事。

未来的 Web 前端开发体验一定是更丰富的，依托 WebComponents 的标准化组件体系，基于 WebAssembly 的高性能运行时代码，以及背靠 HTTP/2 协议的高速资源加载，前端工程师不必在性能上、兼容性上分散太多精力，从而可以专注于开发具备丰富式交互体验的下一代 Web APP，可能是 VR，也可能是游戏。

在迎接 2017 的同时，我们仍然要做好心理准备，新的概念、新的框架和工具以及新的语法依旧会源源不断的生产出来，不完美的现状也依旧会持续。

由于水平有限，笔者在上述内容中难免有失偏颇，请多包涵。

解读 2016，眺望 2017：运维的风口在哪？

兴刚、陈尔冬、赵成、熊胜

编者按

相比开发领域，运维界的发展更显沉稳谨慎。

在 2016 年，有一些值得运维人关注的动态：

Apache 虽然依然是全球范围内使用数量第一的 Web 服务器；但是 Nginx 的热度持续升高，据调查报告显示，繁忙站点更倾向于选择 Nginx，这一现象在中国尤其显著值得关注。

Google 公布了 Web 开发新协议 QUIC（Quick UDP Internet Connections），该协议使用 UDP 作为底层传输协议，旨在通过各种方式减少网络延迟。此外，Google 已将 Chrome 特性在非安全站点上禁用，同时新的特性将只支持 HTTPS。Chrome 和 Firefox 所给出的数据都表明，现

在全世界范围内超过一半的网页采用了 HTTPS。

自动化配置管理工具常见的是 Ansible、Chef、Puppet、SaltStack。其中 Puppet 已问世十一年，相对而言更为成熟也有广泛的平台支持；同时，Puppet 连续多年做出品全球范围的 DevOps 年度报告。而 Ansible 虽然是最年轻的项目，但是它具有清晰的可读语义并且受关注度持续上涨。截止 2016 年 12 月末，Ansible 在 GitHub 上已经拥有 20397 颗 star，远远超出 Chef（4583 颗）、SaltStack（7195 颗）、Puppet（4281 颗）。Ansible 作为后起之秀，执行 SSH 无须 agent，对云设施支持度好，已经引起了广泛关注。

以往运维人常用的语言有 shell、perl、python、ruby；但是运维开发增大了多样性，在一些公司运维开发工作需要使用如 go、Node.js 等。随着 DevOps、SRE 的概念推广和一线互联网公司的实践落地，业界已经形成一种认识——运维工作人员需要学习掌握开发能力。

不过，各家公司的情况迥异，从而运维工作内容、分工和工具选择会有不同甚至很大差异。上述的一些变化不一定会触达到每家公司，而且即使受到影响，因已选软件和语言的所带来的变化也不至于翻天覆地。

这篇 2016 年盘点文章，着眼于运维领域宏观的变化趋势，以期能给运维同仁带来帮助。首先，文章呈现了两个派系运维（自有云和厂商云）的具体实践和心得；随后探讨 SRE 概念并列举了互联网领军公司的 SRE 落地；最后，分析正在兴起的智能化运维并对其影响作以简单展望。

新内容：容器化的运维

如果说 2015 年是容器化的元年，那么 2016 年就是容器化开花结果并丰收的一年。一些互联网大厂已经大部分甚至全线容器化，而维护大规模

容器集群带来了哪些变化又有哪些挑战？

没有容器之前的运维是？

每到业务高峰段（对电商而言，即促销活动）或者重大变更的时期，底层资源扩充都需要机房大规模集中性的机器上电、安装系统、部署网线，并且为各个业务划分物理机器。但是业务并不能立刻启用，为什么？打个比方，新到手的机器如同毛坯房，不经装修无法安家入住。首先需要进行水电类的硬装修，即业务版本底层的依赖包；其次才可进行软装修，即业务版本等。与业务版本相关的一切都是由业务运维部门统一处理，即便可以采用 Ansible 或者自研的自动部署工具提高效率，但是无法避免排队等待上线的时间花费。并且，一次次的部署也会遇到各种问题，如预期与实际环境不完全相同；即便部署成功后依然要处理后期维护，如依赖库跨版本升级。经过这些关卡，满足了某次的业务高峰，那下一次需要新的调整呢？如果要对机器重新分配和再利用，那就需要重新格式化安装。

有了容器之后呢？

一切以镜像为中心，并且资源管理方式完全不一样。

首先，保证环境的一致性。为了配合业务而进行的重大调整，不再需要传统的集中化操作（这也是最容易出问题的环节），需要分配的不再是物理机器，而是核。首先预先估算出整体需要扩充的总核数，交由机房提前统一采购部署。（以京东为例，会根据大数据分析出的业务增长率；不需要逐个统计业务需求。）然后，各个业务通过快照或者 build 方式构建镜像，最大程度地固化运行环境并主动提交；无需运维人员逐台机器检查各种软件版本。随后，业务人员通过自己专属镜像在控制平台的上线业务，并同意完成数据库授权等操作。对于不再使用的容器，业务人员可以在管理平台点击销毁即可释放对应的资源；而释放的资源立刻回到资源池中，

无需人工操作。

其次，生产环境中秒级快速扩容，这一项已经在电商促销情形等经历了很多次业务峰值的考验，并得到了实践者的广泛肯定。当某个业务流量突增并需要资源时，运维人员不再需要找领导申请匹配的机器，也不必辛苦业务运维人员部署环境、再联系网络等等的授权。容器化可以提供横向和纵向的能力：

- 横向——运维人员只需在控制台页面直接申请新的资源加入业务集群，自动化完成审批授权、镜像及统一的配置中心保证环境的一致性。
- 纵向——针对临时压力较大的业务，容器直接在同一台机器上临时借用比自己优先级低且较空闲容器的资源，这种能力可以很好的应对秒杀类电商业务。

容器化运维的挑战是？

对于维护来说，除了后期的扩容、升级等，还有一件更重要的任务——及时发现问题并进行处理，这就涉及到了监控。容器化之后的监控工作有业务、容器及物理机三类级别，并支持业务自助式告警规则。监控层面不仅需要提供某个机器的负载情况，还需要支持某个业务的整体负载，需要用户对业务压力作以评估，并在必要情况下进行扩缩容。同时，若是监控到异常机器可以通过统一日志直接提供给业务分析问题。

容器化遇到的另一个维护难点，就是多业务混布。由于容器是共用内核，为了防止资源碎片化，集群都是业务混布。那怎么做到某个业务发生问题不影响其他业务，以及怎么对已发生问题进行及时处理就能成运维必须考虑的问题。针对这种情况，必须要提供统一的资源限制接口，能针对 swap 磁盘等进行针对性的限制。一定要充分利用容器化的长处，即多副

本及弹性伸缩实现的便利性。对于异常的容器，运维可以直接缩容掉并不影响整体业务，可尽快的防止单个容器可能对物理机造成的影响。容器虽然易于销毁，但要有统一日志或者本地日志备份保证问题的分析。

容器化其实并没有带来更新鲜的运维技术，它一方面是减少了运维人员大量的部署工作，另一方面是改变了一直以来的运维习惯。如果业界都认识到容器是无状态的：忙时动态扩容，闲时自动缩容，容器的异常是常态（经常是物理机导致），那么运维的春天就不远了。

新模式：厂商云上的运维

2016 年见证了中国 IDC 行业进入发展阶段，这一年，业务更加趋向多样化，对应的 IT 技术要求随之增多，并且难度加深。比如，高清视频直播 / 点播，大数据，人工智能、物联网等。中国的 IDC 市场规模增长率为全球平均水平的两倍。据预测，未来三年中国 IDC 市场增长率将保持在 35% 以上。Cisco 公布的报告显示 ，今年全球有 65% 的数据中心基于云服务，这一数字在未来三年内将升至 83%，并且公有云和私有云将各自分担 56% 和 44% 的流量。这意味着企业的运维进入关联厂商云模式将成为必然的发展趋势。

那将数据中心交给云厂商之后，如果做运维呢？云计算到底是“运维”行业的终结者还是神助攻？

一个在运维领域摸爬滚打十余年，并有对于私有云、公有云甚至混合云环境运维都参与过的运维人，在这里想和大家分享一些思考。

中美运维发展分析比较

据笔者观察，欧美互联网企业对于运维的未来定义也具有微小的分歧。

以 AWS 为代表的“谁开发，谁运维”模式，可能是较为激进的一种。

AWS 的服务副总裁 James Hamilton 认为，运维领域中 80% 的问题是设计与开发问题。很显然，采用研发工程师直接运维的方式会直接解决这些问题。然而，我认为中美从业者的职业习惯还是有较大不同的。一方面，主流操作系统、编译器与开发语言大都发源自美国，美国在系统领域享有巨大的技术红利，拥有更多具有系统与运维技能背景的研发工程师；另一方面，即便是 AWS 的研发工程师，对于值守也感觉有较大压力。

虽然 AWS 采用的研发直接运维的形式教难以批量复制，但是 Google SRE 模式则可能被更多的国内公司逐渐采用。我想也恰恰是因为这个原因，最近关于 Google SRE 的讨论在互联网运维圈的讨论中是当之无愧的 Top 1 话题。同时我们也要看到，Google 的 SRE 也是拥有系统与运维技能背景和意愿的开发工程师。（更多内容详见后文）

综合 AWS 和 Google 两种流派可以看到，运维领域是不会消亡的。但是，它的参与主体人群可能会迁移，它对于参与者的能力要求可能会改变。而且可以明确的看到，随着行业的成熟，对于运维从业者开发能力的要求会逐渐提高。

云计算用户的运维如何做

由于每家企业对于云计算的理解与看法并不完全相同，所以在云的使用上也存在一定差异。针对不同的云计算应用策略，运维工作肯定也会存在变化。下面将会对于主流云计算应用形式来分析运维工作的变化：

1、单一公有云用户

公有云代表性产品 AWS 为例，AWS 的服务可以分为三类：无托管、半托管和全托管。全托管类服务我们放在后文中 Serverless 架构中探讨。本节只探讨无托管与半托管两类服务。

针对无托管服务，最典型的莫过于 AWS EC2。EC2 就相当于一台物理服务器，用户需要在 EC2 中部署自己的应用与中间件。针对这种场景，显然云厂商的用户还有大量运维领域的工作需要做。当然，运维工程师完全可以按照传统运维的模式在公有云上完成运维工作。但是，运维工程师想要提高效率并且对稳定性带来实际性的提升，还是有很多额外的工作空间。比如：运维工程师可以自行研发自动化运维系统，通过 API 和 SSH 与 AWS EC2 相集成，开展自动扩容、自动部署、配置变更、容量管理、计费管理以及监控等运维领域工作；又或者借助于 AWS 提供的 CloudFormation、OpsWorks、CodeDeploy 等管理工具更快捷的实现。基于 AWS 的管理工具可以帮助运维工程师便利且快速完成目标，但是自研的系统灵活性更强。

针对半托管服务，比如 AWS RDS。用户并不需要去考虑操作系统层的问题。但是用户仍然需要去考虑高可用、备份、监控以及故障切换等问题。显而易见，使用半托管服务时，用户仍然需要利用公有云 API 来构建高可用架构、设置定期备份、实现监控与故障自动切换并实现自动扩容等功能。

2、异构公有云用户

目前国内可选用的云计算厂家众多，每家的性能、 稳定性与成本都各不相同。不少公有云的用户为了防止被某一家云计算服务商的问题严重影响自己的业务，会考虑同时使用多家云计算厂商。由于每家公有云厂商之间的服务实现和 API 可能会有差别，在这种情况下，用户通常会尽可能只选用类似 AWS EC2 这样的无托管服务。这样，用户不光更容易实现不同公有云之间的迁移，甚至可以轻松迁移到自由基础设施上。

在这种场景下，运维工程师不光需要不依赖公有云管理服务来完成单一公有云用户无托管类服务的工作，还需要实现异构云的管理。

3、混合云用户

异构云的进化版，公有云与自有基础设施的混合使用。除了将异构云管理扩展到自由基础设施外，混合云用户还可以有两个方向的进化：

a）实现应用能够跨越自由基础设施和公有云的弹性部署与扩容。此方向的用户通常会采用 Docker 等容器方案。

b）在自有数据中心实现一套与公有云 API 相同的基础设施；或者在公有云与自由基础设施之上实现一层混合云 PaaS 或 API，使之可以自适应公有云和自有基础设施。

无论哪种方向，运维工程师都有非常多的工作需要做。

4、Serverless 架构用户

Serverless 架构是这几年公有云领域蹿升的热门技术。虽然名词听起来很新，但其概念并不是突然提出来的。我个人认为，Serverless 架构其实是一种更为抽象化的 PaaS 服务。既然并不算全新的领域，所以 Serverless 架构的基础设施落地自然也就不是什么难事。随着 AWS 将要在国内引入 Lambda 服务和国内公有云厂商纷纷推出 Lambda 服务，Serverless 架构在基础设施与服务上已经实现了落地。

然而基础设施落地不代表客户一定会买账。由于 Serverless 架构比类 AppEngine 类 PaaS 服务更为抽象，国内开发者对其接受程度还需要时间来验证。可以肯定的是，最为活跃又乐于尝试新技术的开发者肯定已经开始尝试 Serverless 架构了，但是企业为了业务长远发展是否会在近一两年就考虑 Serverless 架构目前恐怕还不能下结论。总之，我个人观点，虽然基础设施已经落地，但 Serverless 架构真正被企业大规模使用可能还需要时间。

由于 Serverless 架构的高抽象化，完全将云计算与应用代码直接连接，也许确实不太需要运维这个工种甚至领域。但是，Serverless 架构对于开发者也是存在进入成本，现在的运维工程师完全可以通过学习在未来转向为 Serverless 架构开发工程师。

云时代下，运维岗位会怎样？

- 通过上文的分析与总结，我们可以推断未来运维领域的三种可能：
- 像AWS那样趋向岗位统一；
- 像Google那样岗位独立；
- Serverless架构环境下用户无需考虑运维领域，岗位仅剩开发工程师。

其中前两种对运维岗位能力要求趋同。

无论沿着哪个发展方向，开发与工程能力都是当前运维工程师需要注重发展的首要能力。或许未来不需要运维工程师，但你还可以做开发工程师啊！

新主张：从 SRE 谈开来

SRE 的概念前几年业界就有提及，但是当时的资料信息并不是很多，只道是“网站稳定工程师，以保障稳定为主”。今年，Google SRE 团队将经验沉淀落笔成书；同时，笔者近两年有幸与在海外从事 SRE 的专家进行交流，再结合自己多年的互联网运维经历。整理思路之后，将个人理解和观察分享如下。

SRE 由 Google 提出

首先抛个结论出来，SRE 的目标不是 Operation，而是 Engineering，是一个是“通过软件工程的方式开发自动化系统来替代重复和手工操作”

的岗位，为了保证达成这个目标，Google 强制约定了 50% 的工作法则，SRE 至少保证 50% 的时间是在做自动化开发的工作上，实际这个比例可能会更高，所以 SRE 运维的工作内容是低于 50% 的。

我个人觉得更准确的理解应该是，Google 压根就没把 SRE 定义为运维（Operation）的岗位，运维（Operation）这个岗位或工作内容更多的指的是原来传统运维模式下 SA 的职责描述。

Google 团队在书中第一章就分析了从 SA 和 SRE 两个不同的视角来看待 Google 线上系统的区别，正是因为 SA 模式下遇到了很多无法解决的问题，才引入了 SRE 这样的软件工程岗位，而引入这个岗位的目标就是为了消除掉原来 SA 运维模式下的问题、矛盾和冲突。

也正是 Google 换了一个思路，从另外一个维度来解决运维的问题，才把运维做到了另一个境界。下面是文中的几个关于 SRE 的描述，大家可以一起理解下看看。

- By design, it is crucial that SRE teams are focused on engineering;
- SRE模型成功的关键在于对工程的关注；
- SRE is what happens when you ask a software engineer to design an operations team;
- SRE就是让软件工程师来设计一个新型运维团队的结果。

另外，还有一个很有意思的地方，就是整本书中提到 Operation（运维）的地方其实并不多，而且大多以 Operation load、Operation overload、Traditional/Manual/Toil/Repetitive operation works 等词汇出现，理解一下，是不是跟上面的推断也很契合。

SRE 虽然没有被给以直接的定义，但是 Google SRE 给出了职责描述：

- In general, an SRE team is responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of their service(s).
- SRE需要负责可用性、时延、性能、效率、变更管理、监控、应急响应和容量管理等相关的工作。（这里先不做过多的解读，后面详细描述。）

Google SRE 人力技能模型大致分为两类：50-60% 为 SWE，即软件工程师；余下 40-50% 还要至少对 Unix 内核和底层网络（1-3 层）非常精通才可以。从这里也可以大致推断出，Google SRE 的技能要求是非常高的，SWE 只是基础条件。从技能模型上，按照 Google 的标准，原来传统的 SA 或 NE 这样运维角色根本无法胜任 Google SRE 的岗位，势必要进行非常艰难的转型。此外，除了软件开发、内核和网络等硬性的知识技能，还需要具备产品 sense、沟通协作能力、规范制定意识等软技能，因此 SRE 门槛确实很高。

那么 DevOps 和 SRE 的关系是什么呢？在《SRE:Google 运维解密》一书中，谷歌团队认为DevOps所倡导的与SRE的核心思想和实践经验相符合，并且认为 SRE 是 DevOps 模型在 Google 的具体实践。

大厂都怎样落地高端运维

雅虎：作为互联网业界的鼻祖与技术的翘楚，硅谷很多优秀的技术经验都源自雅虎。在雅虎，有个运维的岗位叫 PE（Product Engineer），早期能够走上这个岗位的工程师都是在开发团队承担业务架构师或资深 SWE 这样的角色，因为一个应用或业务上线后，对应用最熟悉的就是这批

人，他们能够很好的跟产品人员协作起来，传递应用在线上运行的状况。同样，产品人员一旦发现问题与 PE 可以顺畅地交流，交流完还可以直接改代码上线。在这种运作模式下，产品、开发、运维协作非常最高效，这种模式被一直延续下来。因此，PE 的岗位职能和角色与 SRE 是基本相同的。

阿里：2005 年，阿里收购了雅虎中国以后，雅虎中国的工程师也被合并进来。阿里应用运维岗位也叫 PE，这个岗位就是传承着雅虎的运维文化和模式而来，据说是现在阿里合伙人之一刘振飞 09 年当时在创建技术保障部时成立了 PE 的团队。但是，这支 PE 团队更多的就是偏应用运维了，绝大部分人是不具备 SWE 能力的，这一点也是受限于当时国内整个技术能力的水平，不可能一下招到这么多的原来雅虎的那种 PE 工程师。（不过这并不是大问题，将在后文中分析为什么。）

Facebook：熟悉 FB 的同学可能也不陌生，FB 的应用运维岗位也叫 PE。师承何处笔者并没有找到第一手资料，前段时间通过与 FB 的一个工程师交流，发现 FB 的 PE 与阿里模式相近更偏应用运维一些。

LinkedIn：很有幸今年 12 月 2 日的 ArchSummit 大会上笔者负责的运维专题邀请到了 LinkedIn SRE 团队主管，并在会议期间深入讨论了的关于 SRE 团队的组建和分工职责等。在 LinkedIn 以应用运维为主，SRE 的职责跟阿里 PE 和 FB PE 相似。

Google 的 SRE 必须具备很强的 SWE 能力，所以可以自行开发很多的自动化和稳定性的模块。但是这种人才很稀缺，对于一般的公司很难招到这样的人或者组成这样的一支团队，所以按照 Google 的模式基本是玩不转的，那应该怎么办呢？答案就是：依靠团队的力量：单个人搞不定的事情，可以靠团队中具备不同能力的人协作，共同达成 SRE 的职责和目标。根据笔者对 FB、LinkedIn 和国内的绝大多数公司的了解，这种方式实际

也是大多数公司所采用的方式。目前对于运维团队的基本组成模式：

- 系统运维：SA、网络工程师和IDC工程师。
- 应用运维：国内大多叫应用运维，国外大多都定义为SRE或PE（国内同样，如阿里叫PE，滴滴、小米、美团等叫SRE）。
- 技术支持：以阿里为例，主要是问题跟踪和一些流程组织及闭关跟踪的事情：故障复盘、改进Action执行跟踪等。其它很多公司可能由QA会承担一部分这样的职责。这个部门相应地国外叫NOC，虽然不参与问题的直接解决，但是对于问题的推进，尤其是对于线上运维规范性的监督作用非常大。
- 工具&平台开发：自动化、监控、持续集成&发布和稳定性平台开发。
- 数据库DBA：DBA有可能也会是独立团队。
- 运维安全：对线上网络、系统和应用安全负责。大多是独立团队，但是即使独立，跟运维团队都是紧密协作的。

还是以阿里为例，阿里之前的技术保障部简称就叫 SRE，是 PE 应用运维、工具开发、技术支持、DBA、安全、系统运维的组合起来的一个大的部门，非常典型的 SRE 团队作战的优秀实践。但是从今年开始，运作模式也发生了很大的变化，特别是应用运维 PE 这个岗位，后面会详细讲到。同时，后面我们再提到 SRE 就不是一个单独的岗位了，而是一个团队或者一种能力，那接下来重点说一下应用运维和工具平台开发的岗位。

两类运维岗位的思考

(1) 应用运维需求上升

随着互联网业务的高速发展，到目前为止已经诞生出太多的大大小小

的互联网公司，各个公司都越来越需要 SRE 或 PE（应用运维）这样的角色。例如，在 Facebook、PE 对开发的比例目标是 1:30；这个比例很有挑战（包括目前 Facebook 自己），大多公司可能还在 1：100，甚至更低。但是从趋势上，可以说明应用运维这个岗位的重要性越来越大，同时也越来越受到重视，对于做运维的小伙伴无疑是个很好的信号。

目前在国内，我们的应用运维岗位还是多以线上的部署、发布、监控和问题的处理为主，其中有很多都还是以手工操作的方式为主，按照之前我们的分析，SRE 的目标不是做这些事情的，或者说不应该是以这些事情为主才对，所以大家可以想一下我们的应用运维在实际日常工作中，是不是以这些事情为主？甚至把这些事情当做了常态？如果是这样，按照 SRE 的标准就不是合格的 SRE。

对于应用运维笔者认为首先要做到意识上的转变，（尽可能将工作提炼并转化成脚本需求）（即产品需求分析能力），同时还要进行标准和规范的制定(SLI、SLA、SLO)与执行(将标准规范和需求功能固化到软件平台)。这一系列的专业工作还需要软性技能：怎样将需求转化成产品层面需求并表达出来，怎样同业务开发同学沟通制定标准和规范等等。

（2）工具平台（运维开发）为实现依赖

这个角色，实际就是 SRE 中 SWE 的能力职责了，要能够准确的理解应用运维同学的需求，是否能够开发出满足实际运维场景的平台，直接依赖于工具平台同学的能力。

这里涉及到怎样理解产品设计进行精确开发，如何将孤立产品整合便于使用。同时工具平台铜须还需要锻炼学习系统、网络和应用运维的一些技能，因为通过这块能力的提升，工具开发同学实际是在转向 Google 标准中的 SRE。

小结

Google 定义的 SRE 的角色如果单兵作战能力达不到，就通过团队协作来达成。这也是基本除了 Google 之外的互联网公司所采取的一种运维模式。

在具体落地时，可以将 SRE 的落地拆分成不同的职责岗位和团队协作。应用运维应该要能够制定和执行各种稳定性的标准和规范，能够将人工和重复的工作提炼成需求，并把这些需求能够转化成产品设计文档，准确的传递到工具平台团队，确保各方理解一致，从而能够使得各种自动化的工具平台落地。各大公司对这个应用运维的角色越来越重视，在笔者看来，应用运维的好坏直接决定了系统的稳定。

SRE 所涵盖的工作内容和职责，其实在国内外的互联网公司也都在做，比如自动化、持续集成和发布、监控等等，对于标准、规范和流程上，每个公司也都有自己的一套适合自己公司业务和技术特点的体系。SRE 不神秘，Google 在做的事情，业内公司其实也在做，从体系上一些大厂也已经非常的完善，但是技术能力上的差距，是我们努力的方向。

新发展：当运维遇上智能

2016 年可以说是人工智能的元年，各行各业都在讨论智能化，都在尝试将智能在各自领域落地。运维是个传统意义中“劳动密集型”行业，公司们希望降低人工运维风险，同时又希望提升监控质量和日志分析能力在部分运维事务中形成闭环，这也就是为什么笔者认为运维对智能化的诉求其实更有现实意义。

其实，人工智能技术已经开始在运维领域小试身手，如大数据 SRE 团队会使用各种机器学习算法加持大数据计算引擎的助力，以期获得自动化

解决方案。在笔者看来，运维生态有了另外一种可能，转向智能化运维时代。

我所理解的智能化运维

智能化运维是一套智能体系，它强调了从监控到分析决策再到执行的整个过程的无人化甚至超人化，突出的是系统的自治能力和预知能力。

智能化运维主要需要做到监控智能化、分析智能化和执行智能化。

监控智能化相较于传统监控，突出的是监控数据的深度和多样性。例如从读数到读图的增强，从当下判断到全周期比较，从静态阀值到动态加速率，从系统基础指标到定义标准运维 metric，一步步将监控从被动接受到主动预知方向推进。

执行智能化是对解决各式各样的运维模式下人工执行的沉淀。今天仍然有很大一部分的运维事务需要由人去处理，如何构建一个灵活的运维事务沉淀框架是一件非常有令人激动的工作。如同蒸汽时代对人类发展的意义，智能执行是所有智能化的基础和前提。

分析智能化是最终替代和超越人的思维的解决方案，也必然是我们智能运维的发展核心。智能化分析的实现离不开各种人工智能技术的应用，当谷歌的 AlphaGo 大胜人类的那刻，人工智能不再属于科幻，而是切切实实的未来。当然广义上的人工智能不是我们今天所立刻能够达到的，在比较长的一段时间里，我们能够企及的是在特定任务中机器的能力可以匹敌人的能力，甚至更好。这种能力是在特定业务场景下的狭义人工智能，也是我们未来几年努力的主要方向。

新的能力需求

谈到智能化运维的具体落地，笔者预测未来在资源规划、故障处理以

及监控预警这几个业务方向将会有实质性的突破，这也意味着对机器学习和数据挖掘人才的需求将是以后的大势所趋，IT 企业在运维自动化和智能化将会有倾向的投入。

在笔者看来，新时代的优秀运维人，应该学习基础系统、业务产品，具备优秀编码能力，并将容器化、大数据、机器学习算法结合到业务场景中。

小结

“运维工作”在今天需要被重新定义，传统意义的运维重视被富裕繁重但是低价值的标签。笔者认为运维分为四个层次：“人工、自动、智能、智慧”，当前的运维行业可以说是处在一个人工、自动、智能互相交叠的时期。

智能化运维的价值在于：降低“传统运维”事务的时间；增大高附加值运维事务的投入；使得运维工作可以向上关联业务发展，向下关联基础建设，对整个公司长期发展将起到举足轻重的作用。2017 年会是运维智能化百花齐放的中兴之年：通过技术、数据提升人力资源、财务资源、基础建设等方面的效率，笔者认为运维将会在整个公司生态中拥有更大的话语权，期待智能化时期业界发掘出真正的运维价值。

作者简介

兴刚，京东商城基础平台集群技术部高级研发工程师。主要参与公司业务容器化相关工作，有效支撑公司各级业务，持续关注容器化进程。

陈尔冬，链家网运维总监。资深互联网专家，曾服务于新浪网，历任研发中心高级技术经理、技术保障部总监，华为企业云资深技术顾问。在高可靠业务、运维自动化、DevOps 等方面有十年积累经验。

赵成，花名谦益，蘑菇街运维经理，ArchSummit 明星讲师。负责蘑菇街运维团队的管理以及运维体系的建设工作。在运维行业中已经做了 8 年，之前有过 5 年左右的业务开发经历。加入蘑菇街之前在华为一直做电信级业务的开发和运维工作。

熊胜，花名池枫，阿里巴巴集团技术专家。2011 年加入阿里巴巴基础架构事业部大数据 SRE 部门，见证阿里大数据产品最快速的发展过程，先后负责阿里 Hadoop、HBase、Apsara、ODPS 等产品运维，全程负责大数据运维自动化体系建设。目前负责大数据 SRE 自动化开发团队，历时 2 年时间带领团队完成 Tesla 自动化体系设计、开发、落地、进化历程。专注智能运维在大规模异构集群下的场景应用，专注业务运维与智能运维结合后的转型道路探求。

版权声明

InfoQ 中文站出品

解读 2016

出版：北京极客邦科技有限公司
北京市朝阳区洛娃大厦 C 座 1607
欢迎共同参与 InfoQ 中文站的内容建设工作，包括原创投稿和翻译，请联系 editors@cn.infoq.com。
网 址：www.infoq.com.cn